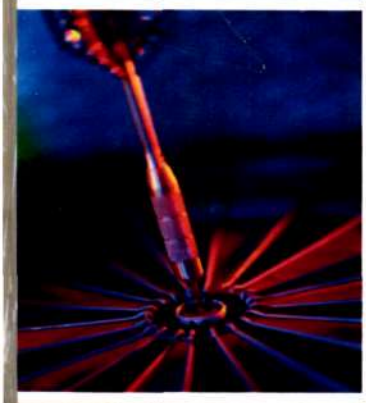


Рябенький В.М.
Ходаков В.Е.
Ушкаренко А.О.

КОМПЬЮТЕРНОЕ УПРАВЛЕНИЕ ВНЕШНИМИ УСТРОЙСТВАМИ ЧЕРЕЗ СТАНДАРТНЫЕ ИНТЕРФЕЙСЫ

ВИДАВНИЦТВО Ольги плюс

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ

НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ КОРАБЛЕСТРОЕНИЯ
ИМЕНИ АДМИРАЛА МАКАРОВА
ХЕРСОНСКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Рябенький В.М.
Ходаков В.Е.
Ушкаренко А.О.

КОМПЬЮТЕРНОЕ УПРАВЛЕНИЕ ВНЕШНИМИ УСТРОЙСТВАМИ ЧЕРЕЗ СТАНДАРТНЫЕ ИНТЕРФЕЙСЫ

Херсон
2008

УДК 004.451.1
ББК 32.973
Р 98

Компьютерное управление внешними устройствами через стандартные интерфейсы. Учебное пособие. / Рябенький В.М., Ходаков В.Е., Ушкаренко А.О. – Херсон: Олди-плюс, 2008. – 380 с.

ISBN 978-966-8447-51-8

Рецензенты:

Смирнов В.С. докт. техн. наук, профессор

Фисун Н.Т., докт. техн. наук, профессор

Одним из наиболее простых и часто используемых способов организации обмена данными между компьютером и внешними устройствами является использование стандартных портов ввода-вывода – таких, как COM, LPT, IrDA и USB, а также промышленных интерфейсов SPI, I^2C, 1-Wire. Данное учебное пособие посвящено объяснению принципов их работы и созданию управляющих пользовательских программ. Рассмотрены вопросы сопряжения устройств с компьютером по указанным интерфейсам. Приводятся примеры схем, а также исходные коды программ для компьютера и микроконтроллеров семейства AVR. Пособие ориентировано на разработчиков электронной аппаратуры, у которых возникает необходимость в обеспечении программной поддержки своих устройств.

Предназначено для студентов специальностей 8.091501 «Компьютерные системы и сети», 8.090803 «Электронные системы» и смежных специальностей, аспирантов, научных сотрудников, а также всех тех, кто интересуется компьютерной электроникой и программированием.

Авторы:

Рябенький Владимир Михайлович, докт. техн. наук, профессор, зав. кафедры теоретической электротехники и электронных систем Национального университета кораблестроения.

Ходаков Виктор Егорович, докт. техн. наук, профессор, зав. Кафедры информационных технологий Херсонского национального технического университета

Ушкаренко Александр Олегович, ст. преподаватель кафедры теоретической электротехники и электронных систем Национального университета кораблестроения.

ISBN 978-966-8447-51-8

Содержание

ВВЕДЕНИЕ

Промышленные и персональные компьютеры (ПК) находят все более широкое использование в задачах управления, сбора и обработки информации, системах автоматики различного назначения. Их широкое использование обусловлено не только низкой стоимостью ПК, но и развитым интерфейсом, то есть широкими возможностями и функциональной гибкостью стандартных портов и протоколов обмена информацией. Полезным дополнением стандартных интерфейсов стали AVR и PIC микроконтроллеры со своими портами и протоколами, а также ряд микросхем, которые позволяют легко согласовать взаимодействие между ПК и практически любыми периферийными устройствами. Следует также учитывать тот факт, что большое количество датчиков, микросхем памяти изготовляются с цифровым выходом и стандартными протоколами низкого уровня для обеспечения непосредственной связи с микро контролерами. Таким образом для большого количества практических задач проблемы аппаратной организации систем сбора информации, как и проблем передачи информации на внешние устройства фактически решены. Поэтому при разработке таких систем на первое место ставятся задачи программной организации взаимодействия ПК с микроконтролерами и другими внешними устройствами на основе использования стандартных портов. Непосредственно этой теме и посвящено учебное пособие, что предлагается читателям.

В пособии детально рассмотрена особенность организации программного взаимодействия ПК через LPT, COM и USB порты, которые фактически полностью вытеснили другие аппаратные средства связи (например, ISA – шину и игровой порт). Каждому из портов по-

священо по два раздела. В первом и втором рассматриваются особенности аппаратной организации, протоколы и программирования LPT – порта. Третий и четвертый разделы дают возможность изучить соответственно особенности взаимодействия ПК с внешними устройствами через COM – порт. Пятый и шестой разделы посвящены, соответственно USB – шине. В каждом из разделов приводится достаточно большое количество примеров с детальным анализом соответствующих программ, что существенно упрощает изучение материала. Программирование ведется под операционные системы семейства Windows NT/2000/XP с использованием Visual Studio NET. В седьмом разделе приводится короткая характеристика доступных программных средств, которые могут использоваться читателями, а в восьмом – особенности организации управления внешними устройствами по компьютерной сети.

В дополнении приводится необходимая для подготовки программного обеспечения информация о стандартных протоколах и шинах микроконтроллеров.

1. ПАРАЛЛЕЛЬНЫЙ ПОРТ

1.1. Аппаратная организация порта

Параллельный порт компьютера (в англоязычной терминологии LPT, аббревиатура произошла от Line PrinTer – построчный принтер) является промышленным стандартом для подключения к компьютеру печатающих устройств, а также может быть использован для обмена данными между компьютером и внешними устройствами, в том числе изготовленными самостоятельно. К LPT-портам подключаются принтеры, плоттеры, сканеры, устройства хранения данных, коммуникационные устройства, электронные ключи, программаторы, прочие приборы и устройства.

С внешней стороны порт имеет восьмибитную шину данных (Data Bus), пятибитную шину сигналов состояния (Status Bus) и четырёхбитную шину управляющих сигналов (Control Bus), выведенные на разъём-розетку DB-25S. В каждую из указанных шин цифровые данные подаются из определённых регистров – соответственно *регистра данных* (Data Register), *регистра состояния* (Status Register) и *регистра управления* (Control Register). На рис. 1.1 показан внешний вид разъёма LPT-порта.

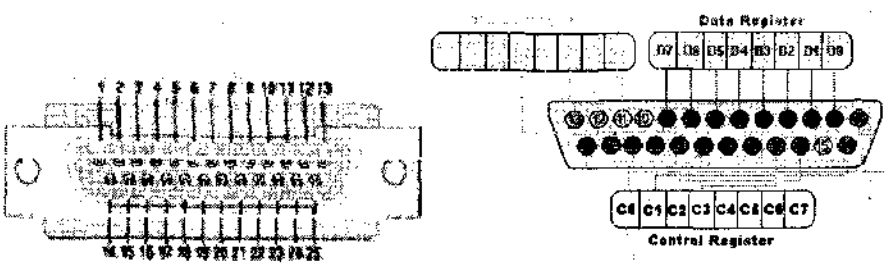


Рис. 1.1. Вид разъёма LPT-порта

Порт параллельного интерфейса был введен в ПК для подключения принтера - отсюда и пошло название LPT – порт (LinePrinTer). Хотя через этот порт подключается и лазерные принтеры, которые печатают не строками, а страницами, название LPT осталась.

Адаптер параллельного интерфейса представляет собой набор регистров, расположенных в пространстве ввода\вывода. Регистры порта адресуются относительно базового адреса порта, стандартными значениями которого являются 3BCh, 378h, 287h. Порт может использовать IRQ7 или IRQ5.

BIOS поддерживает до 4-х LPT портов своим сервисом - прерыванием INT 17h, что обеспечивает через них связь с принтерами или другими устройствами по интерфейсу Centronics. Этим сервисом BIOS осуществляет вывод символа (по опросу готовности, не используя аппаратных прерываний) инициализацию интерфейса и принтера, а также опрос состояния принтера.

Понятие Centronics относится как к набору сигналов и протокола взаимодействия, так и к 25-контактному разъему, который устанавливается на принтерах. Назначение сигналов приведено в табл.1.1.

Таблица 1.1. **Сигналы интерфейса Centronics**

Контакт	Описание	*Направление* (относительно компьютера)
1	Строб данных (инверсная логика)	Выход
2	1-й бит данных (младший)	Выход
3	2-й бит данных	Выход
4	3-й бит данных	Выход
5	4-й бит данных	Выход
6	5-й бит данных	Выход
7	6-й бит данных	Выход
8	7-й бит данных	Выход

Продолжение табл. 1.1.

9	8-й бит данных (старший)	Выход
10	Подтверждение (инверсная логика)	Вход
11	Принтер занят	Вход
12	Конец бумаги	Вход
13	Выбор	Вход
14	Автоматический перевод строки (инверсная логика)	Выход
15	Ошибка, сбой (инверсная логика)	Вход
16	Инициализация принтера (инверсная логика)	Выход
17	Выбор ввода (инверсная логика)	Выход
18...25	Общий провод (возврат)	—

Интерфейс Centronics поддерживают большинство принтеров с параллельным интерфейсом, его отечественным аналогом является ИРПР-М, который может быть программно реализован на традиционном порте.

1.2. Традиционный LPT - порт

Традиционный (стандартный) порт SPP (Standart Parallel Port) является однонаправленным портом, на базе которого программно реализуется протокол обмена Centronics. Порт обеспечивает возможность создания запроса аппаратного прерывания по сигналу на входе Ack#. Сигналы порта выводятся на разъем DB-25S (розетка), который устанавливается непосредственно на плате адаптера (или системной плате), или подсоединен к ней плоским шлейфом.

Стандартный порт имеет три 8-битних регистра, расположенных по соседним адресам в пространстве ввода/вывода, начиная с базового адреса порта (BASE).

Data Register(DR) - регистр данных, адрес = BASE. Данные, которые записаны в этот порт, выводятся на выходные лини интерфейса. Данные, считанные из этого регистра, в зависимости от схемотехники адаптера соответствуют или ранее записанным данным, или сигналам

на тех же линиях, что не всегда одно и тоже. На некоторых старых адаптерах выходной буфер может выключаться перемычкой на плате.

Status Register(SR) - регистр состояния, 5-битный порт ввода сигналов состояния принтера (биты SR.4 - SR.7), адрес = BASE + 1. Бит SR.7 инвертируется - нулевому уровню сигнала соответствует единичное значение бита в регистре. Назначение бит регистра состояния:

SR.7 - BUSY - инверсное отображение состояния линии BUSY (11): при низком уровне сигнала на линии устанавливается единичное значение бита - разрешение на вывод следующего байта.

SR.6 - ACK (Acknowledge) - прямое отображение состояния линии ACK# (10): низкому уровню сигнала на лини соответствует нулевое значение бита.

SR.5 - PE (Paper End) - прямое отображение состояния линии PaperEnd (12). Единица соответствует сигналу об окончании бумаги.

SR.4 - SELECT - прямое отображение состояния линии Select (13). Единица соответствует сигналу о включении принтера.

SR.3 - ERROR - прямое отображение состояния линии Error (15). Ноль - ошибка при печати.

SR.2 - PIRQ - флаг прерывания по сигналу ACK# (только для порта PS/2). Бит сбрасывается, если сигнал ACK# вызвал аппаратное прерывание. Единичное значение устанавливается по аппаратному сбросу, а также после чтения регистра состояния.

SR [1:0] - зарезервированы.

Control Register(CR) - регистр управления, адрес = BASE + 2. Как и регистр данных, этот 4-битный порт вывода (биты (0-3)) разрешает запись и чтение, но его выходной буфер имеет тип "открытый коллектор". Это разрешает более корректно использовать линии дан-

ного регистра, как входные при программировании их в высокий уровень. Биты 0,1,3 инвертируются - единичному значению соответствует низкий уровень сигнала. Назначение бит регистра управления:

CR [7:6] - зарезервированные.

CR.5 - Direction - бит управления направлением передачи (для PS/2). Запись 1 переводит порт данных в режим ввода. При чтении состояние бита неопределенное.

CR.4 - AckInEn (Acknoledge Interrupt Enable) - единица разрешает прерывание по спаду сигнала на линии ACK# - сигнал запроса следующего байта.

CR.3 - SelectIn – единица соответствует низкому уровню сигнала на выходе SelectIn# (17) - сигнал, разрешающий работу принтера по интерфейсу Centronics.

CR.2 - Init - нуль соответствует низкому уровню сигнала на лини Init# (16) - сигнал аппаратного сброса принтера.

CR.1 - AutoLF - единица соответствует низкому уровню сигнала на линии AutoLF - сигнал на автоматический перевод строки (LF - Line Feed) по приему байта возврата каретки (CR - Carrige Return), также имеет названия AutoFD, AutoFDXT.

CR.0 - Strobe - единица соответствует низкому уровню сигнала на линии Strobe# (1) - сигнал стробирования выходных данных.

1.3. Расширения параллельного порта

Недостатки стандартного порта частично устраняли новые типы портов, появившиеся в компьютерах PS/2.

Двунаправленный порт 1 (Type 1 parallel port) - интерфейс, введенный в PS/2. Такой порт, кроме стандартного режима, может работать в режиме ввода или двунаправленном режиме. Прото-

кол обмена формируется программно, а для указания направления передачи в регистр управления порта введен специальный бит CR.5: 0 - буфер данных работает на вывод, 1 - на ввод. Не путайте этот порт, называемый также enhanced bi-directional, с EPP. Данный тип порта прижился и в обычных компьютерах.

Порт с прямым доступом к памяти (Type 3 DMA parallelport) применялся в PS/2 моделей 57, 90, 95. Был введен для повышения пропускной способности и разгрузки процессора при выводе на принтер. Программе, работающей с портом, требовалось только задать в памяти блок данных, подлежащих выводу, а затем вывод по протоколу Centronics производился без участия процессора.

Позже появились другие адаптеры LPT-портов, реализующие протокол обмена Centronics аппаратно - Fast Centronics. Некоторые из них использовали FIFO-буфер данных - Parallel Port FIFO Mode. Не будучи стандартизованными, такие порты разных производителей требовали использования собственных специальных драйверов. Программы, использующие прямое управление регистрами стандартных портов, не умели более эффективно их использовать. Такие порты часто входили в состав мультикарт VLB. Существуют их варианты с шиной ISA, в том числе встроенные.

1.4. Режимы работы параллельного порта

Стандарт на параллельный интерфейс IEEE 1284, принятый в 1994 году, определяет порты SPP, EPP и ECP. Стандарт определяет 5 режимов обмена данными, метод согласования режима, физический и электрический интерфейсы. Согласно IEEE 1284, возможны следующие режимы обмена данными через параллельный порт:

Режим совместимости (Compatibility Mode) - однонаправленный (вывод) по протоколу Centronics. Этот режим соответствует стандартному порту SPP.

Полубайтный режим (Nibble Mode) - ввод байта в два цикла (по 4 бита), используя для приема линии состояния. Этот режим обмена может использоваться на любых адаптерах.

Порты имеют 5 линий ввода состояния, используя которые периферийное устройство может посылать в хост байт тетрадами (nibble - полубайт, 4 бита) за два приема (рис.1.2). Сигнал Ack#, вызывающий прерывание, которое может использоваться в данном режиме, соответствует биту 6 регистра состояния. Это усложняет программные манипуляции с битами при сборке байта.

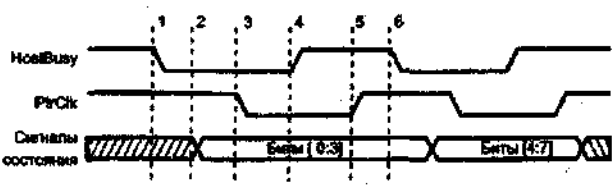


Рис. 1.2. Прием данных в полубайтном режиме

Прием байта данных в полубайтном режиме состоит из следующих фаз:

1. Хост сигнализирует о готовности приема данных установкой низкого уровня на линии HostBusy.

2. Периферийное устройство в ответ помещает тетраду на входные линии состояния.

3. Периферийное устройство сигнализирует о готовности тетрады установкой низкого уровня на линии PtrClk.

4. Хост устанавливает высокий уровень на линии HostBusy, указывая на занятость приемом и обработкой тетрады.

5. Периферийное устройство отвечает установкой высокого уровня на линии PtrClk.

6. Шаги 1-5 повторяются для второй тетрады.

Сигналы порта описаны в табл. 1.2.

Таблица 1.2. **Описание сигналов порта при работе в полубайтном режиме**

Контакт	Сигнал SPP	I/O	Описание
14	AutoFeed#	0	HostBusy - сигнал квитирования. Низкий уровень означает готовность к приему тетрады, высокий подтверждает прием тетрады.
17	Sdectln#	0	Высокий уровень указывает на обмен в режиме IEEE 1284 (в режиме SPP уровень низкий).
10	Ack#	I	PtrClk. Низкий уровень означает готовность тетрады, высокий – ответ на сигнал HostBusy.
11	Busy	I	Прием бита данных 3, затем бита 7
12	PE	I	Прием бита данных 2, затем бита 6
13	Sdect	I	Прием бита данных 1, затем бита 5
15	Error"	I	Прием бита данных 0, затем бита 4

Полубайтный режим сильно нагружает процессор, и поднять скорость обмена выше 50 Кбайт/с не удается. Безусловное его преимущество в том, что он работает на всех портах. Его применяют в тех случаях, когда поток данных невелик.

Байтный режим (Byte Mode) - ввод байта целиком, используя для приема линии данных. Этот режим работает только на портах, допускающих чтение выходных данных (Bi-Directional или PS/2 Type 1).

В этом режиме данные принимаются с использованием двунаправленного порта, у которого выходной буфер данных может отключаться установкой бита CR.5=1. Как и предыдущие, режим является программно-управляемым – все сигналы анализируются и устанавливаются драйвером. Сигналы порта описаны в табл.1.3. Временные диаграммы представлены на рис.1.3.

Таблица 1.3. **Описание сигналов порта в режиме Byte Mode**

Контакт	Сигнал SPP	Имя в байтном режиме	I/O	Описание
*	Strobe"	HostClk	O	Импульс (низкого уровня) подтверждает прием байта в конце каждого цикла
14	AutoFeed#	HostBusy	O	Сигнал квитирования. Низкий уровень означает готовность хоста принять байт; высокий уровень устанавливается по приему байта
17	Selecting	1284Active	O	Высокий уровень указывает на обмен в режиме IEEE 1284 (в режиме SPP уровень низкий)
16	Init#	Init	O	Не используется; установлен высокий уровень
10	Ack#	PtrClk	I	Устанавливается в низкий уровень для индикации действительности данных на линиях Data [0:7]. Низкий уровень устанавливается в ответ на сигнал HostBusy
11	Busy	PtrBusy	I	Состояние занятости прямого канала
12	PE	AckDataReq	I	Устанавливается ПУ для указания на наличие обратного канала передачи
13	Select	Xflag	I	Флаг расширяемости
15	Error#	DataAvail#	I	Устанавливается ПУ для указания на наличие обратного канала передачи
2-9	Data [0:7]	Data [0:7]	I/ O	Двунаправленный (прямой и обратный) канал данных

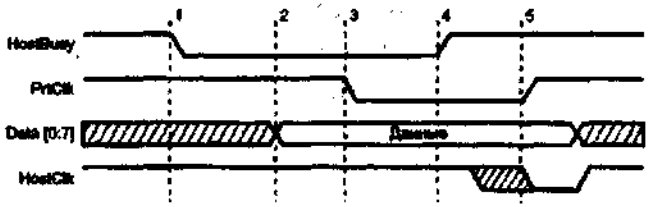


Рис.1.3. Прием данных в байтном режиме

Фазы приема байта данных:

1. Хост сигнализирует о готовности приема данных установкой низкого уровня на линии HostBusy.

2. Периферийное устройство в ответ помещает байт данных на линии Data [0:7].

3. Периферийное устройство сигнализирует о действительности байта установкой низкого уровня на линии PtrClk.

4. Хост устанавливает высокий уровень на линии HostBusy, указывая на занятость приемом и обработкой байта.

5. Периферийное устройство отвечает установкой высокого уровня на линии PtrClk.

6. Хост подтверждает прием байта импульсом HostCIR.

7. Шаги 1-6 повторяются для каждого следующего байта.

Побайтный режим позволяет поднять скорость обратного канала до скорости прямого канала в стандартном режиме. Однако он может работать только на двунаправленных портах.

Режим EPP (Enhanced Parallel Port) (EPP Mode) - двунаправленный обмен данными. Управляющие сигналы интерфейса генерируются аппаратно во время цикла обращения к порту. Эффективен при работе с устройствами внешней памяти и адаптерами локальных сетей.

Протокол EPP обеспечивает четыре типа циклов обмена:

- запись данных;
- чтение данных;
- запись адреса;
- чтение адреса.

Назначение циклов записи и чтения данных очевидно. Адресные циклы используются для передачи адресной, канальной и управляющей информации. Циклы обмена данными отличаются от адресных циклов применяемыми стробирующими сигналами. Назначение сигналов порта EPP и их связь с сигналами SPP объясняются в табл.1.4.

Таблица 1.4. **Назначение сигналов порта EPP**

Контакт	Сигнал SPP	Имя в EPP	I/O	Описание
1	Strobe"	Write#	0	Низкий уровень – цикл записи, высокий – цикл чтения
14	AutoLF#	DataStb#	0	Строб данных. Низкий уровень устанавливается в циклах передачи данных
17	Selecting	AddrStb#	o	Строб адреса. Низкий уровень устанавливается в адресных циклах
16	Init"	Reset"	o	Сброс ПУ (низким уровнем)
10	Ack#	INTR#	I	Прерывание от ПУ
11	Busy	Wait#		Сигнал квотирования. Низкий уровень разрешает начало цикла (установку строба в низкий уровень), переход в высокий - разрешает завершение цикла (снятие строба)
2-9	Data [0:7]	AD[0:7]	I/O	Двунаправленная шина адреса/данных
12	PaperEnd	AckDataReq*	I	Используется по усмотрению разработчика периферии
13	Select	Xflag*	I	Используется по усмотрению разработчика периферии
15	Error#	DataAvail#*	'	Используется по усмотрению разработчика периферии

EPP-порт имеет расширенный набор регистров (табл.1.5), который занимает в пространстве ввода/вывода 5-8 смежных байт.

Таблица 1.5. **Набор регистров порта в режиме EPP**

Имя регистра	Смещение	Режим	R/W	Описание
SPP Data Port	+0	SPP/EPP	W	Регистр данных SPP
SPP Status Port	+1	SPP/EPP	R	Регистр состояния SPP
SPP Control Port	+2	SPP/EPP	W	Регистр управления SPP
EPP Address Port	+3	EPP	R/W	Регистр адреса EPP. Чтение или запись в него генерирует связанный цикл чтения или записи адреса EPP.
EPP Data Port	+4	EPP	R/W	Регистр данных EPP. Чтение (запись) генерирует связанный цикл чтения (записи) данных EPP
Not Defined	+5...+7	EPP	N/A	В некоторых контроллерах могут использоваться для 16-32-битных операций ввода/вывода

В отличие от программно-управляемых режимов, внешние сигналы LPT-порта для каждого цикла обмена формируются аппаратно по одной операции записи или чтения в регистр порта. На рис.1.4 приведена диаграмма цикла записи данных, иллюстрирующая внешний цикл обмена, вложенный в цикл записи системной шины процессора (иногда эти циклы называют связанными). Адресный цикл записи отличается от цикла данных только стробом внешнего интерфейса.

Цикл записи данных состоит из следующих фаз:

1. Программа выполняет цикл вывода (IOWR#) в порт 4 (EPP Data Port).

2. Адаптер устанавливает сигнал Write* (низкий уровень), и данные помещаются на выходную шину LPT-порта.

3. При низком уровне Wait# устанавливается строб данных.

4. Порт ждет подтверждения от ПУ (перевода Wait# в высокий уровень).

5. Снимается строб данных - внешний EPP-цикл завер-шается.

6. Завершается процессорный цикл вывода.

7. Периферийное устройство устанавливает низкий уровень Wait#, указывая на возможность начала следующего цикла.

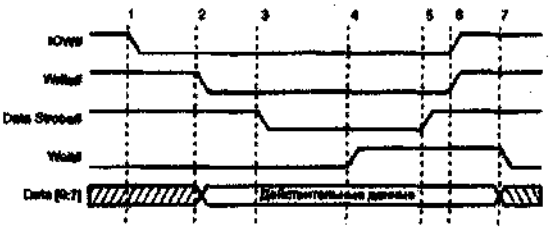


Рис. 1.4. Цикл записи данных EPP

Пример адресного цикла чтения приведен на рис.1.5. Цикл чтения данных отличается только применением другого стробирующего сигнала.

Рис. 1.5. Адресный цикл чтения EPP

Главной отличительной чертой EPP является выполнение внешней передачи во время одного процессорного цикла ввода/вывода. Это позволяет достигать высоких скоростей обмена (0,5...2 Мбайт/с). Протокол блокированного квитирования (interlocked handshakes) позволяет автоматически настраиваться на скорость обмена, доступную и хосту, и периферийному устройству. Периферийное устройство может регулировать длительность всех фаз обмена с помощью всего лишь одного сигнала Wait#. Протокол автоматически

подстраивается под длину кабеля – вносимые задержки приведут только к удлинению цикла.

Естественно, периферийное устройство не должно "подвешивать" процессор на шинном цикле обмена. Это гарантирует механизм тайм-аутов PC, который принудительно завершает любой цикл обмена, длящийся более 15 мкс. В ряде реализации EPP за тайм-аутом интерфейса следит сам адаптер - если периферийное устройство не отвечает в течение определенного времени (5 мкс), цикл прекращается и в дополнительном (не стандартизованном) регистре состояния адаптера фиксируется ошибка.

Важной чертой EPP является то, что обращение процессора к ПУ осуществляется в реальном времени - нет буферизации. Драйвер способен отслеживать состояние и подавать команды в точно известные моменты времени. Циклы чтения и записи могут чередоваться в произвольном порядке или идти блоками. Такой тип обмена удобен для регистро-ориентированных периферийных устройств или устройств, работающих в реальном времени, - сетевых адаптеров, устройств сбора информации и управления и т. п.

Режим ECP (Extended Capability Port) (ECP Mode) - двунаправленный обмен данными с возможностью аппаратного сжатия данных по методу RLE (Run Length Encoding) и использования FIFO - буферов DMA. Управляющие сигналы интерфейса генерируются аппаратно.

В компьютерах с LPT-портом на системной плате режим SPP, EPP, ECP или их комбинация задается в BIOS Setup. Режим совместимости полностью соответствует стандартному порту SPP.

Компрессия в реальном времени по методу RLE (Run-Length Encoding) позволяет достичь коэффициента сжатия 64:1 при передаче

растровых изображений, которые имеют длинные строки повторяющихся байт. Компрессию можно использовать, только если ее поддерживает и хост, и периферийное устройство. Канальная адресация ECP применяется для адресации множества логических устройств, входящих в одно физическое. Например, в комбинированном устройстве факс/принтер/модем, подключаемом только к одному параллельному порту, возможен одновременный прием факса и печать на принтере. В режиме SPP, если принтер установит сигнал занятости, канал будет занят данными, пока принтер их не примет. В режиме ECP программный драйвер просто адресуется к другому логическому каналу того же порта. Протокол ECP переопределяет сигналы SPP (табл. 1.6).

Таблица 1.6. **Описание сигналов порта в режиме ECP**

Контакт	Сигнал SPP	Имя в ECP	I/O	Описание
1	Strobe"	HostClk	O	Используется в паре с PeriphAck для передачи в прямом направлении (вывод)
14	AutoLF#	HostAck	O	Указывает тип цикла (команда/данные) при передаче в прямом направлении. Используется в паре с PeriphClk для передачи в обратном направлении
17	Selecting	1284Active	O	Высокий уровень указывает на обмен в режиме IEEE 1284 (в режиме SPP уровень низкий)
16	Init#	ReverseRequest	O	Низкий уровень переключает канал на передачу в обратном направлении
10	Ack#	PeriphQk	I	Используется в паре с HostAck для передачи в обратном направлении
11	Busy	PeriphAck	I	Используется в паре с HostClk для передачи в обратном направлении. Индицирует тип команда/данные при передаче в обратном направлении
12	PaperEnd	AckReverse#	I	Переводится в низкий уровень как подтверждение сигналу ReverseRequest#
13	Select	Xflag	I	Флаг расширяемости
15	Error#	PeriphRequest#	I	Устанавливается ПУ для указания на доступность (наличие) обратного канала передачи*
2-9	Data [0:7]	Data [0:7]	I/O	Двунаправленный канал данных

Адаптер ECP тоже генерирует внешние протокольные сигналы аппаратно, но его работа существенно отличается от режима EPP.

На рис.1.6 приведена диаграмма двух циклов прямой передачи: за циклом данных следует командный цикл. Тип цикла задается уровнем на линии HostAck: в цикле данных - высокий, в командном цикле - низкий. В командном цикле байт может содержать канальный адрес или счетчик RLE. Отличительным признаком является бит 7 (старший): если он нулевой, то биты 0-6 содержат счетчик RLE (0-127), если единичный - то канальный адрес. На рис.1.7 показана пара циклов обратной передачи.

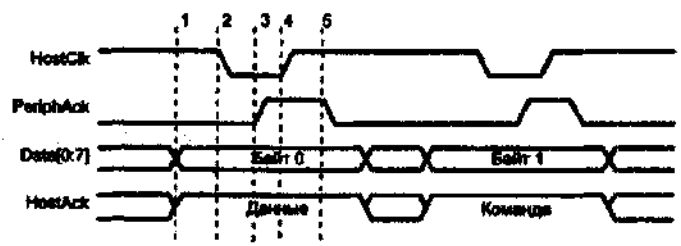


Рис.1.6. Диаграмма двух циклов прямой передачи

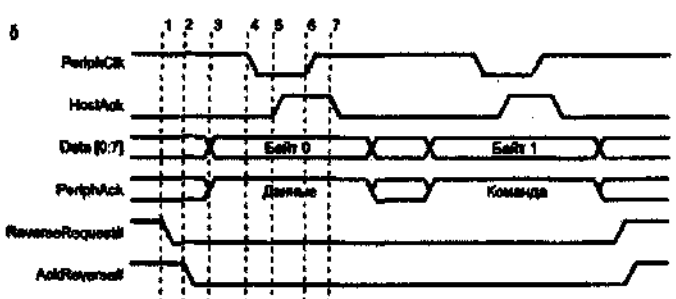


Рис. 1.7. Передача в режиме ECP: а - прямая, б - обратная

В отличие от диаграмм обмена EPP, на рис.1.7 не приведены сигналы циклов системной шины процессора. В данном режиме обмен программы с периферийным устройством разбивается на два относительно независимых процесса, которые связаны через FIFO-буфер. Обмен драйвера с FIFO-буфером может осуществляться с использованием как DMA, так и программного ввода-вывода. Обмен периферийного устройства с буфером аппаратно выполняет адаптер ECP. Драйвер в режиме ECP не имеет информации о точном состоянии процесса обмена, но здесь обычно важно только то, завершен он или нет. Прямая передача данных на внешнем интерфейсе состоит из следующих шагов:

1. Хост помещает данные на шину канала и устанавливает признак цикла данных (высокий уровень) или команды (низкий уровень) на линии HostAck.

2. Хост устанавливает низкий уровень на линии HostClk, указывая на действительность данных.

3. Периферийное устройство отвечает установкой высокого уровня на линии PeriphAck.

4. Хост устанавливает высокий уровень линии HostClk, и этот перепад может использоваться для фиксации данных в периферийное устройство.

5. Периферийное устройство устанавливает низкий уровень на линии PeriphAck для указания на готовность к приему следующего байта.

Поскольку передача в ECP происходит через FIFO-буферы, которые могут присутствовать на обеих сторонах интерфейса, важно понимать, на каком этапе данные можно считать переданными. Данные считаются переданными на шаге 4, когда линия HostClk переходит в

высокий уровень. В этот момент модифицируются счетчики переданных и принятых байт. В протоколе ECP есть условия, вызывающие прекращение обмена между шагами 3 и 4. Тогда эти данные не должны рассматриваться как переданные.

Из рис.1.7 видно и другое отличие ECP от EPP. Протокол EPP позволяет драйверу чередовать циклы прямой и обратной передачи, не запрашивая подтверждения на смену направления. В ECP смена направления должна быть согласована: хост запрашивает реверс установкой ReverseRequest#, после чего он должен дождаться подтверждения сигналом AckReverse#. Поскольку предыдущий цикл мог выполняться по прямому доступу, драйвер должен дождаться завершения прямого доступа или прервать его, выгрузить буфер FIFO, определив точное значение счетчика переданных байт, и только после этого запрашивать реверс.

Обратная передача данных состоит из следующих шагов:

1. Хост запрашивает изменение направления канала, устанавливая низкий уровень на линии ReverseRequest#.

2. Периферийное устройство разрешает смену направления установкой низкого уровня на линии AckReverse#.

3. Периферийное устройство помещает данные на шину канала и устанавливает признак цикла данных (высокий уровень) или команды (низкий уровень) на линии PeriphAck.

4. Периферийное устройство устанавливает низкий уровень на линии PeriphClk, указывая на действительность данных.

5. Хост отвечает установкой высокого уровня на линии HostAck.

6. Периферийное устройство устанавливает высокий уровень линии PeriphClk; этот перепад может использоваться для фиксации данных хостом.

7. Хост устанавливает низкий уровень на линии HostAck для указания на готовность к приему следующего байта.

1.5. Физический и электрический интерфейсы

Стандарт IEEE 1284 определяет физические характеристики приемников и передатчиков сигналов. Спецификации стандартного порта не задавали типов выходных схем, предельных значений величин нагрузочных резисторов и емкости, вносимой цепями и проводниками. На относительно невысоких скоростях обмена разброс этих параметров не вызывал проблем совместимости. Однако расширенные (функционально и по скорости передачи) режимы требуют четких спецификаций. IEEE 1284 определяет два уровня интерфейсной совместимости. Первый уровень (Level I) определен для устройств медленных, но использующих смену направления передачи данных. Второй уровень (Level II) определен для устройств, работающих в расширенных режимах, с высокими скоростями и длинными кабелями. К передатчикам предъявляются следующие требования:

- уровни сигналов без нагрузки не должны выходить за пределы -0,5... +5,5 В.

- уровни сигналов при токе нагрузки 14 мА должны быть не ниже +2,4В для высокого уровня и не выше +0,4 В для низкого уровня на постоянном токе.

- выходной импеданс, измеренный на разъеме, должен составлять 50±5Ом. Для обеспечения заданного импеданса используют последовательные резисторы в выходных цепях передатчика. Согласование импеданса передатчика и кабеля снижает уровень импульсных помех.

- скорость нарастания (спада) импульса должна находиться в пределах 0,05-0,4 В/нс.

Требования к приемникам:

- допустимые пиковые значения сигналов -2,0...+7,0 В.

- пороги срабатывания должны быть не выше 2 В для высокого уровня и не ниже 0,8 В для низкого.

- приемник должен иметь гистерезис в пределах 0,2...1,2 В (гистерезисом обладают специальные микросхемы - триггеры Шмитта).

- входной ток микросхемы (втекающий и вытекающий) не должен превышать 20 мкА, входные линии соединяются с шиной питания +5В резистором 1,2 кОм.

- входная емкость не должна превышать 50 пФ.

Когда появилась спецификация ECP, фирма Microsoft рекомендовала применение динамических терминаторов на каждую линию интерфейса. Однако в настоящее время следуют спецификации IEEE 1284, в которой динамические терминаторы не применяются.

Традиционные интерфейсные кабели имеют от 18 до 25 проводов, в зависимости от числа проводников цепи GND. Эти проводники могут быть как перевитыми, так и нет. К экранированию кабеля жестких требований не предъявлялось. Такие кабели вряд ли будут надежно работать на скорости передачи 2 Мбайт/с и при длине более 2 м.

Стандарт IEEE 1284 регламентирует свойства кабелей.

- все сигнальные линии должны быть перевитыми с отдельными обратными (общими) проводами.

- каждая пара должна иметь импеданс 62±6 Ом в частотном диапазоне 4-16 МГц.

- уровень перекрестных помех между парами не должен превышать 10%.

- кабель должен иметь экран, покрывающий не менее 85% внешней поверхности. На концах кабеля экран должен быть окольцован и соединен с контактом разъема. [3]

1.6. Конфигурирование LPT-портов

Управление параллельным портом разделяется на два этапа – предварительное конфигурирование (Setup) аппаратных средств порта и текущее (оперативное) переключение режимов работы прикладным или системным программным обеспечением. Оперативное переключение возможно только в пределах режимов, разрешенных при конфигурировании. Этим обеспечивается возможность согласования аппаратуры с программным обеспечением и блокирования ложных переключении вызванных некорректными действиями программы.

Конфигурирование LPT-порта зависит от его исполнения. Порт, расположенный на плате расширения (мультикарте), устанавливаемой в слот ISA или ISA+VLB, конфигурируется джамперами на самой плате. Порт на системной плате конфигурируется через BIOS Setup. Конфигурированию подлежат следующие параметры:

- базовый адрес - 3BCh, 378h или 278h. При инициализации BIOS проверяет наличие портов по адресам именно в этом порядке и, соответственно, присваивает обнаруженным портам логические имена LPT1, LPT2, LPT3. Адрес 3BCh имеет адаптер порта, расположенный на плате MDA или HGC. Большинство портов по умолчанию конфигурируются на адрес 378h и могут переключаться на 278h.

- используемая линия запроса прерывания: для LPT - IRQ7, для LPT2 - IRQ5. Традиционно прерывания от принтера не используются, и этот дефицитный ресурс можно сэкономить. Однако при использовании скоростных режимов ECP (или Fast Centronics) работа через

прерывания может заметно повысить производительность и снизить загрузку процессора.

- использование канала DMA для режимов ECP и Fast Centronics - разрешение и номер канала DMA (по умолчанию - 3).

Режимы работы порта:

- SPP - порт работает только в стандартном однонаправленном программно-управляемом режиме.

- PS/2, он же Bi-Directional - отличается от SPP возможностью реверса канала (установкой СЯ.5=7).

- Fast Centronics - аппаратное формирование протокола Centronics с использованием FIFO-буфера и, возможно, DMA.

- EPP - в зависимости от использования регистров порт работает в режиме SPP или EPP.

- ECP - по умолчанию включается в режим SPP или PS/2, записью в ECR может переводиться в любой режим ECP, но перевод в EPP записью в ECR кода 100 не гарантируется.

- ECP+EPP - то же, что и ECP, но запись в ECR кода режима 100 переводит порт в EPP.

Выбор режима EPP, ECP или Fast Centronics сам по себе не приводит к повышению быстродействия обмена с подключенными к порту устройствами, а только дает возможность драйверу и устройствам установить оптимальный режим. Большинство современных драйверов и приложений пытаются использовать эффективные режимы, так что ограничивать их установкой простых режимов без веских оснований не стоит.

Принтеры и сканеры могут требовать режима ECP. Windows NT/2000/XP имеет системные драйверы для этого режима. В среде

DOS печать через ECP поддерживается только специальным загружаемым драйвером.

Большинство современных периферийных устройств, подключаемых к LPT-порту, поддерживает стандарт 1284 и PnP. Для поддержки этих функций компьютером с аппаратной точки зрения достаточно иметь контроллер интерфейса, поддерживающий стандарт 1284. Если подключаемое устройство поддерживает PnP, оно по протоколу согласования режимов 1284 способно "договориться" с портом о возможных режимах обмена. Подключенное устройство должно сообщить операционной системе (ОС) все необходимые сведения о себе – идентификатор производителя, модель и набор поддерживаемых команд. Более подробная информация может содержать идентификатор класса, подробное описание и идентификатор устройства, с которым обеспечивается совместимость. В соответствии с принятой информацией ОС может предпринять действия по установке требуемого программного обеспечения для поддержки данного устройства.

1.7. Неисправности и тестирование параллельных портов

Тестирование параллельных портов разумно начинать с проверки их наличия в системе. Список адресов установленных портов появляется в таблице, выводимой BIOS на экран перед загрузкой ОС. Список можно посмотреть и с помощью тестовых программ или прямо в BIOS Data Area с помощью отладчика, или с помощью диспетчера устройств. Если BIOS обнаруживает меньше портов, чем установлено физически, скорее всего, двум портам присвоен один адрес. При этом работоспособность ни одного из конфликтующих портов не гарантируется: они будут одновременно выводить сигналы, но при чтении состояния конфликт на шине скорее всего приведет к искажению дан-

ных. Программное тестирование порта без диагностической заглушки (Loop Back) не покажет ошибок, поскольку при этом читаются данные выходных регистров, а они у всех конфликтующих (по отдельности исправных) портов совпадут. Именно такое тестирование производит BIOS при проверке на наличие портов. Разбираться с такой ситуацией следует, последовательно устанавливая порты и наблюдая за адресами, появляющимися в списке.

Если физически установлен только один порт, а BIOS его не обнаруживает, то либо порт отключен при конфигурировании, либо он вышел из строя (скорее всего из-за нарушений правил подключения). Иногда вам везет, и неисправность устраняется при "передергивании" платы в слоте - там возникают проблемы с контактами.

Тестирование портов с помощью диагностических программ позволяет проверить выходные регистры, а при использовании специальных заглушек - и входные линии. Поскольку количество выходных линий порта (12) и входных (5) различно, то полная проверка порта с помощью пассивной заглушки принципиально невозможна. Разные программы тестирования требуют применения разных заглушек (рис.1.8).

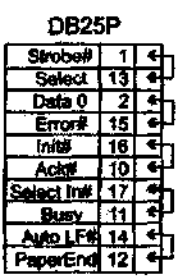


Рис. 1.8. Схема заглушки для тестирования LPT-порта программой Checkit

Большинство неприятностей при работе с LPT-портами связано с разъемами и кабелями. Для проверки порта, кабеля и принтера можно воспользоваться специальными тестами из популярных диагностических программ (Checkit, PCCheck и т. п.), а можно и попытаться вывести на принтер какой-либо символьный файл.

- если вывод файла с точки зрения DOS проходит (копирование файла на устройство с именем LPTn или PRN совершается быстро и успешно), а принтер (исправный) не напечатал ни одного символа - скорее всего, это обрыв (или отсутствие контакта) цепи Strobe.

- если принтер находится в состоянии OnLine, но появляется сообщение о его неготовности, причину следует искать в линии Busy.

- если принтер, подключенный к порту, в стандартном режиме (SPP) печатает нормально, а при переходе в ECP начинаются сбои, следует проверить кабель – соответствует ли он требованиям IEEE 1284. Дешевые кабели с не перевитыми проводами нормально работают на скоростях 50-100 Кбайт/с, но при скорости 1-2 Мбайт/с, обеспечиваемой ECP, могут не работать, особенно при длине более 2 м.

- если при установке драйвера PnP-принтера появилось сообщение о необходимости применения "двунаправленного кабеля", проверьте наличие связи контакта 17 разъема DB-25 с контактом 36 разъема Centronics. Хотя эта связь изначально предусматривалась, в ряде кабелей она отсутствует.

- если принтер искажает информацию при печати, возможен обрыв (или замыкание) линий данных. В этом случае удобно воспользоваться файлом, содержащим последовательность кодов всех печатных символов.[3]

2. ПРОГРАММИРОВАНИЕ LPT-ПОРТА

2.1. Установка драйвера giveio.sys

Операционная система Windows NT/2000/XP имеет ряд программ, которые не позволяют напрямую обращаться к аппаратным ресурсам, в частности к LPT-порту. Попытка выполнения программы, в которой имеется обращение к какому-либо регистру порта, вызовет ошибку времени выполнения. Для преодоления этой проблемы необходимо установить специальный драйвер, а в написанной программе этот драйвер подключить.

Существует целый ряд специальных драйверов, установив которые разработчик сможет получить доступ к аппаратным ресурсам компьютера. Одним из наиболее популярных драйверов является giveio.sys, который можно найти в Internet. Для установки этого драйвера необходимо иметь права администратора и выполнить следующие действия: скопировать файл giveio.sys в директорию c:\windows\system32\drivers. Запустить программу Loaddrv.exe, вид которой представлен на рис.2.1, и в поле Full pathname of driver указать путь к файлу драйвера. После этого следует нажать на кнопку Install.

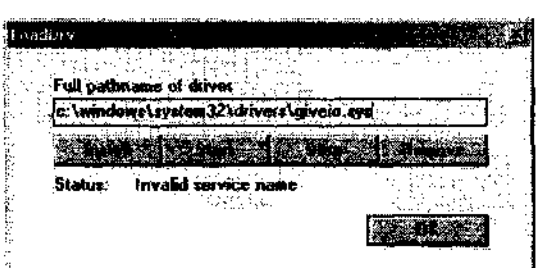


Рис.2.1. Окно программы LoadDrv

После установки драйвера в нижней части диалогового окна появится соответствующее сообщение. Затем следует перезагрузить компьютер. Если драйвер установить не удалось, появится сообщение о причине ошибки.

Теперь драйвер необходимо запустить. Для этого используется диспетчер устройств (рис.2.2). Для того, чтобы увидеть полный список установленных драйверов, необходимо в меню «Вид» установить флажок «Показывать скрытые устройства». Установленный драйвер можно найти в узле «Драйверы устройств не Plug and Play».

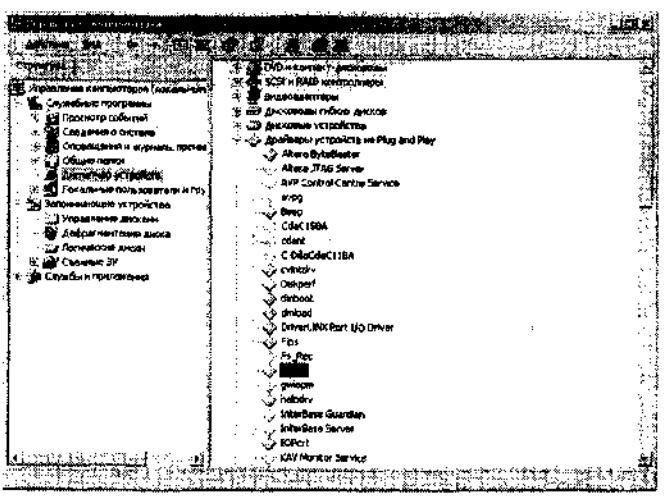


Рис.2.2. Окно «Диспетчера устройств»

Далее, выделив узел «giveio», в контекстном меню выбрать пункт «Свойства». Появится диалоговое окно, показанное на рис.2.3. На закладке «Драйвер» следует нажать на кнопку «Запустить», а в списке

«Автозагрузка» выбрать пункт «Автомат». В таком случае драйвер автоматически будет запускаться при загрузке операционной системы.

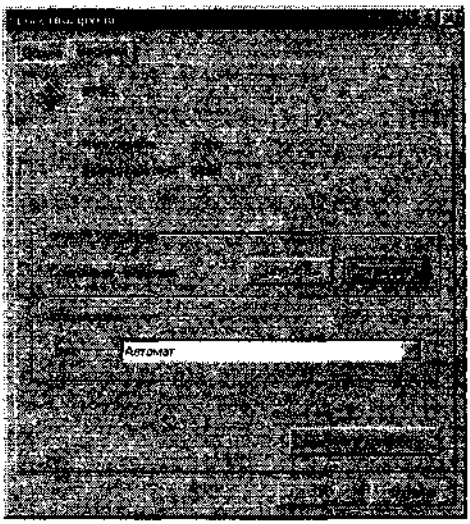


Рис.2.3. Окно свойств драйвера

2.2. Управление состоянием линий LPT-порта

Для изменения уровней сигналов на выводах порта используется обращение к регистру данных и регистру управления портом.

Пример 2.1. Используя среду разработки Visual Studio .NET создать программу, которая позволяла бы устанавливать требуемые сигналы (тактовый сигнал, сигналы данных) на LPT-порту.

Пояснение. Порядок действий:

1. Запускаем среду разработки Visual Studio .NET. При создании нового проекта Visual Studio .NET автоматически создает рабочее пространство (Solution) и помещает в него новый проект (Project). Для

создания нового рабочего проекта необходимо выполнить следующие операции:

a. В меню *File > New* выбирается команда *Project* (рис.2.4).

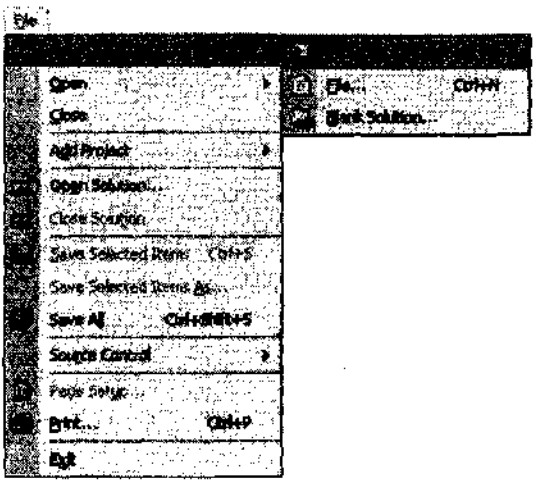


Рис.2.4. Создание нового проекта

b. В окне диалога *New Project*, которое появилось, в окне *Project Type* необходимо раскрыть узел дерева под именем *Visual C++ Projects* и выбрать узел *Win32 Projects*.

c. В окне *Templates* выбирается тип проекта *MFC Application*.

d. В окне *Name* задается имя проекта *LPTTest*.

e. В окне *Location* задается или оставляется без изменений расположение новой директории с файлами рабочего пространства (рис.2.5).

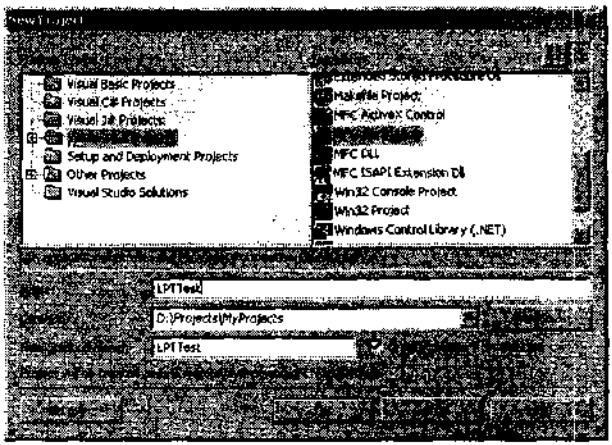


Рис.2.5. Задание названия проекта

После нажатия на кнопку «ОК» запускается мастер создания нового проекта, который позволяет легко выполнить первоначальные настройки.

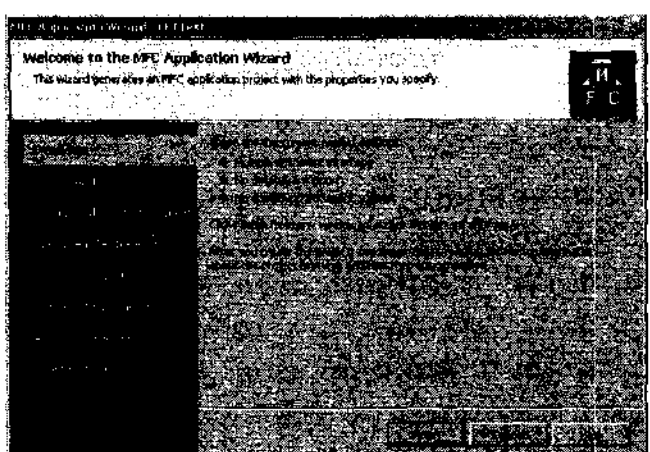


Рис.2.6. Окно мастера создания проекта

В нашем примере приложение создается на основе диалогового окна. Поэтому на втором этапе создания проекта (рис.2.7) необходимо выбрать пункт «Dialog based».

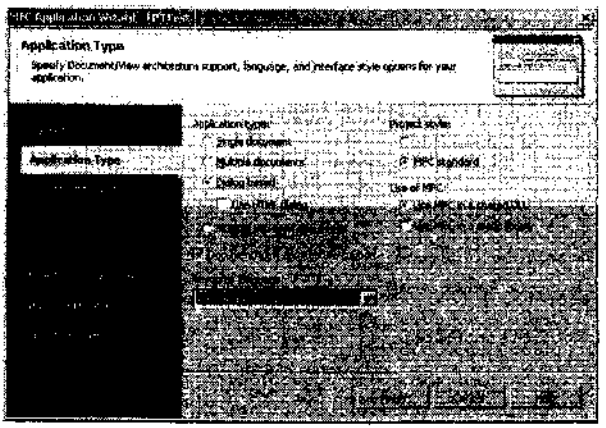


Рис.2.7. Окно выбора типа приложения

Также имеется возможность настроить вид диалогового окна, в частности указать тип границы, наличие кнопок развертки диалогового окна на весь экран и свертки на панель управления. Эти параметры можно оставить по умолчанию (рис.2.8).

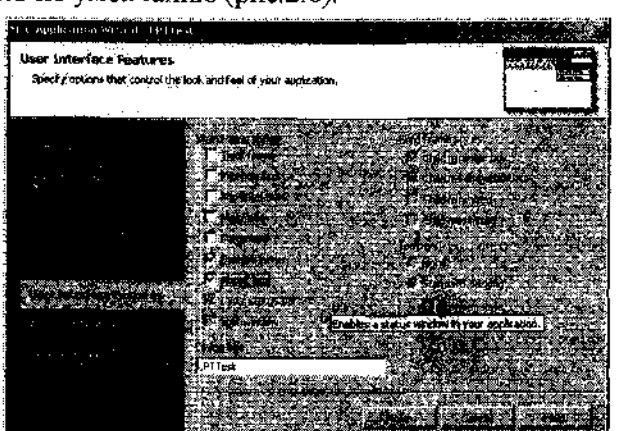


Рис.2.8. Окно настроек вида диалогового окна

После того, как настройки выполнены, следует нажать на кнопку «Finish». Создается новое рабочее пространство и проект. В проекте имеется три класса: CAboutDlg – диалоговое окно с информацией о разработчике программы, CLPTTestApp – класс приложения и CLPTTestDlg – класс главного диалогового окна программы (рис.2.9).

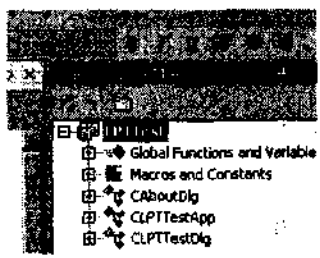


Рис.2.9. Классы, созданные мастером

Именно в классе CLPTTestDlg будет содержаться весь код программы. Кроме того, создается файл ресурсов, в котором находится форма. В дальнейшем, на этой форме будут размещаться элементы управления. В файле LPTTestDlg.cpp также необходимо подключить заголовочный файл. Для этого в начале файла добавить следующую строчку: #include "conio.h". В этом файле находится функция _outp(...), которая будет использоваться для работы с регистрами порта.

2. Проектируем внешний вид диалогового окна. Для этого используется панель инструментов (Toolbox). Путем перетаскивания нужных элементов управления, создается диалоговое окно, вид которого представлен на рис.1.10.

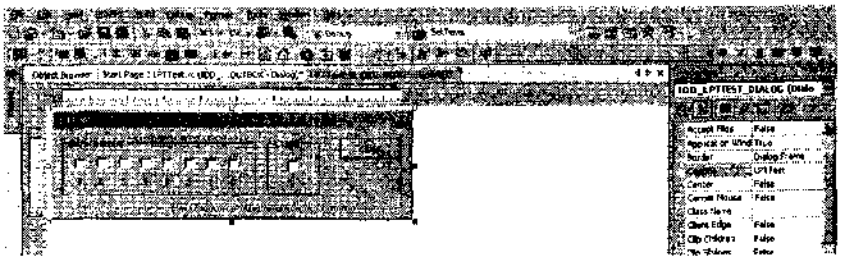


Рис.2.10. Спроектированное диалоговое окно программы

Для установки/сброса сигналов на выводах порта используется элемент управления Check Box. Последовательно перетягивая элементы управления на форму, следует расположить их справа налево, т.е. таким образом, чтобы их ID имели названия IDC_CHECK1 (крайний правый) – IDC_CHECK8 (крайний левый). После этого выносится еще один Check Box, который будет использоваться в качестве тактового (IDC_CHECK9). Проверить ID можно на панели «Properties», которая вызывается при выборе одноименной команды из контекстного меню. На этой же панели можно указать текст, который будет выводиться рядом с элементом управления (поле Caption). Если же текст выводить не надо, поле Caption остается незаполненным. На панели «Properties» можно настроить вид элемента управления (при необходимости). В большинстве случаев можно оставить параметры по умолчанию. Для оформления диалогового окна также используется элемент управления Group Box, который представляет собой рельефную рамку вокруг элементов управления Check Box.

3. Добавляем обработчики событий нажатия на элемент управления Check Box. Для этого выделяем первый элемент и из контекстного меню, которое появится после нажатия на правую кнопку

мыши, следует выбрать команду «Add Event Handler», как показано на рис.2.11.

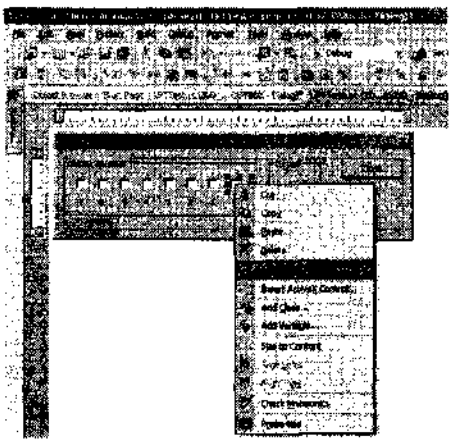


Рис.2.11. Процесс добавления событий

Появится диалоговое окно, представленное на рис.2.12.

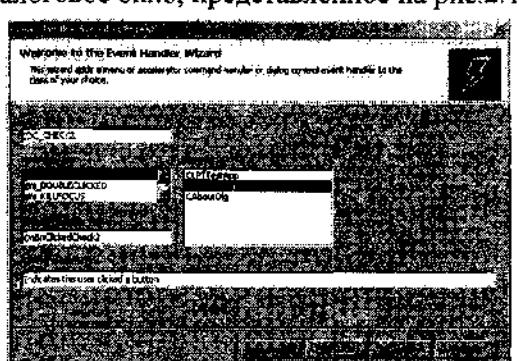


Рис.2.12. Выбор типа события

В этом диалоговом окне выбирается тип события (Message type), по которому будет вызываться соответствующий обработчик. В дан-

ном случае это нажатие на левую кнопку мыши. Также указывается название функции – обработчика события (поле Function handler name). Рекомендуется оставить все параметры по умолчанию. Нажав на кнопку «Add and Edit», весь код, необходимый для обработки события, автоматически будет добавлен в класс CLPTTestDlg. При этом также откроется окно текстового редактора и будет выделено название функции - обработчика события. После строчки «// TODO: Add your control notification handler code here», которая автоматически создается, следует вызвать функцию UpdateLPTSignals(). Эта функция далее будет описана. Аким образом, обработчик нажатия на Check Box выглядит следующим образом:

```
void CLPTTestDlg::OnBnClickedCheck1() // создается автоматически
{
UpdateLPTSignals(); // вызов функции обновления состояния сигналов порта
}
```

Такую же операцию следует повторить для оставшихся 7 элементов управления Check box. Ускорить процесс создания обработчиков события можно путем двойного нажатия левой кнопки мыши на каждом из элементов управления.

4. Реализация функции UpdateLPTSignals(). Для добавления этой функции в класс диалогового окна необходимо выделить класс CLPTTestDlg и вызвать контекстное меню, в котором из подменю «Add» выбрать подпункт «Add Function» (рис.2.13).

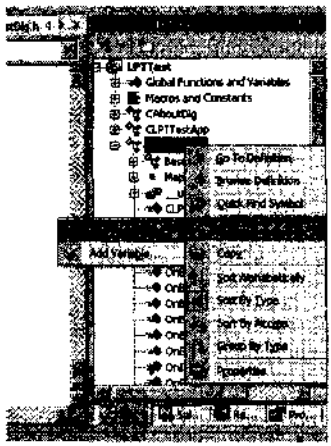


Рис 2.13. Добавление функции в класс

Появится диалоговое окно, показанное на рис.2.14.

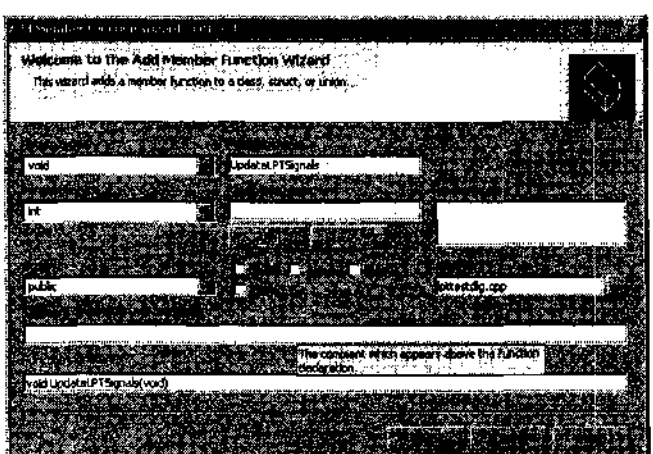


Рис.2.14. Окно добавления функции

Тип возвращаемого функцией (Return type) значения должен быть void. В поле Function Name необходимо ввести название созда-

ваемой функции, после чего нажать на кнопку Finish. Эта функция будет вызываться каждый раз, когда пользователь нажимает левую кнопку мыши на одном из 0 элементов управления Check box. Если элемент Check box находится в отмеченном состоянии, это означает, что на соответствующей ножке LPT-порта будет присутствовать лог.1. В противном случае – лог.0. Реализация функции имеет следующий вид:

```
void CLPTProgramDlg::OnCheck1()
{
char data=0; // переменная для формирования выходных сигналов
for(int i=IDC_CHECK1; i<=IDC_CHECK8; i++) // в цикле // анализируются состояния элементов управления
{
CButton* b=(CButton*)GetDlgItem(i);// получаем указатель на // элемент управления
data=data|b->GetCheck();// анализируется состояние // элемента управления: если отмечен, устанавливается 1 в // младшем разряде
data=data<<1; // сдвиг данных на 1 бит влево
}
_outp(0x378, data);// вывод байта данных в порт
}
```

Наиболее простой способ вывода данных в порт – это непосредственное обращение к регистрам порта. LPT–порт можно рассматривать как три регистра, два из которых доступны для записи (BASE – регистр данных, BASE+2 – регистр линий контроля), а один для чтения (BASE+1 – регистр состояния). Функция

```
int _outp(unsigned short addr, int data)
```

в качестве первого параметра (addr) принимает адрес регистра, в который будут записаны данные (второй параметр функции - data).

5. Формирование сигнала стробирования. Привязываем к элементу управления переменную. Для этого необходимо выделить соответствующий элемент управления и вызвать контекстное меню и выбрать подпункт «Add Variable». Появится диалоговое окно, показанное на рис.2.15.

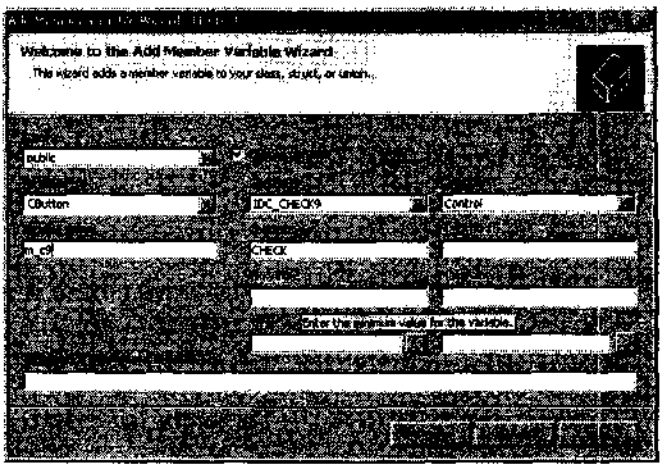


Рис.2.15. Окно добавления переменной

В поле «Variable name» вводится имя переменной. В данном случае это m_c9. Метка «Control variable» должна быть установлена. Это означает, что переменная связывается с элементом управления, ID которой указан в поле «Control ID». Это позволяет получить информацию о состоянии элемента управления. После того, как все параметры настроены, следует нажать на кнопку Finish.

Добавляем обработчик события нажатия на соответствующий элемент управления (IDC_CHECK9). Окончательный вариант обработчика события должен иметь вид:

```
void CLPTTestDlg::OnBnClickedCheck9()
{
if(m_c9.GetCheck()==1)   // проверка состояния элемента //
управления
{     // если отмеченный, выполняются функции ниже
_outp(0x37A, 1); // установка сигнала стробирования
_outp(0x37A, 0);// сброс сигнала стробирования
}
}
```

6. Подключение драйвера giveio. В файле LPTTestDlg.cpp необходимо найти функцию OnInitDialog(). Эта функция вызывается автоматически перед появлением диалога на экране, но после того как созданы все необходимые классы созданы. В конце функции, перед строчкой «return TRUE;», необходимо добавить следующий фрагмент кода:

```
HANDLE h; // дескриптор файла
h=CreateFile("\\\\.\\giveio", GENERIC_READ, 0, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
 // подключение драйвера
if(h==INVALID_HANDLE_VALUE) // если подключение не удалось
{
MessageBox("Giveio Error"); // появляется сообщение об ошибке
return false;
}
CloseHandle(h); // закрытие дескриптора файла
```

При запуске программы создается процесс (программа и является этим процессом). В функции OnInitDialog() вызывается функция CreateFile(...), в качестве параметров которой передается путь к драйверу (системный каталог), а также режим открытия. Функция возвращает дескриптор файла - переменная h. Проверив значение дескриптора можно выяснить, удалось ли подключить драйвер. В случае неудачи возвращается значение INVALID_HANDLE_VALUE. При отсутствии ошибок, все обращения к LPT-порту из данной программы (процесса) будут отслеживаться драйвером giveio. Таким образом появляется возможность работать с регистрами порта.

После компиляции программы появится диалоговое окно, показанное на рис.2.16.

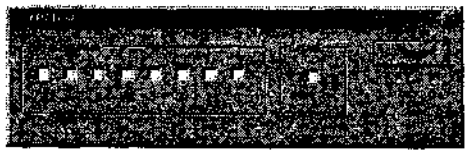


Рис.2.16. Окно программы

При установке/сбросе флажка будет изменяться уровень сигнала на соответствующем выводе порта LPT.

7. Проверка линий состояния порта. Для получения значений на линиях состояния порта необходимо прочитать данные из регистра BASE+1 (BASE – базовый адрес порта, в нашем примере 0x378). На форме необходимо разместить элемент управления «Button» (кнопка) и добавить обработчик события нажатия на эту кнопку (рис.2.17). В свойствах кнопки, в поле «Caption» можно ввести текст «Состояние». Это поможет пользователю понять функции, которые будут выполнены при нажатии на эту кнопку.

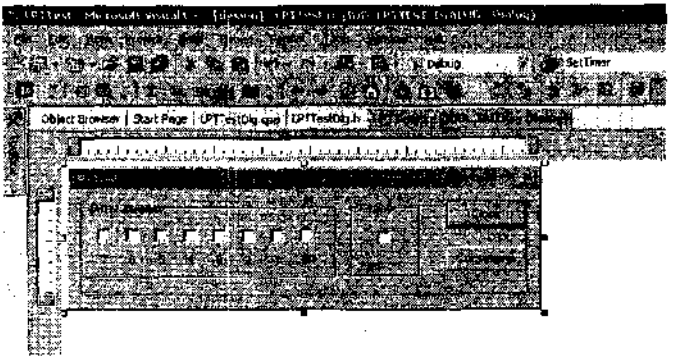


Рис.2.17 Модифицированное окно программы

Обработчик события нажатия на кнопку имеет следующий вид:

```
void CLPTTestDlg::OnBnClickedButton1()
{
int stat; // переменная для считывания данных из регистра
stat=_inp(0x379); // считывание данных.
// Параметр функции – адрес регистра
CString s; // строковая переменная // форматирование строки.
Знак отрицания «!» нужен для // инвертирования

s.Format("BUSY=%d, ACK=%d, PE=%d, SLCT=%d, ER-
ROR=%d",!((stat&0x80)>>7), (stat&0x40)>>6, (stat&0x20)>>5,
(stat&0x10)>>4, !((stat&0x08)>>3));
MessageBox(s);// вывод окна сообщения
}
```

После запуска команды и нажатии на кнопку «Состояние» появится окно сообщения, показанное на рис.2.18. Уровни сигналов могут отличаться от приведенных на рисунке.

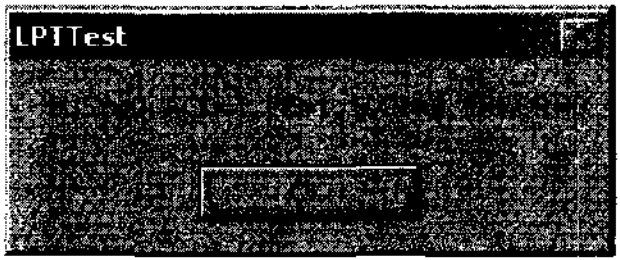


Рис.2.18. Данные о состоянии порта

Параллельный порт может использоваться для сопряжения контроллера с компьютером. Данные будут передаваться по 8-битной шине. Кроме того, порт имеет дополнительные выводы, уровень сигналов на которых может быть легко изменен программно. Некоторое неудобство программирования возникает при считывании данных с порта. Поскольку считывание возможно по 4 бита, формирование протокола и обработка данных должна выполняться программным способом.

