



Павел Агуров



Последовательные интерфейсы ПК

Практика программирования



+ CD-ROM

- Протоколы последовательной связи
- Стандарты RS232/RS485
- Работа в DOS, Windows 98/ME/NT/2000/XP
- Примеры на языке Pascal
- Все уровни доступа
- Поддержка Plug and Play



МАСТЕР ПРОГРАММ

УДК 681.3.06
ББК 32.973-018
А23

Агуров П. В.

А23 Последовательные интерфейсы ПК. Практика программирования. — СПб.: БХВ-Петербург, 2004. — 496 с.: ил.
ISBN 5-94157-468-1

Книга представляет собой практическое руководство по программированию последовательных интерфейсов. В первой части книги представлены теоретические сведения о последовательных интерфейсах, во второй — практические примеры и листинги программ на языках Pascal и Delphi, а третья содержит справочные данные, облегчающие поиск необходимой информации. В приложениях приведены дополнительные сведения и ответы на часто задаваемые вопросы. Большое количество практических советов, примеров программ, а также последовательность и простота изложения позволят читателю уверенно овладеть изложенным в книге материалом. Для удобства читателей все исходные коды приводятся на прилагаемом к книге компакт-диске.

Для программистов, занятых в сфере промышленной автоматизации

УДК 681.3.06
ББК 32.973-018

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Алексей Семенов</i>
Компьютерная верстка	<i>Натальи Смирновой</i>
Корректор	<i>Наталья Першакова</i>
Оформление серии	<i>Via Design</i>
Дизайн обложки	<i>Игоря Цырульников</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 13.02.04.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 39,99.

Тираж 3000 экз. Заказ №

"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02 от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-468-1

© Агуров П. В., 2004
© Оформление, издательство "БХВ-Петербург", 2004

Содержание

Введение.....	1
Для кого эта книга.....	1
Структура книги.....	2
Краткое описание глав.....	3
Программные требования.....	6
О программном коде.....	6
Обозначения.....	7
Аппаратные требования.....	7
Благодарности.....	7
Обратная связь.....	7
 Часть I. Протоколы и интерфейсы.....	9
 Глава 1. Стандарты последовательной связи	11
1.1. Стандарты последовательной связи	11
1.1.1. Протокол RS-232.....	13
1.1.2. Протокол RS-422A.....	15
1.1.3. Протокол RS-423A.....	15
1.1.4. Протокол RS-485.....	15
1.1.5. Протокол RS-499.....	16
1.1.6. Протокол RS-562.....	16
1.1.7. Протокол V.24.....	17
1.1.8. Протокол V.28.....	17
1.1.9. Протокол V.35.....	17
1.1.10. Протокол X.21.....	18
1.1.11. Рекомендация X.21bis.....	18
1.1.12. Краткое сравнение RS-протоколов	18
1.2. Принципы последовательной связи.....	19
1.3. Разъемы коммуникационного порта.....	20
1.3.1. Сигнальная "земля" (AB/SG).....	22
1.3.2. Защитная "земля" (AA).....	22
1.3.3. Передаваемые данные (BA/TxD/TD).....	23
1.3.4. Принимаемые данные (BB/RxD/RD).....	23
1.3.5. Запрос передачи (CA/RTS).....	23

1.3.6. Готовность к передаче (CB/CTS).....	24
1.3.7. Готовность DCE (CC/DSR).....	25
1.3.8. Готовность DTE (CD/DTR)	25
1.3.9. Индикатор вызова (CE/RI)	25
1.3.10. Обнаружение несущей (CF/DCD).....	25
1.3.11. Детектор качества сигнала (CG/SQ)	26
1.3.12. Переключатель скорости передачи данных от DTE (CH)	26
1.3.13. Переключатель скорости передачи данных от DCE (CI)	26
1.3.14. Готовность к приему (CJ).....	26
1.3.15. Местный шлейф (LL).....	27
1.3.16. Удаленный шлейф (RL)	27
1.3.17. Индикатор тестирования (TM)	27
1.3.18. Синхронизация передачи от DTE (DA).....	28
1.3.19. Синхронизация передачи от DCE (DB/TC)	28
1.3.20. Синхронизация приема от DCE (DD/RC)	28
1.3.21. Передаваемые данные дополнительного канала (SBA/STD).....	28
1.3.22. Принимаемые данные дополнительного канала (SBB/SRD).....	28
1.3.23. Запрос передачи по дополнительному каналу (SCA/SRTS)	29
1.3.24. Готовность к передаче по дополнительному каналу (SCB/SCTS)	29
1.3.25. Обнаружение несущей дополнительного канала (SCF/SDCD)	29
1.4. Ресурсы IBM PC для последовательной связи	29
1.4.1. Сервисы BIOS.....	29
1.4.2. Коммуникационные порты	30
1.4.3. Использование прерываний	30
1.4.4. Прямое программирование портов в Windows.....	30
1.4.5. Функции Windows	31
Глава 2. Протоколы	32
2.1. Что такое протокол обмена.....	32
2.2. ASCII-протокол передачи данных.....	33
2.3. Бинарный протокол передачи данных.....	35
2.4. Предотвращение потери данных.....	35
2.4.1. Прерывания и потоки	35
2.4.2. Буферизация.....	37
Синхронизация в DOS.....	38
Синхронизация в Windows	38
2.4.3. Обратная связь.....	40
2.5. Методы обнаружения ошибок передачи	40
2.5.1. Контроль четности	41
2.5.2. Контрольная сумма	42
Простая контрольная сумма.....	42
Контрольная сумма LRC	42
Контрольная сумма CRC16	43

