

ПАВЕЛ АГУРОВ



ИНТЕРФЕЙС USB

**ПРАКТИКА ИСПОЛЬЗОВАНИЯ
И ПРОГРАММИРОВАНИЯ**

USB 1.1/2.0, HID-УСТРОЙСТВА

НАПИСАНИЕ USB-ДРАЙВЕРОВ

РАБОТА В DOS, WINDOWS 98/ME/NT/2000/XP

ПРИМЕРЫ НА ЯЗЫКЕ PASCAL

ПРИМЕРЫ РЕАЛИЗАЦИИ ДЛЯ ATMEL AT89C5131

РАБОТА С МИКРОСХЕМАМИ FTDI



+CD 

**АППАРАТНЫЕ
СРЕДСТВА**

Павел Агуров

ИНТЕРФЕЙС USB

**ПРАКТИКА ИСПОЛЬЗОВАНИЯ
И ПРОГРАММИРОВАНИЯ**

Санкт-Петербург

«БХВ-Петербург»

2004

УДК 681.3.06
ББК 32.973
А27

Агуров П. В.

А27 Интерфейсы USB. Практика использования и программирования. — СПб.: БХВ-Петербург, 2004. — 576 с.: ил.

ISBN 5-94157-202-6

Изложены базовые сведения по интерфейсу USB для ПК: примеры USB-устройств и советы по их выбору, правила установки и конфигурирования устройств, методы решения возникающих проблем. Описаны кабели, разъемы, принципы питания устройств и другое аппаратное обеспечение. Приведено внутреннее устройство USB и его физическая реализация, обсуждены общие вопросы написания драйверов для операционной системы Microsoft Windows 2000/XP с примерами на языке Borland Pascal и в среде Delphi. Рассмотрен процесс создания USB-устройства: от выбора микросхем и схемотехники до написания программы микроконтроллера и WDM-драйвера. В книге содержится большое количество практических советов и примеров программ. Для удобства читателей все исходные коды приводятся на прилагаемом компакт-диске.

Для пользователей ПК, разработчиков аппаратуры и программистов

УДК 681.3.06
ББК 32.973

Группа подготовки издания:

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

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.09.04.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 46,44.

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 ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-202-6

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

Содержание

Введение.....	1
Для кого эта книга.....	2
Что вы найдете в книге	2
Программные требования	3
Аппаратные требования.....	4
О программном коде.....	4
Краткое описание глав	4
Обозначения	6
Благодарности.....	7
 Часть I. Введение в USB	9
 Глава 1. Что такое USB.....	11
1.1. История USB	11
1.2. Сравнение USB с другими интерфейсами	14
1.3. Основные понятия USB	16
1.3.1. Общая архитектура шины.....	16
1.3.2. Физическая и логическая архитектура шины	16
1.3.3. Составляющие USB.....	18
1.3.4. Свойства USB-устройств	18
1.3.5. Свойства хабов.....	19
1.3.6. Свойства хоста	20
1.4. Примеры USB-устройств	20
1.4.1. Мышь и клавиатура.....	21
1.4.2. Мониторы	21
1.4.3. Переходники USB-to-COM и USB-to-LPT	22
1.4.4. Сканеры	23
1.4.5. Модемы.....	23
1.4.6. Звуковые колонки	24
1.4.7. Флеш-диски	25
1.4.8. Хабы	28
1.4.9. Измерительная техника	28
1.4.10. Экзотические устройства	29
1.5. Сетевое соединение через USB	30
1.5.1. Конвертер USB-Ethernet.....	31
1.5.2. Прямое соединение через USB-порт	31
1.6. Передача данных	31
1.6.1. Принципы передачи данных	32
1.6.2. Механизм прерываний.....	32
1.6.3. Интерфейсы хост-адаптера.....	32
1.6.4. Возможность прямого доступа к памяти	34
1.6.5. Режимы передачи данных.....	34

1.7. Установка и конфигурирование USB-устройств	35
1.7.1. Настройки BIOS для USB	38
1.7.2. Устранение проблем.....	41
1.8. Ограничения USB	45
1.9. Если вы покупаете компьютер	46
1.9.1. HS и USB 2.0 — не одно и то же!.....	46
1.9.2. Системная плата	47
1.9.3. Корпус	48
1.9.4. USB для "старых" моделей компьютеров.....	48
1.10. Интернет-ресурсы к этой главе	49
Глава 2. Аппаратное обеспечение USB.....	51
2.1. Кабели и разъемы.....	51
2.1.1. Типы кабелей	52
2.1.2. Длина кабеля.....	53
2.1.3. Разъемы.....	53
2.2. Физический интерфейс	55
2.2.1. Кодирование данных.....	57
2.2.2. Идентификация устройств	58
2.3. Питание.....	59
2.3.1. Типы питания USB-устройств	59
2.3.2. Управление энергопотреблением.....	60
2.3.3. Вход в режим низкого энергопотребления.....	61
2.4. Интернет-ресурсы к этой главе	61
Часть II. Внутренняя организация USB	63
Глава 3. Внутренняя организация шины	65
3.1. Логические уровни обмена данными.....	65
3.1.1. Уровень клиентского ПО	66
3.1.2. Уровень системного драйвера USB.....	67
3.1.3. Уровень хост-контроллера интерфейса.....	68
3.1.4. Уровень шины периферийного устройства	68
3.1.5. Уровень логического USB-устройства	69
3.1.6. Функциональный уровень USB-устройства	69
3.2. Передача данных по уровням	69
3.3. Типы передач данных	71
3.4. Синхронизация при изохронной передаче.....	73
3.5. Кадры.....	77
3.6. Конечные точки	78
3.7. Каналы.....	79
3.8. Пакеты.....	81
3.8.1. Формат пакетов-маркеров IN, OUT, SETUP и PING	83
3.8.2. Формат пакета SOF	83
3.8.3. Формат пакета данных.....	84
3.8.4. Формат пакета подтверждения	84
3.8.5. Формат пакета SPLIT.....	84

3.9. Контрольная сумма.....	85
3.9.1. Алгоритм вычисления CRC.....	86
3.9.2. Программное вычисление CRC	87
3.10. Транзакции.....	90
3.10.1. Типы транзакций	91
3.10.2. Подтверждение транзакций и управление потоком	92
3.10.3. Протоколы транзакций.....	93
Глава 4. Внутренняя организация устройства	96
4.1. Запросы к USB-устройствам.....	96
4.1.1. Конфигурационный пакет.....	96
4.1.2. Стандартные запросы к устройствам	99
4.1.3. Дескрипторы устройства.....	105
Глава 5. Внутренняя организация хоста и хабов	123
5.1. Хабы	123
5.1.1. Взаимодействие хост-контроллера с хабом	126
5.1.2. Дескриптор хаба	127
5.1.3. Запросы хабов	129
5.1.4. Запрос <i>CLEAR_HUB_FEATURE</i>	130
5.1.5. Запрос <i>CLEAR_PORT_FEATURE</i>	130
5.1.6. Запрос <i>GET_BUS_STATE</i>	131
5.1.7. Запрос <i>GET_HUB_DESCRIPTOR</i>	131
5.1.8. Запрос <i>GET_HUB_STATUS</i>	131
5.1.9. Запрос <i>GET_PORT_STATUS</i>	132
5.1.10. Запрос <i>SET_HUB_DESCRIPTOR</i>	134
5.1.11. Запрос <i>SET_HUB_FEATURE</i>	134
5.1.12. Запрос <i>SET_PORT_FEATURE</i>	134
5.2. Совместная работа устройств с разными скоростями	135
Глава 6. USB без ПК.....	137
6.1. Разъемы OTG.....	138
6.2. Типы OTG-устройств.....	138
6.3. Дескриптор OTG-устройства	139
6.4. Интернет-ресурсы к этой главе	140
Часть III. ПРАКТИКА ПРОГРАММИРОВАНИЯ.....	141
Глава 7. Поддержка USB в Windows.....	143
7.1. Модель WDM	144
7.2. Взаимодействие с USB-драйвером	146
Глава 8. HID-устройства.....	149
8.1. Свойства HID-устройства	149
8.2. Порядок обмена данными с HID-устройством	151
8.3. Установка HID-устройства.....	152

8.4. Идентификация HID-устройства	152
8.4.1. Идентификация загрузочных устройств	153
8.4.2. Дескриптор конфигурации HID-устройства	153
8.4.3. HID-дескриптор	154
8.4.4. Дескриптор репорта	156
8.5. Структура дескриптора репорта	156
8.5.1. Структура элементов репорта	156
8.5.2. Типы элементов репорта	157
8.5.3. Примеры дескрипторов	165
8.6. Запросы к HID-устройству	168
8.6.1. Запрос <i>GET_REPORT</i>	169
8.6.2. Запрос <i>SET_REPORT</i>	169
8.6.3. Запрос <i>GET_IDLE</i>	170
8.6.4. Запрос <i>SET_IDLE</i>	170
8.6.5. Запрос <i>GET_PROTOCOL</i>	171
8.6.6. Запрос <i>SET_PROTOCOL</i>	171
8.7. Инструментальные средства	171
8.8. Взаимодействие с HID-драйвером	172
Глава 9. Введение в WDM	181
9.1. Драйверные слои	183
9.2. Символьные имена устройств	184
9.3. Основные процедуры драйвера WDM	189
9.3.1. Процедура <i>DriverEntry</i>	190
9.3.2. Процедура <i>AddDevice</i>	192
9.3.3. Процедура <i>Unload</i>	194
9.3.4. Рабочие процедуры драйвера	196
9.3.5. Обслуживание запросов <i>IOCTL</i>	203
9.4. Загрузка драйвера и обращение к процедурам драйвера	209
9.4.1. Процедура работы с драйвером	209
9.4.2. Регистрация драйвера	210
9.4.3. Обращение к рабочим процедурам	217
9.4.4. Хранение драйвера внутри исполняемого файла	218
9.5. Инструменты создания драйверов	220
9.5.1. NuMega Driver Studio	220
9.5.2. Jungo WinDriver	220
9.5.3. Jungo KernelDriver	220
Глава 10. Спецификация PnP для USB	221
10.1. Общие сведения о системе Plug and Play	221
10.1.1. Задачи и функции Plug and Play	221
10.1.2. Запуск процедуры PnP	222
10.1.3. Программные компоненты PnP	224
10.2. Plug and Play для USB	225
10.2.1. Конфигурирование устройств USB	226
10.2.2. Нумерация устройств USB	226
10.2.3. PnP-идентификаторы устройств USB	228

10.3. Получение списка USB-устройств	229
10.4. INF-файл.....	234
10.4.1. Структура INF-файла.....	234
10.4.2. Секция <i>Version</i>	235
10.4.3. Секция <i>Manufacturer</i>	237
10.4.4. Секция <i>DestinationDirs</i>	239
10.4.5. Секция описания модели	241
10.4.6. Секция <i>xxx.AddReg</i> и <i>xxx.DelReg</i>	242
10.4.7. Секция <i>xxx.LogConfig</i>	244
10.4.8. Секция <i>xxx.CopyFiles</i>	244
10.4.9. Секция <i>Strings</i>	245
10.4.10. Связи секций.....	246
10.4.11. Создание и тестирование INF-файлов.....	247
10.4.12. Установка устройств с помощью INF-файла	248
10.5. Ветки реестра для USB.....	249
Глава 11. Функции BIOS	251
11.1. Сервис BIOS 1AH.....	251
11.1.1. Функция B101H — определение наличия PCI BIOS.....	252
11.1.2. Функция B102H — поиск PCI-устройства по идентификаторам устройства и производителя.....	253
11.1.3. Функция B103H — поиск PCI-устройства по коду класса.....	254
11.1.4. Функция B108H — чтение регистра конфигурации (Byte).....	255
11.1.5. Функция B109H — чтение регистра конфигурации (Word)	256
11.1.6. Функция B10AH — чтение регистра конфигурации (DWord).....	256
11.1.7. Функция B10BH — запись регистра конфигурации (Byte).....	257
11.1.8. Функция B10CH — запись регистра конфигурации (Word).....	257
11.1.9. Функция B10DH — запись регистра конфигурации (DWord)	258
11.2. Пример использования.....	259
Часть IV. Создание USB-устройств	283
Глава 12. USB-периферия	285
12.1. Микросхемы Atmel.....	286
12.1.1. Микроконтроллеры с архитектурой MSC-51	286
12.1.2. Контроллеры хабов.....	289
12.1.3. Микропроцессоры-хабы с ядром AVR.....	289
12.1.4. Другие микросхемы Atmel	290
12.2. Микросхемы Cygnal	291
12.2.1. Микропроцессоры C8051F320 и C8051F321.....	291
12.2.2. Другие микросхемы Cygnal.....	293
12.3. Микросхемы FTDI	296
12.3.1. Микросхемы FT232AM и FT232BM.....	297
12.3.2. Микросхемы FT245AM и FT245BM.....	298
12.3.3. Микросхема FT2232BM.....	299
12.3.4. Микросхема FT8U100AX.....	300

12.3.5. Отладочные комплекты и модули.....	301
12.3.6. Драйверы	302
12.3.7. Дополнительные утилиты	303
12.3.8. Другие модули.....	304
12.4. Микросхемы Intel.....	304
12.5. Микросхемы Microchip.....	308
12.6. Микросхемы Motorola	308
12.7. Микросхемы Philips.....	309
12.7.1. Микросхемы USB.....	310
12.7.2. Хабы	311
12.7.3. Другие микросхемы Philips.....	313
12.8. Микросхемы Texas Instruments	314
12.9. Микросхемы Trans Dimension.....	317
12.10. Микросхемы защиты питания	318
12.11. Интернет-ресурсы к этой главе	319
Глава 13. HID-устройство на основе Atmel AT89C5131	322
13.1. Структурная схема AT89C5131.....	322
13.2. USB-регистры AT89C5131	324
13.2.1. Регистр <i>USBCON</i>	324
13.2.2. Регистр <i>USBADDR</i>	326
13.2.3. Регистр <i>USBINT</i>	327
13.2.4. Регистр <i>USBIEN</i>	328
13.2.5. Регистр <i>UEPNUM</i>	329
13.2.6. Регистр <i>UEPCONX</i>	330
13.2.7. Регистр <i>UEPSTAX</i>	331
13.2.8. Регистр <i>UEPRST</i>	334
13.2.9. Регистр <i>UEPINT</i>	335
13.2.10. Регистр <i>UEPIEN</i>	336
13.2.11. Регистр <i>UEPDATX</i>	337
13.2.12. Регистр <i>UBYCTLX</i>	337
13.2.13. Регистр <i>UFNUML</i>	338
13.2.14. Регистр <i>UFNUMH</i>	338
13.3. Схемотехника AT89C5131.....	338
13.4. Инструменты программирования.....	339
13.4.1. Компилятор.....	341
13.4.2. Программатор	342
13.5. Программа для микропроцессора	349
13.5.1. Первая версия программы для AT89C5131	349
13.5.2. Добавляем строковые дескрипторы.....	369
13.5.3. Добавление конечных точек.....	374
13.5.4. Создание HID-устройства	377
13.5.5. Обмен данными с HID-устройством.....	381
13.6. Чтение репортов в Windows	388
13.7. Дополнительные функции Windows XP.....	396
13.8. Устройство с несколькими репортами.....	397

Глава 14. Создание USB-устройства на основе ATMEL AT89C5131.....	402
14.1. Не-HID-устройство.....	402
14.2. Создание драйвера с помощью Driver Studio.....	405
14.2.1. Несколько слов о библиотеке Driver Studio.....	407
14.2.2. Другие классы Driver Studio.....	411
14.2.3. Создание шаблона драйвера с помощью Driver Studio.....	412
14.2.4. Доработка шаблона драйвера.....	422
14.2.5. Базовые методы класса устройства.....	423
14.2.6. Реализация чтения данных.....	426
14.2.7. Установка драйвера.....	428
14.2.8. Программа чтения данных.....	429
14.2.9. Чтение данных с конечных точек других типов.....	438
14.2.10. "Чистый" USB-драйвер.....	439
Глава 15. Использование микросхем FTDI.....	457
15.1. Функциональная схема FT232BM.....	457
15.2. Схемотехника FT232BM.....	460
15.3. Функции D2XX.....	460
15.4. Переход от COM к USB.....	465
15.4.1. Описание схемы преобразователя.....	465
15.4.2. Установка скорости обмена.....	467
Часть V. СПРАВОЧНИК.....	469
Глава 16. Базовые функции Windows.....	471
16.1. Функции <i>CreateFile</i> и <i>CloseHandle</i> : открытие и закрытие объекта.....	471
16.1.1. Дополнительные сведения.....	472
16.1.2. Возвращаемое значение.....	472
16.1.3. Пример вызова.....	472
16.2. Функция <i>ReadFile</i> : чтение данных.....	473
16.2.1. Дополнительные сведения.....	474
16.2.2. Возвращаемое значение.....	474
16.2.3. Пример вызова.....	474
16.3. Функция <i>WriteFile</i> : передача данных.....	475
16.3.1. Дополнительные сведения.....	476
16.3.2. Возвращаемое значение.....	476
16.3.3. Пример вызова.....	476
16.4. Функция <i>ReadFileEx</i> : APC-чтение данных.....	477
16.4.1. Возвращаемое значение.....	479
16.4.2. Дополнительные сведения.....	479
16.4.3. Пример вызова.....	479
16.5. Функция <i>WriteFileEx</i> : APC-передача данных.....	480
16.5.1. Возвращаемое значение.....	481
16.5.2. Пример вызова.....	481

16.6. Функция <i>WaitForSingleObject</i> : ожидание сигнального состояния объекта	482
16.6.1. Возвращаемое значение	482
16.7. Функция <i>WaitForMultipleObjects</i> : ожидание сигнального состояния объектов	483
16.7.1. Возвращаемое значение	484
16.8. Функция <i>GetOverlappedResult</i> : результат асинхронной операции	484
16.8.1. Возвращаемое значение	485
16.9. Функция <i>DeviceIoControl</i> : прямое управление драйвером	485
16.9.1. Возвращаемое значение	487
16.10. Функция <i>QueryDosDevice</i> : получение имени устройства по его DOS-имени	487
16.10.1. Возвращаемое значение	488
16.10.2. Пример вызова	488
16.11. Функция <i>DefineDosDevice</i> : операции с DOS-именем устройства	489
16.11.1. Возвращаемое значение	490
16.11.2. Пример вызова	490
Глава 17. Функции <i>HID API</i>	492
17.1. Функция <i>HidD_Hello</i> : проверка библиотеки	492
17.2. Функция <i>HidD_GetHidGuid</i> : получение GUID	492
17.3. Функция <i>HidD_GetPreparedData</i> : создание описателя устройства	493
17.4. Функция <i>HidD_FreePreparedData</i> : освобождение описателя устройства	493
17.5. Функция <i>HidD_GetFeature</i> : получение FEATURE-репорта	494
17.6. Функция <i>HidD_SetFeature</i> : передача FEATURE-репорта	494
17.7. Функция <i>HidD_GetNumInputBuffers</i> : получение числа буферов	495
17.8. Функция <i>HidD_SetNumInputBuffers</i> : установка числа буферов	495
17.9. Функция <i>HidD_GetAttributes</i> : получение атрибутов устройства	495
17.10. Функция <i>HidD_GetManufacturerString</i> : получение строки производителя	496
17.11. Функция <i>HidD_GetProductString</i> : получение строки продукта	497
17.12. Функция <i>HidD_GetSerialNumberString</i> : получение строки серийного номера	497
17.13. Функция <i>HidD_GetIndexedString</i> : получение строки по индексу	498
17.14. Функция <i>HidD_GetInputReport</i> : получение INPUT-репорта	498
17.15. Функция <i>HidD_SetOutputReport</i> : передача OUTPUT-репорта	499
17.16. Функция <i>HidP_GetCaps</i> : получение свойств устройства	499
17.17. Функция <i>HidP_MaxDataListLength</i> : получение размеров репортов	500
Глава 18. Хост-контроллер UCN	502
18.1. Регистры управления хост-контроллером	502
18.1.1. Регистр команды USB (<i>USBCMD</i>)	504
18.1.2. Регистр состояния USB (<i>USBSTS</i>)	506
18.1.3. Регистр управления прерываниями (<i>USBINTR</i>)	506
18.1.4. Регистр номера кадра (<i>FRNUM</i>)	507

18.1.5. Регистр базового адреса кадра (<i>FLBASEADD</i>)	508
18.1.6. Регистр модификатора начала кадра (<i>SOFMOD</i>)	508
18.1.7. Регистр состояния и управления порта (<i>PORTSC</i>)	509
18.2. Структуры данных хост-контроллера UCH	510
18.2.1. Список кадров.....	510
18.2.2. Дескриптор передачи	511
18.2.3. Заголовок очереди	514
18.3. Обработка списка дескрипторов UCH	516
Глава 19. Инструменты	518
19.1. Средства Microsoft Visual Studio	518
19.1.1. Depends	518
19.1.2. Error Lookup	518
19.1.3. GuidGen	518
19.2. Средства Microsoft DDK.....	520
19.2.1. DeviceTree	520
19.2.2. DevCon	521
19.2.3. ChkInf и GenInf.....	526
19.3. Средства CompuWare Corporation.....	527
19.3.1. Monitor	527
19.3.2. SymLink.....	527
19.3.3. EzDriverInstaller	527
19.3.4. WdmSniff.....	527
19.4. Средства SysInternals	528
19.4.1. WinObj	528
19.5. Средства USB Forum.....	531
19.5.1. HID Descriptor Tool	531
19.6. Средства HDD Software	533
19.7. Средства Sourceforge.....	533
ПРИЛОЖЕНИЯ.....	535
Приложение 1. Дополнительные функции.....	537
Приложение 2. Таблица идентификаторов языков (LangID).....	539
Приложение 3. Таблица кодов производителей (Vendor ID, Device ID).....	543
Приложение 4. Описание компакт-диска	546
Литература	548
Предметный указатель	549

Введение

Последовательные интерфейсы интересны тем, что позволяют объединить множество устройств, используя всего одну (или две) пары проводов. До 1996 года последовательные интерфейсы персонального компьютера были представлены коммуникационным портом, работающим согласно спецификации RS-232. Хотя RS-232 сохраняет все преимущества последовательной связи, она имеет и ряд недостатков. Самыми существенными из них являются плохая помехозащищенность и отсутствие гальванической развязки. Первое мешает использованию высоких скоростей обмена, а второе — "горячему" подключению устройств. Кроме того, стандарт RS-232 подразумевает подключение только одного устройства к одному последовательному порту.

Еще один недостаток внешних интерфейсов персонального компьютера — строгая предопределенность их использования. Так, COM-порт используется для подключения "мыши" или модема, LPT-порт — для подключения принтера (ну, еще сканера или плоттера), порт клавиатуры предназначен для подключения клавиатуры, и т. д. Кроме необходимости иметь в персональном компьютере множество различных, но, чаще всего, неиспользуемых, разъемов, такое многообразие несет и другие проблемы: для каждого интерфейса необходимо выделять аппаратное прерывание (IRQ), "пустые" разъемы занимают место, что особенно актуально для ноутбуков.

В 1996 году была опубликована версия 1.0 нового интерфейса, названного USB (Universal Serial Bus, универсальная последовательная шина), а осенью 1998 — спецификация 1.1, исправляющая проблемы, обнаруженные в первой редакции. В 2000 году была опубликована версия 2.0, в которой предусматривалось 40-кратное повышение пропускной способности шины.

Шина USB ориентирована на устройства, подключаемые к PC. Изохронные передачи USB позволяют передавать огромные потоки данных, такие как аудиосигналы, а шина USB 2.0 позволяет передавать и видеосигналы. Спецификация USB подразумевает прозрачное подключение устройств к шине и позволяет иметь несколько устройств на одном порту.

Для кого эта книга

Эта книга для вас, если:

- ☐ протирая пыль с компьютера, вы обнаружили два непонятных разъема и хотите узнать, что это такое;
- ☐ вам надо объяснить в бухгалтерии, что "все то же самое, но без USB" дешевле не будет;
- ☐ вам надо подключить к компьютеру два принтера, сканер и цифровой фотоаппарат одновременно;
- ☐ подключив к первому порту USB — мышь, а ко второму USB — клавиатуру, вы хотите понять, куда же подключать обещанные в USB-спецификации 127 устройств;
- ☐ ваш шеф утверждает, что изготовленный вами прибор не работает, т. к. его нельзя подключить к его новому ноутбуку, в котором нет COM-порта;
- ☐ вам обидно, что Windows находит новые устройства, но не говорит об этом вашей программе;
- ☐ вам хочется узнать, что нужно Windows, чтобы ваше устройство было названо по имени;
- ☐ скорость COM-порта вас не удовлетворяет, поэтому хочется использовать USB, но времени на переделку нет.