2.3. Схема стенда для отладки программы

На рис.2.19 представлена схема стенда, который может быть использован при работе с LPT-портом для тестирования программного обеспечения.

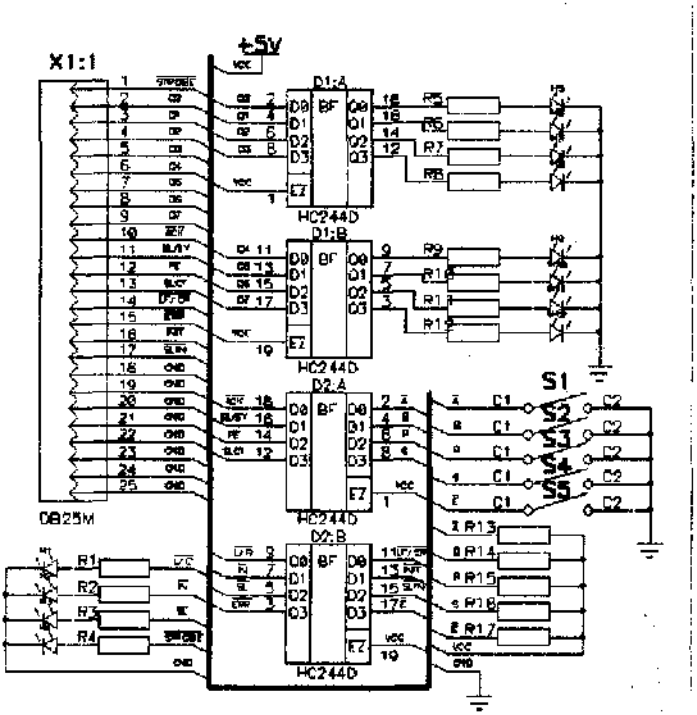


Рис.2.19. Схема стенда

Индикация состояний линий порта (шина данных, сигнал строба, линии контроля) осуществляется с помощью светодиодных индикаторов. Управление индикаторами выполняется через буферные микросхемы. Это позволяет не нагружать порт и избежать выхода порта из строя. Светодиоды через токоограничивающие резисторы R1-R12 сопротивлением 510 Ом подключаются к выходам буферов.

Для изменения уровней сигналов на входных линиях порта могут использоваться переключатели (или кнопки). Если ни одна из кнопок не нажата, на входах порта будут сигналы высокого уровня. Это достигается подтяжкой линий к напряжению питания через резисторы

R13-R17 сопротивлением 1 кОм. Это необходимо для исключения наводки случайных сигналов.

LPT – порт имеет большое количество входных и выходных линий. Поэтому его можно использовать для управления внешними устройствами, подключаемыми к компьютеру (например, реле, шаговым двигателем), а также использовать для отслеживания состояния дискретных датчиков.

Имеется возможность программной организации различных протоколов – например, последовательных синхронных SPI, I2C. Можно перевести выходные (или контрольные) биты LPT портов в режим генератора импульсов с частотой от 5 и до 600-3000 Гц. Максимальная частота зависит от быстродействия компьютера и порта.

2.4. Сопряжение микроконтроллера с LPT-портом

На рис.2 представлена схема сопряжения микроконтроллера AT90S8535 с LPT-портом по параллельной шине.

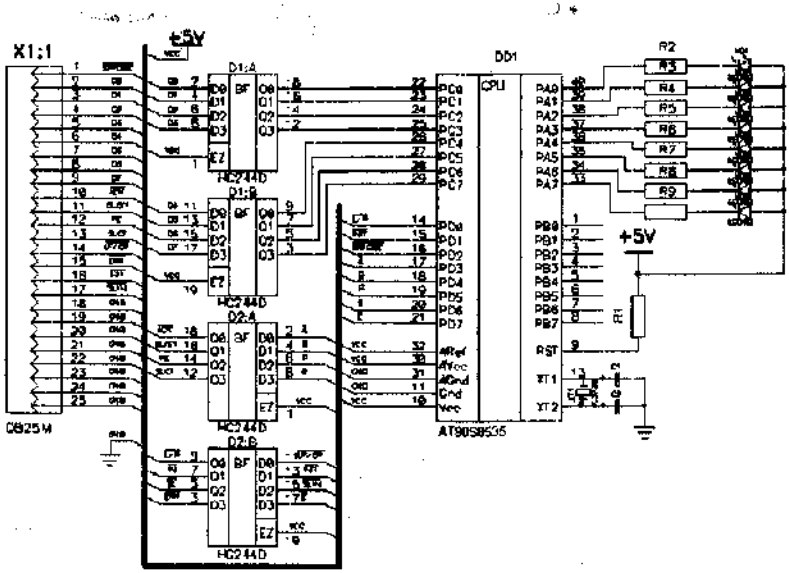


Рис.2.20. Схема сопряжения микроконтроллера с LPT-портом

Сигнал строба LPT-порта, который используется как синхронизирующий, соединен с входом внешнего прерывания микроконтроллера, а линии данных – с входами портом С. При появлении фронта синхронизирующего сигнала микроконтроллер может считывать данные с порта.

Пример 2.2. Написать программу для микроконтроллера которая считывает данные с LPT-порта компьютера по фронту сигнала стробирования и выводит значение в порт А (схема рис.2.20).

1. Запускаем среду разработки ImageCraft и создаем новый проект. Для этого из меню Project выбираем команду New (рис.2.21).

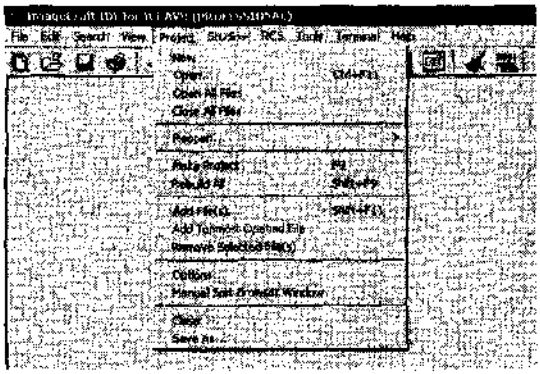


Рис.2.21. Середа разработки ImageCraft

Вводим название проекта (например LPTTest) и сохраняем его.

2. Добавляем файл в проект. Из меню File необходимо выбрать команду New. Откроется текстовый редактор. Сразу сохраняем файл на диск, выбрав из меню File команду Save As. Вводим название файла и обязательно расширение: LPTTest.c. После этого в окне навигатора нажимаем правую кнопку мыши напротив узла дерева Files и выбираем из контекстного меню команду Add File(s) (рис.2.22).

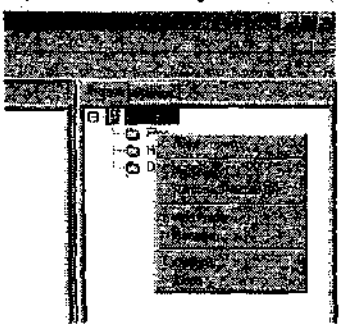


Рис.2.22. Добавление файла в проект

В появившемся окне открытия файла необходимо выбрать файл LPTTest.c

3. Набираем текст программы.

```
#include <io8535v.h> // подключение заголовочных файлов
#include <macros.h> // обработчик внешнего прерывания #pragma
interrupt_handler int0_isr:2 void int0_isr(void)
{
 char c; // объявление переменной c=PINC; // считывание данных
с порта PORTA=~c;
// вывод инверсного значения в порт A
}
// функция инициализации контроллера void init_cpu()
{
DDRA = 0xFF;// настройка порта A на выход
PORTA = 0xFF;      // установка 5В на выводах порта
PORTC = 0x00;
DDRC = 0x00; // настройка порта C на вход
MCUCR = 0x03; // настройка внешнего прерывания
GIMSK = 0x40;      // разрешение внешнего прерывания
SEI();             // разрешение обработки прерываний
}
void main()
{
init_cpu();       // вызов функции инициализации контроллера
while(1); // зацикливание контроллера
}
```

4. Выбираем микроконтроллер, для которого написана программа. Для этого в меню Project необходимо выбрать команду Options. Настройки проекта должны соответствовать приведенным на рис.2.23.

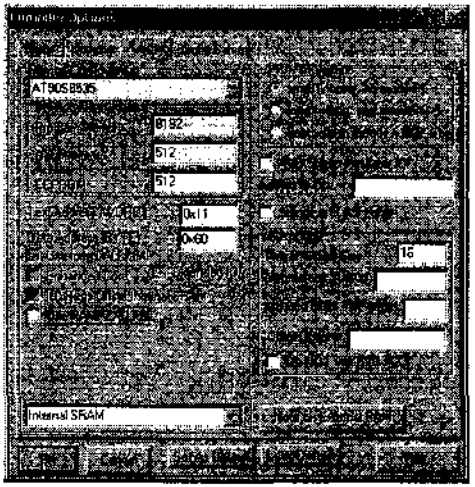


Рис.2.23. Окно настроек проекта

5. Компилируем проект. Для этого из меню Project выбрать команду Rebuild All или нажать сочетание клавиш Shift+F9. При отсутствии ошибок в каталоге, где расположен проект, создастся файл прошивки микроконтроллера LPTTest.hex.

2.5. Аппаратно-программные средства снятия вольт-амперных характеристик полупроводниковых приборов

В практике работы с полупроводниковыми приборами возникает необходимость знать их вольт-амперные характеристики, чтобы произвести расчет электронной схемы. Реальные характеристики позволяют проследить влияние температуры на работу прибора, вычислить коэффициенты усиления, напряжения и токи пробоя, и многое другое.

Лабораторный стенд по снятию вольтамперных характеристик полупроводниковых приборов состоит из следующих блоков:

- блок формирования биполярного напряжения $U_{кэ}$, $U_{си}$;
- блок формирования биполярного тока $I_б$, и биполярного напряжения $U_э$;
- блок регуляторов;
- блок управления;
- блок питания.

Структурная схема стенда показана на (рис.2.21). Принцип работы состоит в следующем: с параллельного порта компьютера (LPT1) на стенд подаются управляющие сигналы и данные, а также со стенда подается выходной сигнал на порт компьютера.

Параллельный (LPT1) порт компьютера имеет в адресном пространстве базовый адрес BASE = 0x378. По этому адресу располагается восьмиразрядная шина ввода/вывода данных. С помощью этой шины восьмиразрядный код подается с компьютера на шину данных лабораторного стенда. По адресу BASE + 1 = 0x379 располагается регистр состояния, представляющий собой 5 – битный порт ввода. На этот порт подается выходной сигнал лабораторного стенда. По адресу BASE + 2 = 0x37A располагается регистр управления, представляющий собой 4 – битный порт вывода. С этого порта на блок управления стенда подаются управляющие сигналы.

Восьмиразрядный сигнал с компьютера подается на блок управления, в котором это сигнал распределяется по трем буферным регистрам с помощью сигналов управления, выходящих с порта 0x37A. Каждый буферный регистр служит для сохранения данных, полученных от компьютера. Верхний регистр (см. рис.2.24) служит для сохранения данных о напряжении $U_{кэ}$ и $U_{си}$. Средний регистр сохраняет выходные данные. Нижний регистр сохраняет данные о напряжении $U_э$ или о токе $I_б$. Каждый регистр имеет синхровход, при подаче высокого

сигнала на который данные со входа устанавливаются на выходе. Регистры управляются с помощью дешифратора, выходы которого подключены к синхровходам регистров.

С верхнего регистра информация поступает на вход цифроаналогового преобразователя, который преобразует цифровой код по линейному закону в пропорциональное этому коду биполярное аналоговое напряжение. Это напряжение подается на вход источника биполярного напряжения управляемого напряжением. На выходе этого источника формируется биполярное напряжение, изменяющееся по линейному закону, с пределом – 40 до + 40 В, которое и является напряжением $U_{кэ}$ и $U_{си}$. Таким образом, изменяя входной цифровой код от 0 до 255, на выходе напряжение изменяется от – 40 В до + 40 В.

С нижнего регистра информация поступает на вход цифроаналогового преобразователя, который преобразует цифровой код по линейному закону в пропорциональное этому коду биполярное аналоговое напряжение. Это напряжение подается на вход источника биполярного тока $I_б$ управляемого напряжением и источника биполярного напряжения $U_э$ управляемого напряжением. Выбор источника тока или источника напряжения производится с помощью переключателя. На выходе источника напряжения формируется биполярное напряжение с пределом от – 10 до + 10 В, которое линейно изменяется в этих пределах при переборе цифрового кода от 0 до 255 приходящего от компьютера. На выходе источника тока формируется биполярный ток с пределами: от – 100 μА до + 100 μА; от – 1 mA до + 1 mA; от – 10 mA до + 10 mA; от – 100 mA до + 100 mA. Выбор режима работы источника тока осуществляется с помощью переключателя.

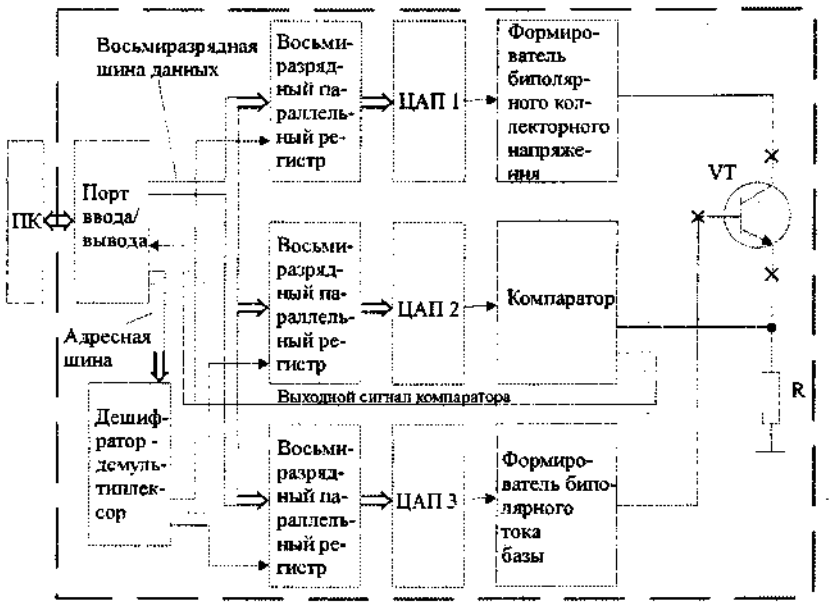


Рис.2.24. Структурная схема стенда

Со среднего регистра информация поступает на вход цифроаналогового преобразователя, который преобразует цифровой код по линейному закону в пропорциональное этому коду биполярное аналоговое напряжение. Это напряжение подается на вход аналогового компаратора. Работа компаратора описана ниже.

2.5.1. Снятие вольт-амперных характеристик биполярных транзисторов.

Исследуемый транзистор подключается к лабораторному стенду, как показано на (рис.2.24). Коллектор подключается к источнику напряжения $U_{кэ}$. База транзистора подключается к источнику тока $I_б$.

Эмиттер транзистора подключается к аналоговому компаратору и резистору R, второй вывод которого подключен к общему проводу. При установке значения тока базы $I_б$ и значения напряжения $U_{кэ}$, через транзистор начиняет течь ток. Так как цепь коллектор – эмиттер транзистора и сопротивление R включены последовательно, то через них течет одинаковый ток (входное сопротивление аналогового компаратора очень большое, поэтому оно не влияет на значение протекающего в цепи транзистор – резистор тока). Этот ток создает падение напряжения на резисторе R. Если известно сопротивление резистора, то по закону Ома можно вычислить ток, протекающий через этот резистор, а следовательно и через исследуемый транзистор. То есть, значение тока находится по формуле:

$$I_k = \frac{U_R}{R}, \text{ А.} \qquad (2.1)$$

Остается только измерить значение напряжения U_R. Это измерение производится с помощью аналогового компаратора. На один вход компаратора подается падение напряжения с резистора R, на другой – биполярное напряжение с цифроаналогового преобразователя. С помощью программы на компьютере цифровой код на входе цифроаналогового преобразователя перебирается от 0 до 255, пока выход компаратора (он является выходным сигналом) не изменит своего состояния на противоположное (с лог. «0» на лог. «1»). Изменение состояния выхода компаратора фиксируется программой на компьютере, и перебор чисел останавливается, например, на числе 123. При известном максимальном напряжении на выходе цифроаналогового преобразователя можно вычислить величину напряжения, соответствующего изменению перебираемого числа на единицу. Это напряжение вычисляется по формуле:

$$\Delta U = \frac{U_{max}}{256}, \text{ В} \qquad (2.2)$$

Если максимальное напряжение U_{max} = 3 В, то получается значение напряжения ΔU = 0,0234 В (напряжение на выходе цифроаналогового преобразователя изменяется в пределах от – 3 В до + 3 В).

Падение напряжения на резисторе R вычисляется по формуле:

$$U_R = \Delta U \cdot N, \text{ В}. \qquad (2.3)$$

где, N – перебираемое программой число. При N = 123 значение напряжения $U_R = 123 \times 0{,}0234 = 2{,}8782$ В.

Если сопротивление резистора R = 10 Ом, то по формуле (2.1) вычисляем значение тока:

$$I_k = \frac{2.8782}{10} = 0.28782, \text{ А}.$$

Таким образом, получается коллекторный ток I_k = 287,82 mA.

Биполярность всех напряжений и токов указывает на различие проводимости исследуемых транзисторов, таких как n – p – n и p – n – p проводимости.

Для биполярных транзисторов n – p – n структуры, напряжение $U_{кэ}$ и ток $I_б$ должны быть положительными относительно эмиттера, а для транзисторов p – n – p должны быть отрицательными относительно эмиттера. В лабораторном стенде это реализуется с помощью цифрового кода, например, для положительных напряжений и токов используется цифровой код от 0 до 127, а для отрицательных напряжений и токов – от 128 до 255. Различие структуры биполярных транзисторов заложено в программу и она сама подает необходимые управляющие сигналы.

Ток $I_б$ источника тока находится по формуле:

$$I_б = \frac{I_{б.max}}{M} \cdot N, \text{ А}. \qquad (2.4)$$

где: $I_{б.макс}$ – максимальный ток источника тока (от 0 до $I_{б.макс}$); M – число, соответствующее цифровому коду, которое характеризует ток одной полярности (например, в данном случае ток положительной полярности использует цифровой код от 0 до 127, то $M = 128$); N – число, перебираемое программой, цифровому коду которого соответствует ток на выходе источника тока.

Например, при $I_{б.макс} = 1$ mA и $N = 56$ получаем:

$$I_б = \frac{1}{128} \cdot 56 = 0.4375 \text{ , mA.}$$

Таким образом, на выходе источника тока устанавливается ток $I_б = 0{,}4375$ mA.

Напряжение $U_{кэ}$ источника напряжения находится аналогично как и ток $I_б$, только вместо I_{max} подставляется $U_{кэ.макс}$. Напряжение $U_{кэ}$ можно вычисляется по формуле:

$$U_{кэ} = \frac{U_{кэ.max}}{M} \cdot N \text{ , В.} \qquad (2.5)$$

где $U_{кэ.макс}$ – максимальное напряжение источника напряжения $U_{кэ}$ (от 0 до $U_{кэ.макс}$); M – аналогично, как и для источника тока (M = 128); N – число, перебираемое программой, цифровому коду которого соответствует напряжение на выходе источника напряжения $U_{кэ}$.

При $U_{кэ.макс} = 40$ В и $N = 74$ получаем:

$$U_{кэ} = \frac{40}{128} \cdot 74 = 23.125 \text{ , В.}$$

Таким образом, на выходе источника напряжения устанавливается напряжение $U_{кэ} = 23{,}125$ В.

Все выше приведенные математические формулы вычисляются с помощью программы на компьютере, а полученные значения напряжений и токов выводится на экран монитора в виде точки вольтамперной характеристики. Семейство точек вольтамперной характери-

стики снимается аналогично при разных значениях напряжения $U_{кэ}$. Семейство вольтамперных характеристик снимается как одна характеристика, но при различных значениях тока базы $I_б$.

2.5.2. Снятие вольтамперных характеристик полевых транзисторов

Исследуемый полевой транзистор подключается стоком к источнику напряжения $U_{си}$, истоком к резистору R, затвором к источнику напряжения $U_з$. Снятие вольтамперной характеристики полевого транзистора производится аналогично, как и для биполярного транзистора. Различие состоит только в том, что вместо тока базы $I_б$ для биполярных транзисторов, подается напряжение затвора $U_з$. Точка вольтамперной характеристики снимается при постоянных напряжениях $U_{си}$ и $U_з$. Одна характеристика снимается при постоянном напряжении $U_з$ и при разных напряжениях $U_{си}$. Семейство вольтамперных характеристик снимается также, как и одна характеристика, но при различных напряжениях $U_з$.

Также как и биполярные транзисторы, полевые транзисторы тоже бывают различной разновидности, а именно: p – n переход, n – канал; p – n переход, p – канал; n – канальный с изолированным затвором обедненного типа; p – канальный с изолированным затвором обедненного типа; n – канальный с изолированным затвором обогащенного типа; p - канальный с изолированным затвором обогащенного типа. Все эти транзисторы требуют свои напряжения питания. Это реализуется, как и для биполярных транзисторов, программно, с помощью цифрового кода. Например, для положительного напряжения используется цифровой код от 0 до 127, а для отрицательного напряжения – от 128 до 255.

Напряжение $U_{си}$ находится аналогично напряжению $U_{кэ}$, так как $U_{си} = U_{кэ}$, а напряжение $U_ч$ находится по формуле (2.5), только вместо значения напряжения $U_{кэ.макс}$ в формулу подставляется значения напряжения $U_{э.макс} = 10$ В.

2.5.3. Снятие вольт-амперных характеристик диодов

Для диода снимается прямая характеристика, поэтому он подключается анодом к источнику тока $I_б$ и аналоговому компаратору, а катодом к общему проводу. Подключение выводов диода производится переключателем. Схема подключения показана на (рис.2.25).

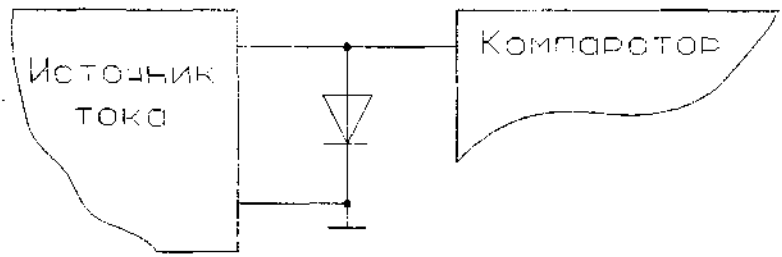


Рис.2.25. Схема подключения исследуемого диода

Снятие одной точки характеристики производится следующим образом: задается положительное значение тока $I_б$. Этот ток, протекая через исследуемый диод, создает на нем падение напряжения. Это падение напряжения измеряется с помощью компаратора, аналогично, как и падение напряжения на резисторе R для транзисторов, только теперь вместо резистора включен диод. Ток источника тока находится по формуле (2.4), а падение напряжение на диоде находится по формуле (2.3). Полученные значения напряжения и тока являются точкой

на вольтамперной характеристике диода. Вся характеристика снимается при различных значениях тока $I_б$. Так как падение напряжения на диоде (в нормальных условиях) не превышает 3В, максимального напряжения компаратора достаточно для снятия вольтамперной характеристики.

2.5.4. Снятие вольт-амперных характеристик стабилитронов

Для стабилитрона снимается обратная характеристика, поэтому он подключается катодом к источнику тока $I_б$ и резистивному делителю напряжения, выход которого подключается к аналоговому компаратору, а анодом к общему проводу. Необходимость в резистивном делителе напряжения возникает потому, что максимальное напряжение на входе аналогового компаратора 3В, а напряжение стабилизации стабилитронов больше трех вольт и компаратор не сможет его измерить. Подключение выводов стабилитрона производится переключателем. Схема подключения показана на (рис.2.26).

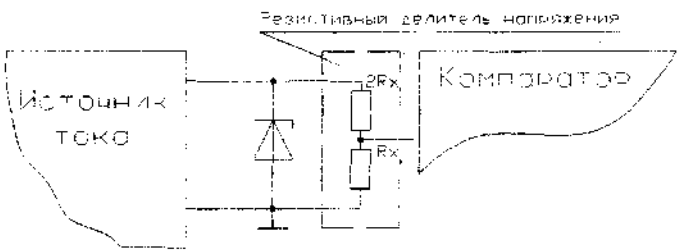


Рис.2.26. Схема подключения исследуемого стабилитрона

Снятие вольтамперной характеристики стабилитрона производится аналогично, как и для диода. Задается положительное значение тока $I_б$. Этот ток, протекая через исследуемый стабилитрон, создает на нем падение напряжения. Это падение напряжения прикладывается к резистивному делителю, который в данном случае делит его на три. Преобразованное напряжение измеряется с помощью компаратора, аналогично, как и падение напряжения на резисторе R для транзисторов, только с учетом коэффициента деления. Ток источника тока находится по формуле (2.4). Падение напряжение на стабилитроне находится по преобразованной формуле (2.3):

$$U_R = \Delta U \cdot N \cdot K, \text{ В}. \qquad (2.6)$$

где, K – коэффициент деления резистивного делителя (К = 3).

Полученные значения напряжения и тока являются точкой на вольтамперной характеристике стабилитрона. Вся характеристика снимается при различных значениях тока $I_б$.

2.5.5. Блок управления

Блок управления представляет собой трехканальный цифро-аналоговый преобразователь. Принципиальная схема блока управления изображена на (рис.2.27). Он включает в себя три восьмиразрядных параллельных регистра DD2 – DD4 (КР1533ИР23), с выхода которых информация подается на цифро-аналоговые преобразователи DD5 – DD7 (К572ПА1) соответственно. Цифро-аналоговый преобразователь DD5 формирует биполярное напряжение управления источником напряжения $U_{кэ}$. Цифро-аналоговый преобразователь DD7 формирует биполярное напряжение управления источником тока $I_б$ и источником напряжения $U_э$. Цифро-аналоговый преобразователь DD6 формирует

биполярное напряжение сравнения на компараторе DA5 (КР1401УД2А). Биполярное напряжение на выходах цифро-аналоговых преобразователей формируется с помощь операционных усилителей DA1 – DA4, DA6, DA7 (КР1401УД2А). Такая необходимость возникает из-за того, что цифро-аналоговые преобразователи имеют токовый выход. Подключение операционных усилителей производится по типовой схеме включения. Диоды VD33 – VD44 служат для защиты выходов цифро-аналоговых преобразователей от прикладываемых к ним напряжений. Работой регистров DD2 – DD4 управляет дешифратор DD1 (КР1533ИД4), который непосредственно подключен к порту компьютера. К порту компьютера также подключены все входы информационных регистров DD2 – DD4. Элементы DD8 – DD11, R1 – R37, R48, VD1 – VD37, VD47 служат для индикации цифровых сигналов, приходящих от компьютера, и подаваемых на него. На элементах R47, VD45, VD46 собран преобразователь уровней напряжения с ±15 В до 0 – 5 В. Это необходимо для согласования уровней напряжения между лабораторным стендом и портом компьютера. Резисторами R39, R40, R42, R43, R45, R46 производится настройка напряжений блока управления. Резисторами R40, R43, R46 производится настройка опорных напряжений цифро-аналоговых преобразователей. Резисторами R39, R42, R45 производится настройка нулевого напряжения на выходе операционных усилителей.

Для управления блоком формирования напряжения $U_{кэ}$, $U_{си}$ на выходе операционного усилителя DA2 задается напряжение с пределом изменения от –2 В до +2 В с помощью опорного резистора R40. Это напряжение снимается с выхода X1 блока управления. Для управления блоком формирования биполярного тока $I_б$, и биполярного напряжения $U_з$ используется напряжение с пределом изменения от –1 В

до +1 В, которое формируется на выходе операционного усилителя DA7 с помощью резистора R46. Это напряжение снимается с выхода X3 блока управления. Напряжение, которое подается на вход аналогового компаратора, снимается с выхода операционного усилителя DA4. Оно изменяется в пределах от –3 В до +3 В и задается с помощью резистора R43. Второй вход аналогового компаратора подключается к выходу X2 блока управления. Выбор операционного усилителя серии КР1402УД2А заключается в том, что данная микросхема является прецизионным усилителем с маленьким напряжением смещения (порядка 5 мВ). Это позволяет ее использовать вместе с цифро-аналоговым преобразователем. Микросхема содержит четыре операционных усилителя не требующих цепи частотной коррекции.

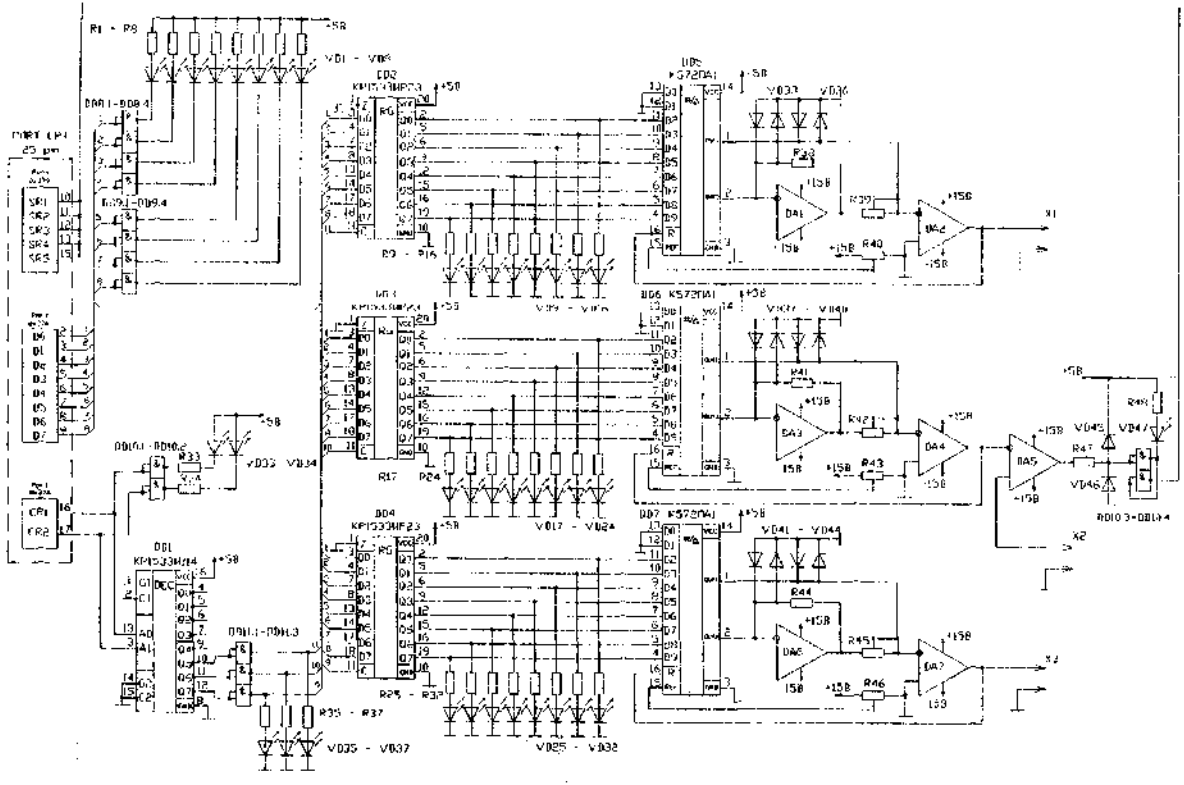


Рис.2.27. Принципиальная схема блока управления

В качестве элементов DD8 – DD11 применяются микросхемы серии К155ЛА12, которые содержат четыре элемента 2И – НЕ. Они используются для усиления тока светодиодов и исключения просадки напряжения сигналов лабораторного стенда. Расчет сопротивлений R1 – R37, R48 производится по формуле:

$$R = \frac{U_п - U_д}{I_д} \qquad (2.7)$$

где $U_п$ – напряжение питания ($U_п$ = 5 В); $U_д$ – падение напряжения на светодиоде ($U_д$ = 2 В); $I_д$ – номинальный ток светодиода ($I_д$ = 10 мА). При таких значениях напряжения и тока, сопротивление резисторов равно 300 Ом.

Дешифратор КР1533ИД4 имеет инверсные выходы, а синхровходы регистров КР1533ИР23 имеют прямые входы, поэтому, микросхема DD11 кроме усиления тока светодиодов, инвертирует состояние синхросигналов.

Микросхема цифро-аналогового преобразователя (К572ПА1) имеет десять информационных входов, а лабораторный стенд имеет восьмиразрядную шину данных, поэтому два младших разряда цифро-аналогового преобразователя присоединяются к общей шине.

Восьмиразрядный цифровой код с порта компьютера подается на входы регистров DD2 – DD4. По сигналу с порта управления дешифратор DD1 подает синхросигнал на один из регистров DD2 – DD4 и информация с входа регистра (например DD1) устанавливается на его выходе. Цифровой код, поданный с компьютера и установившийся на выходе регистра, отображается с помощью светодиодов. Далее, этот цифровой код подается на вход цифро-аналогового преобразователя и на его выходе устанавливается аналоговое напряжение, пропорциональное данному цифровому коду. Полученное аналоговое напряже-

ние подается на соответствующий блок лабораторного стенда. Аналогично работают остальные цифро-аналоговые преобразователи блока управления.

Блок формирования биполярного напряжения $U_{кэ}$, $U_{си}$ представляет собой биполярный регулятор напряжения управляемый биполярным напряжением. Принципиальная схема блока показана на рис.2.28. Основным элементом данного блока является операционный усилитель DA1, включенный по схеме пропорционального инвертирующего усилителя напряжения. Коэффициент усиления усилителя задается с помощь отношения сопротивлений резисторов R5 и R1. В общем случае коэффициент усиления определяется по формуле:

$$K_{ус} = -\frac{R5}{R1} \qquad (2.8).$$

Так как, входное напряжение, подаваемое с блока управления, находится в пределах ±2 В, а на выходе блока формирования должно быть напряжение ±40 В, то коэффициент усиления должен быть равным 20. При значении сопротивления R1 = 10 кОм, сопротивление R5 должно быть равно 200 кОм. Резистор R2 подключается ко входу операционного усилителя для компенсации ошибки, обусловленной током смещения операционного усилителя. Значение этого сопротивления вычисляется по формуле:

$$R2 = \frac{R1 \cdot R5}{R1 + R5} \qquad (2.9).$$

При данных значениях сопротивлений R1 и R5, сопротивление R2 будет равно 9.5 кОм.

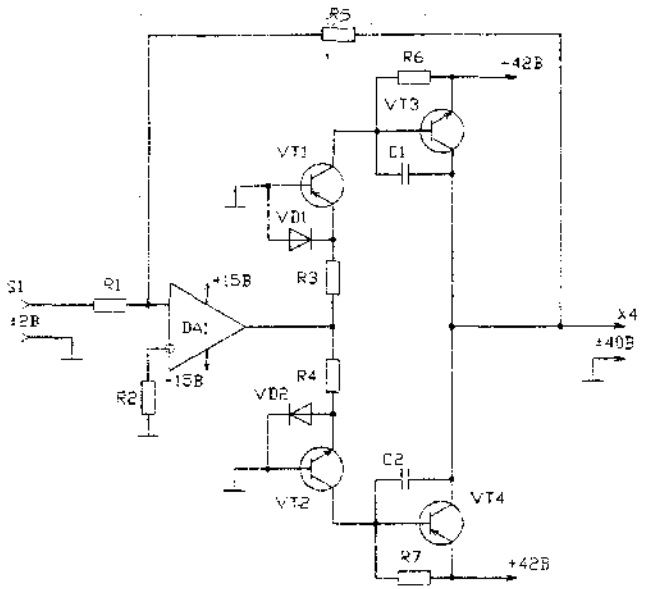


Рис. 2.28. Принципиальная схема блока формирования биполярного напряжения $U_{кз}$, $U_{си}$

Элементы VT1 – VT4, VD1, VD2, C1, C2, R3, R4, R6, R7 представляют собой силовую схему регулятора. Биполярное напряжение с выхода операционного усилителя DA1 через резисторы R3 и R4 прикладывается к диодам VD1, VD2 и транзисторам VT1, VT2. Транзисторы VT1, VT2 выполняют функцию ключей, которые позволяют работать только одному выходному транзистору и исключают возможность открывания транзисторов VT3, VT4 одновременно. В зависимости от полярности напряжения открывается либо диод VD1, либо диод VD2. Резисторы R3 и R4 служат для ограничения тока, протекающего через диоды, и выбираются номиналом 10 кОм. При отрицательном напряжении открывается диод VD1, а диод VD2 закрывается. При протекании через диод VD1 тока, на нем создается падение напряже-

ния порядка 0.6...1 В, которое открывает транзистор VT1. Открытый транзистор VT1 разрешает работу выходного мощного транзистора VT3. При положительном напряжении открывается диод VD2, а диод VD1 закрывается. Падение напряжения на диоде VD2 открывает транзистор VT2. Открытый транзистор VT2 разрешает работу выходного мощного транзистора VT4. Резисторы R6, R7 запирают выходные мощные транзисторы VT3, VT4 при отсутствии управляющего напряжения, подаваемого с коллекторов транзисторов VT1, VT2 соответственно. Номиналы этих сопротивлений выбираются 1 кОм, из расчета тока базы мощных транзисторов VT3, VT4. Конденсаторы C1 и C2 служат для подавления переменной составляющей напряжения. Емкость выбирается порядка 100 пФ.

В схеме применяются диоды VD1 и VD2 типа КД503, что обусловлено скоростью их работы. В качестве транзисторов VT1 и VT2 применяются транзисторы средней мощности, типа: VT1 – КТ814, VT2 – КТ815. В качестве транзисторов VT3 и VT4 применяются транзисторы большой мощности, типа: VT3 – КТ819, VT4 – КТ818. Транзисторы VT3 и VT4 обязательно должны стоять на радиаторах, площадью порядка 100 $см^2$ каждый. Данная схема источника напряжения управляемого напряжением рассчитана на выходное напряжение ± 40 В и максимальный выходной ток ± 3 А.

Принципиальная схема блока формирования биполярного тока $I_б$, и биполярного напряжения $U_э$ изображена на рис.2.29. Блок формирования биполярного тока $I_б$, и биполярного напряжения $U_э$ состоит из двух раздельных блоков, имеющих общий вход.

Первый блок состоит из элементов R1 – R3, DA4. Он представляет собой регулируемый источник напряжения управляемый напряжением. Основным элементом блока является операционный

усилитель DA4, включенный по схеме пропорционального инвертирующего усилителя. Резисторами R1, R3 задается коэффициент усиления регулятора. При значении коэффициента усиления равно 10, значения сопротивлений будут: R1 = 1 кОм; R2 = 10 кОм. Значение сопротивления R3, которое учитывает напряжение смещения операционного усилителя, вычисляется по формуле (2.9), и его сопротивление будет равным 910 Ом. Таким образом, данный блок преобразует входное напряжение ± 1 В в напряжение ± 10 В, которое является напряжением $U_з$.

Второй блок представляет собой источник тока управляемый напряжением. Он преобразует входное управляющее напряжение ± 1 В в пропорциональный этому напряжению ток с пределами изменения: ± 100 мкА; ± 1 мА; ± 10 мА; ± 100 мА. На элементах DA1, VT1, VT2, R4 собран источник тока управляемый напряжением, выходной ток которого устанавливается на коллекторах транзисторов VT1, VT2 пропорционально входному напряжению, прикладывается к резисторам R5, R6 соответственно. За счет транзисторов VT1, VT2 выходной ток распределяется по полярности. С коллектора транзистора VT1 снимается ток положительной полярности, а с коллектора транзистора VT2 снимается ток отрицательной полярности. Значение выходного тока задается опорным резистором R4. значение сопротивления этого резистора выбирается равным 1 кОм. При таком значении сопротивления R4, значение выходного тока вычисляется по формуле:

$$I_T = \frac{U_{BX}}{R4}; \qquad (2.10)$$

где $U_{вх}$ – входное напряжение ($U_{вх}$ = ± 1 В).

Эта схема в общем случае является преобразователем входного напряжения относительно общего провода в однополярное напряже-

ние относительно провода питания. При значениях сопротивлений R5, R6 равных 1 кОм и входном напряжении ± 1 В, на резисторе R5 создается падение напряжения с пределом изменения 0..-1 В, относительно провода питания + 15 В, а на резисторе R6 создается падение напряжения с пределом изменения 0...+ 1 В, относительно провода питания – 15 В. Транзисторы VT1, VT2 также являются ключами для последующей схемы. При открытом транзисторе VT1, транзистор VT2 запирается, и наоборот, при открытом транзисторе VT2, запирается транзистор VT1.

На элементах DA2, VT3, VT4, R7 – R10, SA1.1 собран источник тока положительной полярности, управляемый напряжением относительно провода питания +15 В. На элементах DA3, VT5, VT6, R11 – R14, SA1.2 собран источник тока отрицательной полярности, управляемый напряжением относительно провода питания - 15 В.

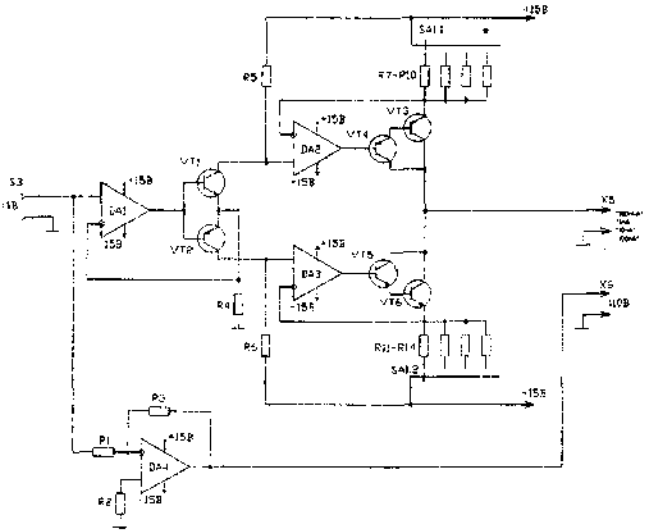


Рис. 2.29. Принципиальная схема блока формирования биполярного тока $I_б$ и биполярного напряжения $U_э$

Входное напряжение, снимаемое с резистора R5, подается на вход операционного усилителя DA2. Операционный усилитель включен по схеме источника тока. Выходной ток источника вычисляется по формуле:

$$I_{ВЫХ} = \frac{U_П - U_R}{R_X}; \qquad (2.11)$$

где $U_п$ – напряжение питания ($U_п$ = + 15 В); U_R – падение напряжения на резисторе R5, снимаемое относительно общего провода; R_x – опорное сопротивление в эмиттерной цепи выходного транзистора VT3. Для получения тока с пределами 100 мкА, 1 мА, 10 мА, 100 мА, по формуле (10) получаются такие значения сопротивлений: 10 кОм, 1 кОм, 100 Ом, 10 Ом соответственно. Соответствующий резистор выбирается с помощью переключателя SA1.1. Включение транзисторов VT3, VT4 произведено по схеме Дарлингтона, для повышения коэффициента усиления. Работа источника отрицательного тока аналогична работе источника положительного тока. В качестве транзисторов VT3, VT4 применяются транзисторы типа: VT3 – КТ814, VT4 – КТ818. В качестве транзисторов VT5, VT6 применяются транзисторы типа: VT5 – КТ815, VT6 – КТ819.

Значение выходного тока всего блока рассчитывается по формуле:

$$I_{ВЫХ} = \left(U_П - \frac{U_{ВХ} \cdot R5}{R4}\right) \cdot \frac{1}{R_X}; \qquad (2.12)$$

где $U_{вх}$ – входное напряжение ($U_{вх}$ = ± 1 В); $U_п$ – напряжение питания ($U_п$ = ± 15 В); R_x – сопротивление, соответствующее пределу изменения тока.

Принципиальная схема блока регуляторов изображена на рис.2.27. Блок содержит три переключателя SA1 – SA3. Переключа-

тель SA1 служит для задания предела изменения тока I_6. Он входит в состав блока формирования биполярного тока I_6, и биполярного напряжения $U_э$. Переключатель SA1 имеет четыре положения, которыми задается пределы изменения тока, а именно: «100 мкА», «1 мА», «10 мА», «100 мА». Переключатель SA1 физически разделен на два переключателя SA1.1 и SA1.2. Переключатель SA1.1 задает предел положительного тока, а переключатель SA1.2 задает предел отрицательного тока.

Переключателем SA2 к выходу X9 лабораторного стенда подключается либо источник тока I_6, либо источник напряжения $U_э$. При подключенном источнике напряжения исследуется работа полевых транзисторов, а при подключенном источнике тока исследуется работа биполярных транзисторов, диодов и стабилитронов.

Переключателем SA3 задается работа лабораторного стенда, а именно: включение в эмиттерную цепь различных опорных сопротивлений при измерении параметров транзисторов, режим измерения прямой характеристики диодов и режим измерения обратной характеристики стабилитронов. К переключателю подключены сопротивления R номиналом 100 Ом, 10 Ом, 1 Ом. Каждое сопротивление рассчитано на определенный ток. Так как компаратор может замерить только напряжение до трех вольт, то максимальный измеряемый ток вычисляется по формуле:

$$I_{изм.макс} = \frac{U_{изм.макс}}{R}; \qquad (2.13)$$

где $U_{изм.макс}$ – максимальное измеряемое напряжение ($U_{изм.макс}$ = 3 В); R – сопротивление опорного резистора. Получается, что при сопротивлении R равном 100 Ом, максимальный измеряемый ток будет равен 30 мА, при R = 10 Ом – $I_{изм.макс}$ = 300 мА, а при R = 1 Ом - $I_{изм.макс}$ = 3 А. Таким образом, максимальный измеряемый ток лабораторным

стендом находится в пределах до 3 А. При снятии вольт-амперных характеристик следует учитывать примерный максимальный ток, протекающий через исследуемый транзистор, а измерение сначала необходимо производить с минимального сопротивления R. Вольт-амперные характеристики диодов и стабилитронов снимаются без сопротивления R, то есть подключается непосредственно на общий провод. Все вышеприведенные параметры задаются с помощью переключателей SA3.1 и SA3.2. Переключателями SA3.3, SA3.4 и резисторами R4 и R5 задается коэффициент деления измеренного напряжения при снятии вольтамперных характеристик стабилитронов.

С выхода X1 блока управления напряжение с пределами ± 2 В подается на вход S1 блока формирования напряжения $U_{кэ}(U_{си})$. С выхода X4 блока формирования напряжения $U_{кэ}(U_{си})$ напряжение с пределом ± 40 В подается на выход X7 лабораторного стенда. Выход X2 блока управления (вход аналогового компаратора) через переключатель SA3 подключается к выходу X8 лабораторного стенда. С выхода X3 блока управления напряжение с пределами ± 1 В подается на вход S3 блока формирования биполярного тока $I_б$, и биполярного напряжения $U_э$. Выходы блока формирования биполярного тока $I_б$, и биполярного напряжения $U_э$ через переключатели SA2 и SA3 подключаются к выходу X9 лабораторного стенда.

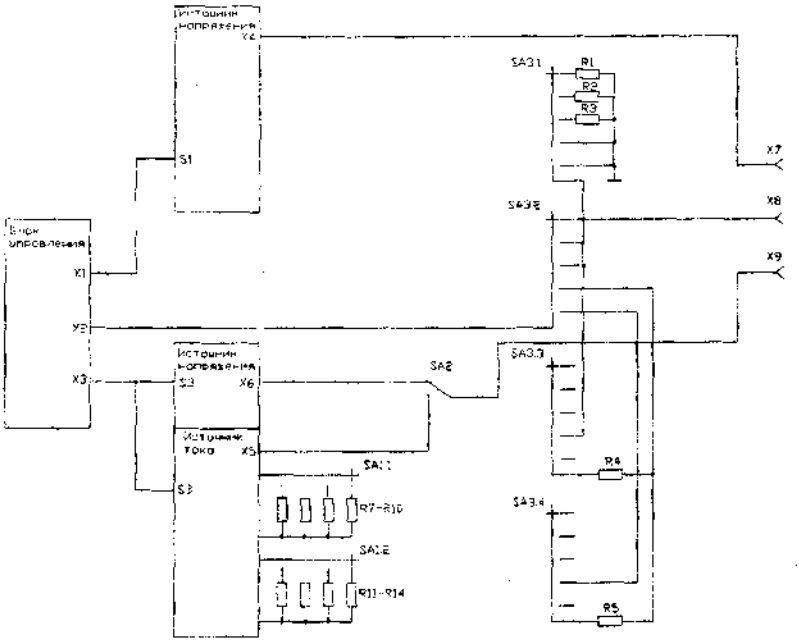


Рис.2.30. Принципиальная схема блока регуляторов

При снятии вольт-амперных характеристик транзисторов, исследуемый транзистор подключается коллектором (стоком) к выходу X7, базой (затвором) к выходу X9, а эмиттером (истоком) к выходу X8 лабораторного стенда.

При снятии вольт-амперных характеристик диодов, исследуемый диод анодом подключается к выходу X9, а катодом к выходу X8 лабораторного стенда. При этом переключатель SA2 обязательно должен находится в положении, соответствующем работе источника тока I_6.

При снятии вольт-амперных характеристик стабилитронов, исследуемый стабилитрон катодом подключается к выходу X9, а анодом к выходу X8 лабораторного стенда. При этом переключатель SA2

также обязательно должен находится в положении, соответствующем работе источника тока I_6.

При снятии вольт-амперных характеристик диодов и стабилитронов выход X7 лабораторного стенда не используется.

Лабораторный стенд использует следующие напряжения: +5 В для питания цифровой части схемы, -15 В, +15 В для питания операционных усилителей, -40 В, +40 В для блока формирования напряжения $U_{кэ}(U_{си})$. При этом эти напряжения должны быть стабильными. Предложенная принципиальная схема блока питания изображена на рис.2.31.

Трансформатор T1 рассчитан на входное напряжение 220 В, которое подается на него через переключатель SA1 и плавкий предохранитель FU1. Трансформатор имеет две обмотки: одна с напряжением 15 В и номинальным током 0.5 А, а вторая с напряжением 140 В и максимальным током 3 А. Вторая выходная обмотка имеет три отвода. При этом отводы сделаны так, что с каждой части обмотки снимается напряжение 35 В. Средний вывод второй выходной обмотки присоединяется к общему проводу.

С первой выходной обмотки напряжение через выпрямительный мост, собранный на диодах VD1 – VD4 (КД105), и сглаживающий конденсатор C1 подается на вход стабилизатора напряжения DA1 (КР142ЕН5А). Включение стабилизатора DA1 производится по типовой схеме включения. С выхода стабилизатора DA1 снимается стабильное напряжение + 5 В.

Напряжения –15 В и +15 В образуются с помощью стабилизаторов DA2 (КР1180ЕН15) и DA3 (КР1179ЕН15), включенных по типовой схеме включения. Дополнительными элементами являются емкости C3 – C6. На микросхемы DA2 и DA3 напряжение подается с вы-

прямительного моста VD5 – VD8 (КД202). При данной нагрузке емкости С1 – С6 выбираются с запасом значением 100 мкФ.

Напряжения ±42 В образуются с помощью параметрического стабилизатора собранного на элементах VD9 – VD14, VT1 – VT6, С7 – С10, R1 – R10. Выходное напряжение выбирается значением ±42 В а не ±40 В потому, что на транзисторах VT3, VT4 (рис.2.28) блока формирования напряжения $U_{кэ}(U_{си})$ создается падение напряжения порядка 2 В. Более точно настройка напряжения осуществляется с помощью подстроечных резисторов R6 и R9. Для объяснения принципа работы параметрического стабилизатора рассмотрим только стабилизатор положительного напряжения, стабилизатор отрицательного напряжения работает аналогично. Основой параметрического стабилизатора является эмиттерный повторитель на транзисторах VT3 и VT5, включенных по схеме Дарлингтона. Схема обеспечивает стабилизацию напряжения на нагрузке, как при изменении сопротивления нагрузки, так и при колебаниях входного напряжения. Принцип работы параметрического стабилизатора заключается в следующем. Допустим, что значение сопротивления нагрузки уменьшилось, тогда ток, протекающий через нагрузку, увеличится. Увеличение тока, в свою очередь, приведет к возрастанию падения напряжения на транзисторе VT3. Тогда напряжение на нагрузке уменьшится. Уменьшение напряжения на нагрузке приведет к уменьшению напряжения на выходе делителя напряжения, собранного на резисторах R5 – R7. В свою очередь изменение напряжения на выходе делителя изменяет потенциал базы транзистора VT1, относительно стабильного напряжения эмиттера.

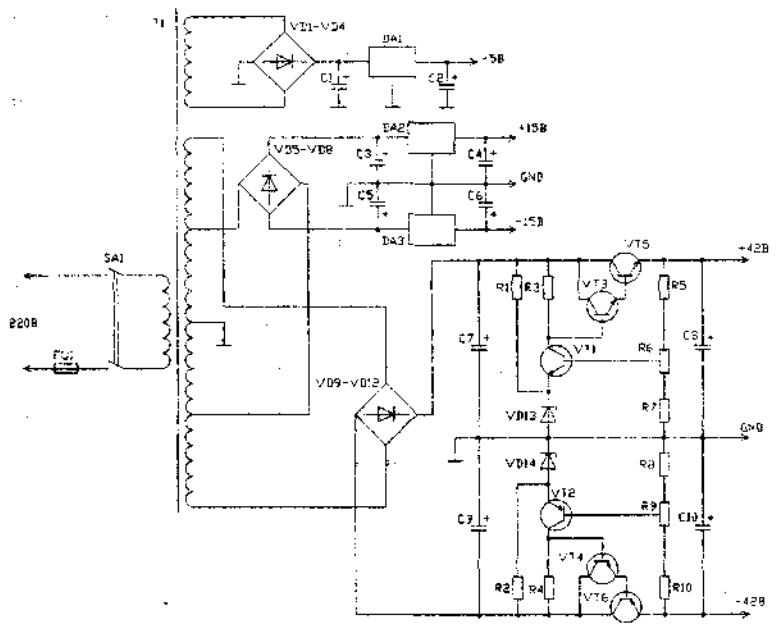


Рис.2.31. Принципиальная схема блока питания

Стабильность напряжения эмиттера транзистора VT1 задается с помощью резистора R1 и стабилитрона VD13. Уменьшится ток базы транзистора VT1, а следовательно уменьшится и его коллекторный ток. Падение напряжения на резисторе R3 уменьшится, а на коллекторе транзистора VT1 возрастет. Возрастание напряжения на коллекторе транзистора VT1 приведет к возрастанию тока базы транзистора VT3 и последующего открывания силового транзистора VT5, что в свою очередь приведет к возрастанию напряжения на нагрузке. В такой же последовательности будут протекать процессы в стабилизаторе при уменьшении или увеличении входного напряжения. При возрастании напряжения питания транзистор VT5 прикроется и падение напряже-

ния на нем возрастет, при уменьшении напряжения питания падение напряжения на транзисторе VT5 уменьшится.

В схеме использованы резисторы R1 – R4 сопротивлением 2 кОм, стабилитроны VD13, VD14 с напряжением стабилизации 33 В, транзисторы VT1, VT3 – КТ315, VT2, VT4 – КТ361, VT5 – КТ819, VT6 – КТ818, конденсаторы С7 – С10 – 1000 мкФ, резисторы R5, R10 – 1 кОм, R7, R8 – 20 кОм, R6, R9 – 10 кОм.

2.5.6. Описание программных средств

Последовательность получения вольт-амперных характеристик полупроводниковых приборов состоит из нескольких этапов. Прежде всего выбирается тип элемента – диод или транзистор (рис.2.32).

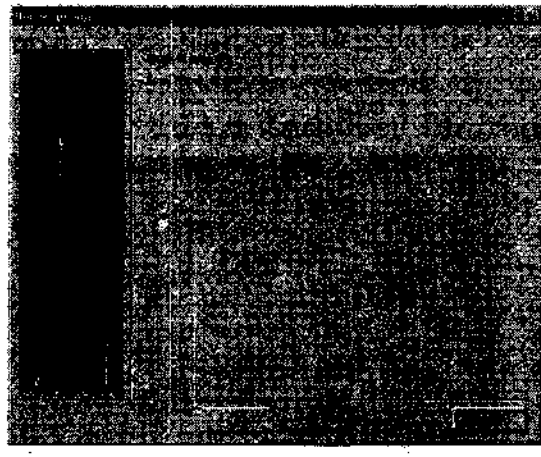
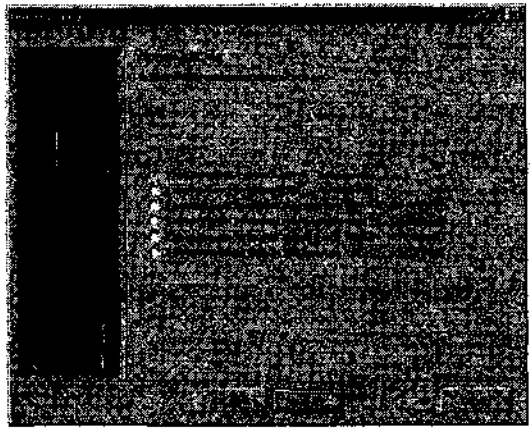


Рис.2.32. Выбор типа элемента

В зависимости от того, какой элемент выбран, далее будет предложено выбрать тип вольт-амперной характеристики (прямая для диода или обратная для стабилитрона), тип транзистора (биполярный или

полевой), а также тип проводимости. На рис.2.33 представлены варианты для полевого транзистора.



2.33. Окно с опциями выбора типа транзистора

Завершающим этапом является указание диапазона изменения токов и напряжений. Для полевого транзистора окно с настройками параметров имеет вид, показанный на рис.2.34.

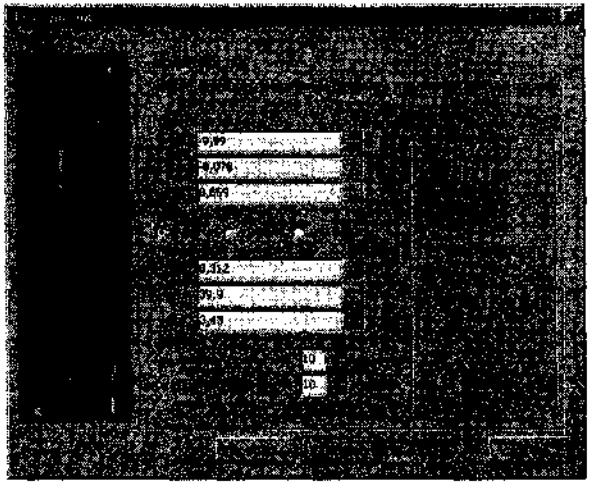


Рис.2.34. Окно задания параметров

После нажатия на кнопку «Start» откроется окно, в котором будут изображены вольт-амперные характеристики исследуемого элемента (рис.2.35).

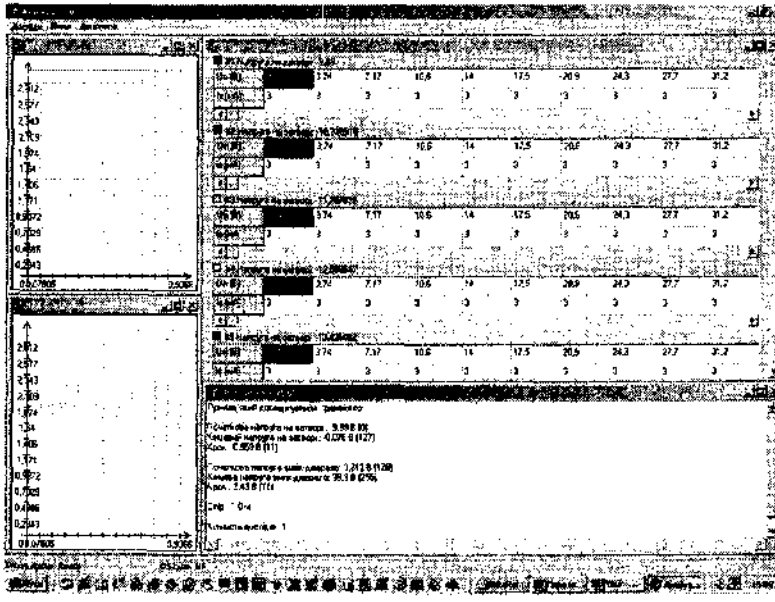


Рис.2.35.Окно с результатами эксперимента

На рис.2.36 и рис.2.37 представлены вольт-амперные характеристики, полученные с помощью стенда.

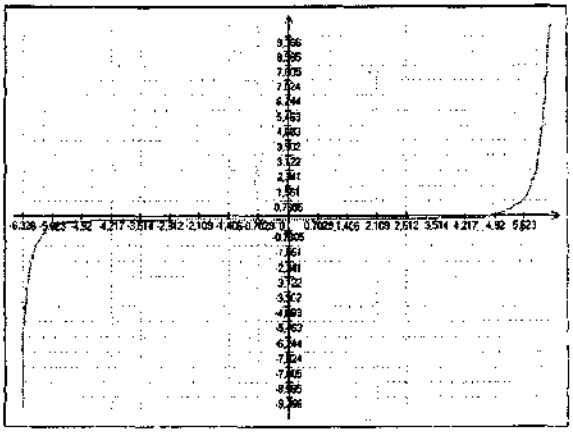


Рис.2.36. ВАХ Стабилитрона КС162А при Iмакс=10мА

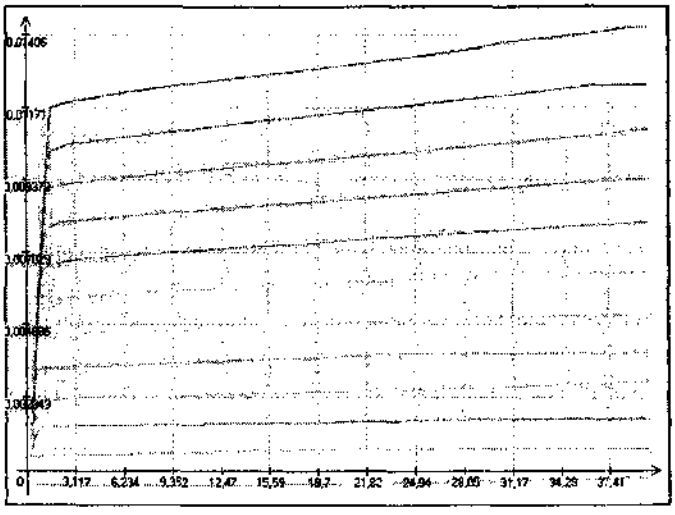


Рис.2.37. ВАХ Транзистора КТ815Г

3. ПОСЛЕДОВАТЕЛЬНЫЙ ПОРТ

3.1. Аппаратная организация порта

Последовательный или коммуникационный порт компьютера является промышленным стандартом для подключения к компьютеру модема, а также может использоваться для многих других целей: подключения ручных манипуляторов типа «мышь», второго компьютера для обмена файлами по нуль–модемному кабелю и, наконец, любых других «нестандартных» электронных устройств, которым необходимо обмениваться данными с персональным компьютером. COM–порт часто используют в качестве канала для обмена информацией в промышленных системах управления и сбора данных – например, для обмена данными с измерительными микроконтроллерами. Большинство ведущих фирм–производителей электронных приборов и устройств серийно выпускают многочисленные измерительные и технологические устройства, подключаемые к компьютеру через последовательный интерфейс.

Последовательный интерфейс для передачи сигнала в одну сторону использует одну сигнальную линию, по которой информационные биты передаются друг за другом последовательно. Такой способ передачи определяет название интерфейса и порта, который его реализует. Эти названия соответствуют английским терминам Serial Interface и Serial Port (иногда в неудачном переводе их называют «серийными», что звучит довольно удивительно). Последовательная передача данных может осуществляться как в асинхронном, так и синхронном режимах.[6]

При асинхронной передаче каждому байту предшествует стартовый бит, который сигнализирует приемнику о начале очередной по-

сылки, за которым следуют биты данных и, возможно, бит паритета (контроля четности). Завершает посылку стоповый бит, который гарантирует определенный интервал между соседними посылками (рис.3.1). Стартовый бит следующего посланного байта может быть отправлен в любой момент после окончания стопового бита, т.е. между передачами возможны паузы произвольной продолжительности.

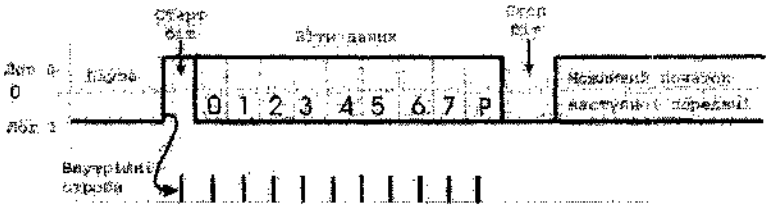


Рис.3.1. Формат асинхронной передачи

Стартовый бит, который имеет всегда строго определенное значение (лог. 0), обеспечивает простой механизм синхронизации приемника по сигналу от передатчика. Предполагается, что приемник и передатчик работают на одной скорости обмена, которая измеряется в количестве передаваемых бит в секунду. Внутренний генератор синхронизации приемника использует счетчик–делитель опорной частоты, обнуляемый в момент приема начала стартового бита. Этот счетчик генерирует внутренние стробы, по которым приемник фиксирует следующие принятые биты. В идеальном случае эти стробы располагаются в середине битовых интервалов, что обеспечивает возможность приема данных и при некотором рассогласовании скоростей приемника и передатчика. Нетрудно заметить, что при передаче 8 бит данных, одного контрольного и одного стопового бита предельно допустимое несогласование скоростей, при котором данные будут верно распозна-

ны, не может превышать 5%. С учетом фазовых искажений (пологих фронтов сигнала) и дискретности работы внутреннего счетчика синхронизации, реально допустимо меньшее отклонение частот. Чем меньше коэффициент деления опорной частоты внутреннего генератора (чем выше скорость передачи), тем большая погрешность привязки стробов к середине битового интервала, и требования к согласованности частот более строгие. Также, чем выше частота передачи, тем большее влияние оказывает искажения фронтов на фазу принятого сигнала. Такое действие этих двух факторов приводит к повышению требований согласованности частот приемника и передатчика с ростом частоты обмена.

Формат асинхронной посылки разрешает обнаруживать возможные ошибки передачи:

– если принят перепад, который сигнализирует о начале посылки, а по стробу стартового бита зафиксирован уровень логической единицы, стартовый бит считается ошибочным и приемник снова переходит в состояние ожидания. Об этой ошибке приемник может и не сообщать.

– если в течении времени, отведенного под стоповый бит(ы), выявлен уровень логической единицы, фиксируется ошибка стопового бита.

– если применяется контроль четности (паритета), то после посылки бит данных, перед стоповым битом передается контрольный бит. Этот бит дополняет количество единичных бит данных к четному или нечетному, в зависимости от принятого соглашения. Прием байта с неверным значением контрольного бита при включенном контроле четности, приводит к фиксации ошибки принятых данных.

Контроль формата разрешает обнаруживать обрыв линии: при этом обычно принимается логический нуль, который сначала трактуется как стартовый бит и нулевые биты данных, но потом сработает контроль стопового бита.

Для асинхронного режима принят ряд стандартных скоростей обмена: 50, 75, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600 и 115200 бит/с. Иногда вместо единицы измерения «бит/с» используют «бод», но в данном случае, при рассмотрении двоичных переданных сигналов, это некорректно. В бодах принято измерять частоту изменения состояния линии, а при недвоичном способе кодирования (широко применяемом в современных модемах) в том же самом канале связи, скорости передачи бит (бит/с) и изменения сигнала (бод) могут отличаться в несколько раз.

Количество бит данных может составлять 5, 6, 7 или 8 (5– и 6–битные форматы мало распространены). Количество стоповых бит может быть 1, 1,5 и 2 (под «полтора бита» подразумевается только продолжительность стопового интервала).

Асинхронный обмен в PC реализуется с помощью COM–порта с использованием протокола RS–232C.

Синхронный режим передачи предполагает постоянную активность канала связи. Посылка начинается с синхробайта, за которым сразу же передается поток информационных бит. Если у передатчика нет данных для передачи, он заполняет паузу непрерывной посылкой байтов синхронизации. Очевидно, что при передаче больших массивов данных, затраты на синхронизацию в данном режиме обмена будут ниже, чем в асинхронном. Однако в синхронном режиме необходима внешняя синхронизация приемника с передатчиком, поскольку даже малое отклонение частот приведет к быстрому накоплению ошибки и

искажению принятых данных. Внешняя синхронизация возможна или с помощью отдельной линии для передачи сигнала синхронизации, или путем использования самосинхронизирующего кодирования данных (например, манчестерский или код NRZ), при котором на приемной стороне из принятого сигнала могут быть выделены импульсы синхронизации. В любом случае синхронный режим требует или дорогих линий связи, или дорогого оборудования (а может, и того и другого). Для PC существуют специальные платы – адаптеры SDLC (довольно дорогие), которые поддерживают синхронный режим обмена. Они используются в основном для связи с большими машинами (mainframes) IBM и в данное время мало распространенные (их вытеснили менее дорогие и более эффективные средства коммуникаций).