2.5.3. Стартовый байт.....	44
2.5.4. Другие способы повышения достоверности данных.....	45
Глава 3. Последовательные протоколы IBM PC.....	46
3.1. Мышь.....	46
3.1.1. Базовый протокол Microsoft Mouse.....	46
3.1.2. Протокол Microsoft Plus.....	47
3.1.3. Протокол 3D Serial Mouse.....	47
3.1.4. Протокол PC Mouse.....	48
3.1.5. Совместимость протоколов.....	48
3.2. Модем.....	48
3.2.1. Команды управления модемом.....	49
3.2.2. Регистры модемов.....	50
3.2.3. Протоколы передачи файлов.....	50
ASCII-протокол.....	50
Протокол XModem.....	51
Протокол XModem-CRC.....	52
Протокол XModem-1K.....	52
Протокол YModem.....	53
Протокол YModem-G.....	54
Протокол ZModem.....	54
Протокол BiModem.....	55
Протокол Kermit.....	55
Глава 4. Промышленные последовательные протоколы.....	56
4.1. Протокол MODBUS.....	56
4.1.1. Протокол MODBUS-ASCII.....	57
4.1.2. Протокол MODBUS-RTU.....	58
4.1.3. Поля MODBUS протокола.....	59
Поле "Адрес абонента".....	59
Поле "Код функции".....	60
Поле "Данные".....	60
Поле "Контрольная сумма".....	60
4.2. Протокол CAN.....	61
4.2.1. Характеристики протокола CAN.....	62
4.2.2. Обмен данными в протоколе CAN.....	63
4.2.3. Обнаружение ошибок в протоколе CAN.....	63
4.3. Протокол Profibus.....	64
Глава 5. COM-порты и Plug and Play.....	66
5.1. Кратко о Plug and Play.....	66
5.2. Запуск процедуры PnP.....	67
5.3. Если устройство не находится автоматически.....	68

5.4. Где хранится информация о найденных устройствах	69
5.4.1. Структура реестра Windows 98	69
5.4.2. Структура реестра Windows 2000	71
5.5. Алгоритм Plug and Play для COM-портов	73
5.5.1. Инициализация порта (<i>Port initialization</i>)	73
5.5.2. Обнаружение устройств (<i>Check for device</i>)	73
5.5.3. Установка устройства, фаза 1 (<i>COM port Setup-1</i>)	74
5.5.4. Ожидание ответа, фаза 1 (<i>Wait for response-1</i>)	74
5.5.5. Установка устройства, фаза 2 (<i>COM port Setup-2</i>)	74
5.5.6. Ожидание ответа, фаза 2 (<i>Wait for response-2</i>)	75
5.5.7. Получение идентификатора (<i>Collect device ID</i>)	75
5.5.8. Проверка отключения (<i>Verify Disconnect</i>)	76
5.5.9. Дежурное состояние (<i>Connect Idle</i>)	76
5.5.10. Состояние ожидания отключения (<i>Disconnect Idle</i>)	76
5.6. Формат данных для передачи PnP-идентификатора	77
5.7. Передача PnP-идентификатора	77
5.8. Поля PnP-идентификатора	78
5.8.1. Поле <i>Other ID</i>	80
5.8.2. Поле <i>Begin PnP</i>	80
5.8.3. Поле <i>PnP Rev</i>	80
5.8.4. Поле <i>EISA ID</i>	80
5.8.5. Поле <i>Product ID</i>	81
5.8.6. Поле <i>Extend</i>	81
5.8.7. Поле <i>Serial Number</i>	81
5.8.8. Поле <i>Class Name</i>	81
5.8.9. Поле <i>Device ID</i>	82
5.8.10. Поле <i>User Name</i>	82
5.8.11. Поле <i>Checksum</i>	82
5.8.12. Поле <i>End PnP</i>	82
5.8.13. Пример PnP-идентификатора	83
5.9. INF-файл и его структура	84
5.9.1. Секция <i>Version</i>	84
5.9.2. Секция <i>Manufacturer</i>	85
5.9.3. Секция <i>Destination Dirs</i>	88
Ключ <i>DefaultDescDir</i>	88
Ключи <i>file-list-section</i>	88
Ключ <i>dirid</i>	88
Ключ <i>subdir</i>	89
5.9.4. Секция описания модели	90
5.9.5. Секции <i>xxx.AddReg</i> и <i>xxx.DelReg</i>	91
5.9.6. Секция <i>xxx.Log Config</i>	92
5.9.7. Секция <i>xxx.CopyFiles</i>	93
5.9.8. Секция <i>Strings</i>	94
5.9.9. Связи секций	95