Другими словами, мы адресуем эту книгу тем, кто хочет использовать в своей работе современный протокол USB. Книга будет интересна тем, кто занимается или собирается заниматься программированием микроконтроллеров. Книга будет полезна разработчикам Windows-драйверов, а также тем, кто хочет повысить свой профессиональный уровень.

Что вы найдете в книге

В этой книге вы сможете найти ответы на некоторые вопросы, касающиеся разработки и программирования USB-устройств.

- ☐ Что такое USB-интерфейс и зачем он нужен:
 - вы хотите узнать, что такое USB? Эта книга ориентируется на практическое использование USB-интерфейса и содержит все сведения, необходимые для разработки своего проекта.
- ☐ В каких случаях нужен USB-интерфейс, и какие преимущества он дает:
 - вы используете COM-интерфейс и считаете, что этого достаточно? Прочитайте эту книгу, — возможно, вы измените свое мнение.

- ❑ Как USB-устройства взаимодействуют с компьютером:
 - вы хотите понять, как работает USB-шина? что нужно сделать, чтобы ваше устройство было опознано системой? Ответы на эти вопросы содержат главы, посвященные архитектуре USB-шины и стандарту Plug and Play.
- ❑ Как разработать приложение, взаимодействующее с USB-контроллером:
 - вы решили использовать USB в своем проекте, вам потребуются сведения о необходимых для работы функциях. Эта книга подробно и просто расскажет обо всех функциях, предоставляемых операционной системой Microsoft Windows. Справочная часть книги позволит быстро найти нужную информацию.
- ❑ Как выбрать микросхему USB-приемопередатчика при разработке своего контроллера:
 - существует множество микросхем для организации USB-интерфейса. Выбрать нужную достаточно сложно. Мы предоставим вам краткий обзор основных микросхем приемопередатчиков, присутствующих на российском рынке.
- ❑ Как создать USB-устройство на основе микропроцессора 8051:
 - мы предлагаем вместе пройти полный путь создания USB-устройства на основе микропроцессора 8051. После прочтения этих глав использование USB-интерфейса станет простым делом, ничуть не сложнее использования обычного COM-порта.

Программные требования

Все программы мы будем реализовывать на языках Borland Delphi 6 и Visual Studio 6. Так как мы не будем использовать никаких специфических функций, присущих именно этим версиям языков, то все примеры могут быть скомпилированы в других версиях практически без модификации.

Версия Windows должна быть или Windows 2000 или Windows XP. Желательно при этом наличие всех доступных пакетов обновлений (service pack). Возможно использование Windows 98, но многие примеры, связанные с написанием драйверов, могут не работать.

Для компиляции драйверов, приведенных в книге, потребуется Windows 2000 DDK или Windows XP DDK, в соответствии с вашей версией Windows.

В качестве дополнительного источника информации мы рекомендуем установить MSDN. Список полезных для работы утилит и программ приводится в справочной части книги, в *главе 19*.

На компакт-диске содержатся полные исходные тексты и скомпилированные модули.

Аппаратные требования

Достаточно обычного домашнего компьютера, на котором компиляция программы в Delphi и Visual Studio занимает приемлемое для вас время.

Установка Delphi 6 потребует примерно 300 Мбайт на жестком диске, установка Visual Studio — 240 Мбайт, MSDN — 1,5 Гбайт, Windows DDK — 700 Мбайт.

Для тестирования программ необходимо иметь одно или несколько USB-устройств. Для создания своих устройств будет необходим соответствующий инструментарий.

О программном коде

Книга содержит полные исходные коды всех программ, однако многие листинги содержат только изменения кода относительно предыдущего листинга. Такое сокращение позволяет не только экономить место, но и улучшить понимание кода, делая акцент только на новой функциональности. Код на компакт-диске содержит тексты без сокращений.

В программах на Delphi мы не приводим код самого проекта (DPR-файл) и код формы (DFM-файлы). Все исходные коды можно найти на компакт-диске, и, мы думаем, желающих набирать модули форм "с листа" найдется немного.

Еще раз повторим, что такие сокращения не означают отсутствие возможности скомпилировать и попробовать приведенные примеры на своем компьютере. Все исходные коды и необходимые модули находятся на компакт-диске.

Краткое описание глав

Первая часть книги содержит главы 1—2. В них приводится спецификация USB, описание USB-интерфейса и его составляющие. В этих главах мы не будем глубоко заглядывать внутрь USB: регистры контроллера, команды и тонкости аппаратной организации мы оставим на потом:

- ❑ глава 1 ("Что такое USB") содержит описание USB-интерфейса с точки зрения пользователя и потенциального покупателя компьютера; примеры USB-устройств и тонкости, которые необходимо знать при их покупке и покупке компьютера; правила установки и конфигурирования устройств, основные проблемы, возникающие при этом, и методы их решения. Кроме того, глава содержит определение основных понятий USB: хабы, концентраторы и т. д.;

- ❑ глава 2 ("Аппаратное обеспечение USB") содержит небольшое описание аппаратной части USB. До схемотехники и микросхем приемопередатчиков мы доберемся в следующих главах, а в этой описываются кабели, разъемы, принципы питания устройств и другие сведения, необходимые не только разработчику аппаратуры, но и обычному пользователю персонального компьютера для грамотного использования интерфейса USB.

Вторая часть книги содержит главы 3—6, которые описывают внутреннюю организацию USB:

- ❑ глава 3 ("Внутренняя организация шины USB") описывает внутреннее устройство шины, организацию и типы передач данных, методы синхронизации данных и правила вычисления контрольных сумм;
- ❑ глава 4 ("Внутренняя организация устройства") описывает структуры и запросы, используемые для обращения к USB-устройствам, дескрипторы устройств и функции их получения;
- ❑ глава 5 ("Внутренняя организация хоста и хабов") описывает структуры и запросы, используемые для взаимодействия хоста, хабов и устройств. Кроме того, глава содержит информацию об организации одновременной работы устройств с разными скоростями;
- ❑ глава 6 ("USB без ПК") описывает расширение спецификации USB, позволяющей соединять USB-устройства между собой без персонального компьютера.

Третья часть книги содержит главы 7—10, которые описывают реализацию поддержки USB-интерфейса в операционной среде Microsoft Windows:

- ❑ глава 7 ("Поддержка USB в Windows") содержит общие сведения о драйверной модели и методах взаимодействия с драйверами;
- ❑ глава 8 ("HID-устройства") содержит описание одного из классов USB-устройств, называемых HID;
- ❑ глава 9 ("Введение в WDM") дает общие сведения о модели драйверов Windows 2000/XP;
- ❑ глава 10 ("Спецификация PnP для USB") содержит описание спецификации Plug and Play для USB, структуры и функции Windows для системы PnP, а также общее описание структуры INF-файла;
- ❑ глава 11 ("Функции BIOS") содержит описание сервисов BIOS, используемых для работы с USB в DOS. Эта информация будет полезна для программистов, использующих промышленные контроллеры с установленным DOS или его клоном.

Четвертая часть книги содержит главы 12—14, которые описывают процесс создания USB-устройства:

- ❑ глава 12 ("USB-периферия") дает обзор микросхем USB-приемопередатчиков. Акцент делается на микросхемах, доступных на российском рынке;

- ❑ глава 13 ("HID-устройство на основе ATMEL AT89C5131") описывает схемотехнику и процесс разработки HID-устройства на основе микропроцессора AT89C5131 и программы, использующей HID-функции;
- ❑ глава 14 ("Создание USB-устройства на основе ATMEL AT89C5131") рассматривает процесс разработки USB-устройства на основе микропроцессора AT89C5131 и драйвера для Microsoft Windows;
- ❑ глава 15 ("Использование микросхемы FT232BM") представляет схемотехнику и процесс разработки USB-устройства на основе преобразователя FT232BM, а также содержит описание схемы преобразователя из COM-интерфейса в USB.

Часть пятая включает *главы 16—19*, в которых содержатся справочные материалы.

Обозначения

При описании некоторых данных мы будем пользоваться битовым представлением, заключая число разрядов каждого поля в квадратные скобки. Например:

- ❑ [5] поле А;
- ❑ [2] поле Б.

Такое описание означает, что поле А содержит 5 битов, а поле Б — 2 бита. Еще одно представление битовых полей — указание конкретных диапазонов битов, с помощью знака ":", например:

- ❑ [16:5] зарезервированы;
- ❑ [4:0] индекс.

Такое описание означает, что биты с 16 по 5 включительно зарезервированы, а биты с 4 по 0 включительно представляют собой индекс. Отличить первое описание от второго обычно легко по контексту изложения.

При написании чисел мы будем придерживаться следующих правил:

- ❑ шестнадцатеричные числа будут иметь префикс "\$", например "\$45";
- ❑ шестнадцатеричные числа могут иметь префикс "0x" или постфикс "H", если того требует контекст изложения или формат строки, например, "INT 3FH";
- ❑ битовые последовательности заключены в угловые скобки, например, "<0010>", либо, при написании двоичного числа, обозначены символом b в конце, например, 10101111b.

Для описания версий протоколов, структур и т. д. будет использоваться специальный тип чисел BCD (Binary-Coded Decimal). Такие числа записываются в шестнадцатеричном виде 0xJJMN для обозначения версии JJ.M.N, т. е.

Ж обозначает старший номер версии, М — младший номер версии и N — номер подверсии. Например, версия 2.1.3 будет представлена числом 0x213, а версия 2.0 будет записана числом 0x0200.

При описании регистров мы будем пользоваться следующими обозначениями режимов доступа:

- ❑ RO (read only) — регистр только для чтения, запись в него невозможна;
- ❑ WO (write only) — регистр только для записи, чтение значения невозможно;
- ❑ R/W (read/write) — возможны и чтение, и запись значения;
- ❑ R/W2 (read/write word) — возможны чтение и запись слова;
- ❑ R/WC (read/write clear) — разрешено как чтение, так и считывание значения, однако при записи в некоторый разряд регистра приводит к его сбросу в ноль.

При необходимости указания версии Windows мы будем использовать следующие сокращения:

- ❑ Windows 9x будет означать семейство Windows 95/98/ME;
- ❑ Windows NT в общем случае будет обозначать семейство Windows NT/2000/XP;
- ❑ при необходимости указания конкретной версии мы будем писать номер этой версии без сокращений, например, Windows NT4 или Windows 98.

Благодарности

В создании этой книги прямо или косвенно участвовало очень много людей. Прежде всего, хочется поблагодарить заместителя главного редактора издательства Евгения Рыбакова, автора идеи создания этой книги.

Нельзя не отметить тех, чьи материалы помогли наполнить книгу полезной и нужной информацией.

Информация о HID-устройствах (*глава 8*), идентификации устройств (*глава 2*) и некоторых микросхемах (*глава 12*) предоставлена Константином Вовк, компания KS Labs (www.is.svtonline.com/vks).

Автор выражает благодарность к. т. н. Малыгину И. В., Лысенко А. А., Назмутдинову Р. Ф. за предоставленные материалы о микросхемах FTDI (*глава 14*), опубликованные на сайте Института радиотехники (www.institute-rt.ru).

Материалы *главы 1* были бы неинтересны без иллюстраций, предоставленных компьютерным супермаркетом NIX (www.nix.ru).

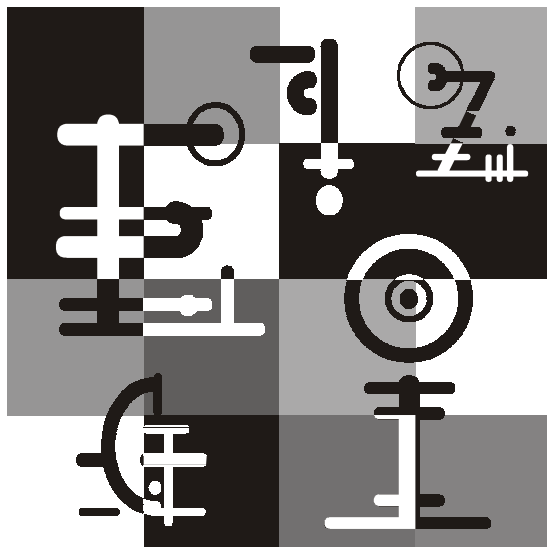
В составлении обзора микросхем автору помогали Сергей Воробьев (ООО "Автоматика-М", vesta.pvasoft.com), Геннадий Курзаев, Сергей Гудков, Игорь Кривченко (компания ООО "Эфо", www.efo.ru).

Спасибо Эрику Тинлоту (Eric Tinlot) — инженеру службы поддержки Atmel — за помощь при разработке примеров для микросхем Atmel (www.atmel.com).

При написании программ к *главе 11* существенную помощь оказывал Максим Локтюхин (компания Intel).

Разработку схем для микропроцессора AT89C5131 производил Сергей Малов. Без его участия и помощи книга лишилась бы практической части.

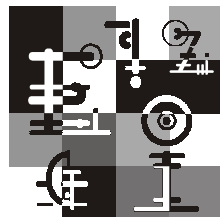
Автор благодарит всех друзей и родных, которые терпели его в процессе написания книги, а также коллектив издательства "БХВ-Петербург".



ЧАСТЬ I

ВВЕДЕНИЕ В USB

Глава 1



Что такое USB

USB — Unusable Serial Interface
(непригодный последовательный интерфейс).

Расшифровка аббревиатуры времен 1998 года

1.1. История USB

Увеличение числа устройств, подключаемых к персональному компьютеру, и, соответственно, развитие внешних интерфейсов привело к довольно неприятной ситуации: с одной стороны, компьютер должен иметь множество различных разъемов, а с другой — бóльшая часть из них не используется. Такая ситуация определяется историческим развитием интерфейсов ПК — каждый интерфейс имел свой специализированный разъем. Например, к последовательному порту можно подключить мышь или модем, к параллельному — принтер или сканер, для клавиатуры стало необходимо иметь два порта — старый клавиатурный и PS/2 и т. д. Более того, к одному порту можно подключить только одно устройство (если не считать подключение "прозрачных" ключей защиты, но это, скорее, исключение). Кроме этой проблемы, многочисленность разнообразных подключений добавляет и другие "радости":

- ☐ практически для каждого из устройств необходимо выделение аппаратного прерывания (IRQ);
- ☐ бóльшая часть устройств требует наличия внешнего блока питания;
- ☐ каждое устройство имеет свой, придуманный разработчиком, протокол обмена, многократно увеличивая необходимое количество драйверов, как в памяти, так и в инсталляции операционной системы;
- ☐ конфигурирование огромного числа устройств, многие из которых не поддерживают спецификации Plug and Play, — практически невыполнимая работа для обычного пользователя;
- ☐ огромное число разнокалиберных шлейфов, тянущихся от компьютера, превращает его перестановку в сложную проблему.

Естественно, что производители компьютерного "железа" задумались о создании единого и универсального интерфейса. В начале 1996 года была опубликована версия 1.0 нового интерфейса, названного USB (Universal Serial Bus, универсальная последовательная шина), а осенью 1998 — спецификация 1.1, исправляющая проблемы, обнаруженные в первой редакции. Весной 2000 года была опубликована версия 2.0, в которой предусматривалось 40-кратное повышение пропускной способности шины. Так, спецификации 1.0 и 1.1 обеспечивают работу на скоростях 12 Мбит/с и 1,5 Мбит/с, а спецификация 2.0 — на скорости 480 Мбит/с. При этом предусматривается обратная совместимость USB 2.0 с USB 1.x, т. е. "старые" USB 1.x устройства будут работать с USB 2.0 контроллерами, правда, на скорости 12 Мбит/с. Скорость 480 Мбит/с достигается только при одновременном использовании USB 2.0 контроллера и USB 2.0 периферии.

Изначально в группу разработчиков входили компании Compaq, DEC, IBM, Intel, Microsoft, NEC и Northern Telecom, а затем количество заинтересованных участников стало расширяться. Шина USB разрабатывалась для обеспечения механизма взаимодействия компьютерных и телефонных систем (CTI, Computer Telephony Integration), однако вскоре члены комитета разработки поняли, что USB может удовлетворить потребности многих приложений и все сферы компьютерной телефонии.

Разработчики шины ориентировались на создание интерфейса, обладающего следующими свойствами:

- ☐ легко реализуемое расширение периферии ПК;
- ☐ дешевое решение, позволяющее передавать данные со скоростью до 12 Мбит/с (480 Мбит/с для USB 2.0);
- ☐ полная поддержка в реальном времени голосовых, аудио- и видеопотоков;
- ☐ гибкость протокола смешанной передачи изохронных данных и асинхронных сообщений;
- ☐ интеграция с выпускаемыми устройствами;
- ☐ охват всевозможных конфигураций и конструкций ПК;
- ☐ обеспечение стандартного интерфейса, способного быстро завоевать рынок;
- ☐ создание новых классов устройств, расширяющих ПК.

Спецификация USB определяет следующие функциональные возможности интерфейса:

- ☐ простота использования для конечного пользователя:
 - простота кабельной системы и подключений;
 - скрывание подробностей электрического подключения от конечного пользователя;

- самоидентифицирующиеся устройства с автоматическим конфигурированием;
 - динамическое подключение и переконфигурирование периферийных устройств;
- широкие возможности работы:
- пропускная способность от нескольких Кбайт/с до нескольких Мбайт/с;
 - поддержка одновременно как изохронной, так и асинхронной передачи данных;
 - поддержка одновременных операций со многими устройствами (multiple connections);
 - поддержка до 127 устройств на шине;
 - передача разнообразных потоков данных и сообщений;
 - поддержка составных устройств (т. е. периферийное устройство, выполняющее несколько функций);
 - низкие накладные расходы передачи данных;
- равномерная пропускная способность:
- гарантированная пропускная способность и низкие задержки голосовых и аудиоданных;
 - возможность использования всей полосы пропускания;
- гибкость:
- поддержка разных размеров пакетов, которые позволяют настраивать функции буферизации устройств;
 - настраиваемое соотношение размера пакета и задержки данных;
 - управление потоком (flow control) данных на уровне протокола;
- надежность:
- контроль ошибок и восстановление на уровне протокола;
 - динамическое добавление и удаление устройств прозрачно для конечного пользователя;
 - поддержка идентификации неисправных устройств;
 - исключение неправильного соединения устройств;
- выгода для разработчиков:
- простота реализации и внедрения;
 - объединение с архитектурой Plug and Play;

❑ дешевая реализация:

- дешевые каналы со скоростью работы до 1,5 Мбайт/с;
- оптимизация для интеграции с периферией;
- применимость для реализации дешевой периферии;
- дешевые кабели и разъемы;
- использование выгодных товарных технологий;

❑ возможность простого обновления.

Практически все поставленные задачи были решены, и весной 1997 года стали появляться компьютеры, оборудованные разъемами для подключения USB-устройств. Иконкой, показанной на рис. 1.1, официально обозначается шина USB, как в Windows, так и на USB-разъемах (подробнее о логотипах и некоторых тонкостях, которые с ними связаны, мы расскажем в *разд. 1.9*).

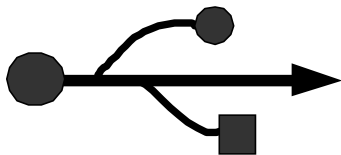


Рис. 1.1. Иконка USB-шины

В феврале 2004 года корпорация Intel совместно с Agere, Systems, HP, Microsoft Corporation, NEC, Philips Semiconductors и Samsung Electronics объявила о создании группы Wireless USB Promoter Group (группа продвижения беспроводного USB). В задачу консорциума входит продвижение первой высокоскоростной технологии для беспроводного подключения внешних устройств Wireless USB на скорости 480 Мбит/с (что сопоставимо с характеристикой стандарта USB 2.0) с дальностью действия при низком энергопотреблении до 10 метров.

1.2. Сравнение USB с другими интерфейсами

В табл. 1.1 приведено сравнение интерфейса USB с другими интерфейсами персонального компьютера. Видно, что достойной альтернативы USB не существует (пожалуй, кроме "изначального" конкурента — FireWire, но у этой шины принципиально другая система соединения¹). Интерфейсы,

¹ FireWire реализует систему "мастер—мастер", а USB — "мастер—ведомый".

сравнимые с USB по скорости обмена, требуют специальных преобразователей. Интерфейсы, не требующие дополнительных элементов, либо низкоскоростные, либо узконаправленные. Кроме того, к несомненным плюсам USB относятся организация помехозащищенности на уровне аппаратного и шинного протоколов и "встроенная" поддержка Plug and Play, а также отсутствие дополнительных элементов для подключения устройств (как, например, терминаторы для SCSI-интерфейса). Пожалуй, единственным минусом можно считать довольно короткое кабельное соединение, но следует помнить, что шина USB разрабатывалась как шина для домашних устройств, стоящих на столе, и дальние соединения не закладывались в нее изначально.

Таблица 1.1. Сравнение USB с другими интерфейсами

Интерфейс	Число устройств / Число проводов / Длина провода (м)	Скорость	Использование
Последовательные			
USB 2.0	127/3/10	1,5 Мбит/с, 12 Мбит/с, 480 Мбит/с	Любые устройства с USB 1.x/2.0 (USB-порт)
RS-232	1/6/50-100	115,2 Кбит/с	Модем, мышь, ключи за- щиты (COM-порт)
RS-485	32/2/4000	10 Мбит/с	Промышленные устройства (COM-порт через преобра- зователь)
FireWire (IEEE-1394)	64/3/15	400 Мбит/с	Видеоданные, дисковые массивы (FireWire-порт)
Ethernet	1024/3/1600	10 Мбит/с 100 Мбит/с 1 Гбит/с	Сетевые соединения ПК (сетевая карта)
Токовая петля			
MIDI	1/3/50	31,5 Кбит/с	Музыкальные устройства
Параллельные			
LPT	1/9/10-30	От 800 Кбит/с до 16 Мбит/с	Принтеры, сканеры, диско- вые устройства

1.3. Основные понятия USB

В этом разделе мы приведем основные понятия, которые используются при работе с шиной USB.

1.3.1. Общая архитектура шины

Обычная архитектура шины USB подразумевает подключение одного или нескольких *USB-устройств* к компьютеру (рис. 1.2). Компьютер в такой конфигурации является главным управляющим устройством и называется *хостом*. Подключение устройств к хосту производится с помощью *кабелей*. Для соединения компьютера и устройства используется *хаб*. Компьютер имеет встроенный хаб, называемый *корневым хабом*.

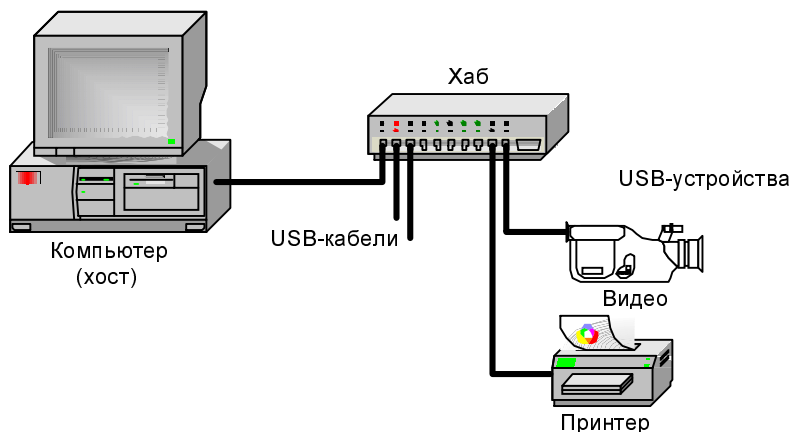


Рис. 1.2. Обычная архитектура USB

1.3.2. Физическая и логическая архитектура шины

Физическая архитектура USB-шины определяется следующими правилами (рис. 1.3):

- ☐ устройства подключаются к хосту;
- ☐ физическое соединение устройств между собой осуществляется по топологии многоярусной звезды, вершиной которой является корневой хаб;
- ☐ центром каждой звезды является хаб;

- ❑ каждый кабельный сегмент соединяет между собой две точки: хост с хабом или функцией (см. далее), хаб с функцией или другим хабом;
- ❑ к каждому порту хаба может подключаться периферийное устройство или другой хаб, при этом допускается до 5 уровней каскадирования хабов, не считая корневого.

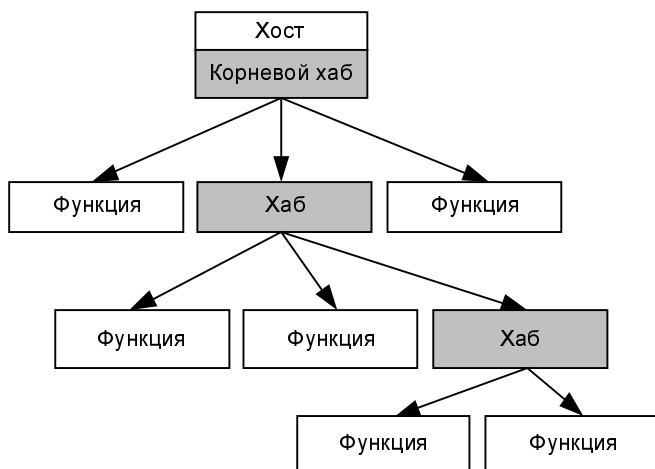


Рис. 1.3. Физическая архитектура USB

Детали физической архитектуры скрыты от прикладных программ в системном ПО, поэтому *логическая архитектура* выглядит как обычная звезда, центром которой является прикладное ПО, а вершинами — набор *конечных точек* (рис. 1.4).