Последовательный интерфейс на физическом уровне может иметь разные реализации, которые различаются способами передачи электрических сигналов. Существует ряд родственных международных стандартов: RS–232C, RS–423A, RS–422A и RS–485. На рис.3.2 приводятся схемы соединения приемников и передатчиков и показаны их ограничения на длину линии (L) и максимальную скорость передачи данных (V).

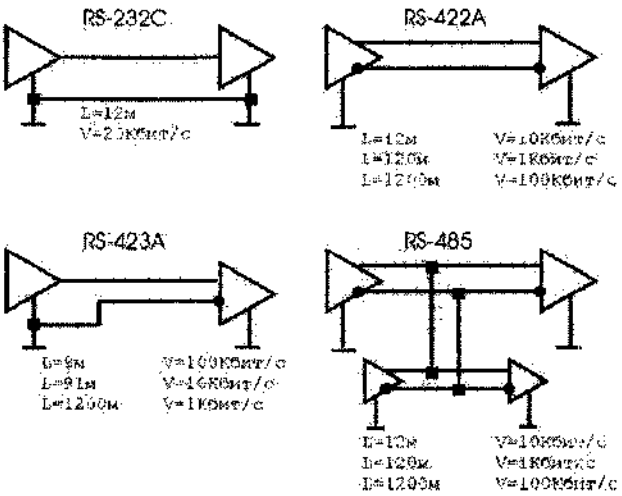


Рис.3.2. Стандарты последовательного интерфейса

Несимметричные линии интерфейсов RS–232C и RS–423A имеют самую низкую защищенность от синфазной помехи, хотя дифференциальный вход приемника RS–423A немного смягчает ситуацию. Лучшие параметры имеет двухточечный интерфейс RS–422A и его магистральный (шинный) аналог RS–485, которые работают на симметричных линиях связи. В них для передачи каждого сигнала используются дифференциальные сигналы с отдельной (витой) парой проводов. Существуют относительно несложные преобразователи сигналов для согласования всех этих интерфейсов.

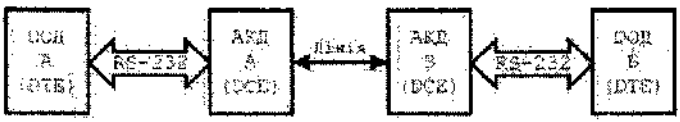


Рис.3.3. Полная схема соединения по RS232

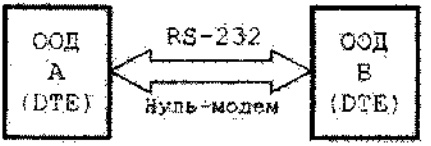


Рис.3.4. Схема соединения по RS232 нуль–модемным кабелем

3.2. Интерфейс RS–232C

Интерфейс RS–232C предназначен для подключения аппаратуры, которая передает или принимает данные (АПД) к аппаратуре каналов данных (АКД). В роли АПД может выступать компьютер, принтер, плоттер и другое периферийное оборудование. Этим устройствам соответствует аббревиатура DTE – Data Terminal Equipment. В роли АКД обычно выступает модем. Этим устройствам соответствует аббревиатура DCE – Data Communication Equipment. Конечной целью подключения являются соединения двух устройств DTE, полная схема соединения приведена на рис.3.3. Интерфейс разрешает исключить канал связи вместе с парой устройств DTE (модемов), путем соединения устройств непосредственно с помощью нуль–модемного кабеля (рис.3.4).

Стандарт описывает управляющие сигналы интерфейса, пересылки данных, электрический интерфейс и типы разъемов. Стандарт описывает асинхронный и синхронный режимы обмена, но COM–порты поддерживают только асинхронный режим. Расположение контактов 9–выводного COM–порта приведено на рис.3.5. Табл.3.1 поясняет назначение его контактов.

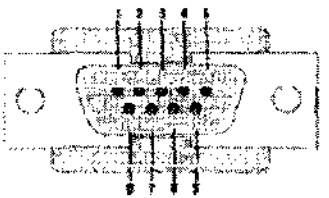


Рис.3.5. Вид разъема COM–порта

Табл.3.1. **Назначение контактов разъема**

Контакт	Сигнал	Направление (относительно компьютера)
1	Детектирование несущей (Data Carrier Detect, DCD)	
2	Приём данных (Received Data, RxD)	
3	Передача данных (Transmitted Data, TxD)	
4	Готовность терминала (Data Terminal Ready, DTR)	
5	Сигнальная «земля» (Signal Ground)	
6	Готовность данных (Data Set Ready, DSR)	
7	Запрос на передачу (Request to Send, RTS)	
8	Готовность к передаче (Clear to Send, CTS)	
9	Индикатор вызова (Ring Indicator, RI)	

3.3. Электрический интерфейс

Стандарт RS–232C использует несимметричные передатчики и приемники – сигнал передается относительно общего провода – схемной земли (симметричные дифференциальные сигналы используются в других интерфейсах, например RS–422). Интерфейс НЕ ОБЕСПЕЧИВАЕТ ГАЛЬВАНИЧЕСКОЙ РАЗВЯЗКИ устройств. Логической единице соответствует уровень напряжения на входе приемника в диапазоне –12...–3 В. Для линий управляющих сигналов это состояние называется ON («включенное»), для линий последовательных данных называется MARK («отмеченное»). Логическому нулю соответствует напряжение в диапазоне +3...+12 В. Для линий управляющих сигналов

это состояние называется OFF («исключенное»), для линий последовательных данных называется SPACE. Между уровнями –3...+3 В имеется зона нечувствительности, которая обеспечивает гистерезис приемника: состояние линии будет считаться измененным только после пересечения соответствующего порога (рис.3.6). Уровни сигналов на выходах передатчиков должны находиться в диапазонах –12...–5В и +5...+12В для представления единицы и нуля соответственно. Разность потенциалов между схемными землями (SG) устройств которые соединяются, должна быть менее 2В. При более высокой разности потенциалов возможен неверный прием сигналов. Интерфейс допускает наличие защитного заземления для соединяемых устройств, если они оба питаются от сети переменного тока и имеют сетевые фильтры.

Подключение и отключение интерфейсных кабелей устройств с автономным питанием должно производиться при отключенном питания. В противном случае разность не выравненных потенциалов устройств в момент коммутации (присоединение или отсоединение разъемов) может оказаться приложенной к выходным или входным цепям интерфейса и вывести микросхему из строя.

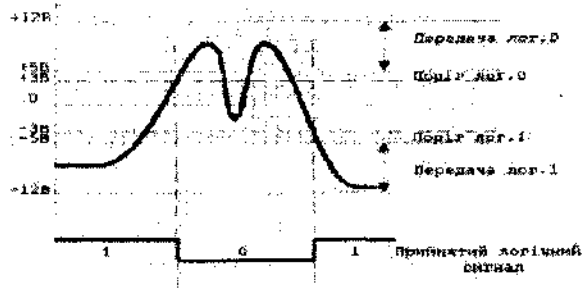


Рис.3.6. Прием сигналов RS-232C

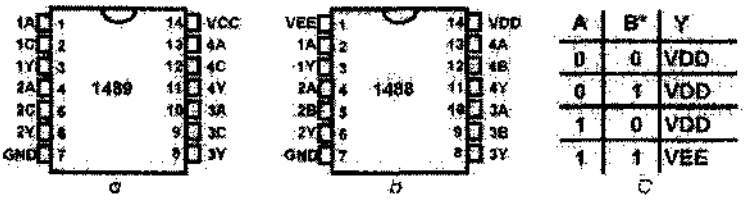


Рис.3.7. Микросхемы формирователей сигналов RS-232C: а - приемник; б – передатчик; с – таблица состояний выходов передатчика

Для интерфейса RS–232C специально выпускаются буферные микросхемы приемников (с гистерезисом) и передатчиков двухполярного сигнала. При несоблюдении правил заземления и коммутации включенных устройств они обычно являются первыми (хорошо, если единственными) жертвами. Иногда их устанавливают в «панельках», что сильно облегчает замену. Расположение выводов популярных микросхем формирователей сигналов RS–232C приводится на рис.3.7. Часто буферные схемы входят прямо в состав интерфейсных БИС. Это делает устройство дешевым, экономит место на плате, но в случае поломки обычно оборачивается большими финансовыми потерями. Выход из строя интерфейсных микросхем замыканием сигнальных цепей маловероятен, поскольку ток короткого замыкания передатчиков обычно ограничен на уровне 20 мА.

Стандарт RS–232C регламентирует типы применяемых разъемов, которые обеспечивает высокий уровень совместимости аппаратуры разных производителей.

На устройствах DTE (в том числе, и на COM–портах PC) принято устанавливать штекеры (male – «папа») DB25–P или более компактный вариант DB9–P.

Девятиштырьковые разъемы не имеют контактов для дополнительных сигналов, необходимых для синхронного режима (в большинстве 25–штырьковых разъемах эти контакты не используются).

На устройствах DCE (модемах) устанавливают розетки (female — «мама») DB25–S или DB–9S. Это правило допускает, что разъемы DCE могут подключаться к разъемам DTE непосредственно (если разрешает геометрия конструктива) или через переходные «прямые» кабели с розеткой и вилкой, в которых контакты соединяются «один в один». Переходные кабели могут быть и переходниками с 9 на 25–штирьковые разъемы (рис.3.8).

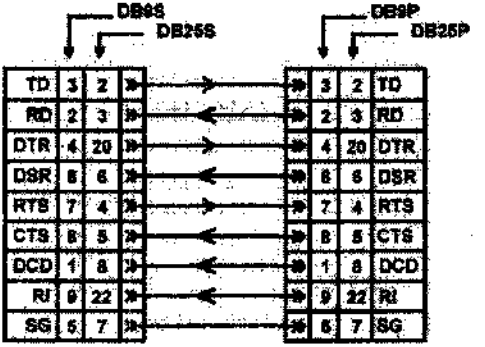


Рис.3.8. Кабель подключения модемов

Если устройства соединяется без модемов, то разъемы устройств соединяются между собой нуль – модемным кабелем, который имеет на обеих концах разъемы, контакты которых соединяются по одной из схем, приведенных на рис.3.9.

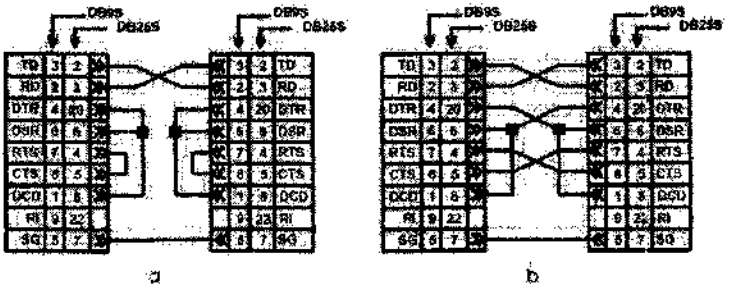


Рис.3.9. Варианты схем нуль-модемного кабеля

Если на каком–нибудь устройстве DTE (принтер, плоттер) установлена розетка – это почти стопроцентный признак того, что к другому устройству (компьютеру) оно должно подключаться прямым кабелем, аналогичным кабелю подключения модема. Розетка устанавливается обычно на те устройства, в которых подключение через модем не предусмотрено

В табл.3.2 приводится назначение контактов разъемов COM–портов (и любых других устройств DTE).

Таблица 3.2. **Разъемы и сигналы интерфейса RS–232C**

Обозначение цепи		Контакт разъема		Провод шлейфа выносного разъема PC				Направление	Название цепи
RS232	Стык 2	DB25S	UB9S	1*	У	у	4*	1/0	
PG	101	1	–	(10)	(10)	(10)	1	–	Protect Ground – Защита на земля
TD	103	2	3	3	5	3	3	0	Transmit Data – Переданные данные
RD	104	3	9	2	3	4	5	1	Receive Data – Принятые данные

Продолжение табл. 3.2.

Обозначение цепи		Контакт разъема		Провод шлейфа выносного разъема PC				Направление	Название цепи
RTS	105	4	7	7	4	8	7	0	Request To Send – Запрос на передачу
CTS	100	5	8	8	0	7	9	1	Clear To Send – Готовность модема к приему данных для передачи
DSR	107	6	6	6	2	9	11	1	Data Set Ready – Готовность модема к работе
SG	102	7	5	5	9	1	13		Signal Ground – Схемная земля
DCD	109	8	1	1	1	5	15		Data Carrier Detected – Несущая выявленная
DTR	108/2	20	4	4	7	2	14	0	Data Terminal Ready – отовность терминала (PC) к работе
RI	125	22	9	9	8	6	18	1	Ring Indicator – Индикатор вызова

1* – шлейф 8–битных мультикарт.

V – шлейф 16–битных мультикарт и портов на системных платах.

C* – вариант шлейфа портов на системных платах.

4* – широкий шлейф к 25–контактному разъему.

Подмножество сигналов RS–232C, которые относятся к асинхронному режиму, рассмотрим с точки зрения COM – порта PC, который является по терминологии RS–232C терминалом данных (DTE). Следует помнить, что активному состоянию сигнала («включенное») отвечает отрицательный потенциал (ниже –3В) сигнала интерфейса, а состоянию «выключено» и логическому нулю – положительный (выше +3В). Сигналы интерфейса имеют следующее назначение:

PG – Защитная земля, соединяется с корпусом устройства и экраном кабеля.

SG – Сигнальная (схемная) земля, относительно которой действуют уровни сигналов.

TD – Последовательные данные – выход передатчика.

RD – Последовательные данные – вход приемника.

RT – Выход запроса передачи данных: состояние «включено» сообщает модему о наличии у терминала данных для передачи. В полудуплексном режиме используется для управления направлением – состояние «включено» является сигналом модема на переключение в режим передачи.

CT – Вход разрешения терминала передавать данные. Состояние «выключен» аппаратно запрещает передачу данных. Сигнал используется для аппаратного управления потоками данных.

DT – Выход сигнала готовности терминала к обмену данными. Состояние «включенный» поддерживает канал, который коммутируется, в состоянии соединения.

DSR – Вход сигнала готовности от устройств передачи данных (модем в рабочем режиме подключен к каналу и закончил действия по согласованию с устройствами на противоположном конце канала).

DC – Вход сигнала выявления изъятого модема.

RI – Вход индикатора вызова (звонка). В канале, который коммутируется, этим сигналом модем сигнализирует о принятии вызова.

3.4. Управление потоком передачи

Для управления потоком данных (Flow Control) могут использоваться два варианта протокола — аппаратный и программный. Иногда управление потоком путают с квитированием, но это разные методы достижения одной цели – согласование темпа передачи и приема. Квотирование (Handshaking) подразумевает посылку сообщения о получе-

нии элемента, в то время как управления потоком допускает посылку сообщения о невозможности следующего приема данных.

Аппаратный протокол управления потоком RTS/CT (Hardware Flow Control) использует сигнал CT, что позволяет остановить передачу данных, если приемник не готов к их приему. Работу этого протокола иллюстрирует рис.3.10. Передатчик «выпускает» очередной байт только при включенном состоянии линии CT. Байт, который уже начал передаваться, задержать сигналом CT невозможно (это гарантирует целостность посылки). Аппаратный протокол обеспечивает быструю реакцию передатчика на состояние приемника. Обычно микросхемы асинхронных приемо–передатчиков имеют не менее двух регистров у приемной части. Это позволяет реализовать обмен с аппаратным протоколом без потери данных, не прибегая к программной буферизации.

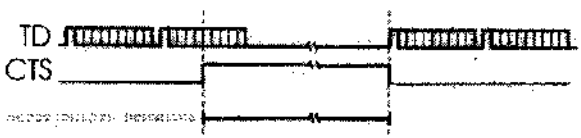


Рис.3.10. Аппаратное управление потоком

Аппаратный протокол удобно использовать при подключении принтеров и плоттеров, если они поддерживают этот режим (рис.3.11). При непосредственном (без модемов) соединении двух компьютеров аппаратный протокол требует перекрестного соединения линий RT – CT.

Если аппаратный протокол не используется, то при непосредственном соединении у передающего терминала должно быть обеспечено состояние «включено» на линии CT (обычно соединением собст-

венных линий RT – CT). В противном случае передатчик будет молчать.

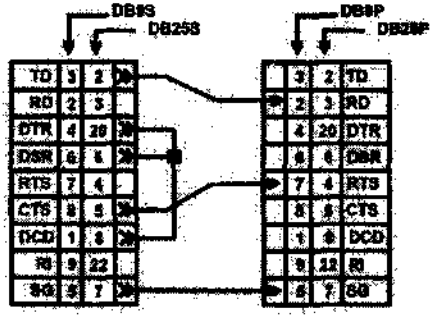


Рис.3.11. Кабель подключения принтера с протоколом RTS-CTS

Программный протокол управления потоком XON/XOFF допускает наличие двунаправленного канала передачи данных. Работает он таким образом: если устройство, которое принимает данные, обнаруживает причины, из–за которых оно не может их дальше принимать, оно по обратному последовательному каналу посылает XOFF (13h). Противоположное устройство, приняв этот символ, прекращает передачу. Дальше, когда принимающее устройство снова становится готовым к приему данных, оно посылает символ XON (11h), приняв который противоположное устройство восстанавливает передачу. Время реакции передатчика на смену состояния приемника в сравнении с аппаратным протоколом увеличивается по крайней мере на время передачи символа (XON или XOFF) плюс время реакции программы передатчика на прием символа (рис.3.12). Из этого следует, что данные без потерь могут приниматься только приемником, который имеет дополнительный буфер принятых данных.

Рис.3.12. Программное управление потоком XON/XOFF

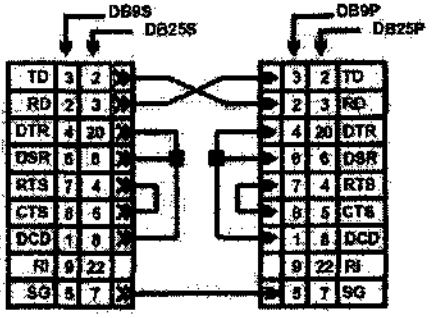


Рис.3.13. Кабель подключения принтера с протоколом XON/XOFF

Преимущество программного протокола при непосредственном соединении устройств состоит в отсутствии необходимости передачи управляющих сигналов интерфейса — минимальный кабель для двустороннего обмена может иметь только 3 провода (см. рис.3.9, а). Недостатком, кроме требования наличия буфера и большего времени реакции (снижающего и общую производительность канала из–за ожидания прохождения сигнала XON), является сложность реализации полнодуплексного режима обмена. В этом случае из потока принятых данных должны выделяться (и обрабатываться) символы управления потоком, который ограничивает набор передаваемых символов.

Кроме этих двух распространенных стандартных протоколов, поддерживаемых и устройствами, и операционными системами, существуют и некоторые другие. Например, некоторые плоттеры с после-

довательным интерфейсом используют программное управление, но посылают не стандартные символы XON/XOFF, а слова (ASCII – предложение). Такой обмен на уровне системной поддержки протокола практически не поддерживается (эти плоттеры рассчитанны на прямой диалог с прикладной программой). Кабель для подключения такого устройства совпадает с приведенным на рис.3.13.

3.4. Интерфейс «токовая петля»

Довольно распространенным вариантом последовательного интерфейса является «токовая петля». В этом интерфейсе электрическим сигналом является не уровень напряжения относительно общего провода, а ток в линии, которая соединяет приемник и передатчик. Обычно логической единице (и состоянию «включено») соответствует протекание тока 20 мА, а логическому нулю – отсутствие тока. Такое представление сигналов для вышеописанного формата асинхронной посылки разрешает обнаруживать состояние обрыва линии – в этом случае приемник обнаружит отсутствие стопового бита (обрыв линии действует как постоянно присутствующий логический нуль).

Токовая петля обычно предполагает наличие гальванической развязки входных цепей приемника (оптрона) от схемы устройства. При этом источником тока в петле будет передатчик (этот вариант называют активным передатчиком). Возможно и питание от приемника (активный приемник). При этом исходный ключ (оптронный) передатчика может быть также гальванически развязан с другой схемой передатчика.

Токовая петля с гальванической развязкой разрешает передавать сигналы на расстояния до единиц километров. Допустимое расстояние определяется сопротивлением пары проводов и уровнем помех. По-

скольку этот интерфейс требует пару проводов для каждого сигнала, обычно используют только два сигнала интерфейса. В случае двунаправленного обмена используются только сигналы переданных и принятых данных, а для управления потоком используется программный метод XON/XOFF. Если двунаправленный обмен не нужен, используют только одну линию данных, а для управления потоком обратная линия используется для сигнала CTS (аппаратный протокол) встречной линии данных (для программного протокола).

Превратить сигналы RS-232C в токовую петлю можно с помощью несложной схемы, приведенной на рис.3.14. В качестве примера выбрано подключение принтера с токовой петлей к COM–порту с аппаратным управлением потоком. Здесь для получения двухполярного сигнала, необходимого для входных сигналов COM–порта, применяется питание от интерфейса.

При некоторой «ловкости» программного обеспечения одной токовой петлей можно обеспечить и двунаправленную полудуплексную связь двух устройств (компьютеров). Спецификой такого соединения есть то, что каждый приемник «слышит» как сигналы передатчика на противоположной стороне канала, так и сигналы своего передатчика. Эта ситуация расценивается коммуникационными пакетами (например, классический KERMIT) просто как эхо–сигнал. Естественно, для безошибочного приема передатчики должны работать только поочередно.

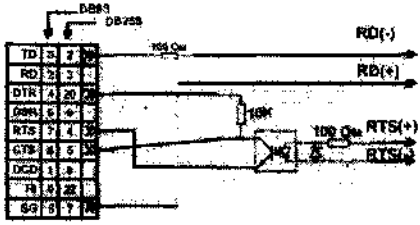


Рис.3.14. Подключение принтера с интерфейсом «токовая петля» к COM-порту

3.6. Инфракрасный интерфейс

Применение излучателей и приемников инфракрасного диапазона разрешает осуществлять беспроводные коммуникации между парой устройств, расположенных на расстоянии до одного метра, а иногда даже нескольких метров. Различают инфракрасные системы связи низкой скорости (до 115,2 Кбит/с) средней и высокой, работающие со скоростями 1,152 и 4 Мбит/с соответственно. Низкоскоростные системы подходят для обмена короткими сообщениями, высокоскоростные — для обмена файлами между компьютерами, подключения к локальной (или глобальной) сети, вывода информации на принтеры, проекционные аппараты и т.п. В перспективе ожидаются и более высокие скорости обмена, которые разрешат передавать даже видео. В 1993 году была создана ассоциация разработчиков систем инфракрасной передачи данных IrDA (Infrared Data Association), призвана обеспечить совместимость оборудования от разных производителей. Первым стандартом, принятым IrDA был Serial Infrared standart (SIR). В данное время действует стандарт IrDA 1.1, кроме которого имеются собственные системы фирм Hewlett Packard HP–SIR (Hewlett Packard Slow Infra Red) и ASK (Amplitude Shifted Keyed IR) фирмы Sharp. Основные (скоростные) характеристики интерфейсов следующие:

4– IrDA SIR (Slow Infra Red), HP–SIR – 9,6–115,2 Кбит/с;

4– IrDA MIR (Middle Infra Red) – 1,2 Мбит/с;

4– IrDA FIR (Fast Infra Red) – 4 Мбит/с;

4 Sharp ASK – 9,6–57,6 Кбит/с.

На скоростях до 115,2 Кбит/с для инфракрасной связи используются UART, совместимые с 16450/16550. Подключается ИК - порт непосредственно к материнской плате, в большинстве случаев расположение контактов на плате соответствует схеме и описано в документации на плату. В случае не совпадения выводов их легко переставить. После установки необходимо включить в BIOS поддержку инфракрасного порта. На использование инфракрасной связи часто может конфигурироваться порт COM2. В этом случае на переднюю панель компьютера устанавливается внешний приемо-передатчик – «инфракрасный глаз», который подключается к разъему IR–Connector системной платы.

На средних и высоких скоростях обмена применяются специализированные микросхемы, ориентированные на интенсивный программно–управляемый обмен или DMA, с возможностью использования прямого управления шиной (Bus Master). Получить доступ к IrDA как к обычному COM порту можно, если устройство подключается в COM порт или в IrDA разъем на материнской плате.

В общем виде схема организации IrDA - канала выглядит примерно так, как показано на рис.3.15.

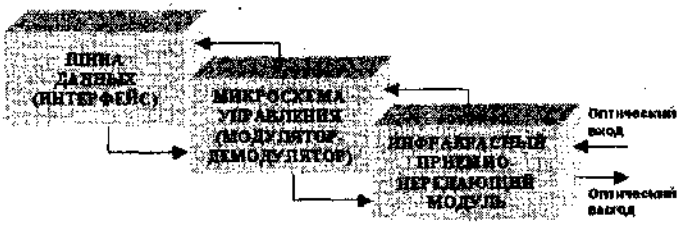


Рис.3.15. Типовая блок-схема организации IrDA – канала

Канал передачи данных состоит из двух основных элементов: микросхемы, обеспечивающей модуляцию и демодуляцию поступающего двоичного сигнала по определенному алгоритму, и инфракрасного приемно-передающего модуля. Рассмотрим SIR-стандарт, обеспечивающий скорость передачи до 115200 Кбит/с. В данном стандарте используется модуляция «3/16». Принцип данного вида модуляции проиллюстрирован на рис.3.16.

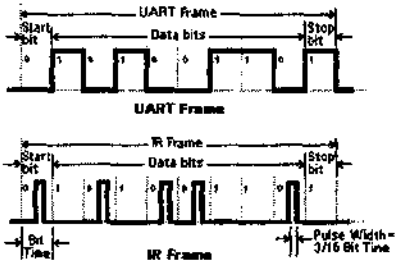


Рис.3.16. Принцип модуляции, используемой в SIR - стандарте

Длительность импульса, передаваемого на приемно-передающий модуль, равна 3/16 от длительности номинального бита данных. Кроме того, при SIR – модуляции используется инверсия бита данных. Эти преобразования обеспечиваются первым основным элементом схемы – модулирующей микросхемой. В зависимости от используемо-

го интерфейса, применяются разные микросхемы. К таким микросхемам относятся: PC87334VLJ, PC87334VJG (National Semiconductors); FDC37C665IR, FDC37C666IR (SMC), TDFS4000. На рис.3.17 представлена принципиальная схема ИК-порта.

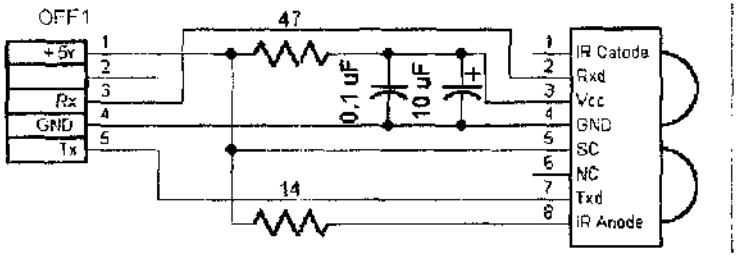


Рис.3.17. Принципиальная схема ИК - порта

Если необходимо организовать ИК - связь через последовательный порт RS-232, то для кодировки сигнала в соответствии с IrDA – стандартом используются два элемента: преобразователь уровней RS-232 (например MAX232), и микросхема кодировки сигнала (HSDL-7000, HSDL-7001). На рис.3.18 представлена структурная схема для организации ИК - связи через последовательный порт.

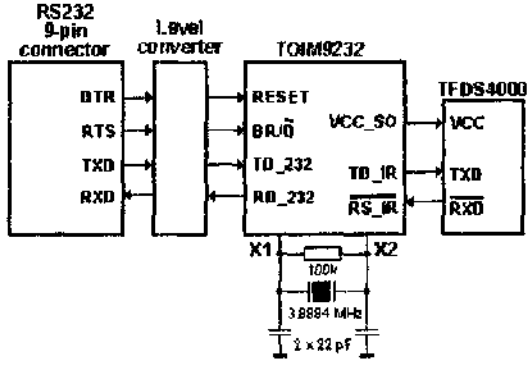


Рис.3.18. Схема организации ИК - связи через последовательный порт

В отличие от других беспроводных систем связи (радиочастотных), инфракрасные излучатели не создают препятствий в радиочастотном диапазоне и обеспечивают достаточный уровень конфиденциальности связи. ИК–лучи не проходят через стены, и расстояние приема ограничивается небольшим, легко контролируемым пространством. Инфракрасный интерфейс имеют и некоторые модели принтеров.

3.7. Интерфейс MIDI

Цифровой интерфейс музыкальных инструментов MIDI (Musical Instrument Digital Interface) является двунаправленным последовательным асинхронным интерфейсом с частотой передачи 31,25 Кбит/с. Этот интерфейс, разработанный в 1983 году, стал фактическим стандартом для соединения компьютеров, синтезаторов, микшеров и другой электромузыкальной техники. В данное время интерфейс MIDI имеют и дорогие синтезаторы, и дешевые музыкальные клавиатуры, которые могут использоваться как устройства ввода в компьютер.

В интерфейсе применяется токовая петля 10 м (возможно и 5 м) с гальванической (оптронной) развязкой входной цепи. Эта развязка исключает связь «схемных земель» устройств, которые соединяются, через интерфейсный кабель, который убирает помехи (фон), нежелательные для звуковой техники. Снижению интерференционных препятствий служит и выбор частоты передачи, которая совпадает с одним из значений частот квантования, принятых в цифровой звукозаписи.

Формат асинхронной посылки содержит стартовый бит, 8 бит информации и 1 стоповый бит. Контроль четности отсутствует. Старший бит посылки является признаком «команда–данные». Его нулевое

значение указывает на наличие семи бит данных в младших разрядах. При единичном значении признака биты [6:4] содержат код команды, а биты [3:0] — адрес приемника. Команды могут быть как адресованными конкретному устройству, так и широковещательными безадресными. К последней группе относятся команды старта, стопа и оценки времени, которые обеспечивают синхронизацию устройств (система синхронизации MTC – MIDI Time Code). Интерфейс определяет три типа портов: MIDI–IN, MIDI–OUT и MIDI–THRU.

Входной порт MIDI–IN представляет собой вход интерфейса «токовая петля 10 м», с гальванически отделенным от приемника оптроном и быстродействием не хуже 2 мкс. Устройство отслеживает информационный поток на этом входе и реагирует на адресованные ему команды и данные.

Исходный порт MIDI–OUT представляет собой выход источника тока 10 м, гальванически связанного со схемой устройства. Ограничительные резисторы предохраняют исходные цепи от повреждения при замыкании на землю или источник 5 В. На выход подается информационный поток от данного устройства. В этом потоке может содержаться и транслированный входной поток, но это далеко не всегда так.

Транзитный порт MIDI–THRU служит только для ретрансляции входного сигнала. Его наличие не является обязательным для всех устройств.

Применяются 5–контактные разъемы DIN, распространенные в бытовой звуковой технике. На всех устройствах устанавливаются розетки, на кабелях – вилы. Все соединительные кабели MIDI унифицированы (см. схему на рис.3.19). В соответствии правилам подключения, контакт 2 – экран кабеля, соединяется с общим проводом только

на стороне передатчика (на разъемах MIDI–OUT и MIDI–THRU). На разъеме MIDI–IN этот контакт свободный.

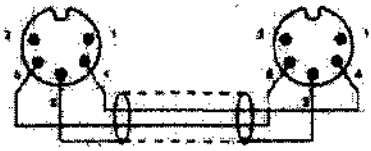


Рис.3.19. Соединительные кабели MIDI

В маркировании входов и выходов, указанной возле разъемов, бывают разночтения. Одни производители считают, что надо писать In или Out соответственно функции разъема данного устройства, и это, наверное, правильно: любой кабель будет соединять In и Out. Другие считают, что подпись должна обозначать функцию устройства, которое подключается, и тогда кабель будет соединять разъемы с обозначениями In — In и Out — Out. Такое маркирование встречается реже, но и ее следует иметь в виду.

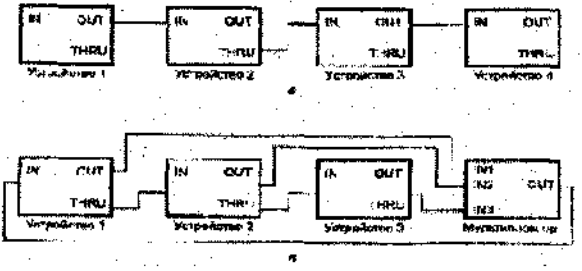


Рис.3.20. Варианты топологии сети MIDI: а – кольцо;
б – кольцо с мультиплексором

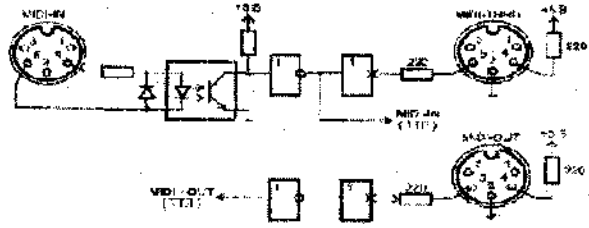


Рис.3.21. Вариант схемы кабеля адаптера MIDI

Интерфейс разрешает объединить группу до 16 устройств в локальную сеть. Возможные варианты топологии должны подчиняться главному правилу: вход MIDI–IN одного устройства должен подключаться к выходу MIDI–OUT или MIDI–THRU другого устройства. При планировании MIDI–сетей необходимо руководствоваться знаниями информационных потоков и связей устройств. Управляющие устройства – клавиатуры, секвенсоры (в режиме воспроизведения), источники синхронизации – должны находиться, естественно, перед управляемыми. Если устройства имеют потребность в двунаправленном обмене, они должны соединяться в кольцо. Возможное применение и специальных устройств – мультиплексоров, которые позволяют логически коммутировать множество входных потоков в один исходный. Несколько вариантов соединения приведены на рис.3.20.

В PC MIDI–порт имеется на большинстве плат звуковых адаптеров, и его сигналы выведены на не используемые контакты (12 и 15) разъема игрового адаптера. При этом для подключения стандартных устройств MIDI нужен переходной адаптер, который реализует интерфейс «токовая петля». Переходной адаптер обычно встраивается в специальный кабель, типовая схема которого приведена на рис.3.21.

Некоторые модели PC имеют встроенные адаптеры и стандартные 5-штырьковые разъемы MIDI.

В PC для MIDI–порта обычно применяются микросхемы UART, совместимые с MPU401. Эти микросхемы отличаются от обычных UART 8250 или 8251 в основном тем, что имеют регистр адреса устройства. При приеме команды с адресом устройства, которое совпадает с заданным в этом регистре (или с широковещательным адресом), вырабатывается запрос аппаратного прерывания. Это разрешает интерфейсу игнорировать команды, не адресованные данному устройству, без привлечения к фильтрации ресурсов процессора.

На некоторых системных платах применяются БИС контроллеров интерфейсов, в которых UART, используемый для COM–порта и управляемый через BIOS SETUP, может быть переведен в режим MIDI–порта.

3.8. Конфигурирование COM-портов

Начиная с первых моделей, в PC есть последовательный интерфейс – COM–порт (Communications Port – коммуникационный порт). Этот порт обеспечивает асинхронный обмен по стандарту RS–232C. Компьютер может иметь до четырех последовательных портов COM1–COM4 (для машин класса AT типично наличие двух портов). COM–порты имеют внешние разъемы (Male – «папа») DB25–S или DB9P, которые выведены на заднюю панель компьютера (назначение выводов приведено в табл.3.1).

COM–порты реализуются на микросхемах UART, совместных с семейством i8250. Они занимают в пространстве ввода/вывода по 8 8–битных регистров и могут располагаться по стандартным базовым адресам 3F8h (COM1), 2F8h (COM2), 3E8h (COM3), 2E8h (COM4). Для

портов COM3 и COM4 возможны альтернативные адреса 3EOh; 338h и 2EOh, 238h соответственно. Для PS/2 стандартными для портов COM3–COM8 являются адреса 3220h, 3228h, 4220h, 4228h, 5220h и 5228h соответственно.

Порты могут вырабатывать аппаратные прерывания IRQ4 (обычно используются для COM1 и COM3) и IRQ3 (для COM2 и COM4). Кроме того, возможно использование линий прерываний IRQ11 (вместо IRQ4) и IRQ10 (вместо IRQ3). Возможность разделяемого использования одной линии запроса несколькими портами (или ее разделения с другими устройствами) зависит от реализации аппаратного подключения и программного обеспечения. При использовании портов, установленных на шину ISA, разделенные прерывания обычно не работают.

Управление последовательным портом разделяется на два этапа – предварительное конфигурирование (Setup) аппаратных средств порта и текущее (оперативное) переключение режимов работы прикладным или системным программным обеспечением. Способ и возможности конфигурирования COM–портов зависят от способа его реализации и местоположения. Порт, расположенный на плате расширения, устанавливаемой в слот ISA или ISA+VLB, обычно конфигурируется с помощью джамперов на самой плате. Порт, расположенный на системной плате, обычно конфигурируется через BIOS Setup. Конфигурированию подлежат следующие параметры:

– базовый адрес, который может иметь значения 3F8h, 2F8h, 3E8h (3E0h 338h), 2E8h (2E0h, 238h). При инициализации BIOS проверяет наличие портов по адресам именно в этом порядке и, соответственно, присваивает выявленным портам логические имена COM1, COM2, COM3 и COM4.

– используемая линия запроса прерывания: для COM1 и COM3 обычно используется IRQ4 или IRQ11, для COM2 и COM4 – IRQ3 или IRQ10. В принципе номер прерывания можно назначать в произвольных сочетаниях с базовым адресом (номером порта), но некоторые программы и драйверы (например, драйверы последовательной мыши) настроены только на стандартные соединения. Каждому порту, который нуждается в аппаратном прерывании, обычно назначают отдельную линию, которая не совпадает с линиями запроса прерываний других портов или устройств. Разделяемое использование линий прерывания адаптеров шин ISA проблематично. Прерывания необходимы для портов, к которым подключаются устройства ввода (мышь, дигитайзер), UPS и модемы. Прерываниями обычно не пользуются и при связи двух компьютеров нуль–модемным кабелем.

– использование канала DMA (для UART 16450 или 16550, расположенных на системной плате) – разрешение использования и номер канала DMA. Режим DMA при работе с COM–портами используют редко, поэтому в большинстве случаев каналы DMA портам не назначают. Режим работы порта по умолчанию (2400 бит/с, 7 бит данных, 1 стоповый бит и контроль четности), заданный при инициализации порта во время BIOS POST, может изменяться в любой момент при настройке коммуникационных портов.

3.9. Использование COM–портов

Вопреки названию, COM–порты чаще всего используют для подключения манипуляторов (мышь, трекбол). В этом случае порт используется в режиме последовательного ввода, обеспечивая питание устройства от интерфейса. Мышь может подключаться к любому исправному порту, для согласования разъемов порта и мыши возможно

применение переходника DB9S–DB25P или наоборот, DB25S–DB9P. Для работы с мышью обязательно нужно использование линии прерывания, причем для порта COM1 – IRQ4, а для COM2 – IRQ3.

Следующим по популярности идет подключение внешних модемов для связи с компьютерами или выхода в глобальные сети. Модемы должны подключаться полным (9–выводным) кабелем DTE–DCE, схема которого приведена на рис. 9.16. Этот же кабель может использоваться и для согласования разъемов (по количеству контактов). Возможно и применение переходников DB9S–DB25P, предназначенных для манипуляторов типа «мышь». Для работы коммуникационного программного обеспечения обычно необходимо использование прерываний, но здесь, как правило, больше свободы выбора порта и номера линии прерывания. Если предполагается работа на скоростях 9600 бит/с и выше, то COM–порт должен быть реализован на микросхеме UART 16550A или совместимой с ней. Возможности работы с использованием FIFO–буферов и обмена по каналам DMA зависят от коммуникационного програмного обеспечения.

Для связи двух компьютеров, расположенных друг от друга на небольшом расстоянии, используют и непосредственное соединение нуль–модемным кабелем (варианты схем приведены на рис.3.9). Использование программ типа Norton Commander или Interink MS–DOS разрешает обмениваться файлами со скоростью передачи до 115,2 Кбит/с без использования аппаратных прерываний. Это же соединение может использоваться и сетевым пакетом Lantastic, который предоставляет более развитый сервис.

Подключение принтеров и плоттеров к COM–порту требует применения кабеля, который соответствует выбранному протоколу управления потоком: программному XON/XOFF или аппаратному

RTS/CTS. Схемы кабелей приведены на рис.3.11 и рис.3.13. Аппаратный протокол предпочтительнее, поскольку он не требует программной поддержки со стороны PC. Прерывание при выводе средствами DOS (командами COPY или PRINT) не используются.

COM–порт иногда используется и для подключения электронных ключей (Security Devices), предназначенных для защиты от нелицензионного использования программных продуктов. Эти устройства могут быть как «прозрачными», разрешая воспользоваться тем же портом и для подключения периферии, так и целиком занимают порт.

COM–порт при наличии соответствующей программной поддержки позволяет превратить PC в терминал, эмитируя систему команд распространенных специализированных терминалов (VT–52, VT–100 и других).

Этим списком, конечно же, возможности использования COM–порта не исчерпываются. Интерфейс RS–232C широко распространен в разных периферийных устройствах и терминалах. Все они, при наличии надлежащей программной поддержки, могут подключаться к PC. Кроме использования по прямому назначению, COM–порт может использоваться и как двунаправленный интерфейс, у которого имеется 3 программно–управляемые исходные линии и 4 входные линии. Возможность их использования ограничивается только фантазией разработчика. Существует, например, схема однобитного широтно–импульсного преобразователя, который разрешает записывать звуковой сигнал на диск PC, используя входную линию COM–порта. Воспроизведение этой записи через обычный динамик обеспечивает разборчивость языка. Конечно, в настоящее время, когда звуковая карта имеется в каждом PC, это уже не удивляет, но в свое время такое решение было довольно интересным.

3.10. Неисправности и тестирование COM–портов

Проблемы с COM–портами чаще всего случаются при установке новых портов или после неудачного подключения внешнего устройства (при нарушении требования отключения источника питания перед всеми перекоммутациями).

3.10.1. Проверка конфигурирования

Тестирование последовательных портов (как и параллельных) начинают с проверки их узнавания системой. Список адресов установленных портов обычно появляется в таблице заставки, выведенной BIOS на экран перед загрузкой ОС. Кроме этой таблицы, список можно посмотреть и с помощью тестовых программ или прямо в BIOS DATA AREA с помощью любого отладчика, или диспетчера устройств операционной системы (рис.3.22).

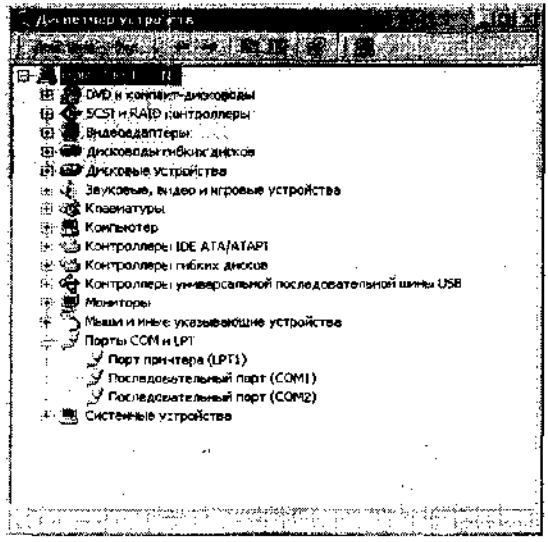


Рис.3.22. Окно диспетчера устройств

Если BIOS обнаруживает меньше портов, чем установлено физически, скорее всего каким–нибудь двум портам присвоен один и тот же адрес, или установлен нестандартный адрес какого–нибудь порта. Проблемы могут возникать с адресами портов COM3 и COM4: не все версии BIOS будут искать порты по альтернативным адресам 3E8h, 338h, 2E8h и 238h, а иногда не производится поиск и по адресам 3E8h и 2E8h.

Если двум портам назначен одинаковый адрес, тестовая программа найдет ошибки порта только с использованием внешней заглушки (External LoopBack). Программное тестирование порта без заглушки не покажет ошибок, поскольку при этом включается диагностический режим и конфликтующие порты будут работать параллельно, обеспечивая совпадение информации, которая считывается. В реальной работе, естественно, нормальный ввод данных (и управляющих сигналов интерфейса) для конфликтующих портов невозможен. Разбираться с конфликтом адресов удобно последовательно устанавливая порты и наблюдая по адресам, которые появляются в списке.

Если физически установлен только один порт и его не обнаруживает BIOS, то причины могут быть теми же, что и с LPT–портом: или он отключен при конфигурировании, или вышел из строя скорее всего из–за нарушения правил подключения. При работе с COM–портом часто используются аппаратные прерывания — их используют при подключении модема, мыши и других устройств ввода. Неработоспособность этих устройств может быть вызвана некорректными настройками запроса прерывания. Здесь возможны как конфликты с другими устройствами, так и несоответствие номера прерывания адресу порта.

3.10.2. Функциональное тестирование

В первом приближении COM–порт можно проверить диагностической программой (например, Checkit) без использования заглушек. Этот режим тестирования проверяет функционирование микросхемы UART (внутренний диагностический режим) и работу прерываний, но он не касается входных и выходных буферных микросхем, которые являются более частыми источниками неприятностей. Если тест не проходит, причину следует искать в конфликте адресов (или прерываний, если на это явным образом указывает сообщение теста), или в самой микросхеме UART.

Для более достоверного тестирования портов с помощью диагностических программ рекомендуется использование внешней заглушки, которая подключается к разъему COM–порта. В отличие от LPT–порта, в COM–порте количество входных сигналов превышает количество выходных, что разрешает выполнить полную проверку всех цепей. Схема заглушки для тестирования COM–порта программой Checkit приведена на рис.3.23. Заглушка соединяет выход приемника с входом передатчика, замыкая информационное кольцо. Обязательная для всех схем заглушек перемычка RTS–CTS разрешает работать передатчику – без неё символы не смогут передаваться. Исходный сигнал DTR обычно используют для проверки входных линий DSR, DCD и RI. Если тест с внешней заглушкой не проходит (при успешной внутренней диагностике UART), причину следует искать во внешних буферах или их питании (±)12 В, или в шлейфах подключения внешних разъемов. Здесь может помочь осциллограф или просто вольтметр.

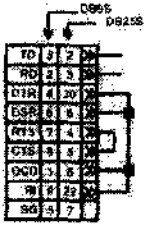


Рис.3.23. Заглушка для проверки COM-портов

Последовательность проверки может быть следующей:

1. Проверить наличие двухполярного питания исходных схем передатчиков (этот шаг логически первый, но поскольку он технически самый сложный, его можно и отложить на крайний случай, когда появится желание заменять буферные микросхемы).

2. Проверить напряжение на выходах TD, RTS и DTR: после аппаратного сброса на выходе TD должен быть отрицательный потенциал приблизительно –12 В (по крайней мере, ниже –5 В), а на выходах RTS и DTR – такой же положительный. Если этих потенциалов нет, возможная ошибка подключения разъема к плате через шлейф.

Распространенные варианты:

- шлейф не подключен;
- шлейф подключен неправильно;
- раскладка шлейфа не отвечает разъему платы.

Первые два варианта проверяются внимательным осмотром, а третий может потребовать некоторых усилий.

Если дело только в ошибочной раскладке, то эти три исходных сигнала можно найти на других контактах разъемов (на входных контактах потенциал совсем небольшой). Если эти сигналы найти не удалось, то возможно вышли из строя буферные формирователи.

Соединив контакты линий RTS и CTS (или установив заглушку), следует попробовать вывести небольшой файл на COM–порт (например, командой COPY C:\AUTOEXEC. BAT COM1:). С исправным портом эта команда успешно выполнится за несколько секунд с сообщением об успешном копировании. При этом потенциалы на выходах RTS и DTR должны измениться на отрицательные, а на выходе TD должна появиться пачка двуполярних импульсов с амплитудой более 5 В. Если потенциалы RTS и DTR не изменились, ошибка опять–таки в буферных формирователях. Если на выходе RTS (и входе CTS) появился отрицательный потенциал, а команда COPY завершается с ошибкой, скорее всего, вышел из строя приемник линии CTS (другой вариант – опять–таки ошибка в шлейфе). Если команда COPY успешно проходит, а изменения на выходе TD не появляются, то проблема в буферном передатчике сигнала TD.

Если буферные элементы включены в состав интерфейсной БИС (что теперь очень распространенно), то такой порт почти неремонтопригодный (по крайней мере, в обычных условиях). Неисправный COM–порт, установленный на системной плате, можно попробовать отключить опциями BIOS SETUP, но порт мог сгореть и вместе со схемой своего отключения – тогда он может остаться «живым мертвецом» в карте портов ввода/вывода и прерываний. Иногда такой «мертвец» целиком выводит из строя системную плату.

При работе с COM–портами источниками ошибок могут быть разъемы и кабели. Разъемы могут иметь плохие контакты, а кабели кроме возможных обрывов, могут иметь плохие частотные характеристики. Частотные свойства кабелей обычно обозначаются при большой длине (десятки метров) на высоких скоростях обмена (56 или 115 Кбит/с). При необходимости использования длинных кабелей на вы-

соких скоростях (например, для связи двух PC) сигнальные провода данных должны быть перевитыми с отдельными проводами «схемной земли».

При подключении к COM–порту устройств с небольшим энергопотреблением возникает соблазн использования питания от выходных линий интерфейса. Если линии управления DTR и RTS не используются по прямому назначению, их можно использовать как линии питания с напряжением около 12В на холостом ходу. Ток короткого замыкания на «схемную землю» ограничен буферной микросхемой передатчика на уровне 20 мА. Линия TD в покое находится в состоянии логической единицы, которая на выходе вырабатывает отрицательное напряжение. Потенциалами этих линий можно управлять через регистры COM–порта (выход TD вырабатывает положительное напряжение, если установить бит BRCON) или API–функции. [3]

3.11. Программирование UART для микроконтроллеров

Микроконтроллеры Atmel оснащены полнодуплексным универсальным приемо–передатчиком (UART). Его основные возможности следующие:

- Генератор обеспечивает любую скорость передачи информации в бодах;
- Высокая скорость передачи при низкой частоте тактового генератора;
- 8–разрядный или 9–разрядный форматы данных;
- Фильтрация шума;
- Выявление переполнения;
- Выявление ошибок формирования кадров;
- Детектирование бита ошибочного старта;

- Три отдельных прерывания: по завершению передачи (TX Complete), по пустому регистру передачи данных (TX Data Register Empty) и по завершению приема (RX Complete).

3.11.1. Передача данных

Передача данных инициируется записью передаваемых данных в регистр данных ввода–вывода UART (UDR). Данные пересылаются из регистра UDR в сдвиговый регистр передачи в следующих случаях:

– новый символ записан в UDR после того как был передан из регистра сдвига стоповый бит предыдущего символа. Сдвиговый регистр загружается немедленно.

– новый символ записан в UDR прежде, чем был передан стоповый бит предыдущего символа. Данные в регистр сдвига загружаются после выхода стопового бита переданного символа.

– если с 10 (11) – разрядного сдвигового регистра передачи выведена вся информация, в него пересылаются данные с UDR. В это время устанавливается бит UDRE (UART Data Register Empty) регистра состояния UART (USR). При установленном в состояние 1 бите UDRE (регистр данных пуст), UART готов принять следующий символ. Флаг UDRE сбрасывается при записи данных в регистр UDR. В то же время, когда данные пересылаются из регистра UDR в 10 (11)–разрядный регистр сдвига, бит 0 сдвигового регистра устанавливается в состояние 0 (состояние 0 – стартовый бит), а бит 9 или 10 устанавливается в состояние 1 (состояние 1 – стоповый бит). Если в регистре управления UART (UCR) установлен бит CHR9 (т.е. выбран режим 9–разрядного слова данных), то бит TXB8 регистра UCR пересылается в бит 9 сдвигового регистра передачи.

Сразу после пересылки данных в сдвиговый регистр, стартовый бит передается на вывод TXD. За ним происходит передача остальных бит данных. Когда будет выдан стоповый бит, сдвиговый регистр загружается новой порцией данных, если они была записана в UDR во время передачи. В процессе загрузки бит UDRE находится в установленном состоянии. Если же новые данные не будут загружены в UDR к выдаче стопового бита, флаг UDRE остается установленным. В этом случае, после того, как стоповый бит будет присутствовать на выводе TXD на протяжении одного такта, в регистре состояния UART (USR) устанавливается флаг завершения передачи TXC (TX Complete Flag).

Установленный в состояние 1 бит TXEN регистра UCR разрешает передачу UART. При очищении бита TXEN (сброс в состояние 0), вывод PD1 может быть использован как порт ввода/вывода общего назначения. При установленном бите TXEN передатчик UART подключается к PD1 и использует его как вывод выхода, независимо от установки бита 1 в регистре направления DDRD. На рис.3.24 представлена программная модель передатчика UART.

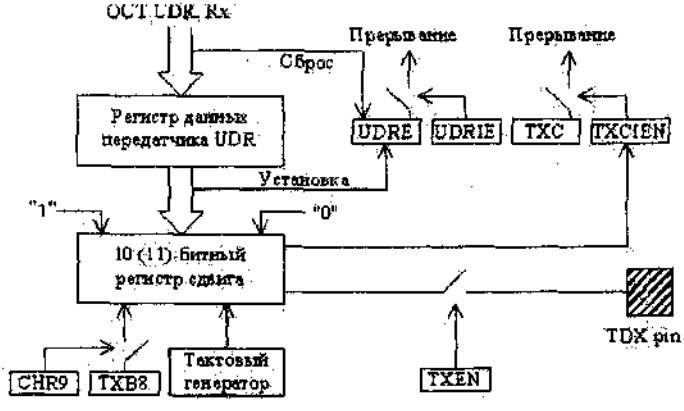


Рис.3.24. Программная модель передатчика UART

3.11.2. Прием данных

Логика восстановления данных делает выборку состояний вывода RXD с частотой в 16 раз большей, чем скорость обмена. Когда линия находится в пассивном состоянии, одиночная выборка нулевого логического уровня будет интерпретироваться как падающий фронт стартового бита и будет запущена последовательность детектирования стартового бита. Считается, что первая выборка нашла первый нулевой логический уровень вероятного стартового бита. На выборках 8, 9 и 10 приемник снова тестирует вывод RXD на смену логических состояний. Если две или более из этих трех выборок найдут логические 1, то данный вероятный стартовый бит откидывается как шумовой всплеск и приемник начнет обнаруживать и анализировать следующие переходы с 1 в 0.

Если же был выявлен действительный стартовый бит, то начинает генерироваться выборка следующих за стартовым битом информационных бит. Эти биты также тестируются на выборках 8, 9 и 10. Логическое состояние бита принимается по двум и или трем одинаковым состояниям выборок. Все биты вводятся в сдвиговый регистр приемника с тем значением, которое было определено тестированием выборок.

Рис.3.25. Тестирование выборок принимаемых данных

При поступлении стопового бита необходимо чтобы не менее двух выборок из трех подтвердили прием стопового бита (показали высокий уровень). Если же две или более выборки покажут значение 0, то при пересылке принятого байта в UDR, в регистре статуса UART

(USR) устанавливается бит ошибки кадра FE (Framing Error). Для выявления ошибки кадра пользователь перед считыванием регистра UDR должен проверять состояние бита FE. Флаг FE очищается при считывании содержания регистра данных UART (UDR).

Вне зависимости от того, принят правильный стоповый бит или нет, данные пересылаются в регистр UDR и устанавливается флаг RXC в регистре состояния UART (USR). Регистр UDR фактически есть двумя физически отдельными регистрами, один из которых служит для передачи данных, а другой для приема. При считывании UDR обращение ведется к регистру приема данных, при записи обращения ведется к регистру передачи. Если выбран режим обмена 9–разрядными словами данных (установлен бит CHR9 регистра UCR), при пересылке данных в UDR бит RXB8 регистра UCR загружается в бит 9 сдвигового регистра передачи. Если после получения символа к регистру UDR не было обращения, начиная с последнего приема, в регистре UCR устанавливается флаг переполнение (OR). Это означает, что новые данные, которые пересылаются в сдвиговый регистр, не могут быть переданы в UDR и будут потеряны. Пользователю для выявления переполнения необходимо всегда проверять флаг OR после считывания содержания регистра UDR.

При сброшенном бите RXEN регистра UCR приемник запрещен. Это означает, что вывод PD0 может использоваться как порт ввода/вывода общего назначения. При установленном бите RXEN, приемник UART подключается к выводу PD0, который работает как вывод входа, вне зависимости от значения ссобветствующего бита в регистре направления передачи данных бита DDRD. На рис.3.26 представлена программная модель приемника UART.

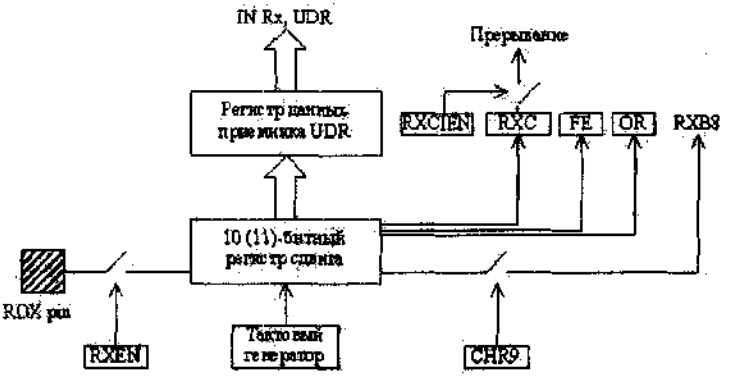


Рис.3.26. Программная модель приемника UART

3.11.3. Управление UART

РЕГИСТР ДАННЫХ UART – UDR – (UART I/O Data Register)

Бит	7	6	5	4	3	2	1	0	
$0C ($2C)	MSB							LSB	UDR
Чтение/Запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Начальное значение	0	0	0	0	0	0	0	0	

В действительности регистр UDR есть двумя физически разделенными регистрами – регистром передачи данных и регистром приема данных, которые используют те самые адреса I/O. При записи в регистр запись генерируется в регистр передачи данных UART, при считывании проходит считывание содержания регистра приема данных UART.

РЕГИСТР СОСТОЯНИЯ UART – USR – (UART Status Register)

Бит	7	6	5	4	3	2	1	0	
$08 ($28)	RXC	TXC	UDRE	FE	DOR	-	-	-	USR
Чтение/Запись	R	R/W	R	R	R	R	R	R	
Начальное значение	0	0	0	0	0	0	0	0	

Регистр USR обеспечивает только считывание информации о состоянии UART.

• Bit 7 – RXC: UART Receive Complete – Прием завершен. Данный бит устанавливается в состояние 1 при пересылке принятого символа из регистра приема в UDR. Бит устанавливается вне зависимости от того имеются ли ошибки при приеме кадра. При установленном в UCR бите RXCIE и установленном бите RXC выполняется прерывание по завершению приема UART. Бит RX очищается при считывании UDR. При приеме данных возникает прерывание. Подпрограмма обработки прерывания по завершению приема UART должна считывать значение из регистра данных UDR для того, чтобы очистить бит RXC, иначе при выходе из подпрограммы обработки прерывания произойдет новое прерывание.

• Bit 6 – TXC: UART Transmit Complete – Передача завершена. Данный бит устанавливается в состояние 1 когда весь символ (включая стоповый бит) передан из сдвигового регистра передачи и в UDR не записаны новые данные. Этот флаг используется при полудуплексном связном интерфейсе, когда оборудование передачи должно установить режим приема и освободить коммуникационную шину сразу после завершения передачи. При установленном в регистре UCR бите TXCIE, установка TXC приведет к выполнению прерывания по завершению передачи UART. Флаг TXC очищается аппаратно при выполнении обработки соответствующего вектора прерывания. Очистить бит TXC можно записью значения 1 в эту позицию.

• Bit 5 – UDRE: UART Data Register Empty – Регистр данных пустой. Данный бит устанавливается в состояние 1 когда символ, записанный в UDR, пересылается в сдвиговый регистр передачи. Установка этого бита означает, что передатчик готов к получению нового символа для передачи. Если бит UDRIE в UCR установлен, выполняется прерывание по завершению передачи UART до тех пор, пока будет установлен бит

UDRE. Бит UDRE сбрасывается при записи данных в регистр UDR. При приеме данных возникает прерывание. Подпрограмма обработки прерывания по пустому регистру данных UART должна записать данные в регистр UDR для того, чтобы очистить UDRE, иначе по окончанию подпрограммы прерывания возникнет новое прерывание.

• Bit 4 – FE: Framing Error – Ошибка кадрирования. Данный бит устанавливается в состояние 1 при выявлении условий ошибочного приема кадра, т.е. когда стоповый бит входного символа в состоянии 0. Бит FE очищается при приеме стопового бита с логическим уровнем 1.

• Bit 3 – DOR: Data OverRun – Переполнение данных Бит DOR устанавливается в состояние 1 при обнаружении условий переполнения, т.е. когда символ, который уже находится в регистре UDR не был считан перед пересылкой нового символа из сдвигового регистра приема. Бит DOR буферизирован. Это означает, что он будет оставаться установленным пока не будут правильно считаны данные из регистра UDR. Бит DOR сбрасывается когда данные приняты и переданы в регистр UDR.

• Bits 2..0 – Res: Reserved bits – Зарезервированные биты. Эти биты в микроконтроллерах ATmega зарезервированные и при считывании всегда имеют значение 0.

РЕГИСТР УПРАВЛЕНИЯ UART – UCR – (UART Control Register)

Бит	7	6	5	4	3	2	1	0	
$0A ($2A)	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8	UCR
Чтение/Запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Начальное значение	0	0	0	0	0	0	0	0	

• Bit 7 – RXCIE: RX Complete Interrupt Enable – Разрешение прерывания по завершению приема. При установленном в состояние 1 бите RXCIE и установленном разрешении глобального прерывания, установка бита RXC в регистре USR приведет к выполнению прерывания по завершению приема.

• Bit 6 – TXCIE: TX Complete Interrupt Enable – Разрешение прерывания по завершению передачи. При установленном в состояние 1 бите TXCIE и установленном разрешении глобального прерывания, установка бита TXC в регистре USR приведет к выполнению прерывания по завершению передачи.

• Bit 5 – UDRIE: UART Data Register Empty Interrupt Enable – Разрешение прерывания по пустому регистру данных. При установленном в состояние 1 бите UDRIE и установленном разрешении глобального прерывания, установка бита UDRE в регистре USR приведет к выполнению прерывания по пустому регистру данных UART.

• Bit 4 – RXEN: Receiver Enable – Разрешение приемника. Установленный в состояние 1 бит RXEN разрешает работу приемника UART. Если приемник запрещен, то флаги состояния TXC, DOR и FE установить невозможно. Если флаги установлены, то сброс бита RXEN не приведет к сбросу этих флагов.

• Bit 3 – TXEN: Transmitter Enable – Разрешение передатчика. Установленный в состояние 1 бит TXEN разрешает работу передатчика UART. При запрете работы передатчика во время передачи символа, передатчик не будет заблокирован до того, как будет полностью передан символ из регистра сдвига плюс любой следующий символ, который находится в UDR.

• Bit 2 – CHR9:9 Bit Characters – Режим 9–разрядных символов. При установленном в состояние 1 бите CHR9 передаются и принимаются 9–разрядные символы плюс стартовый и стоповый биты. Девятые биты читаются и записываются с использованием бит RXB8 и TXB8 регистра UCR. Девятый бит данных может использоваться как дополнительный стоповый или бит контроля четности.

• Бит 1 – RXB8: Receive Data Bit 8 – Прием 8–разрядных данных. При установленном в состояние 1 бите CHR9, бит RXB8 является девятым битом данных принятого символа.

• Бит 0 – TXB8: Transmit Data Bit 8 – Передача 8–разрядных данных. При установленном в состояние 1 бите CHR9, бит TXB8 является девятым битом в передаваемом символе.

3.11.4. Бод–генератор (Baud Rate Generator)

Бод–генератор представляет собой делитель, который генерирует импульсы передачи с частотой, определенной формулой:

$$BAUD = \frac{f_{CK}}{16(UBRR+1)}, \quad \text{где:}$$

BAUD = скорость передачи данных в бодах.

f_{CK} = частота кварцевого генератора.

UBRR – содержание регистра UBRR (Baud Rate register = 0 – 255).

При использовании стандартных кварцевых генераторов, наиболее часто используемые скорости передачи (в бодах) могут быть получены установками UBRR.

РЕГИСТР БОД-ГЕНЕРАТОРА UART – UBRR (UART Baud Rate Register)

Бит	7	6	5	4	3	2	1	0	
$29 ($29)	MSB							LSB	UBRR
Чтение/Запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Начальное значение	0	0	0	0	0	0	0	0	

Регистр UBRR – это 8–разрядный регистр с возможностью считывания/записи данных. Скорость передачи данных UART определяется значением, записанным в этот регистр.

3.12. Сопряжение компьютера с микроконтроллером по COM-порту

На рис.3.27 представлена схема микропроцессорной системы контроля температуры.

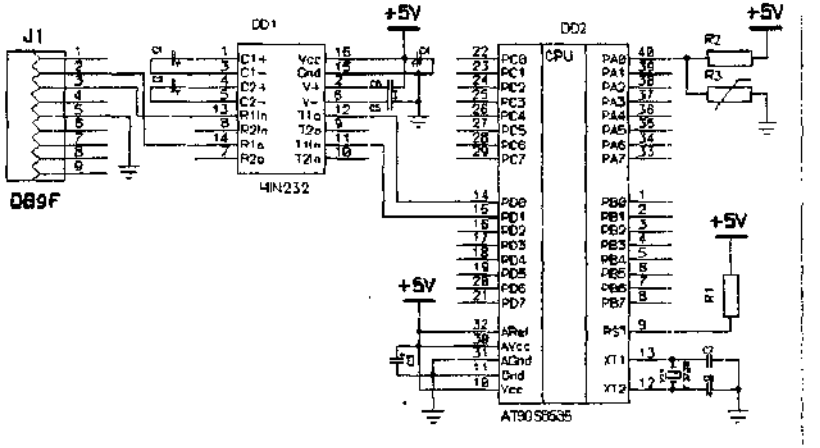


Рис.3.27. Микропроцессорная система контроля температуры

Для согласования интерфейсов COM-порта и микроконтроллера использована микросхема преобразователя уровней HIN232 (MAX232). Для её работы необходимо подключит 5 внешних элементов – конденсаторов емкостью 1 мкФ. Линии RxD и TxD соединяются с выводами TxD и RxD микроконтроллера. Частота кварца выбрана равной 7,3728 МГц. На такой частоте работы микроконтроллера, при использовании встроенного UART, не возникает ошибка кадрирования. Емкости конденсаторов C7 и C8 равны 27 пФ, C3 – 1 мкФ. Для работы микроконтроллера необходимо установить уровень лог.1 на входе RST, потому этот вход подтянут к линии питания через сопро-

тивление 4,3 кОм. Опорное напряжение аналого-цифрового преобразователя равно 5В.

3.13. Программа для микроконтроллера

Микроконтроллеры ATMega оснащены полнодуплексным универсальным асинхронным приемо-передатчиком. Передача данных инициируется записью передаваемого байта данных в регистр данных UDR, после чего сразу же помещается в регистр сдвига. По окончанию передачи байта данных устанавливается бит UDRE регистра статуса UART (USR), что говорит о готовности передачи очередного байта. Если был принят байт, установится бит RXC регистра статуса. Принятый байт может быть считан из регистра UDR. Если бит RXEN регистра управления UART сброшен, работа приемника запрещена. Это означает, что выход контроллера, подключенный к UART, может быть использован как вывод общего назначения. При установленном бите RXEN приемник подключается к этому выводу. Работа передатчика разрешается установкой бита TXEN в регистре управления.

Пример 3.1. Написать программу для микроконтроллера ATMega8535, которая выполняет оцифровку аналогового сигнала (снимаемого с выхода датчика температуры), который подается на вход PA0. Полученное значение преобразовать к 8 битам и отправить в компьютер. Преобразование АЦП начинается по приему команды 0xBC.

Текст программы приводится ниже.

```
#include <iom8535v.h>// подключение заголовочных файлов
#include <macros.h>
void init_device(void)  // функция инициализации контроллера
```

```
{
DDRD=0b00000010;   // настройка второго вывода порта D на
выход
ADCSRA=0x86;       // настройка АЦП; частота работы – 125
кГц;
// преобразование выполняется за 14 тактов
ADMUX=0;  // канал мультиплексора АЦП - 0
UCSRA=0x00;
UCSRB=0x98; // настройка UART – разрешена работа прием-
ника
// и передатчика, а также прерывание по приему данных
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x30;    // 9600 бит/с при 7,3728 МГц
SEI();         // разрешение прерываний
}
// обработчик прерывания по приему данных
#pragma interrupt_handler uart0_rx_isr:12
void uart0_rx_isr(void)
{
 char rc;              // дополнительные переменные
 char data;
 rc=UDR;               // считывание принятого байта данных
 if(rc==0xBC)          // если принята команда 0xBC
 {
 data=read_adc(0);     // вызов функции преобразования АЦП
 UDR=data;             // передача байта в компьютер
 }
```

```
}
// функция перобразования АЦП
unsigned char read_adc(char adc_input)
{
unsigned char adcw;   // переменная для хранения данных
char a;               // дополнительные переменные
char b;
ADMUX=adc_input;   // установка канала мультиплексора
ADCSRA|=0x40;         // начало преобразования АЦП
while(!(ADCSRA&(1<<ADIF)));  // ожидание завершения пре-
образования
ADCSRA&=~0x10;              // сброс флага завершения преоб-
разования
a=ADCL;                  // считывание младшего байта
b=ADCH;                  // считывание старшего байта
adcw=b<<6;               // формирование 8-битного значения
a=a>>2;                  // 2 младших бита отбрасываются
adcw=adcw+a;
return adcw;             // выход из функции
}
void main(void)   // главная функция
{
 init_device();      // вызов функции инициализации контролле-
ра
 while(1);           // вечный цикл – ожидается прерывание
}
```

Весь алгоритм работы программы реализован в обработчике прерывания по приему данных. Полученное значение сравнивается с командой 0xBC (начало преобразования). Если условие выполняется, то вызывается функция read_adc(0), после выполнения которой возвращается значение, пропорциональное амплитуде аналогового сигнала на входе. Параметр функции – номер канала АЦП. В приведенном примере – это 0 канал. Запуск преобразования АЦП выполняется установкой бита ADSC. Затем выполнение программы приостанавливается до тех пор, пока не установится бит ADIF в регистре состояния. Это говорит о том, что преобразование завершилось и данные могут быть считаны из регистров. Разрядность АЦП в микроконтроллере 10 бит. Поэтому необходимо привести 10-битное значение к 8 битному. В примере это выполнено отбрасыванием младших 2 бит. Полученное значение записывается в регистр UDR, после чего оно сразу же передается в компьютер.