Часть II. ПРАКТИКА ПРОГРАММИРОВАНИЯ	97
Глава 6. Использование сервисов BIOS	99
6.1. Подготовка	99
6.2. Первая программа последовательной связи	100
6.3. Другие функции BIOS	102
Глава 7. Прямое программирование портов	104
7.1. Коммуникационные порты	104
7.2. Программа определения наличия COM-портов	105
7.3. Обходим ограничения BIOS	106
7.4. Чтение и запись с помощью модуля <i>RS232DOS</i>	111
Глава 8. Использование обработки прерываний	113
8.1. Прерывания	113
8.2. Модуль <i>RS232Int</i> для работы с прерываниями	115
8.3. Последовательный обмен с помощью прерываний	124
Глава 9. Переход в Windows	127
9.1. Переходим из DOS в Windows	127
9.2. Первая программа для Windows	131
9.3. Перенос программ из Windows 9x в Windows NT/2000	142
9.3.1. Получение доступа к портам в Windows 2000/XP	143
9.3.2. Расширение возможностей GiveIO	148
9.3.3. Работа с драйвером GiveIOEx	157
9.3.4. Еще немного о прямом доступе к портам	165
Глава 10. Использование функций Windows	166
10.1. Обзор функций Windows для работы с последовательными портами	166
10.2. Специальная настройка порта	167
10.3. Получение состояния линий модема	168
10.4. Используем функции Windows	168
10.5. Несколько замечаний о контроле четности	183
Глава 11. Использование потоков	185
11.1. Преимущества потоков	185
11.2. Создание потока опроса порта	185
Глава 12. Функции асинхронного доступа и события	194
12.1. Асинхронный доступ	194
12.2. События последовательного порта	197
12.3. Использование событий	197

Глава 13. Специальные коммуникационные функции	201
13.1. Выполнение дополнительных операций.....	201
13.2. Прямое управление драйвером	203
13.3. Обнаружение всех портов системы	222
13.4. Имена портов.....	225
13.5. APC-функции Windows 2000/XP	231
Глава 14. Реализация протоколов обмена.....	234
14.1. Прием и передача простого пакета	234
14.2. Создание компонента	241
14.3. Реализация ASCII-протокола.....	254
14.4. Реализация бинарного протокола.....	262
Глава 15. Вычисление контрольных сумм	273
15.1. Вычисление простой контрольной суммы	273
15.2. Вычисление LCR.....	274
15.3. Вычисление CRC16.....	275
15.4. Вычисление CRC32.....	279
Глава 16. Интерфейс RS-485	282
16.1. Стандарт RS-485	282
16.2. Как работает COM-порт IBM PC.....	283
16.3. Реализация RS-485	288
Глава 17. Работа Plug and Play.....	291
17.1. Эмулятор устройства.....	291
17.2. Установка драйвера устройства.....	300
17.3. Обнаружение изменений	303
Часть III. Справочник	309
Глава 18. Сервис BIOS INT14H/INT21H	311
18.1. Сервис BIOS INT14H.....	311
18.1.1. Функция 00: инициализация порта	312
18.1.2. Функция 01: посылка символа в порт.....	313
18.1.3. Функция 02: получение символа	313
18.1.4. Функция 03: получить статус порта.....	313
18.1.5. Функция 04: расширенная инициализация (System/2)	314
18.1.6. Функция 05: расширенное управление портом (System/2)	315
18.2. Сервис BIOS INT21H.....	316
18.2.1. Функция 03H: вспомогательный ввод.....	317
18.2.2. Функция 04H: вспомогательный вывод.....	317