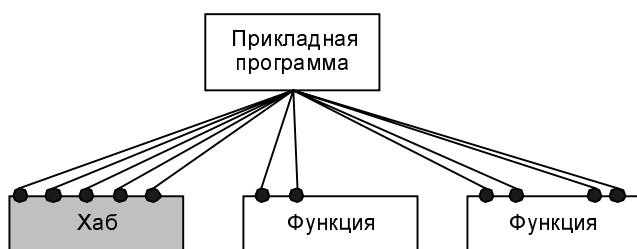


Рис. 1.4. Логическая архитектура USB

Прикладная программа ведет обмен информацией с каждой конечной точкой.

1.3.3. Составляющие USB

Шина USB состоит из следующих элементов.

Хост-контроллер (Host Controller) — это главный контроллер, который входит в состав системного блока компьютера и управляет работой всех устройств на шине USB. Для краткости мы будем писать просто "хост". На шине USB допускается наличие только одного хоста. Системный блок персонального компьютера содержит один или несколько хостов, каждый из которых управляет отдельной шиной USB. В *главе 6* мы рассмотрим одно исключение — соединение двух USB-устройств без персонального компьютера.

Устройство (Device) может представлять собой хаб, функцию или их комбинацию (Compound Device). Примеры USB-устройств приведены в *разд. 1.4*.

Порт (Port) — точка подключения.

Хаб (Hub, другое название — *концентратор*) — устройство, которое обеспечивает дополнительные порты на шине USB. Другими словами, хаб преобразует один порт (*восходящий порт*, Upstream Port) во множество портов (*нисходящие порты*, Downstream Ports). Архитектура допускает соединение нескольких хабов (не более 5). Хаб распознает подключение и отключение устройств к портам и может управлять подачей питания на порты. Каждый из портов может быть разрешен или запрещен и сконфигурирован на полную или ограниченную скорость обмена. Хаб обеспечивает изоляцию сегментов с низкой скоростью от высокоскоростных. Хаб может ограничивать ток, потребляемый каждым портом.

Корневой хаб (Root Hub) — это хаб, входящий в состав хоста.

Функция (Function) — это периферийное устройство (ПУ) или отдельный блок периферийного устройства, способный передавать и принимать информацию по шине USB. Каждая функция предоставляет конфигурационную информацию, описывающую возможности ПУ и требования к ресурсам. Перед использованием функция должна быть сконфигурирована хостом — ей должна быть выделена полоса в канале и выбраны опции конфигурации.

Логическое устройство (logical device) USB представляет собой набор конечных точек.

1.3.4. Свойства USB-устройств

Спецификация USB достаточно жестко определяет набор свойств, которые должно поддерживать любое USB-устройство:

- **адресация** — устройство должно отзываться на назначенный ему уникальный адрес, и только на него;

- ❑ **конфигурирование** — после включения или сброса устройство должно предоставлять нулевой адрес для возможности конфигурирования его портов;
- ❑ **передача данных** — устройство имеет набор конечных точек для обмена данными с хостом. Для конечных точек, допускающих разные типы передач, после конфигурирования доступен только один из них;
- ❑ **управление энергопотреблением** — любое устройство при подключении не должно потреблять от шины ток, превышающий 100 мА. При конфигурировании устройство заявляет свои потребности тока, но не более 500 мА. Если хаб не может обеспечить устройству заявленный ток, устройство не будет использоваться;
- ❑ **приостановка** — устройство USB должно поддерживать *приостановку* (Suspended Mode), при которой его потребляемый ток не превышает 500 мкА. Устройство должно автоматически приостанавливаться при прекращении активности шины;
- ❑ **удаленное пробуждение** — возможность *удаленного пробуждения* (Remote Wakeup) позволяет приостановленному устройству подать сигнал хосту, который тоже может находиться в приостановленном состоянии. Возможность удаленного пробуждения описывается в конфигурации устройства. При конфигурировании эта функция может быть запрещена.

1.3.5. Свойства хабов

Хаб выполняет коммутацию сигналов и выдачу питающего напряжения, а также отслеживает состояние подключенных к нему устройств, уведомляя хост об изменениях. Хаб состоит из двух частей — *контроллера* (Hub Controller) и *повторителя* (Hub Repeater).

Контроллер содержит регистры для взаимодействия с хостом. Доступ к регистрам осуществляется по специфическим командам обращения к хабу. Команды позволяют конфигурировать хаб, управлять нисходящими портами и опрашивать их состояние.

Повторитель представляет собой управляемый ключ, соединяющий выходной порт со входным. Он имеет средства сброса и приостановки передачи сигналов.

Нисходящие порты хабов могут находиться в следующих состояниях:

- ❑ **Питание отключено** (Powered off) — на порт не подается питание (возможно только для хабов, коммутирующих питание). Выходные буферы переводятся в высокоимпедансное состояние, входные сигналы игнорируются;
- ❑ **Отсоединен** (Disconnected) — порт не передает сигналы ни в одном направлении, но способен обнаружить подключение устройства;

- ❑ **Запрещен** (Disabled) — порт передает только сигнал сброса (по команде контроллера), сигналы от порта (кроме обнаружения отключения) не воспринимаются;
- ❑ **Разрешен** (Enabled) — порт передает сигналы в обоих направлениях. По команде контроллера или по обнаружении ошибки кадра порт переходит в состояние **Запрещен**, а по обнаружении отключения — в состояние **Отсоединен**;
- ❑ **Приостановлен** (Suspended) — порт передает сигнал перевода в состояние останова ("спящий режим"). Если хаб находится в активном состоянии, сигналы через порт не пропускаются ни в одном направлении.

Состояние каждого порта идентифицируется контроллером хаба с помощью отдельных регистров. Имеется общий регистр, биты которого отражают факт изменения состояния каждого порта. Это позволяет хосту быстро узнать состояние хаба, а в случае обнаружения изменений специальными транзакциями уточнить состояние.

1.3.6. Свойства хоста

Хост имеет следующие обязанности:

- ❑ обнаружение подключения и отключения устройств USB;
- ❑ управление потоками данных;
- ❑ сбор статистики;
- ❑ обеспечение энергосбережения подключенными ПУ.

Системное ПО контроллера управляет взаимодействием между устройствами и их ПО, функционирующим на хост-компьютере, для согласования:

- ❑ нумерации и конфигурирования устройств;
- ❑ изохронных передач данных;
- ❑ асинхронных передач данных;
- ❑ управления энергопотреблением;
- ❑ информации об управлении устройствами и шиной.

1.4. Примеры USB-устройств

Обычно USB-устройство представляет собой USB-функцию с портом для подключения. Типичными примерами функций являются:

- ❑ указатели: мышь, планшет, световое перо;
- ❑ устройства ввода: клавиатура, сканер;
- ❑ устройства вывода: принтер, звуковые колонки, монитор;

- ❑ телефонный адаптер ISDN;
- ❑ флеш-диски.

Часто USB-устройство имеет встроенный хаб, позволяющий подключать к нему другие устройства.

1.4.1. Мышь и клавиатура

Подключение USB-мыши может быть оправдано при необходимости освободить драгоценный последовательный порт, например, для подключения оборудования. Однако для мыши остается еще порт PS/2, поэтому USB-мышь, в общем-то, не особенно актуальное устройство, за исключением возможности конфигурирования частоты опроса (обычная последовательная мышь передает 40—60 пакетов данных в секунду, тогда как USB-мышь можно сконфигурировать на значительно бóльшую частоту), что оценят любители компьютерных игр.

Использование USB-клавиатуры, пожалуй, интересно только возможностью подключения USB-мыши прямо к клавиатуре, а также экономией системных ресурсов.

Если уж вы решили использовать USB-мышь или USB-клавиатуру, обязательно установите все обновления (service pack) для операционной системы. Без них с этими устройствами возникают проблемы. Например, "чистая" Windows 2000 "теряет" их при выходе из спящего режима. Кроме того, для использования их в старых операционных системах, например, в DOS, необходимо включать соответствующую опцию в BIOS (см. разд. 1.7).



Рис. 1.5. USB-мышь

1.4.2. Мониторы

Многих пользователей персонального компьютера название "USB-монитор" вводит в заблуждение, чаще всего строящееся на аналогии со звуковыми USB-колонками. Действительно, USB-колонки не требуют звуковой карты, однако USB-монитор все же требует графический адаптер (видеокарту). "USB" в названии означает наличие USB-портов, позволяющих подключать

USB-устройства непосредственно к монитору, а также возможность программного конфигурирования настроек монитора по USB-интерфейсу. При выборе монитора обращайте внимание на поддержку USB 2.0 и мощность USB-хаба, встроенного в монитор.



Рис. 1.6. USB-монитор

1.4.3. Переходники USB-to-COM и USB-to-LPT

Конвертеры USB-to-COM и USB-to-LPT незаменимы в тех случаях, когда последовательные и параллельные порты в системе уже заняты (или "сожжены", что с учетом отсутствия на них гальванической развязки довольно актуально). Как следует из названия, эти устройства позволяют подключать к USB-порту устройства с последовательным (мышь, модем) и параллельным (принтер, сканер) интерфейсами. Переходник USB-to-COM будет также полезен пользователям ноутбуков, т. к. в них имеется всего один последовательный порт.



Рис. 1.7. Переходник USB-to-COM

1.4.4. Сканеры

Основной интерес USB-сканера заключается, наверное, в отсутствии внешнего питания, что сокращает число необходимых розеток. Скорость работы таких сканеров ничем не отличается от обычных, т. к. основной определяющей является не скорость передачи данных, а скорость движения сканирующей головки.



Рис. 1.8. USB-сканер

1.4.5. Модемы

Разработчики USB не прошли и мимо модемов. Естественно, такие модемы не требуют внешнего питания и работают полностью от шины. С одной стороны, это позволило значительно уменьшить размеры самих модемов, но с другой, такие модемы имеют все достоинства и недостатки программных модемов (soft modem). На многих USB-модемах производители уменьшили число индикаторов состояния или используют программное отображение, что не очень удобно.



Рис. 1.9. USB-модем

1.4.6. Звуковые колонки

В отличие от мониторов, приставка "USB" для звуковых колонок означает более глобальные изменения в устройстве. USB-колонки не требуют звуковой карты, а преобразование сигнала в аналоговый происходит в самих колонках через встроенный аналого-цифровой преобразователь.

Важно.

Следует отличать понятия "USB-колонки" от "колонок с питанием от USB" ("USB Powered" или "Powered Speaker"). Второй вариант представляет собой обычные колонки, требующие звуковой карты, но без отдельного блока питания. Кроме наклейки, их можно отличить по дополнительному разъему к звуковой карте. При подключении к шине такие колонки даже не опознаются системой как новое устройство.

Некоторые трудности могут возникнуть при воспроизведении музыкальных дисков: прослушивать диски стандарта CD-DA возможно только тогда, когда в CD-ROM имеется поддержка Digital Playback (цифровое проигрывание). Качество звука, получаемое при использовании USB-колонок, значительно выше, чем с применением обычных колонок совместно с большинством звуковых карт. Единственное ограничение — компьютер должен иметь достаточную производительность для обеспечения непрерывного потока данных на колонки, иначе любое движение мыши способно привести к исчезновению звука.



Рис. 1.10. USB-колонки

Следует отметить, что поток данных, передаваемых на USB-колонки, довольно большой, что создает заметный трафик от компьютера к колонкам. По этой причине рекомендуется подключение колонок непосредственно к компьютеру либо к ближайшему хабу (см. разд. 2.2).

Итак, к плюсам USB-колонок можно отнести:

- ❑ простоту подключения — не надо открывать системный блок, добавлять платы и т. д.;

- ❑ помехозащищенность — т. к. аудиоинтерфейс не устанавливается внутрь компьютера, то наводки от системной платы, винчестера и т. д. отсутствуют полностью;
- ❑ мобильность и компактность — т. к. не требуется дополнительная плата внутри компьютера, то корпус может быть любой, даже ноутбук или портативный компьютер;
- ❑ возможность питания от шины (не надо лишних розеток).

К минусам можно отнести необходимость достаточного запаса производительности компьютера, а также возможные микрозадержки при воспроизведении. В обычных условиях они незначительны, но для студийной записи могут оказаться критичными.

1.4.7. Флеш-диски

Флеш-диски с USB-интерфейсом претендуют стать современным аналогом дискет. Совмещая все преимущества дискет, они обладают значительными преимуществами:

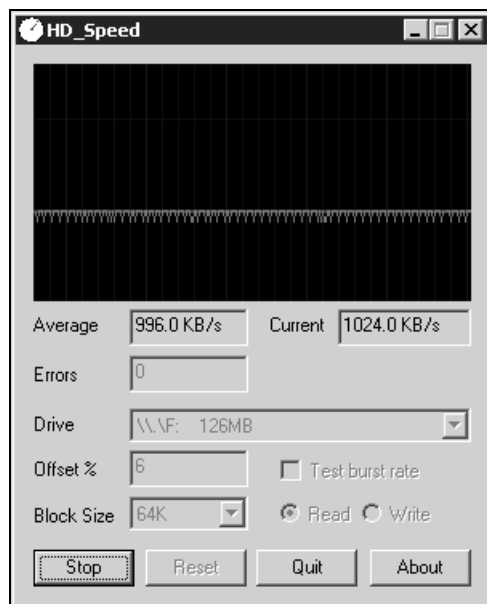
- ❑ флеш-диск можно подключить практически к любому современному компьютеру без выключения;
- ❑ диск может быть загрузочным;
- ❑ специальный переключатель позволяет заблокировать запись на диск, и он будет доступен только для чтения;
- ❑ поддерживается практически всеми операционными системами;
- ❑ скорость записи превышает запись на дискету, хотя и медленнее обычного жесткого диска (см. примечание к табл. 1.2);
- ❑ отсутствует вечная проблема дискет — сбойные секторы;
- ❑ ударостойкость около 1000 G (значительно больше любого современного жесткого диска);
- ❑ время хранения данных — не менее 10 лет;
- ❑ число циклов записи — не менее миллиона;
- ❑ объем диска значительно больше обычной дискеты: если дискета вмещает 1,2 Мбайт, то флеш-диск вмещает 64, 128 или 256 Мбайт.

При выборе флеш-диска стоит обратить внимание на поддерживаемый протокол. Если ваш компьютер поддерживает USB 2.0, то, несомненно, стоит выбрать соответствующий флеш-диск. В табл. 1.2 мы привели данные тестирования двух флеш-дисков: Transcend JetFlash 128 Мбайт (USB 1.1) и OTi Flash Disk 128 Мбайт (USB 2.0). Тестирование проводилось с помощью программы HD_Speed (рис. 1.11) на Pentium 4. Видно, что интерфейс 2.0 выигрывает в скорости обмена данными примерно в 8 раз.

Таблица 1.2. Сравнение скорости работы флеш-дисков 1.1 и 2.0

	USB 1.1	USB 2.0
Чтение	996 Кбайт/с	7846 Кбайт/с
Запись	940 Кбайт/с	7604 Кбайт/с

Для сравнения: скорость чтения данных с жесткого диска составляет примерно 22 Мбайт/с, а с дискеты — примерно 20 Кбайт/с.

**Рис. 1.11.** Тестирование флеш-диска

Немалую роль играют размеры и вес устройства. Отдельно можно обратить внимание на габариты — при подключении к порту устройство не должно мешать подключениям к другим портам, хотя при использовании USB-удливателя эта проблема не актуальна.

**Рис. 1.12.** Флеш-диски с USB-интерфейсом

Флеш-диски с интерфейсом USB 2.0 можно использовать не только как дискету, но и как переносной жесткий диск (быстродействия USB 1.1 может быть недостаточно для комфортной работы). С USB-диска можно запускать программы, редактировать файлы, не прибегая к помощи жесткого диска компьютера. Это значительно облегчает синхронизацию файлов между компьютерами, например, домашним и рабочим.

Примечание

Согласно спецификации USB к одному порту можно подключить 127 устройств с помощью соответствующего хаба. Однако на практике число флеш-дисков ограничено числом свободных букв дисков (26 букв минус A, B и C, т. е. максимум 23 диска).

Флеш-диски легко (и очень быстро) форматируются средствами Windows. При этом пользователь может сам выбирать тип файловой системы (FAT или FAT32). При форматировании диска с помощью утилиты Format необходимо указать тип файловой системы с помощью ключа FS, например:

```
format F: /FS:FAT32
```

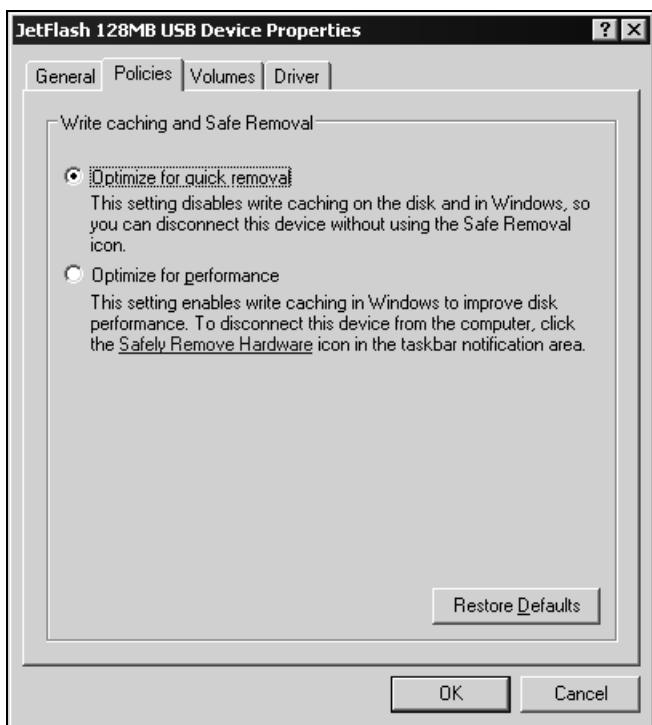


Рис. 1.13. Настройка кэширования флеш-диска

В операционной системе Windows доступна настройка кэширования (рис. 1.13). Кэширование может ускорить работу с диском, но пользоваться им следует с осторожностью: перед отключением диска следует убедиться, что все данные действительно успели записаться. Для корректного отключения нужно использовать иконку, появляющуюся рядом с системными часами. Двойной щелчок на этой иконке открывает диалог отключения диска. Для активизации отключенного диска нужно отключить его от USB-кабеля и подключить снова.

1.4.8. Хабы

Хабы не являются как таковыми USB-устройствами. Их задача — преобразовывать один USB-порт в несколько портов. Модели классифицируются по числу предоставляемых портов, поддерживаемым стандартам (обычно часть портов поддерживает USB 2.0, а несколько портов работают с USB 1.1) и типу питания.

Хабы могут быть внутренние (вставляемые в PIC-шину) и внешние (рис. 1.14). Питание внешних хабов обычно внешнее, однако бывают и исключения. Например, хаб Surecom EP-1004P может брать питание непосредственно от USB-шины, но, если к нему подключается особенно мощное устройство, допускается использование и внешнего блока питания.



Рис. 1.14. Хаб

1.4.9. Измерительная техника

Скорость передачи данных USB-канала позволяет использовать USB-шину для подключения измерительных приборов, таких как цифровые осциллографы (рис. 1.15), логические анализаторы, генераторы сигналов и т. п. В таких устройствах USB используется как для передачи данных в компью-

тер для их последующей обработки и отображения, так и для задания параметров приборов.



Рис. 1.15. USB-осциллограф

1.4.10. Экзотические устройства

В заключение приведем несколько экзотических устройств, появившихся на рынке компьютерной продукции. Воспользовавшись возможностью получать питание для устройств прямо от шины, разработчики задались вопросом "что еще можно подключить к USB". Список устройств, в общем-то, не относящихся к компьютерной периферии, довольно большой. Интересны, например, USB-фонарик (рис. 1.16), позволяющий осветить клавиатуру или рабочее место, или USB-вентилятор (рис. 1.17). Среди экзотических устройств можно найти USB-подогреватель для чашки, USB-грелку и даже USB-зубную щетку.



Рис. 1.16. USB-фонарик

Интерфейс USB проникает и в другие отрасли, казалось бы, совсем не связанные с компьютерами. Швейцарская компания Victorinox (www.victorinox.com), занимающаяся производством знаменитых армейских ножей, снабдила свою продукцию запоминающим устройством с портом USB (рис. 1.18).



Рис. 1.17. USB-вентилятор



Рис. 1.18. Нож с USB-накопителем

1.5. Сетевое соединение через USB

Существует два варианта сетевого соединения через USB-порт:

- ☐ конвертер USB-Ethernet;
- ☐ соединение двух компьютеров между собой через USB-порт.

Последний вариант — аналог нуль-модемного соединения по последовательному или параллельному порту, но с использованием USB.

1.5.1. Конвертер USB-Ethernet

Конвертер USB-Ethernet (рис. 1.19) обеспечивает подключение компьютера к сети Ethernet через порт USB без установки на компьютер специальной сетевой карты. Скорость обмена по такому соединению зависит от типа адаптера. Если адаптер поддерживает спецификацию USB 2.0, то скорость ограничивается 480 Мбит/с (естественно, оба компьютера также должны поддерживать USB 2.0, *см. разд. 1.9.1*), иначе скорость будет ограничена 12 Мбит/с.

По размерам преобразователь чуть больше спичечного коробка. Он годится как для настольного компьютера, так и для ноутбука, тем более, если PCMCIA-слот последнего занят под другие адаптеры.



Рис. 1.19. Конвертер USB-Ethernet
(TRENDnet TU2-ET100 USB2.0)

Такой конвертер можно назвать "сетевой картой с интерфейсом USB", что принципиально отличает его от прямого соединения через порт USB.

1.5.2. Прямое соединение через USB-порт

Кабель-адаптер для прямого соединения через USB-порт является аналогом нуль-модемного кабеля, но обеспечивает обмен со скоростью 480 Мбит/с для USB 2.0 (это почти в 500 раз быстрее соединения по параллельному порту!) и 12 Мбит/с для USB 1.1. Можно сказать, что это идеальное решение при необходимости объединения в сеть двух компьютеров (например, настольного компьютера и ноутбука).

1.6. Передача данных

Интерфейс USB предоставляет разработчику множество возможностей, избавляя его от самостоятельной реализации байтовых протоколов обмена,

подсчета контрольных сумм и других забот, необходимых для обеспечения надежной связи с устройствами.

1.6.1. Принципы передачи данных

Механизм передачи данных является асинхронным и блочным. Блок передаваемых данных называется *USB-фреймом* или *USB-кадром* (см. разд. 3.5) и передается за фиксированный временной интервал. Оперирование командами и блоками данных реализуется при помощи логической абстракции, называемой *каналом* (см. разд. 3.7). Внешнее устройство также делится на логические абстракции, называемые *конечными точками* (см. разд. 3.6). Таким образом, канал является логической связкой между хост-контроллером и конечной точкой внешнего устройства. Канал можно сравнить с открытым файлом.

Для передачи команд (и данных, входящих в состав команд) используется канал по умолчанию, а для передачи данных открываются либо *потокковые каналы*, либо *каналы сообщений* (см. разд. 3.7).

1.6.2. Механизм прерываний

Для шины USB настоящего механизма прерываний (как, например, для последовательного порта) не существует. Вместо этого хост-контроллер опрашивает подключенные устройства на предмет наличия данных о прерывании. Опрос происходит в фиксированные интервалы времени, обычно каждые 1—32 мс. Устройству разрешается посылать до 64 байт данных.

С точки зрения драйвера, возможности работы с прерываниями фактически определяются хост-контроллером, который и обеспечивает поддержку физической реализации USB-интерфейса. Подробности механизма прерываний будут рассмотрены в разд. 3.10.

1.6.3. Интерфейсы хост-адаптера

На сегодняшний день существует три типа интерфейса хост-контроллера (HC, Host Controller):

- ❑ UHCI (Universal Host Controller Interface, универсальный интерфейс хост-контроллера), разработка компании Intel;
- ❑ OHCI (Open Host Controller Interface, открытый интерфейс хост-контроллера), разработка Compaq, Microsoft, National Semiconductor;
- ❑ EHCI (Enhanced Host Controller Interface, расширенный интерфейс хост-контроллера), разработка Intel.

Интерфейс UHCI

Хост-контроллер UHCI отслеживает список кадров с 1024 указателями на структуры данных, соответствующих отдельному кадру. Он понимает два различных типа данных: описатели передач (TD, Transfer Descriptor) и начала очереди (QH, Queue Heads). Каждый TD представляет пакет, передаваемый от или в конечную точку устройства. QH имеют смысл для объединения TD (и QH) вместе.