4. ПРОГРАММИРОВАНИЕ COM-ПОРТОВ

4.1. Открытие порта

С последовательными портами в Win32 работают как с файлами. Следовательно, начинать надо с открытия порта как файла, используя функцию CreateFile. В качестве одного из параметров функции передается имя порта (например, COM1), с которым будет выполняться работа. Попытка создать файл с таким именем приводит к перенаправлению информации на соответствующий порт. Прототип функции выглядит следующим образом:

```
HANDLE CreateFile(
LPCTSTR lpFileName,
DWORD dwDesiredAccess,
DWORD dwShareMode,
LPSECURITY_ATTRIBUTES lpSecurityAttributes,
DWORD dwCreationDispostion,
DWORD dwFlagsAndAttributes,
HANDLE hTemplateFile
);
```

Описание параметров функции приводится ниже.

lpFileName

В виде строки передается имя открываемого порта (например, "COM1").

dwDesiredAccess

Определяет режим доступа к порту. Приложение может получить доступ на чтение, запись, чтение/запись данных из/в порт. Этот параметр может быть GENERIC_READ – чтение, GENERIC_WRITE –

запись, GENERIC_READ | GENERIC_WRITE – возможность выполнения операций чтения и записи.

dwShareMode

Определяет режим разделения порта между разными процессами. К порту одновременно могут обратиться несколько процессов. Этот параметр должен быть равен 0. Это означает, что после открытия порта его нельзя будет повторно открыть до тех пор, пока он не будет закрыт использующим его приложением.

lpSecurityAttributes

При работе с портами этот параметр не используется и должен принимать значение NULL.

dwCreationDispostion

Задаёт способ создания файла. Для коммуникационных портов в это поле должен быть занесен флаг OPEN_EXISTING, указывающий на то, что открывать следует существующий порт

dwFlagsAndAttributes

Задаёт атрибуты создаваемого файла, а также управляет различными режимами обработки. В случае выполнения синхронных операций чтения-записи этот параметр принимает значение FILE_ATTRIBUTE_NORMAL.

hTemplateFile

Задатся дескриптор файла-шаблона, который в данном случае не используется. Этот параметр должен быть равным NULL.

При успешном открытии порта функция CreateFile возвратит в основную программу его дескриптор (HANDLE). Если же по каким-либо причинам открыть порт не удалось, этой переменной присваивается значение INVALID_HANDLE_VALUE. Таким образом, проверка идентификатора порта на его равенство INVALID_HANDLE_VALUE

позволяет выполнить обработку исключительных ситуаций (например, отсутствие открываемого порта на материнской плате или его использование операционной системой).

Пример 1.

Используя среду разработки Visual Studio .NET, создать программу, которая по нажатию кнопки «**Открыть порт**» открывала коммуникационный порт COM2 для обмена информацией с внешним устройством в режиме синхронного чтения-записи и отображала в текстовом поле результат его открытия.

Пояснение. Порядок действий.

1. Запускаем среду разработки Visual Studio .NET. Создаем новый проект с названием COMTest. В нашем примере приложение создается на базе диалогового окна.

2. Проектируем внешний вид диалогового окна. Помещаем на форму кнопку (IDC_BUTTON1) и два текстовых поля (IDC_STATIC1, IDC_STATIC2), формируем размер окна, задаём заголовки элементов в поле «Caption» на странице свойств (рис.4.1). Следует обратить внимание, что при добавлении на форму элемента управления «Static text» (текстовое поле), ID по умолчанию принимает значение IDC_STATIC. Поэтому ID необходимо отредактировать.

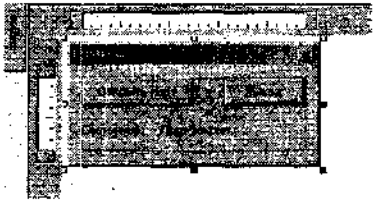


Рис.4.1. Вид диалогового окна

3. Привязываем к элементу управления текстовое поле (IDC_STATIC2) переменную m_status. Для этого необходимо выделить элемент управления и из контекстного меню выбрать команду

«Add Variable». Появится диалоговое окно, показанное на рис.4.2. В поле «Variable name» задать имя переменной.

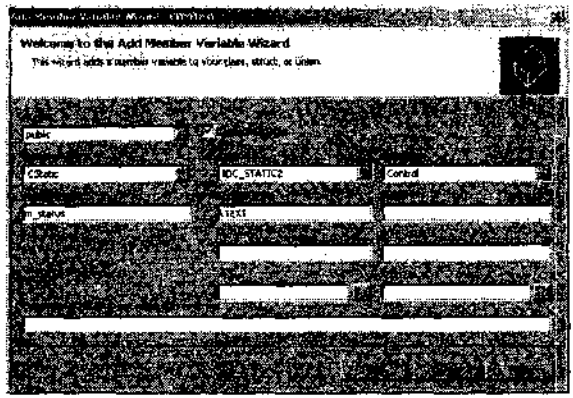


Рис.4.2. Добавление переменной-члена класса

4. Добавляем переменную – член класса CCOMTestDlg, в которой будет храниться дескриптор порта. Для этого в окне «Class View» выделяем класс «CCOMTestDld» и из контекстного меню, в подменю «Add» выбираем «Add Variable» (рис.4.3).

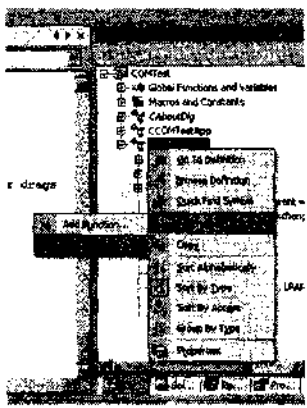


Рис.4.3. Добавление переменной

Имя переменной должно быть m_hCom, а тип переменной – HANLDE.

5. Добавляем обработчик нажатия на кнопку «Открыть порт». Функция – обработчик события нажатия на кнопку должна иметь следующий вид:

```
void CCOMTestDlg::OnBnClickedButton1()
{
// открываем последовательный порт
  m_hCom=CreateFile("COM2",GENERIC_READ | GE-
NERIC_WRITE,0,NULL,OPEN_EXISTING,0,NULL);
  if(m_hCom==INVALID_HANDLE_VALUE)
        m_status.SetWindowText("Невозможно открыть порт!");
  // если невозожно открыть порт, выводится информация в
текстовом поле
  else  // если порт открыт
        m_status.SetWindowText("Порт открыт.");
// сообщаем, что порт открыт
}
```

На рис.4.4. показано диалоговое окно программы после компиляции проекта.

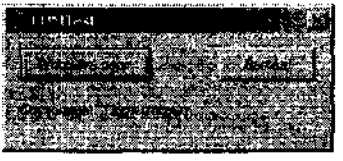


Рис.4.4. Окно программы

Нажав на кнопку «Открыть порт» и при успешном выполнении операции, появится соответствующая надпись.

Недостатком созданного фрагмента программы является отсутствие у пользователя возможности интерактивного выбора открываемого порта. В том случае, если порт COM2, привязка к которому жёстко

указана в параметре lpFileName функции CreateFile, отсутствует или используется операционной системой компьютера для обслуживания мыши или модема, в текстовом поле появится сообщение «Невозможно открыть порт», а обратиться к свободному порту – например, COM1 – из этой же программы не представляется возможным.

Для демонстрации более гибкого способа инициализации последовательных портов рассмотрим следующий пример.

Пример 2. Модифицировать проект, созданный в предыдущем примере, так, чтобы предоставить пользователю возможность самостоятельного назначения используемого коммуникационного порта из всех присутствующих в системе.

Пояснение. Порядок действий:

1. Запускаем среду разработки Visual Studio .NET и открываем предыдущий проект (COMTest).

2. Добавляем на форму элемент управления «Combo Box» - список, из которого пользователь сможет выбрать нужный порт. Вид диалогового окна программы представлен на рис.4.5.

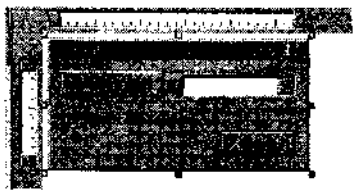


Рис.4.5. Модифицированный вид диалогового окна

В поле «Type» в окне свойств списка выбрать тип «DropList».

3. Привязываем к элементу управления «Combo Box» переменную m_port. На рис.4.6 представлено диалоговое окно добавления переменной с требуемыми параметрами.

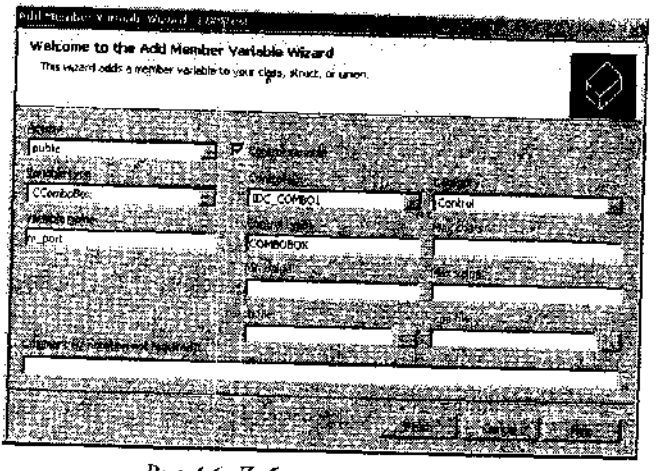


Рис.4.6. Добавление переменной

4. Поиск последовательных портов, установленных в системе, будет осуществляться с помощью анализа ключей системного реестра ОС Windows. Для работы с реестром используется класс CRegKey. В начале файла COMTestDlg.cpp необходимо добавить строку:

#include "Atlbase.h"

Следующий код необходимо добавить в конец функции OnInit-Dialog() класса CCOMTestDlg, которая вызывается перед появлением окна программы на экране.

```
CRegKey reg; // класс для работы с реестром
HKEY key;  // дескриптор ключа реестра
LONG result;
result=reg.Open(HKEY_LOCAL_MACHINE,
"HARDWARE\\DEVICEMAP\\SERIALCOMM"); // открытие
ключа реестра
```

```
if(result!=ERROR_SUCCESS) MessageBox("Нет доступа к ре-
естру!");
else
{
char s[255]; // буфер
DWORD size;              // переменная для передачи в функцию
for(int i=0;i<30;i++)// в цикле считываются значения из реестра
{
CString valname; // строковая переменная
valname.Format("\\Device\\Serial%d",i);
// форматирование строки
result=reg.QueryValue(s,valname,&size);
// чтение данных из реестра
if(result==ERROR_SUCCESS)    // если данные из реестра по-
лучены
{
CString tmp(s);
m_port.AddString(tmp); // добавляем в список название портов
}
}
}
m_port.SetCurSel(0);    // установка текущего выделения
reg.Close();          // закрываем реестр
m_status.SetWindowText("Порт закрыт.");
// обновление информации в строке статуса
m_hCom=NULL; // присвоение дескриптору порта значения
NULL
```

5. Корректируем обработчик нажатия на кнопку «Открыть порт».

```
void CCOMTestDlg::OnBnClickedButton1()
{
CString sp;
// получаем имя выбранного порта
m_port.GetLBText(m_port.GetCurSel(),sp);
// открываем последовательный порт
m_hCom=CreateFile(sp,GENERIC_READ | GE-
NERIC_WRITE,0,NULL,OPEN_EXISTING,0,NULL);
if(m_hCom==INVALID_HANDLE_VALUE)
m_status.SetWindowText("Невозможно открыть порт!");
// если невозожно открыть порт, выводится информация в тек-
стовом поле
else     // если порт открыт
m_status.SetWindowText("Порт открыт.");
// сообщаем, что порт открыт
}
```

6. Результат работы программы показан на рис 4.7.

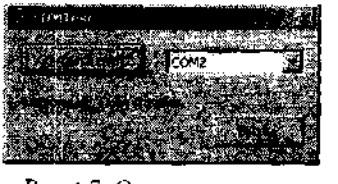


Рис.4.7. Окно программы

В обоих приведенных выше примерах закрытие используемой программой коммуникационного порта происходит при закрытии пользователем диалогового окна (т.е. при завершении работы программы). В процессе работы программы может возникнуть необходи-

мость внести изменения в параметры работы порта. Кроме того, Закрытие порта осуществляется с помощью функции CloseHandle, которая имеет единственный параметр – дескриптор закрываемого объекта. При успешном завершении функция возвращает TRUE, а при ошибке – FALSE. Ниже приведен пример, который демонстрирует использование функции CloseHandle.

Пример 3. Дополнить окно программы, созданной в предыдущем примере, кнопкой **«Закрыть порт»**, которая выполняла бы одноимённую функцию. Обеспечить возможность одновременной работы только с одним портом.

Пояснение. Порядок действий:

1. Запускаем среду разработки Visual Studio .NET. Открываем предыдущий проект.

2. Добавляем на форму кнопку и изменяем текст в поле «Caption» на странице свойств, на «Закрыть порт» (рис.2.8.). Привязываем к кнопке «Открыть порт» переменную с именем m_bOpen, а к кнопке «Закрыть порт» переменную с именем m_bClose.

3. Добавляем обработчик события нажатия на кнопку «Закрыть порт» и изменяем его следующим образом:

```
void CCOMTestDlg::OnBnClickedButton2()
{
CloseHandle(m_hCom);        // закрытие порта
m_status.SetWindowText("Порт закрыт.");
m_port.EnableWindow(1);    // разблокируем элементы управления
m_bOpen.EnableWindow(1); // список и кнопка «Открыть порт»
```

```
m_bClose.EnableWindow(0); // блокировка кнопки «Закрыть
порт»
}
```

4. Для обеспечения возможности работы только с одним портом, изменяем обработчик события нажатия на кнопку «Открыть порт» следующим образом:

```
void CCOMTestDlg::OnBnClickedButton1()
{
CString sp; // получаем имя выбранного порта
m_port.GetLBText(m_port.GetCurSel(),sp);
// открываем последовательный порт
m_hCom=CreateFile(sp,GENERIC_READ | GE-
NERIC_WRITE,0,NULL,OPEN_EXISTING,0,NULL);
if(m_hCom==INVALID_HANDLE_VALUE)
{
m_status.SetWindowText("Невозможно открыть порт!");
return;
}
// если невозожно открыть порт, выводится информация в тек-
стовом поле
else    // если порт открыт
m_status.SetWindowText("Порт открыт.");
m_bOpen.EnableWindow(0); // блокировка кнопки «Открыть
порт»
m_bClose.EnableWindow(1); // разблокируем кнопку «Закрыть
порт»
m_port.EnableWindow(0); // блокировка списка
}
```

После нажатия на кнопку, в случае успешного открытия порта, элемент управления «Combo Box» и кнопка «Открыть порт» станут неактивными, и будут находиться в таком состоянии до тех пор, пока не будет нажата кнопка «Закрыть порт».

5. В конце функции OnInitDialog необходимо добавить следующую строку для того, чтобы кнопка «Закрыть порт» была неактивной при запуске программы.

```
m_bClose.EnableWindow(0);
```

Теперь проект можно компилировать.

На рис.4.8 представлено диалоговое окно модифицированной программы.

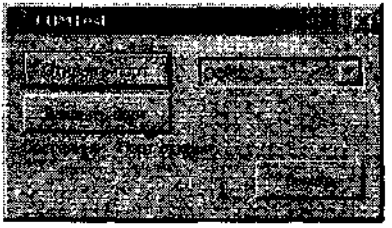


Рис.4.8. Окно программы

Создайте копию этого проекта. Далее, на основе этого примера, будет разработана программа обмена данными между двумя компьютерами.

3.2. Настройка параметров порта

Открыв порт, мы получили его в свое распоряжение. Теперь с портом может работать только наша программа. Однако, прежде чем организовывать обмен данными, необходимо настроить порт. Это ка-

сается только последовательных портов, для которых должна быть задана скорость обмена, параметры четности, формат данных и прочее. Кроме того, существует несколько специфичных для Windows параметров. Речь идет о тайм-аутах, которые позволяют контролировать как интервал между принимаемыми байтами, так и общее время приема сообщения. Есть возможность управлять состоянием сигналов управления модемом.

Последовательность шагов, которые необходимо проделать в процессе работы с портом, приводится на рис.4.9.

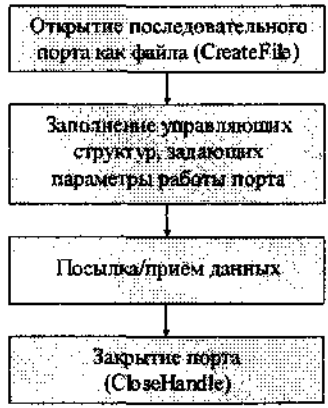


Рис.4.9. Процесс работы с портом

Основные параметры последовательного порта описываются структурой DCB:

```
typedef struct _DCB {
 DWORD DCBlength;
 DWORD BaudRate;
 DWORD fBinary:1;
 DWORD fParity:1;
 DWORD fOutxCtsFlow:1;
 DWORD fOutxDsrFlow:1;
```

```
DWORD fDtrControl:2;
DWORD fDsrSensitivity:1;
DWORD fTXContinueOnXoff:1;
DWORD fOutX:1;
DWORD fInX:1;
DWORD fErrorChar:1;
DWORD fNull:1;
DWORD fRtsControl:2;
DWORD fAbortOnError:1;
DWORD fDummy2:17;
WORD wReserved;
WORD XonLim;
WORD XoffLim;
BYTE ByteSize;
BYTE Parity;
BYTE StopBits;
char XonChar;
char XoffChar;
char ErrorChar;
char EofChar;
char EvtChar;
WORD wReserved1;
} DCB;
```

Эта структура содержит почти всю управляющую информацию, которая в реальности располагается в различных регистрах последовательного порта. Теперь разберемся, что означает каждое из полей самой важной структуры:

DCBlength

Задает длину, в байтах, структуры DCB. Используется для контроля корректности структуры при передаче ее адреса в функции настройки порта.

BaudRate

Скорость передачи данных. Возможно указание следующих констант: CBR_110, CBR_300, CBR_600, CBR_1200, CBR_2400, CBR_4800, CBR_9600, CBR_14400, CBR_19200, CBR_38400, CBR_56000, CBR_57600, CBR_115200, CBR_128000, CBR_256000. Как видно, эти константы соответствуют всем стандартным скоростям обмена. На самом деле, это поле содержит числовое значение скорости передачи, а константы просто являются символическими именами. Поэтому можно указывать, например, и CBR_9600, и просто 9600. Однако рекомендуется указывать символические константы.

fBinary

Включает двоичный режим обмена. Win32 не поддерживает недвоичный режим, поэтому данное поле всегда должно быть равно 1, или логической константе TRUE (что предпочтительней).

fParity

Включает режим контроля четности. Если это поле равно TRUE, то выполняется проверка четности, при ошибке, в вызывающую программу, выдается соответствующий код завершения.

fOutxCtsFlow

Включает режим слежения за сигналом CTS. Если это поле равно TRUE и сигнал CTS сброшен, передача данных приостанавливается до установки сигнала CTS. Это позволяет подключенному к компьютеру

прибору приостановить поток передаваемой в него информации, если он не успевает ее обрабатывать.

fOutxDsrFlow

Включает режим слежения за сигналом DSR. Если это поле равно TRUE и сигнал DSR сброшен, передача данных прекращается до установки сигнала DSR.

fDtrControl

Задает режим управления обменом для сигнала DTR. Это поле может принимать следующие значения:

DTR_CONTROL_DISABLE - Запрещает использование линии DTR

DTR_CONTROL_ENABLE - Разрешает использование линии DTR

DTR_CONTROL_HANDSHAKE - Разрешает использование рукопожатия для выхода из ошибочных ситуаций. Этот режим используется, в частности, модемами при восстановлении в ситуации потери связи.

fDsrSensitivity

Задает чувствительность коммуникационного драйвера к состоянию линии DSR. Если это поле равно TRUE, то все принимаемые данные игнорируются драйвером (коммуникационный драйвер расположен в операционной системе) за исключением тех, которые принимаются при установленном сигнале DSR.

fTXContinueOnXoff

Задает, прекращается ли передача при переполнении приемного буфера и передаче драйвером символа XoffChar. Если это поле равно TRUE, то передача продолжается, несмотря на то, что приемный бу-

фер содержит более XoffLim символов и близок к переполнению, а драйвер передал символ XoffChar для приостановления потока принимаемых данных. Если поле равно FALSE, то передача не будет продолжена до тех пор, пока в приемном буфере не останется меньше XonLim символов и драйвер не передаст символ XonChar для возобновления потока принимаемых данных. Таким образом, это поле вводит некую зависимость между управлением входным и выходным потоками информации.

fOutX

Задает использование XON/XOFF управления потоком при передаче. Если это поле равно TRUE, то передача останавливается при приеме символа XoffChar, и возобновляется при приеме символа XonChar.

fInX

Задает использование XON/XOFF управления потоком при приеме. Если это поле равно TRUE, то драйвер передает символ XoffChar, когда в приемном буфере находится более XoffLim, и XonChar, когда в приемном буфере остается менее XonLim символов.

fErrorChar

Указывает на необходимость замены символов с ошибкой четности на символ, задаваемый полем ErrorChar. Если это поле равно TRUE, и поле fParity равно TRUE, то выполняется замена.

fNull

Определяет действие, выполняемое при приеме нулевого байта. Если это поле TRUE, то нулевые байты отбрасываются при передаче.

fRtsControl

Задает режим управления потоком для сигнала RTS. Если это поле равно 0, то по умолчанию подразумевается

RTS_CONTROL_HANDSHAKE. Поле может принимать одно из следующих значений:

RTS_CONTROL_DISABLE - Запрещает использование линии RTS

RTS_CONTROL_ENABLE - Разрешает использование линии RTS

RTS_CONTROL_HANDSHAKE - Разрешает использование RTS рукопожатия. Драйвер устанавливает сигнал RTS когда приемный буфер заполнен менее, чем на половину, и сбрасывает, когда буфер заполняется более чем на три четверти.

RTS_CONTROL_TOGGLE - Задает, что сигнал RTS установлен, когда есть данные для передачи. Когда все символы из передающего буфера переданы, сигнал сбрасывается.

fAbortOnError

Задает игнорирование всех операций чтения/записи при возникновении ошибки. Если это поле равно TRUE, драйвер прекращает все операции чтения/записи для порта при возникновении ошибки. Продолжать работать с портом можно будет только после устранения причины ошибки и вызова функции ClearCommError.

fDummy2

Зарезервировано и не используется.

wReserved

Не используется, должно быть установлено в 0.

XonLim

Задает минимальное число символов в приемном буфере перед посылкой символа XON.

XoffLim

Определяет максимальное количество байт в приемном буфере перед посылкой символа XOFF. Максимально допустимое количество байт в буфере вычисляется вычитанием данного значения из размера приемного буфера в байтах.

ByteSize

Определяет число информационных бит в передаваемых и принимаемых байтах.

Parity

Определяет выбор схемы контроля четности. Данное поле должно содержать одно из следующих значений:

EVENPARITY	Дополнение до четности
MARKPARITY	Бит четности всегда 1
NOPARITY	Бит четности отсутствует
ODDPARITY	Дополнение до нечетности
SPACEPARITY	Бит четности всегда 0

StopBits

Задает количество стоповых бит. Поле может принимать следующие значения:

ONESTOPBIT	Один стоповый бит
ONE5STOPBIT	Полтора стоповых бита
TWOSTOPBIT	Два стоповых бита

XonChar

Задает символ XON используемый как для приема, так и для передачи.

XoffChar

Задает символ XOFF используемый как для приема, так и для передачи.

ErrorChar

Задает символ, использующийся для замены символов с ошибочной четностью.

EofChar

Задает символ, использующийся для сигнализации о конце данных.

EvtChar

Задает символ, использующийся для сигнализации о событии.

wReserved1

Зарезервировано и не используется.

Так как поля структуры DCB используются для конфигурирования микросхем портов, на них накладываются некоторые ограничения. Размер байта должен быть 5, 6, 7 или 8 бит. Комбинация из пяти битного байта и двух стоповых бит является недопустимой. Так же как и комбинация из шести, семи или восьми битного байта и полутора стоповых бит.

Рассмотренная структура DCB самая большая из всех, использующихся для настройки последовательных портов. Но она и самая важная. Заполнение всех полей этой структуры может вызвать затруднения, так как надо очень четко представлять, как работает последовательный порт. Поэтому ручную установку полей можно порекомендовать опытным программистам. Один из способов заполнения структуры - воспользоваться функцией BuildCommDCB, которая позволяет заполнить поля структуры DCB на основе строки, по синтаксису аналогичной строке команды mode. Прототип этой функции выглядит следующим образом:

BOOL BuildCommDCB(LPCTSTR lpDef, LPDCB lpDCB);

Функция имеет всего два параметра:

lpDef

Указатель на строку с конфигурационной информацией в формате команды mode. Например, следующая строка задает скорость 1200, без четности, 8 бит данных и 1 стоповый бит:

baud=1200 parity=N data=8 stop=1

lpDCB

Указатель на заполняемую структуру DCB. При этом структура должна быть уже создана и заполнена нулями, кроме поля DCBlength, которое должно содержать корректное значение. Возможно так же использование уже заполненной структуры DCB, например полученной вызовом одной из функций чтения параметров порта.

В случае успешного завершения функция BuildCommDCB возвращает не нулевое значение. В случае ошибки возвращается 0.

Второй способ предполагает выполнение такой последовательности шагов:

1. Считать текущие значения полей структуры DCB, установленные операционной системой по умолчанию;
2. Внести необходимые изменения в поля этой структуры;
3. Назначить вновь заполненную структуру для дальнейшего использования.

Заполнить поля структуры DCB сведениями о текущем состоянии устройства (точнее, о его настройках) позволяет функция GetCommState, прототип которой:

```
BOOL GetCommState(
HANDLE hFile,
LPDCB lpDCB
);
```

Функция очень проста и имеет всего два параметра:

hFile

Дескриптор открытого файла коммуникационного порта. Этот дескрипор возвращается функцией CreateFile. Следовательно, прежде чем получить параметры порта, Вы должны его открыть. Для функции BuildCommDCB это не требовалось.

lpDCB

Указатель на DCB. Для DCB должен быть выделен блок памяти. При успешном завершении функция возвращает ненулевое значение. При ошибке нуль. Получить параметры порта можно в любой момент, а не только при начальной настройке.

Заполнив DCB можно приступать к конфигурированию порта. Это делается с помощью функции SetCommState:

```
BOOL SetCommState(
HANDLE hFile,
LPDCB lpDCB
);
```

Эта функция имеет точно такие же параметры, как GetCommState. Различается только направление передачи информации. GetCommState считывает информацию из внутренних управляющих структур и регистров порта, а SetCommState наоборот заносит ее. Следует быть осторожным при вызове функции SetCommState, поскольку она изменит параметры даже в том случае, если очереди приема/передачи не пусты, что может вызвать искажение потока передаваемых или принимаемых данных. В случае успешного завершения возвращается отличное от нуля значение, а в случае ошибки – нуль.

4.3. Настройка тайм-аутов

Следующей важной управляющей структурой является COMMTIMEOUTS. Она определяет параметры временных задержек при приеме и передаче. Значения, задаваемые полями этой структуры, оказывают большое влияние на работу функций чтения/записи.

```
typedef struct _COMMTIMEOUTS {
DWORD ReadIntervalTimeout;
DWORD ReadTotalTimeoutMultiplier;
DWORD ReadTotalTimeoutConstant;
DWORD WriteTotalTimeoutMultiplier;
DWORD WriteTotalTimeoutConstant;
} COMMTIMEOUTS,*LPCOMMTIMEOUTS;
```

Поля структуры COMMTIMEOUTS имеют следующие значения:

ReadIntervalTimeout

Максимальное время, в миллисекундах, допустимое между двумя последовательными символами, считываемыми с коммуникационной линии. Во время операции чтения временной период начинает отсчитываться с момента приема первого символа. Если интервал между двумя последовательными символами превысит заданное значение, операция чтения завершается и все данные, накопленные в буфере, передаются в программу. Нулевое значение данного поля означает, что данный тайм-аут не используется. Значение MAXDWORD, вместе с нулевыми значениями полей ReadTotalTimeoutConstant и ReadTotalTimeoutMultiplier, означает немедленный возврат из операции чтения с передачей уже принятого символа, даже если ни одного символа не было получено из линии.

ReadTotalTimeoutMultiplier

Задает множитель, в миллисекундах, используемый для вычисления общего тайм-аута операции чтения. Для каждой операции чтения данное значение умножается на количество запрошеных для чтения символов.

ReadTotalTimeoutConstant

Задает константу, в миллисекундах, используемую для вычисления общего тайм-аута операции чтения. Для каждой операции чтения данное значение прибавляется к результату умножения ReadTotalTimeoutMultiplier на количество запрошенных для чтения символов. Нулевое значение полей ReadTotalTimeoutMultiplier и ReadTotalTimeoutConstant означает, что общий тайм-аут для операции чтения не используется.

WriteTotalTimeoutMultiplier

Задает множитель, в миллисекундах, используемый для вычисления общего тайм-аута операции записи. Для каждой операции записи данное значение умножается на количество записываемых символов.

WriteTotalTimeoutConstant

Задает константу, в миллисекундах, используемую для вычисления общего тайм-аута операции записи. Для каждой операции записи данное значение прибавляется к результату умножения WriteTotalTimeoutMultiplier на количество записываемых символов. Нулевое значение полей WriteTotalTimeoutMultiplier и WriteTotalTimeoutConstant означает, что общий тайм-аут для операции записи не используется.

Если не используются тайм-ауты для синхронной операции чтения и внешнее устройство, выйдя из строя, прекратило передачу, то

программа будет вечно ожидать завершения этой операции (другими словами, она «зависнет»). Если же тайм-ауты используются, то операция чтения завершится корректно – только количество фактически считанных байт будет меньше количества байт, запрошенных для чтения, но это не обязательно свидетельствует об ошибке. Например, программа может по тайм-ауту определять конец очередного блока данных. Аналогично и для операции записи, с той лишь разницей, что неполная передача данных из буфера, скорее всего, будет свидетельствовать о проблеме во внешнем устройстве, то есть будет считаться ошибкой.

Как и для заполнения структуры DCB, для COMMTIMEOUTS существует функция считывания установленных в системе значений. Это функция GetCommTimeouts:

```
BOOL GetCommTimeouts(
HANDLE hFile,
LPCOMMTIMEOUTS lpCommTimeouts
);
```

В качестве первого параметра передается дескриптор открытого порта. Второй параметр – указатель на структуру COMMTIMEOUTS. При успешном выполнении функции возвращается ненулевое значение.

Заполнив структуру COMMTIMEOUTS можно вызывать функцию установки тайм-аутов порта. Это функция называется SetCommTimeouts:

```
BOOL SetCommTimeouts(
HANDLE hFile,
LPCOMMTIMEOUTS lpCommTimeouts
);
```

Установку тайм-аутов можно производить как до установки параметров порта, так и после. Последовательность вызова функций SetCommState и SetCommTimeouts не имеет никакого значения. Главное, что бы все настройки были завершены до начала ввода/вывода информации.

Пример 4. Модифицировать программу, созданную в предыдущем примере, так, чтобы по нажатию кнопки «Открыть порт» она открывала коммуникационный порт компьютера, выбранный пользователем, и устанавливала скорость обмена 9600 бод и следующий формат посылки: количество бит в информационном блоке – 8; количество стоповых бит – 1; проверка на чётность – отсутствует. Также настроить значения тайм-аутов.

Пояснение. Порядок действий:

1. Запускаем среду разработки Visual Studio .NET. Открываем предыдущий проект.

2. Вносим изменения в код обработчика события нажатия на кнопку «Открыть порт» (IDC_BUTTON1).

```
void CCOMTestDlg::OnBnClickedButton1()
{
    CString sp;
    m_port.GetLBText(m_port.GetCurSel(),sp);
// получение названия выбранного в списке порта
    m_hCom=CreateFile(sp,GENERIC_READ | GE-
NERIC_WRITE,0,NULL,OPEN_EXISTING,0,NULL);
// открываем последовательный порт
    if(m_hCom==INVALID_HANDLE_VALUE)
    {
        m_status.SetWindowText("Невозможно открыть порт!");
```

```
// если невозможно открыть порт, выдаем сообщение
    return;
    }
    else  // если порт открыт
    {
        m_status.SetWindowText("Порт открыт.");
// сообщаем, что порт открыт
        m_bOpen.EnableWindow(0);     // блокировка элементов
управления
        m_bClose.EnableWindow(1);
        m_port.EnableWindow(0);

    }
    DCB *pDCB;   // объявление указателя на структуру
    COMMTIMEOUTS ct;     // структура для настройки тайм-аутов
    pDCB=new DCB;
    memset(pDCB,0,sizeof(DCB)); // обнуление всех полей перемен-
ной pDCB
    if(!GetCommState(m_hCom,pDCB))  // получение состояния порта
{
        MessageBox("Невозможно получить параметры порта!");
return;
}
    if(!GetCommTimeouts(m_hCom, &ct)) // получение значений
тайм-аутов
{
    MessageBox("Невозможно получить значения тайм-аутов!");
return;
```

```
}
// заполнение полей структуры pDCB
    pDCB->DCBlength=sizeof(DCB); // размер структуры
    pDCB->BaudRate=CBR_9600;  // скорость работы порта
    pDCB->ByteSize=8;              // размер передаваемых слов
    pDCB->StopBits=ONESTOPBIT;     // количество стоповых бит -
1
    pDCB->Parity=NOPARITY;         // четность отсутствует
    ct.ReadIntervalTimeout=10;     // значение тайм-аут чтения
    ct.ReadTotalTimeoutConstant=100;  // значение тайм-аут записи
    ct.ReadTotalTimeoutMultiplier=2;  // множитель тайм-аута чте-
ния
    ct.WriteTotalTimeoutConstant=100;  // множитель тайм-аута запи-
си
ct.WriteTotalTimeoutMultiplier=2;  // общий множитель
if(! SetCommState(m_hCom,pDCB))// установка новых параметров
порта
{
MessageBox("Невозможно установить параметры порта!");
return;
}
    if(!SetCommTimeouts(m_hCom,&ct)) // установка тайм-аутов
{
        MessageBox("Невозможно установить значения тайм-
аутов!");
return;
}
      }
```

В тексте программы присутствуют проверки на возможные ошибки, которые могут возникнуть по ходу её выполнения.

Для понимания тайм-аутов предположим следующее. Пусть мы считываем 50 символов из порта со скоростью 9600. При этом используется 8 бит на символ, дополнение до четности и один стоповый бит. Таким образом, на один символ в физической линии приходится 11 бит, включая стартовый. 50 символов на скорости 9600 будут приниматься 50·11/9600 = 0.0572916 секунд, или примерно 57.3 миллисекунды, при условии нулевого интервала между приемом последовательных символов. Если интервал между символами составляет примерно половину времени передачи одного символа, т.е. 0.5 миллисекунд, то время приема будет 50·11/9600+49·0.0005=0.0817916 секунд, или примерно 82 миллисекунды. Если в процессе чтения прошло более 82 миллисекунд, то мы вправе предположить, что произошла ошибка в работе внешнего устройства и прекратить считывание, избежав тем самым зависания программы. Это и есть общий тайм-аут операции чтения. Аналогично существует и общий там-аут операции записи. Формула для вычисления общего тайм-аута операции, например чтения, выглядит так:

NumOfChar · ReadTotalTimeoutMultiplier +
ReadTotalTimeoutConstant,

где NumOfChar - это число символов запрошенных для операции чтения.

Теперь небольшой пример. ReadTotalTimeoutMultiplier = 2, ReadTotalTimeoutConstant = 1, ReadIntervalTimeout = 1, считывается 250 символов. Если операция чтения завершится за 250 * 2 + 1 = 501 миллисекунду, то будет считано все сообщение. Если операция чтения не завершится за 501 миллисекунду, то она все равно будет завершена.

При этом будут возвращены символы, прием которых завершился до истечения тайм-аута операции. Остальные символы могут быть получены следующей операцией чтения. Если между началами двух последовательных символов пройдет более 1 миллисекунды, то операция чтения так же будет завершена.

4.4. Использование стандартного диалога настроек порта

Не всегда настройку порта можно жестко зашить в код программы. Внешние устройства могут позволять изменять параметры линии связи, чаще всего скорость обмена, которая зависит от длины соединительного кабеля. В таких случаях разумно предоставить пользователю самому задавать режимы обмена. Можно самому разработать соответствующий настроечный диалог, а можно воспользоваться стандартным, предоставляемым операционной системой, а точнее, производителем порта. Стандартный диалог выводится функцией CommConfigDialog, которая работает со структурой COMMCONFIG. Как и в случае со структурой DCB, заполнять структуру COMMCONFIG можно вручную или вызовом соответствующих функций.

Структура COMMCONFIG имеет следующие поля:

```
typedef struct _COMM_CONFIG {
DWORD dwSize;
WORD wVersion;
WORD wReserved;
DCB dcb;
DWORD dwProviderSubType;
DWORD dwProviderOffset;
DWORD dwProviderSize;
```

```
WCHAR wcProviderData[1];
} COMMCONFIG, *LPCOMMCONFIG;
```

Основной частью этой структуры является DCB. Остальные поля содержат вспомогательную информацию, которая, для наших целей, не представляет особого интереса (однако эта информация может быть полезной для получения дополнительных данных о порте). Познакомимся поближе с полями:

dwSize

Задает размер структуры COMMCONFIG в байтах

wVersion

Задает номер версии структуры COMMCONFIG. Должен быть равным 1.

wReserved

Зарезервировано и не используется

dcb

Блок управления устройством (DCB) для порта RS-232.

dwProviderSubType

Задает тип устройства и формат устройство-зависимого блока информации. Фактически это тип порта. Конкретные значения данного поля приведены в описании структуры **COMMPROP** выше.

dwProviderOffset

Смещение, в байтах, до устройство-зависимого блока информации от начала структуры.

dwProviderSize

Размер, в байтах, устройство-зависимого блока информации.

wcProviderData

Устройство-зависимый блок информации. Это поле может быть любого размера или вообще отсутствовать.

Несмотря на то, что нужна только DCB, приходится иметь дело со всеми полями. Заполнение данной структуры противоречивыми данными может привести к неправильной настройке порта, поэтому следует пользоваться функцией GetCommConfig:

```
BOOL GetCommConfig(
HANDLE  hCommDev,
LPCOMMCONFIG lpCC,
LPDWORD lpdwSize
);
```

Параметры функции следующие:

hCommDev

Дескриптор открытого коммуникационного порта.

lpCC

Адрес выделеного и заполненого нулями, кроме поля dwSize, блока памяти под структуру COMMCONFIG. В поле dwSize нужно занести размер структуры COMMCONFIG. После вызова функции все поля структуры будут содержать информацию о текущих параметрах порта.

lpdwSize

Адрес двойного слова, которое после воврата из функции будет содержать число фактически переданных в структуру байт.

В случае успешного завершения функция возвращает ненулевое значение. Параметры функции CommConfigDialog следующие:

lpszName

Указатель на строку с именем порта, для которого отображается диалоговое окно. К реальному имени порта эта строка не имеет никакого отношения, она просто выводится в заголовке окна.

hWnd

Дескриптор окна, которое владеет данным диалоговым окном. Должен быть передан корректный дескриптор окна-владельца или NULL, если у диалогового окна нет владельца.

lpCC

Указатель на структуру COMMCONFIG. Эта структура содержит начальные установки, используемые для отображения в диалоговом окне, и установленные пользователем изменения, при завершении диалога.

Как и большинство других функций Win32 API, функция CommConfigDialog возвращает отличное от нуля значение, в случае успешного завершения, и нуль, если возникла ошибочная ситуация.

Фактическая настройка порта выполняется функцией SetCommConfig:

```
BOOL SetCommConfig(
HANDLE hCommDev,
LPCOMMCONFIG lpCC,
DWORD dwSize
);
```

Параметры имеют то же самое значение, как и в функции GetCommConfig. На рис.2.10 показано соответствие между полями структуры DCB и полями диалогового окна настроек порта.

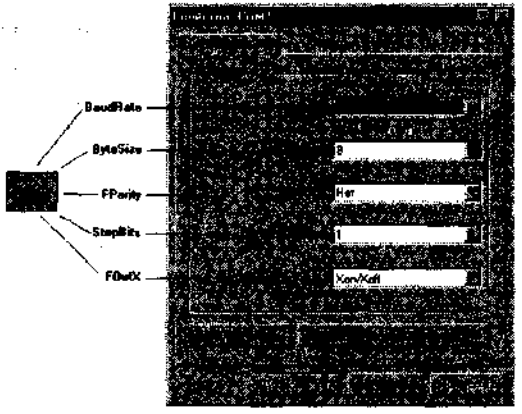


Рис.4.10. Соответствие полей структуры DCB

Пример 5. Выполнить задание предыдущего примера, но с возможностью настройки параметров порта через стандартное диалоговое окно.

Пояснение. Порядок действий:

1. Запускаем среду разработки Visual Studio .NET. Открываем проект, созданный в примере 4.

2. Добавляем на форму элемент управления кнопка и изменяем надпись на «Настроить порт» (поле «Caption» на странице свойств) (рис.4.11).

Рис.4.11. Окно программы

3. Добавляем обработчик события нажатия на кнопку «Настроить порт». Обработчик события должен выглядеть следующим образом:

```
void CCOMTestDlg::OnBnClickedButton3()
{
    DWORD sz;
    COMMCONFIG comm;    // переменная структуры
    if(!GetCommConfig(m_hCom,&comm,&sz)) // получение параметров порта
    {
        MessageBox("Невозможно получить параметры порта!");
        return;
    }
    CommConfigDialog("COM2",NULL,&comm); // вызов диалога настройки порта
    if(!SetCommConfig(m_hCom,&comm,sz)) // установка параметров порта
        MessageBox("Невозможно установить параметры порта!");
}
```

После компиляции проекта следует открыть порт. После этого нажатие на кнопку «Настроить порт» приведет к появлению диалогового окна, показанного на рис.4.12.

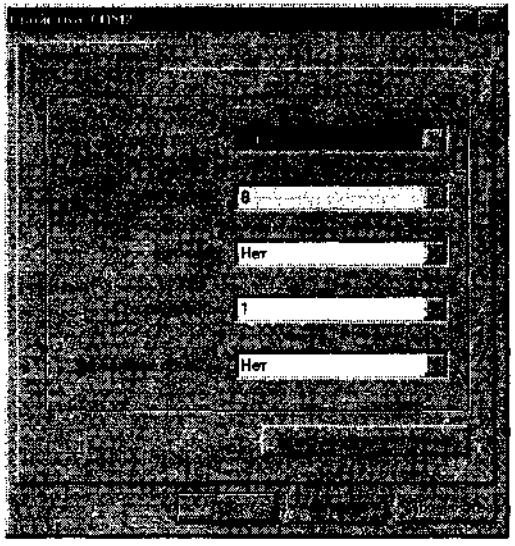


Рис.4.12. Окно настройки порта

Если нажать на кнопку «Настроить порт», предварительно не открыв порт, появится окно сообщения с надписью "Невозможно получить параметры порта!".

Функция CommConfigDialog не выполняет настройки порта. Она все лишь позволяет пользователю изменить некоторые поля в блоке DCB, содержащемся в структуре COMMCONFIG.

4.5. Прием и передача данных

Прием и передача данных выполняется функциями ReadFile и WriteFile, то есть теми же самыми, которые используются для работы с дисковыми файлами. Вот как выглядят прототипы этих функций:

```
BOOL ReadFile(
```

```
HANDLE hFile,
LPVOID lpBuffer,
DWORD nNumOfBytesToRead,
LPDWORD lpNumOfBytesRead,
LPOVERLAPPED lpOverlapped
);
BOOL WriteFile(
HANDLE hFile,
LPVOID lpBuffer,
DWORD nNumOfBytesToWrite,
LPDWORD lpNumOfBytesWritten,
LPOVERLAPPED lpOverlapped
);
```

Описание параметров функций:

hFile

Дескриптор открытого файла коммуникационного порта.

lpBuffer

Адрес буфера. Для операции записи данные из этого буфера будут передаваться в порт. Для операции чтения в этот буфер будут помещаться принятые данные.

nNumOfBytesToRead, nNumOfBytesToWrite

Число ожидаемых к приему или предназначенных к передаче байт.

nNumOfBytesRead, nNumOfBytesWritten

Число фактически принятых или переданных байт. Если принято или передано меньше данных, чем запрошено, то для дискового файла это свидетельствует об ошибке, а для коммуникационного порта совсем не обязательно. Причина в тайм-аутах.

lpOverlapped

Адрес структуры OVERLAPPED, используемой для асинхронных операций. Для синхронных операций данный параметр должен быть равным NULL.

Иногда требуется срочно передать символ, имеющий определенное специальное значение, а в очереди передатчика уже есть данные, которые нельзя терять. В этом случае можно воспользоваться функцией:

```
BOOL TransmitCommChar(
HANDLE hFile,
char cChar
);
```

Данная функция передает один (и только один) внеочередной байт в линию, не смотря на наличие данных в очереди передатчика, и перед этими данными. Первый параметр – дескриптор открытого порта. Второй параметр – символ для передачи.

Пример 6. Микропроцессорная система сбора данных, подключенная к ПК через COM-порт, выполняет контроль температуры некоторого объекта. Написать программу, которая принимает значение температуры и отображает ее в виде графика. Для получения данных необходимо отправить запрос – команда 0xBC. Периодичность запроса значения температуры – 100 мс. Скорость передачи составляет 9600 бит/с. Формат посылки – 8 бит данных в информационном блоке, 1 стоповый бит, проверка на чётность отсутствует.

Пояснение. Порядок действий:

1. Запускаем среду разработки Visual Studio .NET. Открываем проект, созданный в примере 5, который берём за основу.

2. Изменяем вид диалогового окна. Выносим на форму элемент управления «Static Text» корректируем его ID на IDC_STATIC3. Также очищаем поле «Caption» чтобы не отображался текст. Затем необходимо растянуть этот элемент управления так, чтобы по ширине он занял всю форму, а по высоте – приблизительно 70%. В поле «Border» на странице свойств установить значение «True». В области, которую занял элемент управления «Static Text» будет отображаться график изменения температуры. Также необходимо привязать к этому элементу управления переменную с именем m_graph. На рис.4.13 приводится вид диалогового окна.

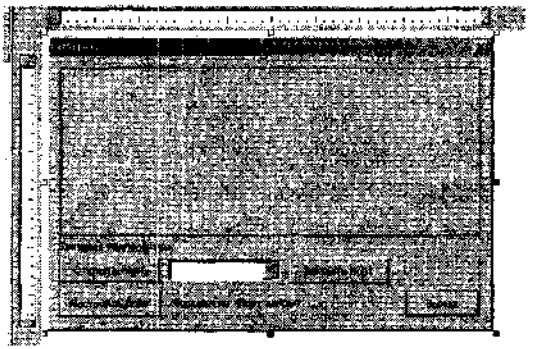


Рис.4.13. Окно программы

Кроме того, необходимо добавить элемент управления «Static Text» для отображения текущей температуры. Изменить ID на IDC_STATIC4. К этому элементу управления привязать переменную с именем m_curtemp.

3. Для хранения полученных данных о температуре используется переменная m_temp и имеющая тип CByteArray. Эта переменная – член класса CCOMTestDlg. Для того, чтобы добавить переменную, в окне «Class View» выделить класс «CCOMTestDlg», вызвать контекст-

ное меню и из подменю «Add» выбрать команду «Add Variable». Появившееся диалоговое окно заполнить так, как показано на рис.4.14.

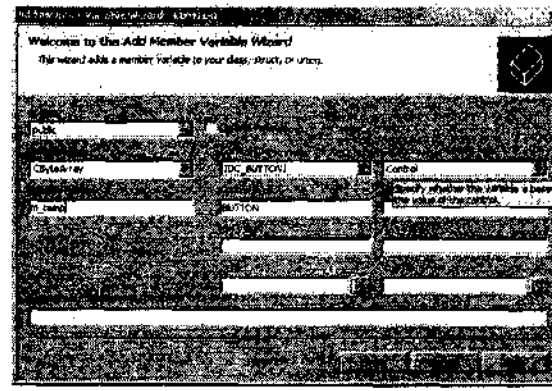


Рис.4.14. Окно добавление переменной

4. Добавляем обработчик события таймера. Для этого из страницы свойств (Properties) вызвать окно «Messages» и напротив позиции «WM_TIMER» выбрать «OnTimer» (рис.4.15).

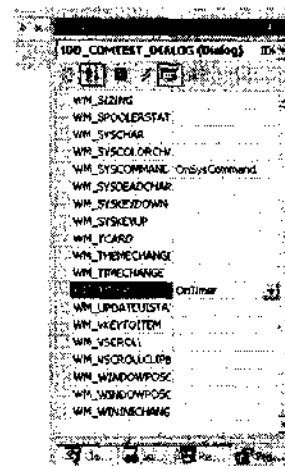


Рис.4.15 Создание обработчика события таймера

5. Добавляем в функцию - обработчик события таймера код, как показано ниже:

```
void CCOMTestDlg::OnTimer(UINT nIDEvent)
{
    TransmitCommChar(m_hCom,(char)0xBC); // передача значения 0xBC
    BYTE rdata;
    DWORD size;
    ReadFile(m_hCom,&rdata,1,&size,NULL);// чтение 1 байта с порта
    if(size!=1) // если 1 байт не считан
    {
        // выдаем сообщение
        SetWindowText("Отсутствуют данные о температуре!");
        return;     // выход из функции
    }
    m_temp.Add(rdata); // добавляем данные в массив
    CString s;
    // форматирование строки
    s.Format("Текущая температура: %d град.",rdata);
    m_currtemp.SetWindowText(s);
    DrawGraph();   // обновление графика на экране
    CDialog::OnTimer(nIDEvent);
}
```

В этой функции запрос на получение данных о температуре отправляется с помощью функции TransmitCommChar. Затем происходит считывание с порта 1 байта данных. Если данные были получены,

значение переменной size (количество считанных байт) будет равно 1. Если же это значение отличается от 1, выдается сообщение о том, что данные о температуре отсутствуют. Благодаря настройкам тайм-аутов, программа не «зависнет» из-за отсутствия принятых данных.

6. Реализуем функцию void DrawGraph(void). Для этого ее необходимо сделать членом класса CCOMTestDlg. Реализация функции имеет следующий вид:

```
void CCOMTestDlg::DrawGraph(void)
{
    CClientDC dc(&m_graph);// создание контекста для рисования
    CPen p(PS_SOLID,2,RGB(200,0,0));// создание пера красного цвета
    CPen pp(PS_DOT,1,RGB(0,0,0));
    CRect rect;
    CRgn rgn;
    CPen* op;
    m_graph.RedrawWindow();      // перерисовка окна рисования графика
    m_graph.GetClientRect(&rect);
    // получение размеров области рисоания
    rgn.CreateRectRgn(0,0,rect.Width(),rect.Height());
// создание региона для ограничения области рисования
    op=dc.SelectObject(&pp);  // установка пера
    for(int x=20;x<rect.Width();x=x+20)  // рисование сетки
    {
        dc.MoveTo(x,0);      // перемещение начальной позиции
```

```
                dc.LineTo(x,rect.Height());// рисование линии
            }
            dc.SetTextColor(RGB(50,50,50));      // установка цвета тек-
ста
            dc.SetBkColor(RGB(210,210,210));   // установка цвета фона
            for(int y=20;y<rect.Height();y=y+20)
            {
                dc.MoveTo(0,rect.Height()-y);
                dc.LineTo(rect.Width(),rect.Height()-y);
                CString s;
                s.Format("%d",y);     // форматирование строки
                dc.TextOut(rect.Width()-30,rect.Height()-y,s);
        // вывод текстовых меток
            }
            dc.SelectObject(&p); // выбор красного пера в контекст
            dc.SelectClipRgn(&rgn);   // установка региона
            if(rect.Width()<=m_temp.GetCount()) m_temp.RemoveAll();
            // если значение элементов в массиве превышает размер об-
ласти
        // рисования, то удаляются все элементы массива
            for(int i=1;i<m_temp.GetCount();i++)
            {
                // рисование графика температуры
                dc.MoveTo(i,rect.Height()-m_temp.GetAt(i));
                dc.LineTo(i-1,rect.Height()-m_temp.GetAt(i-1));
            }
            dc.SelectObject(op);
        }
```

7. Корректируем обработчик нажатия на кнопку «Открыть порт». Добавляем в этот обработчик строку

```
SetTimer(1,200,NULL);
```

При нажатии на эту кнопку, в случае успешного открытия порта, запускается таймер, который будет срабатывать через каждые 200 мс. При этом будет вызываться функция OnTimer, в которой реализован код по запросу данных о температуре и их отображению.

8. Корректируем обработчик события нажатия на кнопку «Закрыть порт». Поскольку при закрытии порта данные невозможно будет считать, необходимо остановить таймер. Для этого в обработчик добавляем следующую строчку:

```
KillTimer(1);
```

9. Откомпилировав проект, появится диалоговое окно, как показано на рис.4.16 (при условии, что данные принимаются через порт).

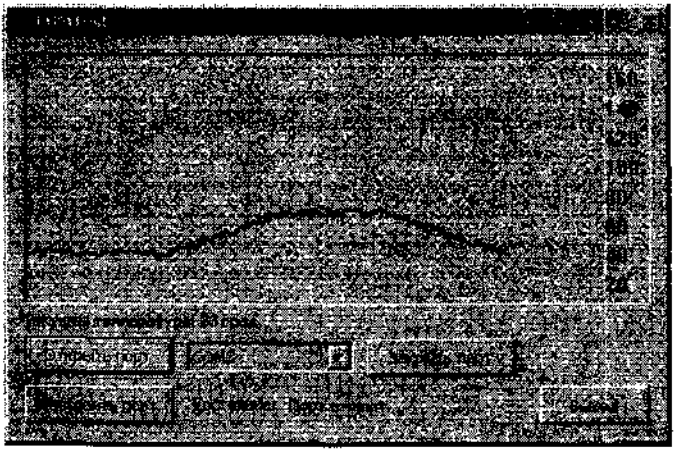


Рис.4.16. Окно программы

4.6. Использование потоков

В предыдущем примере функция ReadFile завершала свое выполнение даже если данные не поступали в порт. Путем анализа количества считанных байт можно выяснить причину завершения работы функции (приняты данные, истек тайм-аут). Существует ряд задач, в которых поток данных не является равномерным. Например, пожарная сигнализация. При нормальной обстановке в компьютер данные могут не поступать, а обрабатываться на локальных объектах. В случае пожара на главный компьютер передаются данные о месте возникновения пожара, а также другая информация. Для реализации такой задачи использование циклического опроса порта будет не эффективным. В этом случае более выгодно использовать потоки.

Каждый поток начинает выполнение с некоторой входной функции. Для создания вторичного (рабочего, фонового) потока, в нем тоже должна быть входная функция, прототип которой которая выглядит так:

```
DWORD WINAPI ThreadFunction(PVOID pvParam){
...
return(0);
}
```

Функция потока может выполнять любые задачи. В нашем случае это ожидание данных с порта и передача управления главной программе. События, связанные с портом, отслеживаются с помощью функции:

```
BOOL SetCommMask(
 HANDLE hFile,
 DWORD dwEvtMask
 );
```

В качестве первого параметра передается дескриптор открытого порта. Для слежения за возникновением определенных событий, связанных с портом, необходимо установить маску отслеживаемых событий, которая задается вторым параметром. Можно указывать любую комбинацию следующих значений:

EV_BREAK - Состояние разрыва приемной линии

EV_CTS - Изменение состояния линии CTS

EV_DSR - Изменение состояния линии DSR

EV_ERR - Ошибка четности

EV_RING - Входящий звонок на модем (сигнал на линии RI порта)

EV_RLSD - Изменение состояния линии RLSD (DCD)

EV_RXCHAR - Символ принят и помещен в приемный буфер

EV_RXFLAG - Принят символ заданный полем EvtChar структуры DCB использованной для настройки режимов работы порта

EV_TXEMPTY - Из буфера передачи передан последний символ

Если dwEvtMask равно нулю, то отслеживание событий запрещается.

Разумеется, всегда можно получить текущую маску отслеживаемых событий с помощью функции

```
BOOL GetCommMask(
HANDLE hFile,
LPDWORD lpEvtMask
);
```

Вторым параметром задается адрес переменной принимающей значение текущей установленной маски отслеживаемых событий.

Когда маска отслеживаемых событий задана, можно приостановить выполнение программы до наступления события. При этом про-

грамма не будет занимать процессор. Это выполняется вызовом функции

```
BOOL WaitCommEvent(
HANDLE hFile,
LPDWORD lpEvtMask,
LPOVERLAPPED lpOverlapped,
);
```

где hFile – дескриптор открытого порта. В переменной, адресуемой вторым параметром lpEvtMask, в единичное состояние установятся только те биты, которые соответствуют реально произошедшим событиям.

Адрес структуры OVERLAPPED (третий параметр функции – lpOverlapped) требуется для асинхронного ожидания. Для синхронных операций этот параметр должен быть NULL.

Функция WaitCommEvent приостанавливает выполнение потока до тех пор, пока не возникнет какое-либо из событий, заданных маской. Для того, чтобы сообщить основной программе, что возникло событие, связанное с портом, можно использовать механизм передачи сообщений, созданных пользователем. Для этого используются следующая функция:

```
LRESULT SendMessage(HWND hWnd, UINT Msg, WPARAM wParam,
LPARAM lParam );
```

Эта функция WinAPI посылает определенное сообщение окну или окнам. Функция вызывает оконную процедуру (WndProc) определенного окна и не выходит из процедуры, пока та не обработает сообщение. Другими словами, выполнение программы не будет продолжено, пока сообщение не будет обработано.

hWnd - дескриптор окна, которое получает сообщение;

Msg - определенное сообщение;

wParam - содержит определенную информацию о сообщении (зависит от сообщения);

lParam - содержит определенную информацию о сообщении (зависит от сообщения).

Возвращаемое значение функции зависит от типа посылаемого сообщения.

Если диалоговое окно получает сообщение, вызывается определенная функция. Какая именно функция будет вызвана, указывается в карте сообщений (Message Map). Прототип функции – обработчика пользовательского сообщения имеет вид:

LRESULT OnMyMessage(WPARAM wp, LPARAM lp);

Каждое сообщение имеет свой номер. Некоторые из сообщений заняты операционной системой, поэтому их нельзя использовать. Сообщения с номерами больше чем WM_USER не задействованы, поэтому их можно использовать.

Пример 7. Написать программу обмена данными между двумя компьютерами по COM-порту (чат). Скорость обмена данными – 9600 бит/с.

Пояснение. Порядок действий:

1. Запускаем среду разработки Visual Studio .NET. Открываем проект, созданный в примере 4.

2. Добавляем на форму следующие элементы управления: Edit Control, Button, List Box, а также Static Text, и располагаем их на диалоговом окне так, как показано на рис.4.17.

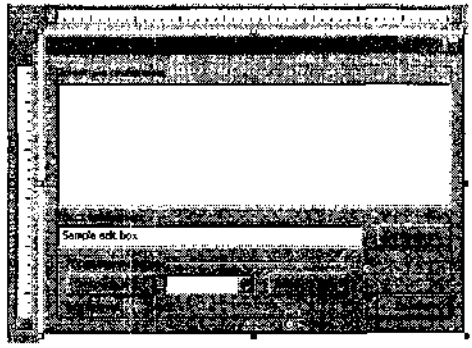


Рис.4.17. Спроектированное диалоговое окно

В элементе управления Edit Control пользователь будет вводить текстовое сообщение. При нажатии на кнопку «Отправить» введенное сообщение будет передано в порт. Принимаемые сообщения будут отображаться в элементе управления «Список» (List Box). Для этого элемента управления на странице свойств в поле «Sort» установить значение «False». В противном случае строки, добавляемые в элемент управления, будут отсортированы по алфавиту, и порядок приема сообщений не будет сохраняться.

3. Связываем с элементами управления переменные. К «List Box» привязывается переменная с именем m_recmess, а к текстовому полю «Edit Control» переменная с именем m_sendmess. Для выполнения этих действий необходимо по очереди выделить каждый элемент управления и вызвать контекстное меню, из которого выбрать команду «Add Variable» (рис.4.18).

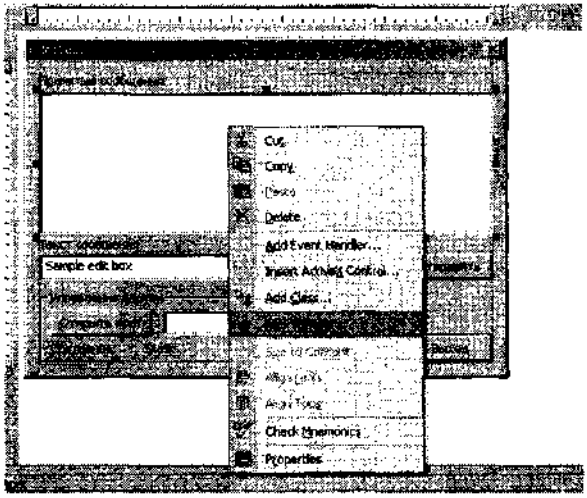


Рис.4.18. Добавление переменной

4. Создаем глобальную переменную hCom. В этой переменной будет храниться дескриптор открытого порта. Поскольку переменная глобальная, доступ к ней возможен из функции потока. Также необходимо определить пользовательское сообщение. Для этого в начале файла COMTestDlg.cpp, после секции #include, добавляем строки:

```
volatile HANDLE hCom;  // глобальная переменная – дескриптор
#define WM_MESSRECIEVED WM_USER+1  // пользовательское сообщение
```

Для остановки работы потока необходимо знать указатель на него. В библиотеке MFC для работы с потоками используется класс CWinThread. В класс CCOMTestDlg необходимо добавить переменную-член класса:

```
CWinThread* m_thread;
```

При создании потока в этой переменной будет храниться указатель на него.

5. Добавляем функцию рабочего потока, в котором будет ожидаться прием сообщения. Эта функция не является членом класса диалогового окна CCOMTestDlg. Ее описание следует расположить в файле COMTestDlg.cpp, перед началом описания обработчиков событий нажатия на кнопку. Функция выглядит следующим образом:

```
UINT ThreadFunction(LPVOID param)
{
    DWORD cm;
    SetCommMask(hCom, EV_RXCHAR);    // установка маски событий порта
    PurgeComm(hCom, PURGE_RXCLEAR | PURGE_TXCLEAR);
// очистка очередей приема и передачи
    WaitCommEvent(hCom, &cm, NULL); // ожидание события
    ::SendMessage((HWND)param, WM_MESSRECIEVED, 0, 0);
    // посылка сообщения главному окну программы
    return 0;
}
```

6. Изменяем обработчик события нажатия на кнопку «Открыть порт». В случае успешного открытия порта, глобальной переменной hCom необходимо присвоить дескриптор открытого порта. Кроме того, необходимо запустить поток. Обработчик события нажатия на кнопку «Открыть порт» имеет вид:

```
void CCOMTestDlg::OnBnClickedButton1()
{
CString sp;
m_port.GetLBText(m_port.GetCurSel(),sp);
```

```
m_hCom=CreateFile(sp,GENERIC_READ | GE-
NERIC_WRITE,0,NULL,OPEN_EXISTING,0,NULL);
// открываем последовательный порт
if(m_hCom==INVALID_HANDLE_VALUE)
{
m_status.SetWindowText("Невозможно открыть порт!");
// если невозможно открыть порт, выдаем сообщение
return;
}
else    // если порт открыт
m_status.SetWindowText("Порт открыт.");
// сообщаем, что порт открыт
m_bOpen.EnableWindow(0);// изменение состояний
m_bClose.EnableWindow(1);       // элементов управления
m_port.EnableWindow(0);
}
DCB *pDCB;      // указатель на структуру DCB
COMMTIMEOUTS ct; // переменная для настройки тайм-аутов
pDCB=new DCB;
memset(pDCB,0,sizeof(DCB));    // обнуление полей структу-
ры
if(!GetCommState(m_hCom,pDCB)) // получение состояния
порта
{
MessageBox("Невозможно получить параметры порта!");
return;
}
if(!GetCommTimeouts(m_hCom, &ct)) // получение тайм-аутов
```

```
{
MessageBox("Невозможно получить значения тайм-аутов!");
return;
}
pDCB->DCBlength=sizeof(DCB); // размер структуры
pDCB->BaudRate=CBR_9600;     // скорость передачи данных
pDCB->ByteSize=8;            // размер слова
pDCB->StopBits=ONESTOPBIT; // количество стоповых бит
pDCB->Parity=NOPARITY;       // четность отсутствует
ct.ReadIntervalTimeout=10;  // значение тайм-аута чтения
ct.ReadTotalTimeoutConstant=300;
ct.ReadTotalTimeoutMultiplier=2;
ct.WriteTotalTimeoutConstant=300;     // значение тайм-аута
записи
ct.WriteTotalTimeoutMultiplier=2;
if(! SetCommState(m_hCom,pDCB)) // установка параметров
порта
{
MessageBox("Невозможно установить параметры порта!");
return;
}
if(!SetCommTimeouts(m_hCom,&ct))   // установка тайм-аутов
{
MessageBox("Невозможно установить значения тайм-аутов!");
return;
}
hCom=m_hCom;
// присвоение глобальной переменной дескриптора порта
```

```
m_thread=AfxBeginThread(ThreadFunction,        m_hWnd,
THREAD_PRIORITY_NORMAL); // запуск рабочего потока
}
```

7. Изменяем обработчик события нажатия на кнопку «Закрыть порт». При нажатии на кнопку необходимо остановить выполнение потока. Обработчик события нажатия на кнопку «Закрыть порт» имеет вид:

```
void CCOMTestDlg::OnBnClickedButton2()
{
::TerminateThread(m_thread->m_hThread,0);    // остановка потока
CloseHandle(m_hCom);    // закрытие порта
m_status.SetWindowText("Порт закрыт.");
m_port.EnableWindow(1);
// изменение состояний элементов управления
m_port.EnableWindow(1);
m_bOpen.EnableWindow(1);
m_bClose.EnableWindow(0);
}
```

8. Создаем функцию – обработчик сообщения WM_MESSRECIEVED, которое передается главному окну программы из рабочего потока при приеме данных. Эта функция является членом класса диалогового окна. В окне «Class View» следует выделить класс CCOMTestDlg и из контекстного меню выбрать команду «Add Function». Появится диалоговое окно, в котором необходимо указать название функции, а также тип и имена параметров, передаваемых в функцию. Вид этого окна представлен на рис.4.19.

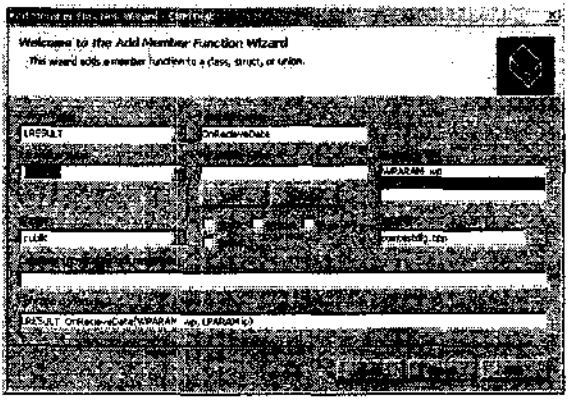


Рис.4.19. Окно добавления функции в класс

Также необходимо указать, что при появлении сообщения WM_MESSRECIEVED будет вызвана функция OnRecieveData. Для этого карту сообщений нужно изменить следующим образом:

```
BEGIN_MESSAGE_MAP(CCOMTestDlg, CDialog)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    //}}AFX_MSG_MAP
    ON_BN_CLICKED(IDC_BUTTON1, OnBnClickedButton1)
    ON_BN_CLICKED(IDC_BUTTON2, OnBnClickedButton2)
    ON_MESSAGE(WM_MESSRECIEVED, OnRecieveData)
END_MESSAGE_MAP()
```

Описание карты сообщений находится в начале файла COMTestDlg.cpp.