18.2.3. Функция 3FH: чтение данных через описатель	317
18.2.4. Функция 40H: запись данных через описатель	318
Глава 19. Порты IBM PC	319
19.1. Порт 3F8H (BasePort+0): данные и делитель	320
19.2. Порт 3F9H (BasePort+1): разрешение прерываний и делитель	320
19.3. Порт 3FAH (BasePort+2): идентификация прерываний	321
19.4. Порт 3FBH (BasePort+3): управление линией	322
19.5. Порт 3FCH (BasePort+4): управление модемом	323
19.6. Порт 3FDH (BasePort+5): статус линии	324
19.7. Порт 3FEH (BasePort+6): статус модема	325
19.8. Порт 3FFH (BasePort+7): заглушка	325
19.9. Скорость передачи	325
19.10. Порты контроллера прерываний	326
Глава 20. Структуры Windows для работы с COM-портами	328
20.1. Структура настроек порта <i>COMMCONFIG</i>	328
20.2. Структура <i>COMMPROP</i>	330
20.3. Структура тайм-аутов <i>COMMTIMEOUTS</i>	336
20.4. Структура статуса порта <i>COMSTAT</i>	339
20.5. Структура <i>DCB</i>	340
Глава 21. Функции Windows для работы с COM-портами	347
21.1. Функции <i>CreateFile</i> и <i>CloseHandle</i> : открытие и закрытие порта	347
21.1.1. Дополнительные сведения	349
21.1.2. Возвращаемое значение	349
21.1.3. Пример вызова	349
21.2. Функция <i>ReadFile</i> : чтение данных из порта	350
21.2.1. Дополнительные сведения	351
21.2.2. Возвращаемое значение	351
21.2.3. Пример вызова	351
21.3. Функция <i>WriteFile</i> : передача данных	352
21.3.1. Дополнительные сведения	353
21.3.2. Возвращаемое значение	353
21.3.3. Пример вызова	353
21.4. Функция <i>ReadFileEx</i> : APC-чтение данных	354
21.4.1. Возвращаемое значение	356
21.4.2. Дополнительные сведения	356
21.4.3. Пример вызова	356
21.5. Функция <i>WriteFileEx</i> : APC-передача данных	357
21.5.1. Возвращаемое значение	358
21.5.2. Пример вызова	358

21.6. Функция <i>BuildCommDCB</i> : создание структуры <i>DCB</i> из строки.....	359
21.6.1. Дополнительные сведения.....	360
21.6.2. Возвращаемое значение	360
21.6.3. Пример вызова.....	360
21.7. Функция <i>BuildCommDCBAndTimeouts</i> : создание структуры <i>DCB</i> и тайм-аутов из строки.....	361
21.8. Функции <i>SetCommBreak</i> и <i>ClearCommBreak</i> : управление выводом данных	362
21.8.1. Возвращаемое значение	362
21.9. Функция <i>ClearCommError</i> : получение и сброс ошибок порта	363
21.9.1. Возвращаемое значение	364
21.10. Функция <i>CancelIo</i> : прерывание асинхронных операций.....	364
21.10.1. Возвращаемое значение	364
21.11. Функция <i>EscapeCommFunction</i> : управление портом	365
21.11.1. Возвращаемое значение	366
21.12. Функции <i>GetCommMask</i> и <i>SetCommMask</i> : маска вызова событий	366
21.12.1. Возвращаемое значение	367
21.12.2. Пример вызова.....	367
21.13. Функция <i>CreateEvent</i> : создание объекта ожидания.....	367
21.13.1. Возвращаемое значение	368
21.13.2. Пример вызова.....	368
21.14. Функция <i>WaitCommEvent</i> : ожидание события СОМ-порта	368
21.14.1. Возвращаемое значение	369
21.14.2. Дополнительные сведения	369
21.14.3. Пример вызова.....	369
21.15. Функции <i>GetCommConfig</i> и <i>SetCommConfig</i> : конфигурирование параметров порта.....	372
21.15.1. Возвращаемое значение	372
21.15.2. Пример вызова.....	373
21.16. Функция <i>CommConfigDialog</i> : диалог конфигурирования порта	374
21.16.1. Возвращаемое значение	375
21.16.2. Дополнительные сведения	375
21.16.3. Пример вызова.....	375
21.17. Функция <i>GetCommProperties</i> : прочитать свойства порта.....	375
21.17.1. Возвращаемое значение	376
21.17.2. Пример вызова.....	376
21.18. Функции <i>GetCommState</i> и <i>SetCommState</i> : состояние порта	376
21.18.1. Возвращаемое значение	377
21.18.2. Пример вызова.....	377
21.19. Функции <i>GetCommTimeouts</i> и <i>SetCommTimeouts</i> : тайм-ауты порта.....	377
21.19.1. Возвращаемое значение	378
21.19.2. Пример вызова.....	378