Каждая передача состоит из одного или нескольких пакетов. Драйвер UHCI разделяет большие объемы передач на множество пакетов. Для каждой передачи, за исключением изохронных передач, формируется QH. Для каждого типа передачи эти QH объединяются в QH для этого типа. Изохронные передачи выполняются в первую очередь из-за фиксированных требований к устойчивости и непосредственно ссылаются по указателю на список кадров. Последний изохронный TD ссылается на QH для передачи прерываний для этого кадра. Все QH для передач прерываний указывают на QH для управляющих передач, которые, в свою очередь, указывают на QH для основных передач.

Более детальное описание UHCI содержится в *главе 18*.

Интерфейс OHCI

Спецификация OHCI разработана для уменьшения аппаратной сложности, требуя от драйвера хост-контроллера поддержки полного распределения передач для каждого кадра. Контроллеры типа OHCI гораздо более независимы и дают более абстрактный интерфейс, выполняя много работы самостоятельно.

Контроллер полагает, что имеется набор конечных точек, и заботится о планировании приоритетов и порядке следования типов передач в кадре. Основной структурой данных, используемой хост-контроллером, является описатель конечной точки (ED, Endpoint Descriptor), которому назначается очередь описателей передач (TD, Transfer Descriptors). ED хранит максимальный размер пакета, разрешенный для конечной точки, а аппаратура контроллера выполняет разбиение на пакеты. Указатели на буферы данных обновляются после каждой передачи, и когда начальный и конечный указатели совпадут, TD отбрасывается в очередь выполненных описателей. Четыре типа конечных точек имеют свои собственные очереди. Управляющие и обычные конечные точки ставятся каждая в свою собственную очередь. ED прерываний ставятся в очередь в дерево с уровнем в дереве, задающим частоту, с которой они выполняются.

Контроллер сначала выполняет очереди неперiodического управления и обычную очередь до момента времени, устанавливаемого драйвером ОС. Затем выполняются прерванные передачи для этого количества кадров, ис-

пользуя младшие пять бит номера кадра в качестве индекса в уровне 0 дерева прерываний ED. В конце этого дерева подключаются изохронные ED, и они выполняются последовательно. Изохронные TD содержат номер первого кадра, с которого должна начаться передача. После выполнения всех периодических передач снова обрабатываются управляющая и обычная очереди. Периодически вызывается подпрограмма обслуживания прерываний для обработки очереди выполненного и вызова соответствующих функций для каждой передачи и перепланирования изохронных конечных точек и прерываний.

1.6.4. Возможность прямого доступа к памяти

Устройства USB-шины не имеют прямого доступа к системной памяти (DMA, Direct Memory Access). Они изолированы от системных ресурсов USB-интерфейсом хост-компьютера и не поддерживают способа DMA передачи данных в привычном смысле. Тем не менее, USB-интерфейс хост-компьютера обеспечивает "иллюзию" поддержки DMA для логических каналов, обеспечивающих доступ к конечным точкам внутри подключенных к шине устройств. По мере получения данных от подключенных USB-устройств интерфейс хост-контроллера использует DMA-доступ для того, чтобы поместить полученные из устройства данные в системную память.

1.6.5. Режимы передачи данных

Пропускная способность шины USB, соответствующей спецификации 1.1, составляет 12 Мбит/с (т. е. 1,5 Мбайт/с). Спецификация 2.0 определяет шину с пропускной способностью 400 Мбайт/с. Полоса пропускания делится между всеми устройствами, подключенными к шине.

Шина USB имеет три режима передачи данных (табл. 1.3):

- ☐ низкоскоростной (LS, Low-speed);
- ☐ полноскоростной (FS, Full-speed);
- ☐ высокоскоростной (HS, High-speed, только для USB 2.0).

Обозначение

При необходимости указания конкретного режима работы мы будем пользоваться сокращениями LS, FS, HS. Для удобства мы добавим еще одно обозначение — AS ("Any Speed"), указывающее возможность работы в любом из перечисленных режимов.

Для краткости устройства, совместимые с режимом HS, мы будем называть *HS-устройствами*. Подразумевается, что при подключении таких устройств к интерфейсу USB 1.1 они могут работать в режиме FS.

Таблица 1.3. Категории USB-устройств по требуемой производительности

Категория	Приложения	Свойства
Низкоскоростные 1,5 Мбит/с	Диалоговые устройства: • клавиатура • мышь • световое перо • игровые приставки	• низкая стоимость • простота использования • "горячее" подключение/отключение • подключение нескольких устройств к одному порту
Полноскоростные 12 Мбит/с	Голосовой, аудио- и сжатый видеопоток: • телефонная станция • широкополосная сеть* • устройство звукозаписи • микрофон	• все свойства низкоскоростной категории • гарантированная полоса пропускания • гарантированная величина запаздывания
Высокоскоростные до 480 Мбит/с (только для USB 2.0)	Видео- и дисковые накопители: • видеокамеры • USB-диски • устройства резервного копирования	• все свойства полноскоростной категории • высокая полоса пропускания

* Технология, способная обеспечить одновременную передачу голоса, данных, видео; обычно это осуществляется путем мультиплексирования с разделением частот

1.7. Установка и конфигурирование USB-устройств

Спецификация USB была разработана с непосредственной поддержкой спецификации Plug and Play. Каждое устройство при подключении к шине USB сигнализирует о своем существовании и сообщает идентификатор производителя и идентификатор устройства. Эти идентификаторы являются определяющей информацией при выборе загружаемого драйвера, информация о котором ищется в реестре. Если подходящего драйвера в реестре не обнаружено, производится процедура установки нового устройства (драйвера). Спецификация Plug and Play предполагает прозрачное подключение и автоматическое конфигурирование устройств. Вмешательство пользователя требуется в том случае, когда система либо не смогла найти нужный драйвер и запрашивает его местоположение у пользователя, либо системе не удалось корректно распределить системные ресурсы.

Для подключения устройств к шине не требуется дополнительных действий (как, например, установка перемычек при подключении жестких дисков к IDE-интерфейсу), а возможность неправильного подключения исключается разными разъемами.

Примечание

В Windows 98 подключение некоторых устройств может вызвать некоторые затруднения. Чаще всего это связано с необходимостью установки дополнительных драйверов. Так, например, при установке флеш-диска рекомендуется сначала установить драйвер, предоставляемый производителем, и только после этого подключать само устройство.

В Windows XP небольшое окошко с сообщением позволяет пользователю узнать о появлении нового оборудования (рис. 1.20), а затем рядом с системными часами появляется еще одно сообщение, уведомляющее о добавлении нового устройства в систему (рис. 1.21). Никаких дополнительных действий от пользователя не требуется.

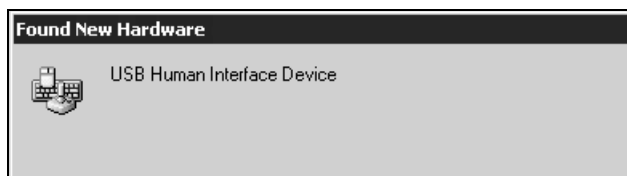


Рис. 1.20. Уведомление о подключении нового устройства в Windows XP



Рис. 1.21. Новое устройство добавлено в систему

Многие производители предоставляют свои драйверы для устройств. Например, с USB-мышью Dialog MO-03U предоставляется дискета с драйверами. Windows 2000 и XP прекрасно работают с помощью своих собственных драйверов, но установка драйверов производителя позволяет конфигурировать дополнительные функции, например, реакцию на нажатие третьей клавиши мыши (рис. 1.22).

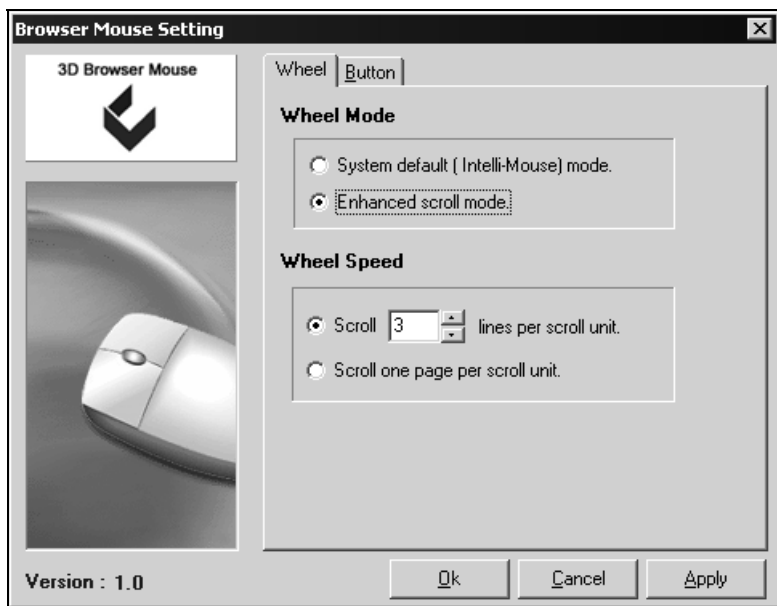


Рис. 1.22. Дополнительные настройки свойств USB-мыши

Так как обмен данными по шине USB идет только между компьютером и устройствами, то при подключении устройств следует учитывать потребляемую ими полосу пропускания. Устройства с большими объемами приема и/или передачи данных должны подключаться либо к самому компьютеру, либо к ближайшему свободному узлу. На рис. 1.23 показан пример правильной разводки сети: наивысший трафик у USB-колонок (почему это так, описано в *разд. 1.4.6*), затем идут сканер и принтер, а завершает цепь клавиатура и мышь, трафик которых практически нулевой.

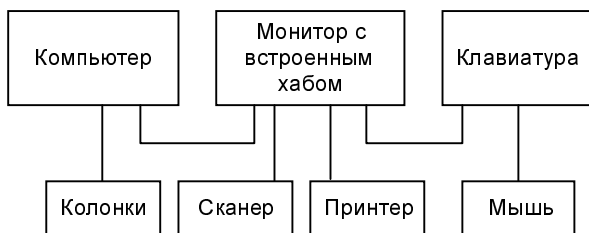


Рис. 1.23. Правильная разводка USB-сети

С другой стороны, при подключении устройств следует учитывать и поддерживаемые ими стандарты USB: устройства USB 2.0 должны быть под-

ключены к портам 2.0, а устройства с интерфейсом 1.1 можно подключать как к портам 2.0, так и к портам 1.1. Вполне понятно, что устройство USB 2.0 сможет реализовать высокую скорость обмена, только если по пути от него к хост-контроллеру (тоже 2.0) будут встречаться только хабы 2.0. Если это правило нарушить, и между ним и контроллером 2.0 окажется "старый" хаб, то связь может быть установлена только в режиме FS. Если такая скорость устройство и клиентское ПО устроит (к примеру, для принтера и сканера это выльется только в большее время ожидания пользователя), то подключенное устройство работать будет, но появится сообщение о неоптимальной конфигурации соединений. Устройства и ПО, критичные к полосе пропускания шины, в неправильной конфигурации работать откажутся и категорично потребуют переключений. Если же хост-контроллер старый, то все достоинства USB 2.0 окажутся недоступны пользователю. В этом случае придется менять хост-контроллер, т. е. системную плату. Контроллер и хабы USB 2.0 позволяют повысить суммарную пропускную способность шины и для старых устройств. Если устройства FS подключать к разным портам хабов USB 2.0 (включая и корневой), то для них суммарная пропускная способность возрастет по сравнению с 12 Мбит/с во столько раз, сколько используется портов высокоскоростных хабов.

1.7.1. Настройки BIOS для USB

BIOS (Base Input Output System) — базовая система в/в, является самым низким программным уровнем компьютера. Разные версии BIOS предоставляют разные настройки для конфигурирования интерфейса USB. В этом разделе мы рассмотрим основные параметры, позволяющие более гибко использовать возможности USB и решать многие проблемы. Названия пунктов отсортированы по алфавиту, некоторые пункты могут иметь несколько названий.

Assign IRQ For USB **(выделение прерывания для USB)**

Возможные названия: USB IRQ, Use IRQ for USB.

Опции: Enabled (включено), Disabled (выключено).

Эта опция разрешает или запрещает назначение прерывания для контроллера шины USB. Поскольку в компьютере часто не хватает прерываний, разрешать этот параметр следует только при наличии устройства на шине USB в системе.

Port 64/60 Emulation (эмуляция портов 64/60)

Возможные названия: USB Port 64/60 Emulation.

Опции: Enabled (включено), Disabled (выключено).

Если опция установлена в Disabled, USB-клавиатура будет нормально функционировать в различных ОС. Установка в Enabled включает эмуляцию одноканальных портов 0060 и 0064, которые необходимы для работы под Windows NT, взаимодействующей с некоторым периферийным "железом" иначе, чем другие операционные системы.

USB 1.1 Controllers (поддержка USB 1.1)

Опции: Enabled (включено), Disabled (выключено).

Позволяет отключать поддержку контроллером стандарта USB 1.1.

USB 2.0 Controller (поддержка USB 2.0)

Опции: Enabled (включено), Disabled (выключено).

Если не используются устройства стандарта USB 2.0, эту опцию можно выставить в Disabled. В связи с появлением большого числа устройств нового стандарта эту опцию лучше оставить в Enabled.

USB 2.0 HS Reference Voltage (управление уровнем напряжения)

Опции: Low (низкий), Medium (средний), High (высокий), Max (максимальный).

Многие системные платы позволяют программно управлять уровнем напряжения, подаваемым высокоскоростным USB-контроллером спецификации 2.0 в интерфейс. Это может оказаться полезным, если к одной шине подключено несколько устройств, не имеющих собственного питания. Для подключения одного устройства оптимальное значение Low.

USB Beep Message (звуковые сообщения от USB)

Опции: Enabled (включено), Disabled (выключено).

Эта опция включает или выключает звуковой сигнал при проверке состояния подключенных устройств во время проведения первоначального тестирования системы или в рабочем режиме. AMIBIOS8 обеспечивает три различных типа звуковых сигналов: при подключении устройства, при отсоединении устройства и при ошибке подключения.

USB Controllers (USB контроллер)

Возможные названия: USB Interface, Integrated USB Controller, On Board USB Controller, USB Host Controller, USB Function, On Board USB Function, USB Legacy Support, Legacy USB Support или On Chip USB.

Опции: Enabled (включено), Disabled (выключено), Auto (автоматически).

Эта опция позволяет включать или выключать установленный на материнской плате контроллер USB. Отключение USB (в случае отсутствия USB-устройств) позволяет освободить занятое контроллером прерывание. Современные системные платы позволяют конфигурировать USB 1.1 и USB 2.0 отдельно и, кроме того, позволяют настраивать каждый из USB-портов. Некоторые платы предоставляют опцию Auto, означающую автоматическое включение контроллера USB при обнаружении подключенного USB-устройства.

USB Function for DOS (поддержка USB для DOS)

Возможные модификации: USB KB/Mouse Legacy Support, USB Keyboard Legacy Support, USB Keyboard Support, USB Mouse Support, USB Keyboard Support Via, USB Mouse Support Via.

Опции: Enabled (включено), Disabled (выключено).

Поддержка USB может осуществляться двумя способами: либо на уровне драйверов операционной системы, либо на уровне BIOS. Эта опция позволяет включать поддержку USB для операционных систем, не поддерживающих USB, например, DOS. Для Windows эту опцию лучше выключить.

Модификации этой опции могут позволять включать поддержку BIOS отдельно для мыши и клавиатуры или позволять выбирать опции OS (поддержка через операционную систему), BIOS (поддержка через BIOS).

USB IRQ Remapping (переназначение прерывания)

Опции: Enabled (включено), Disabled (выключено).

Позволяет USB-контроллеру выбирать другое (нестандартное) прерывание в случае конфликтной ситуации. Установка этой опции в Disable рекомендуется только при отсутствии USB-устройств или абсолютно стабильной ситуации с распределением прерываний.

USB Latency Time (PCI CLK) (установка интервала владения шиной)

Опции: 32, 64, 96, 128, 160, 192, 224, 248, Disabled (выключено).

Устанавливает минимальный временной интервал владения шиной PCI. Опция отсутствует в современных платах с USB 2.0, т. к. в них интерфейс функционирует через специальные контроллеры, вынесенные за пределы PCI-интерфейса.

USB Passive Release

(режим параллельной работы ISA и PCI)

Опции: Enabled (включено), Disabled (выключено).

Эта опция включает/выключает механизм параллельной работы шин ISA и PCI. Если этот параметр разрешен (Enabled), то доступ процессора к шине PCI разрешен во время "пассивного разделения" или, как говорят иногда, ее "освобождения". Обычные циклы шины PCI состоят из 8 тактов. Интерфейс шины USB допускает более короткие циклы, освобождая ведущую шину в середине обычного цикла. Это ускоряет доступ к шине других устройств.

USB Speed

(скорость работы USB)

Опции: "24 MHz", "48 MHz".

Позволяет изменять скорость работы USB-шины. Поддерживается не всеми системными платами и версиями BIOS.

USB Storage Device Support

(поддержка устройств хранения информации)

Опция разрешает или запрещает поддержку подключенных к интерфейсу устройств хранения информации и их инициализацию во время старта компьютера.

Wake On USB for STR State

(режим "пробуждения" по USB)

Опции: Enabled (включено), Disabled (выключено).

Опция разрешает "пробуждение" компьютера, находящегося в "спящем" состоянии. Как правило, для того, чтобы этот параметр действовал, необходима установка соответствующего переключателя на материнской плате. Кроме этого, источник питания компьютера должен поддерживать ток до 2 А по цепи +5 В.

1.7.2. Устранение проблем

Если подключенное USB-устройство не распознается или работает с ошибками, прежде всего убедитесь, что на вашем компьютере установлены все доступные на текущий момент пакеты обновления Windows (Windows Service Pack). Особенно это относится к Windows XP, в котором работа с USB без (хотя бы) SP1 затруднительна.

Техническая служба Microsoft описывает следующие возможные причины неполадок:

- ☐ неработающее или неправильно настроенное оборудование;
- ☐ неработающий, неправильно настроенный или отсутствующий драйвер устройства;
- ☐ использование неподходящих кабелей;
- ☐ устаревшие микропрограммы или базовая система ввода/вывода (BIOS);
- ☐ неправильно настроенный корневой концентратор (хаб).

Неработающее или неправильно настроенное оборудование

Обычно подключение неработающего или неправильно настроенного оборудования к порту USB приводит к тому, что компьютер перестает отвечать на запросы (зависает). В большинстве случаев после этого необходимо выключить и повторно включить компьютер для перезапуска шины. Определить неработающее или неправильно настроенное устройство иногда бывает достаточно сложно. Подключите устройство к другому компьютеру и проследите, возникает ли та же проблема повторно.

Если устройство подключено к вспомогательному концентратору, попробуйте подключить его непосредственно к корневому концентратору.

Неисправности могут быть вызваны определенными неполадками оборудования (слишком высокое или слишком низкое напряжение в сети, не хватает пропускной способности, неработающие или неправильно настроенные микропрограммы).

Откройте диспетчер устройств и убедитесь, что корневой концентратор функционирует правильно. Если корневой концентратор отображается с восклицательным знаком, заключенным в желтый круг, убедитесь, что BIOS назначает запрос на прерывание (*см. разд. 1.6.1*) корневому контроллеру USB. Это необходимо для загрузки драйвера устройства.

Если ни одно из устройств не работает при подключении через корневой концентратор, убедитесь, что напряжение не превышает потребляемую мощность шины. Устройства USB потребляют не более 500 мА на каждое соединение. Также необходимо помнить, что если устройство потребляет меньше 50 мА, то порт не активируется.

На вкладке **Power** (Питание) в окне свойств USB корневого концентратора проверьте потребление питания шины USB (рис. 1.24).

Некорректные настройки параметров BIOS могут привести к сбою загрузки USB-драйверов. В этом случае при загрузке системы отображается сообщение, показанное на рис. 1.25.

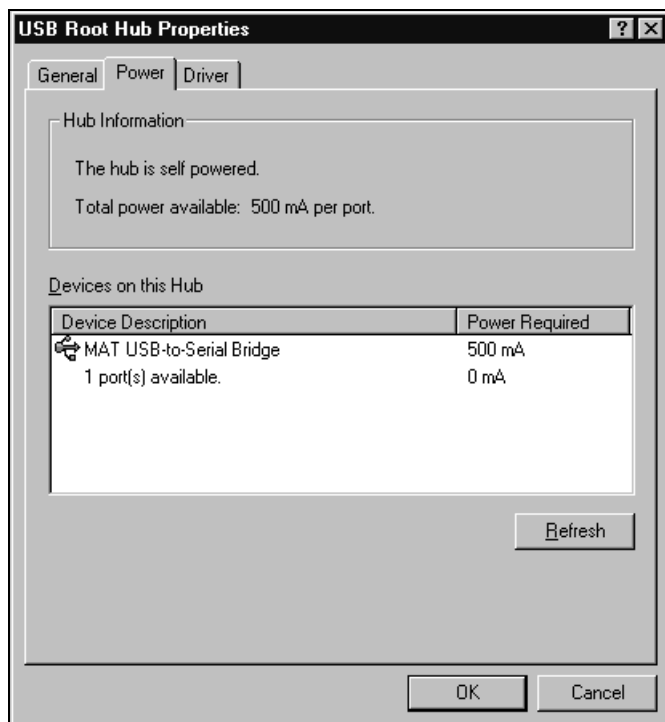


Рис. 1.24. Потребление питания шины USB

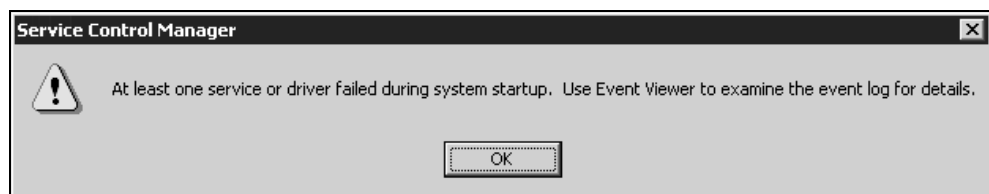


Рис. 1.25. Сбой загрузки драйверов

Неработающий, неправильно настроенный или отсутствующий драйвер устройства

При подключении устройства USB операционная система Windows загружает и настраивает драйверы для нового устройства полностью автоматически, за исключением случаев, когда подходящие драйверы отсутствуют. Устройство с неустановленными драйверами работать не будет. Драйверы можно найти на компакт-диске или диске, поставляющей вместе с устройством или в Интернете, например, на сайте производителя устройства.

Неподходящие кабели

Существуют высокоскоростные и низкоскоростные кабели USB. Основное их различие заключается в экранировании. Подключение высокоскоростного устройства с помощью низкоскоростного кабеля на больших расстояниях может вызвать искажение сигнала.

Проверьте всю цепь USB и убедитесь, что устройство, получающее питание от концентратора, не подключено в цепь с обратной стороны несилового концентратора. Это приводит к временному отключению концентратора и всех устройств, расположенных дальше по цепи. Убедитесь, что электропитание концентратора настроено правильно.

Устаревшие микропрограммы или базовая система ввода/вывода (BIOS)

В работе всех устройств USB основную роль играют микропрограммы (firmware) обслуживания USB, содержащие все сведения об устройстве. Компьютер также имеет такую микропрограмму, являющуюся частью BIOS. Рекомендуется всегда использовать последние версии BIOS.

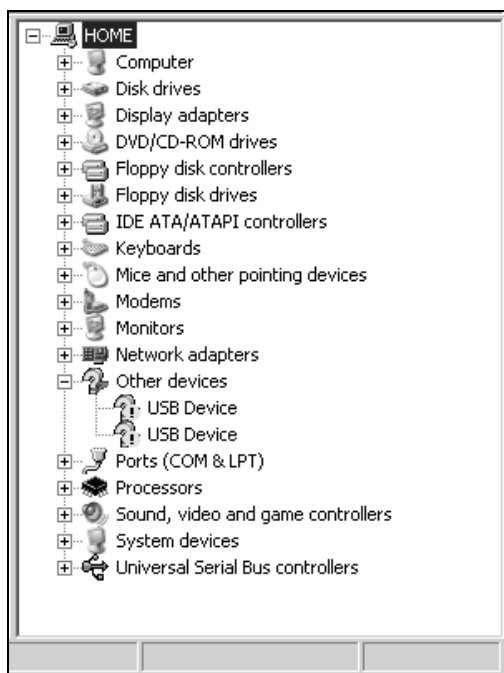


Рис. 1.26. Неверная установка драйвера устройства

Признаки неисправности или неправильной конфигурации микропрограмм могут быть очень разнообразными. Как правило, после удаления и повторной установки устройства USB оно снова становится доступным. Устройство может отображаться на экране и загружаться в диспетчер устройств в качестве второго экземпляра. Если отображается дубликат устройства, убедитесь, что для него используются обновленные микропрограммы. Эта проблема часто возникает при использовании принтеров и модемов USB. Подобная проблема возникает, если устройство загружает драйвер, а затем добавляет вторую копию устройства, для которой нет драйвера. В окне диспетчера устройств вторая копия устройства выводится с восклицательным знаком, заключенным в желтый круг (рис. 1.26). Само устройство может функционировать нормально, но удалить "призрак" можно только отключив устройство, "породившее" его. Возможно, устранить неполадку удастся путем обновления микропрограмм или драйвера устройства.