9. Добавляем обработчик события нажатия на кнопку «Отправить». При нажатии на эту кнопку строка, введенная в текстовом поле, должна быть передана в порт. Реализация функции приводится ниже:

```
void CCOMTestDlg::OnBnClickedButton3()
{
    if(m_hCom==NULL)// если порт закрыт
```

```
        {
        MessageBox("Порт закрыт!"); // выдается сообщение
        return;
        }
        CString text;
        char *data;
        DWORD size, sz;
        m_sendmess.GetWindowText(text);
    // получение введенного пользователем текста
        size=text.GetLength()+1;// определение длины введенной
строки
        data=new char[size];  // выделение буфера для строки
        strcpy(data,text.GetBuffer(size));       // копирование данных
        text.ReleaseBuffer();
        ::TerminateThread(m_thread->m_hThread,0); // остановка по-
тока
        WriteFile(m_hCom,data,size,&sz,NULL);   // передача данных
    m_thread=AfxBeginThread(ThreadFunction, m_hWnd,
THREAD_PRIORITY_NORMAL); // запуск потока
        delete data;      // освобождение выделенной под буфер па-
мяти
    }
```

10. Реализуем функцию, которая выполняется при приеме данных в порт (OnRecieveData). Эта функция имеет следующий вид:

```
    LRESULT CCOMTestDlg::OnRecieveData(WPARAM wp,
LPARAM lp)
    {
        char rdata[256]; // буфер для данных
        DWORD size;
        ReadFile(m_hCom,rdata,256,&size,NULL); // считывание
данных
```

```
    m_thread=AfxBeginThread(ThreadFunction, m_hWnd,
THREAD_PRIORITY_NORMAL); // запуск потока
        if(size<=0)       return 0;.  // если данных нет, выход из
функции
        rdata[size]='\0';
        CString text(rdata);    // создание строки
        m_recmess.AddString(text);       // добавление строки в List
Box
        return 0;
    }
```

Теперь можно компилировать проект. При отсутствии синтаксических ошибок, на экране появится диалоговое окно, показанное на рис.4.20. С помощью нуль-модемного кабеля можно соединить два компьютера и проверить работу программы. Если же нет возможности использовать два компьютера, то можно поставить перемычку между 2 и 3 выводами порта (для 9-штырькового разъема). В этом случае сообщение, которое было отправлено, будет отображаться в списке.

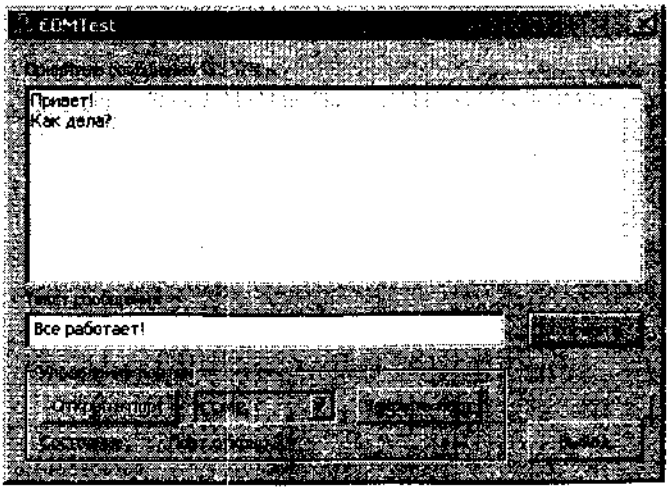


Рис.4.20. Окно программы

5. ШИНА USB

5.1. Аппаратная организация шины

USB (Universal Serial Bus – универсальная последовательная шина) является промышленным стандартом расширения архитектуры PC, ориентированным на интеграцию с телефонией и устройствами бытовой электроники. Первоначально (в версиях 1.0 и 1.1) шина обеспечивала две скорости передачи информации: полную скорость FS (full speed) - 12 Мбит/с и низкую скорость LS (Low Speed) - 1,5 Мбит/с. В версии 2.0 определена еще и высокая скорость HS (High Speed) - 480 Мбит/с, которая позволяет существенно расширить круг устройств, подключаемых к шине. В одной и той же системе могут присутствовать и одновременно работать устройства со всеми тремя скоростями. Шина позволяет соединять устройства, удаленные от компьютера на расстояние до 25 м (с использованием промежуточных хабов).

USB обеспечивает обмен данными между хост-компьютером и множеством периферийных устройств. Согласно спецификации USB, устройства могут являться хабами, функциями или их комбинацией. Хаб (hub) только обеспечивает дополнительные точки подключения устройств к шине. Устройство USB должно иметь интерфейс USB, обеспечивающий полную поддержку протокола USB, выполнение стандартных операций (конфигурирование и сброс) и предоставление информации, описывающей устройство. Работой всей системы USB управляет хост-контроллер, являющийся программно-аппаратной подсистемой хост-компьютера. Шина позволяет подключать, конфигури-

ровать, использовать и отключать устройства во время работы хоста и самих устройств.

Для подключения устройств и хабов к компьютеру существуют два типа разъемов:

USB разъём Тип А USB разъем тип В

Рис.5.1. Типы USB-разъемов

Разъем А всегда символизирует исходящий поток данных. Типичное месторасположение разъема А это хосты и концентраторы. Их всегда можно увидеть на материнских платах. Разъем В всегда символизирует нисходящий поток данных. Их можно увидеть на любом периферийном оборудовании подключаемом к USB шине.

Кабель USB содержит одну экранированную витую пару с импедансом 90 Ом для сигнальных цепей и одну неэкранированную для подачи питания (+5 В), допустимая длина сегмента - до 5 м. Для низкой скорости может использоваться не витой неэкранированный кабель длиной до 3 м. Система кабелей и коннекторов USB не дает возможности ошибиться при подключении устройств. Для распознавания разъема USB на корпусе устройства ставится стандартное символическое обозначение. Гнезда типа «А» устанавливаются только на нисходящих портах хабов, вилки типа «А» - на шнурах периферийных устройств или восходящих портов хабов. Гнезда и вилки типа «В» используются только для шнуров, отсоединяемых от периферийных устройств и восходящих портов хабов (от «мелких» устройств - мышей, клавиатур и т. п. кабели как правило, не отсоединяются). Хабы и устройства обеспечивают возможность «горячего» подключения и от-

ключения. Для этого разъемы обеспечивают более раннее соединение и позднее отсоединение питающих цепей по отношению к сигнальным и предусмотрен протокол сигнализации подключения и отключения устройств. Назначение выводов разъемов USB иллюстрирует табл.5.1.

Таблица 5.1. **Назначение выводов разъемов USB**

Контакт	Цепь	Цвет
1	Vbus	Красный
2	D-	Белый
3	D+	Зеленый
4	GND	Черный

Стандартная цветовая гамма проводников внутри USB кабеля облегчает идентификацию проводов.

Шина USB является хост-центрической: единственным ведущим устройством, которое управляет обменом, является хост-компьютер, а все присоединенные к ней периферийные устройства - исключительно ведомые. В этом она отличается от шины FireWire, где все устройства равноправны. Физическая топология шины USB - многоярусная звезда. Ее вершиной является хост-контроллер, объединенный с корневым хабом (root hub), как правило, двухпортовым. Хаб является устройством-разветвителем. Кроме того, он может являться источником питания для подключенных к нему устройств. К каждому порту хаба может непосредственно подключаться периферийное устройство или промежуточный хаб шина допускает до 5 уровней каскадирования хабов (не считая корневого). Поскольку комбинированные устройства внутри себя содержат хаб, их подключение к хабу 6-го яруса уже недопустимо. Каждый промежуточный хаб имеет несколько нисходящих (downstream) портов для подключения периферийных устройств (или нижележащих хабов) и один восходящий (upstream) порт для под-

ключения к корневому хабу или нисходящему порту вышестоящего хаба. Логическая топология USB - просто звезда: для хост-контроллера хабы создают иллюзию непосредственного подключения каждого устройства.

В шине используется дифференциальный способ передачи сигналов D+ и D- по двум проводам. Сигналы синхронизации и данные кодируются по методу NRZI. В этой кодировке логическая 1 представляет собой смену уровня на противоположный (фронт) на протяжении битового интервала. Для низкоскоростных и полноскоростных устройств дифференциальная 1 передается путем подтяжки линии D– к напряжению более 2,8 В а линии D– к напряжению менее 0,3 В. При этом линии D+ и D– терминированы на стороне хоста (нисходящего потока) резисторами 15кОм, подключенными к земле. Дифференциальный ноль передается путем подтяжки линии D– к напряжению менее 0,3 В, а линии D+ к напряжению более 2,8 В. Приемник определяет дифференциальную единицу только в том случае, когда напряжение на линии D+ больше на 200 мВ, чем на линии D–, а дифференциальный 0, когда напряжение на линии D+ меньше на 200мВ чем на линии D-.

Скорость, используемая устройством, подключенным к конкретному порту, определяется хабом по уровням сигналов на линиях D+ и D–, смещаемых нагрузочными резисторами приемопередатчиков: устройства с низкой скоростью «подтягивают» к высокому уровню линию D–, с полной – D+. Передача по двум проводам в USB не ограничивается дифференциальными сигналами. Кроме дифференциального приемника каждое устройство имеет линейные приемники сигналов D+ и D–, а передатчики этих линий управляются индивидуально. Это позволяет различать более двух состояний линии, используемых для организации аппаратного интерфейса. Скорость передачи данных вы-

бирается разработчиком периферийного устройства в соответствии с потребностями этого устройства.

Каждое устройство на шине USB (их может быть до 127) при подключении автоматически получает свой уникальный адрес. Логически устройство представляет собой набор независимых конечных точек (endpoint), с которыми хост-контроллер (и клиентское ПО) обменивается информацией. Каждая конечная точка имеет свой номер и описывается следующими параметрами:

- требуемая частота доступа к шине и допустимые задержки обслуживания;
- требуемая полоса пропускания канала;
- требования к обработке ошибок;
- максимальные размеры передаваемых и принимаемых пакетов;
- тип передачи;
- направление передачи (для передач массивов и изохронного обмена).

Каждое устройство обязательно имеет конечную точку с номером 0, используемую для инициализации, общего управления и опроса его состояния. Эта точка всегда оказывается сконфигурированной при включении питания и подключении устройства к шине. Она поддерживает передачи типа «управление» (см. ниже).

Кроме нулевой точки устройства-функции могут иметь дополнительные точки, реализующие полезный обмен данными. Низкоскоростные устройства могут иметь до двух дополнительных точек, полноскоростные - до 15 точек ввода и 15 точек вывода (протокольное ограничение). Дополнительные точки (а именно они и предоставляют по-

лезные для пользователя функции) не могут быть использованы до их конфигурирования (установления согласованного с ними канала).

Каналом (pipe) в USB называется модель передачи данных между хост-контроллером и конечной точкой устройства. Имеются два типа каналов: потоки и сообщения. Поток (stream) доставляет данные от одного конца канала к другому, он всегда однонаправленный. Один и тот же номер конечной точки может использоваться для двух поточных каналов - ввода и вывода. Поток может реализовывать следующие типы обмена: передача массивов, изохронный и прерывания. Сообщения (message) имеют формат, определенный спецификацией USB. Хост посылает запрос к конечной точке, после которого передается (принимается) пакет сообщения, за которым следует пакет с информацией состояния конечной точки. Последующее сообщение нормально не может быть послано до обработки предыдущего, но при отработке ошибок возможен сброс не обслуженных сообщений. Двусторонний обмен сообщениями адресуется к одной и той же конечной точке.

С каналами связаны характеристики, соответствующие конечной точке (полоса пропускания, тип сервиса, размер буфера и т. п.). Каналы организуются при конфигурировании устройств USB. Для каждого включенного устройства существует канал сообщений (Control Pipe 0), по которому передается информация конфигурирования, управления и состояния.

Все обмены (транзакции) с устройствами USB состоят из двух-трех пакетов. Каждая транзакция планируется и начинается по инициативе контроллера, который посылает пакет-маркер (token packet). Он описывает тип и направление передачи, адрес устройства USB и номер конечной точки. В каждой транзакции возможен обмен только между адресуемым устройством (его конечной точкой) и хостом. Ад-

ресуемое маркером устройство распознает свой адрес и готовится к обмену. Источник данных (определенный маркером) передает пакет данных (или уведомление об отсутствии данных, предназначенных для передачи). После успешного приема пакета приемник данных посылает пакет квитирования (handshake packet).

Хост-контроллер организует обмены с устройствами согласно своему плану распределения ресурсов. Контроллер циклически (с периодом 1,0 ± 0,0005 мс) формирует кадры (frames), в которые укладываются все запланированные транзакции. Каждый кадр начинается с посылки маркера SOF (Start Of Frame), который является синхронизирующим сигналом для всех устройств, включая хабы. В конце каждого кадра выделяется интервал времени EOF (End Of Frame), на время которого хабы запрещают передачу по направлению к контроллеру. В режиме HS пакеты SOF передаются в начале каждого микрокадра (период 125 ± 0,0625 мкс). Хост планирует загрузку кадров так, чтобы в них всегда находилось место для транзакций управления и прерывания. Свободное время кадров может заполняться передачами массивов (bulk transfers). В каждом (микрокадре) может быть выполнено несколько транзакций, их допустимое число зависит от длины поля данных каждой из них.

Для обнаружения ошибок передачи каждый пакет имеет контрольные поля CRC-кодов, позволяющие обнаруживать все одиночные и двойные битовые ошибки. Аппаратные средства обнаруживают ошибки передачи, а контроллер автоматически производит трехкратную попытку передачи. Если повторы безуспешны, сообщение об ошибке передается клиентскому ПО.

Все подробности организации транзакций изолируются от клиентского ПО контроллером USB и его системным программным обеспечением.

У каждой шины USB должен быть один (и только один) хост-компьютер с контроллером USB. Хост делится на три основных уровня.

- Интерфейс шины USB обеспечивает физический интерфейс и протокол шины. Интерфейс шины реализуется хост-контроллером, имеющим встроенный корневой хаб, обеспечивающий точки физического подключения к шине (гнезда USB типа «А»). Хост-контроллер отвечает за генерацию (микро) кадров. На аппаратном уровне хост-контроллер обменивается информацией с основной памятью компьютера путем прямого управления шиной (bus-mastering) с целью минимизации нагрузки на центральный процессор.
- Система USB, используя хост-контроллер(ы), транслирует клиентское видение обмена данными с устройствами в транзакции, выполняемые с реальными устройствами шины. Система отвечает и за распределение ресурсов USB - полосы пропускания и мощности источников питания (для устройств, питающихся от шины). Система состоит из трех основных частей.
- Драйвер хост-контроллера - HCD (Host Controller Driver) - модуль, привязанный к конкретной модели контроллера, обеспечивающий абстрагирование драйвера USB и позволяющий в одну систему включать несколько разнотипных контроллеров.
- Драйвер USB - USBD (USB Driver) - обеспечивает основной интерфейс (USBDI) между клиентами и устройствами USB. Интерфейс HCDI (Host Controller Driver Interface) между USBD и HCD спецификацией USB не регламентируется. Он определяет-

ся разработчиками ОС и должен поддерживаться разработчиками хост-контроллере в, желающих иметь поддержку своих изделий конкретными ОС. Клиенты не могут пользоваться интерфейсом HCDI, для них предназначен интерфейс USBDI, USBD обеспечивает механизм обмена в виде пакетов IRP (I/O Request Packet - пакет запроса ввода-вывода), состоящих из запросов на транспортировку данных по заданному каналу. Кроме того, USBD отвечает за некоторое абстрактное представление устройства USB клиенту, которое позволяет выполнять конфигурирование и управление состоянием устройств (включая и стандартное управление через конечную точку «О»). Реализация интерфейса USBDI определяется операционной системой, в спецификации USB излагаются только общие идеи.

- Программное обеспечение хоста реализует функции, необходимые для функционирования системы USB в целом: обнаружение подключения и отключения устройств и выполнение соответствующих действий по этим событиям (загрузки требуемых драйверов), нумерацию устройств, распределение полосы пропускания и потребляемой мощности и т. п.
- Клиенты USB - программные элементы (приложения или системные компоненты), взаимодействующие с устройствами USB. Клиенты могут взаимодействовать с любыми устройствами (их конечными точками), подключенными к системе USB. Однако система USB изолирует клиентов от непосредственного обмена с какими-либо портами (в пространстве ввода-вывода) или ячейками памяти, представляющими интерфейсную часть контроллера USB.

- В совокупности уровни хоста имеют следующие возможности:
- обнаружение фактов подключения и отсоединения устройств USB;
- манипулирование потоками управления между устройствами и хостом;
- манипулирование потоками данных;
- сбор статистики активности и состояний устройств;
- управление электрическим интерфейсом между хост-контроллером и устройствами USB, включая управление электропитанием.

Хост-контроллер является аппаратным посредником между устройствами USB и хостом. Программная часть хоста в полном объеме реализуется операционной системой. До загрузки ОС может функционировать лишь усеченная часть ПО USB, поддерживающая только устройства, требующиеся для загрузки. Так в BIOS современных системных плат имеется поддержка клавиатуры USB, реализующая функции сервиса Int 10h. При загрузке системы USB эта «дозагрузочная» поддержка игнорируется - система начинает работу с контроллером «с чистого листа», то есть со сброса и определения всех подключенных устройств. По окончании работы ОС передача состояния USB «дозагрузочной» поддержке не предусматривается, так что для нее это событие тоже может рассматриваться как первоначальное включение. В спецификации PC'2001 выдвигается требование к поддержке USB со стороны BIOS в такой мере, чтобы обеспечивалась загрузка ОС с устройств USB.

USB поддерживает динамическое подключение и отключение устройств. Нумерация устройств шины является постоянным процессом, отслеживающим изменения физической топологии.

Все устройства подключаются через порты хабов. Хабы определяют факты подключения и отключения устройств к своим портам и сообщают состояние портов при запросе от контроллера. Хост разрешает работу порта и адресуется к устройству через канал управления, используя нулевой адрес - USB Default Address. При начальном подключении или после сброса все устройства адресуются именно так.

Хост определяет, является новое подключенное устройство хабом или функцией, и назначает ему уникальный адрес USB. Хост создает канал управления (control pipe) с этим устройством, используя назначенный адрес и нулевой номер точки назначения.

Если новое устройство является хабом, хост определяет подключенные к нему устройства, назначает им адреса и устанавливает каналы. Если новое устройство является функцией, уведомление о подключении передается диспетчером USB.

Когда устройство отключается, хаб автоматически запрещает соответствующий порт и сообщает об отключении контроллеру, который удаляет сведения о данном устройстве из всех структур данных. Если отключается хаб, процесс удаления выполняется для всех подключенных к нему устройств. Если отключается функция, уведомление посылается заинтересованному ПО.

Преобразователи интерфейсов позволяют через порт USB, имеющийся теперь практически на всех компьютерах, подключать устройства с самыми разнообразными интерфейсами: Centronics и IEEE 1284 (LPT-порты), RS-232C (эмуляция UART 16550A – основы COM-портов) и другие последовательные интерфейсы (RS-422, RS-

485, V.35), эмуляторы портов клавиатуры и даже Game-порта, переходники на шину ATA, ISA, PC Card и любые другие, для которых достаточно производительности. Здесь USB становится палочкой-выручалочкой, когда встает проблема 2-го (3-го) LPT или COM-порта и в других ситуациях. При этом программное обеспечение преобразователя может обеспечить эмуляцию классического варианта «железа» стандартных портов IBM PC, но только под управлением ОС защищенного режима. Скорость передачи данных через конвертер USB-LPT может оказаться даже выше, чем у реального LPT-порта, работающего в режиме SPP.

Область применения USB не ограничена мультимедийными приложениями. Этот скоростной, рассчитанный на обслуживание большого числа устройств интерфейс удобен для аппаратуры связи, сбора и хранения информации, которую традиционно подключают к портам COM и LPT компьютеров.

К сожалению, замена интерфейса в существующем устройстве довольна сложна. Один из способов решения проблемы – применение преобразователей различных интерфейсов в USB. Протокол обмена данными по USB сложен и реализовать его до недавнего времени было не под силу многим специалистам. Сегодня, установив в разрабатываемом приборе одну из микросхем фирмы FTDI, можно преобразовать USB в "виртуальный" последовательный или параллельный порт и вести скоростной обмен данными привычными хорошо известными методами, не учитывая многих особенностей работы с USB.

5.2. Преобразователи USB-FIFO

В настоящее время компания FTDI (Future Technology Devices International) выпускает серии многофункциональных микросхем

FT8U100AX, FT8U232AM, FT232BM, FTU245AM, FT245BM. Первая из них позволяет создать семипортовый USB-хаб. Остальные предназначены для сопряжения различных устройств с шиной USB.

Микросхема FT245BM (FTU245AM) представляет собой преобразователь USB в параллельный интерфейс, удобно стыкуется с любыми микропроцессорами, используя их каналы прямого доступа к памяти (DMA) или порты ввода-вывода. Функциональная схема FT245BM представлена на рис.5.2.

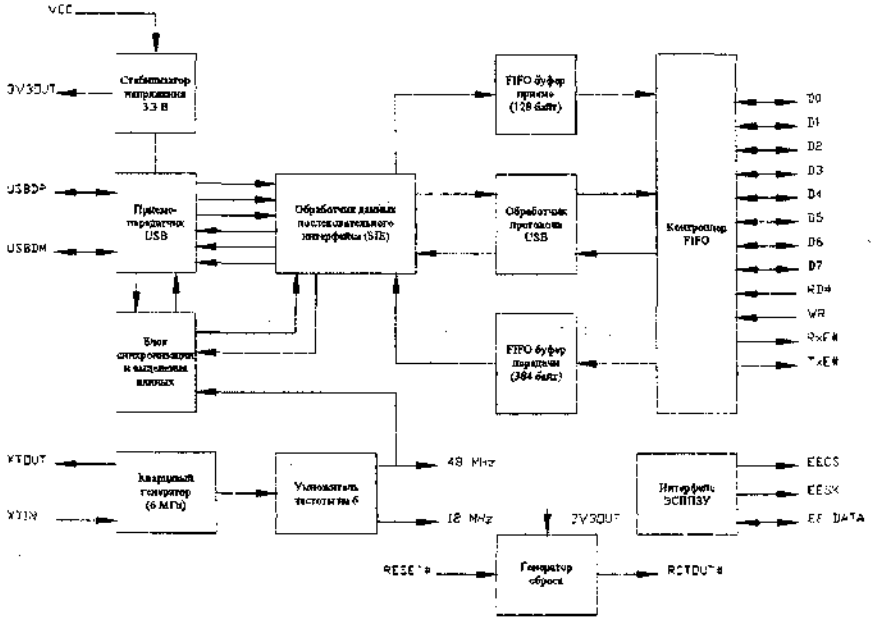


Рис.5.2. Функциональная схема FT245BM

Её основа – приемопередатчик USB и контроллер FIFO. Приемопередатчик USB снабжен двумя информационными выводами USBDP и USBDM, образующими двунаправленный канал передачи данных.

Блок SIE преобразует последовательный код в параллельный и обратно, выполняет процедуры битстаффинга, генерирует (для исходящего потока данных) и проверяет (для входящего) контрольные коды.

Обработчик протокола USB нижнего уровня формирует ответы на запросы host-контроллера (компьютера). Через него управляют работой FIFO контроллера. В микросхеме предусмотрены два буфера промежуточного хранения данных (FIFO) емкостью 384 байта (на прием) и 128 байт (на передачу). Управление FIFO возложено на соответствующий контроллер.

Задающий генератор микросхемы работает от внешнего кварцевого или керамического резонатора на 6 МГц. Далее его частоту умножают на 8 (до 48 МГц).

Выводы EECS, EESK, EEDATA микросхемы FT245BM предназначены для подключения внешней энергонезависимой памяти – микросхемы ЭСППЗУ AT93C46, в которой хранят идентификаторы изготовителя (VID) и персональный (PID) заводской номер изделия и другие данные. Это необходимо, если по USB с компьютером одновременно связаны несколько устройств на микросхемах FT245BM. Особенно важен серийный номер, так как программный драйвер полагается на его уникальность.

5.3. Подключение микросхемы FT245BM к USB

На рис.5.3 представлена схема подключения микросхемы FT245BM к USB. Питание устройства автономное.

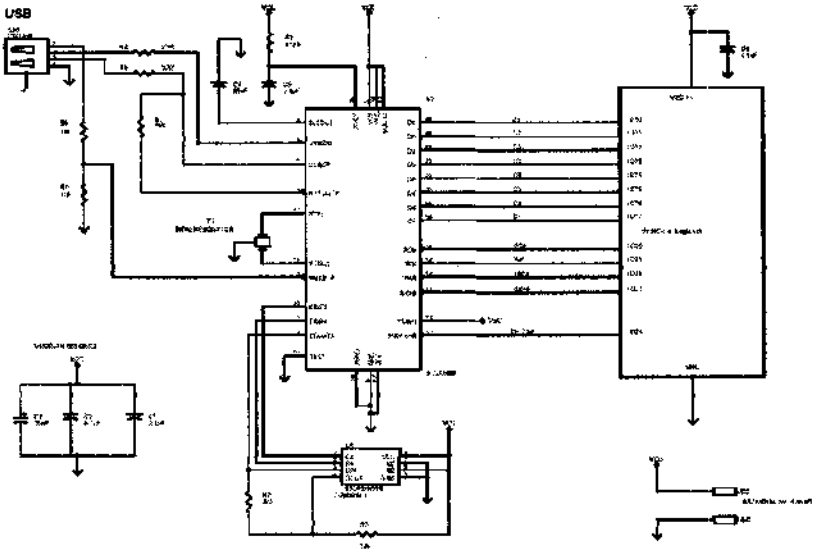


Рис.5.3. Пример схемы с автономным питанием

В схеме с автономным питанием нет необходимости принимать меры по обеспечению перехода устройства в состояние ожидания вместе с шиной USB, поскольку оно не потребляет от неё ток. Тем не менее, использование вывода PWREN# позволяет определить устройству что компьютер перешел в режим ожидания и не инициировать процедуру обмена данными. На рис.5.4 представлена схема подключения устройства, в которой используется питание от шины USB. Линии данных D0-D7 подключаются к одному из портов микроконтроллера. Также необходимы еще 4 линии ввода/вывода микроконтроллера для управляющих сигналов и сигналов состояния. Вывод RXF#, высокий уровень которого означает, что очередь приема пуста, удобно соединить с входом внешнего прерывания. В таком случае нет необходимости проверять состояние этого вывода на предмет наличия дан-

ных в буфере приема. Это можно реализовать в обработчике прерывания. При этом необходимо настроить обработку прерывания по спаду сигнала.

На рис.5.4 представлена схема сопряжения микросхемы FT245BM с микроконтроллером. В схеме использовано питание от шины USB.

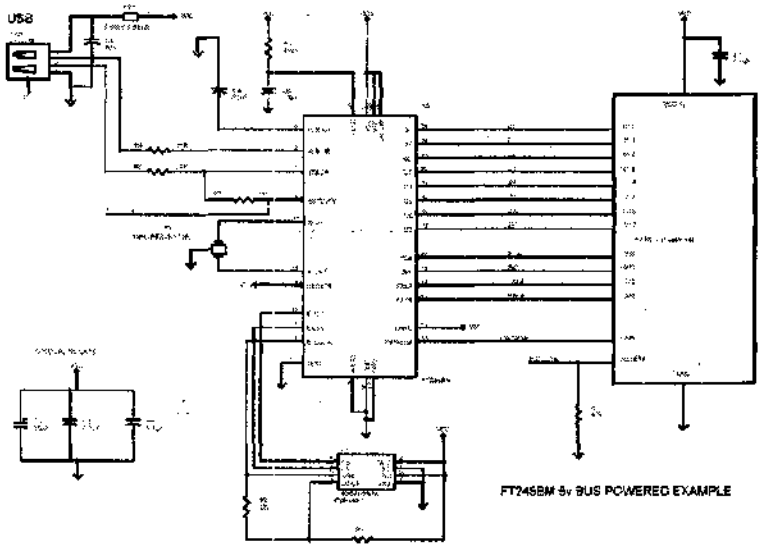


Рис.5.4. Пример схемы с питанием от шины USB

Величина тока, потребляемого схемой с питанием от шины USB не должна превышать значения 100 мА. Микроконтроллер может определить режим ожидания шины USB, анализируя значение на выводе PWREN# и также перейти в режим ожидания, при этом ток, потребляемый от шины USB, не будет превышать 500мкА. После подключения устройства к шине на этом выводе устанавливается сигнал высо-

кого уровня и он будет удерживаться до тех пор, пока не завершится конфигурация устройства и оно будет готово для работы. Вывод RSTOUT# используется для обеспечения сброса схемы при включении. Вывод сброса микроконтроллера должен быть соединен с выводом RSTOUT# и подтянут к линии земли через сопротивление 47 кОм. Для организации обмена данными в микроконтроллере использованы два порта. Один 8-разрядный порт используется для передачи данных, а второй используется для формирования сигналов управления RD#, WR, TXE#, RXF#. Для повышения помехоустойчивости схемы рекомендуется по цепи питания использовать ферритовое кольцо.

В табл.5.2. приводится описание выводов микросхемы FT245BM.

Таблица 5.2. **Описание выводов микросхемы**

Название вывода	Тип	Описание
USBDP	двунапр.	Линия данных шины USB (+)
USBDM	двунапр.	Линия данных шины USB (–)
VCCIO	питание	Питание интерфейса FIFO и группы управляющих сигналов. Соединяется с линией питания.
GND	питание	Земля
3V3OUT	выход	Используется для питания внутренней логики микросхемы. Соединен с землей через конденсатор 100 нФ. При токе потребления не более 50 мА может использоваться для питания внешней логики.
VCC	питание	Питание микросхемы
AGND	питание	Аналоговая земля для работы внутреннего умножителя частоты.
NC	NC	Не используется
RESET#	вход	Используется для сброса микросхемы внешним устройством. Должен быть подтянут к линии питания.

Продолжение табл 5.2.

Название вывода	Тип	Описание
TEST	вход	Переводит устройство в тестовый режим. При нормальной работе заземлен.
OSCI	вход	Вход генератора частотой 12 МГц. При нормальной работе остается не подсоединенным
OSCO	выход	Выход генератора частотой 12 МГц. При нормальной работе остается не подсоединенным
RD#	вход	Низкий уровень устанавливает данные из буфера FIFO на линиях D0-D7.
WR	вход	По спаду сигнала данные на линиях D0-D7 записываются в буфер передачи FIFO.
TXE#	выход	Высокий уровень означает переполнения буфера передачи. Запись разрешена только при низком уровне сигнала.
RXF#	выход	Высокий уровень означает что очередь приема пуста и данные нельзя считывать. Низкий уровень означает, что в буфере приема есть данные и они могут быть считаны.

Временные диаграммы чтения и записи байта представлены на рис.5.5 и рис.5.6 соответственно. В табл.5.3 приведены минимальные и максимальные значения ряда временных параметров.

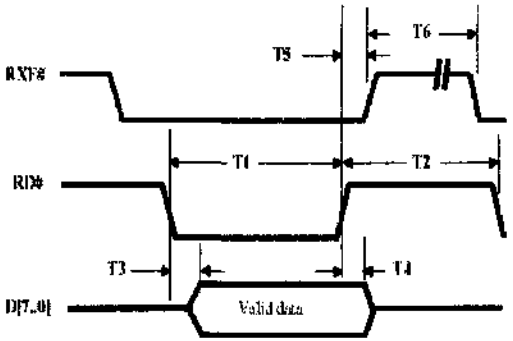


Рис.5.5. Чтение байта из буфера

Переход уровня сигнала RXF# из состояния лог.1 в состояние лог.0 означает, что в буфере приема FT245BM имеются данные. Для считывания данных необходимо установить на выводе RD# сигнал низкого уровня. При этом байт данных будет установлен на выводах D0-D7. Операцию считывания данных нужно выполнять до тех пор, пока на выходе RXF# не появится высокий уровень сигнала.

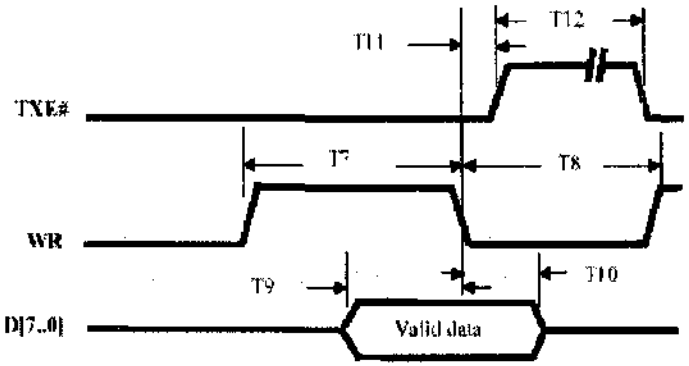


Рис.5.6. Запись байта в буфер

Для передачи данных по USB из контроллера в компьютер необходимо установить высокий уровень сигнала WR. После этого на шину данных D0-D7 вывести передаваемый байт. При переходе уровня сигнала WR из высокого уровня в низкий, байт данных будет записан в буфер передачи. После заполнения всех 384 байт буфера передачи остается высоким уровень сигнала TXE# и микросхема перестает воспринимать новые данные, пока содержимое буфера не будет переправлено по USB в компьютер.

Чтобы не задерживать данные, предусмотрен таймер на 16 мс. Если в течение этого интервала буфер передачи не заполнен до конца и новые данные не поступают, содержимое буфера автоматически пересылается в компьютер.

Таблица 5.3. **Значения временных параметров**

Время	Описание	Min	Max	Ед.
T1	Длина импульса чтения	50	-	нс
T2	Время между импульсами чтения	50	-	нс
T3	Время между спадом RD# и данными	-	30	нс
T4	Время удержания данных после фронта RD#	10	-	нс
T5	Время перехода RXF# в пассивное состояние после фронта RD#	5	25	нс
T6	Время пассивного состояния RXF# после цикла чтения	80	-	нс
T7	Длина импульса записи	50	-	нс
T8	Время между импульсами записи	50	-	нс
T9	Время установления данных перед спадом WR	-	20	нс
T10	Время удержания данных после спада WR	10	-	нс
T11	Время перехода TXE# в пассивное состояние после спада WR	5	25	нс
T12	Время пассивного состояния TXE# после цикла записи	80	-	нс

Компания FTDI Chip предлагает удобное решение – отладочный набор (модуль) UM245R, в котором имеется микросхема FT245BM. Этот модуль содержит все необходимые внешние элементы для рабо-

ты преобразователя, а также имеет удобный интерфейс для сопряжения с каким-либо устройством. На рис.5.7 представлен внешний вид такого модуля и расположение выводов.

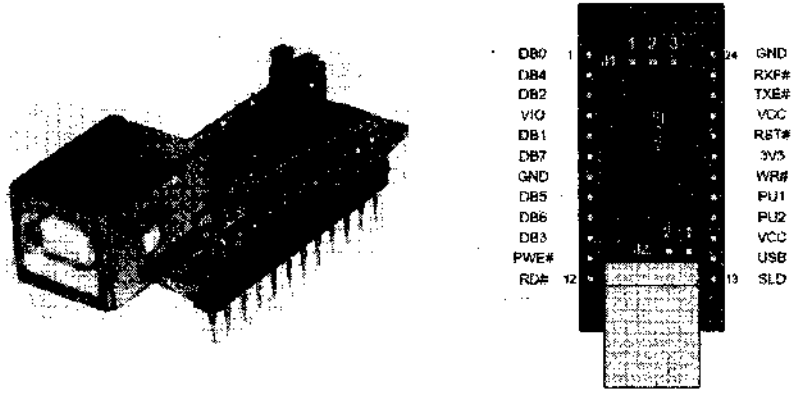


Рис.5.7. Модуль UM245R, а – внешний вид; б – расположение выводов;

Такой модуль содержит все необходимые внешние элементы для работы микросхемы FT245BM, память EEPROM, имеет удобное расположение выводов и разъем USB.

5.4. Преобразователи USB-RS232

Микросхемы FT8U232AM, FT232BM – это преобразователи USB в традиционный последовательный интерфейс – можно устанавливать в USB-модемах, переходниках COM-USB, сканерах штрих-кода, измерительной аппаратуре – фактически в любых устройствах, ранее использовавших сравнительно медленные интерфейсы RS-232, RS-422, RS-485. Микросхема FT8U232AM способна передавать данные в обе стороны со скоростью до 2000 кБит/с, причем пользователю не требуется никаких знаний об устройстве и работе USB. Поставляемые компанией FTDI программные драйвера создают впечатление, что обмен идет через обычный COM-порт.

5.5. Подключение микросхемы FT232BM к USB

На рис.5.8 представлена одна из возможных схем подключения микросхемы FT232BM к USB. Питание устройства автономное.

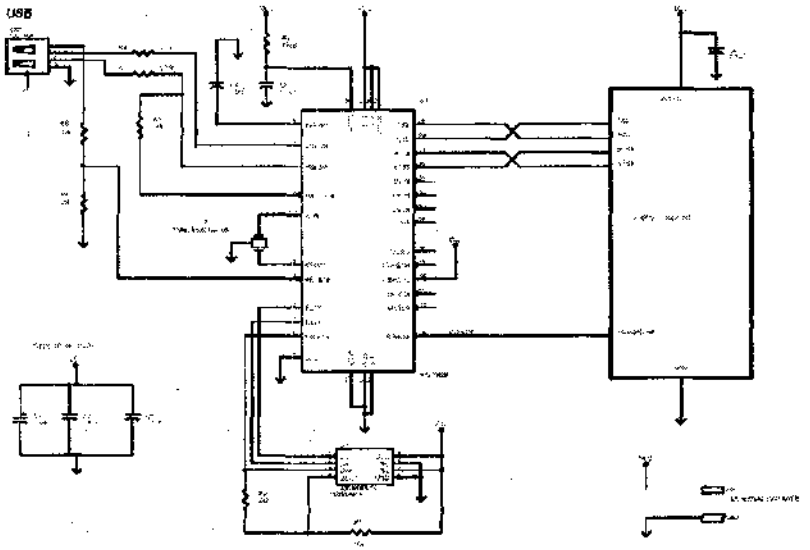


Рис.5.8.

В данном устройстве нет необходимости следить за током потребления, которой в случае питания от USB не должен превышать 500 мА. В этом примере удобно подключить вывод SLEEP# к одному из входов микроконтроллера, поскольку это позволит узнать о том, что контроллер в режиме ожидания и не способен начать обмен данными. Вход PWRCTL подтянут к питанию, что означает, что устройство имеет автономное питание.

На рис.5.9 представлена схема подключения микросхемы FT232BM к USB. Устройство питается от шины USB.

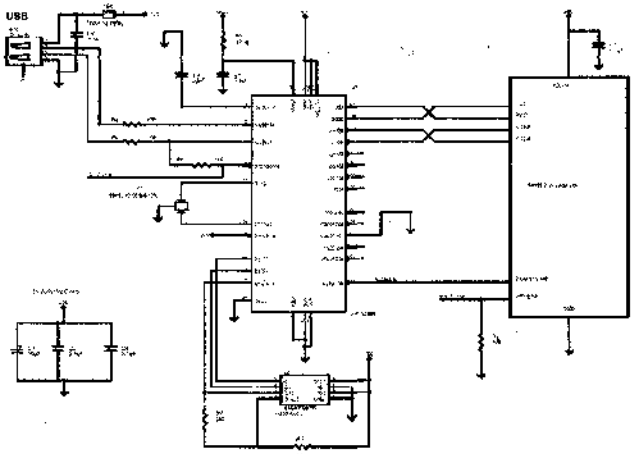


Рис.5.9.

Выход передатчика UART микросхемы (TxD) соединяется с входом RxD микроконтроллера. Выход передатчика микроконтроллера соединяется с входом приемника микросхемы. Во многих приложениях достаточно этих двух соединений. Микросхема FT232BM также имеет выходы, которые имеются на стандартном COM – порте. При необходимости их можно использовать. Уровни напряжений на выходах микросхемы соответствуют уровням ТТЛ, поэтому для сопряжения с микроконтроллером нет необходимости использовать специальные микросхемы-преобразоватли уровней.

Наличие выводов как в COM-порте позволяет использовать микросхему для подключения модемов, принтеров, других устройств к компьютеру по шине USB. При установке специальных драйверов (см.разд.6) работа с шиной USB не будет отличаться от работы с COM - портом. В следующем разделе подробно описаны принципы программирования USB с использованием библиотеки FTD2XX.

6. ПРОГРАММИРОВАНИЕ USB -ШИНЫ

6.1. Установка драйверов

Для того чтобы система USB заработала, необходимо, чтобы были загружены драйверы хост-контроллера (или контроллеров, если их несколько). При подключении устройства к шине USB ОС Windows выдает сообщение «Обнаружено новое устройство» и, если устройство подключается впервые, предлагает загрузить для него драйверы (рис.6.1).

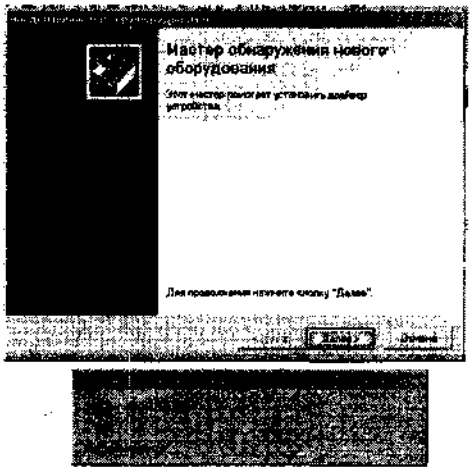


Рис.6.1. Запуск мастера установки оборудования

Нажав кнопку «Далее» можно перейти к следующему этапу установки драйвера (рис.6.2).

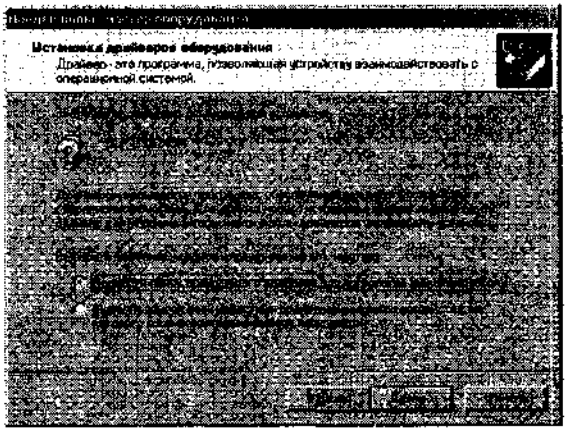


Рис.6.2.

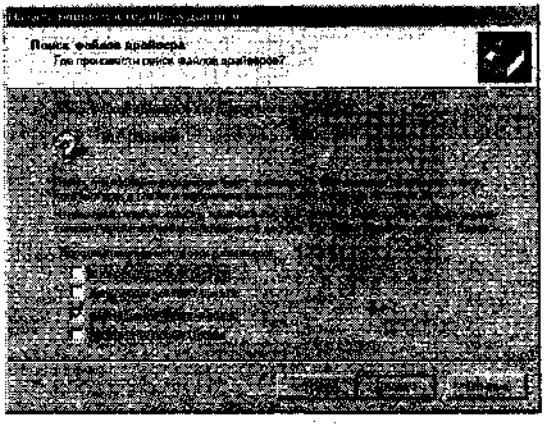


Рис.6.3.

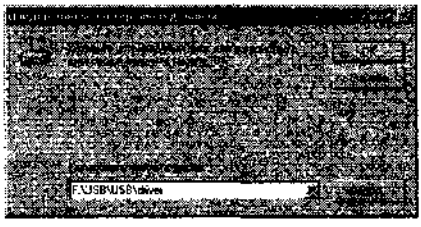


Рис.6.4.

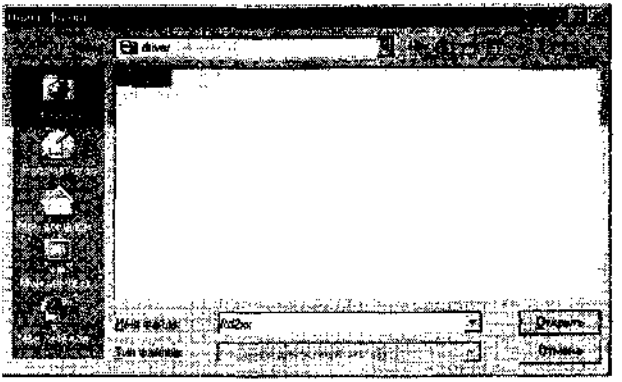


Рис.6.5.

Нажав на кнопку «Далее» выполнится установка драйвера, о чем сообщит диалоговое окно на рис.6.6.

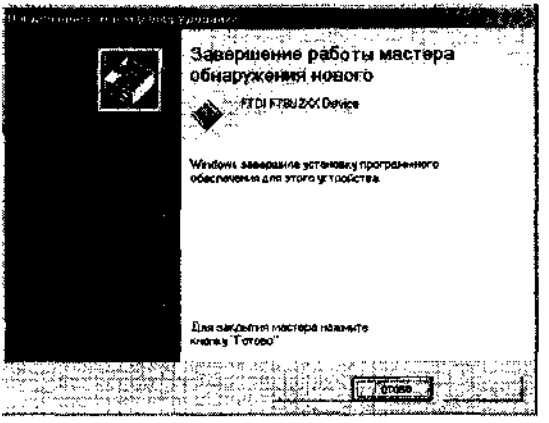


Рис.6.6.

В диспетчере устройств появится новый пункт (рис.6.7).

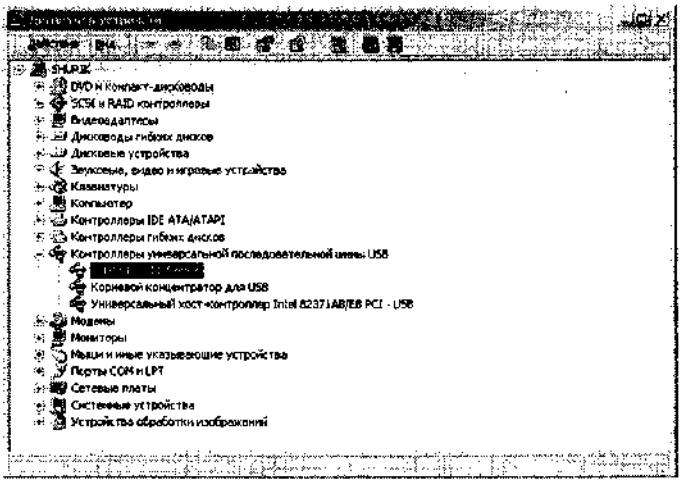


Рис.6.7. Диспетчер устройств

Вместе с драйвером поставляются следующие файлы (рис.6.8).

Имя	Размер	Тип
FTD2XX	15 KB	C compiler header file
FTD2XX	16 KB	C compiler library file
D2XX Release Info	2 KB	Text file
D2XX_EX	20 KB	Документ Microsoft Word
FTD2XX.dll	68 KB	Компонент приложения
FTD2XXUN	1 KB	Параметры конфигурации
FTD2XXUN	397 KB	Приложение
ftd2xx	3 KB	Сведения для установки
FTD2XX.sys	25 KB	Системный файл

Рис.6.8. Файлы, поставляемые с драйвером

Компания FTDI предлагает решение, не требующее драйверов, эмулирующих последовательный порт. Архитектура, названная её авторами D2XX, основана на технологии WDM. Программирование устройства ведется через стек USB и динамическую библиотеку драйвера.

Архитектуру D2XX можно представить в виде схемы (рис.9).

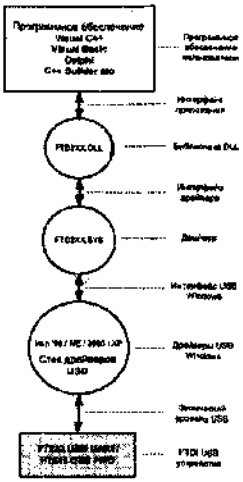


Рис.6.9. Архитектура D2XX

6.2.Определение подключенных устройств. Получение информации об устройстве.

Для получения информации о подключенном устройстве используется функция, прототип которой имеет вид:

```
FT_STATUS FT_ListDevices(
PVOID pvArg1,
PVOID pvArg2,
DWORD dwFlags
);
```

К USB одновременно могут быть подключены несколько устройств, и с помощью этой функции имеется возможность получить количество подсоединенных устройств, их серийные номера и описание. При успешном выполнении функции возвращается значение FT_OK. При возникновении ошибки возвращается код ошибки. В за-

висимости от значения флагов (параметр *dwFlags*), через параметры *pvArg1* и *pvArg1* может возвращаться различная информация.

В самом простом варианте вызова этой функции возвращается количество подключенных устройств. В этом случае параметр *dwFlags* должен принимать значение FT_LIST_NUMBER_ONLY. Первый параметр (*pvArg1*) интерпретируется как указатель на переменную типа DWORD, в которой будет храниться значение количества подсоединенных устройств, а второй параметр должен принимать значение NULL. Если параметр *dwFlags* имеет значение FT_LIST_BY_INDEX, то в параметре *pvArg1* передается номер устройства, а в параметре *pvArg2* передается указатель на буфер, в который будет помещена информация о соответствующем устройстве. Какая именно информация будет размещена в буфере, выбирается установкой одного из флагов (параметр *dwFlags*):

FT_OPEN_BY_SERIAL_NUMBER – возвращается серийный номер устройства;

FT_OPEN_BY_DESCRIPTION – возвращается описание устройства.

Если был передан неправильный номер устройства, функция возвратит значение FT_DEVICE_NOT_FOUND.

Функция FT_ListDevices важна тем, что позволяет получить информацию обо всех устройствах, подключенных в данный момент. Это позволяет выбрать устройство, с которым будет организован обмен данными. Это важно, поскольку часто пользователь может подключить несколько однотипных устройств. При этом возникнут трудности обращения к одному из них. Если же в программе предусмотреть возможность выбора определенного устройства (например, с помощью списка), то проблема легко решается. Нумерация устройств

начинается с 0. Поэтому для получения информации о каждом из подключенных устройств достаточно увеличивать индекс на 1 и повторно вызывать эту функцию. Второй способ заключается в том, что в качестве флага передается значение FT_LIST_ALL. В этом случае первый параметр (*pvArg1*) интерпретируется как массив указателей на буферы, в которых будет размещена информация о подключенных устройствах. Важно, чтобы последний элемент в массиве указателей на буферы был равен NULL. Второй параметр – указатель на переменную типа DWORD, в которой будет храниться количество подключенных устройств. Следующий пример показывает различные способы использования функции FT_ListDevices.

Пример 1.

Используя среду разработки Visual Studio .NET, создать программу, которая по нажатию одной из кнопок выдавала информацию о количестве подключенных устройств, а также их серийные номера и описание.

Пояснение. Порядок действий.

1. Запускаем среду разработки Visual Studio .NET. Создаем новый проект с названием USBTest. В нашем примере приложение создается на базе диалогового окна.

2. Проектируем внешний вид диалогового окна. Помещаем на форму три кнопки (рис.6.10).

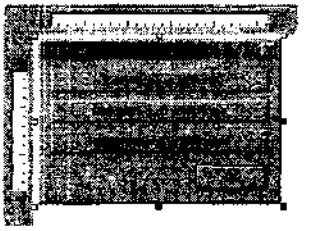


Рис.6.10. Спроектированное диалоговое окно программы

3. В каталог, где размещаются файлы созданного проекта, скопировать следующие файлы: FTD2XX.h, FTD2XX.lib, FTD2XX.dll.

В начале файла USBTestDlg.cpp, для подключения библиотеки для работы с USB необходимо добавить следующие две строки:

```
#include "FTD2XX.h"        // подключение заголовочного файла
#pragma comment(lib,"FTD2XX.lib")// подключение библиотеки
```

4. Добавляем обработчики события нажатия на кнопки. Обработчик нажатия на кнопку «Количество устройств» имеет следующую реализацию:

```
void CUSBTestDlg::OnBnClickedButton1()
{
    FT_STATUS fts;
    DWORD num;
    CString str;
fts=FT_ListDevices(&num, NULL, FT_LIST_NUMBER_ONLY);
// получение количества подключенных устройств
if(fts!=FT_OK) // проверка результата выполнения функции
    {
        MessageBox("Ошибка выполнения функции");
        return;
    }
str.Format("Количество подключенных устройств: %d",num);
    MessageBox(str); // вывод сообщения
}
```

5. Обработчик нажатия на кнопку «Информация об устройстве» имеет следующий вид:

```
void CUSBTestDlg::OnBnClickedButton2()
```

```
{
    FT_STATUS fts;
    DWORD index=0;
    char buffer[16];
// буфер для хранения описания устройства
// получение описания устройства
    fts=FT_ListDevices((PVOID)index,buffer,FT_LIST_BY_INDE
X | FT_OPEN_BY_SERIAL_NUMBER);
    if(fts!=FT_OK) // если функция не выполнилась
    {
        MessageBox("Ошибка выполнения функции");
        return;
    }
    CString str(buffer); // создание строки
    str="Серийный номер устройства: "+str;
    MessageBox(str);      // вывод сообщения
}
```

6. Добавляем обработчик нажатия на кнопку «Информация об устройствах».

```
void CUSBTestDlg::OnBnClickedButton3()
{
    FT_STATUS fts;
    char* buffptr[3];       // массив указателей на строки
    char buffer1[64]=""; // объявление буферов
    char buffer2[64]="";
    DWORD num;
    buffptr[0]=buffer1; // сохранение указателей
    buffptr[1]=buffer2;    // в массиве
```

```
        buffptr[2]=NULL;     // последний указатель NULL
        //получение информации о подключенных устройствах
        fts=FT_ListDevices(buffptr, &num, FT_LIST_ALL |
FT_OPEN_BY_DESCRIPTION);
        if(fts!=FT_OK)
        {
            MessageBox("Ошибка выполнения функции");
            return;
        }
        CString desc="";
    // в цикле формируем строки с информацией об устройствах
        for(int i=0; i<2; i++)
        {
            CString str(buffptr[i]);
            desc=desc+"Устройство: "+str+'\n';
        }
        MessageBox(desc); // вывод сообщения
    }
```

7. Откомпилировав проект, после нажатия на каждую из кнопок, появятся соответствующие сообщения.

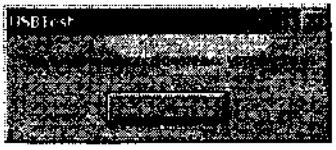


Рис.6.11. Сообщение при нажатии на кнопку «Количество устройств»

Рис.6.12. Сообщение при нажатии на кнопку «Информация об устройстве»

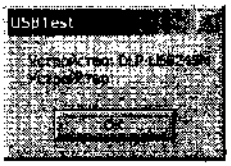


Рис.6.13. Сообщение при нажатии на кнопку «Информация об устройствах»

Для работы с устройством необходимо получить его дескриптор. Это можно выполнить с помощью функции FT_Open, прототип которой имеет вид:

```
FT_STATUS FT_Open(
int iDevice,
FT_HANDLE *ftHandle
);
```

Первым параметром (*iDevice*) в функцию передается номер открываемого устройства. Если подключено только одно устройство, значение этого параметра должно быть равно 0. Второй параметр (*ftHandle*) – указатель на переменную типа FT_HANDLE, в которой будет храниться дескриптор открытого устройства. Через этот дескриптор будет осуществляться доступ к устройству. При успешном выполнении функции возвращается значение FT_OK. Функция FT_Open может использоваться для открытия одного из нескольких устройств путем установки параметра *iDevice* в 0, 1,2 и т.д. Однако нет возможности открыть определенное устройство, поскольку номера назнача-

ются произвольным образом. Для решения этой проблемы следует использовать функцию FT_OpenEx, прототип которой имеет вид:

```
FT_STATUS FT_OpenEx(
PVOID pvArg1,
DWORD dwFlags,
FT_HANDLE *ftHandle
);
```

Значение первого передаваемого параметра (*pvArg1*) зависит от флагов (второй параметр *dwFlags*). В любом случае этот параметр интерпретируется как указатель на строку, ограниченную нулем. Второй параметр (*dwFlags*) может принимать одно из двух значений: FT_OPEN_BY_SERIAL_NUMBER – означает, что в параметре *pvArg1* содержится серийный номер устройства; FT_OPEN_BY_DESCRIPTION – в параметре *pvArg1* содержится описание устройства. Дескриптор устройства возвращается через указатель типа FT_HANDLE (третий параметр *ftHandle)*. Этот дескриптор будет использоваться для доступа к устройству. Для открытия определенного устройства в качестве параметра можно передать серийный номер или описание устройства, предварительно полученный с помощью функции FT_ListDevices.

После того, как работа с устройством завершена, его необходимо закрыть. Для этого используется функция FT_Close, прототип которой имеет вид:

FT_STATUS FT_Close(FT_HANDLE *ftHandle*);

Единственный параметр, передаваемый в функцию (*ftHandle*) – дескриптор открытого устройства. При успешном выполнении возвращается значение FT_OK.

Еще одна функция, которая позволяет получить информацию об устройстве – это FT_GetDeviceInfo. Прототип этой функции имеет вид:

```
FT_STATUS FT_GetDeviceInfo(
FT_HANDLE fth,
FT_DEVICE *ftd,
DWORD *id,
PVOID snum,
PVOID desc,
NULL);
```

Параметры функции:

fth – дескриптор открытого устройства, информацию о котором необходимо получить;

ftd – указатель на переменную типа FT_DEVICE, в которой хранится тип устройства;

id – указатель на переменную типа DWORD, в которой храниться номер устройства;

snum – указатель на буфер, в котором размещается серийный номер устройства;

desc – указатель на буфер, в котором размещается описание устройства.

Следующий пример демонстрирует принципы работы с перечисленными функциями.

Пример 2.

Используя среду разработки Visual Studio .NET, создать программу, которая по нажатию одной из кнопок открывает подключенное устройство и выводит информацию о нем.

Пояснение. Порядок действий.

1. Запускаем среду разработки Visual Studio .NET. Создаем новый проект с названием USBTest. В нашем примере приложение создается на базе диалогового окна.

2. В каталог, где размещаются файлы созданного проекта, скопировать следующие файлы:

FTD2XX.h

FTD2XX.lib

FTD2XX.dll

В начале файла USBTestDlg.cpp, для подключения библиотеки для работы с USB необходимо добавить следующие две строки:

```
#include "FTD2XX.h"   // подключение заголовочного файла
#pragma comment(lib,"FTD2XX.lib")   // подключение библиотеки
```

3. Размещаем на диалоговом окне кнопку. Форма имеет вид, показанный на рис.6.14.

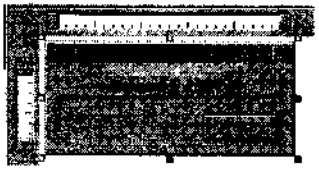


Рис.6.14. Окно программы

Обработчик события нажатия на кнопку имеет вид:

```
void CUSBTestDlg::OnBnClickedButton1()
{
 FT_HANDLE fth=NULL;
     FT_STATUS fts=NULL;
FT_DEVICE ftd;
     DWORD id;
char snum[256];
```

```
    char desc[256];
    fts=FT_Open(0,&fth);        // открытие устройства
    if(fts!=FT_OK)
    {
        // если устройство не открыто, выдаем сообщение
        MessageBox("Error opening!");
        return;
    }
// получение информации об устройстве
    fts=FT_GetDeviceInfo(fth,&ftd,&id,snum,desc,NULL);
    if(fts!=FT_OK)
    {
        MessageBox("Error get info");
        return;
    }
    else
    {
    // формирование строк
        CString s3(snum);
        CString s4(desc);
        CString s1;
        CString s2;
        s1.Format("Device type: %d\n",ftd);
        s2.Format("Device Id: %d\n",id);
        s3="Device serial number: "+s3+'\n';
        s4="Device description: "+s4+'\n';
        MessageBox(s1+s2+s3+s4);
    }
```

```
        fts=FT_Close(fth);     // закрытие устройства
        if(fts!=FT_OK)
        {
            MessageBox("Error close!");
            return;
        }
}
```

После выполнения и нажатия на кнопку появится сообщение, показанное на рис.6.15.

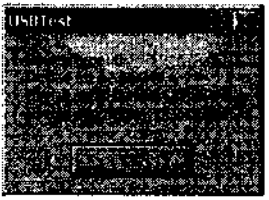


Рис.6.15. Сообщение с информацией об устройстве

6.3. Организация обмена данными

К основным операциям обмена данными относятся запись данных в устройство и чтение данных из устройства. Для передачи данных используется функция FT_Write, прототип которой имеет следующий вид:

```
FT_STATUS FT_Write(
FT_HANDLE ftHandle,
LPVOID lpBuffer,
DWORD dwBytesToWrite,
LPDWORD lpdwBytesWritten
);
```

Параметры функции имеют следующие значения:

ftHandle – дескриптор открытого устройства, полученный с помощью функции FT_Open или FT_OpenEx;

lpBuffer – указатель на буфер данных, которые будут переданы в устройство;

dwBytesToWrite – количество байт для передачи;

lpdwBytesWritten – указатель на переменную типа DWORD, в которой возвращается количество переданных байт.

Если функция выполнилась успешно, возвращается значение FT_OK, в противном случае – код ошибки.

Для считывания данных из устройства используется функция FT_Read. Прототип этой функции имеет следующий вид:

```
FT_STATUS FT_Read
(
FT_HANDLE ftHandle,
LPVOID lpBuffer,
DWORD dwBytesToRead,
LPDWORD lpdwBytesReturned
);
```

Параметры, передаваемые в функцию:

ftHandle – дескриптор устройства, из которого будет происходить считывание данных;

lpBuffer – указатель на буфер, в котором будут размещены считанные данные;

dwBytesToRead – количество байт, которые должны быть считаны из устройства;

lpdwBytesReturned – указатель на переменную типа DWORD, в котором возвратится количество считанных из устройства байт.

Если функция выполнилась успешно, возвращается значение FT_OK, в противном случае – код ошибки.

Особенность функции FT_Read в том, что она не возвратит значение до тех пор, пока не будет считано количество байт, указанное в параметре *dwBytesToRead*. Это может привести к зависанию программы, если подключенное к компьютеру устройство окажется неработающим. Есть несколько способов решения этой проблемы. Первый из них заключается в использовании функции FT_GetQueueStatus. С помощью этой функции можно узнать количество байт в очереди приема. Прототип функции имеет вид:

```
FT_STATUS FT_GetQueueStatus(
FT_HANDLE ftHandle,
LPDWORD lpdwAmountInRxQueue
);
```

Первый параметр *ftHandle* – дескриптор устройства, из которого выполняется считывание данных. Второй параметр *lpdwAmountInRxQueue* – указатель на переменную типа DWORD, в которой возвращается количество байт в очереди приема. Перед тем, как вызывать функцию FT_Read, необходимо знать количество байт в очереди приема. Поэтому предварительно следует вызвать функцию FT_GetQueueStatus, которое возвратит количество байт в очереди приема, и это значение передать в функцию FT_Read.

Еще одна функция, которая может быть использована для получения состояния устройства – FT_GetStatus. Прототип этой функции имеет вид:

```
FT_STATUS FT_GetStatus(
FT_HANDLE ftHandle,
LPDWORD lpdwAmountInRxQueue,
```

LPDWORD lpdwAmountInTxQueue,
LPDWORD lpdwEventStatus
);

Параметры, передаваемые в функцию:

ftHandle – дескриптор устройства, информацию о состояние которого необходимо получить;

lpdwAmountInRxQueue – указатель на переменную типа DWORD, в которой возвращается значение количества байт в очереди приема;

lpdwAmountInTxQueue – указатель на переменную типа DWORD, в которой возвращается значение количества байт в очереди передачи;

lpdwEventStatus – указатель на переменную типа DWORD, в которой возвращается текущее событие.

Последний параметр может использоваться для обнаружения одного из следующих событий:

FT_EVENT_RXCHAR – событие, связанное с приемом данных в очередь передачи устройства;

FT_EVENT_MODEM_STATUS – событие, связанное с изменением сигналов состояния модема (для микросхемы FT232B(M)).

Следующий пример демонстрирует работу перечисленных функций.

Пример 3.

Используя среду разработки Visual Studio .NET, создать программу, которая выполняет контроль температуры некоторого технологического процесса. Данные о температуре принимаются через USB-порт и должны отображаться в виде гистограммы. Принятое значение соответствует температуре. Для получения данных необходимо отправить запрос – команда 0xFF.

Пояснение. Порядок действий.

1. Запускаем среду разработки Visual Studio .NET. Создаем новый проект с названием TempControl. В нашем примере приложение создается на базе диалогового окна.

2. В каталог, где размещаются файлы созданного проекта, скопировать следующие файлы:

FTD2XX.h

FTD2XX.lib

FTD2XX.dll

В начале файла TempControlDlg.h, для подключения библиотеки для работы с USB необходимо добавить следующие две строки:

```
#include "FTD2XX.h"
#pragma comment(lib,"FTD2XX.lib")
```

3. Проектируем вид диалогового окна. На форму необходимо вынести элементы управления Static Text, в котором будет отображаться значение температуры в графическом виде; две кнопки (Button) для открытия и закрытия устройства; список (Combo Box), в котором будут отображаться номера подключенных устройств. На рис.6.17 представлен внешний вид диалогового окна.

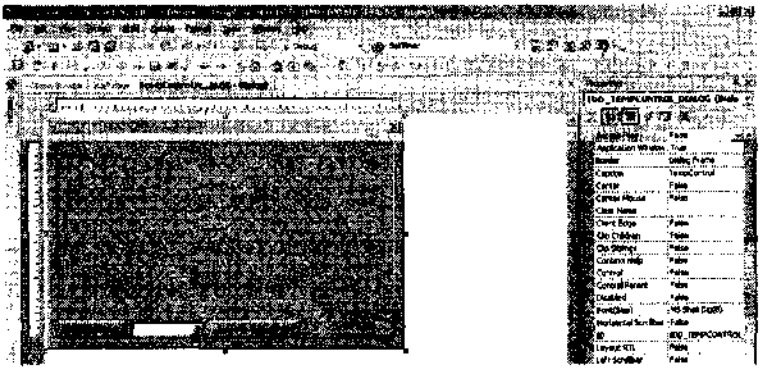


Рис.6.17. Вид спроектированного диалогового окна

Для элемента управления Static Text (текстовое поле) установить поле Sunken на странице свойств в True. Вокруг элемента управления появится рельефная рамка. Также необходимо изменить ID этого элемента управления на IDC_STATIC1 и сформировать требуемый размер окна. Привязываем к элементу управления текстовое поле (IDC_STATIC1) переменную m_graph. Для этого необходимо выделить элемент управления и из контекстного меню выбрать команду «Add Variable». Появится диалоговое окно, показанное на рис.6.18. В поле «Variable name» задать имя переменной.

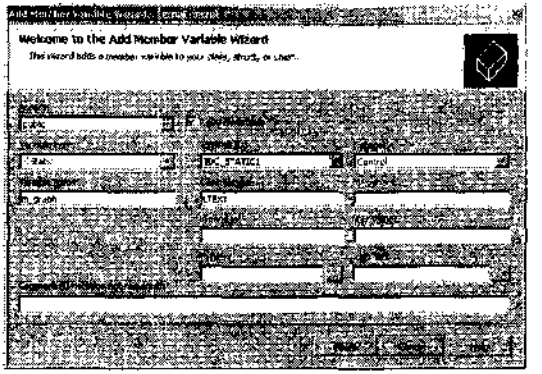


Рис. 6.18. Окно добавления переменной

4. На странице свойств элемента управления «Combo Box» (IDC_COMBO1) установить в поле «Type» значение «Drop List». Привязываем к этому элементу управления переменную m_devices.

5. Добавляем переменную – член класса CTempControlDlg, в которой будет храниться дескриптор порта. Для этого в окне «Class View» выделяем класс «CTempControlDld» и из контекстного меню, в подменю «Add» выбираем «Add Variable» (рис.6.19).

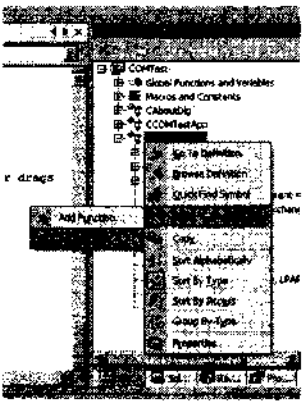


Рис.6.19.

Имя переменной должно быть m_hUSB, тип переменной – FT_HANLDE, тип доступа (Access) - public.

6. Для хранения полученных данных о температуре используется переменная m_temp и имеющая тип CByteArray. Эта переменная – член класса CCOMTestDlg. Для того, чтобы добавить переменную, в окне «Class View» выделить класс «CCOMTestDlg», вызвать контекстное меню и из подменю «Add» выбрать команду «Add Variable».