21.20. Функция <i>PurgeComm</i> : сброс буферов порта	378
21.20.1. Возвращаемое значение	379
21.20.2. Пример вызова	379
21.21. Функция <i>SetupComm</i> : конфигурирование размеров буферов	380
21.21.1. Возвращаемое значение	381
21.22. Функции <i>GetDefaultComm Config</i> и <i>SetDefaultComm Config</i> : настройки порта по умолчанию	381
21.22.1. Возвращаемое значение	382
21.23. Функция <i>TransmitComm Char</i> : передача специальных символов	382
21.23.1. Возвращаемое значение	382
21.24. Функция <i>GetCommModemStatus</i> : статус модема	383
21.24.1. Возвращаемое значение	383
21.24.2. Пример вызова	383
21.25. Функция <i>WaitForSingleObject</i> : ожидание сигнального состояния объекта	384
21.25.1. Возвращаемое значение	385
21.26. Функция <i>WaitForMultipleObjects</i> : ожидание сигнального состояния объектов	385
21.26.1. Возвращаемое значение	386
21.27. Функция <i>GetOverlappedResult</i> : результат асинхронной операции	387
21.27.1. Возвращаемое значение	388
21.28. Функция <i>DeviceIoControl</i> : прямое управление драйвером	388
21.28.1. Возвращаемое значение	390
21.29. Функция <i>EnumPorts</i> : перечисление портов	390
21.29.1. Дополнительные сведения	391
21.29.2. Возвращаемое значение	391
21.29.3. Пример вызова	392
21.30. Функция <i>QueryDosDevice</i> : получение имени устройства по его DOS-имени	393
21.30.1. Возвращаемое значение	393
21.30.2. Пример вызова	394
21.31. Функция <i>DefineDosDevice</i> : операции с DOS-именем устройства	395
21.31.1. Возвращаемое значение	395
21.31.2. Пример вызова	395
Глава 22. Функции <i>DeviceIoControl</i> для последовательных портов	397
22.1 Функции драйвера последовательного порта	397
22.2. Структуры драйвера последовательного порта	404
22.3. Соответствие функций <i>DeviceIoControl</i> и Windows API	406
Глава 23. АТ-команды и регистры модемов	409
23.1. Основные АТ-команды модемов	409
23.2. Основные регистры модемов	416

Часть IV. Приложения.....	419
Приложение 1. Микросхемы UART	421
Общее описание	421
Характерные особенности	421
Описание контактов	422
Типы микросхем UART.....	426
Приложение 2. Библиотека функций для Pascal	427
Приложение 3. Формат <i>SINGLE</i> чисел с плавающей точкой	429
Приложение 4. Описание функций Windows для использования в Visual Basic	430
Приложение 5. Использование ActiveX-компонента <i>MSComm32</i>.....	433
Приложение 6. Сервис BIOS — обслуживание INT14H	436
Приложение 7. Как работает GiveIO	442
Приложение 8. Реализация процедуры преобразования <i>SINGLE</i>-чисел в строку.....	450
Приложение 9. Индикаторы состояния модема.....	455
Приложение 10. Инструменты	456
Порт-монитор	456
Разработка драйверов Driver Wizard.....	457
Разработка драйверов NuMega Driver Studio	457
Отладчик NuMega SoftICE	458
Эмулятор компьютера Connectix Virtual PC.....	458
Терминал HyperTerminal	458
Приложение 11. Формат команды <i>Mode</i>.....	460
Приложение 12. Часто задаваемые вопросы (FAQ).....	462
FAQ: Общие вопросы	462
1. Что такое COM и RS и чем они отличаются?	462
2. Что такое порт?	462
3. Чем "протокол RS-232" отличается от "интерфейса RS-232"?.....	462
4. Что такое контрольная сумма и как ее считать?	463

5. Откуда взялось ограничение числа абонентов в протоколе RS-485?	463
6. Почему нужно подключать COM-устройства при выключенном питании?	463
7. Откуда берется питание для мыши?	463
8. Что означает режим <i>HANDSHAKING</i> ?	464
9. Что такое дуплексная и полудуплексная передача?	464
FAQ: Программирование в DOS	464
1. Что такое базовый адрес порта?	464
2. Написал программу обработки прерываний порта, а она периодически "виснет". Для других прерываний все нормально работает	464
3. При выводе на экран данных, полученных в обработчике прерывания, выводится "мусор"	465
4. Прерывание и основная программа используют один буфер. Как их синхронизировать в DOS?	465
5. Как бы перехватить вывод программы в COM-порт под DOS?	465
FAQ: Программирование в Windows	466
1. Почему в Delphi пропал оператор <i>Port</i> и как тогда обращаться к портам?	466
2. Можно ли использовать прямое обращение к портам в Windows NT/2000?	466
3. Программа асинхронно работает с портом. В Windows 98 все работало, а в Windows 2000 или "виснет", или не работает	466
4. Зачем надо использовать <i>WaitForSingleObject</i> , если уже есть вызов <i>GetOverlappedResult</i> ?	466
5. Как найти все доступные COM-порты?	466
6. Как бы перехватить вывод программы в COM-порт под Windows?	467
7. Как выключить Plug and Play для COM-порта при загрузке Windows?	467
FAQ: Связь, модемы, терминалы	467
1. Что такое FIFO и FOSSIL?	467
Приложение 13. Описание компакт-диска	469
Литература и Интернет-ресурсы	471
Предметный указатель	472

Глава 2



Протоколы

"Теперь позвольте пару фраз без протокола..."