Неправильная настройка корневого концентратора

Контроллеры USB требуют назначенного запроса на прерывание (IRQ). Линия запроса на прерывание назначается в BIOS (обычно это IRQ 9).

1.8. Ограничения USB

Согласно спецификации USB, к одному разъему можно подключить до 127 устройств, но на практике это совсем не так. Ограничения накладывают пропускная способность и мощность USB-шины. Для USB 1.1 между устройствами делится всего 12 Мбит/с, поэтому одновременная работа особо "прожорливых" устройств будет затруднена. Интерфейс USB 2.0 более быстрый, однако ограничение и тут существует. Ограничение мощности актуально для устройств, которые берут питание непосредственно от шины, однако от него легко избавиться добавлением хаба с питанием от сети.

С точки зрения работоспособности устройств USB-шина может ограничивать их быстродействие только ограниченностью пропускной способности. Так, при подключении к USB 2.0 CDROM может работать на скорости 52x, но для USB 1.1 не стоит ожидать более 8x.

Еще одно ограничение: стандарт USB был разработан как "настольная шина" (desktop bus), с тем расчетом, что все внешние устройства будут в пределах досягаемости. Максимальная длина кабеля для FS-режима может быть всего 5 метров (и 3 метра в LS-режиме). Другие интерфейсы, такие как RS-232, RS-485 или Ethernet, позволяют использовать значительно более длинные кабели (см. табл. 1.1). Конечно, можно использовать 5 хабов для увеличения длины кабеля до 30 метров или преобразователи в RS-485.

1.9. Если вы покупаете компьютер

Если вы ориентируетесь на активное использование USB, при выборе компьютера следует обратить внимание на системную плату и корпус. При выборе USB-устройств, кроме характеристик, специфичных для каждого устройства, обратите внимание на мелкий шрифт рядом с этикеткой USB.

1.9.1. HS и USB 2.0 — не одно и то же!

Мы уже рассказывали про наклейку "USB Powered" на звуковых колонках, обозначающую всего лишь питание устройства от USB-шины, а вовсе не USB-интерфейс. Еще один тонкий момент связан с обозначением "USB 2.0". Согласно спецификации USB 2.0 пропускная способность этого интерфейса составляет 480 Мбит/с, однако в спецификации заложена возможность функционирования устройств в режимах FS и HS. Таким образом, реальную пропускную способность 480 Мбит/с могут обеспечить только устройства, способные работать в режиме HS.

"Основатели" USB рекомендуют использовать логотип "USB 2.0" только для HS-устройств, но многие производители используют его и для FS-устройств, являющихся, по сути, обычными устройствами USB 1.1. Для избежания путаницы между "USB 2.0" и "Hi-speed USB" (высокоскоростной USB) производителям рекомендуется использовать соответствующий логотип (рис. 1.27, 1.28).



Рис. 1.27. Логотип USB



Рис. 1.28. Логотип высокоскоростного USB (Hi-speed USB)

1.9.2. Системная плата

Конечно, описывать общие критерии выбора системной платы мы не будем. А вот с точки зрения использования USB, стоит обратить внимание на число USB-разъемов, USB-контроллеров, поддерживаемые протоколы и возможность подключения дополнительных разъемов для передней панели корпуса (см. разд. 1.9.3).

В табл. 1.4 мы привели несколько системных плат и их характеристики. Не стоит расценивать эту таблицу как наилучший выбор, а скорее, просто как возможные варианты наборов USB.

Таблица 1.4. Сравнительная таблица системных плат для Pentium 4

Плата	Чипсет	Конфигурация USB
ABIT IC7-MAX3	Intel 875P (RG82875P + FW82801ER)	4 USB 2.0 + 2 разъема по 2 USB 2.0
ABIT IS7-G	Intel 865PE/ICH5R (RG82865PE + FW82801ER)	4 USB 2.0 + 2 разъема по 2 USB 2.0
ABIT IS7-V	Intel 848P/ICH5 (RG82848P + FW82801EB)	4 USB 2.0 + 2 разъема по 2 USB 2.0
ABIT VI7	VIA PT800 (PT800 + VT8237)	4 USB 2.0 + 2 разъема по 2 USB 2.0
Albatron PX848 Like Pro	ALi 1683 (M1683 + M1563)	2 USB 2.0 + 2 разъема по 2 USB 2.0
Albatron PX865PE Lite Pro	Intel 848P/ICH5 (RG82848P + FW82801EB)	2 USB 2.0 + 2 разъема по 2 USB 2.0
Albatron PX865PE Pro	Intel 865PE/ICH5 (RG82865PE + FW82801EB)	2 USB 2.0 + 3 разъема по 2 USB 2.0
AOpen AX4SPE Max	Intel 865PE/ICH5R (RG82865PE + FW82801ER)	6 USB 2.0 + 1 разъем на 2 USB 2.0
ASUS P4P800 Deluxe	Intel 865PE/ICH5R (RG82865PE + FW82801ER)	4 USB 2.0 + 2 разъема по 2 USB 2.0
ASUS P4P800S-E Deluxe	Intel 848P/ICH5R (RG82848P + FW82801ER)	4 USB 2.0 + 2 разъема по 2 USB 2.0
ASUS P4P800S-E Deluxe	Intel 848P (RG82848P + FW82801ER)	4 USB 2.0 + 2 разъема по 2 USB 2.0
ASUS P4P8X	Intel 865P/ICH5 (RG82865P + FW82801EB)	4 USB 2.0 + 2 разъема по 2 USB 2.0
ASUS P4S800D Deluxe	SiS655FX (SiS655FX + SiS964)	4 USB 2.0 + 2 разъема по 2 USB 2.0

Таблица 1.4 (окончание)

Плата	Чипсет	Конфигурация USB
Canyon 9I6GM-L	Intel 865G/ICH5 (RG82865G + FW82801EB)	4 USB 2.0 + 2 разъема по 2 USB 2.0
Canyon 9I6PEA-L	Intel 865PE/ICH5 (RG82865PE + FW82801EB)	2 USB 2.0 + 3 разъема по 2 USB 2.0
Canyon 9I7PA-L	Intel 875P/ICH5R (RG82875P + FW82801ER)	4 USB 2.0 + 2 разъема по 2 USB 2.0
DFI 648FX-ALE	SiS648FX (SiS648FX + SiS963L)	2 USB 2.0 + 2 разъема по 2 USB 2.0
DFI 848P-AL	Intel 848P/ICH5 (RG82848P + FW82801EB)	4 USB 2.0 + 2 разъема по 2 USB 2.0
ECS Photon PF1	Intel 865PE/ICH5 (RG82865PE + FW82801EB)	4 USB 2.0 + 2 разъема по 2 USB 2.0
Fujitsu-Siemens D1627-A	Intel 865PE/ICH5R (RG82865PE + FW82801ER)	4 USB 2.0 + 2 разъема по 2 USB 2.0
Gigabyte 8PENXP	Intel 865PE/ICH5 (RG82865PE + FW82801EB)	4 USB 2.0 + 2 разъема по 2 USB 2.0
MSI 848P Neo-LS	Intel 848P/ICH5 (RG82848P + FW82801EB)	4 USB 2.0 + 2 разъема по 2 USB 2.0
MSI 865PE Neo2-FIS2R	Intel 865PE/ICH5R (RG82865PE + FW82801ER)	6 USB 2.0 + 1 разъем на 2 USB 2.0
Shuttle AB60/RS	Intel 865PE/ICH5 (RG82865PE + FW82801EB)	4 USB 2.0 + 2 разъема по 2 USB 2.0

1.9.3. Корпус

Если не считать "обычных" факторов (таких как цвет, размер, удобство сборки, удобство крепежа и т. п.), при выборе корпуса стоит обратить внимание на наличие интерфейсов на лицевой панели. Многие корпуса имеют один или два USB-порта прямо на лицевой панели (рис. 1.29), что удобно при частом использовании этого интерфейса. Правда, не все системные платы имеют дополнительные выводы для подключения этих разъемов.

1.9.4. USB для "старых" моделей компьютеров

В "старых" моделях системных плат USB-интерфейс отсутствует. Для подключения USB-устройств к таким компьютерам используются специальные платы расширения (USB-адаптеры). Большинство имеющихся адаптеров

подсоединяется через шину PCI, но можно найти их и в варианте для ISA. В установке они не сложнее других плат расширения.

Пример платы расширения показан на рис. 1.30. Эта плата (Controller Adaptec AUA-5100B (ОЕМ) PCI) предоставляет один внутренний и 5 внешних USB-портов стандарта USB 2.0.



Рис. 1.29. Корпус с двумя USB-разъемами на передней панели Inwin s535 (каталог компании НИКС)



Рис. 1.30. Плата расширения Controller Adaptec AUA-5100B (каталог компании НИКС)

1.10. Интернет-ресурсы к этой главе

К разделу 1.4:

☐ Компьютерный супермаркет NIX

- www.nix.ru

☐ Что можно воткнуть в USB?

- <http://www.compulenta.ru/dk/p/2004/1/28/44688/>

- ☐ Немного о современных USB-устройствах
 - <http://baranov.ru/news/viewnews.php?id=397>
- ☐ Устранение неполадок при использовании устройств USB в Windows XP
 - <http://support.microsoft.com/default.aspx?scid=kb;ru;310575>
- ☐ Швейцарский армейский нож оснащают портативным накопителем с портом USB
 - <http://lenta.ru/internet/2004/03/12/knife/>
- ☐ Измерительные USB-приборы
 - <http://www.aktakom.ru/product/kio/vpr.htm>
- ☐ USB-устройства для звукозаписи и многоканального звука
 - <http://www.musicmag.ru/info/soundcards/usbaudio.htm>
- ☐ 2:0 в пользу USB
 - <http://www.ferra.ru/online/storage/13374/>
- ☐ Обзор миниатюрного переносного накопителя Apacer HandyDrive 64 MB
 - <http://www.ferra.ru/online/storage/15047/>

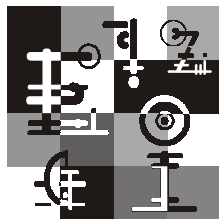
К разделу 1.7:

- ☐ Полные настройки BIOS
 - <http://www.3dnews.ru/cpu/bios/index.htm>
- ☐ Современный и несовременный BIOS компьютера
 - <http://www.istc.kiev.ua/~santana/bios/contents.html>
- ☐ Железо: BIOS: Описание настроек нового AWARD BIOS (Phoenix BIOS 4.0)
 - <http://www.oszone.net/hardware/bios/award/4/18.shtml>
- ☐ Описание настроек Setup BIOS
 - http://www.bios.ru/manuals/award4/PCI_configuration_setup.shtml
- ☐ Настройки Setup BIOS
 - <http://www.spline.ru/motherboard/bios.htm>

К разделу 1.9:

- ☐ USB и Hi-Speed USB. В чем различие?
 - http://www.concom.ru/kbEvents/kb_Events.aspx?ID=32
- ☐ Устройства USB 2.0 — скорость не гарантируется?
 - <http://www.artcomsib.ru/news/news1201.html>

Глава 2



Аппаратное обеспечение USB

Настоящий USB-программист должен отличать кабель от кобеля и понимать, что слово "хаб" — это не ругательство.

В этой главе мы рассмотрим аппаратное обеспечение USB со стороны компьютера, не касаясь периферийных элементов (микросхемы USB-приемопередатчиков мы рассмотрим в *главе 12*).

2.1. Кабели и разъемы

Спецификация USB предъявляет несколько требований к кабельному соединению:

❑ предотвращение ошибки соединения разъемов (рис. 2.1):

- соединения выходного порта одного хаба с выходным портом другого, поскольку это приводит к образованию замкнутых контуров в иерархии USB;
- соединения входного порта одного устройства с входным портом другого, поскольку в этом случае оба устройства "изымаются" из иерархии;

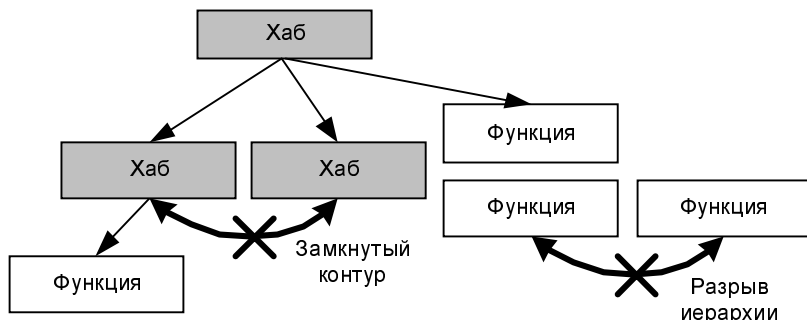


Рис. 2.1. Возможные ошибки соединения USB-устройств

- ❑ простота кабельного соединения:
 - простое подключение устройств;
 - дешевые кабели и разъемы;
 - одинаковые кабели и разъемы для всех устройств;
- ❑ возможность подключения устройств с питанием от шины и возможность подключения устройств, имеющих внешнее питание.

2.1.1. Типы кабелей

Соединительный кабель, используемый для подключения устройств с интерфейсом USB, представляет собой четырехжильный кабель в экранирующей оплетке и защитным покрытием из полихлорвинила. Цвета проводников в кабеле жестко заданы в спецификации (табл. 2.1, 2.2), а цвет самого кабеля выбирается производителем, но рекомендуемые цвета — белый, серый или черный.

Спецификация USB 2.0 определяет три возможных типа используемых кабелей:

- ❑ стандартный съемный кабель;
- ❑ высокоскоростной (полноскоростной) несъемный кабель;
- ❑ низкоскоростной несъемный кабель.

Стандартный съемный кабель служит для соединения хоста или хаба с устройством. С одной стороны он заканчивается разъемом типа "A" для подключения к хосту или хабу, а с другой — разъемом типа "B" или "mini-B" для подключения к устройству. Оба разъема маркируются логотипом USB.

Несъемный кабель заканчивается с одной стороны разъемом типа "A" (с маркировкой) для подключения к хосту или хабу, а с другой стороны жестко присоединен к устройству, т. е. имеет всего один разъем.

Высокоскоростной и полноскоростной кабель имеет импеданс $90 \pm 15\%$ Ом и полную задержку распространения сигнала 26 нс. Кабель обязательно должен иметь витую пару из сигнальных проводников (данные "минус", данные "плюс") и экранирующую оплетку. Такой кабель можно использовать и для низкоскоростного соединения.

Низкоскоростной кабель предназначен для работы на скоростях до 1,5 Мбайт/с. В связи с этим к кабелю предъявляются меньшие требования: низкоскоростной кабель не имеет витой пары из сигнальных проводников и экранирующей оплетки. Он должен иметь емкость в диапазоне 200—450 пФ и задержку на распространение сигнала не более 18 нс.

2.1.2. Длина кабеля

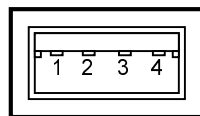
Длина соединительного кабеля определяется импедансом и задержкой распространения сигнала. В среднем длина составляет три-пять метров, но может быть и до десяти. Определяющим фактором является качество изготовления и используемый материал.

2.1.3. Разъемы

Для предотвращения ошибочных соединений USB использует USB-кабели с различными разъемами. Согласно спецификации, все устройства, работающие с шиной USB, могут использовать только три типа разъемов: "А", "В" и "mini-B". Разъемы "А" (рис. 2.2, 2.3) обозначают принадлежность к "ведущему" устройству, они используются в хостах и хабах. Их всегда можно встретить, например, на современных материнских платах персональных компьютеров. Разъемы "В" (рис. 2.4, 2.5) используют "ведомые" устройства, например, мониторы, сканеры, принтеры и т. д. Тип разъемов "mini-B" (рис. 2.6, 2.7) появился в спецификации в 2000 году с введением стандарта USB 2.0. Этот разъем позиционируется для применения в малогабаритных мобильных устройствах, например, в сотовых телефонах, когда габариты самого устройства соизмеримы с размерами разъема.



а

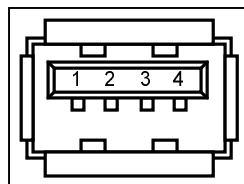


б

Рис. 2.2. Разъем типа "А"
(кабель подключения устройства к хосту)

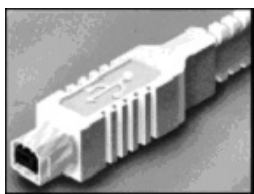


а

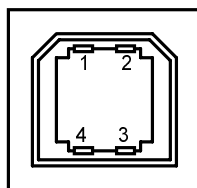


б

Рис. 2.3. Разъем типа "А" (на хосте или хабе)

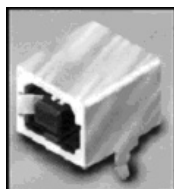


а

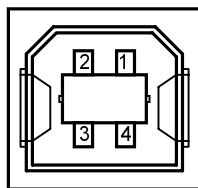


б

Рис. 2.4. Разъем типа "B"
(кабель от хоста к устройству)

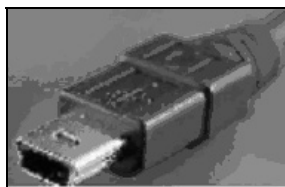


а

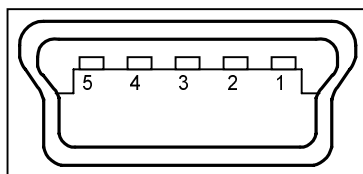


б

Рис. 2.5. Разъем типа "B"
(на устройстве)

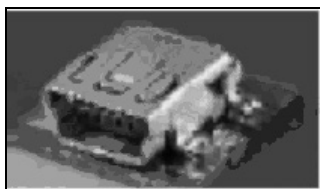


а

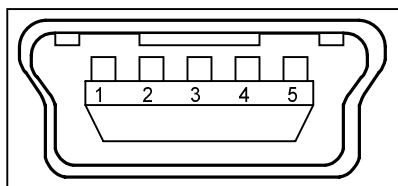


б

Рис. 2.6. Разъем типа "mini-B" (вход от хоста)



а



б

Рис. 2.7. Разъем типа "mini-B" (на устройстве)

Конструктивно разъемы задуманы так, что сначала происходит соединение шины питания, потом шины данных.

Спецификация USB определяет стандартную цветовую гамму для проводников внутри USB-кабеля (табл. 2.1, 2.2), что значительно облегчает идентификацию проводов при применении кабелей от разных производителей.

Кабель также имеет линии VBus и GND для передачи питающего напряжения 5 В к устройствам. Сечение проводников выбирается в соответствии с длиной сегмента для обеспечения гарантированного уровня сигнала и питающего напряжения.

Таблица 2.1. Цветовая гамма проводников разъема "А" и "В"

Номер контакта	Цвет	Описание
1	Красный	+5 В, питание
2	Белый	D−, данные "минус"
3	Зеленый	D+, данные "плюс"
4	Черный	GND, земля
Корпус	Медная оплетка	Экран

Таблица 2.2. Цветовая гамма проводников разъема "mini-B"

Номер контакта	Цвет	Описание
1	Красный	+5 В, питание
2	Белый	D−, данные "минус"
3	Зеленый	D+, данные "плюс"
4	Не подключен	Не подключен
5	Черный	GND, земля
Корпус	Медная оплетка	Земля

2.2. Физический интерфейс

Как говорилось выше, информационные сигналы и питающее напряжение 5 В передаются по четырехжильному кабелю. Для передачи данных по шине используется дифференциальный способ передачи сигналов D+ и D− по двум проводам. Сигналы синхронизации и данные кодируются по методу NRZI (см. разд. 2.2.1). В этой кодировке логическая "1" представлена неизменным уровнем на протяжении битового интервала, а логический 0 пред-

ставляет собой смену уровня на противоположный 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- терминируются на стороне хоста (нисходящего потока) резисторами 15 кОм¹, подключенными к земле. Скорость, используемая устройством, подключенным к конкретному порту, определяется хабом по уровням сигналов D+ и D-, смещаемых нагрузочными резисторами приемопередатчиков: устройства с низкой скоростью "подтягивают" к высокому уровню линию D- (рис. 2.8), с полной — D+ (рис. 2.9). Подключение HS-устройств определяется на этапе конфигурирования.

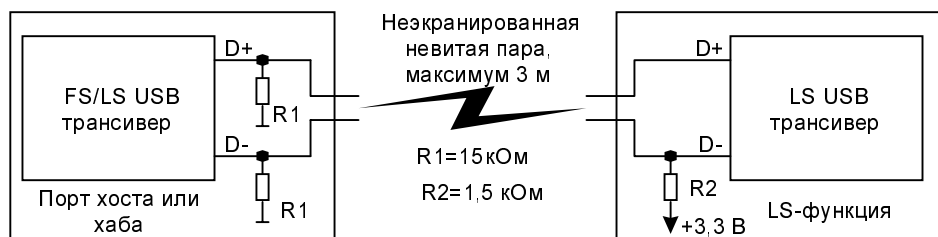


Рис. 2.8. Подключение низкоскоростного устройства

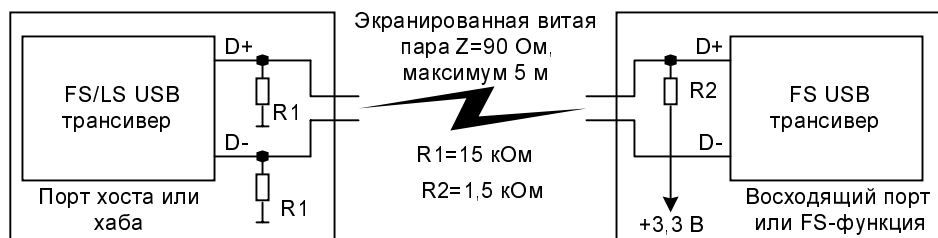


Рис. 2.9. Подключение полноскоростного устройства

Дифференциальный 0 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 200 мВ, чем на линии D-.

¹ В спецификации USB указано сопротивление 15 кОм, однако в дополнительном документе "USB Engineering Change Notice. Pull-up/pull-down resistors" разрешается установка резисторов 14,25—24,80 кОм.

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. Состояние, при котором разность потенциалов на линиях D+ и D− составляет более 200 мВ при условии, что на одной из линий потенциал выше порога срабатывания V_{SE} , называются *состоянием Diff0 или Diff1*. Состояние, при котором на обоих входах D+ и D− присутствует низкий уровень, называется *линейным нулем SE0 (Single Ended Zero)*.

Интерфейс определяет следующие состояния:

- ❑ *Data J State* и *Data K State* (или просто J и K) — состояния передаваемого бита, определяются через состояния Diff0 и Diff1;
- ❑ *Idle State* — состояние паузы на шине;
- ❑ *Resume State* — сигнал "пробуждения" для вывода устройства из "спящего" режима;
- ❑ *Start of Packet (SOP)* — начало пакета (переход из Idle State в K);
- ❑ *End of Packet (EOP)* — конец пакета;
- ❑ *Disconnect* — устройство отключено от порта;
- ❑ *Connect* — устройство подключено к порту;
- ❑ *Reset* — сброс устройства.

Состояния определяются сочетаниями дифференциальных и линейных сигналов. Для полной и низкой скоростей состояния Diff0 и Diff1 имеют противоположное назначение. В декодировании состояний Disconnect, Connect и Reset учитывается время нахождения линий (более 2,5 мс) в определенных состояниях.

2.2.1. Кодирование данных

Как уже говорилось, данные по шине USB передаются битовыми последовательностями. Все данные кодируются с помощью метода, называемого *NRZI with bit stuffing* (NRZI — Non Return to Zero Invert, метод возврата к нулю с инвертированием единиц).

Вместо кодирования логических уровней как уровней напряжения USB определяет логический 0 как изменение напряжения, а логическую 1 как неизменение напряжения. Вообще говоря, этот метод представляет собой модификацию обычного потенциального метода кодирования NRZ (Non Return to Zero, невозврат к нулю), когда для представления 1 и 0 используются потенциалы двух уровней, но в методе NRZI потенциал, используемый для кодирования текущего бита, зависит от потенциала, который использо-

вался для кодирования предыдущего бита. Если текущий бит имеет значение 1, то текущий потенциал представляет собой инверсию потенциала предыдущего бита, независимо от его значения. Если же текущий бит имеет значение 0, то текущий потенциал повторяет предыдущий.