7. При запуске программы в список необходимо добавить все подключенные к USB устройства, чтобы в дальнейшем выбрать требуемое. Для этого в конце функции OnInitDialog класса CTemControlDlg, которая вызывается перед появлением диалога на экране, добавить следующий код:

```
        FT_STATUS fts;
        DWORD num;
    // получение количества подключенных устройств
        fts=FT_ListDevices(&num, NULL,
FT_LIST_NUMBER_ONLY);
        if(fts!=FT_OK)
```

```
        {
            MessageBox("Невозможно обнаружить устройство!");
            return FALSE;
        }
        for(int i=0;i<num;i++)      //
        {
            char desc[16];
    // в цикле получаем информацию о каждом подключенном устройстве
    fts=FT_ListDevices((PVOID)i, desc, FT_LIST_BY_INDEX|
FT_OPEN_BY_DESCRIPTION);
            if(fts==FT_OK)
            {
            CString str(desc);
            // добавление в список информации о подключенных
устройствах
            m_devices.AddString(str);
            }
            else MessageBox("Устройство не найдено!");
        }
        m_devices.SetCurSel(0);
            m_hUSB=NULL;
        m_temp.RemoveAll();      // очистка массива
```

8. Добавляем обработчик события нажатия на кнопку «Подключиться». Необходимо открыть устройство, номер которого выбран в списке. Функция - обработчик события имеет вид:

```
void CTempControlDlg::OnBnClickedButton1()
{
```

```
        FT_STATUS fts;
        char desc[16];
        m_devices.GetLBText(m_devices.GetCurSel(), desc);
    // получение описания устройства из выделенного элемента списка
        fts=FT_OpenEx(desc, FT_OPEN_BY_DESCRIPTION, &m_hUSB);
        // открытие устройства по его описанию
        if(fts!=FT_OK) m_status.SetWindowText("Состояние: Невозможно открыть выбранное устройство!");
        else
        {
            m_status.SetWindowText("Состояние: Устройство открыто!");
            SetTimer(1,200,NULL);    // запуск таймера
        }
        m_devices.EnableWindow(0);
    }
```

9. Добавляем обработчик события нажатия на кнопку «Закрыть устройство». Функция - обработчик события имеет вид:

```
    void CTempControlDlg::OnBnClickedButton2()
    {
        if(m_hUSB!=NULL)
        {
            FT_Close(m_hUSB); // закрытие устройства
            m_hUSB=NULL;
            m_status.SetWindowText("Состояние: Устройство закрыто!");
```

```
            m_devices.EnableWindow(1);
            KillTimer(1);    // остановка таймера
        }
        else m_status.SetWindowText("Состояние: Устройство уже
закрыто!");
    }
```

10. Добавляем обработчик события таймера. Для этого из страницы свойств (Properties) вызвать окно «Messages» и напротив позиции «WM_TIMER» выбрать «OnTimer» (рис.6.20).

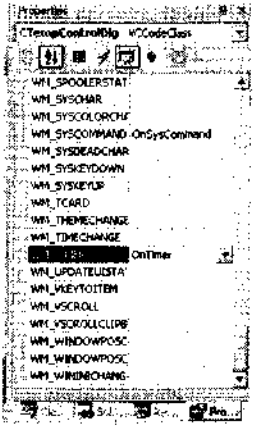


Рис.6.20. Создание обработчика события таймера

11. Добавляем в функцию – обработчик события таймера код, как показано ниже:

```
void CTempControlDlg::OnTimer(UINT nIDEvent)
{
    FT_STATUS fts;
    DWORD sb,rb=0;
    unsigned char buff[2];
    buff[0]=0xFF;
```

```
        fts=FT_Write(m_hUSB, buff,1,&sb);  // запись данных в уст-
ройство
        if((fts!=FT_OK) || (sb!=1))
        {
            m_status.SetWindowText("Состояние: Ошибка при пе-
редаче данных!");
            return;
        }
            // получение количества байт в очереди приема
            fts=FT_GetQueueStatus(m_hUSB,&rb);
            if(fts!=FT_OK)
            {
        m_status.SetWindowText("Состояние: Состояние очереди не
определено!");
            return;
            }
            if(rb<1)    // если в очереди приема нет данных
            {
        m_status.SetWindowText("Состояние: Данные о температуре
отсутствуют!");
            // выводим сообщение
    return;
            }
        fts=FT_Read(m_hUSB, buff, 1, &rb); // чтение данных из уст-
ройства
        if((fts!=FT_OK) | (rb!=1))
        {
```

```
            m_status.SetWindowText("Состояние: Ошибка при
приеме данных!");
            return;
        }
        m_temp.Add(buff[0]);        // добавление полученных данных
в массив
        CString str;
        str.Format("Текущая температура: %d", buff[0]);
        m_status.SetWindowText(str);
        DrawGraph();            // вызов функции рисования гисто-
граммы
        CDialog::OnTimer(nIDEvent);
    }
```

12. Реализуем функцию void DrawGraph(void). Для этого ее необходимо сделать членом класса CTempControlDlg. Реализация функции имеет следующий вид:

```
    void CTempControlDlg::DrawGraph(void)
    {
        CClientDC dc(&m_graph);// создание контекста для рисова-
ния
        CPen p(PS_SOLID,2,RGB(200,0,0)); // создание пера
        CPen pp(PS_SOLID,1,RGB(0,100,0));
        CRect rect;
        CRgn rgn;
        CPen* op;
        CBrush br(RGB(180,180,180)); // создание кисти серого цве-
та
        m_graph.RedrawWindow();        // перерисовка окна
```

```
        m_graph.GetClientRect(&rect);  // получение размеров окна
        rgn.CreateRectRgn(0,0,rect.Width(),rect.Height());
        op=dc.SelectObject(&pp);  // выбор пера в контекст
        dc.FillRect(&rect,&br);      // заполнение области рисования
серым цветом
        for(int x=20;x<rect.Width();x=x+20)
        {
            // рисование вертикальных линий
            dc.MoveTo(x,0);
            dc.LineTo(x,rect.Height());
        }
        dc.SetTextColor(RGB(0,0,0)); // установка цвета текста
        dc.SetBkColor(RGB(180,180,180));   // установка цвета фона
        br.DeleteObject();
        br.CreateSolidBrush(RGB(0,0,160));  // создание синей кисти
        for(int y=20;y<rect.Height();y=y+20)
        {
            // рисование горизонтальных линий
            dc.MoveTo(0,rect.Height()-y);
            dc.LineTo(rect.Width(),rect.Height()-y);
            CString s;
            s.Format("%d",y);
            // вывод текстовых меток
            dc.TextOut(rect.Width()-30,rect.Height()-y,s);
        }
        dc.SelectObject(&p);
        dc.SelectClipRgn(&rgn); // ограничение области рисования
        if(rect.Width()<=m_temp.GetCount()) m_temp.RemoveAll();
```

```
            for(int i=0;i<m_temp.GetCount();i++)
            {
                CRect r(i*5,rect.Height(),i*5+3,rect.Height()-
m_temp.GetAt(i));
                dc.FillRect(&r,&br);  // рисование гистограммы
            }
            dc.SelectObject(op);
        // если элементов в массиве больше, чем размер области перери-
совки
        // то удаляются все данные из массива
            if(rect.Width()/5<m_temp.GetCount()) m_temp.RemoveAll();
        }
```

После компиляции проекта и подключения к устройству, появится диалоговое окно, представленное на рис.6.21.

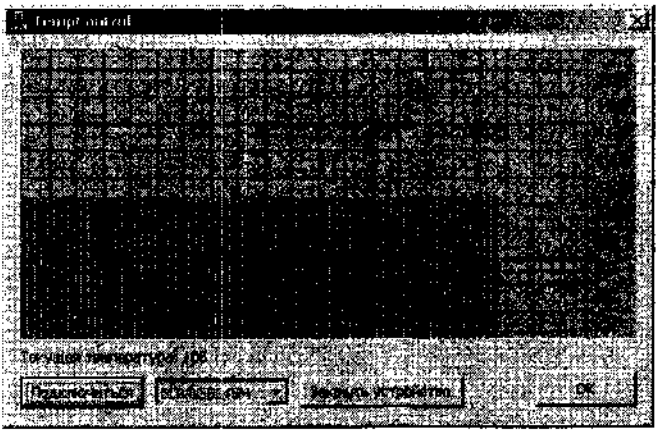


Рис.6.21. Окно программы

6.4. Программа для контроллера AVR

Микросхема FT245BM (FTU245AM) удобно стыкуется с любыми микропроцессорами, используя порты ввода-вывода. О наличии полученных по USB данных (объем буфера приема – 128 байт) свидетельствует низкий уровень сигнала $\overline{R \times F}$. Данные считывают, пока буфер не опустеет и уровень $\overline{R \times F}$ не станет высоким. После заполнения всех 384 байт буфера передачи остается высоким уровень сигнала $\overline{T \times E}$, и микросхема перестает воспринимать новые данные, пока содержимое буфера не будет перенаправлено по USB в компьютер.

Пример 4: На аналоговый вход микроконтроллера (PA0) подается сигнал, пропорциональный температуре. Написать программу, которая выполняет оцифровку этого сигнала при приеме кода 0xFF. Оцифрованный сигнал (1 байт) передается в компьютер по USB. В качестве преобразователя используется микросхема FT245BM. Микросхема подключена к портам B, D микроконтроллера. Порт C не используется. Тактовая частота кварцевого генератора – 8 МГц. Тактовая частота АЦП – 125 кГц, преобразование осуществляется за 14 тактов. Схема соединения микроконтроллера и микросхемы показана на рис.6.22.

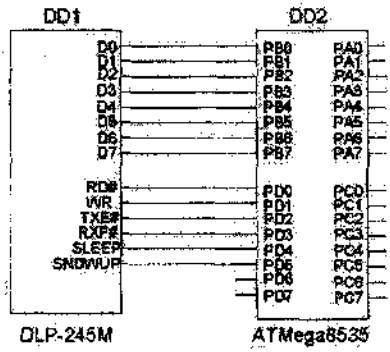


Рис.6.22. Схема сопряжения контроллера и отладочного комплекта FT245BM.

Ниже приводится текст программы для микроконтроллера.

```
#include <iom8535v.h>      // подключение заголовочных файлов
#include <macros.h>
#define RXF PIND&0b00001000
#define TXE PIND&0b00000100
#define SLEEP PIND&0b00010000
#define RD() PORTD|=0b00000001
#define NRD() PORTD&=~0b00000001
#define WR() PORTD|=0b00000010
#define NWR() PORTD&=~0b00000010
 void init_cpu()        // функция инициализации контроллера
{
 DDRD=0b11000011;       // настройка 1,2,6 и 7 выводов на выход
 PORTD=0b00000001; // установка лог.1 на 0 выводе
 DDRB=0;                // порт B настроен на ввод
 PORTB=0;
 ADCSRA=0x86;           // настройка АЦП, частота 125 кГц
 ADMUX=0;           // выбор 0 канала мультиплексора
}
// функция преобразования АЦП
unsigned char read_adc(char adc_input)
{
unsigned char adcw; // переменная для хранения данных
char a;                // дополнительные переменные
char b;
ADMUX=adc_input;// установка канала мультиплексора
ADCSRA|=0x40;          // начало преобразования
// ожидание окончания преобразования АЦП
while ((ADCSRA & 0x10)==0);
ADCSRA&=~0x10; // сброс флага завершения преобразования
a=ADCL;       // считывание младшего байта
b=ADCH;       // считывание старшего байта
adcw=b<<6; // формирование 8-битного значения
a=a>>2; // два младших бита отбрасываются
adcw=adcw|a;
return adcw;
}
void delay(int ms)    // функция задержки
{
 int a,b;          // дополнительные переменные
 for(a=0;a<ms;a++)          // в цикле вызывается «пустая операция»
```

```
for(b=0;b<970;b++) NOP();
}
void main()          // главная функция программы
{
char rdata;          // переменная для хранения считанных данных
char sdata;          // переменная для хранения значения с АЦП
init_cpu();          // вызов функции инициализации контроллера
while(1)                  // вечный цикл
{
while(RXF);          // ожидание приема байта
NOP();               // «пустая» операция (задержка)
NRD();               // установка сигнала #RD в 0
NOP();
rdata=PINB;          // считывание данных
RD();                // установка сигнала #RD в 1
if(rdata==0xFF)      // если принято значение 0xFF
{
  sdata=read_adc(0);       // вызов функции преобразования АЦП
while(TXE);          // ожидание опустошения буфера передачи
DDRB=0xFF;           // настройка порта B на выход
PORTB=sdata;              // вывод данных в порт
NOP();               // пауза
WR();                // установка сигнала WR
NOP();               // пауза
NWR();               // сброс сигнала WR
PORTB=0;                  // вывод значения 0 в порт B
DDRB=0;                   // перевод порта B в Z-состояние
}
}
}
```

При включении устройства происходит вызов функции инициализации контроллера init_cpu(). Выполняется настройка портов ввода-вывода, а также настройка АЦП. После этого выполнение программы приостанавливается до тех пор, пока не будет принят байт данных. Если данные получены, то на линии RXF# устанавливается низкий уровень сигнала. После этого следует установить лог.0 на линии RD#. На шине данных (D0-D7) установится принятый байт данных, который можно считать в контроллер. Если из компьютера было передано

несколько байт, то необходимо выполнять операцию чтения до тех пор, пока сигнал на линии RXF# не примет значения лог.1. Если принята команда начала преобразования, вызывается функция преобразования АЦП. После того, как преобразование закончено, происходит ожидание установки на линии TXE# сигнала низкого уровня. Это свидетельствует о том, что буфер передачи пуст и в него может быть записан байт данных. Для этого устанавливается высокий уровень сигнала на линии WR, порт B настраивается на выход, а затем на линии WR устанавливается низкий уровень. По спаду сигнала WR данные записываются в очередь передачи и они доступны для считывания из компьютера по USB.

6.5. Использование тайм-аутов

Как было сказано выше, функция FT_Read не возвратит значение до тех пор, пока в буфере приема не будет количества байт, равного значению переданного параметра *dwBytesToRead*. Поэтому существует вероятность «зависания» программы, если возникнут какие-либо проблемы с работой подключенного устройства (например, отключение устройства). Для того, чтобы решить эту проблему, можно использовать тайм-ауты. Функция, которая позволяет установить тайм-ауты, имеет следующий прототип:

```
FT_STATUS FT_SetTimeouts(
FT_HANDLE ftHandle,
DWORD dwReadTimeout,
DWORD dwWriteTimeout
);
```

Параметры функции имеют следующее значение:

ftHandle – дескриптор устройства;

dwReadTimeout – значение тайм-аута чтения (в мс);

dwWriteTimeout – значение тайм-аута записи (в мс).

При успешном выполнении функции возвращается значение FT_OK. После установки тайм-аутов вызов функции чтения (записи) данных не вызовет зависания программы при отсутствии данных в буфере приема. Функция возвратит значение по истечению времени тайм-аута. Следует отметить, что при возникновении тайм-аута и отсутствии данных в очереди приема, значение, возвращаемое функцией FT_Read, будет FT_OK. Для отслеживания таких ситуаций необходимо проверять параметр *LpdwBytesReturned* , в котором возвращается количество принятых байт. Если при чтении данных в очереди приема окажется меньшее количество байт, чем было запрошено, то по истечению времени тайм-аута, будет считано доступное количество байт. Если данные отсутствуют, значение параметра *LpdwBytesReturned* будет равно 0.

В предыдущем примере получение данных с устройств осуществлялось циклически, с периодом 200 мс. Запрос данных о температуре выполнялся посылкой команды 0xFF в устройство, а готовность данных проверялась с помощью функции FT_GetQueueStatus. Следующий пример демонстрирует использование тайм-аутов.

Пример 5.

Модифицировать программу из предыдущего примера так, чтобы при работе с устройством использовались тайм-ауты.

Пояснение. Порядок действий.

1. Открываем созданный в предыдущем примере проект TempControl.

2. Модифицируем обработчик нажатия на кнопку «Подключиться». Функция имеет следующий вид:

```
void CTempControlDlg::OnBnClickedButton1()
{
  FT_STATUS fts;
  char desc[16];
  // получение описания устройства из списка
  m_devices.GetLBText(m_devices.GetCurSel(), desc);
  // открытие устройства по его описанию
  fts=FT_OpenEx(desc, FT_OPEN_BY_DESCRIPTION, &m_hUSB);
  if(fts!=FT_OK) m_status.SetWindowText("Состояние: Невозможно от-
крыть выбранное устройство!");
  else
  {
      m_status.SetWindowText("Состояние: Устройство открыто!");
      FT_SetTimeouts(m_hUSB,100,100); // установка тайм-аутов
      SetTimer(1,200,NULL); // установка таймеров
  }
  m_devices.EnableWindow(0);
}
```

3. Обработчик таймера имеет вид:

```
void CTempControlDlg::OnTimer(UINT nIDEvent)
{
  FT_STATUS fts;
  DWORD sb,rb=0;
  unsigned char buff[2];
  buff[0]=0xFF;
// передача данных в устройство
  fts=FT_Write(m_hUSB, buff,1,&sb);
  if((fts!=FT_OK) || (sb!=1))
  {
  m_status.SetWindowText("Состояние: Ошибка при передаче дан-
ных!");
      return;
```

```
    }
    fts=FT_Read(m_hUSB, buff, 1, &rb);        // чтение данных из устрой-
ства
    if((fts!=FT_OK)|(rb!=1))
    {
    m_status.SetWindowText("Состояние: Ошибка при приеме данных!");
        return;
    }
    m_temp.Add(buff[0]);
    CString str;
    // формирование строки
    str.Format("Текущая температура: %d", buff[0]);
    m_status.SetWindowText(str);
    DrawGraph();      // рисование графика
    CDialog::OnTimer(nIDEvent);
}
```

После компиляции проекта появится диалоговое окно, показанное на рис.6.21. При отключении устройства программа не «зависнет» благодаря тому, что установлены тайм-ауты. В этом случае в строке «Статус» появится сообщение «Ошибка при передаче данных!».

Для очистки очередей приема и передачи используется функция FT_Purge, прототип которой имеет вид:

```
FT_STATUS FT_Purge(
FT_HANDLE ftHandle,
DWORD dwMask
);
```

Первый параметр, передаваемый в функцию (*ftHandle*) – дескриптор открытого устройства. В качестве второго параметра (*dwMask*) может передаваться любая из комбинаций следующих значений:

FT_PURGE_RX – очищается очередь приема;

FT_PURGE_TX – очищается очередь передачи.

Иногда может потребоваться выполнить сброс устройства. Для этого используется функция, прототип которой имеет вид:

```
FT_STATUS FT_ResetDevice(
FT_HANDLE ftHandle
);
```

Единственный параметр, передаваемый в функцию – дескриптор открытого устройства. При успешном выполнении этих функций возвращается значение FT_OK. В противном случае возвращается код ошибки.

При работе устройства может возникнуть задача прервать работу с ним. Для этого используется функция FT_SetBreakOn, прототип которой выглядит следующим образом:

```
FT_STATUS FT_SetBreakOn(
FT_HANDLE ftHandle
);
```

Для возобновления работы с устройством используется функция FT_SetBreakOff , прототип которой не отличается от приведенного выше. В качестве единственного параметра в обе функции передается дескриптор открытого устройства.

6.6. Программирование устройств на базе FT232

Рассмотренные выше функции также могут быть использованы для устройств, в которых имеются микросхемы FT245 (преобразователь USB - FIFO) и FT232 (преобразователь USB - UART). Однако для устройств на базе FT232 имеется ряд дополнительных функций, позволяющих в полном объеме реализовать возможности универсального асинхронного приемо-передатчика. Это функции, которые управ-

ляют скоростью передачи данных, форматом передаваемых байт, состоянием линий DTR и RTS.

После открытия устройства, необходимо установить скорость передачи данных. Для этого используется функция FT_SetBaudRate, прототип которой имеет вид:

```
FT_STATUS FT_SetBaudRate(
FT_HANDLE ftHandle,
DWORD dwBaudRate
);
```

Первый параметр, передаваемый в функцию (*ftHandle*) – дескриптор открытого устройства. Второй параметр (*dwBaudRate*) определяет скорость передачи данных и может принимать одно из следующих значений: FT_BAUD_300, FT_BAUD_600, FT_BAUD_1200, FT_BAUD_2400, FT_BAUD_4800, FT_BAUD_9600, FT_BAUD_14400, FT_BAUD_19200, FT_BAUD_38400, FT_BAUD_57600, FT_BAUD_115200, FT_BAUD_230400, FT_BAUD_460800, FT_BAUD_921600.

Для настройки формата передаваемых байт используется функция FT_SetDataCharacteristics. Ее прототип имеет вид:

```
FT_STATUS FT_SetDataCharacteristics(
FT_HANDLE ftHandle,
UCHAR uWordLength,
UCHAR uStopBits,
UCHAR uParity
);
```

Параметры функции имеют следующее значение:

ftHandle – дескриптор открытого устройства;

uWordLength – количество бит в слове – должно быть FT_BITS_8 или FT_BITS_7;

uStopBits – количество стоповых бит – должно быть FT_STOP_BITS_1 или FT_STOP_BITS_2;

uParity – определяет бит четности; возможные значения:

FT_PARITY_NONE – нет бита четности;

FT_PARITY_ODD – проверка нечетности;

FT_PARITY_EVEN – проверка четности;

FT_PARITY_MARK – бит четности всегда равен 1;

FT_PARITY_SPACE – бит четности всегда равен 0.

Для изменения состояний управляющих сигналов используются следующие четыре функции:

FT_STATUS FT_SetDTR(FT_HANDLE *ftHandle*) – установка сигнала DTR;

FT_STATUS FT_ClrDTR(FT_HANDLE *ftHandle*)– сброс сигнала DTR;

FT_STATUS FT_SetRTS(FT_HANDLE *ftHandle*) – установка сигнала RTS;

FT_STATUS FT_ClrRTS(FT_HANDLE *ftHandle*) – сброс сигнала RTS.

Единственные параметр, передаваемый в эти функции – дескриптор устройства.

Функция FT_GetModemStatus позволяет определить состояние сигналов CTS и DSR. Ее прототип имеет вид:

```
FT_STATUS FT_GetModemStatus(
FT_HANDLE ftHandle,
LPDWORD lpdwModemStatus
);
```

Параметр *lpdwModemStatus* – указатель на переменную типа DWORD, в которой возвратится значение, показывающее состояние указанных сигналов. Для определения состояния линий DSR и CTS может использоваться следующая конструкция (предполагается, что устройство уже открыто, а fth – дескриптор):

```
DWORD Status;
FT_GetModemStatus(ftHandle,&Status);          if (Status & 0x00000010) {
// CTS имеет высокий уровень
}
else {
// CTS имеет низкий уровень
}
if (Status & 0x00000020) {
// DSR имеет высокий уровень
}
else {
// DSR имеет низкий уровень
}
```

Во многих устройствах информационный поток от устройства в компьютер является неравномерным. Так, данные могут поступать с разными интервалами времени и циклический опрос устройства будет неэффективным. В этих случаях целесообразней использовать механизм ожидания событий. Создав рабочий поток, его можно заблокировать до тех пор, пока не возникнет определенное событие, связанное с устройством, а при возникновении события, главной программе передать сообщение. При этом вызовется некоторая функция, в которой следует считывать данные из устройства.

Для установки условий уведомления при возникновении событий используется функция FT_SetEventNotification, прототип которой имеет вид:

```
FT_STATUS FT_SetEventNotification(
FT_HANDLE ftHandle,
DWORD dwEventMask,
```

PVOID pvArg

);

Описание параметров функции:

ftHandle – дескриптор устройства, для которого устанавливаются условия уведомлений при возникновении событий;

dwEventMask – маска, которая определяет условия установки события; этот параметр может принимать любую комбинацию из следующих значений:

FT_EVENT_RXCHAR – в очереди приема есть данные;

FT_EVENT_MODEM_STATUS – изменение сигналов состояния модема (CTS, DSR).

pvArg – дескриптор объекта события.

В приложении эта функция может использоваться для установки условий, которые блокируют поток до тех пор, пока они не возникнут. Обычно, в приложении создается событие, вызывается эта функция, а затем блокируется поток. Для создания события используется следующий фрагмент кода:

```
HANDLE hEvent;     // дескриптор события
hEvent = CreateEvent(NULL, false, false, ""); // создание события
```

Для блокировки потока до наступления определенного события используется функция

```
WaitForSingleObject(hEvent, INFINITE);
```

Первым параметром передается дескриптор события. Второй параметр определяет время, в течении которого поток будет заблокирован если событие не наступит. В приведенном примере это бесконечность.

Пример 5.

Используя среду разработки Visual Studio .NET, создать программу, которая выполняет прием и отображение значения некоторого

технологического параметра. Данные принимаются через USB и должны отображаться в виде графика. В устройстве использована микросхема FT232 (преобразователь USB – UART). Скорость обмена данными равна 115200 бит/с. В программе использовать рабочий поток, в котором ожидается событие приема данных в устройство.

Пояснение. Порядок действий.

1. Запускаем среду разработки Visual Studio .NET. Создаем новый проект с названием USBTest. Приложение создается на базе диалогового окна.

2. Создаем интерфейс программы. Помещаем на диалоговое окно элементы управления как показано на рис.6.23

3.

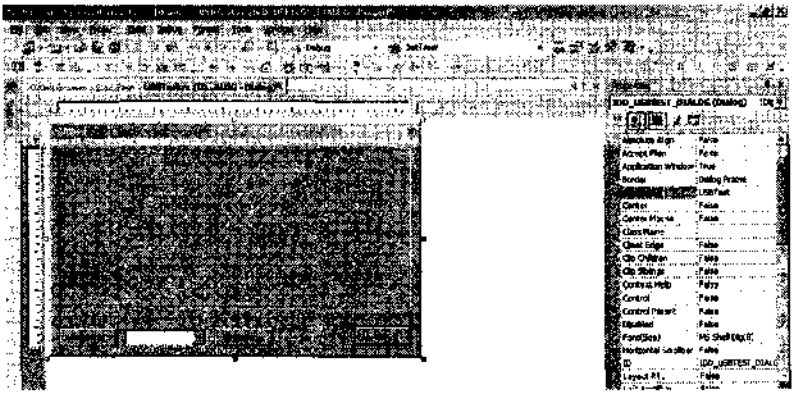


Рис.6.23. Окно программы

4. С элементом управления «Static Text» связываем переменную с именем m_graph, предварительно изменив ID этого элемента управления на IDC_STATIC1. К элементу управления «Combo Box» привязываем переменную с именем m_devices. Добавляем в класс CUSBTestDlg переменную-член класса типа FT_HANDLE с именем m_hUSB; переменную типа CByteArray с именем m_data, в которой

будут сохраняться принятые данные. В начале файла OSBTestDlg.h, для подключения библиотеки FTD2XX, необходимо добавить следующие две строки:

```
#include "FTD2XX.h"    // подключение заголовочного файла
#pragma comment(lib,"FTD2XX.lib") // подключение библиотеки
```

Также необходимо скопировать файлы FTD2XX.h, FTD2XX.lib, FTD2XX.dll в директорию, где расположены файлы проекта.

4. Добавляем глобальную переменную типа HANDLE с именем hEvent для отслеживания событий, связанных с устройством, а также глобальную переменную типа FT_DEVICE с именем hUSB. Эта переменная будет использоваться для передачи дескриптора открытого устройства в функцию потока. Также необходимо определить пользовательское сообщение. Для этого в начале файла USBTestDlg.cpp, после секции #include, добавляем строки:

```
volatile FT_HANDLE hUSB;  // глобальные переменные
HANDLE hEvent;
#define WM_DATARECIEVED WM_USER+1
```

Для остановки работы потока необходимо знать указатель на него. В библиотеке MFC для работы с потоками используется класс CWinThread. В класс CUSBTestDlg необходимо добавить переменную-член класса:

```
CWinThread* m_thread;
```

При создании потока в этой переменной будет храниться указатель на него.

3. Добавляем функцию рабочего потока, в котором будет ожидаться прием сообщения. Эта функция не является членом класса диалогового окна CUSBTestDlg. Ее описание следует расположить в файле USBTestDlg.cpp, перед началом описания обработчиков событий нажатия на кнопку. Функция выглядит следующим образом:

```
UINT ThreadFunction(LPVOID param)
{
  WaitForSingleObject(hEvent,INFINITE);   // ожидание события
  // посылка сообщения главной программе
  ::SendMessage((HWND)param, WM_MESSRECIEVED, 0, 0);
  return 0;
}
```

Также необходимо указать, что при появлении сообщения WM_DATARECIEVED будет вызвана функция OnRecieveData. Для этого карту сообщений нужно изменить следующим образом:

```
BEGIN_MESSAGE_MAP(CUSBTestDlg, CDialog)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    //}}AFX_MSG_MAP
    ON_BN_CLICKED(IDC_BUTTON1, OnBnClickedButton1)
    ON_BN_CLICKED(IDC_BUTTON2, OnBnClickedButton2)
    ON_MESSAGE(WM_DATARECIEVED, OnRecieveData)
END_MESSAGE_MAP()
```

Описание карты сообщений находится в начале файла USBTestDlg.cpp.

5. Создаем функцию – обработчик сообщения WM_DATARECIEVED, которое передается главному окну программы из рабочего потока при приеме данных. Эта функция является членом класса диалогового окна. В окне «Class View» следует выделить класс CUSBTestDlg и из контекстного меню выбрать команду «Add Function». Появится диалоговое окно, в котором необходимо указать название функции, а также тип и имена параметров, передаваемых в функцию. Вид этого окна представлен на рис.6.24.

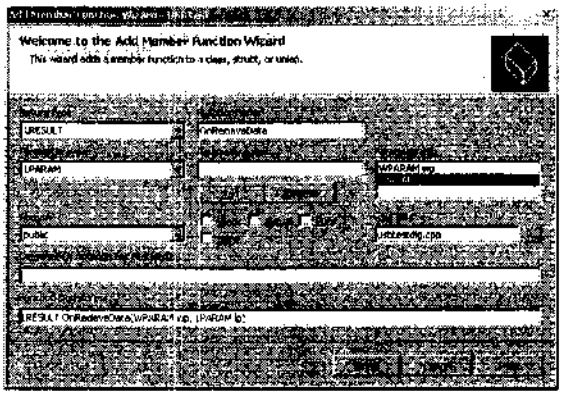


Рис.6.24. Окно добавления функции

Ниже приводится реализация функции:

```
LRESULT CUSBTestDlg::OnRecieveData(WPARAM wp, LPARAM lp)
{
   DWORD size,rsize;
   char buff[256];
   FT_STATUS fts;
   // получение количества байт в очереди приема
   fts=FT_GetQueueStatus(m_hUSB,&size);
   if(fts!=FT_OK)
   {
         MessageBox("Неизвестное количество байт в очереди приема!");
         return 0;
   }
   fts=FT_Read(m_hUSB, buff, size, &rsize); // чтение данных
   if((fts!=FT_OK) || (rsize<1))
   {
         MessageBox("Ошибка при чтении данных из устройства!");
         return 0;
   }
   // добавление данных в массив
   for(int i=0;i<rsize;i++)   m_data.Add(buff[i]);
// очистка очередей приема и передачи
   FT_Purge(m_hUSB, FT_PURGE_RX | FT_PURGE_TX);
   ResetEvent(hEvent);      // сброс события
   DrawGraph();      // рисование графика
```

```
m_thread=AfxBeginThread(ThreadFunction, m_hWnd,
THREAD_PRIORITY_NORMAL);    // запуск функции потока
  return 1;
                                    }
```

6. В конце функции OnInitDialog класса CUSBTestDlg добавляем следующий код:

```
FT_STATUS fts;
DWORD num;
// получение количества подключенных устройств
fts=FT_ListDevices(&num, NULL, FT_LIST_NUMBER_ONLY);
if(fts!=FT_OK)
{
MessageBox("Невозможно обнаружить устройство!");
return FALSE;
}
for(int i=0;i<num;i++)
{
char desc[16];
fts=FT_ListDevices((PVOID)i, desc, FT_LIST_BY_INDEX |
FT_OPEN_BY_DESCRIPTION); // получение описания каждого из
// подключенных устройств
if(fts==FT_OK)
{
CString str(desc);
m_devices.AddString(str);
}
else MessageBox("Устройство не найдено!");
}
m_devices.SetCurSel(0);
m_hUSB=NULL;    // присвоение дескрипторам значения NULL
hUSB=NULL;
m_data.RemoveAll();        // очистка мас сива данных
```

7. Обработчик нажатия на кнопку «Открыть»:

```
void CUSBTestDlg::OnBnClickedButton1()
{
FT_STATUS fts;
char desc[16];
m_devices.GetLBText(m_devices.GetCurSel(), desc);
```

```
// открытие выбранного устройства
fts=FT_OpenEx(desc, FT_OPEN_BY_DESCRIPTION, &m_hUSB);
if(fts!=FT_OK) MessageBox("Невозможно открыть выбранное устрой-
ство!");
else
{
MessageBox("Устройство открыто!"),
hUSB=m_hUSB; // присвоение дескриптора глобальной переменной
// установка скорости передачи данных
FT_SetBaudRate(m_hUSB, FT_BAUD_115200);
// установка формата посылаемых байт
FT_SetDataCharacteristics(m_hUSB, FT_BITS_8, FT_STOP_BITS_1,
FT_PARITY_ODD);
// создание события
hEvent=CreateEvent(NULL, false, false, "");
// очистка очередей приема и передачи
FT_Purge(m_hUSB, FT_PURGE_RX | FT_PURGE_TX);
// установка маски событий
fts=FT_SetEventNotification(m_hUSB,FT_EVENT_RXCHAR,hEvent);
if(fts!=FT_OK) MessageBox("Ошибка при установке маски событий!");
// запуск потока
m_thread=AfxBeginThread(ThreadFunction, m_hWnd,
THREAD_PRIORITY_NORMAL);
}
m_devices.EnableWindow(0);
}
```

8. Обработчик нажатия на кнопку «Закрыть»:

```
void CUSBTestDlg::OnBnClickedButton2()
{
  if(m_hUSB!=NULL)
  {
        // остановка потока
        ::TerminateThread(m_thread->m_hThread,0);
        FT_Close(m_hUSB);     // закрытие устройства
        m_hUSB=NULL;
        MessageBox("Устройство закрыто!");
        m_devices.EnableWindow(1);
  }
  else MessageBox("Устройство уже закрыто!");
}
```

9. Реализуем функцию void DrawGraph(void). Для этого ее необходимо сделать членом класса CCOMTestDlg. Реализация функции имеет следующий вид:

```
void CUSBTestDlg::DrawGraph(void)
{
  CClientDC dc(&m_graph);     // создание контекста для рисования
  CPen p(PS_SOLID,2,RGB(200,0,0));// создание пера
  CPen pp(PS_DOT,1,RGB(0,0,0));
  CRect rect;
  CRgn rgn;
  CPen* op;
  m_graph.RedrawWindow();    // преерисовка области рисования
  m_graph.GetClientRect(&rect);        // получение размеров области
рис-я
  rgn.CreateRectRgn(0,0,rect.Width(),rect.Height());
  op=dc.SelectObject(&pp);
  for(int x=20;x<rect.Width();x=x+20)
  {
        // рисование вертикальных линий сетки
        dc.MoveTo(x,0);
        dc.LineTo(x,rect.Height());
  }
  dc.SetTextColor(RGB(50,50,50));     // установка цвета текста
  dc.SetBkColor(RGB(210,210,210));  // установка цвета фона
  for(int y=20;y<rect.Height();y=y+20)
  {
        // рисование горизонтальных линий сетки
        dc.MoveTo(0,rect.Height()-y);
        dc.LineTo(rect.Width(),rect.Height()-y);
        CString s;
        s.Format("%d",y);
// вывод текстовых меток
        dc.TextOut(rect.Width()-30,rect.Height()-y,s);
  }
  dc.SelectObject(&p);
  dc.SelectClipRgn(&rgn);
// если в массиве данных больше, чем размер области рисования,
// очищается массив с данными
  if(rect.Width()<=m_data.GetCount()) m_data.RemoveAll();
```

```
    for(int i=1;i<m_data.GetCount();i++)
    {
        // рисование графика
        dc.MoveTo(i,rect.Height()-m_data.GetAt(i));
        dc.LineTo(i-1,rect.Height()-m_data.GetAt(i-1));
    }
    dc.SelectObject(op);
}
```

Подключив к компьютеру устройство и откомпилировав проект, на экране появится диалоговое окно, вид которого представлен на рис.6.25.

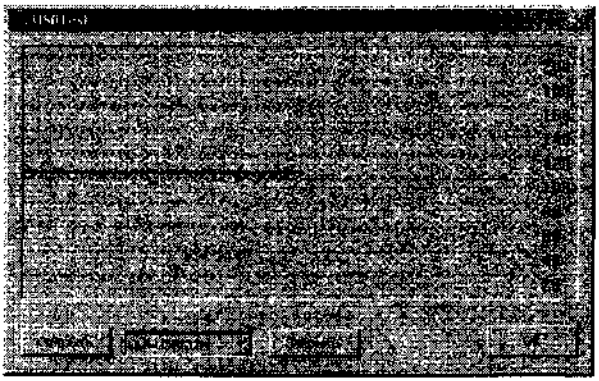


Рис.6.25. Окно программы

6.7. Программирование EEPROM

В библиотеке D2XX имеется несколько функций для программирования EEPROM. Для записи данных в EEPROM используется функция FT_EE_Program, прототип которой имеет вид:

```
FT_STATUS FT_EE_Program(
FT_HANDLE ftHandle,
PFT_PROGRAM_DATA lpData
);
```

Первый параметр (*ftHandle*) – дескриптор устройства. Второй параметр интерпретируется как указатель на структуру FT_PROGRAM_DATA, в которой содержатся данные для записи в EEPROM. Данные, записанные в память, затем считываются и проверяются. При успешном выполнении функции возвращается значение FT_OK. Для считывания данных из EEPROM используется функция FT_EE_Read, прототип которой такой же, как и у функции FT_EE_Program.

Структура FT_PROGRAM_DATA имеет следующие поля:

```
WORD VendorId;  // ID поставщика
WORD ProductId; // ID устройства
char *Manufacturer; // Производитель
char *ManufacturerId; // ID Производителя
char *Description;  // Описание устройства
char *SerialNumber; // Серийный номер
WORD MaxPower;  // 0 < Максимальная мощность <= 500
WORD PnP;  // опция Plug&Play: 0-отключена, 1–включена
WORD SelfPowered; // Собственное питание: 0-питание от шины, 1-автономное питание
WORD RemoteWakeup; // Дистанционное пробуждение: 0-не способно, 1-способно
// Rev4 расширения
```

bool Rev4; // истина, если микросхема Rev4, ложь в противном случае

bool IsoIn; // истина если входящая конечная точка изохронная

bool IsoOut; // истина если выходящая конечная точка изохронная

bool PullDownEnable; // истина если подтяжка включена

bool SerNumEnable; // истина если использован серийный номер

bool USBVersionEnable; // истина если чип использует USBVersion

WORD USBVersion; // версия USB (0x0200 => USB2)

Не используемое пространство памяти EEPROM называется пользовательской областью (User Area). Размер этой области можно определить с помощью функции FT_EE_UASize:

FT_STATUS FT_EE_UASize(
FT_HANDLE *ftHandle*,
LPDWORD *lpdwSize*
);

Параметры, передаваемые в функцию – это дескриптор открытого устройства и указатель на переменную типа DWORD, в которой возвратится размер пользовательской области EEPROM.

Для записи данных в память используется функция FT_EE_UAWrite:

FT_STATUS FT_EE_UAWrite(
FT_HANDLE *ftHandle*,
PUCHAR *pucData*,
DWORD *dwDataLen*
);

Параметры функции имеют следующее значение:

ftHandle – дескриптор устройства;

pucData – указатель на буфер, в котором хранятся записываемые данные;

dwDataLen – размер буфера (в байтах).

Для считывания данных из буфера используется функция FT_EE_UARead:

```
FT_STATUS FT_EE_UARead(
FT_HANDLE ftHandle,
PUCHAR pucData,
DWORD dwDataLen,
LPDWORD lpdwBytesRead
);
```

Параметры, передаваемые в функцию:

ftHandle – дескриптор устройства;

pucData – указатель на буфер, в котором будут храниться считанные данные;

dwDataLen – количество байт для считывания. Значение этого параметра должно быть равно размеру буфера. Максимальное значение, которое может принимать этот параметр – размер пользовательской области EEPROM;

lpdwBytesRead – указатель на переменную типа DWORD, в которой будет храниться количество считанных байт.

Для того, чтобы стереть все содержимое памяти EEPROM, используется функция FT_EraseEE. Единственным параметром, передаваемым в эту функцию, является дескриптор устройства:

```
FT_STATUS FT_EraseEE(FT_HANDLE ftHandle);
```

При вызове этой функции также стирается содержимое пользовательской области EEPROM. В следующем примере демонстрируются принципы работы с EEPROM.

Пример 6.

Написать программу, которая по нажатию на кнопку считывает данные из EEPROM и выводит сообщение с полученной информацией. Также должна быть возможность чтения (записи) данных из пользовательской области.

Пояснение. Порядок действий.

1. Запускаем среду разработки Visual Studio .NET. Создаем новый проект с названием EEPROMTest. В нашем примере приложение создается на базе диалогового окна.

2. Проектируем внешний вид диалогового окна. Помещаем на форму кнопки и элемент управления «Текстовое поле», в которое будет вводиться информация для записи в EEPROM, а также отображаться считанные данные (рис.6.26).

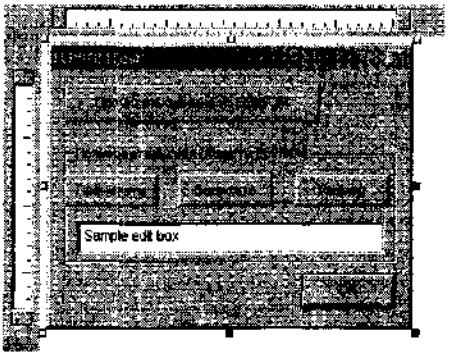


Рис.6.26. Окно программы

С элементом управления «Текстовое поле» (Edit box) необходимо связать переменную, имя которой m_text.

3. В каталог, где размещаются файлы созданного проекта, скопировать следующие файлы:

FTD2XX.h

FTD2XX.lib

FTD2XX.dll

В начале файла EEPROMTestDlg.cpp, для подключения библиотеки для работы с USB необходимо добавить следующие две строки:

```
#include "FTD2XX.h"
#pragma comment(lib,"FTD2XX.lib")
```

4. Добавляем обработчик нажатия на кнопку «Прочитать данные из EEPROM». Функция имеет следующий вид:

```
void CEEPROMTestDlg::OnBnClickedButton1()
{
  FT_HANDLE fth;
  FT_STATUS fts = FT_Open(0,&fth); // открытие устройства
  if (fts != FT_OK) {
  MessageBox("Невозможно открыть устройство!");
  return;
  }
  FT_PROGRAM_DATA ftData;
  // объявление буферов для хранения данных
  char ManufacturerBuf[32];
  char ManufacturerIdBuf[16];
  char DescriptionBuf[64];
  char SerialNumberBuf[16];
  ftData.Manufacturer = ManufacturerBuf;
  ftData.ManufacturerId = ManufacturerIdBuf;
  ftData.Description = DescriptionBuf;
  ftData.SerialNumber = SerialNumberBuf;
  fts = FT_EE_Read(fth,&ftData);       // чтение данных из EEPROM
  if (fts == FT_OK)
  {
// формирование строк
        CString s("Изготовитель: ");
```

```
        CString s1(s+ManufacturerBuf);
        s="ID изготовителя: ";
        CString s2(s+ManufacturerIdBuf);
        s="Описание: ";
        CString s3(s+DescriptionBuf);
        s="Серийный номер: ";
        CString s4(s+SerialNumberBuf);
        MessageBox(s1+'\n'+s2+'\n'+s3+'\n'+s4);
    }
    else MessageBox("Ошибка при чтении данных из EEPROM!");
    FT_Close(fth);      // закрытие устройства
}
```

5. Добавляем обработчик нажатия на кнопку «Прочитать»:

```
void CEEPROMTestDlg::OnBnClickedButton2()
{
    FT_HANDLE fth;
    FT_STATUS fts = FT_Open(0,&fth); // открытие устройства
    if (fts != FT_OK)
    {
    MessageBox("Невозможно открыть устройство!");
    return;
    }
    char buff[38]="";
    DWORD BytesRead;
// чтение из пользоваетльской области EEPROM
    fts=FT_EE_UARead(fth,(PUCHAR)buff,38,&BytesRead);
    if(fts!=FT_OK) MessageBox("Ошибка чтения EEPROM!");
    CString str(buff);
    m_text.SetWindowText(str);
    FT_Close(fth);      // закрытие устройства
}
```

6. Добавляем обработчик нажатия на кнопку «Записать»:

```
void CEEPROMTestDlg::OnBnClickedButton3()
{
    FT_HANDLE fth;
    FT_STATUS fts = FT_Open(0,&fth); // открытие устройства
    if (fts != FT_OK)
    {
```

```
MessageBox("Невозможно открыть устройство!");
  return;
  }
  char buff[38];
  m_text.GetLine(0,buff);
  // запись данных в пользовательскую область EEPROM
  fts=FT_EE_UAWrite(fth,(PUCHAR)buff,strlen(buff));
  if(fts!=FT_OK) MessageBox("Ошибка записи EEPROM!");
  m_text.SetWindowText("");
  FT_Close(fth);     // закрытие устройства
}
```

7. Добавляем обработчик нажатия на кнопку «Размер»:

```
void CEEPROMTestDlg::OnBnClickedButton4()
{
  FT_HANDLE fth;
  FT_STATUS fts = FT_Open(0,&fth);
  if (fts != FT_OK)
  {
  MessageBox("Невозможно открыть устройство!");
  return;
  }
DWORD size;
 // получение данных о размере пользовательской области
  fts = FT_EE_UASize(fth,&size);
  if (fts == FT_OK)
  {
  CString str;
  // формирование строки
  str.Format("Размер пользовательской области EEPROM: %d",size);
  MessageBox(str);  // вывод сообщения
  }
  else MessageBox("Невозможно получить информацию о EEPROM!");
  FT_Close(fth);     // закрытие устройства
}
```

После компиляции проекта и запуска программы можно проверить ее работу, подключив к USB устройство. При нажатии на кнопку «Прочитать данные из EEPROM» появится сообщение, показанное на рис.6.27.

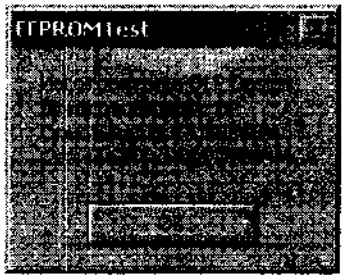


Рис.6.27. Информация из EEPROM

Можно узнать объем пользовательской области EEPROM, нажав на кнопку «Размер» (рис.6.28).

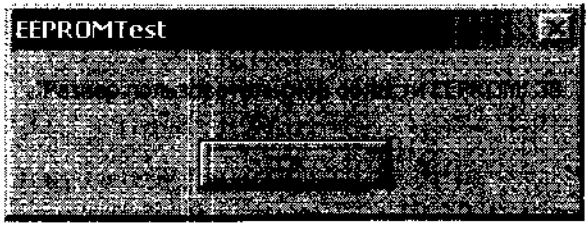


Рис.6.28.

Набрав в текстовом поле какую либо строку и нажав на кнопку «Записать», эти данные запишутся в пользовательскую область, откуда их можно получить нажатием на кнопку «Прочитать» (Рис.6.29).

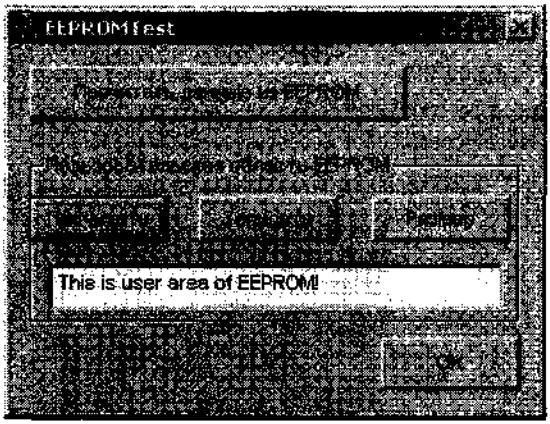


Рис.6.29. Результат выполнения программы

Приведенный пример достаточно простой и демонстрирует принципы работы с EEPROM. Данные, которые сохраняются в памяти, могут представлять собой какие-либо настройки устройства, коэффициенты, которые могут быть считаны и изменены в процессе работы.

6.8. Коды ошибок при работе с USB

Большинство функций возвращают значение типа FT_STATUS. При успешном выполнении функции возвращается значение FT_OK. В противном случае возвращается код ошибки. Ниже приводится список возможных ошибок:

FT_INVALID_HANDLE – в функцию передан неправильный дескриптор;

FT_DEVICE_NOT_FOUND – устройство не обнаружено;

FT_DEVICE_NOT_OPENED – устройство не удалось открыть;

FT_IO_ERROR – ошибка ввода-вывода;

FT_INSUFFICIENT_RESOURCES – недостаточно ресурсов;

FT_INVALID_PARAMETER – переданы неправильные параметры;

FT_INVALID_BAUD_RATE – неправильная скорость передачи данных;

FT_DEVICE_NOT_OPENED_FOR_ERASE – устройство не открыто для стирания;

FT_DEVICE_NOT_OPENED_FOR_WRITE – устройство не открыто для записи;

FT_FAILED_TO_WRITE_DEVICE – невозможно записать данные в устройство;

FT_EEPROM_READ_FAILED – ошибка чтения EEPROM;

FT_EEPROM_WRITE_FAILED – ошибка записи EEPROM;

FT_EEPROM_ERASE_FAILED – ошибка стирания EEPROM;

FT_EEPROM_NOT_PRESENT – EEPROM отсутствует;

FT_EEPROM_NOT_PROGRAMMED – EEPROM не программируется;

FT_INVALID_ARGS – неправильные аргументы;

FT_OTHER_ERROR – неизвестная ошибка.

7. ОБЗОР ПРОГРАММНЫХ СРЕДСТВ ДЛЯ РАБОТЫ С ПОРТАМИ

7.1. Proteus

Симулятор электронных устройств Proteus позволяет моделировать системы достаточно большой сложности, в составе которых имеется один и более микроконтроллеров. Proteus поддерживает большое количество микроконтроллеров: AVR, PIC16, PIC18, Philips ARM7, Motorola MC68HC11, Zilog, а также огромную библиотеку электронных компонентов и электромеханических устройств. Кроме того, имеются компоненты, которые позволяют созданному виртуальному устройству (модели) подключиться к реальным портам компьютера. К таким компонентам относятся COMPIM – позволяет подключиться к COM-порту, LPTPIM – позволяет подключиться к LPT-порту. Компонент для USB уже анонсирован и скоро будет доступен. Например, можно подключить к компьютеру с помощью кабеля мобильный телефон и отлаживать устройство на микроконтроллере, которое должно управлять им, или подключить к порту любое реальное устройство, которым созданная модель будет управлять или обмениваться данными.

На рис.7.1 представлена рабочая область симулятора Proteus. Интерфейс Proteus не имеет линеек прокрутки. Для навигации по рабочей области удобно использовать инструменты масштабирования. Масштаб изображения можно менять с помощью колеса на мышке или с помощью инструментов "лупа +" и "лупа –" в верхней панели инструментов.

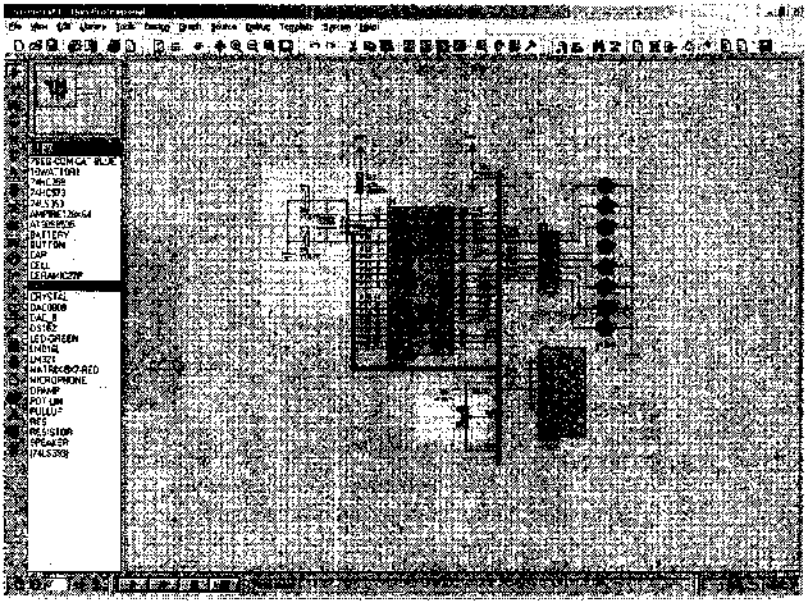


Рис.7.1. Интерфейс программы Proteus

В левой верхней области экрана изображен мини макет страницы и чуть ниже панель DEVICES (компоненты проекта). В этом окне отображаются все элементы, используемые в схеме. Кнопка с буквой "P" открывает форму поиска компонента в библиотеках PROTEUS для добавления в схему (рис.7.2).

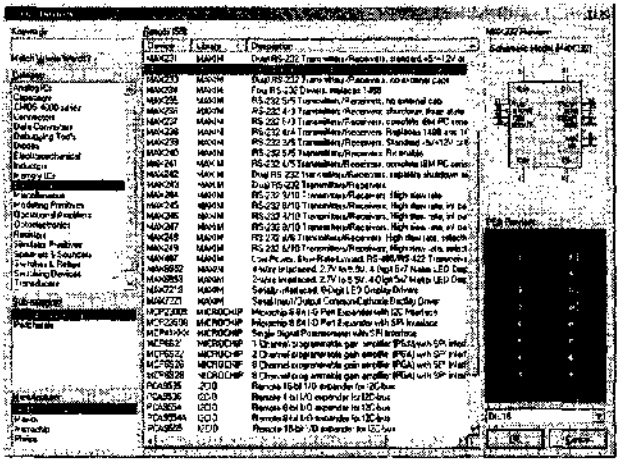


Рис.7.2. Окно поиска компонентов

При вводе в поле "Keywords" (ключевые слова) названия компонента отобразится весь список доступных элементов, в названии которых присутствуют введенные символы. Поместить компонент на схему можно, нажав на кнопку "ОК". После этого следует поместить указатель мыши в нужное место на листе схемы и нажать левую кнопку мыши. Компонент окажется на схеме. Выделить компонент или любой элемент схемы можно путем нажатия по нему правой кнопкой мыши. При этом он станет красным и одновременно будут выделены все проводники, подходящие к компоненту. Если компонент уже имеется в схеме, то его можно скопировать, используя команды «Copy» и «Paste» из меню «Edit». В этом случае придется вручную присвоить ему порядковый номер. Отменить выделение всех выделенных компонентов можно нажатием правой кнопкой мыши в пустом месте схемы. Удалить компонент или любой элемент со схемы можно двойным нажатием правой кнопкой мыши по нему. Ниже приводится описание кнопок панели инструментов.

открыть панель DEVICES - компоненты проекта и поиск новых

поставить точку соединения проводников вручную

дать название проводу – одноименные провода электрически соединены

добавить текст в произвольное место схемы

проложить шину; на схеме жирная темно синяя линия

создать подсхему

нажатие на компоненте сразу открывает окно редактирования его свойств

TERMINALS - питание, земля. межблочные соединения, выводы

добавить вывод к создаваемому компоненту

графическое отображение, сохранение и анализ результатов симуляции

"магнитофон" для записи в файл и воспроизведения данных

генераторы разных сигналов.

указать точку измерения напряжения на проводнике

указать точку измерения тока на проводнике.

Virtual Instruments – измерительные приборы

создание проводников на схеме

Также на панели инструментов находятся кнопки, позволяющие изменять изображение элемента на схеме (поворачивать, зеркально отображать). В нижней части рабочей области находятся четыре кнопки: «Пуск» – запуск симуляции или продолжение приостановленной симуляции, «Шаг» – выполнить минимальный шаг по программе микроконтроллера (обычно это одна инструкция на ассемблере), «Пауза» – пауза симуляции (можно продолжить кнопками «Пуск» или «Шаг»), «Стоп» – остановка симуляции.

На рис.7.3 представлена схема подключения компонента COMPIM к виртуальному терминалу. При запуске процесса моделирования откроется окно терминала. В окне терминала будут отображаться принимаемые символы. Передача данных через COM-порт осуществляется при вводе данных с клавиатуры в окне терминала.

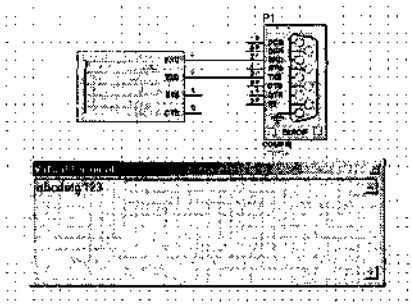


Рис.7.3. Подключение виртуального терминала к COM-порту

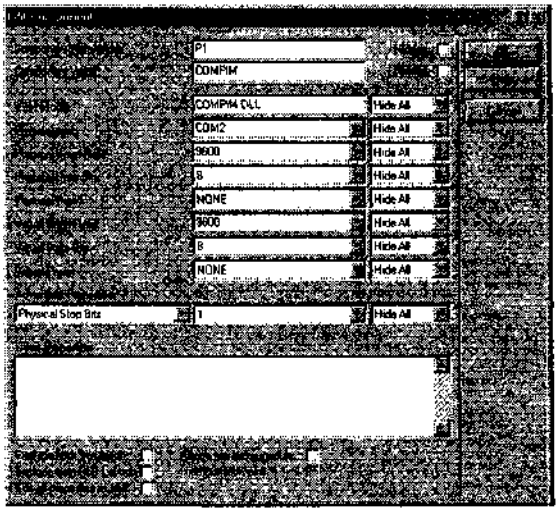


Рис.7.4. Окно настроек компонента COMPIM

В диалоговом окне настроек компонента COMPIM настраиваются такие параметры как номер порта (Physical port), действительная

скорость передачи данных (Physical Baud Rate), количество бит данных (Physical Data Bits), проверка четности. Как правило, компонент COMPIM подключается к микроконтроллеру. Скорость передачи данных в модели устройства может отличаться от скорости передачи данных между компьютером и реальным устройством. Поэтому в настройках компонента также следует указать виртуальную скорость передачи данных (Virtual Baud Rate), количество информационных бит (Virtual Data Bits) и режим проверки четности (Virtual Parity). Также настраивается количество стоповых бит. Компонент COMPIM можно подключать к микроконтроллеру напрямую, без использования микросхемы преобразователя уровней RS232-ТТЛ. На рис.7.5 представлена схема подключения микроконтроллера к COM-порту с помощью элемента COMPIM.

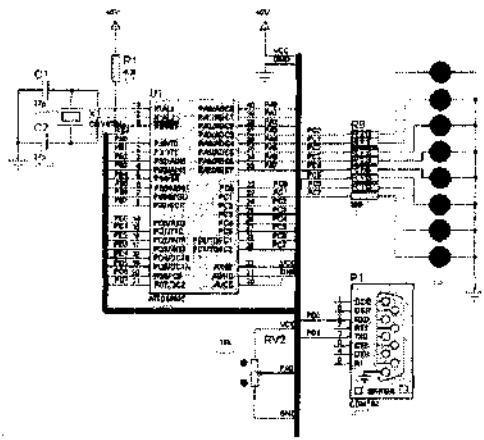


Рис.7.5. Схема подключения микроконтроллера к COM-порту

В окне свойств микроконтроллера (рис.7.6) необходимо указать путь к файлу прошивки (.hex) – программа, по которой он будет работать. Также в этом окне можно указать тактовую частоту работы мик-

роконтроллера, частоту сторожевого таймера. Кварц и конденсаторы не нужны для симуляции, их устанавливают на схему только для того чтобы учесть при разводке печатной платы устройства.

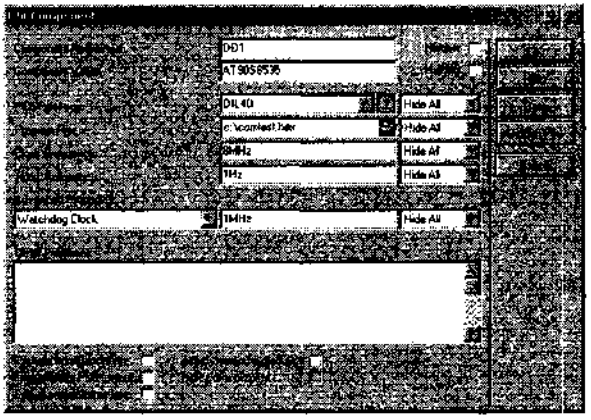


Рис.7.6. Окно настроек микроконтроллера

В модели можно использовать несколько компонентов COMPIM. При этом каждый из них должен иметь различные номера портов.

Принцип использования компонента LPTPIM аналогичный. Пример схемы для работы с LPT - портом показан на рис.7.7.

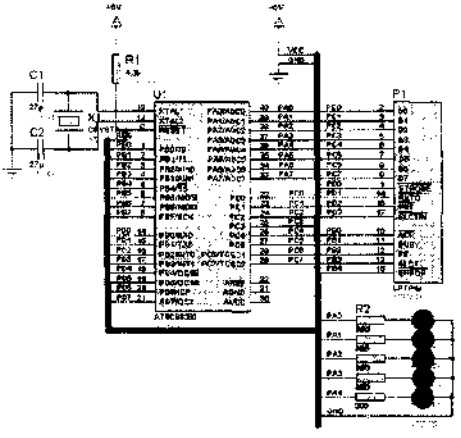


Рис.7.7. Схема подключения микроконтроллера к LPT - порту

На рис.7.8 представлено диалоговое окно с настройками компонента LPTPIM.

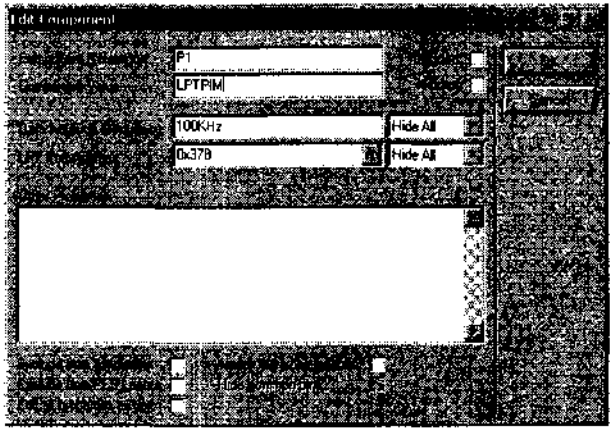


Рис.7.8. Окно настроек компонента LPTPIM

Для настройки компонента LPTPIM необходимо указать два параметра – адрес LPT - порта (LPT Port Address) и частоту, с которой будет обновляться состояние порта (Data Aquision Frequency).

7.2. SCADA-системы

SCADA (Supervisory Control And Data Acquisition) система – это совокупность аппаратно-программных средств, которые обеспечивают возможность мониторинга, анализа и управление параметрами технологического процесса человеком. Она является составной частью автоматизированной системы. На рис.7.9 представленная общая функциональная схема современного производства.

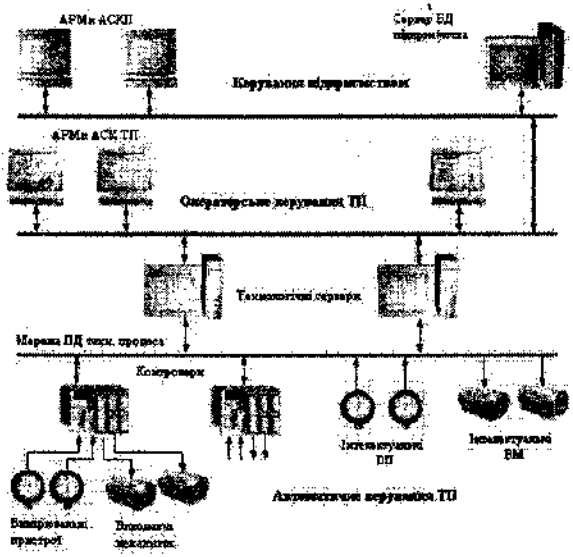


Рис. 7.9. Общая функциональная схема современного производства

Нижний уровень этой схемы составляют измерительные устройства и измерительные механизмы. На сегодняшний день они могут быть аналоговыми или цифровыми (интеллектуальными). Аналоговые представляют измеренную величину в виде уровня напряжения или тока. Цифровые имеют встроенные логические схемы и представляют величину в виде цифрового сигнала, который отвечает спецификации протокола передачи данных, определенного для этих устройств. Для обмена информацией с устройствами первого типа необходимо использовать АЦП/ЦАП (аналогово-цифровые / цифро-аналоговые преобразователи). С устройствами второго типа можно обмениваться информацией непосредственно по сети передачи данных.

Следующий уровень схемы - контроллеры. Они выполняют функцию автоматического управления технологическим процессом. Целью управления являются представления сигналов на исполнитель-

ные механизмы в следствие обработки данных в состоянии технологических параметров, которые получены с помощью измерительных устройств, по определенным алгоритмам.

Серверы технологических данных обеспечивают обмен информацией между технологическими устройствами и сетью персональных компьютеров. Они поддерживают протокол работы с технологическими устройствами и протокол работы с сетью персональных компьютеров.

Данные о текущих параметрах технологического процесса могут быть использованы для контроля состояния технологического процесса и управление им из автоматизированных рабочих мест операторов; для архирования истории изменения технологических параметров; для формирования суммарных отчетных форм с целью представления информации управляющему персоналу. В этой схеме SCADA система представлена серверами технологических данных и автоматизированными рабочими местами операторов.

Подводя итог сказанному выше, отметим функции SCADA систем:

- сбор данных с контроллерного уровня, в том числе на основе стандартных протоколов;
- отображение данных с использованием графических анимированных объектов;
- обработка данных с использованием встроенных языков программирования;
- архирование и хранение данных.

7.2.1. Принцип работы SCADA систем

SCADA-пакеты состоят из нескольких программных блоков: модули доступа и управления, сигнализации, базы данных реального времени, базы данных и модули ввода-вывода аварийных ситуаций.

Главное требование к SCADA-системам - корректная работа в режиме реального времени. При этом главный приоритет при передаче и обработке имеют сигналы, которые поступают от технологического процесса или на него и влияют на его работу. Они имеют приоритет даже больший, чем обращение к диску или действию оператора по перемещению манипулятора "мыш", или свертыванию окон. Для этих целей много пакетов реализованы с использованием операционных систем (ОС) реального времени, однако в последнее время все больше разработчиков разрабатывают свои SCADA-продукты на платформе Microsoft Windows, встраивая в нее подсистемы жесткого реального времени RTX (Real Time Extension). При таком подходе становится возможным использовать Windows как единую ОС при создании многоуровневых систем, привлекать стандартные функции Win32 API и строить интегрированные информационные системы.

Источники данных в системах SCADA могут быть следующими:

- драйверы связи с контролерами. Очень важна надежность драйвера связи. Драйверы должны иметь средства защиты и восстановления данных при сбоях, автоматически сообщать оператора и системе о потере связи, при необходимости подавать сигнал тревоги;
- реляционные базы данных. SCADA-системы поддерживают протоколы, которые не зависят от типа базы данных, благодаря чему в качестве источника данных может выступать большинство популярных СКБД: Access, Oracle, SQL Server и т.д. Такой

подход разрешает оперативно изменять настройки технологического процесса и анализировать его ход за пределами систем реального времени, разнообразными, специально созданными для этого программами;

- программы, которые содержат стандартный интерфейс DDE (Dynamic Data Exchange) или OLE-технологию (Object Linking and Embedding), которая разрешает включать и встраивать объекты. Это дает возможность использовать в качестве источника данных даже некоторые стандартные офисные программы, например Microsoft Excel.

Ввод получаемых и вывод передаваемых данных организованы как система специальных функциональных блоков. Текущая информация о процессе сохраняется в специальных базах ввода-вывода. Входные блоки получают информацию и преобразуют ее к виду, необходимому для дальнейшего анализа и обработки. Блоки обработки реализуют алгоритмы контроля и управления, такие как ПИД-регулирование, задержка, статистическая обработка; над цифровыми данными могут проводиться операции булевой алгебры и т.д. Исходные блоки передают управляющий сигнал от системы к объекту. Для связи с объектами используются широко распространенные интерфейсы RS-232, RS-422, RS-485, Ethernet. Для повышения скорости передачи применяются разнообразные методы кеширования данных, что исключает перегрузку сетей с низкой скоростью. Другими словами, если два разных клиента запрашивают у сервера одни и те же данные, он посылает контролеру не два пакета, а лишь один, возвращая второму клиенту данные из кэш-памяти.

Едва не самый важный момент при создании АСКТВ – это организация такой системы управления, которая обеспечила бы надеж-

ность и оперативную обработку аварийных ситуаций, как в самой системе управления, так и в технологическом процессе. Аварийная сигнализация и обработка аварийных ситуаций в технологическом процессе в большинстве SCADA-систем выделяются в отдельный модуль с высочайшим приоритетом. Надежность системы управления достигается за счет "горячего" резервирования. Можно зарезервировать все: сервер, его отдельные задачи, сетевые соединения и отдельные (или все) связи с аппаратурой. Резервирование проходит по интеллектуальному алгоритму: чтобы не создавать двойную нагрузку на сеть, основной сервер взаимодействует с аппаратурой и периодически посылает сообщение резервному серверу, который сохраняет в памяти текущее состояние системы. Если основной сервер выйдет из строя, резервный берет управление на себя и работает до тех пор, пока основной не начнет работу. Сразу после этого, базы данных основного сервера обновляются данными с резервного и управления возвращается к основному серверу.

Все SCADA-системы открытые для дальнейшего расширения и усовершенствования и имеют для этих целей встроенные языки программирования высокого уровня, чаще всего Basic, или допускают подключения программных кодов, которые написаны пользователем. Кроме этого, к системам можно подключать разработки других фирм, объекты ActiveX, стандартные библиотеки DLL Windows. Для реализации этих технологий разработаны специальные инструментальные средства и специализированный интерфейс.

SCADA-система может быть интегрирована с самыми разными сетями: другими SCADA-системами, офисными сетями предприятия, регистрирующими и сигнализирующими сетями и т.д. Для эффективной работы в этой разнородной среде SCADA-системы используют

стандартные протоколы NETBIOS и TCP/IP. SCADA-системы могут работать и в Интернете, тем более что все более актуальной становится передача оперативной и статистической информации о процессе на Web-узлы.