В. Высоцкий, "Милицейский протокол"

В этой главе описываются типы протоколов обмена, методы организации передачи данных и методы повышения достоверности передачи.

2.1. Что такое протокол обмена

Чтобы принимающее устройство умело правильно интерпретировать полученные данные, необходима договоренность между приемником и передатчиком — *протокол обмена*. Протокол обмена состоит из нескольких частей:

- ☐ договоренность о параметрах связи (скорость, четность, число бит в байте и т. д.);
- ☐ договоренность о последовательности передачи данных и форматах представления чисел;
- ☐ договоренность об инициаторе обмена (если необходимо);
- ☐ процедура восстановления связи (если необходимо);
- ☐ дополнительные договоренности (например, передача прав главного хоста в сетях RS-485).

Параметры связи обязывают приемник и передатчик работать с одинаковыми аппаратными настройками. Обычно промышленные устройства используют строго определенные настройки обмена, некоторые — позволяют задавать скорость обмена (как программно, так и с помощью перемычек). Настройки четности и число бит в байте чаще всего фиксируются и не могут настраиваться со стороны аппаратуры.

Иногда, для согласования скорости работы, устройства выбирают фиксированную скорость начала обмена, устанавливают соединение и, обмениваясь данными на этой скорости, "договариваются" об использовании других скоростей и параметров обмена.

Для того чтобы принимающее устройство могло воспользоваться полученной информацией, ему необходимо знать, как ее интерпретировать. Для этого приемник и передатчик декларируют последовательность передачи данных. Часто именно эту часть договоренностей называют протоколом обмена, подразумевая наличие остальных частей.

При использовании приемником и передатчиком одной линии связи (как, например, в интерфейсе RS-485), необходима договоренность об инициаторе обмена. Это тем более необходимо в тех случаях, когда приемник один, а передатчиков много.

При разрыве связи одно из устройств может ждать ответа другого, а то, в свою очередь, ответа первого. Таким образом, без договоренности о действиях в случае разрыва связи устройства не смогут связаться друг с другом повторно. Часто выбирается очень простой способ — если в течение некоторого времени устройства не получают данных друг от друга, они переходят в режим "первого запуска". Однако возможны и более экзотические договоренности.

С точки зрения формата передаваемых данных, типы протоколов обмена можно поделить на две категории — строковые и бинарные. В первом случае информация передается обычными ASCII-символами (American Standard Code for Informational Interchange, американский стандартный код для обмена информацией), а во втором — бинарным потоком данных. Далее мы рассмотрим оба эти типа, а их реализацией займемся в *гл. 14*.

2.2. ASCII-протокол передачи данных

В таком протоколе управляющие символы и данные передаются с помощью ASCII-символов.

Пример пакета данных в таком формате приведен в табл. 2.1. Пакет данных начинается со стартовой последовательности — одного или нескольких символов, обозначающих начало данных. Символы стартовой последовательности не должны появляться внутри блока данных. Получив байты начала посылки, все абоненты настраиваются на получение полного пакета данных. Затем, получив и сравнив номер абонента со своим номером, абонент либо продолжает обработку пакета, либо снова переходит в режим ожидания начала посылки. Таким образом, пакет обрабатывает только один абонент сети. После получения байта конца посылки абонент проверяет контрольную сумму и обрабатывает полученные данные. Конец посылки также может представляться несколькими символами. Разумеется, строка конца посылки не должны встречаться внутри самих данных.

Естественно, если связь осуществляется между двумя абонентами, поле "номер абонента" может отсутствовать.

Таблица 2.1. Структура ASCII-протокола

Байт	Описание	Примеры	
Начало посылки	Идентификатор начала посылки. Любой символ (или несколько символов), не встречающийся в символах данных	\$	!!!
Номер абонента	Две десятичные цифры номера абонента (дополненные при необходимости нулем). В некоторых случаях могут оговариваться специальные номера абонентов, например, "***", означающая посылку всем абонентам сети	01	15
Данные	Последовательность цифр данных. Числа с плавающей точкой разделяются знаком ".". Части данных могут разделяться пробелом	011 03.1	155 17.5
Контроль	Контрольная сумма данных. Обычно один или два байта суммы всех байт данных с переполнением. Байты суммы также передаются в десятичном виде (три десятичные цифры с выравниванием нулями слева)	002	104
Конец посылки	Идентификатор конца посылки. Для ASCII-протоколов обычно символ LR (перевод строки)	\$0D	***

Достоинство такого протокола в удобстве отладки: прием и передачу команд и данных можно осуществлять с помощью обычной программы-терминала (например, Telnet).