Очевидно, что если данные содержат нули, то приемнику и передатчику достаточно легко поддерживать синхронизацию — уровень сигнала будет постоянно меняться. А вот если данные содержат длинную последовательность единиц, то уровень сигнала меняться не будет, и возможна рассинхронизация. Следовательно, для надежной передачи данных нужно исключить из кодов слишком длинные последовательности единиц. Это действие называется *стаффинг* (Bit stuffing): после каждых шести единиц автоматически добавляется 0 (рис. 2.10).

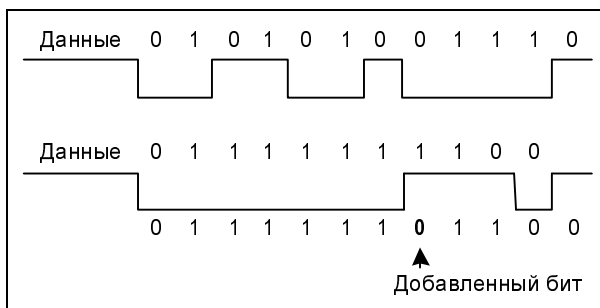


Рис. 2.10. Кодирование USB

Легко проверить существует только три возможных байта с шестью последовательными единицами: 00111111, 01111110, 11111100.

Стаффинг может увеличить число передаваемых бит до 17%, но на практике эта величина значительно меньше.

Для устройств, подключаемых к шине USB, кодирование происходит прозрачно: USB-контроллеры производят кодирование и декодирование автоматически. Конечно, кодирование данных затрудняет отладку схем с помощью осциллографа, но, с другой стороны, значительно повышает надежность передачи данных.

2.2.2. Идентификация устройств

USB-устройство должно указывать свою скорость путем подтяжки линии D+ или D− посредством сопротивления 1,5 кОм к напряжению 3,3 В (рис. 2.8, 2.9). Это подтягивающее сопротивление хост или хаб используют для обнаружения подключения устройства к своему порту. Без этого резистора ни хост, ни хаб не поймут, что к USB-шине подключено новое уст-

ройство. Некоторые микросхемы контроллеров USB-шины имеют встроенный подтягивающий резистор, который может подключаться к напряжению 3,3 В под управлением программы, другие контроллеры требуют наличия внешнего подтягивающего резистора.

Например, в микросхемах фирмы National Semiconductors USBN9603/4 имеется возможность программного управления подключением (отключением) устройства от USB-шины. Когда устройство подключается к USB-шине, сначала выполняется процедура инициализации контроллера, лишь затем программа устанавливает в регистре контроля подключение подтягивающего резистора.

Лишь со времени подключения резистора к напряжению 3,3 В хост считает, что на шине появилось новое устройство. Теперь он может инициировать сброс устройства и запрос дескрипторов, которые описывают его функциональные особенности и возможности. Если устройство работает в высокоскоростном режиме, то подтягивающий резистор затем отключается, чтобы сбалансировать линию.

Другие производители микросхем, например, Cypress Semiconductors, также используют программируемый резистор, только уже для процесса ренумерации (Re-Numeration). На этих устройствах выполнен очень интересный принцип загрузки программы в устройство. При первом включении микроконтроллер определяется в системе с идентификаторами производителя, принадлежащими компании Cypress Semiconductors, и выполняет загрузку во внутреннее ОЗУ программы пользователя. После этого происходит отключение подтягивающего резистора от шины питания (отсоединение устройства), и управление передается загруженной программе пользователя, которая, в свою очередь, снова подключает подтягивающий резистор, иницируя подключение нового устройства.

Программную часть процесса идентификации мы рассмотрим в *главе 10*.

2.3. Питание

Спецификация USB жестко оговаривает условия питания устройств, подключенных к шине и, кроме того, определяет дополнительные возможности по управлению энергопотреблением.

2.3.1. Типы питания USB-устройств

В зависимости от принципа питания можно выделить три класса USB-устройств:

- с питанием от шины (Bus Powered Devices) и малым потреблением — должны потреблять не более 100 мА;

- ❑ с питанием от шины и большим потреблением — должны потреблять не более 100 мА при включении и не более 500 мА после конфигурирования;
- ❑ с собственным источником питания (Self Powered Devices) — должны потреблять ток от шины не более 100 мА, а остальную мощность потреблять от собственного блока питания. При пропадании питания устройство должно обеспечивать потребление тока от шины, не превышающее 100 мА.

Хост обеспечивает питанием непосредственно подключенные к нему ПУ. Каждый хаб, в свою очередь, обеспечивает питание устройств, подключенных к его нисходящим портам.

Устройство указывает потребляемую мощность в дескрипторе конфигурации, который передается хосту при нумерации устройств на шине (см. *разд. 10.2.2*). Причем потребляемый от шины ток указывается с дискретностью 2 мА. Например, если устройство потребляет ток от USB-шины 100 мА, то в дескрипторе конфигурации должна фигурировать цифра 50 ($50 \times 2 = 100$ мА). Потребление тока устройством не должно превышать значения, указанного в процессе нумерации.

Потребляемая мощность, согласно спецификации USB, измеряется в блоках (unit). Один блок составляет 100 мА. Таким образом, устройства с малым потреблением используют 1 блок, а с большим потреблением — до 5 блоков.

2.3.2. Управление энергопотреблением

Все устройства должны поддерживать режим низкого энергопотребления (Suspend Mode). Ток, потребляемый устройством в таком режиме, пропорционален указанному потреблению в блоках. Для устройства с потреблением тока в 1 блок (100 мА) ток в энергосберегающем режиме не должен превышать 500 мкА (с учетом тока через подтягивающий резистор, подключенный к одной из сигнальных линий). На концентраторах обе сигнальные линии D+ и D- имеют 15 кОм резисторы, притягивающие их к земле, поэтому в целях энергосбережения суммарное сопротивление, подключенное к шине питания +3,3 В, составляет 16,5 кОм. При этом ток потребления через эту цепочку резисторов составляет 200 мкА. Также большой ток потребляют регуляторы напряжения на 3,3 В, встроенные в микросхемы контроллеров USB-шины. Для полной уверенности, что устройство не выйдет за недопустимую черту потребления тока в режиме энергосбережения, необходимо погрузить и управляющий микроконтроллер в режим "сна" (Idle Mode). При превышении лимита хаб отключит устройство от шины.

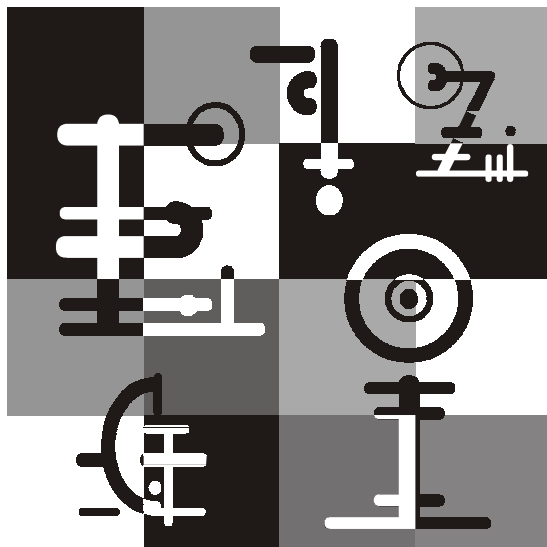
2.3.3. Вход в режим низкого энергопотребления

USB-устройство входит в режим пониженного энергопотребления, когда на шине нет активности более 3 мс. У USB-устройства есть еще 7 мс для того, чтобы погрузить себя в состояние "сна" и установить ток потребления от шины не более заявленного. Чтобы поддерживать связь с хостом в состоянии пониженного энергопотребления, устройство должно обеспечить протекание тока через подтягивающий резистор выбора скорости, наличие которого будет сигнализировать о том, что устройство все еще подключено к шине. Хост-контроллер периодически может выдавать на шину сигнал начала пакета SOF (см. разд. 3.8.2), для того, чтобы предотвратить погружение подключенных устройств в энергосберегающий режим.

Термин "*глобальное энергосбережение*" используется в тех случаях, когда полностью вся шина погружается в режим пониженного энергопотребления. Каждое устройство в отдельности тоже может быть погружено в состояние "сна", для этого хост инициирует хабу команду погружения в сон устройства, подключенного к его нисходящему порту. Этот момент называется "*селективный режим пониженного энергопотребления*". Устройство возобновит свою работу, как только на шине будет замечена любая активность. Если USB-устройство имеет функцию "*удаленная побудка*" (remote wakeup), то оно может сообщить хосту о своем желании возобновить с ним связь.

2.4. Интернет-ресурсы к этой главе

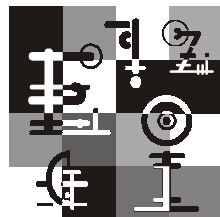
- ☐ Сетевые устройства и кабели
 - <http://postman.ru/~makarov/pclink/networkstaff.htm>
- ☐ Конвертеры интерфейсов и протоколов
 - <http://postman.ru/~makarov/pclink/converter.htm>
- ☐ Шина USB
 - <http://www.spline.ru/interfaces/usb.htm>



ЧАСТЬ II

ВНУТРЕННЯЯ ОРГАНИЗАЦИЯ USB

Глава 3



Внутренняя организация шины

Сверху дали "добро", внизу зло выругались, но пошли выполнять.

Так добро побеждает зло

Все операции по передаче данных по шине USB инициируются хостом. Периферийные устройства сами начать обмен данными не могут. Они могут только реагировать на команды хоста. В этой главе мы рассмотрим общую схему обмена данными по шине USB, а подробности каждого шага обмена рассмотрим в следующих главах.

3.1. Логические уровни обмена данными

Система USB разделяется на три *логических уровня* с определенными правилами взаимодействия. Устройство USB содержит интерфейсную, логическую и функциональную части. Хост тоже делится на три части — интерфейсную, системную и программное обеспечение (ПО). Каждая часть отвечает только за определенный круг задач. Логическое и реальное взаимодействие между ними показано на рис. 3.1.

Таким образом, операция обмена данными между прикладной программой и шиной USB выполняется путем передачи буферов памяти через следующие уровни:

- уровень клиентского ПО в хосте:
 - обычно представляется драйвером устройства USB;
 - обеспечивает взаимодействие пользователя с операционной системой с одной стороны и системным драйвером с другой;
- уровень системного обеспечения USB в хосте (USBD, Universal Serial Bus Driver):
 - управляет нумерацией устройств на шине;
 - управляет распределением пропускной способности шины и мощности питания;
 - обрабатывает запросы пользовательских драйверов;

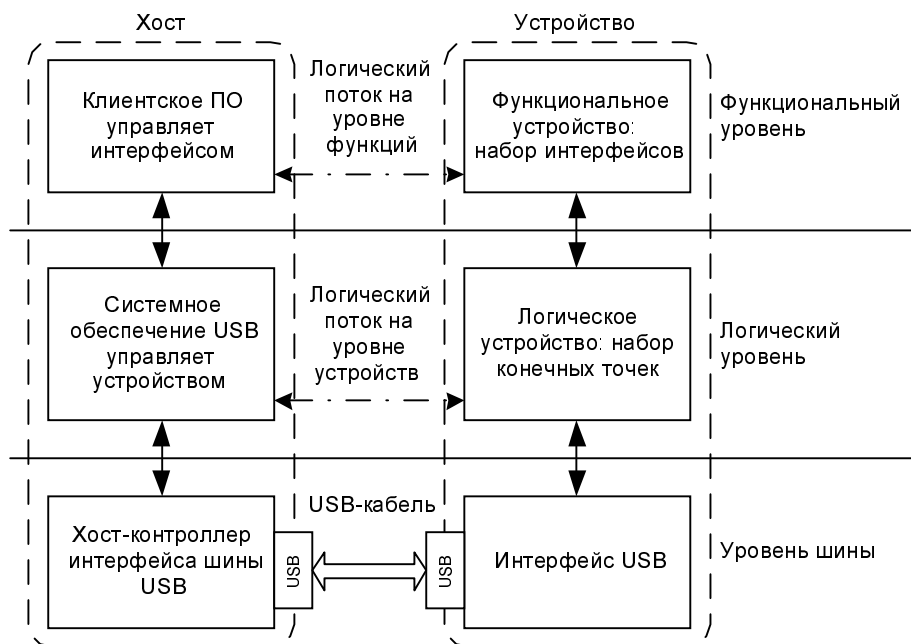


Рис. 3.1. Взаимодействие компонентов USB

□ хост-контроллер интерфейса шины USB (HCD, Host Controller Driver):

- преобразует запросы ввода/вывода в структуры данных, по которым хост-контроллер выполняет физические транзакции;
- работает с регистрами хост-контроллера.

Рассмотрим каждый уровень более подробно.

3.1.1. Уровень клиентского ПО

Уровень клиентского программного обеспечения определяет тип передачи данных, необходимый для выполнения затребованной прикладной программой операции. После определения типа передачи данных этот уровень передает системному уровню следующее:

- буфер памяти, называемый клиентским буфером;
- *пакет запроса на в/в* (IRP, Input/output Request Packet), указывающий тип необходимой операции.

IRP содержит только сведения о запросе (адрес и длина буфера в оперативной памяти). Непосредственно обработкой запроса занимается системный драйвер USB. Запросы клиентского ПО мы рассмотрим в *главе 4*.

3.1.2. Уровень системного драйвера USB

Уровень системного драйвера USB необходим для управления ресурсами USB. Он отвечает за выполнение следующих действий:

- ☐ распределение полосы пропускания шины USB;
- ☐ назначение логических адресов устройств каждому физическому USB-устройству;
- ☐ планирование транзакций.

Распределение полосы пропускания

До установления каналов передач из хоста в конечную точку какого-либо устройства системное программное обеспечение USB должно сначала определить, может ли шина USB обеспечить минимальную требуемую полосу пропускания для данной точки. В каждом устройстве есть специальная таблица, содержащая дескрипторы конечных точек устройства, в которых хранится значение минимально допустимой полосы пропускания для соответствующей конечной точки. В процессе определения устройств в фазе начальной инициализации системное программное обеспечение читает эти дескрипторы и определяет суммарную полосу пропускания для данного устройства. Хранящееся в дескрипторе значение определяет, какая доля пропускной способности шины необходима для передачи информации в (или от) конечную точку. При этом, однако, не учитываются никакие накладные расходы. Определяя общую потребность для поддержки канала к каждой конечной точке, системное обеспечение USB учитывает следующее:

- ☐ число байтов данных;
- ☐ тип передачи данных;
- ☐ время восстановления хоста;
- ☐ время заполнения битами;
- ☐ уровень вложенности топологии.

Назначение логических адресов

Логическое устройство USB представляет собой набор независимых конечных точек (см. *разд. 3.6*), с которыми клиентское ПО обменивается информацией. Каждому логическому устройству USB (как функции, так и хабу) назначается свой адрес (1—127), уникальный на данной шине USB. Каждая конечная точка логического устройства идентифицируется своим номером (0—15) и направлением передачи (IN — передача к хосту, OUT — от хоста).

Планирование транзакций

Транзакция на шине USB — это последовательность обмена пакетами между хостом и ПУ, в ходе которой может быть передан или принят один пакет данных. Когда клиентское ПО передает IRP уровню системного драйвера, USB-драйвер преобразует их в одну или несколько транзакций шины и затем передает получившийся перечень транзакций драйверу контроллера хоста.

Архитектура системного драйвера USB

Системный драйвер USB состоит из *драйвера USB* и *драйвера хост-контроллера*.

Когда клиентский уровень передает IRP уровню системного обеспечения USB, USB-драйвер преобразует их в одну или несколько транзакций шины и затем передает получившийся перечень транзакций драйверу контроллера хоста.

Драйвер контроллера хоста принимает от системного драйвера шины перечень транзакций и выполняет следующие действия:

- ❑ планирует исполнение полученных транзакций, добавляя их к списку транзакций;
- ❑ извлекает из списка очередную транзакцию и передает ее уровню хост-контроллера интерфейса шины USB;
- ❑ отслеживает состояние каждой транзакции вплоть до ее завершения.

При выполнении всех связанных с командным пакетом транзакций системный уровень уведомляет об этом клиентский уровень.

3.1.3. Уровень хост-контроллера интерфейса

Уровень хост-контроллера интерфейса шины USB получает отдельные транзакции от драйвера контроллера хоста (в составе уровня системного обеспечения USB) и преобразует их в соответствующую последовательность операций шины. В результате этого USB-пакеты передаются вдоль всей физической иерархии хабов (на рис. 3.1 мы изобразили последовательность хабов как одну логическую линию, но физически это может быть как один USB-кабель, так и последовательность хабов) до периферийного USB-устройства (правая часть рис. 3.1).

3.1.4. Уровень шины периферийного устройства

Нижний уровень периферийного USB-устройства называется уровнем интерфейса шины USB. Он взаимодействует с интерфейсным уровнем шины USB на стороне хоста и передает пакеты данных от хоста периферийному

устройству в формате, определяемом спецификацией USB. Затем он передает пакеты вверх — уровню логического USB-устройства.

3.1.5. Уровень логического USB-устройства

Средний уровень периферийного USB-устройства называется уровнем логического USB-устройства. Каждое логическое USB-устройство представляется набором своих конечных точек, с которыми может взаимодействовать системный уровень USB-хоста. Эти точки являются источниками и приемниками всех коммуникационных потоков между хостом и периферийными USB-устройствами.

3.1.6. Функциональный уровень USB-устройства

Самый верхний уровень периферийного USB-устройства называется функциональным уровнем. Этот уровень соответствует уровню клиентского обеспечения хоста. С точки зрения клиентского уровня, нижележащие уровни нужны для организации между ним и конечными точками прямых "каналов данных" (см. *разд. 3.7*), которые идут вплоть до функционального уровня периферийного устройства. А с точки зрения нашей схемы функциональный уровень выполняет следующие действия:

- ❑ получает данные, посылаемые клиентским уровнем хоста из конечных точек каналов данных нижележащего уровня логического USB-устройства;
- ❑ посылает данные клиентскому уровню хоста, направляя их в конечные точки каналов данных нижележащего уровня логического USB-устройства.

3.2. Передача данных по уровням

Логически передача данных между конечной точкой и ПО производится с помощью выделения канала и обмена данными по этому каналу, а с точки зрения представленных уровней, передача данных выглядит следующим образом (рис. 3.2).

- ❑ Клиентское ПО посылает IPR-запросы уровню USBД.
- ❑ Драйвер USBД разбивает запросы на транзакции по следующим правилам:
 - выполнение запроса считается законченным, когда успешно завершены все транзакции, его составляющие;
 - все подробности отработки транзакций (такие как ожидание готовности, повтор транзакции при ошибке, неготовность приемника и т. д.) до клиентского ПО не доводятся;

- ПО может только запустить запрос и ожидать или выполнения запроса или выхода по тайм-ауту;
 - устройство может сигнализировать о серьезных ошибках (см. *разд. 3.8*), что приводит к аварийному завершению запроса, о чем уведомляется источник запроса.
- ❑ Драйвер контроллера хоста принимает от системного драйвера шины перечень транзакций и выполняет следующие действия:
- планирует исполнение полученных транзакций, добавляя их к списку транзакций;
 - извлекает из списка очередную транзакцию и передает ее уровню хост-контроллера интерфейса шины USB;
 - отслеживает состояние каждой транзакции вплоть до ее завершения.
- ❑ Хост-контроллер интерфейса шины USB формирует кадры;
- ❑ Кадры передаются последовательной передачей бит по методу NRZI (см. *разд. 2.2.1*).

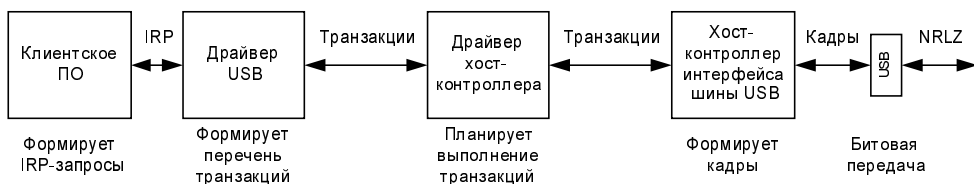


Рис. 3.2. Уровни передачи данных

Таким образом, можно сформировать следующую *упрощенную* схему (рис. 3.3):

- ❑ каждый кадр состоит из наиболее приоритетных посылок, состав которых формирует драйвер хоста;
- ❑ каждая передача состоит из одной или нескольких транзакций (см. *разд. 3.10*);
- ❑ каждая транзакция состоит из пакетов;
- ❑ каждый пакет состоит из идентификатора пакета, данных (если они есть) и контрольной суммы.

В следующих разделах мы рассмотрим все составляющие передачи более подробно.

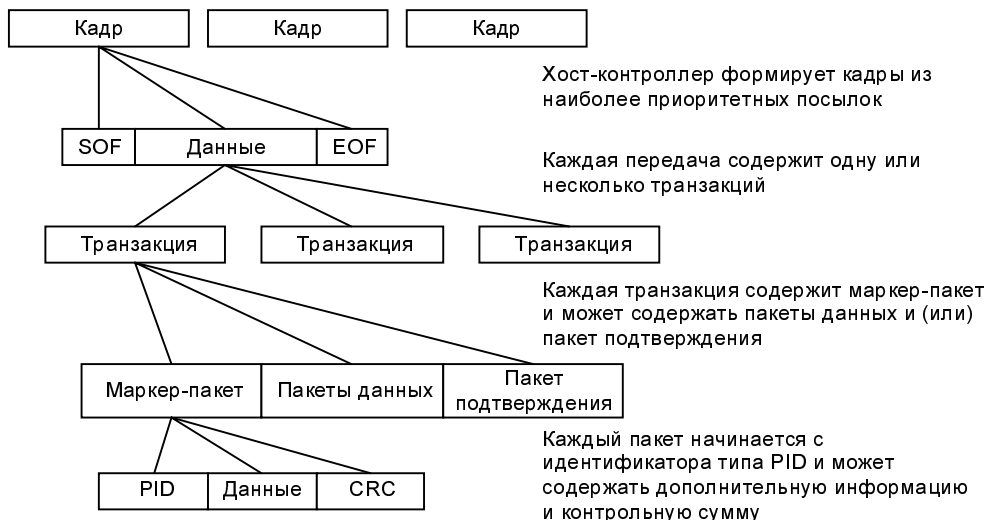


Рис. 3.3. Общая схема составляющих USB-протокола

3.3. Типы передач данных

Спецификация шины определяет четыре различных типа передачи (transfer type) данных для конечных точек (табл. 3.1):

- ❑ *управляющие передачи* (Control Transfers) — используются хостом для конфигурирования устройства во время подключения, для управления устройством и получения статусной информации в процессе работы. Протокол обеспечивает гарантированную доставку таких посылок. Длина поля данных управляющей посылки не может превышать 64 байт на полной скорости и 8 байт на низкой. Для таких посылок хост гарантированно выделяет 10% полосы пропускания;
- ❑ *передачи массивов данных* (Bulk Data Transfers) — применяются при необходимости обеспечения гарантированной доставки данных от хоста к функции или от функции к хосту, но время доставки не ограничено. Такая передача занимает всю доступную полосу пропускания шины. Пакеты имеют поле данных размером 8, 16, 32 или 64 байт. Приоритет у таких передач самый низкий, они могут приостанавливаться при большой загрузке шины. Допускаются только на полной скорости передачи. Такие посылки используются, например, принтерами или сканерами;
- ❑ *передачи по прерываниям* (Interrupt Transfers) — используются в том случае, когда требуется передавать одиночные пакеты данных небольшого размера. Каждый пакет требуется передать за ограниченное время. Операции передачи носят спонтанный характер и должны обслуживаться не

медленнее, чем того требует устройство. Поле данных может содержать до 64 байт на полной скорости и до 8 байт на низкой. Предел времени обслуживания устанавливается в диапазоне 1—255 мс для полной скорости и 10—255 мс — для низкой. Такие передачи используются в устройствах ввода, таких как мышь и клавиатура;

- ❑ *изохронные передачи* (Isochronous Transfers) — применяются для обмена данными в "реальном времени", когда на каждом временном интервале требуется передавать строго определенное количество данных, но доставка информации не гарантирована (передача данных ведется без повторения при сбоях, допускается потеря пакетов). Такие передачи занимают предварительно согласованную часть пропускной способности шины и имеют заданную задержку доставки. Изохронные передачи обычно используются в мультимедийных устройствах для передачи аудио- и видеоданных, например, цифровая передача голоса. Изохронные передачи разделяются по способу синхронизации конечных точек — источников или получателей данных — с системой: различают асинхронный, синхронный и адаптивный классы устройств, каждому из которых соответствует свой тип канала USB.

Таблица 3.1. Типы передач по шине USB

Тип передачи	Направление	Частота запуска	Гарантия доставки
Управляющие послышки	Двунаправленные	Не гарантируется	Есть
Изохронные (только для высокоскоростных устройств)	Однонаправленные	Каждые 1 мс	Нет
Передача по прерываниям	Только ввод	Определяется частотой опроса	Есть
Передача массивов данных	Двунаправленные	Не гарантируется	Есть

Все операции по передаче данных инициируются только хостом независимо от того, принимает ли он данные или пересылает в периферийное устройство. Все невыполненные операции хранятся в виде четырех списков по типам передач. Списки постоянно обновляются новыми запросами.