7.2.2. Система Genie

SCADA система Genie предоставляет удобный графический интерфейс пользователя для создания систем управления технологическим процессом. Для создания систем управления используются редактор форм отображения (Display Designer), в котором создается интерфейс пользователя, и редактор задач (Task Designer), в котором реализуется логика управления. На рис.7.10 представлено диалоговое окно системы Genie и вид интерфейса системы управления некоторым технологическим процессом, в котором использованы различные элементы управления и отображения данных. Связь с внешним оборудованием может осуществляться через порты COM, LPT, а также специальные платы расширения, которые используют шину PCI или ISA.

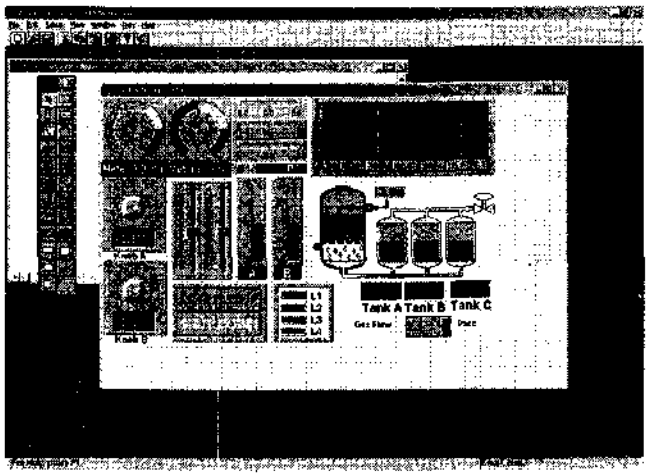


Рис.7.10. Пример интерфейса системы управления в Genie

Рассмотрим элемент управления «Аналоговый регулятор» (рис.7.11).

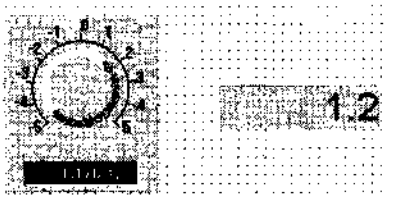


Рис.7.11. Элементы управления «Аналоговый регулятор» и «Цифровой индикатор»

Элемент управления «Аналоговый регулятор» может быть помещен в окно формы отображения и связан с входной переменной функционального блока задачи, входящей в стратегию (Strategy – так называется проект, созданный в Genie), и элементами отображения. Аналоговый регулятор предназначен для ввода оператором числовых значений с помощью клавиатуры или мыши и передачи введенных значений связанному функциональному блоку стратегии, что позволяет реализовывать функции оперативного диспетчерского управления.

Аналоговый регулятор передает связанным функциональным блокам действительные (с плавающей точкой) значения, которые могут отображаться на совмещенном с регулятором цифровом индикаторе. Имеется возможность выбора количества цифр после десятичной точки. Все параметры элемента управления настраиваются в диалоговом окне свойств (рис.7.12).

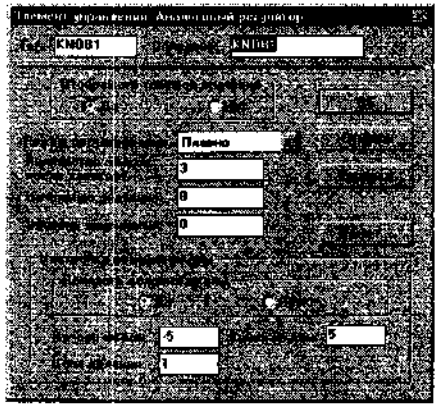


Рис.7.12. Окно свойств элемента управления «Аналоговый регулятор»

Для того, чтобы на цифровом индикаторе отображалось значение, установленное аналоговым регулятором, необходимо отредактировать окно свойств индикатора (рис.7.13).

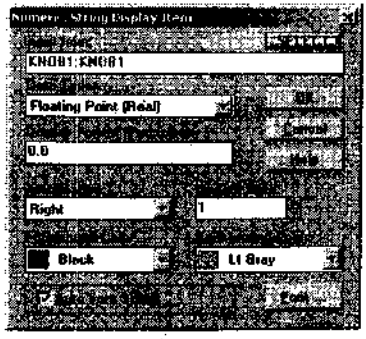


Рис.7.13. Окно настроек «Цифрового индикатора»

В этом окне необходимо указать тип данных, отображаемых индикатором, а также параметры отображения (точность, цвет). Нажав

на кнопку «Select» откроется диалоговое окно (рис.7.14), в котором выбирается источник данных.

Рис.7.14. Окно настройки соединения с источником данных

В списке «Task/Display/Virtual» выбирается редактор, в котором расположен источник данных (в приведенном примере это редактор форм отображения). В поле «TagName» выбирается имя тэга (элемента управления) – KNOB1. Запуск стратегии производится выбором команды «Start» из меню «Run». Перед запуском проект следует сохранить на диск.

Для настройки параметров задачи (Task) из меню «Setup» необходимо выбрать команду «Task Properties». Появится диалоговое окно, показанное на рис.7.15.

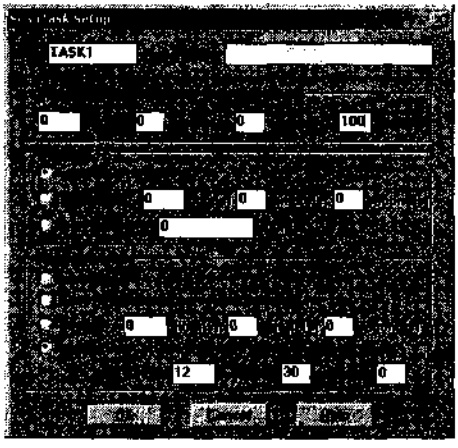


Рис.7.15. Окно настроек задачи

В окне настроек задачи указываются такие параметры, как период обновления данных, режим работы (непрерывный, заданный промежуток времени), способ запуска задачи (сразу после команды Run, по команде из скрипта, в определенное время).

Редактор режимов отображения используется для предоставления пользователю информации о состоянии параметров объекта в различном виде (текст, график), а также средства управления объектом. Организация обмена данными происходит с помощью редактора задач.

Редактор задач содержит блоки для аналогового и дискретного ввода/вывода (для использования этих блоков необходимы специальные платы расширения), клиента и сервера динамического обмена данными (для связи с другими приложениями, например Excel), ПИД-регулятор, цифровой фильтр скользящего среднего, блок архивации данных и другие. Блок обмена через последовательный порт (или блок последовательного интерфейса) предназначен для организации информационного обмена между компьютером, на котором исполняется стратегии GENIE, и другими устройствами и компьютерами, поддерживающими интерфейс последовательной связи RS-232/422/485.

Данные, определяемые полем «Строка для передачи» диалоговой панели настройки параметров блока (рис.7.16), могут быть переданы устройству по последовательному каналу связи. Ответ устройства в виде строки символов может быть передан другому функциональному блоку стратегии. При введении пользователем в поле «Строка для передачи» строки символов на этапе разработки стратегии, указанная строка будет в неизменном виде передаваться устройству, подключенному к последовательному порту компьютера. Кроме того, имеется возможность динамической передачи устройству различных символь-

ных строк при использовании дополнительного функционального блока процедуры пользователя, присоединенного к блоку последовательного интерфейса и формирующего строки, подлежащие передаче.

Двойной щелчок левой клавишей мыши на пиктограмме блока приведен к появлению на экране монитора диалоговой панели настройки его параметров. Для таких параметров, как идентификатор порта, скорость обмена, количество бит данных, количество стоповых бит и контроль по четности используются стандартные соглашения, принятые в вычислительной технике.

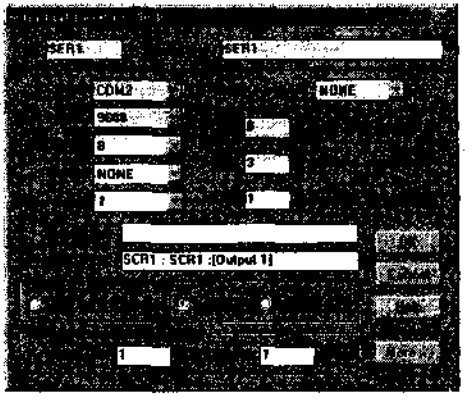


Рис.7.16. Окно настроек блока последовательного интерфейса

Поле «Пауза перед приемом» содержит интервал времени в миллисекундах, по истечении которого GENIE начнет ожидание ответа устройства. Данная возможность является весьма удобной при организации взаимодействия с устройствами, имеющими длительное время реакции на запрос по последовательному каналу связи. Использование паузы перед приемом позволяет избежать возникновения ошибок по тайм-ауту при работе с указанными устройствами.

Поле «Время ожидания ответа» содержит интервал времени в миллисекундах, в течение которого GENIE ожидает ответ устройства по истечении интервала времени, определенного в поле «Пауза перед приемом». Если по истечении времени ожидания ответа не будет получена строка ответа устройства либо в принятой строке будет отсутствовать символ, заданный в поле «Завершающий символ», GENIE выполнит повторную передачу строки, заданной в поле «Строка для передачи». Количество повторов определяется значением в поле «Количество повторов» и может составлять от 0 до 3. Сумма интервалов времени, заданных в полях «Пауза перед приемом» и «Время ожидания ответа», должна быть менее периода опроса задачи, содержащей блок.

Поле «Строка инициализации устройства» может содержать строку символов, которая будет передана устройству для его инициализации один раз при запуске стратегии. При этом ответ устройства не будет передан другим функциональным блокам стратегии, а во всех последующих циклах стратегии в адрес устройства будут передаваться строки, определяемые полем «Строка для передачи».

После передачи строки символов, определяемой полем «Строка для передачи», блок последовательного интерфейса по истечении интервала, заданного полем «Пауза перед приемом», осуществляет ввод строки из приемного буфера последовательного порта и передачу указанной строки присоединенному к выходу блока другому функциональному блоку стратегии. При этом список «Начать с» группы параметров «Значащие символы в принятой строке 0», должен содержать номер символа в принятой строке, начиная с которого будет начато формирование строки, передаваемой блоком последовательного интерфейса другим функциональным блокам стратегии. В списке

«Окончить на» указанной группы параметров должен устанавливаться номер символа в принятой строке, на котором будет завершено формирование строки, передаваемой блоком последовательного интерфейса другим функциональным блокам стратегии.

Таким образом, функциональному блоку, присоединенному к выходу блока последовательного интерфейса, будет передаваться строка, содержащая символы с номерами от «Начать с» до «Окончить на» из строки, принятой от устройства в ответ на запрос блока, определяемый полем «Строка для передачи».

Дополнительные строки, принимаемые от устройства, могут быть определены путем нажатия кнопки «Дополнительно...» диалоговой панели настройки параметров блока и заполнения полей диалоговой панели «Дополнительные принятые строки», показанной ниже.

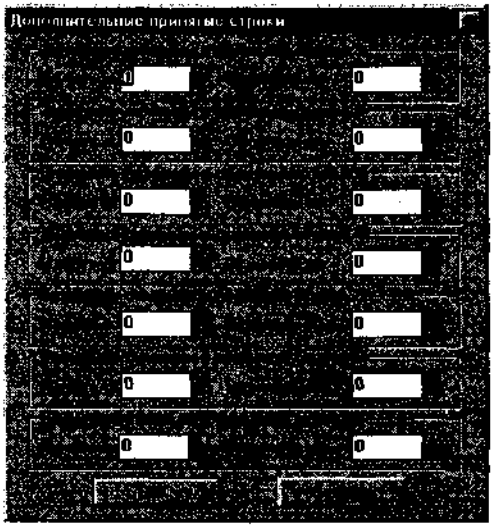


Рис.7.17. Окно настроек приема дополнительных строк

Поля «Строка инициализации устройства» и «Строка для передачи» допускают ввод пользователем управляющих символов в формате ASCII, коды которых должны лежать в диапазоне от 0 до 31.

Например, введенная строка "This ^A^B^C^[String^M" будет преобразована к следующему виду:

"This <Ctrl_A><Ctrl_B><Ctrl_C><ESC>String<CR>",

где <Ctrl_A> ASCII код 1 и <ESC> ASCII код 27.

Таблица 7.1. **Коды для передачи не печатаемых символов**

Код ASCII	Формат ввода	Назначение
0	^@	NULL
1	^A	Начало заголовка
2	^B	Начало текста
3	^C	Конец текста
4	^D	Конец ленты
5	^E	Запрос
6	^F	Подтверждение
7	^G	Звонок
8	^H	Символ забоя (Backspace)
9	^I	Горизонтальная табуляция
10	^J	Перевод строки
11	^K	Вертикальная табуляция
12	^L	Подача бланков (Form Feed)
13	^M	Возврат каретки
14	^N	Сдвиг без сохранения выдвигаемых разрядов (Shift Out)
15	^O	Сдвиг (Shift In)
16	^P	Смена активного канала данных (Data Link Escape)
17	^Q	Device Control 1
18	^R	Device Control 2
19	^S	Device Control 3
20	^T	Device Control 4
21	^U	Знак отрицательного квитирования
22	^V	Синхронизировать
23	^W	Конец передаваемого блока
24	^X	Отмена
25	^Y	Конец носителя
26	^Z	Замена
27	^[	Потеря (Escape)
28	^\	Разделитель файлов
29	^]	Разделитель групп данных
30	^^	Разделитель записей
31	^	Разделитель элементов данных

Вводимая строка преобразуется в строку, передаваемую устройству, по следующим правилам:

1. Символ "^", предшествующий символам от "@" до "_", интерпретируются как младшие 32 управляющих символа таблицы кодов ASCII, приведенной выше.

2. Символ обратной кавычки "`" означает, что все последующие символы будут интерпретироваться как буквы без преобразования в специальные символы. Например, последовательность "`|" представляет код ASCII 124, "`^" будет преобразована в символ "^".

Блок RS-232 имеет вход, позволяющий динамически вводить строку, подлежащую передаче, от другого функционального блока, обеспечивающего возможность вывода символьных строк. Если строка для передачи введена пользователем в соответствующем поле диалоговой панели настройки параметров блока в процессе разработки стратегии, то при каждом вызове задачи, содержащей блок, в последовательный порт будет выводиться только эта строка. Блок RS-232 имеет выход, через который выводится строка, принятая от устройства, подключенного к последовательному порту.

Рассмотрим пример системы управления источником тока. На рис.7.18 представлен вид интерфейса пользователя, который содержит элемент «Аналоговый регулятор» для установки величины тока, индикатор для отображения значения тока и элемент управления «График», с помощью которого отображается осциллограмма тока.

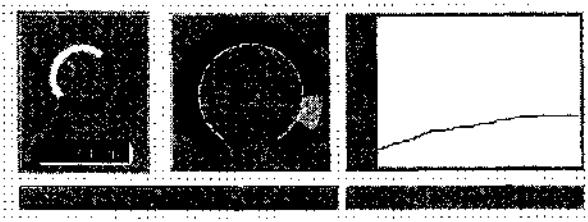


Рис.7.18. Система управления источником тока

При нажатии на кнопку «Установить тою» формируется команда и по последовательному порту отправляется в микропроцессорную систему управления источником тока. На рис.7.19 представлено окно редактора задач, в котором расположены четыре тэга (блока).

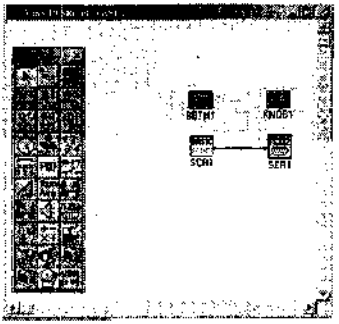


Рис.7.19. Окно редактора задач

Элемент BBTN1 используется для определения состояния кнопки (нажата или не нажата). С помощью элемента KNOB1 данные с аналогового регулятора в редакторе форм отображения передаются в редактор задач. Блок SCR1 представляет собой скрипт, написанный на языке Basic. Этот блок соединен с блоком SER1, с помощью которого выполняется передача данных через последовательный порт. На рис.7.20 представлено диалоговое окно настроек тэга BBTN1.

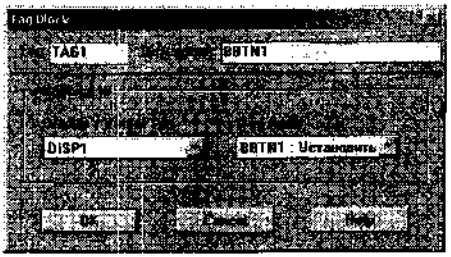


Рис.7.20. Окно настроек тэга BBTN1

В этом окне указывается место расположения кнопки (редактор форм отображения – DISP1), а также имя кнопки (соответствует надписи на кнопке). Аналогичные настройки имеет тэг KNOB1.

Текст программы для блока Basic Script имеет вид:

```
Sub SCR1()          // начало программы
DIM btn as Tag          // объявление переменной btn для определения
// состояния кнопки
DIM knob as Tag  // объявление переменной knob для определения
                 // значения аналогового регулятора
dim str as string    // объявление переменной str – строка
set btn=GetTag("DISP1","BBTN1")  // привязка к переменной тэга
// BBTN1, который передает состояние кнопки
set knob=GetTag("DISP1","KNOB1")// привязка к переменной тэга
// KNOB1, который передает значение аналогового регулятора
dim cval as integer// объявление переменной цело численного типа
cval=knob.value*10      // масштабирование значения,
// установленного аналоговым регулятором
dim cvali as integer     // дополнительная переменная
cvali=cval*10                 // выполняем масштабирование
dim isfix as long        // переменная типа long
isfix=btn.value          // получаем состояние кнопки
dim c1, c2, c3 as string  // дополнительные строковые перем.
dim tmp as integer       // временная переменная
dim command as string  // строка для формир. команды
if isfix<>0 then         // если кнопка нажата, формируем команду
tmp=255                       // запись в переменную значения 255
tmp=tmp And cvali             // операция «И» с маской
c1=Chr(13)               // формирование команды для
c2=Chr(cvali\256)        // передачи в систему управления
c3=Chr(tmp)                   // в виде ASCII-символов
command=c1+c2+c3              // объединение символов в строку
```

```
outputs 1,command          // вывод команды в канал 1
outputf 3,knob.value    // вывод значения аналогового
                        // регулятора в канал 3
end if                  // конец условия
End Sub                       // конец программы
```

В приведенном примере данные с аналогового регулятора поступают в блок в виде числа с плавающей запятой. Для передачи этого значения через последовательный интерфейс, в программе выполняется преобразование в троку символов. Микропроцессорная система управления источником тока после приема команды выполняет преобразование команды и изменяет значение тока. В свою очередь, значение тока поступает в компьютер в виде числа, поэтому необходимости в каких-либо преобразованиях нет. Значение тока сразу отображается с помощью графика.

Приведенный пример показывает особенности использования блока последовательного порта в Genie, а также принципы организации обмена данными между компьютером и внешним устройством.

7.3. Terminal

Программа предназначена для передачи/приема данных по COM-порту. Кроме организации обмена данными имеется возможность изменять значения сигналов на выводах DTR и RTS, а также контролировать состояния линий CTS и DSR. Это можно легко сделать нажатием левой кнопки мыши на стилизированном изображении индикатора сигнала. При этом он изменит свой цвет с серого на зеленый, что соответствует лог.1. На рис.7.21 представлено диалоговое окно программы.

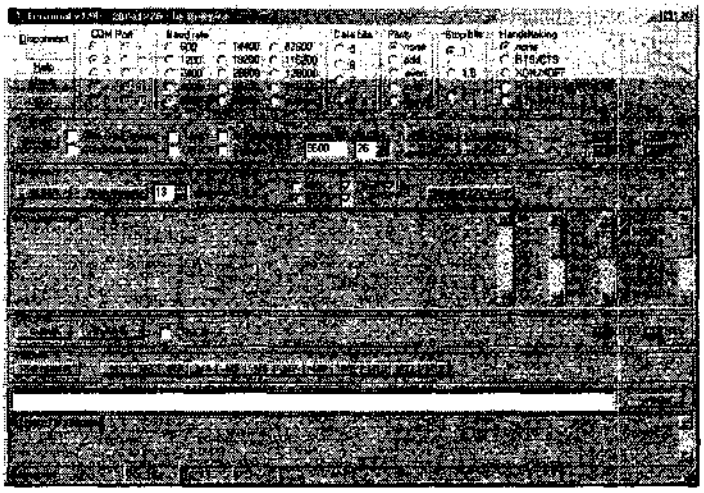


Рис.7.21. Окно программы Terminal

Перед тем, как начать работу с COM-портом, необходимо выполнить некоторые настройки. Прежде всего, следует указать скорость передачи данных. Имеется возможность установки любой из стандартных скоростей. Для этого в группе Baud Rate нужно установить флажок напротив требуемого значения. После этого указать количество передаваемых (принимаемых) бит данных – от 5 до 8, значение бита четности (группа Parity): none – отсутствует, odd – контроль по нечетности, even – контроль по четности, mark – всегда 1, space – всегда 0, а также количество стоповых бит (группа Stop Bits). При организации сложных протоколов обмена данными возможно потребуется механизм подтверждения приема. Для этого в группе Handshaking выбирается тип контроля передачи. После того, как все настройки выполнены, необходимо нажать на кнопку Connect.

Для удобства работы предусмотрена возможность отображения принимаемых данных в различных форматах: десятичном (Dec), ше-

стнадцатиричном (Hex), двоичном (Bin) и ASCII-кодах. Для того чтобы данные отображались в перечисленных форматах необходимо установить флажки Dec, Hex и Bin соответственно. При этом появятся дополнительные области в правой части поля Receive.

Большим преимуществом программы является возможность отображения принимаемых данных в виде графика. К компьютеру может быть подключено какое-либо устройство, выполняющее, например, контроль температуры. Включив режим отображения данных в виде графика можно наблюдать характер изменения принимаемых данных (температуры). Однако ограничение заключается в том, что можно отобразить только один график (рис.7.22).

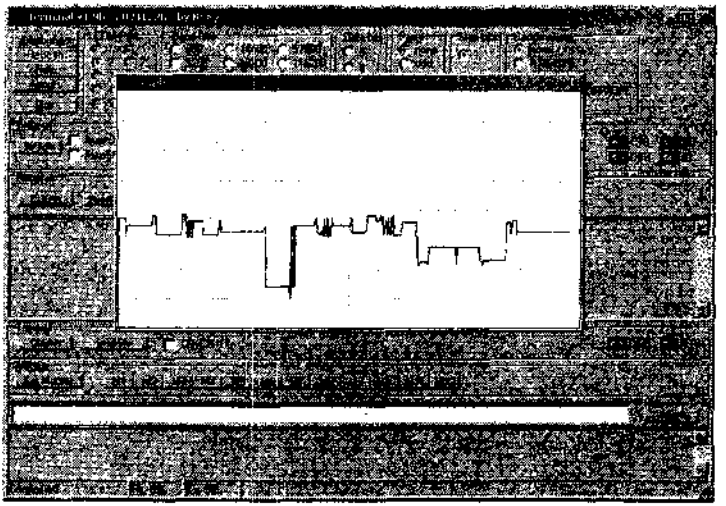


Рис.7.22. Режим отображения данных в виде графика

Передаваемые данные вводятся в поле Transmit. Одновременно с нажатием клавиши символ передается в порт. При необходимости передачи сразу строки символов следует использовать макросы. В поле Macros вводится строка для передачи. При нажатии на кнопку Send

символы из строки передаются в порт. Также с помощью макросов можно организовать более сложные протоколы. При нажатии на кнопку Macros откроется диалоговое окно, показанное на рис.7.23.

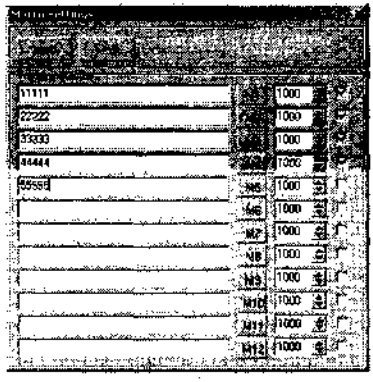


Рис.7.23. Окно настройки макросов

В каждом текстовом поле вводится строка для передачи. Установка флажка напротив поля активизирует макрос. Числовое значение, которое также можно настраивать, соответствует интервалу времени (в миллисекундах), через который указанная строка будет передаваться. После передачи всех указанных строк цикл повторяется. Если флажок напротив поля не устанавливать, то строка будет передана только при нажатии на кнопку с номером макроса (M1 – M12). Для передачи служебных символов вводится символ #, за которым следует ASCII-код. Например, для передачи команды возврата каретки (cariage return) следует ввести строку: #013.

Программа Terminal является свободно распространяемой и доступна в сети Internet по адресу: http://bray.velenje.cx/avr/terminal.

7.4. Winscope

Эта программа позволяет использовать компьютер как осциллограф и анализатор спектра, используя звуковую карту как аналого-цифровой преобразователь. Имеется возможность в режиме реального времени увидеть кривую изменения сигнала, определить его частоту, изучить спектр сигнала, строить фигуры Лиссажу, а также измерять коэффициент взаимной корреляции двух сигналов. Однако программа имеет несколько недостатков – сравнительно низкий диапазон частот (до 20 кГц из-за ограничения звуковой карты), нет возможности калибровки амплитуды. Существует вероятность повреждения звуковой карты, если сигналы на её входе превышают максимально допустимый уровень. Поэтому при подключении каких-либо внешних устройств к звуковой карте компьютера следует принимать меры по ограничению максимальной амплитуды на входе. Безопасно подключать аудио устройства, которые имеют стандартные разъемы.

На рис.7.24 представлено диалоговое окно программы Winscope.

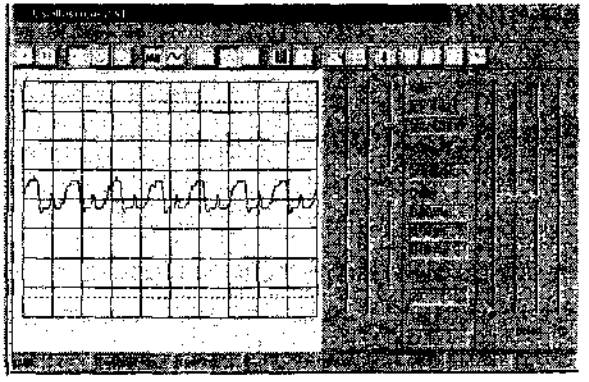


Рис.7.24. Окно программы Winscope

Управление режимами работы программы выполняется с помощью кнопок на панели инструментов. Нажатие на кнопку On Line включает осциллограф. В области построения появится осциллограмма сигнала. Для отображения 2-х каналов одновременно необходимо нажать на кнопку Dual Trace. При наведении курсора мыши на кнопку панели инструментов появляется всплывающая подсказка с информацией о назначении кнопки. Переход в режим анализатора спектра осуществляется нажатием на кнопку FFT. На рис.7.25 представлено окно программы в режиме анализатора спектра.

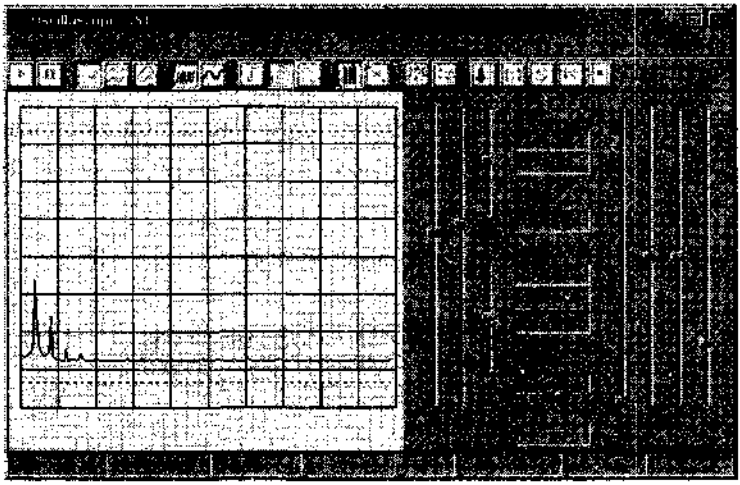


Рис.7.25. Режим анализатора спектра

В строке состояния отображается информация об амплитуде сигнала, а также моменту времени, в который сигнал был измерен. Информация динамически изменяется при перемещении курсора над осциллограммой. В режиме анализатора спектра в строке состояния отображается значение частоты.

С помощью элементов управления в правой части диалогового окна пользователь может установить коэффициент усиления (программный), позицию по вертикали (уровень постоянной составляющей). Ползунок Y1 настраивает левый канал, Y2 – правый. Также можно настроить масштаб времени (в режиме осциллографа) и частоты (в режиме анализатора), уровень напряжения, при котором происходит срабатывание триггера синхронизации. Полученные данные можно сохранить в файл (меню File->Save data). Для настройки частоты дискретизации в меню Options необходимо выбрать команду Timing. Появится диалоговое окно, показанное на рис.7.26.

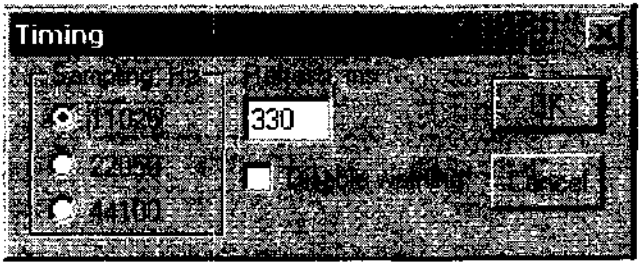


Рис.7.26. Окно настроек частоты дискретизации

Параметр Rsfresh определяет интервал времени, через который будет происходить обновление области построения осцилограммы. Программа Winscope является свободно распространяемой и доступна в сети Internet по адресу: www.freeware.com.

8. ПРИНЦИПЫ ОРГАНИЗАЦИИ СЕТЕВЫХ КОММУНИКАЦИЙ

В большинстве программ используются похожие принципы работы в сетях, которые одинаковые как для Internet, так и для локальных сетей. Одна из программ ожидает, пока другая установит с ней связь по сети. Одновременно другая программа, которая находится на втором компьютере (хотя и не обязательно) пытается установить связь с первой. После установления связи между двумя программами они могут обмениваться сообщениями в прямом и обратном направлении (рис.8.1).

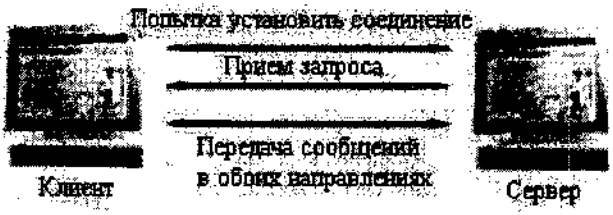


Рис.8.1. Базовая схема процесса соединения

Такое соединение представляет собой двусторонний канал связи, в котором обе стороны могут принимать и передавать информацию. Когда одна из сторон завершает диалог, она разрывает соединение. Другая сторона может обнаружить этот факт и разорвать связь на своем конце. Описанный процесс является иллюстрацией базовой схемы сетевого соединения между двумя или несколькими программами. Такой принцип сетевых коммуникаций используется в рамках протокола TCP/IP, который является основным протоколом связи в сети Internet. Многие протоколы являются вариациями описанной схемы. В некоторых протоколах (например, UDP) не происходит установление канала

связи между программами, а следовательно не контролируется достоверность передачи данных. Одна программа посылает сообщение, а другая программа должна принять все сообщения и выполнить связанные с ними действия.

8.1. Использование Windows Sockets

Спецификацией Windows Sockets (Winsock) определяется интерфейс библиотеки, которая динамически загружается, файл которой, как правило, называется winsock.dll или wsock32.dll. Функции этой библиотеки реализуются разработчиками. Этот интерфейс обеспечил разработчиков программ общим подходом для реализации сетевых коммуникаций, независимо от характера сети, которая используется, и программного обеспечения.

При программировании с использованием Winsock важным понятием является гнездо (socket), которое является базовым объектом, что используется в программах для выполнения большинства сетевых соединений. Каждому компьютеру присваивается числовой адрес, что называется IP-адресом, который записывается в виде четырех чисел, разделенных точками. Пусть, например, один компьютер имеет адрес 192.168.7.62. Программы, которые работают на нем, должны с помощью Winsock соединиться с другим компьютером. Возникает вопрос: если по адресу 192.168.7.62 поступает запрос, какая программа должна его обрабатывать?

Каждый запрос, который поступает в компьютер, содержит номер порта – число от 1024 и выше, которое указывает, какой программе назначен запрос. Некоторые номера портов зарезервированы для стандартного использования. Например, порт 80 традиционно исполь-

зуется серверами Web для получения запросов на документы от таких программ, как Internet Explorer и других.

Работа с Winsock основана на соединении: две программы устанавливают связь с помощью гнезд на каждом конце, а затем передают и принимают данные по этому соединению. Некоторые программы могут передавать данные без установления соединения, но в этом случае нет гарантии, что данные будут переданы по назначению.

Технология гнезд позволяет организовать связь между программами по аналогии с процессами записи и считывания файлов. Однако для организации соединения между программами необходимо наличие некоторой другой информации, чем для открытия файла. Для открытия файла нужно знать его название и расположение. Для открытия гнездового соединения нужно знать IP – адрес компьютера, на котором выполняется программа, с которой нужно установить связь, а также адрес порта, который прослушивается этой программой. Схема сетевых коммуникаций представлена на рис.8.2.

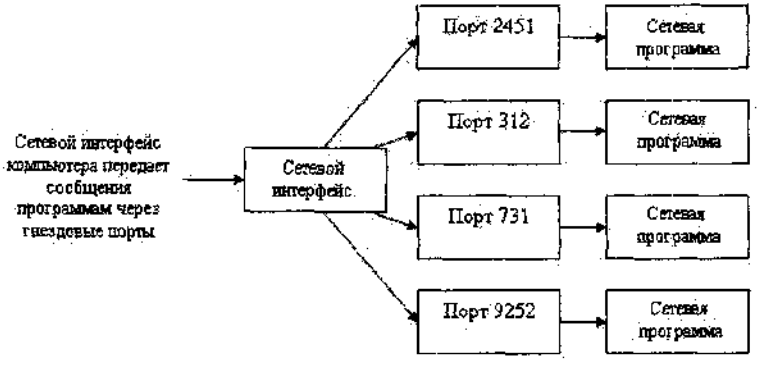


Рис.8.2. Порты используются для установления сетевых соединений

Заметим, что лишь одна программа может прослушивать сообщения, которые поступают на заданный порт некоторого компьютера. Хотя на одном и том же компьютере сразу несколько программ могут ожидать запросов установления связи, однако, каждая из них будет прослушивать отдельный порт.

При создании программ в среде Visual Studio .NET достаточно организовать в нем поддержку сетевых коммуникаций с помощью классов интерфейса Winsock. Базовый класс этого интерфейса, CAsyncSocket, обеспечивает полную поддержку гнездовой связи, которая руководствуется сообщениями. Можно создать свой собственный производный класс, который будет перехватывать соответствующие сообщения, и выполнять соответствующие действия. Класс CSocket есть производным от CAsyncSocket. Он инкапсулирует в себе и упрощает некоторые функциональные возможности базового класса. В табл.8.1 представлены методы класса CAsyncSocket и их назначение.

Таблица 8.1. **Методы класса CAsyncSocket**

Метод	Назначение
Accept	Обрабатывает запрос на соединение, который поступает на гнездо, заполняет его информацией об адресе
AsyncSelect	Организует посылку сообщения Windows при переходе гнезда в состояние готовности
Attach	Связывает дескриптор гнезда с экземпляром класса CAsyncSocket, чтобы иметь возможность сформировать соединение с другим компьютером
Bind	Ассоциирует адресс с гнездом
Close	Закрывает гнездо
Conect	Подключает гнездо к отдаленному адресу и порту
Create	Завершает процесс инициализации, начатый конструктором
Detach	Разрывает соединение с гнездом с заданным дескриптором
FromHandle	Возвращает указание на объект класса CAsyncSocket дескриптору, с которым он связан
GetLastError	Возвращает код ошибки гнезда. Вызов этого метода выполняется после обнаружения ошибки, чтобы выяснить ее причину
GetPeerName	Возвращает IP адрес и номер порта отдаленного гнезда, с которым связан объект, или заполняет этой информацией структуру гнезда

Продолжение табл. 8.1.

Метод	Назначение
GetSockName	Возвращает IP адрес та номер порта объекта this или заполняет этой информацией структуру гнезда
GetSockOpt	Возвращает текущие параметры гнезда
Listen	Принуждает гнездо следить за запросами на соединение
OnAccept	Обрабатывает сообщение Windows, которое формируется при приеме гнездом запроса на соединение. Часто переопределяется в производных классах
OnClose	Обрабатывает сообщение Windows, которое формируется при закрытии гнезда. Часто переопределяется в производных классах
OnConnect	Обрабатывает сообщение Windows, которое формируется при установлении соединения, или при неудаче его установления. Часто переопределяется в производных классах
OnOutOfBandData	Обрабатывает сообщение Windows, которое формируется при появлении срочных данных, доступных для считывания. Часто переопределяется в производных классах
OnRecieve	Обрабатывает сообщение Windows, которое формируется при появлении данных, которые можно считать с помощью функции Receive(). Часто переопределяется в производных классах
OnSend	Обрабатывает сообщение Windows, которое формируется при готовности гнезда принять данные, которые передаются с помощью Send(). Часто переопределяется в производных классах
Receive	Считывает данные из отдаленного гнезда, к которому подключено заданное гнездо
RecieveFrom	Считывает дейтаграмму (пакет в сети передачи данных, который передается через сеть без установления логического соединения) из отдаленного гнезда без установления связи
Send	Передает данные отдаленному гнезду
SendTo	Передает дейтаграму без установления связи
SetSockOpt	Устанавливает параметры гнезда
ShutDown	Оставляет гнездо открытым, но предупреждает последующие вызовы Send() или Receive()

8.2. Инициализация Winsock

Перед использованием любого из классов интерфейса Winsock необходимо инициализировать соответствующую среду для своей программы. Эта операция выполняется с помощью функции AfxSocketInit при инициализации самой программы. Функция принимает у единственного необязательного аргумента структуру WSADATA. Если передать эту структуру при вызове функции

AfxSocketInit, она будет заполнена информацией о текущей версии интерфейса Winsock. Если информация, которая передается в структуре, не нужна, можно не задавать ее в качестве параметра при вызове функции AfxSocketInit:

```
BOOL CTestNetworkApp::InitInstance()
{
...
if (!AfxSocketInit()) // инициализация Winsock
   {
   AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
// сообщение об ошибке
   return FALSE;      // выход из программы
   }
...
}
```

При использовании мастеров Visual C++ для создания оболочки проекта (рис.8.3), в которых будет указана необходимость поддержки интерфейса Winsock, функция AfxSocketInit будет автоматически прибавлена в программный код.

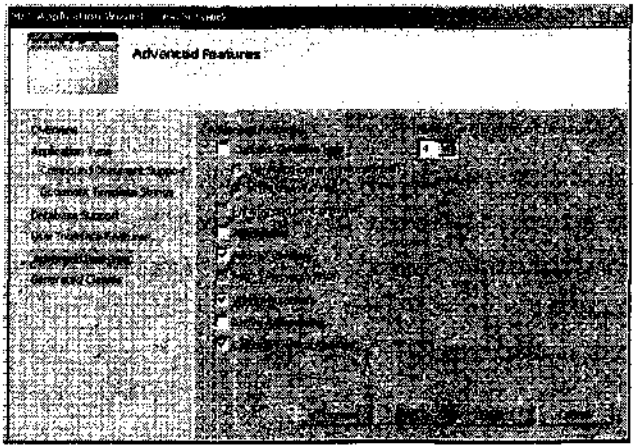


Рис.8.3. Включение поддержки Winsock при создании проекта

8.3. Создание гнезда и открытие соединения

Для создания гнезда, которое будет использоваться в программе, необходимо, прежде всего, создать переменные-члены класса CAsyncSocket.

```
class CTestNetworkDlg : public CDialog
{
public:
    ...
    CAsyncSocket m_mySock;
...
}
```

Перед тем, как использовать гнездо, нужно вызывать метод Create. Этот метод выполняет создание гнезда и подготовку его к работе. Синтаксис вызова функции Create определяется конкретными потребностями программы. Если гнездо будет использоваться для связи с другими программами, которые выступают инициаторами соединения (программами-клиентами), при вызове метода Create не нужно передавать никаких параметров.

```
if(m_mySock.Create())
  {
  // продолжение
  }
  else // обработка ошибок
```

Однако если гнездо предназначено для прослушивания канала связи в ожидании вызова со стороны другой программы (сервера), при создании гнезда необходимо указать номер порта, который будет прослушиваться.

```
if(m_mySock.Create(7000))
  {
  // продолжение
  }
  else // обработка ошибок
```

Вызов метода Create может сопровождаться другими параметрами, например, указанием типа создаваемого гнезда, сообщениями, на какое гнездо должно реагировать, а также адресом порта, который должен прослушиваться. Можно создавать два типа гнезд: поточные, или TCP-гнезда, и дейтаграммные, или UDP-гнезда. Поточные гнезда основаны на создании логического соединения и содержат встроенные функции, которые обеспечивают гарантированную доставку сообщений. Гнезда-дейтаграммы не создают логического соединения. Для создания поточного гнезда необходимо при вызове метода Create задать в качестве второго аргумента параметр SOCK_STREAM. Для создания гнезда-дейтаграммы в качестве второго аргумента следует задать параметр SOCK_DGRAM. Если создается программа, которая будет работать на компьютере с несколькими сетевыми картами, при создании гнезда нужно указать адрес порта, который будет прослушиваться. Адрес порта задается в качестве четвертого параметра для класса CAsyncSocket, или третьего для класса CSocket. Адрес передается в виде строки, например „192.168.7.62".

После создания гнезда можно открыть с ним соединение. Этот процесс состоит из трех этапов. Два из них выполняются сервером (программой, которая ожидает соединение), а один – клиентом (программой, которая выполняет вызов).

С точки зрения клиента открытие соединения заключается в вызове функции Connect. При этом, программа-клиент должна передать методу Connect два параметра: имя компьютера или его адрес, а также номер порта, с которым нужно установить связь.

```
if(m_mySock.Connect("192.168.7.62",5321)
  {
  // продолжение
  }
else // обработка ошибок
```

или

```
if(m_mySock.Connect("имя компьютера",5321)
  {
  // продолжение
  }
else // обработка ошибок
```

После того, как соединение установлено, в случае использования класса CAsyncSocket или производного от него, генерируется сообщение об установлении соединения, или о возникновении ошибки. При работе с классом CSocket функция не возвращает результат до тех пор, пока не будет установлено соединение, или не возникнет ошибка. Это может привести к зависанию программы.

С точки зрения сервера, программа сначала должна отправить гнезду команду на прослушивание вызовов, которые поступают. Это выполняется с помощью метода Listen. Этот метод принимает один параметр, который не обязательно задавать. Этот параметр определяет количество отложенных соединений, которые могут находиться в очереди сообщений в ожидании завершения предыдущих операций. Вызов метода Listen выглядит следующим образом:

```
if(m_mySock.Listen())
  {
  // продолжение
  }
else // обработка ошибок
```

Если другая программа пытается установить соединение с программой, которая прослушивает поступающие сообщения, класс CAsyncSocket (или производный от него) генерирует сообщение, которое информирует о поступлении запроса на установление соединения. Программа, которая прослушивает, должна принять запрос путем вызова метода Accept. Этот метод нуждается в использовании второго класса CAsyncSocket, который связывается с программой.

```
if(m_mySock.Accept(m_otherSock))
  {
  // продолжение
  }
else // обработка ошибок
```

При поступлении запроса на соединение, гнездо, которое прослушивает, создает другое гнездо, которое подключается к другой программе. При этом метод Create для второго гнезда не вызывается, поскольку оно создается методом Accept. Функция Accept для класса CSocket не завершает своей работы до тех пор, пока запрос на установление соединения не будет получен.

8.4. Отправление и получение сообщений

Гнезда могут использоваться для передачи любой информации, причем совсем не имеет значения какой тип имеют данные, поскольку функции, которые выполняют передачу и прием данных, ожидают передачи в них указателя на буфер. В случае передачи данных этот буфер должен содержать информацию, которая будет передана. При получении данных, они будут скопированы в этот буфер. При отправлении текстовых данных, диалог с буфером достаточно простой и выполняется с помощью переменных класса CString.

Для отправления сообщения через гнездовое соединение используется метод Send. Этот метод требует задавать два обязательных параметра, а также имеет третий необязательный параметр, который может использоваться для управления отправлением сообщения. Первый параметр представляет собой указание на буфер, которое содержит данные для передачи. Если сообщение является переменной типа CString, то для его передачи возможно использовать функцию LPCTSTR. В качестве второго параметра передается длина буфера. Метод возвращает значение, которое указывает на объем переданных

данных. При возникновении ошибки возвращается значение SOCKET_ERROR. Метод Send вызывается следующим образом:

```
CString str("Сообщение для передачи");// строка для передачи
  int len;      // переменная для определения длины сообщения
  int send;     // переменная для определения количества переданных
байт
  len=str.GetLength();      // определение длины сроки
  send=m_mySock.Send((LPCTSTR)str,len); // передача данных
  if(send==SOCKET_ERROR)  // проверка на ошибки
  {
  // обработка ошибок
  }
  else // продолжения
```

При готовности приема данных от другой программы, программа, которая выполняет прием данных, генерирует сообщение. Чтобы принять сообщение необходимо вызывать метод Receive. Первый параметр, который передается в функцию, представляет собой указатель на буфер, в который будет скопировано сообщение. Второй параметр – это размер буфера. При наличии ошибки метод Receive также возвращает значение SOCKET_ERROR. Если принято текстовое сообщение, оно может быть непосредственно скопировано в переменную типа CString. Метод Receive может быть использован следующим образом:

```
char rbuff[1024];      // буфер приема
  int buffsize=1023; // размер буферу
  int recieved;// переменная для определения количества
// принятых байт
  CString str; // срочная переменная
  recieved=m_mySock.Receive(rbuff,buffsize);      // прием данных
  if(recieved==SOCKET_ERROR)      // проверка на ошибки
  {
  // обработка ошибок
  }
  else
  {
  rbuff[ercieved]='\0';       // ограничение сроки
```

```
    str=rbuff;    // копирование указания на буфер
}
```

При получении текстовых сообщений следует записывать ограничитель '0' после последнего принятого символа. Если этого не сделать, то символы, которые остались в буфере, ошибочно будут приняты как часть полученного сообщения.

Для класса CSocket функция Receive не завершит свою работу до тех пор, пока не будут приняты данные. При использовании дейтаграмних гнезд можно использовать альтернативные версии двух последних методов: SendTo и RecieveFrom.

Когда программа завершит обмен данными, она может закрыть соединение, вызывав метод Close. Этот метод не использует входных параметров и синтаксис его вызова выглядит следующим образом:

```
m_mySock.Close();
```

Перед закрытием гнезда его можно отключить. Это выполняется с помощью метода ShutDown. Единственный параметр, который передается в эту функцию, указывает на необходимость отключения приема или передачи данных для данного гнезда: 0 – завершает прием пакетов; 1 – завершает передачу пакетов; 2 – завершает и прием, и передачу данных через гнездо.

8.5. Управление процессом генерации сообщений

Основная причина, по которой следует создавать производные классы от CAsyncSocket и CSocket заключается в том, что они позволяют перехватывать сообщения, которые генерируются при получении данных, установлении или расторжении связи и пр. Класс CAsyncSocket содержит набор функций, которые вызываются в ответ на каждое из сообщений. В производных классах эти функции должны

быть переписанными. В табл.8.2 приводятся эти функции та их назначение.

Функция	Описание
OnAccept	Вызывается для гнезда, которое прослушивает, чтобы сообщить, что ожидается запрос на соединение, который поступил от другой программы.
OnClose	Вызывается, чтобы сообщить гнезду, что программа на втором конце соединения закрыла гнездо из своей стороны, или о том, что соединение разорвано. Это сообщение должно сопровождаться закрытием гнезда.
OnConnect	Вызывается для гнезда, чтобы просигнализировать ему, что соединение завершено и программа может принимать и передавать данные.
OnOutOfBandData	Вызывается в случае приема служебных данных.
OnRecieve	Сигнализирует о поступлении данных по сетевому соединению, какие доступные для считывания методом Receive.
OnSend	Сигнализирует о готовности гнезда для передачи данных. Вызывается сразу же после завершения соединения.

Класс CAsyncSocket вызывает все функции, приведенные в табл.8.2, а класс CSocket – ни одну из них. Для того, чтобы вызывалась часть из приведенных функций, необходимо при вызове метода Create передать в качестве третьего параметра набор флажков, которые определяют, какие из сообщений будут обработаны. Другой способ заключается в вызове функции AsyncSelect, которая выполняет фильтрацию сообщений:

```
if(m_mySock.AsyncSelect(FD_READ | FD_CONNECT |
FD_CLOSE)==SOCKET_ERROR)
{
  // обработка ошибок
}
else
{
// все хорошо
}
```

Значения, которые могут передаваться в функцию AsyncSelect, представлены в табл.8.3.

Таблица 8.3. **Флажки, которые определяют обрабатываемые сообщения**

FD_READ	При поступлении данных вызывается функция OnRecieve, которая информирует о наличии данных для считывания
FD_WRITE	Вызывается функция OnSend, которая информирует о готовности к передаче данных
FD_OOB	Вызывается функция OnOutOfBandData при поступлении служебных данных
FD_ACCEPT	Вызывается функция OnAccept, которая информирует программу о том, что на гнездо поступил запрос на входное соединение
FD_CONNECT	Вызывается функция OnConnect, которая информирует программу о завершении запроса на соединение. Это сообщение сопровождается вызовом функции OnSend.
FD_CLOSE	Вызывается функция OnClose, которая информирует программу о том, что гнездо было закрыто.

Иногда программе необходима информация о состоянии гнезд, например адреса порта на другом конце соединения, а также о том, находится гнездо в заблокированном состоянии в ожидании окончания некоторой операции, или нет. При наличии гнезда, подключенного к другой программе, можно определить адрес компьютера, на котором эта программа выполняется. Это делается с помощью метода GetPeerName с передачей ему указания объекта CString и переменной целого типа. После выполнения функции GetPeerName эти параметры заполняются адресом и номером порта компьютера, с которым установлена связь.

```
iCString addr; // объявление переменных
  UINT port;
  if(m_mySock.GetPeerName(&addr,&port)!=SOCKET_ERROR)
  {
  MessageBox(addr);        // вывод сообщения с адресом
// отдаленной машины
  }
  else
  {
        // обработка ошибок
}
```

Аналогичным образом можно получить данные о гнезде, которое прослушивает сообщение. Для этого необходимо вызывать функцию

GetSockName с такими же параметрами, как и для функции GetPeerName.

8.6. Пример разработки программы

Для того, чтобы продемонстрировать основные возможности интерфейса Winsock, рассмотрим пример.

Пример 8.1. Создать программу, которая будет функционировать как в роли клиента, так и в роли сервера. После того, как программа установит соединение с другой программой по сети, пользователь может вводить текстовые сообщения и передавать их по сети на другой компьютер. Каждое принятое сообщение записывается в список принятых сообщений.

Порядок действий:

1. Создаем проект на базе диалогового окна. При создании проекта включить поддержку Windows Sockets (рис.8.3). На диалоговом окне должен быть расположен переключатель, который определяет режим работы программы, – клиент или сервер. Для этого удобно использовать элемент управления CheckBox. Также должны содержаться элементы управления для ввода адреса компьютера, с которым происходит соединение, и номера порта. Для записи принятых сообщений на форме следует расположить элемент управления ListBox, а для ввода сообщений – элемент управления EditBox. Сообщение будет отправлено при нажатии на кнопку „Отправить". На рис.8.4 представлен вид спроектированного интерфейса программы.

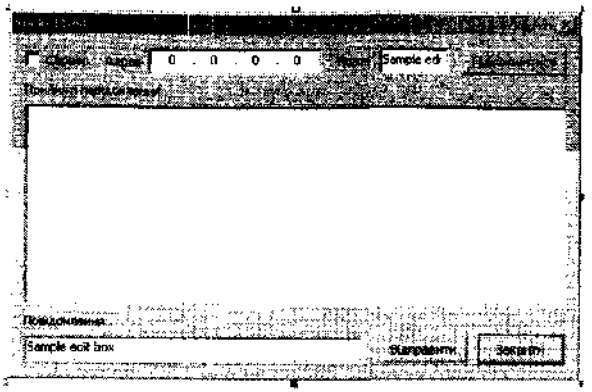


Рис.8.4. Создано диалоговое окно программы

2. Связываем с элементами управления переменные. Для этого из контекстного меню, которое появляется при нажатии правой кнопки мыши на элементе управления, выбрать команду Add Variable (рис.8.5).

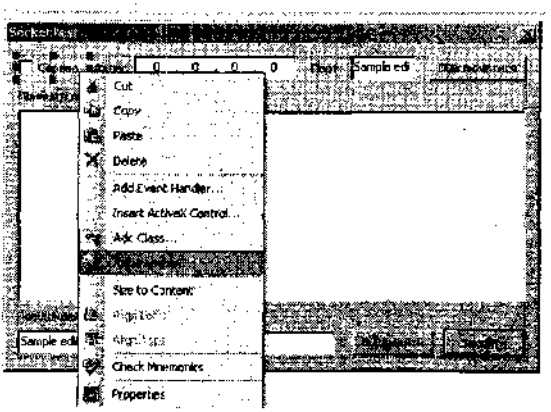


Рис.8.5. Добавление переменной, связанной с элементом управления

Для элемента управления CheckBox название переменной m_serv (рис.8.6). С элементом управления IP Address связана переменная

m_ipaddr, из EditBox – m_port, из LisBox – m_messages, со вторым EditBox – m_smsg. При добавлении каждой переменной флажок Control Variable должен быть установлен (рис.8.6).

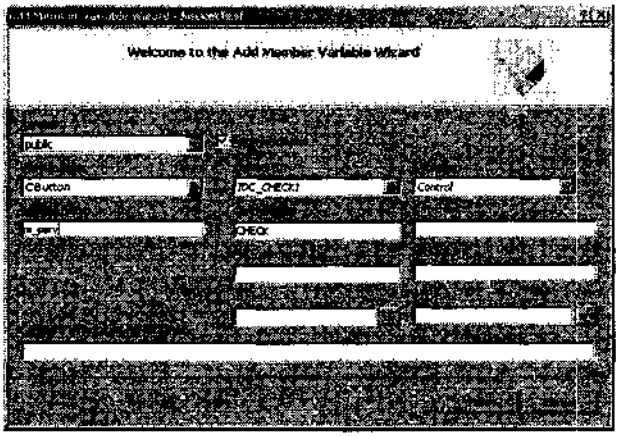


Рис.8.6. Окно добавления переменной

3. Для того, чтобы иметь возможность перехватывать сообщения, связанные с гнездом, и реагировать на них, нужно создать собственный класс, производный от CAsyncSocket. Этот класс будет содержать собственные версии процедур обработки событий, а также собственные средства передачи этих сообщений диалоговому окну. Для передачи всех сообщений классу на уровне диалогового окна, нужно прибавить указание класса диалогового окна в виде переменной – члена класса гнезда. Это указание будет использоваться для вызова функций обработки событий, связанных с гнездом. Для создания класса из меню Project выбрать команду Add Class. В диалоговом окне, что появится, следует выбрать MFC Class. Потом ввести название нового класса (CMySocket), и выбрать базовый класс (CAsyncSocket). На рис.8.7 представлено диалоговое окно создания нового класса.

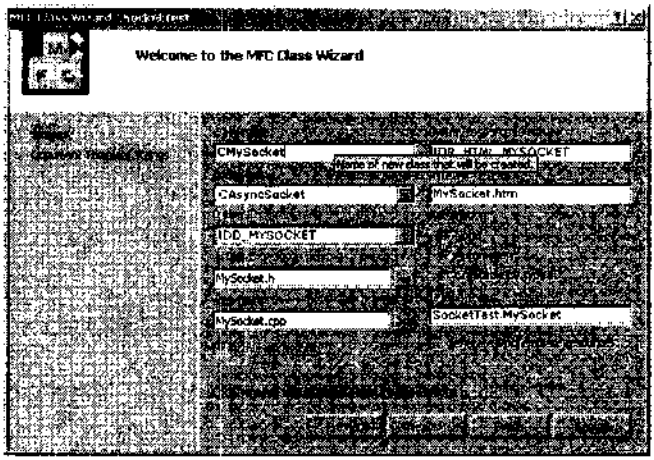


Рис.8.7. Создание производного класса

После создания класса добавим в него переменную -член, которая будет указанием на диалоговое окно. Тип переменной CDialog*, а название – m_pWnd. Кроме того, необходимо добавить метод, предназначенный для установления значения указания. Потом в класс CMySocket добавляем еще один метод:

void SetParent(CDialog* pdlg);

В теле функции должна выполняться операция присвоения указания на диалоговое окно. В целом, реализация этой функции выглядит следующим образом:

```
void CMySocket::SetParent(CDialog* pdlg)
{
  m_pWnd=pdlg;    // копирование указания
}
```

4. Вводим в класс гнезда функции обработки событий, которые будут вызывать одноименные функции класса диалогового окна. Программный код функции OnAccept имеет вид:

```
void CMySocket::OnAccept(int err)
{
```

```
    if(err==0) ((CSocketTestDlg*)m_pWnd)->OnAccept();
    // вызывается одноименная функция класса диалогового окна
}
```

Аналогично создаем функцию обработки события OnRecieve, которое предназначено для вызова одноименных функций класса диалогового окна.

```
void CMySocket::OnReceive(int err)
{
  ((CSocketTestDlg *)m_pWnd)->OnReceive(this);
  // вызывается одноименная функция класса диалогового окна
}
```

Добавив эту функцию, также необходимо подключить заголовочный файл. Для этого в начале файла MySocket.cpp добавляем строку: #include "SocketTestDlg.h".

5. Для режима сервера в классе диалогового окна должны быть две переменные: первая из них будет прослушивать запросы на соединение, а вторая – обеспечивать связь с другой программой. Тип обоих переменных должен отвечать типу класса гнезда, то есть CMySocket. Добавляем две переменные в класс CSocketTestDlg. Для этого в файл SocketTestDlg.h прибавить следующие строки:

```
CMySocket m_socket;       // гнездо для связи
CMySocket m_listensock;  // гнездо для прослушивания запросов
```

Также необходимо выполнить инициализацию этих переменных. Для этого в конце функции CSocketTestDlg::OnInitDialog() добавляем следующие строчки:

```
m_socket.SetParent(this);       // передача указания на
    m_listensock.SetParent(this);   // объект диалоговое окна
```

6. Когда пользователь устанавливает флажок выбора режима сервера, нужно выполнить создание гнезда. Добавляем обработчик события нажатия на элемент управления CheckBox, дважды нажав на

нем левую кнопку мыши. Отредактируем функцию обработчик события следующим образом:

```
void CSocketTestDlg::OnBnClickedCheck1()
{
  if(m_serv.GetCheck()!=0)       // если программа работает в режиме
клиента
  {
  int iport;     // переменная для сохранения номера порта
  CString sport;       // срочная переменная для считывания
  // номера порта из элемента управления
  m_port.GetWindowTextW(sport);     // считывание номера порта
  iport=StrToInt(sport);     // превращение сроки в число
  m_listensock.Create(iport);       // создание гнезда для
  // прослушивания
  m_listensock.AsyncSelect(FD_ACCEPT|FD_READ);
  // задается маска событий, которые будут обработаны
  m_listensock.Listen();     // начало прослушивания порта
  CString s("Server is running"); // сообщение о
  // запуске сервера
  MessageBox(s);     // выдача сообщения
  }
  else    // если программа работает в режиме клиента
  {
  CString s("Server is stopped");
  MessageBox(s); // выдача сообщения об остановке сервера
  m_listensock.Close();     // закрытие гнезда прослушивания
  }
}
```

Функция Create вызывается для соответствующей переменной-члена класса. Для запуска сервера вызывается функция Listen. Как параметр в эту функцию передается номер порта, из которого ожидаются сообщения. Номер порта вводится пользователем в соответствующем поле на диалоговом окне. Введенное значение превращается из сроки в число с помощью функции StrToInt. Для того, чтобы при компиляции не возникло ошибки, следует подключить заготовочный файл: #include "shlwapi.h"

7. Добавляем обработчик события нажатия на кнопку „Подключиться". Функция-обработчик события нажатия выглядит следующим образом:

```
void CSocketTestDlg::OnBnClickedButton1()
{
  if(m_serv.GetCheck()!=0)       // если режим сервера
  {
  MessageBox((LPCTSTR)"Программа работает в режиме сервера.");
  // выдача сообщения
  return;       // выход из функции
  }
  CString sip, sport; // дополнительные срочные переменные
  int iport;
  if(m_socket.Create()==SOCKET_ERROR) // создание гнезда
  {
  CString s("Error create socket!");
  MessageBox(s);   // выдача сообщения об ошибке
  return;       // выход из функции
  }
  m_ipaddr.GetWindowTextW(sip);    // получение адреса
  m_port.GetWindowTextW(sport);    // получение номера порта
  iport=StrToInt(sport);    // превращение сроки в число
  // попытка подключиться к серверу
  if(m_socket.Connect(sip,iport)==SOCKET_ERROR)
  {
  // если подключение не удалось, выдача сообщения
  // о невозможности установить соединение
  CString s("Connection is impossible!");
  MessageBox(s);
  return;
  }
  else   // если подключение удалось
  {
  // выдается соответствующее сообщение
  CString s("Connection was succesfull!");
  MessageBox(s);
  }
}
```

Для установления соединения вызывается функция Connect для соответствующей переменной. Параметры, а именно адрес отдаленно-

го компьютера и номер порта, вводятся пользователем с помощью элементов управления на диалоговом окне.

8. Добавляем в класс диалогового окна CSocketTestDlg функции OnAccept и OnRecieve для обработки соответствующих событий. Эти функции вызываются одноименными процедурами класса гнезда. Тип обеих функций -- void. Следовательно, функция OnAccept выглядит так:

```
void CSocketTestDlg::OnAccept()
{
  m_listensock.Accept(m_socket);      // при поступлении
// запросу на соединение происходит установление канала связи
  CString str("Connection was accepted!");
  MessageBox(str);
}
```

При установлении соединения выдается соответствующее сообщение. Функция OnReceive выглядит следующим образом:

```
void CSocketTestDlg::OnReceive(CMySocket* sock)
{
  char buff[1024];   // буфер для принятых данных
  int rsize;
  rsize=m_socket.Receive(buff,1023);  // считывание принятых
// данных
  if(rsize==SOCKET_ERROR)
  {
  MessageBox((LPCTSTR)"Receive error!");
  return;
  }
buff[rsize]='\0';
CString str(buff);     // формирование сроки
m_messages.AddString(str);       // сообщение добавляется в список
}
```

Каждое сообщение, которое будет принято, добавляется в список.

9. Добавляем обработчик события нажатия на кнопку „Отправить". При нажатии на эту кнопку текстовое сообщение в поле

ввода передается на отдаленный компьютер. Функция–обработчик события нажатия на кнопку выглядит следующим образом:

```
void CSocketTestDlg::OnBnClickedButton2()
{
CString s;      // дополнительная срочная переменная
int len;
// считывание сообщения из элемента управления
m_smsg.GetWindowTextW(s);
len=s.GetLength(); // определение длины сообщения
m_socket.Send((LPCTSTR)s,5);  // передача данных по сети
}
```

Теперь можно компилировать проект. Для проверки работы программы необходимо запустить две его копии (если тестирование происходит на одном и том же компьютере). Одна из программ работает в режиме сервера. Перед тем, как установить отметку „Сервер", в поле „Порт" необходимо ввести номер порта, который будет прослушиваться (рис.8.8). В случае удачного выполнения процедуры запуска сервера появится соответствующее сообщение.

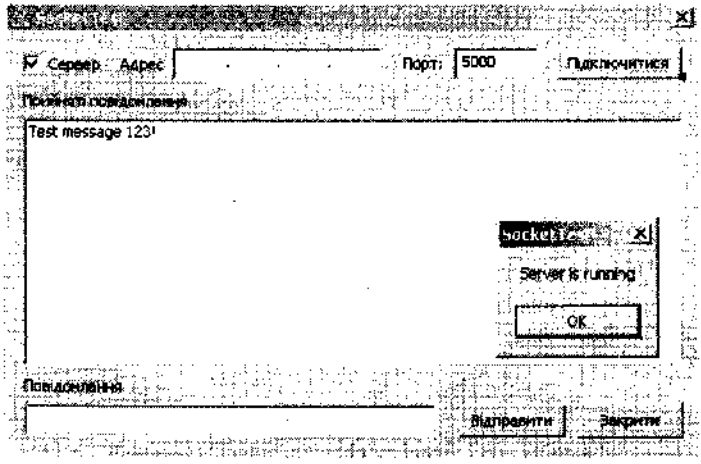


Рис.8.8. Программа работает в режиме сервера

Для установления соединения необходимо указать IP-адрес компьютера, с которым устанавливается связь, а также номер порта. При удачном подключении появится соответствующее сообщение. На рис.8.9 представлено диалоговое окно программы при работе в режиме клиента.

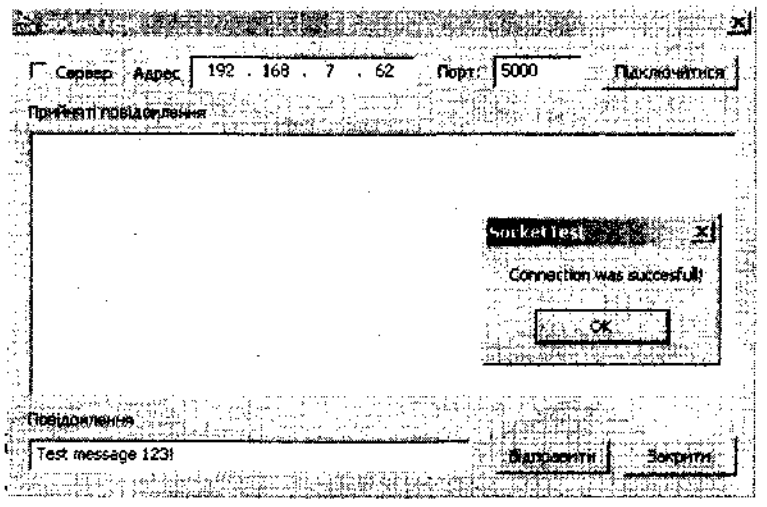


Рис.8.9. Программа работает в режиме клиента

Приведенный пример демонстрирует основные принципы организации связи между компьютерами по локальной сети с использованием классов библиотеки MFC. Программа позволяет передавать сообщения в обоих направлениях: от клиента на сервер и наоборот. При реализации систем управления организуются более сложные протоколы, в зависимости от конкретных требований, однако принцип работы с локальной сетью остается неизменным. Существует много фирменных программ, которые выполняют управление технологическим объектом или процессом по локальной сети. Одной из таких программ

есть PowerSuite, интерфейс которой представлен на рис.8.10. Эта программа предназначена для управления электроприводом Altivar, который соединяется с компьютером через последовательный порт. Обмен данными происходит по протоколу MODBUS. Для организации управления электроприводом по сети необходимо знать формат команд, который можно найти в соответствующей документации.

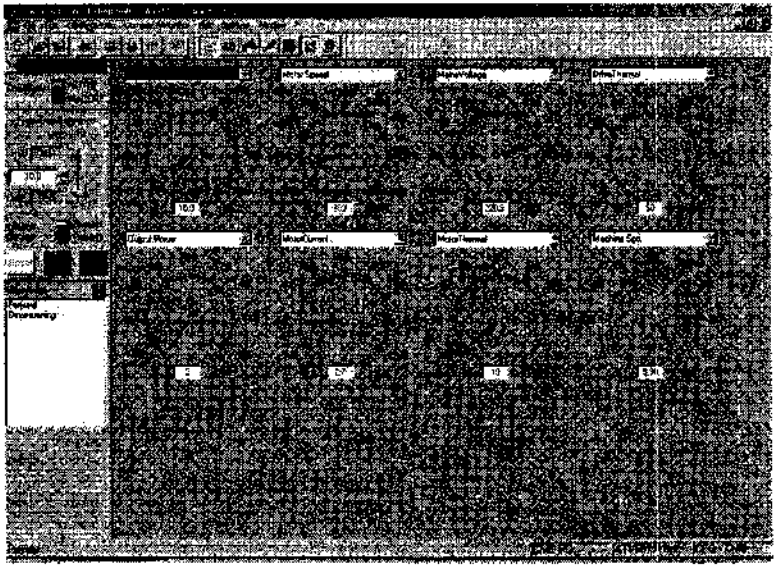


Рис.8.10. Программа PowerSuite

На рис.8.11 представлено окно программы, которая выполняет управление электроприводом по локальной сети. Программа разработана на кафедре ТЕЕС НУК.