Основной недостаток — большой объем данных, необходимых для передачи. Например, для передачи одного байта шестнадцатеричной строкой потребуется два байта (число 230 будет передано последовательностью \$E, \$6). А при передаче обычными десятичными цифрами — три байта (число 230 будет передано тремя цифрами "2,3,0"). Передача же чисел с плавающей точкой потребует количество байт, пропорциональное точности передаваемого числа (плюс десятичный разделитель). Например, число 221,73 будет передаваться с помощью последовательности символов "2,2,1,.,7,3", т. е. с помощью 6 байт.

Второй недостаток — необходимость преобразования двоичных данных в ASCII-строки и обратно. Если на языке высокого уровня (C, Pascal) это не проблема, то, например, для 8-битового контроллера может потребоваться значительный объем кода (пример кода, реализующего такое преобразование приводится в *прил. 8*).

Хорошим примером ASCII-протокола является MODBUS-ASCII (*см. разд. 4.1.1*), а программной реализацией своего ASCII-протокола мы займемся в *гл. 14*.

2.3. Бинарный протокол передачи данных

Бинарная передача значительно более компактна. Например, для передачи числа с плавающей точкой типа `SINGLE` (см. *прил. 3*) требуется 4 байта независимо от числа знаков после десятичной точки.

Так же как в `ASCII`-протоколе, выбирается некоторый символ для обозначения начала посылки. Завершение ожидания данных производится либо по получению необходимого числа байт, либо по тайм-ауту: принимающее устройство отсчитывает время с получения последнего байта до приема следующего. Если эта пауза превышает некоторый интервал (обычно 1,5—2 байта), посылка считается потерянной и приемный буфер сбрасывается. После получения необходимого числа байт (или символа конца посылки) проверяется правильность полученных данных и производится их обработка.

Примером бинарного протокола может служить протокол `MODBUS-RTU` (см. *разд. 4.1.2*), а реализацией своего бинарного протокола мы займемся в *гл. 14*.

2.4. Предотвращение потери данных

Причины потери данных при передаче или приеме можно разделить на две категории: аппаратные и программные.

К *аппаратным причинам потери данных* относятся помехи на линии, некачественный проводник, несоответствие длины провода и скорости передачи и т. д. Борьбаться с такими причинами, конечно, надо, но не на программном уровне. Правила прокладки сетей и расчет максимальной длины проводника можно найти в соответствующей литературе, например, см. в [2] и [9]. От программной составляющей в этом случае требуется только определить, что данные были переданы с ошибкой. Способы обнаружения ошибок мы рассмотрим дальше.

Потеря данных может возникать и при неправильной организации программы. Последовательная передача подразумевает, что все биты, посланные с одной стороны, будут приняты с другой. Более того, во время передачи и приема байта нельзя допускать никаких задержек: если байт не считан вовремя, он теряется и его место занимает следующий переданный байт. Аналогично, если хотя бы часть пакета данных будет потеряна, это, чаще всего, означает потерю всего пакета данных.

2.4.1. Прерывания и потоки

Основная причина потери данных — программа занята обработкой других данных и не опрашивает порт. Возможно, в этот момент идет обработка предыдущего полученного пакета данных или программа занята диалогом с

пользователем. Все эти причины сводятся к одной — неправильной организации программы.

Рассмотрим, например, алгоритм, приведенный в листинге 2.1.

Листинг 2.1. Неправильная организация опроса порта

```
Begin
  While (True) do begin
    { Опрос клавиатуры }
    If KeyPressed then
      Case ReadKey of
        #27: Break; {выход}
        'q': Begin
          Write('Введите K'); ReadLn(K);
        End;
      End;
    End;

    { Опрос порта }
    DoReadPort;

    { Есть новые данные! }
    If DataAvailable then begin
      WriteLn('Значение температуры:', Code_ADC * K);
      DataAvailable:= False;
    End;

  End;
End.
```

При всей заманчивой простоте такого кода так делать не надо! Все будет хорошо, пока пользователь не нажмет клавишу <q>, вызывающую диалог ввода нового коэффициента. А вот во время диалога порт опрашиваться не будет. В программе для Windows можно сделать аналогичную ошибку, вызвав метод DoModal формы диалога параметров.