Планирование операций по передаче информации в соответствии с упорядоченными в виде списков запросами выполняется хостом с интервалом один кадр (см. разд. 3.5). Обслуживание запросов выполняется в соответствии со следующими правилами:

- ❑ наивысший приоритет имеют изохронные передачи;
- ❑ после отработки всех изохронных передач система переходит к обслуживанию передач прерываний;

- ❑ в последнюю очередь обслуживаются запросы на передачу массивов данных;
- ❑ по истечении 90% указанного интервала хост автоматически переходит к обслуживанию запросов на передачу управляющих команд независимо от того, успел ли он полностью обслужить другие три списка или нет.

Выполнение этих правил гарантирует, что управляющим передачам всегда будет выделено не менее 10% пропускной способности шины USB. Если передача всех управляющих пакетов будет завершена до истечения выделенной для них доли интервала планирования, то оставшееся время будет использовано хостом для передач массивов данных. Таким образом:

- ❑ изохронные передачи гарантированно получают 90% пропускной способности шины;
- ❑ передачи прерываний занимают оставшуюся после изохронных операций часть этой 90-процентной доли.
- ❑ под передачу данных большого объема выделяется все время, оставшееся после изохронных передач и передач прерываний (по-прежнему в рамках 90% доли пропускной способности);
- ❑ управляющим передачам гарантируется 10% пропускной способности шины;
- ❑ если передача всех управляющих пакетов будет завершена до завершения выделенного для них 10-процентного интервала, то оставшееся время будет использовано для передач данных большого объема.

3.4. Синхронизация при изохронной передаче

Изохронная передача данных связана с синхронизацией устройств, объединенных в единую систему [2]. Возьмем пример использования USB, когда к компьютеру подключен микрофон USB (источник данных) и колонки USB (приемник данных), и эти аудиоустройства связаны между собой через программный микшер (клиентское ПО). Каждый из компонентов может иметь свои "понятия" о времени и синхронизации: микрофон, к примеру, может иметь частоту выборки 8 кГц и разрядность данных 1 байт (поток 64 Кбит/с), стереоколонки — 44,1 кГц и разрядность 2×2 байта (176,4 Кбит/с), а микшер может работать на частоте выборок 32 кГц. Микшер в этой системе является связующим звеном, и его источник синхронизации будем считать главным (master clock). Программный микшер обрабатывает данные пакетами, сеансы обработки выполняются регулярно с определенным периодом обслуживания (скажем, в 20 мс — частота 50 Гц). В микшере должны быть модули согласования частот выборки, которые объединяют несколько выборок в одну, если входная частота выше выходной, или "со-

чиняют" (интерполируют) новые промежуточные выборки, если выходная частота выше. В системе с USB приходится иметь дело со следующими частотами:

- ❑ F_S (sample clock, частота данных) — частота выборки (sample rate) для источников (source) и приемников (sink) данных;
- ❑ F_{SOF} (bus clock, частота шины) — частота кадров (1 кГц) для полной скорости и микрокадров (8 кГц). С этой частотой все устройства USB "видят" маркеры начала кадров SOF;
- ❑ F_X (service clock, частота обслуживания) — частота, с которой клиентское ПО обращается к драйверам USB для передачи и приема изохронных данных.

В системе без общего источника синхронизации между парами синхросигналов возможны отклонения следующих типов:

- ❑ дрейф (clock drift) — отклонения формально одинаковых частот от номиналов (не бывает двух абсолютно одинаковых генераторов);
- ❑ дрожание (clock jitter) — колебание частот относительно номинала;
- ❑ фазовый сдвиг (clock-to-clock phase differences), если сигналы не связаны системой фазовой автоподстройки ФАПЧ (PLL).

В цифровой системе передачи данных эти отклонения выливаются в то, что у источника или приемника данных может образовываться излишек или недостаток данных, колеблющийся или накапливающийся во времени. Согласование скоростей выполняется с использованием механизма прямого объявления скорости (feed forward) или механизма обратной связи (feedback). Какой из механизмов используется, зависит от типа синхронизации, поддерживаемого изохронной конечной точкой данного устройства.

В USB по способу синхронизации конечных точек (источников или получателей данных) с системой различают асинхронный, синхронный и адаптивный классы устройств (точнее, конечных точек), каждому из которых соответствует свой тип канала USB. Тип синхронизации задается битами 3 и 2 байта атрибутов в дескрипторе конечной точки (см. разд. 4.1.3).

Асинхронные устройства не имеют возможности согласования своей частоты выборок с метками SOF или иными частотами системы USB. Частота передачи данных у них фиксированная или программируемая. Число байт данных, принимаемых за каждый кадр USB, не является постоянным. *Асинхронный источник данных* неявно объявляет свою скорость передачи числом выборок, передаваемых им за один кадр — клиентское ПО будет обрабатывать столько данных, сколько реально поступило. *Асинхронный приемник данных* должен обеспечивать *явную обратную связь* для адаптивного драйвера клиентского ПО, чтобы согласовать темп выдачи потока (см. далее). Примерами асинхронного устройства-источника может быть CD-плеер с синхро-

низацией от кварцевого генератора или приемник спутникового телевидения. Пример приемника — дешевые колонки, работающие от внутреннего источника синхронизации.

Синхронные устройства имеют внутренний генератор, синхронизируемый с метками SOF (системная частота 1 или 8 кГц); на высокой частоте передачи более точную синхронизацию обеспечивает связь с микрокадрами. Источники и приемники за каждый кадр генерируют (потребляют) одинаковое количество байт данных, которое устанавливается на этапе программирования каналов. Примером синхронного источника может быть цифровой микрофон с частотой выборки, синтезируемой по SOF. Синтезатор частоты должен учитывать возможность пропадания 1—2 маркеров (из-за возможных ошибок передачи), поддерживая постоянную частоту. Эти точки используют *неявную обратную связь*, подстраиваясь под частоту шины. С программной точки зрения организация каналов с такими устройствами проще всего.

Адаптивные устройства имеют возможность подстройки своей внутренней частоты под требуемый поток данных (в определенных границах). Адаптивный источник позволяет менять скорость под управлением приемника, обеспечивающего явную обратную связь. Для адаптивного приемника информацию о частоте задает входной поток данных. Он определяет мгновенное значение частоты по количеству данных, принятых за некоторый интервал усреднения. Таким образом, осуществляется неявное прямое объявление частоты. Примером адаптивного источника является CD-плеер со встроенным конвертером частоты SRC (sample rate converter), а приемника — высококачественные колонки или наушники USB.

Контроллер USB позволяет подстраивать частоту кадров, но, естественно, под частоту внутренней синхронизации только одного устройства. Подстройка осуществляется через механизм обратной связи, который позволяет изменять период кадра в пределах ± 1 битового интервала. *Обратная связь*, позволяющая согласовать значения частот устройств с частотой шины, может быть явной (explicit feedback) или неявной (implicit feedback).

Асинхронный приемник должен явным образом сообщать хост-контроллеру желаемую частоту передачи данных относительно частоты кадров. Это позволит хост-контроллеру постоянно корректировать число передаваемых байтов за каждый кадр, не допуская переполнения или опустошения буфера устройства-приемника. Отношение

$$F_f = F_s / F_{\text{SOF}}$$

показывает число байтов, передаваемых за один кадр. Это число может оказаться не целым числом, тогда его целая часть определяет постоянный объем передач (размер поля данных) с данной конечной точкой в каждом кадре, а дробная часть — это накапливающийся остаток, который будет вызывать периодическое увеличение объема передач в некоторых кадрах. Значение F_f в данном случае должен вычислять приемник на интервале ус-

реднения не менее 1 с. Это отношение может меняться во времени (хотя бы из-за погрешности округления), так что хост должен периодически запрашивать его у устройства, что и будет являться *явной обратной связью* (explicit feedback data).

Максимальный размер передачи в кадре и частота кадров равны 1023 байта и 1 кГц в режиме FS и 3072 байта и 8 кГц в режиме HS, поэтому для передачи данных обратной связи требуется:

- в FS-режиме: по 10 бит для целой и дробной частей;
- в HS-режиме: 12 битов для целой части и 13 битов для дробной части.

Данные явной обратной связи устройства берутся с конечной точки, имеющей тот же номер, что и у точки, используемой для данных. Эта точка тоже изохронная, в ее дескрипторе биты [5:4]=01 байта атрибутов указывают на использование для обратной связи (*см. разд. 4.1.3*). В дескрипторе задается и интервал опроса (поле `bInterval`), с которым хост должен запрашивать данные обратной связи, чтобы своевременно отследить изменения. Это позволяет хост-контроллеру постоянно корректировать число передаваемых байтов за каждый кадр, не допуская переполнения или опустошения буфера устройства-приемника. Если с прошлого опроса изменений нет, точка может ответить на опрос пакетом данных нулевой длины.

Аналогично *адаптивный источник* должен воспринимать информацию обратной связи от хоста, чтобы за каждый кадр генерировать ровно столько данных, сколько требуется хост-контроллеру.

Для точек, требующих обратной связи, в некоторых случаях можно избежать выделения в устройстве специальной точки обратной связи, используя неявную обратную связь (*implicit feedback*). Это возможно, если в устройстве есть группа функционально-связанных изохронных точек, работающих от общего генератора синхронизации, и среди них есть точка с направлением, противоположным точке, требующей явной обратной связи. Если требуется обратная связь для асинхронного приемника, то информация неявной обратной связи берется из скорости передачи данных синхронизированного с ним передатчика. Аналогично для адаптивного источника информация неявной обратной связи берется из скорости синхронизированного с ним приемника. Конечная точка данных, которую можно использовать как источник неявной обратной связи, в байте атрибутов `bmAttributes` имеет значение бит [5:4] = 10 (*см. разд. 4.1.3*). Связи по синхронизации в группе устанавливаются на основе номеров точек. Для того чтобы найти источник неявной обратной связи для какой-либо точки, ищется изохронная точка противоположного направления с таким же или меньшим номером, имеющая в байте атрибутов биты [5:4] = 10.

Шина USB позволяет устройству и хосту расставлять метки времени в непрерывном потоке изохронных передач для любой конечной точки. Для

этого хост посылает устройству специальный управляющий запрос SYNC_FRAME (см. разд. 4.1.2), в котором указывает номер кадра (в обозримом будущем) и номер конечной точки, к которой относится данная метка времени. Устройства и хост имеют общее представление о времени по номеру кадра, передаваемому в маркере SOF. Для HS-устройств подразумевается синхронизация по нулевому микрокадру указанного кадра. Метка времени может использоваться, например, для указания момента начала изохронной передачи (хост-контроллеру ОНС в дескрипторе изохронной передачи указывается номер стартового кадра; для УНС драйвер сам размещает дескрипторы изохронных транзакций в списке кадров). Таким образом, устройство может заранее подготовиться к началу изохронного обмена.

Хост-контроллер USB имеет возможность подстройки частоты кадров. Например, в УНС имеется регистр SOF_Modify, через который ПО может изменять коэффициент деления частоты 12 МГц для получения частоты кадров 1 кГц в пределах $\pm 0,5\%$. Естественно, хост-контроллер может подстроиться под частоту внутренней синхронизации только одного устройства.

3.5. Кадры

Любой обмен по шине USB инициируется хост-контроллером. Он организует обмены с устройствами согласно своему плану распределения ресурсов.

Контроллер циклически (с периодом $1,0 \pm 0,0005$ мс) формирует *кадры* (frames), в которые укладываются все запланированные передачи (рис. 3.4). Каждый кадр начинается с посылки пакета-маркера SOF (Start Of Frame, начало кадра, см. разд. 3.8.1), который является синхронизирующим сигналом для всех устройств, включая хабы. В конце каждого кадра выделяется интервал времени EOF (End Of Frame, конец кадра), на время которого хабы запрещают передачу по направлению к контроллеру. Если хаб обнаружит, что с какого-то порта в это время ведется передача данных, этот порт отключается.

В режиме высокоскоростной передачи (см. разд. 3.3) пакеты SOF передаются в начале каждого *микрокадра* (период $125 \pm 0,0625$ мкс).

Хост планирует загрузку кадров так, чтобы в них всегда находилось место для наиболее приоритетных передач, а свободное место кадров заполняется низкоприоритетными передачами больших объемов данных. Спецификация USB позволяет занимать под периодические транзакции (изохронные и прерывания) до 90% пропускной способности шины.

Каждый кадр имеет свой номер. Хост-контроллер оперирует 32-битным счетчиком, но в маркере SOF передает только младшие 11 бит (см. разд. 3.8.1). Номер кадра циклически увеличивается во время EOF.

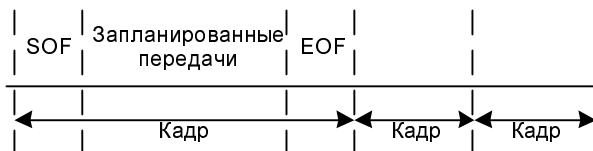


Рис. 3.4. Поток кадров USB

Для изохронной передачи (см. разд. 3.3) важна синхронизация устройств и контроллера. Есть три варианта синхронизации:

- ☐ синхронизация внутреннего генератора устройства с маркерами SOF;
- ☐ подстройка частоты кадров под частоту устройства;
- ☐ согласование скорости передачи (приема) устройства с частотой кадров.

В каждом кадре может быть выполнено несколько транзакций, их допустимое число зависит от скорости, длины поля данных каждой из них, а также от задержек, вносимых кабелями, хабами и устройствами. Все транзакции кадров должны быть завершены до момента времени EOF. Частота генерации кадров может немного варьироваться с помощью специального регистра хост-контроллера, что позволяет подстраивать частоту для изохронных передач. Подстройка частоты кадров контроллера возможна под частоту внутренней синхронизации только одного устройства.

3.6. Конечные точки

Конечная точка (Endpoint) — это часть USB-устройства, которая имеет уникальный идентификатор и является получателем или отправителем информации, передаваемой по шине USB. Проще говоря, это буфер, сохраняющий несколько байт. Обычно это блок данных в памяти или регистр микроконтроллера. Данные, хранящиеся в конечной точке, могут быть либо принятыми данными, либо данными, ожидающими передачи. Хост также имеет буфер для приема и передачи данных, но хост не имеет конечных точек.

Конечная точка имеет следующие основные параметры:

- ☐ частота доступа к шине;
- ☐ допустимая величина задержки обслуживания;
- ☐ требуемая ширина полосы пропускания канала;
- ☐ номер конечной точки;
- ☐ способ обработки ошибок;
- ☐ максимальный размер пакета, который конечная точка может принимать или отправлять;

- ❑ используемый конечной точкой тип посылок;
- ❑ направление передачи данных.

Любое USB-устройство имеет конечную точку с нулевым номером (Endpoint Zero). Эта точка позволяет хосту опрашивать устройство с целью определения его типа и параметров, выполнять инициализацию и конфигурирование устройства.

Кроме нулевой точки, устройства, обычно, имеют дополнительные конечные точки, которые используются для обмена данными с хостом. Дополнительные точки могут работать либо только на прием данных от хоста (входные точки, IN), либо только на передачу данных хосту (выходные точки, OUT). Число дополнительных конечных точек устройств определяется режимом передачи (см. разд. 1.6.5). Для низкоскоростных устройств допускается наличие одной или двух дополнительных конечных точек, а для высокоскоростных — до 15 входных и 15 выходных дополнительных точек.

Нулевая точка устройства доступна после того, как устройство подключено к шине, включено и получило сигнал сброса по шине (bus reset). Все остальные конечные точки после включения питания или сброса находятся в неопределенном состоянии и недоступны для работы до тех пор, пока хост не выполнит процедуру конфигурирования устройства.

3.7. Каналы

Канал (pipe¹) — это логическое соединение между конечной точкой устройства и ПО хоста (рис. 3.5).

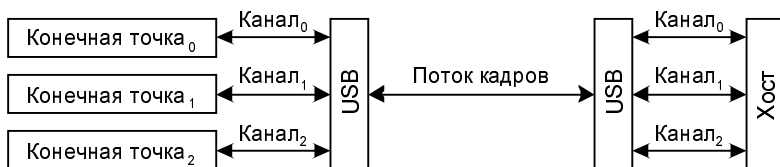


Рис. 3.5. Каналы USB

Существует две модели каналов:

- ❑ **поточковый канал** (или просто *поток*, streaming pipe) — это канал для передачи данных, структура которых определяется клиентским ПО. Потoki используются для передачи массивов данных, передачи данных по пре-

¹ Слово "pipe" дословно означает "труба", но мы будем пользоваться более подходящим и благозвучным переводом "канал".

рываниям и изохронной передаче данных. Поток всегда однонаправленный. Один и тот же номер конечной точки может использоваться для двух разных потоковых каналов — ввода и вывода. Передачи данных в потоковых каналах подчиняются следующим *правилам синхронизации*:

- запросы клиентских драйверов для разных каналов, поставленные в определенном порядке друг относительно друга, могут выполняться в другом порядке;
- запросы для одного канала будут исполняться строго в порядке их поступления;
- если во время выполнения какого-либо запроса происходит серьезная ошибка (STALL), поток останавливается;

□ **канал сообщений** (message pipe или control pipe) — это канал для передачи данных, структура которых определяется спецификацией USB. Каналы этого типа двунаправленные и применяются для передачи управляющих посылок. Каналы сообщений строго синхронизированы — спецификация USB запрещает одновременную обработку нескольких запросов: нельзя начинать передачу нового сообщения, пока не завершена обработка предыдущего. В случае возникновения ошибки передача сообщения может быть прервана хостом, после чего хост может начать передачу нового сообщения.

Основными характеристиками каналов являются:

- полоса пропускания канала;
- используемый каналом тип передачи данных;
- характеристики, соответствующие конечной точке: направление передачи данных и максимальный размер пакета.

Полоса пропускания шины делится между всеми установленными каналами. Выделенная полоса закрепляется за каналом, и если установление нового канала требует такой полосы, которая не списывается в уже существующее распределение, запрос на выделение канала отвергается. Архитектура USB предусматривает внутреннюю буферизацию всех устройств, причем, чем большей полосы пропускания требует устройство, тем больше должен быть его буфер. Шина USB должна обеспечивать обмен с такой скоростью, чтобы задержка данных в устройстве, вызванная буферизацией, не превышала нескольких миллисекунд.

Канал сообщений, связанный с нулевой конечной точкой, называется *Основным каналом сообщений* (Default Control Pipe или Control Pipe 0). Владелец этого канала является USB-D, и он используется для конфигурирования устройства. Основной канал сообщений поддерживает только управляющие передачи. Остальные каналы (они называются *клиентскими каналами*, Client Pipe) создаются в процессе конфигурирования устройства. Их владельцами

являются драйверы устройств. По клиентским каналам могут передаваться как потоки, так и сообщения с помощью любых типов передач.

Набор клиентских каналов, с которыми работает драйвер устройства, называется *интерфейсом устройства* или *связкой клиентских каналов* (pipe's bundle).

3.8. Пакеты

Информация по каналу передается в виде *пакетов* (Packet, рис. 3.6). Каждый пакет начинается с *поля синхронизации* SYNC (SYNChronization), за которым следует *идентификатор пакета* PID (Packet IDentifier), значения которого приведены в табл. 3.2. Поле Check представляет собой побитовую инверсию PID.

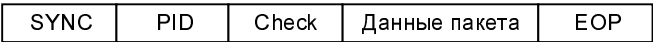


Рис. 3.6. Структура пакета

Таблица 3.2. Список кодов PID

Обоз- начение	Код PID	Источ- ник	Описание
Идентификаторы маркер-пакетов (Token Packet)			
OUT	0001b	Хост	Маркер транзакции вывода, передает адрес и номер конечной точки при передаче от хоста к функции
IN	1001b		Маркер транзакции ввода, передает адрес и номер конечной точки при передаче от функции к хосту
SOF	0101b		Маркер начала кадра, содержит номер кадра
SETUP	1101b		Маркер транзакции управления: передает адрес и номер конечной точки при передаче команды от хоста к функции
Идентификаторы пакетов данных (Data Packet)			
Data0	0011b	Хост, ус тр-во	Пакеты данных с четным и нечетным PID, чередуются для точной идентификации подтверждений
Data1	1011b		

Таблица 3.2 (окончание)

Обоз- начение	Код PID	Источ- ник	Описание
Идентификаторы пакетов данных (Data Packet)			
Data2	0111b		Дополнительные типы пакетов данных, используемые в транзакциях с широкополосными изохронными точками (в USB 2.0 для HS)
MData	1111b		
Идентификаторы пакетов-подтверждений (Handshake)			
ACK	0010b	Хост, устр-во	Подтверждение безошибочного приема пакета
NAK	1010b	Устр-во	Приемник не сумел принять или передатчик не сумел передать данные. Может использоваться для управления потоком данных ("ответ на запрос не готов"). В транзакциях прерываний является признаком отсутствия необслуживаемых прерываний
STALL	1110b	Устр-во	Произошел сбой в конечной точке или запрос не поддерживается, требуется вмешательство хоста
NYET	0110b	Устр-во	Подтверждение безошибочного приема, но указание на отсутствие места для приема следующего пакета максимального размера (USB 2.0)
Идентификаторы специальных пакетов (Special Packet)			
PRE	1100b	Хост	Специальный маркер, сообщающий, что следующий пакет будет передаваться в режиме LS (разрешает трансляцию данных на низкоскоростной порт хаба)
ERR		Устр-во, хаб	Сигнализация ошибки в расщепленной транзакции (USB 2.0)
SPLIT (SS/CS)	1000b	Хост	Маркер расщепленной транзакции (USB 2.0). В зависимости от назначения обозначается как SS (маркер запуска) и CS (маркер завершения), назначение определяется битом SC в теле маркера
PING	0100b	Хост	Пробный маркер высокоскоростного управления потоком (USB 2.0)
RESERV	0000b		Зарезервированный PID

Из таблицы видно, что два младших бита идентификатора определяют группу, к которой он принадлежит:

- ☐ 00 — специальный пакет (Special);
- ☐ 01 — маркер (Token);

- ❑ 10 — подтверждение (Handshake);
- ❑ 11 — пакет данных (Data).

Структура данных пакета зависит от группы, к которой он относится.

3.8.1. Формат пакетов-маркеров IN, OUT, SETUP и PING

Поле данных пакетов типа IN, OUT, SETUP и PING содержит следующие поля (полный формат пакета показан на рис. 3.7):

- ❑ [7] адрес функции;
- ❑ [4] адрес конечной точки;
- ❑ [5] циклический контрольный код.

Маркер транзакции отмечает начало очередной транзакции на шине USB и позволяет адресовать до 127 функций USB (нулевой адрес используется для конфигурирования) и по 16 конечных точек в каждой функции. Поле CRC вычисляется по полям Func и EndP.

SYNC	PID[4]	Check[4]	Func[7]	EndP[4]	CRC[5]	EOP
------	--------	----------	---------	---------	--------	-----

Рис. 3.7. Формат пакета типов IN, OUT, SETUP и PING

3.8.2. Формат пакета SOF

Поле данных пакета SOF содержит (полный формат пакета показан на рис. 3.8):

- ❑ [11] номер кадра
- ❑ [5] циклический контрольный код.

Пакет SOF используется для отметки начала кадра (см. разд. 3.5). Хотя хост-контроллер оперирует 32-битным счетчиком кадров, в маркере SOF передаются только младшие 11 бит. Контрольная сумма вычисляется по 11 битам поля Frame.

SYNC	PID[4]	Check[4]	Frame[11]	CRC[5]	EOP
------	--------	----------	-----------	--------	-----

Рис. 3.8. Формат пакета типа SOF

3.8.3. Формат пакета данных

В поле данных пакетов типа Data0, Data1, Data2 и MData может содержаться от 0 до 1023 байт данных, за которыми следует 16-разрядный циклический контрольный код (см. *разд. 3.9*), вычисленный по полю Data. Пакет данных всегда должен посылать целое число байт. Для режима LS максимальный размер пакета равен 8 байтам, для FS — 1023 байта, а для HS — 1024 байта.

Полный формат пакета данных показан на рис. 3.9.

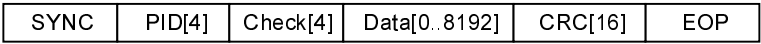


Рис. 3.9. Формат пакетов данных

3.8.4. Формат пакета подтверждения

Поле данных пакетов подтверждения пустое. Полный формат пакетов подтверждения показан на рис. 3.10.

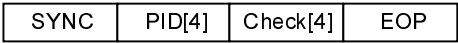


Рис. 3.10. Формат пакета подтверждения

3.8.5. Формат пакета SPLIT

Поле данных пакета SPLIT содержит (полный формат пакета показан на рис. 3.11):

- [7] адрес хаба;
- [1] флаг SC;
- [7] адрес порта;
- [1] поле S;
- [1] поле E;
- [1] тип конечной точки ET;
- [5] циклический контрольный код.