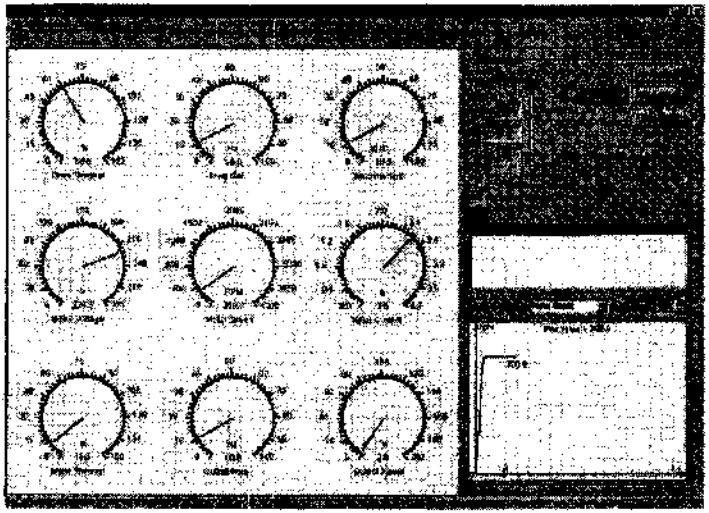


Рис.8.11. Программа управления электроприводом по локальной сети

При написании этой программы была использована технология Winsock. Программа может выступать как в роли клиента, так и в роли сервера. При работе в режиме клиента происходит формирование пакетов и их передача по локальной сети, а также прием данных и их отображение. При работе программы в режиме сервера пакеты, которые поступают от клиента, перенаправляются в COM-порт и передаются в Altivar.

Большинство современных компьютеров обеспечивают добрые показатели надежности, но они тоже выходят из строя, особенно при использовании в жестких производственных условиях. Если какие-либо компоненты производственного процесса (или весь процесс) являются критически важными или стоимость остановки производства очень высока, возникает необходимость построения систем резервирования. В системах с резервированием выход из строя одного компонента не тянет за собой остановку всей системы.

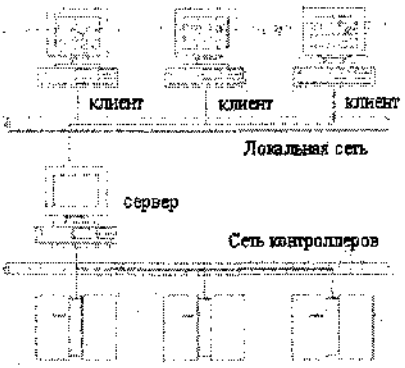


Рис.8.12. Клиент-серверная архитектура простой системы

Повысить эффективность и скорость работы всей системы позволяет распределение процессов управления и контроля по нескольким компьютерам, которые соединены в локальную сеть. В простой системе компьютер, соединенный с промышленным оборудованием, становится сервером, предназначенным для взаимодействия с контролерами, в то время как компьютеры локальной сети клиентами (рис.8.12). Когда компьютеру - клиенту нужны данные для отображения, он запрашивает их у сервера и потом обрабатывает локально.

Для обеспечения резервирования в систему может быть прибавлен второй (резервный) сервер, также предназначенный для взаимодействия с промышленным оборудованием (рис.8.13).

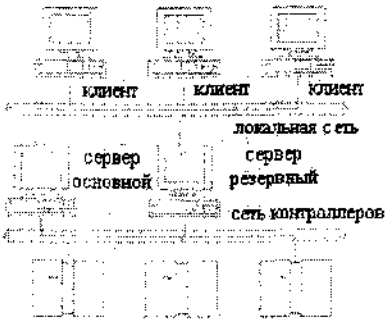


Рис.8.13. Система с дублируемым сервером

Если основной сервер выходит из строя, запить клиентов направляются к резервному. Резервный сервер не должен при этом полностью дублировать работу основного, поскольку в этом случае обе сервера взаимодействуют с контролерами, увеличивая нагрузку на промышленную сеть, уменьшая, таким образом, общую производительность. В целом, организация обмена данных между компьютерами также может быть организована с использованием сокетов.

ПРИЛОЖЕНИЯ

Приложение А.
Последовательный периферийный интерфейс - SPI

Последовательный периферийный интерфейс обеспечивает высокоскоростной синхронный обмен данными между контроллером и периферийными устройствами или между несколькими микроконтроллерами. Основные характеристики интерфейса SPI:

- Полнодуплексный 3-проводный синхронный обмен данными
- Режим работы ведущий или ведомый
- Обмен данными с передаваемыми первыми старшим или младшим битами
- Четыре программируемые скорости обмена данными
- Флаг прерывания по окончанию передачи
- Выход из режима пониженного потребления энергии (только в режиме ведомого)

Соединение между ведущим и ведомым устройствами показано на рис.А1.

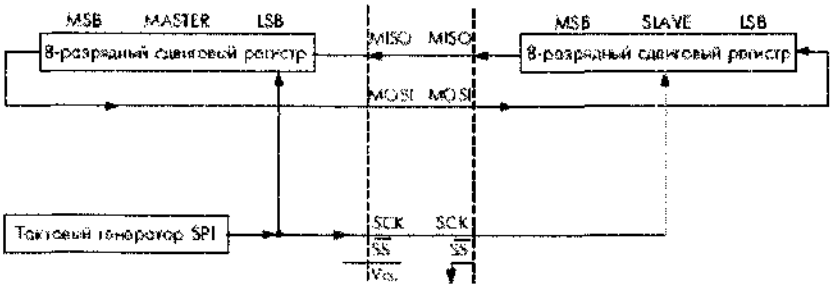


Рис.А1. Соединение ведущего и ведомого устройств

Два сдвиговых регистра ведущего и ведомого можно рассматривать как один разнесенный 16-разрядный циклический сдвиговый регистр. При сдвиге данных из ведущего микроконтроллера в ведомый одновременно происходит сдвиг данных из ведомого микроконтроллера в ведущий, т.е. в течение одного цикла сдвига происходит обмен данными между ведущим и ведомым микроконтроллерами.

Вывод SCK является выходом тактового сигнала ведущего микроконтроллера и входом тактового сигнала ведомого. При записи ведущим микроконтроллером данных в регистр SPI начинает работать тактовый генератор и записанные данные сдвигаются через вывод выхода MOSI ведущего микроконтроллера на вывод входа MOSI ведомого. После передачи одного байта генератор останавливается, устанавливая флаг завершения передачи SPIF. Если в регистре SPCR установлен флаг разрешения прерывания (SPIE), то произойдет запрос прерывания. Вход выбора ведомого SS, для выбора индивидуального SPI устройства в качестве ведомого, устанавливается на низкий уровень. При установке высокого уровня на выводе SS порт SPI деактивируется. Режим ведущий/ведомый может быть установлен программным способом установкой или очисткой бита MSTR в регистре управления SPI. На рис.А2 представлена блок-схема SPI.

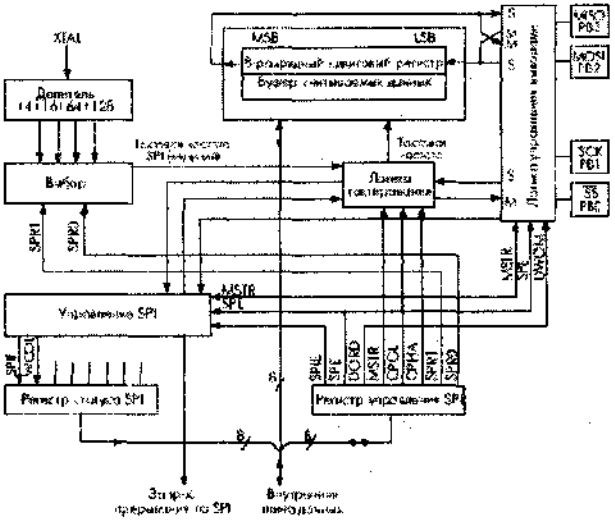


Рис.А2. Блок-схема SPI

При работе SPI ведущим (бит MSTR регистра SPCR установлен), пользователь имеет возможность установить направление работы вывода #SS. Если вывод #SS сконфигурирован как выход, то вывод является выводом общего назначения и он не активируется системой SPI. Если же вывод #SS сконфигурирован как вход, то для обеспечения работы ведущего SPI он должен удерживаться на высоком уровне. Если в режиме ведущего вывод #SS является входом и внешней периферийной системой на него подан низкий уровень, то SPI воспримет его как обращение другого ведущего SPI к себе как к ведомому. Чтобы избежать конфликтной ситуации на шине, система SPI выполняет следующие действия:

1. Бит MSTR в регистре SPCR очищается и SPI система становится ведомой. Результатом этого является то, что MOSI и SCK выводы становятся входами.

2. Устанавливается флаг SPIF регистра SPSR и, если разрешено прерывание SPI, начинается выполнение подпрограммы обработки прерывания.

Если SPI работает в режиме ведомого, то вывод #SS постоянно работает на вход. Если на вывод #SS подан низкий уровень, то SPI активируется и вывод MISO становится выходом. Все остальные выводы являются входами. Если вывод #SS удерживается на высоком уровне, то все выводы являются входами, SPI пассивен, что означает, что он не будет получать входящих данных.

Существует четыре варианта комбинации фазы и полярности SCK относительно последовательных данных, определяемые управляющими битами CPHA и CPOL. Форматы передачи данных показаны на рис.А3 и рис.А4.

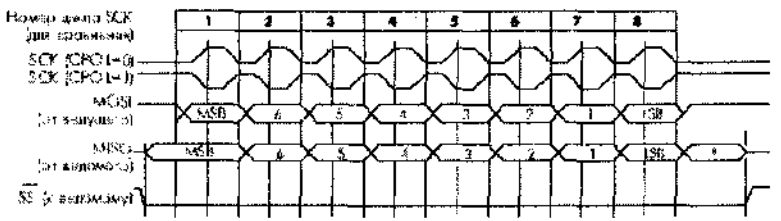


Рис.А3. Формат посылок SPI при CPHA=0

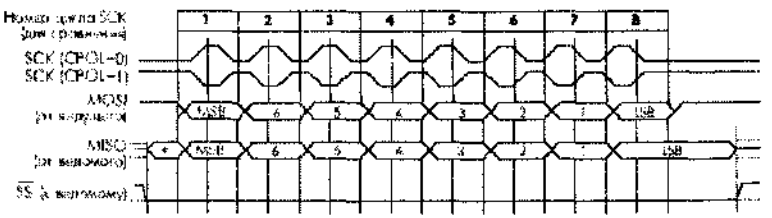


Рис.А4. Формат посылок SPI при CPHA=1

Конфигурация SPI

Описание регистров SPI для контроллеров AVR приводится ниже.

Регистр SPCR - регистр управления

SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

SPIE – разрешение прерывания SPI.
SPE – SPI включен. Если установлен этот бит, то выводы #SS, MISO, MOSI и SCK работают как выводы SPI, иначе – как простые выводы порта.
DORD – направление передачи данных. Если установлен, то передача идёт с младшего бита, если сброшен – со старшего.
MSTR – если установлен, контроллер работает как Master, если сброшен – как Slave. Управляется также выводом #SS, если он настроен на ввод – при подаче "0" на #SS бит MSTR сбрасывается.
CPOL – определяет уровень на выходе SCK в режиме ожидания – SCK = CPOL.
CPHA – если установлен, передача и приём бита производится по спаду сигнала, если сброшен – по фронту.
SPR1, SPR0 – делитель тактовой частоты (F – частота генератора контроллера).

SPR1	SPR0	Частота
0	0	F/4
0	1	F/16
1	0	F/64
1	1	F/128

Регистр SPSR - регистр состояния

SPIF	WCOL	–	–	–	–	–	–

SPIF – флаг прерывания по окончанию обмена данными. Устанавливается и сбрасывается аппаратно. Если бит SPIE в регистре SPCR установлен, и разрешено глобальное прерывание, генерируется сигнал прерывания. Бит SPIF может быть очищен также при первом считывании регистра состояния.

WCOL – флаг ошибки при записи. Устанавливается, когда принятый байт наложился на ещё не считанный из регистра данных. Сбрасывается чтением регистра состояния.
Биты 5..0 – зарезервированные биты. При считывании всегда покажут 0.

Регистр SPDR – регистр данных

MSB	–	–	–	–	–	–	LSB

Регистр данных SPI представляет собой регистр с возможностью чтения/записи и предназначен для пересылки данных. Запись в регистр SPDR инициирует передачу данных, считывание регистра приводит к чтению сдвигового регистра приема. Передача и приём тактируются по линии SCK в соответствии с конфигурацией, заданной в регистре управления. После того, как байт передан, тактирование прекращается, в регистре состояния устанавливается флаг SPIF и вызывается прерывание окончания передачи, если оно разрешено. Запись в регистр данных передающего контроллера невозможна, пока не освобождён регистр сдвига SPI.

Интерфейс SPI часто используется для обмена данными между микроконтроллерами. На рис.А5 представлена схема сопряжения 2-х микроконтроллеров AT90S8535.

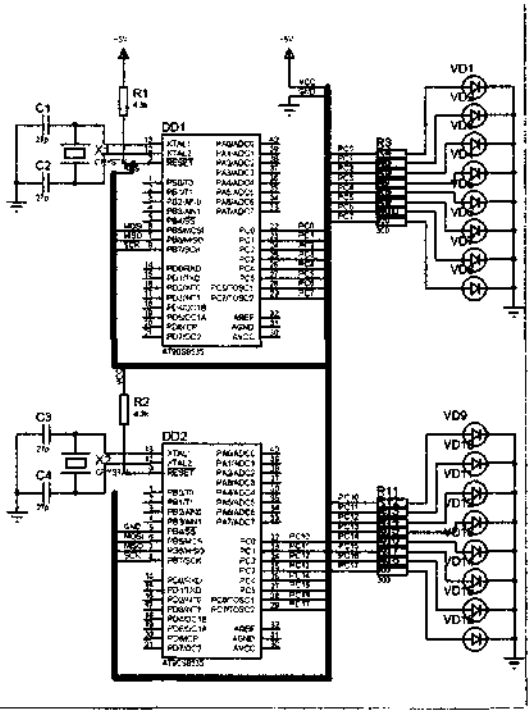


Рис.А5. Схема сопряжения 2-х микроконтроллеров по интерфейсу SPI

Пример А1. Написать программы для ведущего и ведомого микроконтроллеров AT90S8535, которые реализуют следующий алгоритм: ведущий микроконтроллер (DD1) выполняет инкрементирование переменной-счетчика и каждые 200мс передает значение этой переменной по интерфейсу SPI. Принятое значение выводится в порт С. Ведомый микроконтроллер выводит в порт С принятое по SPI значение. Принятое число инвертируется и передается обратно ведущему микроконтроллеру.

1. Запускаем среду разработки ImageCraft и создаем новый проект с названием Master (см. раздел 2). В настройках проекта выбираем

контроллер AT90S8535. Текст программы для микроконтроллера DD1, который работает в режиме ведущего, представлен ниже.

```
#include <io8535v.h>        // подключение заголовочных файлов
#include <macros.h>
// функция инициализации микроконтроллера
void init_cpu()
{
 DDRA=0x00;          // порт A настроен на вход
 DDRB=0b10110000;        // линии SS, MOSI, SCK настроены на выход
 DDRC=0xFF;          // порт C настроен на выход
 DDRD=0x00;          // порт D настроен на вход
 SPCR=0x53;          // настройка SPI: SPI включен в режиме
                // ведущего, делитель тактовой частоты 128
}
void delay_ms(int ms)       // функция задержки
{
 int c1, c2;                 // дополнительные переменные
 for(c1=0;c1<ms;c1++) // контроллер выполняет «пустую» операцию
 for(c2=0;c2<970;c2++) NOP();
}
void main()             // главная функция
{
 char counter=0;         // объявление переменной-счетчика
 char rdata;             // дополнительная переменная
 init_cpu();             // вызов функции инициализации контроллера
 while(1)                    // вечный цикл
 {
 SPDR=counter;       // передача значения переменной counter
 while(!(SPSR&0x80));    // ожидание завершения передачи
 rdata=SPDR; // чтение регистра данных SPI
 PORTC=rdata;                // вывод считанного значения в порт C
 counter++;          // инкрементирование значения переменной
 delay_ms(200);      // вызов функции задержки
 }
}
```

После компиляции проекта будет создан файл прошивки микроконтроллера.

2. Создаем новый проект с названием Slave. Текст программы для микроконтроллера DD2, который работает в режиме ведомого, имеет вид:

```
#include <io8535v.h>        // подключение заголовочных файлов
#include <macros.h>
// подпрограмма обработки прерываний
#pragma interrupt_handler spi_stc_isr:11
void spi_stc_isr(void)
{
 char data;             // объявление дополнительной переменной
 data=SPDR;             // считывание принятого значения из регистра
                        // данных SPI
 PORTC=data;            // вывод данных в порт С
 SPDR=~data;            // запись в регистр данных инвертированного
                        // значения
}
// функция инициализации микроконтроллера
void init_cpu()
{
DDRA=0;                     // порт А настроен на вход
DDRB=0b01000000;            // вывод MISO настроен на выход
DDRC=0xFF;             // порт С настроен на выход
DDRD=0;                     // порт D настроен на вход
SPCR=0xC3;             // настройка SPI: SPI включен в режиме
                  // ведомого, делитель тактовой частоты 128
                        // разрешено прерывание по приему
SEI();                  // разрешение обработки прерываний
}
void main()             // глафная функция
{
 char c;                // объявление дополнительной переменной
 init_cpu();            // вызов функции инициализации контроллера
 while(1);              // вечный цикл
}
```

Приложение В.
Шина I2C

Шина I2C широко используется в бытовой электронике, передаче данных и промышленной электронике. Разработанная фирмой Philips простая двунаправленная 2-проводная шина для эффективного управления и взаимодействия различных блоков телевизоров, она стала применяться для связи между собой однокристальных микроконтроллеров, ЖКИ-индикаторов, микросхем памяти, аналого-цифровых и цифро-аналоговых преобразователей, часов реального времени и др.

I2C шина является одной из модификаций последовательных протоколов обмена данных. В стандартном режиме обеспечивается передача последовательных 8-битных данных со скоростью до 100 кбит/с, и до 400 кбит/с в "быстром" режиме. Для осуществления процесса обмена информацией по I2C шине, используется всего два сигнала линия данных SDA линия синхронизации SCL. Для обеспечения реализации двунаправленности шины без применения сложных арбитров шины выходные каскады устройств, подключенных к шине, имеют открытый сток или открытый коллектор для обеспечения функции монтажного "И".

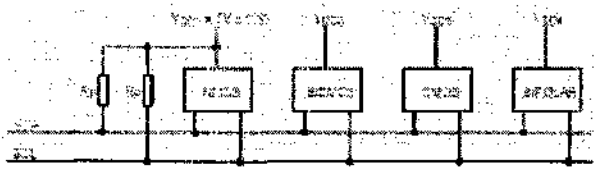


Рис.В1. Принцип подключения нескольких устройств на шину I2C

Простая двухпроводная последовательная шина I2C минимизирует количество соединений между интегральными схемами. Как результат - печатные платы становятся более простыми и технологич-

ными при изготовлении. Интегрированный I2C-протокол устраняет необходимость в дешифраторах адреса и другой внешней логике согласования. Максимальное допустимое количество микросхем, подсоединённых к одной шине, ограничивается максимальной емкостью шины 400 пФ.

Шина I2C имеет следующие преимущества:

- Только две линии – последовательная линия данных (SDA) и последовательная линия синхронизации (SCL).

- Каждый элемент, соединенный с шиной, является программно-адресуемым своим уникальным адресом. При этом отношения между ними могут быть построены по простому принципу master/slave.

- Последовательная 8-разрядная передача данных может проводиться со скоростью от 0 до 100 кБит/с в стандартном режиме и до 400 кБит/с в быстром режиме. При реализации собственных устройств возможно применение и более высоких скоростей.

- Низкое энергопотребление, высокая помехоустойчивость, широкий диапазон питающих напряжений.

Все абоненты шины делятся на два класса – «Master» и «Slave». Устройство «Master» генерирует тактовый сигнал (SCL), и как следствие, является ведущим. Оно может самостоятельно выходить на шину и адресовать любое «Slave» - устройство с целью передачи или приема информации. Все «Slave» - устройства «слушают» шину на предмет обнаружения собственного адреса, и, распознав его, выполняют предписываемую операцию. Кроме того, возможен режим «Multi-master» - режим, когда на шине установлено несколько ведущих устройств, которые либо совместно разделяют общие «Slave» - устройства, либо попеременно являются то ведущими устройствами, когда сами инициируют обмен информацией, или ведомыми устройствами, когда

ожидают обращения от другого ведущего устройства. Режим «Multi-master» требует арбитража и распознавания конфликтов. Он сложнее реализуется и поэтому реже используется в устройствах.

В начальный момент времени – в режиме ожидания – обе линии SDA и SCL находятся в состоянии логической единицы. В режиме передачи (рис.В2) бит данных SDA стробируется положительным импульсом SCL. Смена информации на линии SDA производится при нулевом состоянии линии SCL. «Slave» - устройство может удерживать линию SCL в нулевом состоянии на время обработки очередного принятого байта, при этом «Master» - устройство должно дождаться освобождения линии SCL, прежде чем продолжить передачу информации.

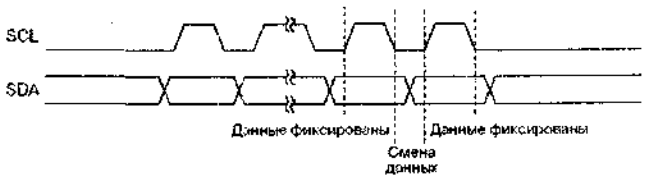


Рис.В2. Диаграмма передачи данных по I2C

Для синхронизации пакетов шины I2C различают два условия – «Start» и «Stop», ограничивающие начало и конец информационного пакета (рис.В3). Для кодирования этих условий используется изменение состояния линии SDA при единичном состоянии линии SCL, что недопустимо при передаче данных. «Start» - условие образуется при отрицательном перепаде линии SDA, когда линия SCL находится в единичном состоянии. «Stop» - условие образуется при положительном перепаде линии SDA при единичном состоянии линии SCL.

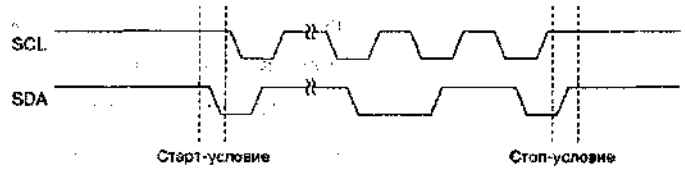


Рис.В3. Формирование «Старт»/«Стоп» условий

Состояния «Старт» и «Стоп» всегда вырабатываются ведущим. Считается, что шина занята после фиксации состояния «Старт». Шина считается освободившейся через некоторое время после фиксации состояния «Стоп».

Передача данных начинается по первому положительному импульсу на линии SCL, которым стробируется старший бит первого информационного байта. Каждый информационный байт содержит 9 тактовых периодов линии SCL. В девятом такте устройство-получатель выдает подтверждение (#ACK) – отрицательный импульс, свидетельствующий о приеме данных (рис.В4). Отсутствие подтверждения интерпретируется как ошибка.

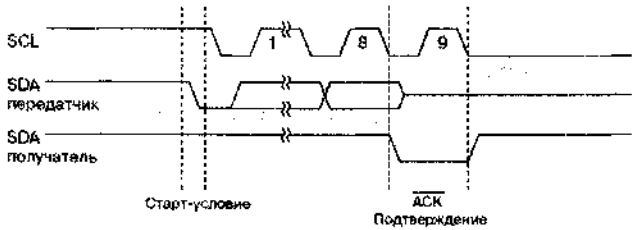


Рис.В4. Диаграмма подтверждения приема байта

При приеме последнего байта в серии, вместо сигнала ACK мастер может выставить состояние «Стоп», при этом подчиненный передатчик должен освободить линию.

Адресация

Каждое устройство, подключённое к шине, может быть программно адресовано по уникальному адресу. Для выбора приемника сообщения ведущий использует уникальную адресную компоненту в формате посылки. При использовании однотипных устройств, ИС часто имеют дополнительный селектор адреса, который может быть реализован как в виде дополнительных цифровых входов селектора адреса, так и в виде аналогового входа. При этом адреса таких однотипных устройств оказываются разнесены в адресном пространстве устройств, подключенных к шине. В обычном режиме используется 7-битная адресация. Все интегральные микросхемы, поддерживающие работу в стандарте шины I2C, имеют набор фиксированных адресов, перечень которых указан производителем в описаниях контроллеров.

Чтобы начать операцию обмена устройство «Master» выдает на шину «Start» - условие, за которым следует байт с адресом ведомого устройства, состоящий из 7-битного адреса устройства и одно битового флага операции «R/#W», определяющего направление обмена, причем 0 означает передачу от ведущего к ведомому, а 1 – чтение из ведомого устройства. Все биты по шине I2C передаются в порядке старший – младший, т.е. первым передается 7-й бит, последним 0-й бит. После того, как адрес послан, каждое устройство в системе сравнивает первые семь бит после сигнала «Старт» со своим адресом. При совпадении устройство полагает себя выбранным как ведомый-приёмник или как ведомый-передатчик, в зависимости от бита направления. Адрес ведомого может состоять из фиксированной и программируемой части.

За адресом могут следовать один или более информационных байтов (в направлении, определенном флагом «R/#W»), биты которых стробируются сигналом SCL из «Master» - устройства.

При выполнении операции чтения «Master» – абонент должен сопровождать прочитанный байт сигналом #ACK если необходимо прочитать следующий байт и не выдавать сигнал #ACK если требуется закончить чтение пакета. Логическая реализация протоколов на шине I2C может быть произвольной для каждой конкретной интегральной схемы.

Рис.В5. Передача данных от «Master» к «Slave»

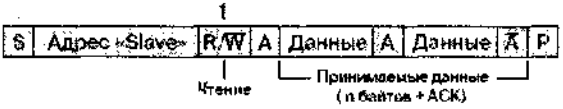


Рис.В6. Чтение данных из «Slave»

Рис.В7. Комбинированный формат – передача/чтение

На приведенных диаграммах «S» – условие «Start», «A» – бит подтверждения приема #ACK, «P» – условие «Stop».

Цифровой термометр на базе DS1621

I2C – абоненты жестко разделяются по классам: «Master» – и «Slave» – устройство. Тот факт, что сигнал SCL всегда генерируется «Master» – устройством означает, что «Master» – абонент может быть

достаточно легко реализован чисто программными средствами, так как все изменения на шине будут происходить только по сигналу SCL. Реализация Slave» – устройства требует аппаратной поддержки. В типичной микропроцессорной системе с применением I2C – устройств две линии портов отведены для управления линиями SDA и SCL. На рис.В8 представлена схема микропроцессорной системы контроля температуры. В схеме используется датчик DS1621, соединенный с микроконтроллером AT90S8535 по шине I2C.

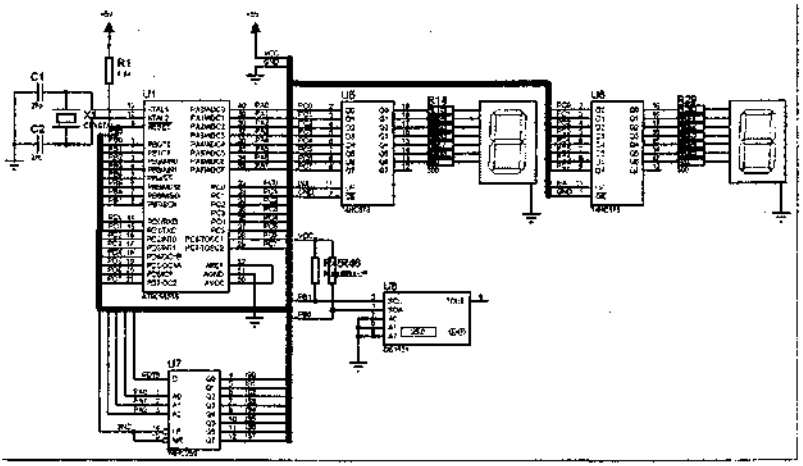


Рис.В8. Схема цифрового термометра на базе датчика DS1621

Микросхема DS1621 – это цифровой термометр и термостат. Прибор является законченным конвертером класса «температура – цифра» и не требует при использовании никаких внешних компонентов. Отрицательные значения температуры представлены в коде с дополнением до 2-х. Прибор состоит из пяти основных компонентов:

– Прецизионный датчик температуры;

– Аналого-цифровой преобразователь;

– Электроника двухпроводного интерфейса;

– Регистр данных;

– Компаратор термостата.

Ведущий шины может периодически считывать значение из температурного регистра, который содержит результат предыдущего преобразования.

Первый байт, передаваемый ведущим устройством после условия «Старт», содержит адрес устройства. Для DS1621 это 1001. Далее следуют 3 бита выбора устройства A2, A1, A0. Если используется только одно устройство на шине I2C, то значения этих бит может быть 0 при условии, что одноименные выводы микросхемы заземлены. Последний бит – бит чтения/записи – определяет направление передачи данных. Далее следует команда, которая определяет дальнейшие действия по организации обмена данными. Набор команд для DS121 приводится в табл.B1.

Таблица B1. **Команды DS1621**

Команда	Описание	Код команды
Чтение температуры	Читает последнее значение темп. преобразования из регистра температуры	0xAA
Чтение счетчика	Читает значение отсчета, остающегося в счетчике	0xA8
Чтение TH	Чтение хранимого значения предела высокой температуры из регистра TH, R/W=0	0xA1
Чтение TL	Чтение хранимого значения предела низкой температуры из регистра TL, R/W=0	0xA2

Продолжение табл. B1.

Команда	**Описание**	**Код команды**
Запись TH	Запись значения предела высокой , температуры в регистра TH, R/W=1	0xA1
Запись TL	Запись значения предела низкой температуры в регистр TL, R/W=1	0xA2
Запись конфигурации	Запись данных конфигурации в регистр конфигурации, R/W=1	0xAC
Чтение конфигурации	Чтение данных из регистра конфигурации, R/W=0	0xAC
Начать преобразование	Начинает преобразование температуры, R/W=0	0xEE
Остановить преобразование	Останавливает преобразование температуры в непрерывном режиме, R/W=0	0x22

Состояние датчика DS1621, а также его режим работы определяется с помощью регистра состояния и управления:

DONE	THF	TLF	NVB	1	0	POL	1SHOT

DONE – бит завершения преобразования температуры. «1» - преобразование закончено, «0» - преобразование выполняется.
THF – этот бит устанавливается если температура превышает значение, установленное в регистре TH. Бит можно сбросить путем записи 0 в эту позицию, или отключением питания датчика.
TLF – этот бит устанавливается если температура меньше значения, установленного в регистре TL. Бит можно сбросить путем записи 0 в эту позицию, или отключением питания датчика.
NVB – флаг занятости разделяемой памяти. «1» - выполняется запись в ячейку памяти.
POL – бит полярности выходного сигнала TOUT. «1» – активный высокий уровень, «0» – активный низкий уровень.
1SHOT – если значение бита «1», преобразование температуры выполняется 1 раз. Если значение бита «0» – преобразование температуры выполняется непрерывно.

В табл.В2 представлено соответствие значения температуры цифровому коду.

Таблица В2. **Соответствие между температурой и кодом**

Температура, ^{0}C	Код (BIN)	Код (HEX)
+ 125	01111101 00000000	7B00h
+ 25	00011001 00000000	1900h
+ 0,5	00000001 00000000	0080h
0	00000000 00000000	0000h
– 0,5	11111111 10000000	FF80h
– 25	11100111 00000000	E700h
– 55	11001001 00000000	C900h

Пример В1. Написать программу для микроконтроллера AT90S8535, которая выполняет считывание температуры с датчика DS1621 по интерфейсу I2C и выводит значение температуры на 2 семисегментных индикатора.

Текст программы для микроконтроллера, созданной с использованием компилятора ImageCraft, представлен ниже.

```
#include <io8535v.h>        // подключение заголовочных файлов
#include <macros.h>
#define SETSDA() PORTB|=0b00000001              // уст. сигнала SDA
#define SETSCL() PORTB|=0b00000010              // уст. сигнала SCL
#define RSTSDA() PORTB&=~0b00000001  // сброс сигнала SDA
#define RSTSCL() PORTB&=~0b00000010  // сброс сигнала SCL
  // функция инициализации контроллера
void init_cpu()
{
  DDRA=0x07;      // 3 младших бита порта A настроены на выход
  DDRC=0xFF;      // порт C настроен на выход
  DDRD=0x80;      // настройка вывода 7 порта D на выход
  PORTD=0x80;     // установка 1 на выводе 7 порта D
}
// функция задержки; параметр – задержка в мс
void delay_ms(int ms)
{
  int c1, c2;   // объявление дополнительных переменных
  for(c1=0;c1<ms;c1++)   // в цикле выполняется «пустая»
  for(c2=0;c2<970;c2++) NOP();        // операция
}
  // функция задержки; параметр – количество тактов задержки
void delay_us(int st)
{
  int c1;               // объявление дополнительной переменной
  for(c1=0;c1<st;c1++) NOP(); // в цикле выполняется «пустая»
                                        // операция
}
// функция проверки «занятости» шины
char i2c_isbusy()
{
  char st;              // дополнительная переменная
  DDRB=0x00;      //настройка порта B на вход
  PORTB=0x00;     // отключение подтягивающих резисторов
  if((PINB&0x03)!=0x03) st=1;  // проверка состояния линий
  else st=0;    // функция возвращает 0 если шина свободна
```

```
   return st;   // иначе возвращается значение 1
}
// функция формирования сигнала начала передачи данных
void i2c_start()
{
   DDRB=0x03;       // настройка 2 младших бит порта B на выход
   SETSDA();        // установка сигнала SDA
   SETSCL();        // установка сигнала SCL
   delay_us(2);// задержка
   RSTSDA();        // сброс сигнала SDA
   delay_us(2);// задержка
   RSTSCL();        // сброс сигнала SCL
   delay_us(2);// задержка
}
// функция формирования сигнала завершения обмена данными
void i2c_stop()
{
   DDRB=0x03;       // настройка 2 младших бит порта B на выход
   RSTSDA();        // сброс сигнала SDA
   SETSCL();        // установка сигнала SCL
   delay_us(2);     // задержка
   SETSDA();        // установка сигнала SDA
   delay_us(2);     // задержка
}
   // функция посылки 1 байта; параметр – передаваемый байт
unsigned char i2c_sendbyte(char d)
{
 unsigned char i=8,acknowledge=0;    // дополнительные переменные
 do {             // в цикле выполняется посылка 8 бит
 if((d & 0x80)!=0) SETSDA();    // установка сигнала SDA если
   // старший бит равен 1
else RSTSDA();       // иначе сброс сигнала SDA
 d =d << 1;      // сдвиг передаваемого байта на 1 разряд влево
 delay_us(2);  // формирование задержки
// формирование тактового импульса
  SETSCL();   // установка сигнала SCL
delay_us(2);   // задержка
RSTSCL();            // сброс сигнала SCL
delay_us(2);   // задержка
 }
 while (--i);            // цикл выполняется 8 раз
```

```
// формирование тактового импульса для бита подтверждения
 DDRB=0x02;          // перевод линии SDA в трете состояние
 RSTSDA();           // сброс сигала SDA
 SETSCL();           // установка сигнала SCL
 delay_us(2);        // задержка
 if ((PINB&0x01)==0) acknowledge=1; // если на линии SCL низкий
          // уровень, бит подтверждения принят
 RSTSCL();           // сброс сигнала SCL
 delay_us(2);        // задержка
 DDRB=0x03;          // настройка 2 младших бит порта B на выход
 return acknowledge; // если бит подтверждения принят, возвр.1
}
   // функция чтения одного байта
unsigned char i2c_readbyte(unsigned char acknowledge)
{
 unsigned char i=8,i2c_data=0;   // дополнительные переменные
 char d;
 DDRB=0x02;                  // перевод линии SDA в третье состояние
 do {
 i2c_data<<=1;               // сдвиг на 1 разряд влево
 SETSCL();                   // установка сигнала SCL
delay_us(2);            // задержка
   d=PINB;                   // считывание состояния порта B
   d=d&0x01;                 // выделение 0 бита
 i2c_data|=d;            // копирование 0 бита в переменную
 RSTSCL();                   // сброс сигнала SCL
delay_us(2);            // задержка
 }
 while (--i);                // цикл выполняется 8 раз
 if (acknowledge)            // если необходим бит подтверждения
   {
   DDRB=0x03;                // настройка 2-х младших бит на вывод
   RSTSDA();                 // сброс сигнала SDA
   }
// формирование тактового импульса
 SETSCL();                   // установка сигнала SCL
 delay_us(2);            // задержка
 RSTSCL();                   // сброс сигнала SCL
DDRB=0x02;              // перевод линии SDA в третье состояние
 return i2c_data;            // функция возвращает принятый байт
}
```

```
// главная функция
int main ()
{
   int c1=0,c2=0;     // объявление дополнительных переменных
   int t1, t2;
char seg[10]={0b00111111, 0b00000110, 0b01011011, 0b01001111,
0b01100110, 0b01101101, 0b01111101, 0b00000111, 0b01111111,
0b01101111}; // данные с кодами символов для 7-сегм. индикатора
init_cpu();      // вызов функции инициализации контроллера
while(1)         // вечный цикл
{
while (i2c_isbusy()!=0;      // ждем пока линия не освободится
DDRB=0x03; // настройка 2-х младших бит порта B на выход
i2c_start();      // функции формирования сигнала начала передачи
i2c_sendbyte(0b10010000);          // отправка байта 0x90 (адрес)
!i2c_sendbyte(0xEE);               // отправка байта 0xEE (команда)
i2c_stop();              // формирование сигнала окончания передачи
i2c_start();             // начало передачи данных
i2c_sendbyte(0b10010000);          // отправка байта 0x90 (адрес)
i2c_sendbyte(0xAA); // отправка байта 0xAA
i2c_stop();              // завершение передачи данных
i2c_start();             // повторный старт
i2c_sendbyte(0b10010001);          // отправка команды чтения данных
c1=i2c_readbyte(1);           // считывание старшего байта температуры
c2=i2c_readbyte(0);           // считывание младшего байта температуры
i2c_stop();              // освобождение линии I2C
t1=c1/10;                // формирование значения температуры (десятки)
t2=c1-t1*10;   // формирование значения температуры (единицы)
PORTA=4;                 // включение левого индикатора
PORTC=seg[t2];     // вывод значения температуры на индикатор
PORTA=3;                 // включение правого индикатора
PORTC=seg[t1];
delay_ms(1000);      // задержка 1 с
}
}
```

На рис.B9 представлена диаграмма работы интерфейса I2C. Тактовый сигнал SCL формирует микроконтроллер на выводе PB1. На линии данных SDA (вывод PB0) устанавливается уровни лог.1 и 0 в соответствии с передаваемым байтом. После 8 тактовых импульсов, на

9 происходит проверка бита подтверждения, который формирует ведомое устройство (датчик температуры DS1620).

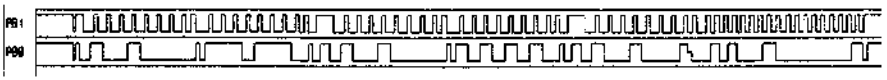


Рис.В9. Временная диаграмма работы интерфейса I2C

Приложение С

Шина 1-Wire

Однопроводной интерфейс 1-Wire , разработанный в конце 90-х годов фирмой Dallas Semiconductor Corp., регламентирован разработчиками для применения в трех основных сферах-приложениях:

– приборы в специальных корпусах MicroCAN для решения проблем идентификации, переноса или преобразования информации (технология iButton);

– программирование встроенной памяти интегральных компонентов;

– системы автоматизации (технология сетей 1-Wire-сетей).

Преимущества 1-Wire:

- простое и оригинальное решение адресуемости абонентов;
- несложный протокол;
- простая структура линии связи;
- малое количество необходимых компонентов;
- легкое изменение конфигурации сети;
- значительная протяженность линий связи;
- исключительная дешевизна всей технологии в целом.

1-Wire-net представляет собой информационную сеть, использующую для осуществления цифровой связи одну линию данных и один возвратный (или общий) провод. Таким образом, для реализации

среды обмена этой сети могут быть применены доступные кабели, содержащие неэкранированную витую пару той или иной категории, и даже обычный телефонный провод. Такие кабели при их прокладке не требуют наличия какого-либо специального оборудования, а ограничение максимальной длины однопроводной линии регламентировано разработчиками на уровне 300м. Основой архитектуры 1-Wire-сетей, является топология общей шины, когда каждое из устройств подключено непосредственно к единой магистрали, без каких-либо каскадных соединений или ветвлений. При этом в качестве базовой используется структура сети с одним ведущим мастером и многочисленными ведомыми. Хотя существует ряд специфических приемов организации работы однопроводных систем в режиме мультимастера. Конфигурация любой 1-Wire-сети может произвольно меняться в процессе ее работы, не создавая помех дальнейшей эксплуатации и работоспособности всей системы в целом, если при этих изменениях соблюдаются основные принципы организации однопроводной шины. Эта возможность достигается благодаря присутствию в протоколе 1-Wire-интерфейса специальной команды поиска ведомых устройств (поиск ПЗУ), которая позволяет быстро определить новых участников информационного обмена. Стандартная скорость отработки такой команды составляет ~75 узлов сети в секунду.

Благодаря наличию в составе любого устройства, снабженного сетевой версией 1-Wire-интерфейса, уникального индивидуального адреса (отсутствие совпадения адресов для приборов, когда-либо выпускаемых Dallas Semiconductor Corp., гарантируется самой фирмой-производителем), такая сеть имеет практически неограниченное адресное пространство. При этом каждый из однопроводных приборов сразу готов к использованию в составе 1-Wire-сети, без каких-либо

дополнительных аппаратно-программных модификаций. Однопроводные компоненты являются самотактируемыми полупроводниковыми устройствами, в основе обмена информацией между которыми, лежит управление изменением длительности временных интервалов импульсных сигналов в однопроводной среде и их измерение. Передача сигналов, для 1-Wire-интерфейса, асинхронная и полудуплексная, а вся информация, циркулирующая в сети, воспринимается абонентами либо как команды, либо как данные. Команды сети генерируются мастером и обеспечивают различные варианты поиска и адресации ведомых устройств, определяют активность на линии даже без непосредственной адресации отдельных компонентов, управляют обменом данными в сети и т.д.

Стандартная скорость работы 1-Wire-сети, которая составляет 15,4 Кбит/сек, была выбрана, во-первых, с учетом обеспечения максимальной надежности передачи данных на большие расстояния, и, во-вторых, с учетом быстродействия наиболее широко распространенных типов микроконтроллеров, которые в основном должны использоваться при реализации ведущих устройств однопроводной шины. Это значение скорости обмена может быть уменьшено до любого возможного значения благодаря введению принудительной задержки между передачей в линию отдельных битов данных (растягиванию временных слотов протокола). Или увеличено за счет перехода на специальный ускоренный режим обмена (скорость Overdrive – до 125Кбит/сек), который допускается для отдельных типов однопроводных компонентов на небольшой по расстоянию, качественной, не перегруженной другими приборами линии связи.

Отдельные виды адаптеров, которые позволяют наделить любой персональный компьютер возможностью обслуживать в качестве мас-

тера 1-Wire-сеть, выпускаются самой фирмой Dallas Semiconductor Corp. К ним относятся адаптеры для параллельного порта типа DS1410E, для COM-порта типа DS9097E и DS9097U, для USB-порта типа DS9490R. Эти приборы имеют различные функциональные возможности и конструктивные особенности, что обеспечивает разработчику максимальную свободу выбора при конструировании.

Часто в качестве ведущего однопроводной шины выступает не компьютер, а простейший универсальный микроконтроллер. Для организации его сопряжения с 1-Wire-сетью используются различные программно-аппаратные методы.

Четырехканальный однопроводной АЦП типа DS2450 и двухканальный однопроводной счетчик, совмещенный с буферной памятью, типа DS2423 позволяют решать задачи, связанные с оцифровкой аналоговых и импульсно-временных сигналов. Наиболее незаменимыми при автоматизации являются универсальные сдвоенные адресуемые транзисторные ключи типа DS2406P. На базе этих устройств может быть реализована масса применений, и, прежде всего, узлы контроля логических состояний (уровней) и схемы обслуживания датчиков "сухого контакта", а также разнообразные ключевые схемы. именно благодаря использованию этих компонентов осуществляется сбор дискретной информации с территориально рассредоточенных датчиков (мониторов дверей, контакторов положения арматуры, любых датчиков имеющих выход ДА/НЕТ, как-то датчики положения, прохода, присутствия, пожарной и охранной сигнализации и т.д.). Подобные же приборы обеспечивают управление переключением любых видов силового оборудования, которые имеют два рабочих состояния: включено/выключено (нагревателей, кондиционеров, моторов, вентиляторов, арматурных задвижек и т.д.). Значительно расширяет возможности

однопроводных сетей по аналоговому управлению рассредоточенным, в том числе силовым, оборудованием цифровой потенциометр DS2890 укомплектованный сетевым 1-Wire-интерфейсом. Используя этот прибор можно создавать самые разнообразные системы удаленного безударного управления, благодаря возможности плавного изменения аналогового регулирующего сигнала по 256 градациям.

Наиболее популярными ведомыми компонентами, на базе которых реализовано, пожалуй, наибольшее количество применений, являются цифровые термометры типа DS18S20. Рассмотрим принципы работы и программирования устройств с 1-Wire на примере датчика температуры DS18B20 и микроконтроллера AT90S8535.

Широко распространенная микросхема цифрового термометра DS18B20 обеспечивает измерение температуры в диапазоне –55..+125°C. Разрешение температуры может быть от 9 до 12 разрядов (настраивается). Информация предается из/в DS18B20 передается по одно проводному интерфейсу. DS18B20 допускает напряжение питания от +3 до +5,5В. В режиме ожидания потребляемый ток близок к нулю (менее 1мкА), а во время преобразования температуры он равен примерно 1мА. Процесс преобразования длится максимум 750мс. Они позволяют не только осуществлять непосредственный мониторинг температуры в режиме реального времени, но и благодаря наличию встроенной энергонезависимой памяти температурных уставок, могут обеспечивать приоритетную сигнализацию в линию о факте выхода контролируемого параметра за пределы заданных значений.

Принцип действия такого цифрового датчика температуры основан на подсчете количества импульсов, вырабатываемых генератором с низким температурным коэффициентом во временном интервале, который формируется генератором с большим температурным коэф-

фициентом. Счетчик инициализируется значением, соответствующим –55°С (минимальной измеряемой температуре). Если счетчик достигает нуля перед тем, как заканчивается временной интервал (это означает, что температура больше –55°С), то регистр температуры, который также инициализирован значением –55°С, инкрементируется. Одновременно счетчик предустанавливается новым значением, которое задается схемой формирования наклона характеристики. Эта схема нужна для компенсации параболической зависимости частот генераторов от температуры. Счетчик снова начинает работать, и если он опять достигает нуля, когда интервал еще не закончен, процесс повторяется снова. Схема формирования наклона загружает счетчик значениями, которые соответствуют количеству импульсов генератора на один градус для каждого конкретного значения температуры. По окончанию процесса преобразования регистр температуры будет содержать значение температуры.

На рис.С1 показаны основные узлы датчика.

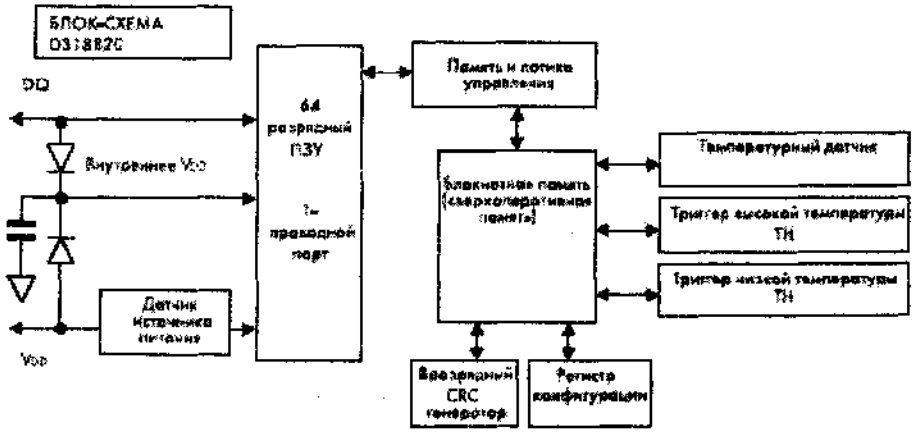


Рис.С1. Блок-схема датчика температуры DS18B20

Каждый датчик содержит уникальный 48-битный серийный номер, записанный при производстве, поэтому на одну шину могут быть подключены несколько датчиков. Это позволяет разместить датчики в разных местах и собирать информацию по простому 2-х проводному кабелю. Кроме серийного номера в ПЗУ содержится код семейства (для DS18B20 это 10h) и контрольная сумма. Также DS18B20 имеет температурный датчик, регистр конфигурации, энергозависимые температурные сигнальные триггеры TH и TL, промежуточное ОЗУ объемом 8 байт (рис.С2).

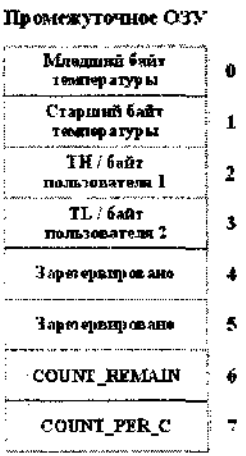


Рис.С2. Карта памяти DS18B20

Байты TH и TL представляют собой температурные пороги, с которыми сравниваются 8 бит каждого измеренного значения температуры (младший бит отбрасывается). С помощью специальной команды можно организовать сигнализацию выхода температуры за пределы этих порогов. Если такая функция не нужна, байты TH и TL можно использовать для хранения любых данных пользователя.

Считывание значения измеренной температуры, а также передача команды начала преобразования и других команд производится с помощью 1-проводного интерфейса (1-Wire). На основе этого интерфейса фирма DALLAS создала сеть, называемую MicroLAN. Для работы в этой сети выпускается целый ряд устройств, таких как адресуемые ключи, АЦП, термометры, часы реального времени, цифровые потенциометры.

Протокол, который используется 1-проводным интерфейсом, достаточно прост. В любой момент времени на 1-проводной шине можно выделить устройство-мастер, которым может быть микропроцессор или компьютер, и подчиненное устройство. Если на шине присутствуют только мастер и всего одно подчиненное устройство, можно опустить всё то, что связано с адресацией устройств. В результате требуется знать лишь протокол передачи байтов, которые могут являться командами или данными.

На рис.С3 показана аппаратная конфигурация интерфейсной части DS18B20 и мастера шины. У каждого 1-проводного устройства к шине подключен вход приемника и выход передатчика с открытым стоком. Открытый сток позволяет подключать к шине множество устройств, обеспечивая логику «монтажное или». Генератор тока 5мкА обеспечивает на входе 1-проводного устройства низкий логический уровень, когда шина не подключена. Так как линия тактового сигнала отсутствует, обмен является асинхронным. Это означает, что в процессе обмена нужно достаточно точно выдерживать требуемые временные соотношения.

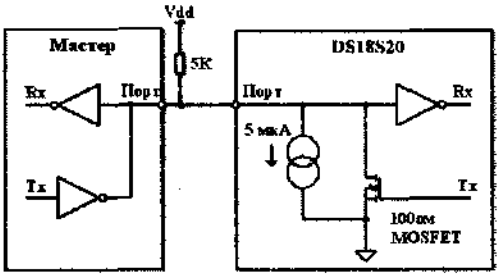


Рис.С3. Аппаратная конфигурация интерфейсной части

1-проводная шина оперирует с TTL-уровнями, т.е. логическая единица представлена уровнем напряжения около 5В, а логический ноль – напряжением вблизи 0В. В исходном состоянии на линии присутствует уровень логической единицы, который обеспечивается подтягивающим резистором номиналом около 5кОм.

Инициатором обмена по 1-проводной шине всегда выступает мастер. Все пересылки начинаются с процесса инициализации. Инициализация производится в следующей последовательности (рис.С4):

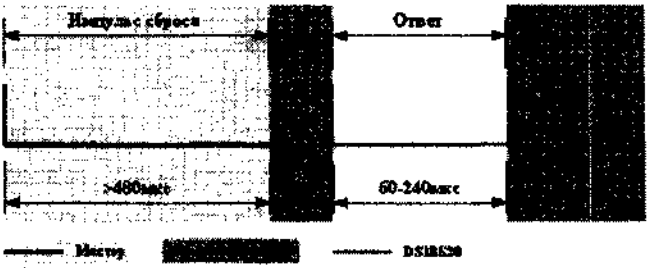


Рис.С4. Инициализация обмена по 1-проводной шине.

Мастер посылает импульс сброса (reset pulse) – сигнал низкого уровня длительностью не менее 480 мкс. За импульсом сброса следует ответ подчиненного устройства (presence pulse) – сигнал низкого

уровня длительностью 60 - 240 мкс, который генерируется через 15 - 60 мкс после завершения импульса сброса.

Ответ подчиненного устройства даёт мастеру понять, что на шине присутствует термометр и он готов к обмену. После того, как мастер обнаружил ответ, он может передать термометру одну из команд. Передача ведётся путём формирования мастером специальных временных интервалов (time slots). Каждый временной интервал служит для передачи одного бита. Первым передаётся младший бит. Интервал начинается импульсом низкого уровня, длительность которого лежит в пределах 1 – 15 мкс. Поскольку переход из единицы в ноль менее чувствителен к ёмкости шины (он формируется открытым транзистором, в то время как переход из ноля в единицу формируется подтягивающим резистором), именно этот переход используют 1-проводные устройства для синхронизации с мастером. В подчиненном устройстве запускается схема временной задержки, которая определяет момент считывания данных. Номинальное значение задержки равно 30 мкс, однако, оно может колебаться в пределах 15 – 60 мкс. За импульсом низкого уровня следует передаваемый бит. Он должен удерживаться мастером на шине в течение 60 – 120 мкс от начала интервала. Временной интервал завершается переводом шины в состояние высокого уровня на время не менее 1 мкс. Нужно отметить, что ограничение на это время сверху не накладывается. Аналогичным образом формируются временные интервалы для всех передаваемых бит (рис.С5):

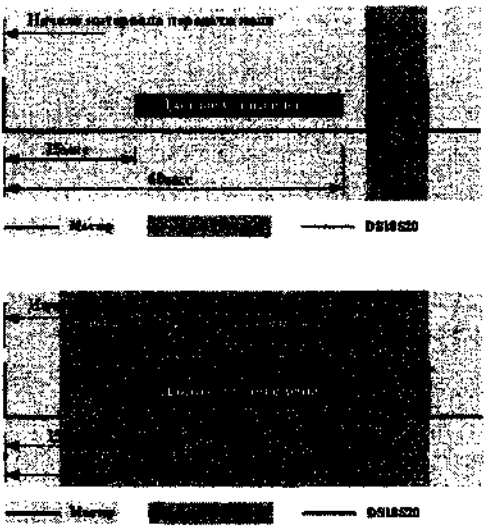


Рис.С5. Передача бита по 1-проводной шине

Первой командой, которую должен передать мастер для DS18B20 после инициализации, является одна из команд функций ПЗУ. Всего DS18B20 имеет 5 команд функций ПЗУ:

- **Read ROM [33h]**. Эта команда позволяет прочитать содержимое ПЗУ. В ответ на эту команду **DS18B20** передает 8-битный код семейства (10h), затем 48-битный серийный номер, а затем 8-битную CRC для проверки правильности принятой информации.
- **Match ROM [55h]**. Эта команда позволяет адресовать на шине конкретный термометр. После этой команды мастер должен передать нужный 64-битный код, и только тот термометр, который имеет такой код, будет «откликаться» до следующего импульса сброса.
- **Skip ROM [CCh]**. Эта команда позволяет пропустить процедуру сравнения серийного номера и тем самым сэкономить время в системах, где на шине имеется всего одно устройство.

- **Search ROM [F0h]**. Эта довольно сложная в использовании команда позволяет определить серийные номера всех термометров, присутствующих на шине.
- **Alarm Search [ECh]**. Эта команда аналогична предыдущей, но «откликаться» будут только те термометры, у которых результат последнего измерения температуры выходит за предустановленные пределы TH и TL.

Поскольку у нас всего одно устройство, наиболее подходящей для нас функцией является функция Skip ROM. Кроме неё ещё может быть полезной функция Read ROM, которая позволяет идентифицировать подключенное на шину устройство по его коду семейства и серийному номеру.

При приеме данных от подчиненного устройства временные интервалы для принимаемых бит тоже формирует мастер. Интервал начинается импульсом низкого уровня длительностью 1 – 15 мкс. Затем мастер должен освободить шину, чтобы дать возможность термометру вывести бит данных. По переходу из единицы в ноль DS18B20 выводит на шину бит данных и запускает схему временной задержки, которая определяет, как долго бит данных будет присутствовать на шине. Это время лежит в пределах 15 – 60 мкс. Для того чтобы данные на шине, которая всегда обладает некоторой ёмкостью, гарантировано установились, требуется некоторое время. Поэтому момент считывания данных мастером должен отстоять как можно дальше, но не более чем на 15 мкс от начала временного интервала (рис.С6):

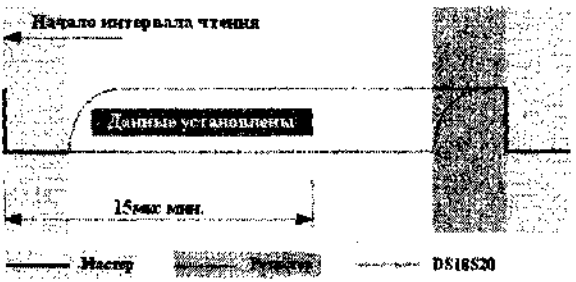


Рис.С6. Чтение бита по 1-проводной шине

Прием байта начинается с младшего бита. Вначале идет байт кода семейства. За кодом семейства идет 6 байт серийного номера, начиная с младшего. Затем идет байт контрольной суммы (CRC). В вычислении байта контрольной суммы принимают участие первые 7 байт, или 56 передаваемых бит. После приема данных мастер должен вычислить контрольную сумму и сравнить получившееся значение с переданной CRC. Если эти значения совпадают, значит, прием данных прошел без ошибок. Можно также вычислить контрольную сумму для всех 64 принятых бит, которая в этом случае должна быть равна нулю.

После обработки одной из команд функций ПЗУ, DS18B20 способен воспринимать еще несколько команд:

- **Write Scratchpad [4Eh]**. Эта команда позволяет записать данные в промежуточное ОЗУ **DS18B20**.
- **Read Scratchpad [BEh]**. Эта команда позволяет считать данные из промежуточного ОЗУ.
- **Copy Scratchpad [48h]**. Эта команда копирует байты TH и TL из промежуточного ОЗУ в энергонезависимую память. Эта операция требует около 10мс.

- **Convert T [44h]**. Эта команда запускает процесс преобразования температуры.
- **Recall E2 [B8h]**. Эта команда действует обратным образом по отношению к команде Copy Scratchpad, т.е. она позволяет считать байты TH и TL из энергонезависимой памяти в промежуточное ОЗУ. При включении питания эта команда выполняется автоматически.
- **Read Power Supply [B4h].** Эта команда позволяет проверить, использует ли DS18B20 паразитное питание. Дело в том, что DS18B20 можно подключать всего с помощью двух проводов, в этом случае для питания используется линия данных. Особенности этого режима мы здесь рассматривать не будем.

При использовании DS18B20 только для измерения температуры нужны всего две из этих команд: Convert T и Read Scratchpad. Последовательность действий при измерении температуры должна быть следующей:

- Посылаем импульс сброса и принимаем ответ термометра.
- Посылаем команду Skip ROM [CCh].
- Посылаем команду Convert T [44h].
- Формируем задержку минимум 750мс.
- Посылаем импульс сброса и принимаем ответ термометра.
- Посылаем команду Skip ROM [CCh].
- Посылаем команду Read Scratchpad [BEh].
- Читаем данные из промежуточного ОЗУ (8 байт) и CRC.
- Проверяем CRC, и если данные считаны верно, вычисляем температуру.

Микропроцессорная система контроля температуры

На рис.С7 представлена схема подключения датчика температуры DS18B20 к микроконтроллеру AT90S8535. Микроконтроллер под-

ключается к компьютеру и по интерфейсу RS232 передает значение температуры. Для согласования уровней напряжений COM-порта компьютера и микроконтроллера использована микросхема-преобразователь уровней MAX232.

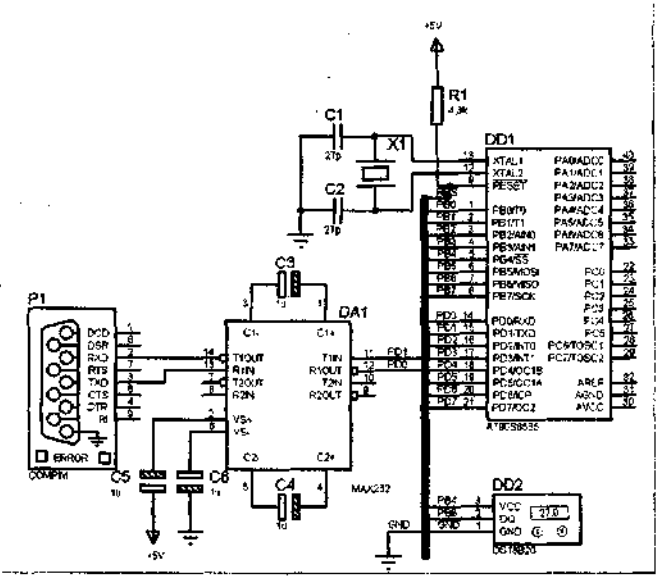


Рис.С7. Схема цифрового термометра на базе датчика DS18B20

Пример С1. Написать программу для микроконтроллера AT90S8535, которая выполняет считывание температуры с датчика DS18B21 по интерфейсу 1-Wire и передает значение температуры по COM-порту.

1. Запускаем среду разработки CodeVision и создаем новый проект, выбрав из меню File команду New. В появившемся окне (рис.С8) отметить позицию Project.

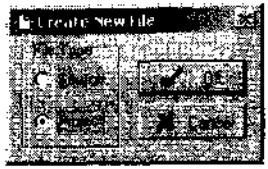


Рис.С8. Окно создания нового проекта

В появившемся окне с предложением использования мастера проекта (CodeWizard) нажать на кнопку «NO». Далее следует ввести название проекта (например TestWire).

2. В появившемся окне настроек проекта установить значения, как показано на рис.С9.

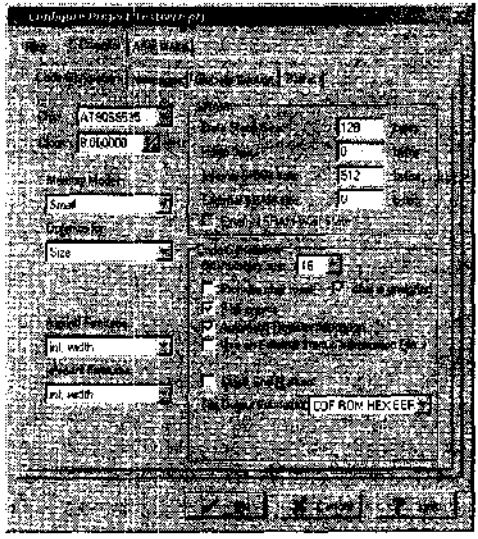


Рис.С9. Окно настроек проекта

3. Добавляем в проект файл с исходным кодом. Для этого в меню File выбрать команду New. Программа для микроконтроллера AT90S8535, созданная в компиляторе CodeVision, представлена ниже.

```
#asm                          // указание выводов, к которым
подключается датчик
 .equ __w1_port=0x18
 .equ __w1_bit=6              // линия данных на выводе 6 порта B
#endasm
#include "1wire.h"            // подключение заголовочных файлов
#include "90s8535.h"
#include "delay.h"
#include "math.h"
unsigned char delay;          // задержка опроса датчика
bit start_stop;          // если бит =1, то начать преобразование
температуры
// объявление прототипов функций
void main(void);              // основная функция программы
void init(void);              // инициализация микроконтроллера
void temp(void);              // опрос датчика температуры
void set_config(unsigned char config);  // изменяет байт конфигурации
датчика
  // подпрограмма обработки прерывания
interrupt [UART_RXC] void uart_rxc(void) // вызывается по приему
символа с UART
{
  unsigned char temp;              // объявление дополнительной
переменной
  temp = UDR;                      // считать принятый байт
  if(temp==0xBC)   start_stop = 1;       // если принята команда 0xBC,
начать преобр.
}
// функция опроса датчика температуры
void temp()
{
  unsigned char buf[9];            // буфер считанных данных
  unsigned char i;            // дополнительные переменные
  unsigned char k;
  if(w1_init())               // вызов функции инициализации датчика
  {
        w1_write(0xCC);  // пропуск ПЗУ
        w1_write(0x44);  // преобразование температуры
        delay_ms(delay);  // ожидание конца преобразования
температуры
  for (k=0; k<3; k++)              // цикл чтения блокнотной памяти
датчика
```

```
{
        w1_init();              // инициализация
        w1_write(0xCC);  // пропуск ПЗУ
        w1_write(0xBE);  // чтение блокнотной памяти
        for (i=0; i<9; i++) // считать девять байт
        {
                buf[i] = w1_read();        // запись считанного байта в
буфер
        }
        if(buf[8]==w1_dow_crc8(&buf[0], 8)) // если ошибка, то
повторить цикл
        break;
        }
        if(buf[8]!=w1_dow_crc8(&buf[0], 8)) // выйти, если датчик не
отвечает
        return;
        else                            // пересылка двух байт температуры в
компьютер                {
        while(!USR.5);                  // если бит UDRE == 1, то отправить
байт
                UDR = buf[0];
        while(!USR.5);
                UDR = buf[1];      // если бит UDRE == 1, то отправить
байт
        }
  }
  start_stop=0;                 // сброс флага разрешения преобразования
  return;
}
  // функция установки байта конфигурации
void set_config(unsigned char config)
{
  while(!w1_init());            // вызов функции инициализации датчика
                                // повторять пока датчик не ответит
  switch(config)
  {
        // разрешающая способность
        case 0                          :       // 9 бит
                config = 0b00011111;
                delay = 100;
                break;
        case 1:                                 // 10 бит
```

```
            config = 0b00111111;
            delay = 188;
            break;
        case 2:                                  // 11 бит
            config = 0b01011111;
            delay = 375;
            break;
        case 3:                                  // 12 бит
            config = 0b01111111;
            delay = 750;
            break;
        default:                                 // ошибка
            return;
    }
    // перепрошивка байта конфигурации датчика 9-12 разрядов
    w1_write(0xCC);          // пропуск ПЗУ
    w1_write(0x4E);          // запись в блокнотную память

    w1_write(0x00);          // запись Твыс
    w1_write(0x00);          // запись Тниз
    w1_write(config);        // запись байта конфигурации
}
    // функция инициализации микроконтроллера
void init_cpu(void)
{
    DDRB = 0b00010100;   // подключение DS18B20: 6 - DQ, 4 - UCC
    PORTB = 0xFF;        // вывести единицу на PortB.4 - ножка
питания датчика
        // инициализация UART
    DDRD = 0b10000010;   // настройка выводов UART
    UCR = 0b10011000;    // разрешение приема/передачи и прерывания
// когда символ поступил в UDR
    UBRR = 51;                   // скорость обмена 9600 Бит/с при
частоте 8 MHz
        // конфигурация по умолчанию
    delay      = 100;            // задержка
    start_stop = 0;  // флаг разрешения преобразования температуры

    #asm("sei")              // разрешить прерывания
}
void main(void)      // главная функция
{
```

```
   init_cpu();                    // вызов функции инициализации
микроконтроллера
   set_config(0);                        // установка 9-битного режима
преобразования T
   while(1)                       // вечный цикл
   {
         if(start_stop!=0) temp(); // если бит разрешения преобразования
         // температуры установлен, вызвать функцию преобразования
температуры
   }
}
```

Сохранив файл на диск с именем TestWire.c, проект можно компилировать. Для этого в меню Project выбрать команду Compile или нажать клавишу F9. В каталоге, где расположены файлы проекта, будет создан файл для прошивки микроконтроллера TestWire.hex.

ЛИТЕРАТУРА

1. Арчер Т, Уайтчепел Э. Visual C++ .Net. Библия пользователя. – М.: «Диалектика», 2003. –1210 с.
2. Грегори К. Использование Visual C++ 6. Специальное издание. – М.; СПб.; К.: «Вильямс», 2000. – 864 с.
3. Гук М. Аппаратные интерфейсы ПК. Энциклопедия. С-П.: «Питер», 2002, 527с.
4. Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы Atmel. –М.: Диалектика, 2004.
5. Луис Д. С и С++. Справочник. –М: Бином, 1997. – 590 с.
6. Пей Ан. Сопряжение ПК с внешними устройствами. –М.: ДМК, 2001.
7. Шпак Ю.А. Программирование на языке С для AVR и PIC. – К.: МК-Пресс, 2006.
8. Стешенко В.Б. Практика автоматизированного проектирования радиоэлектронных устройств. – М.: Нолидж, 2001. – 765 с.
9. www.atmel.com
10. www.atmel.ru
11. www.labcenter.co.uk
12. www.ftdichip.com

Рябенький Владимир Михайлович

Ходаков Виктор Егорович

Ушкаренко Александр Олегович

КОМПЬЮТЕРНОЕ УПРАВЛЕНИЕ ВНЕШНИМИ УСТРОЙСТВАМИ ЧЕРЕЗ СТАНДАРТНЫЕ ИНТЕРФЕЙСЫ

Підп. до друку 10.01.2008.

Формат 60x84/16. Папір офсетний. Наклад пр.
Обл.-вид. арк.23,75. Ум.друк.арк.22,09. Гарн.Таймс.

Видавництво "Олді-плюс", м. Херсон, вул. Комсомольська, 2. оф. 108,
тел./факс (0552) 42-08-19; e-mail: grin@public.kherson.ua
Ліцензія сер. ХС № 2 від 16.08.2000 р