Конечно, в Windows можно воспользоваться компонентом TTimer и делать опрос порта в обработчике OnTimer с некоторой фиксированной частотой. Однако и этот способ далек от совершенства. Дело в том, что, во-первых, при большой скорости передачи данных буфер приема драйвера порта может переполниться, и часть данных будет потеряна. А, во-вторых, таймер

работает с помощью посылки сообщения WM_TIMER окну программы. Это сообщение имеет низший приоритет, и вероятность того, что считывание порта будет произведено вовремя, достаточно мала. В Windows 95/98 достаточно, щелкнув на заголовке окна, удерживать кнопку "мыши" в нажатом состоянии — и сообщения WM_TIMER не будут обрабатываться. Таким образом, если пользователь будет перетаскивать окно на другое место, порт опрашиваться не будет. И еще один недостаток: несмотря на то, что редактор свойств позволяет установить свойство Interval (частоту вызова обработчика таймера) в любое значение, большее нуля, реально работающим минимальным значением будет 70—100 мс.

Раскритиковав все способы работы, стоит предложить какой-нибудь правильный. Для DOS таким способом является обработка прерываний, рассматриваемая в гл. 8. А для Windows правильным способом обработки порта является создание отдельного потока для чтения данных, как будет показано в гл. 11 и 12.

2.4.2. Буферизация

При большой скорости передачи данных, очевидно, может возникнуть ситуация, когда время на обработку полученного пакета данных (например, рисование графика, анализ данных и сохранение их в файл) больше, чем время между получением пакетов.

В таких случаях необходимо некоторое промежуточное хранилище полученных данных — буфер.

Буферизация позволяет основному модулю не заботиться о скорости обработки данных. Конечно, размер буфера тоже ограничен, но обычно его выбирают достаточно большим.

На уровне приема одного байта существуют свои буферы приема:

- ☐ в DOS можно включить специальный режим буферизации FIFO;
- ☐ Windows имеет свои буферы, размер которых можно изменять вызовом функции SetupComm.

Однако чаще всего программе важнее иметь свой буфер, хранящий полные пакеты данных, а системные буферы использовать как дополнительный метод повышения надежности (см. гл. 14).

Единственная сложность буферизации — необходимость синхронизации процедуры выборки данных из буфера и процедуры добавления в буфер. Синхронизация необходима, т. к. в DOS добавление в буфер будет производиться из прерывания, а в Windows — из другого потока. В обоих случаях вероятность того, что основная программа будет прервана на обновление буфера в середине процедуры выборки данных, достаточно высока. Очевидно, в таком случае возникнет ошибка.

Синхронизация в DOS

В DOS синхронизация возможна только одним способом — запрещением прерываний на время выборки данных из порта, как показано в листинге 2.2.

Листинг 2.2. Синхронизация в DOS

```
{Обработчик прерывания}
Procedure OnCOMInterrupt(B : Byte);
Begin
  { Добавить байт в буфер }
  AddByteToBuffer(B);
End;

Var
  B : Byte;
Begin
  {Основной цикл программы}
  While (True) do begin
    {Запретить прерывание}
    DisableIRQ;
    {Забрать байт, если он есть}
    If GetSizeBuffer > 0 then B:= GetByteFromBuffer;
    {Разрешить прерывания}
    EnableIRQ;

    ... обработка нового байта B ...

  End;
End.
```

Обратите внимание, что обработка байта производится вне блока [DisableIRQ, EnableIRQ]. Таким образом, запрещение прерываний происходит на очень маленький промежуток времени выборки данных из буфера.

Синхронизация в Windows

В Windows существуют свои методы синхронизации. Для синхронизации двух потоков нужно создать специальный объект синхронизации, например, TCriticalSection. Скелет такой программы приводится в листинге 2.3.

Листинг 2.3. Пример синхронизации в Windows

```
Uses SyncObjs; {нужно для использование объектов синхронизации}
Var
  CS : TCriticalSection;

{Создание объекта синхронизации}
procedure TForm1.FormCreate(Sender: TObject);
begin
  CS:= TCriticalSection.Create;
end;

{Уничтожение объекта синхронизации}
procedure TForm1.FormDestroy(Sender: TObject);
begin
  CS.Free;
end;

{Добавление байта в буфер}
procedure TThread1.DoAddToBuffer(:);
begin
  CS. Acquire;
  Try
    :добавить в буфер:
  Finally
    CS. Release;
  End;
end;

{Выборка байта из буфера}
procedure TThread2.DoGetByteFromBuffer(:);
begin
  CS. Acquire;
  Try
    :забрать из буфера:
  Finally
    CS. Release;
  End;
end;
```

Подробнее про объекты синхронизации можно прочитать в [11].