Флаг SC (Start/Complete) принимает значение 0, если это *маркер запуска* SS (Start Split) расщепленной транзакции, и 1, если это *маркер завершения* CS (Complete Split).

В зависимости от типа транзакции поля S и E трактуются по-разному. Для управляющих транзакций и прерываний поле S определяет скорость (0 — FS, 1 — LS). Для остальных транзакций, кроме OUT, поля S и E содержат нули. Для транзакции OUT значение полей [S:E] трактуется следующим образом:

- ❑ 10 — стартовый пакет;
- ❑ 01 — последний пакет;
- ❑ 00 — промежуточный пакет;
- ❑ 11 — в пакете все данные транзакции.

Поле ET описывает тип целевой конечной точки, с которой будет производиться транзакция:

- ❑ 00 — управление;
- ❑ 01 — изохронная транзакция;
- ❑ 10 — передача массива данных;
- ❑ 11 — прерывание.

Контрольная сумма CRC вычисляется по полям от HubAddr до ET включительно.

SYNC	PID[4]	Check[4]	HubAddr[7]	CS[1]	Port[7]	S[1]	E[1]	ET[2]	CRC[5]	EOP
------	--------	----------	------------	-------	---------	------	------	-------	--------	-----

Рис. 3.11. Формат пакета SPLIT

3.9. Контрольная сумма

Протокол USB использует циклический избыточный код (CRC, Cyclic Redundancy Checksums) для защиты полей пакета. CRC-контроль является более мощным методом обнаружения ошибок и используется для обнаружения ошибок на уровне блоков данных. Он основан на делении и умножении многочленов. В определенном смысле CRC-контроль является алгоритмом *хэширования*, который отображает (хэширует) элементы большого набора на элементы меньшего набора. Процесс хэширования приводит к потере информации. Хотя каждый отдельный элемент набора данных отображается на один и только один элемент хэш-набора — обратное не верно. При CRC-контроле большой набор всех возможных двоичных чисел отображается на меньший набор всех возможных CRC.

3.9.1. Алгоритм вычисления CRC

Вычисление и использование кода CRC производится в соответствии со следующей последовательностью действий.

К содержимому кадра, описываемого полиномом $F(x)$, добавляется набор единиц

$$L(x) = \sum_{n=0}^{15} x^n = 1111111111111111,$$

количество которых равно длине поля CRC.

Образованное таким образом число $x^{16} \times F(x) + x^k \times L(x)$, где k — степень $F(x)$, делится на производящий полином $g(x)$.

Остаток $O(x)$ от такого деления, определяемый из соотношения

$$Q(x) g(x) = x^{16} \times F(x) + x^k \times L(x) + O(x),$$

где $Q(x)$ — частное от деления $x^{16} \times F(x) + x^k \times L(x)$ на $g(x)$, в инвертированном виде помещается в контрольное поле кадра.

На приемной стороне выполняется деление содержимого кадра с полем CRC $x^{16} \times F^*(x) + x^k \times L(x) + O(x)$ на полином $g(x)$, где $F^*(x) = x^{16} \times F(x) + L(x) + O(x)$ — передаваемая кодовая комбинация. Результат такого деления можно привести к виду:

$$\begin{aligned} x^{16} \times [x^{16} \times F(x) + x^k \times L(x) + O(x)] / g(x) + x^{16} \times L(x) / g(x) = \\ = x^{16} [Q(x) \times g(x)] / g(x) + x^{16} \times L(x) / g(x). \end{aligned}$$

Числитель первого слагаемого делится на $g(x)$, поэтому в приемнике, если при передаче не было ошибок, остаток получается равным остатку от деления постоянного числителя второго слагаемого ($x^{16} \times L(x) / g(x)$) и имеет вид

$$x^{12} + x^{11} + x^{10} + x^8 + x^3 + x^2 + x + 1 = 1110100001111.$$

Таким образом, если результат вычислений на приемной стороне равен некоторому определенному числу (в некоторых системах нулю, либо другому числу, не совпадающему с приведенным выше), то считается, что передача выполнена без ошибок.

При выборе порождающего полинома руководствуются желаемой разрядностью остатка и его способностью выявлять ошибки. Ряд порождающих полиномов принят международными организациями. В протоколе USB используются два порождающих полинома — один для пакетов маркеров и второй для пакетов данных. Для маркеров используется полином $x^5 + x^2 + x^0$, а для данных — $x^{16} + x^{15} + x^2 + x^0$. Соответственно, получаемый контрольный код имеет размерность 5 битов и 16 битов.

Алгоритм вычисления 16-разрядной контрольной суммы (ее более привычное название — CRC16) выглядит следующим образом. В 16-битный регистр

CRC предварительно загружается число \$FFFF. Процесс начинается с добавления байтов сообщения к текущему содержимому регистра. Для генерации CRC используются только 8 бит данных. Старт и стоп-биты, бит паритета, если он используется, не учитываются в CRC.

В процессе генерации CRC каждый 8-битный символ складывается по **ИСКЛЮЧАЮЩЕМУ ИЛИ** (XOR) с содержимым регистра. Результат сдвигается в направлении младшего бита с заполнением 0 старшего бита. Младший бит извлекается и проверяется. Если младший бит равен 1, то содержимое регистра складывается с определенной ранее фиксированной величиной по **ИСКЛЮЧАЮЩЕМУ ИЛИ**. Если младший бит равен 0, то **ИСКЛЮЧАЮЩЕЕ ИЛИ** не делается.

Этот процесс повторяется, пока не будет сделано 8 сдвигов. После последнего (восьмого) сдвига следующий байт складывается с содержимым регистра, и процесс повторяется снова. **Финальное содержание регистра после обработки всех байт сообщения и есть контрольная сумма CRC.**

Таким образом, алгоритм генерации CRC выглядит так:

1. 16-битный регистр загружается числом \$FFFF и используется далее как регистр CRC.
2. Первый байт сообщения складывается по **ИСКЛЮЧАЮЩЕМУ ИЛИ** с содержимым регистра CRC. Результат помещается в регистр CRC.
3. Регистр CRC сдвигается вправо (в направлении младшего бита) на 1 бит, старший бит заполняется 0.
4. (Если младший бит 0): Повторяется шаг 3 (сдвиг).
5. (Если младший бит 1): Делается операция **ИСКЛЮЧАЮЩЕЕ ИЛИ** регистра CRC и полиномиального числа \$A001.
6. Шаги 3 и 4 повторяются восемь раз.
7. Повторяются шаги со 2 по 5 для следующего сообщения. Это повторяется до тех пор, пока все байты сообщения не будут обработаны.

Финальное содержание регистра CRC и есть контрольная сумма.

3.9.2. Программное вычисление **CRC**

При вычислении следует учитывать порядок передачи битов в протоколе USB. Листинг 3.1 показывает пример вычисления CRC5 и CRC16 для битовых последовательностей, а рис. 3.12 — результат выполнения этой программы.

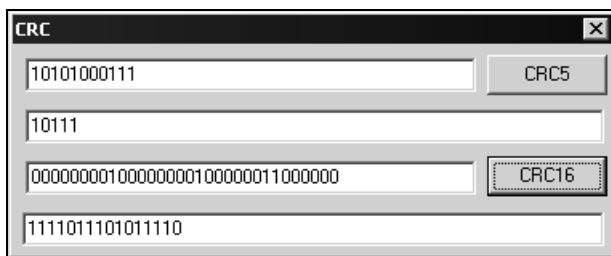


Рис. 3.12. Вычисление CRC5 и CRC16

Листинг 3.1. Вычисление CRC5 и CRC16 для битовых последовательностей

```

unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    btnCRC5: TButton;
    E_CRC5: TEdit;
    Edit2: TEdit;
    btnCRC16: TButton;
    E_CRC16: TEdit;
    procedure btnCRC5Click(Sender: TObject);
    procedure btnCRC16Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

```

```
{ $R *.dfm }

// Сдвиг битовой строки влево:
// результат - самый левый символ
// удаляет первый символ
// добавляет в конец 0
function Shift(var S : String) : Char;
begin
    Result:= S[1];
    Delete(S, 1, 1);
    S:= S + '0';
end;

// Операция XOR для битовых строк
procedure XorStr(var S : String; XStr : String);
var i : Byte;
begin
    for i:= 1 to Length(XStr) do begin
        if S[i] = XStr[i] then S[i]:= '0' else S[i]:= '1';
    End;
end;

// Вычисление CRC5
procedure TForm1.btnCRC5Click(Sender: TObject);
var S : String; i : Byte;
    Result : String;
begin
    Result:= '11111';

    S:= Edit1.Text;
    For i:= 1 to Length(S) do
        If S[i] <> Shift(Result) then
            XorStr(Result, '00101');
    XorStr(Result, '11111');

    E_CRC5.Text:= Result;
end;

// Вычисление CRC16
procedure TForm1.btnCRC16Click(Sender: TObject);
```

```

var S : String; i : Byte;
    Result : String;
begin
    Result:= '1111111111111111';

    S:= Edit2.Text;
    For i:= 1 to Length(S) do
        If S[i] <> Shift(Result) then
            XorStr(Result, '1000000000000101');
        XorStr(Result, '1111111111111111');

    E_CRC16.Text:= Result;
end;

end.

```

3.10. Транзакции

Все обмены (*транзакции*) по USB состоят из трех пакетов (рис. 3.13). Каждая транзакция планируется и начинается по инициативе хост-контроллера, который посылает *маркер-пакет* (т. е. пакет типа token). Он описывает тип и направление передачи, адрес устройства USB и номер конечной точки. В каждой транзакции возможен обмен только между устройством (его конечной точкой) и хостом. Адресуемое 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 подтверждения* (т. е. пакет типа Handshake).



Рис. 3.13. А — передача данных от хоста,
Б — передача данных хосту

Периферийное устройство не может выдавать на шину какую-либо информацию по собственной инициативе и не может самостоятельно посылать запросы прерываний.

3.10.1. Типы транзакций

Спецификация USB определяет следующие типы транзакций:

□ передача команды:

- хост посылает маркер **SETUP**, содержащий номер функции и номер конечной точки, для которой предназначена команда;
- хост посылает выбранной конечной точке пакет данных со сброшенным битом синхронизации (т. е. пакет типа **Data0**), содержащий 8-байтный код команды;
- функция посылает хосту пакет подтверждения;

□ изохронная передача данных:

- хост посылает маркер **OUT**, содержащий номер функции и номер конечной точки, для которой предназначены данные;
- хост посылает выбранной конечной точке пакет данных со сброшенным битом синхронизации (т. е. пакет типа **Data0**);

□ передача данных с подтверждением:

- хост посылает маркер **OUT**, содержащий номер функции и номер конечной точки, для которой предназначены данные;
- хост посылает выбранной конечной точке пакет данных;
- функция посылает хосту пакет подтверждения.

□ изохронный прием данных:

- хост посылает маркер **IN**, содержащий номер функции и номер конечной точки, от которой запрашиваются данные;
- выбранная конечная точка передает хосту пакет данных со сброшенным битом синхронизации (т. е. пакет типа **Data0**).

□ прием данных с подтверждением:

- хост посылает маркер **IN**, содержащий номер функции и номер конечной точки, от которой запрашиваются данные;
- выбранная конечная точка передает хосту пакет данных или пакет подтверждения (**NAK** — данные не готовы, **STALL** — сбой);
- если хост получил пакет данных, он посылает пакет подтверждения (**ACK**).

3.10.2. Подтверждение транзакций и управление потоком

При выполнении транзакций используются три типа пакетов подтверждения, называемые *пакетами квитирования* (handshake packets, см. разд. 4.5):

- ❑ ACK — информация принята получателем без ошибок, операция успешно завершена;
- ❑ NAK — функция занята (не готова к приему или передаче данных);
- ❑ STALL — произошел сбой при выполнении операции, функция не может принять или передать данные.

Если транзакция выполнена успешно — передается пакет ACK, а в случае ошибки возможно несколько вариантов:

- ❑ при неготовности устройства к выполнению операции (нет данных для передачи хосту, отсутствует место в буфере для приема, не завершена операция управления и т. д.) передается пакет NAK, заставляющий хост-контроллер повторить транзакцию позже. Такая ситуация, в общем-то, является вполне нормальной и клиентское ПО не уведомляется об ошибке;
- ❑ если при выполнении транзакции передачи данных с подтверждением в пакете данных обнаружена ошибка по контрольному коду, получатель пакета не высылает пакет подтверждения. Отправитель при отсутствии подтверждения от получателя должен зафиксировать ошибку передачи данных и повторить транзакцию;
- ❑ при серьезной ошибке передается пакет STALL, обозначающий, что без вмешательства хоста работа с конечной точкой невозможна. В отличие от NAK, ответ STALL доводится до сведения драйвера USB, который отменяет все дальнейшие транзакции с этой точкой, и до клиентского драйвера.

Ответ NAK позволяет только уведомить о невозможности приема данных, в то время как сам пакет данных уже отправлен. Такое поведение попусту растрчивает пропускную способность шины. В спецификации USB 2.0 добавлен еще один пакет, позволяющий более гибко управлять транзакциями передачи больших объемов данных — пакет PING. С помощью этого пакета хост может предварительно опросить готовность устройства к приему пакета максимального размера. В ответ на этот запрос устройство должно либо ответить пакетом ACK, подтвердив готовность к приему пакета данных, либо пакетом NAK, если устройство не готово. Отрицательный ответ заставит хост повторить пробу позже, а положительный разрешает ему выполнить транзакцию вывода данных. На транзакцию вывода данных после положительного ответа на пробу устройство может ответить следующими пакетами:

- ❑ ACK — прием пакета успешен и устройство готово к приему следующего полноразмерного пакета;

- ❑ NYET — успешный прием, но неготовность к следующему пакету;
- ❑ NAK — неготовность устройства (может случиться, что за время от пакета пробы до отправки пакета устройство перестало быть готово к приему).

Высокоскоростное устройство в дескрипторах конечных точек сообщает о возможной интенсивности посылок NAK: поле `bInterval` для конечных точек передачи данных и управления указывает число микрокадров, приходящееся на один ответ NAK (0 означает, что устройство никогда не ответит посылкой пакета NAK на транзакцию вывода).

3.10.3. Протоколы транзакций

В зависимости от типа передачи данных каждая посылка состоит из одной или нескольких транзакций.

Управляющие посылки

Существуют три типа управляющих посылок (рис. 3.14):

- ❑ *посылка записи данных* (Control Write) состоит из следующих транзакций:
 - передача команды;
 - передача с подтверждением одного или нескольких пакетов данных;
 - прием с подтверждением пустого пакета данных, подтверждающего успешное завершение операции;
- ❑ *посылка чтения данных* (Control Read) состоит из следующих транзакций:
 - передача команды;
 - прием с подтверждением одного или нескольких пакетов данных;
 - передача с подтверждением пустого пакета данных, подтверждающего успешное завершение операции;
- ❑ *посылка без данных* (No-data Control) состоит из следующих транзакций:
 - передача команды;
 - прием с подтверждением пустого пакета данных, подтверждающего успешное завершение операции.

При выполнении транзакции передачи данных признак синхронизации данных должен быть сброшен в ноль (блок данных, содержащий код команды, имеет тип `Data0`).

Если команда предполагает прием или передачу данных, то после каждой транзакции признак синхронизации данных инвертируется: первый блок имеет тип `Data1`, второй — `Data0`, третий — `Data1` и т. д.

Пустой пакет данных, подтверждающий завершение управляющей посылки, должен иметь тип `Data1`.

При передаче управляющей посылки максимальный размер пакета для пол-
носкоростного устройства может составлять 8, 16, 32 или 64 байта, а для
низкоскоростного всегда равен 8 байтам.

Для передачи сообщений по Основному каналу сообщений всегда использу-
ется максимальный размер пакета, равный 8 байтам.

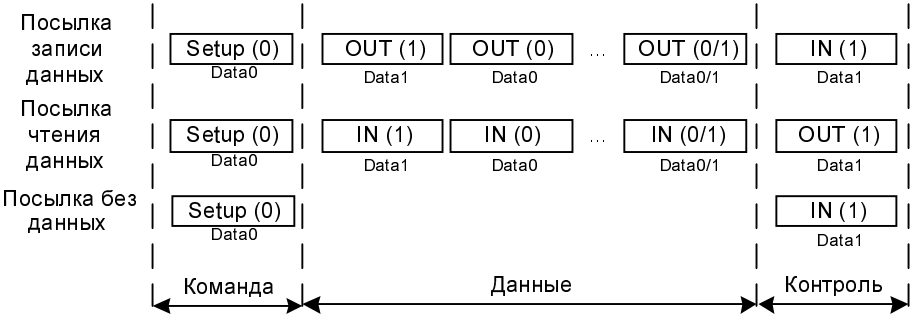


Рис. 3.14. Формат управляющих посылок

Передачи массивов данных

Существуют два типа передачи массивов (рис. 3.15):

- ❑ передача массива данных от хоста к конечной точке (Bulk Write);
- ❑ прием хостом массива данных от конечной точки (Bulk Read).

Передача данных от хоста к конечной точке состоит из следующих друг за другом транзакций передачи данных с подтверждением, а передача данных от конечной точки к хосту — из следующих друг за другом транзакций приема с подтверждением. И в том, и в другом случае перед началом пере-
дачи массива триггер синхронизации данных должен быть сброшен в 0: при
выполнении первой транзакции блок данных имеет тип Data0, второй —
Data1, третий — Data0 и т. д.

Прием и передачу массивов данных могут выполнять только полноскорост-
ные устройства. Максимальный размер пакета при передаче массива может
быть равен 8, 16, 32 или 64 байтам.

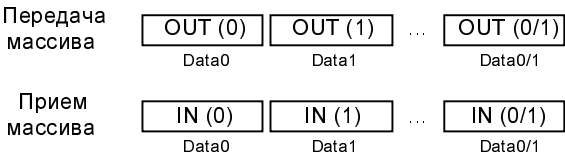


Рис. 3.15. Формат посылок передачи данных

Передачи по прерываниям

Существуют два типа передачи по прерываниям:

- ☐ передача массива данных от хоста к конечной точке по прерыванию;
- ☐ прием хостом массива данных от конечной точки по прерыванию.

Передача данных по прерыванию заключается в выполнении транзакции передачи пакета данных с подтверждением от хоста к конечной точке. Прием заключается в выполнении транзакции приема пакета данных с подтверждением от конечной точки. При приеме или передаче каждого блока данных происходит переключение триггера данных. Первый передаваемый (или принимаемый) блок должен иметь тип `Data0`, следующий — `Data1` и т. д.

Максимальный размер пакета при передаче по прерыванию для низкоскоростных устройств не может быть более 8 байт, а для высокоскоростных — более 64 байт.

Изохронные передачи

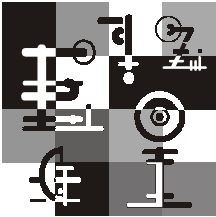
Существуют два типа изохронной передачи:

- ☐ изохронная передача данных от хоста к конечной точке;
- ☐ изохронный прием данных хостом от конечной точки.

Изохронная передача данных заключается в выполнении транзакции передачи пакета данных без подтверждения от хоста к конечной точке. Изохронный прием заключается в выполнении транзакции приема пакета данных без подтверждения от конечной точки.

Состояние триггера данных при изохронной передаче игнорируется, но рекомендуется сбросить его в ноль перед началом передачи.

Изохронную передачу могут выполнять только полноскоростные устройства. Максимальный размер пакета данных при изохронной передаче — 1023 байта.



Внутренняя организация устройства

Миксер — это устройство, приводящее продукты в замешательство.

4.1. Запросы к USB-устройствам

Все USB-устройства принимают запросы от хост-контроллера и отвечают на них через Основной канал сообщений (см. разд. 3.7). Запросы выполняются при помощи управляющих посылок (см. разд. 3.10.3).

4.1.1. Конфигурационный пакет

Запрос и его параметры передаются устройству в конфигурационном пакете (Setup Packet). Конфигурационный пакет имеет размер 8 байт (табл. 4.1). Структура конфигурационного пакета на языке Pascal показана в листинге 4.1.

Таблица 4.1. Конфигурационный пакет

Смещение	Поле	Размер	Описание
0	bmRequestType	BY TE	Тип запроса
1	bRequest	BYTE	Код запроса
2	wValue	WORD	Параметр запроса
4	wIndex	WORD	Индекс или смещение
6	wLength	WORD	Число байт для передачи

Листинг 4.1. Конфигурационный пакет

```
TSetupPacket = packed record
    bmRequest : UCHAR;
    bRequest  : UCHAR;
```

```

wValue      : Array [1..2] of UCHAR;
wIndex      : Array [1..2] of UCHAR;
wLength     : Array [1..2] of UCHAR;
End;
```

Тип запроса `bmRequestType` имеет размер 1 байт и состоит из следующих битов:

- [7] направление передачи:
 - 0 — от хоста к устройству;
 - 1 — от устройства к хосту;
- [6:5] код типа запроса:
 - 0 — стандартный запрос;
 - 1 — специфический запрос для данного класса;
 - 2 — специфический запрос изготовителя;
 - 3 — зарезервирован;
- [4:0] код получателя:
 - 0 — устройство;
 - 1 — интерфейс;
 - 2 — другой получатель;
 - 4—31 зарезервированы.

Поле кода запроса определяет операцию, выполняемую запросом. В спецификации USB определены только коды стандартных запросов к устройству (листинг 4.2).

Листинг 4.2. Коды стандартных запросов

```

// GET_STATUS (определение состояния устройства)
#define USB_REQUEST_GET_STATUS                0x00
// CLEAR_FEATURE (сброс устройства)
#define USB_REQUEST_CLEAR_FEATURE             0x01
// код 2 зарезервирован
// SET_FEATURE (установить свойство)
#define USB_REQUEST_SET_FEATURE               0x03
// код 4 зарезервирован
// SET_ADDRESS (установить адрес)
#define USB_REQUEST_SET_ADDRESS               0x05
```

```
// GET_DESCRIPTOR (получить дескриптор)
#define USB_REQUEST_GET_DESCRIPTOR 0x06
// SET_DESCRIPTOR (загрузить дескриптор)
#define USB_REQUEST_SET_DESCRIPTOR 0x07
// GET_CONFIGURATION (получить код текущей конфигурации)
#define USB_REQUEST_GET_CONFIGURATION 0x08
// SET_CONFIGURATION (установить конфигурацию)
#define USB_REQUEST_SET_CONFIGURATION 0x09
// GET_INTERFACE (получить код интерфейса)
#define USB_REQUEST_GET_INTERFACE 0x0A
// SET_INTERFACE (установить интерфейс)
#define USB_REQUEST_SET_INTERFACE 0x0B
// SYNC_FRAME (кадр синхронизации)
#define USB_REQUEST_SYNC_FRAME 0x0C
```

Значения параметров Value и Index зависят от типа запроса. В запросах на прием или передачу дескрипторов параметр Value содержит тип дескриптора (листинг 4.3) в старшем байте и индекс дескриптора — в младшем.

Листинг 4.3. Типы дескрипторов

```
// Стандартный дескриптор устройства
#define USB_DEVICE_DESCRIPTOR_TYPE 0x01
// Дескриптор конфигурации
#define USB_CONFIGURATION_DESCRIPTOR_TYPE 0x02
// Дескриптор строки
#define USB_STRING_DESCRIPTOR_TYPE 0x03
// Дескриптор интерфейса
#define USB_INTERFACE_DESCRIPTOR_TYPE 0x04
// Дескриптор конечной точки
#define USB_ENDPOINT_DESCRIPTOR_TYPE 0x05
// Уточняющий дескриптор устройства
#define DEVICE_QUALIFIER 0x06
// Дескриптор дополнительной конфигурации
#define OTHER_SPEED_CONFIGURATION 0x07
// Дескриптор управления питанием интерфейса
#define INTERFACE_POWER 0x08
// Дескриптор OTG
#define OTG 0x09
```

```
// Отладочный дескриптор
#define DEBUG                                0x0A

// Дополнительный дескриптор интерфейса
#define INTERFACE_ASSOCIATION               0x0B
```

Поле Index обычно используется для задания номера интерфейса или конечной точки. Если поле Index задает конечную точку, то оно имеет следующий формат:

- ☐ [15:8] зарезервированы и должны содержать нули;
- ☐ [7] направление передачи конечной точки:
 - 0 — выход (OUT, от хоста);
 - 1 — вход (IN, к хосту);
- ☐ [6:4] зарезервированы и должны содержать нули;
- ☐ [3:0] номер конечной точки.

Если поле Index задает номер интерфейса, то оно имеет следующий формат:

- ☐ [15:8] зарезервированы и должны содержать нули;
- ☐ [7:0] номер интерфейса.

Если устройство получает некорректный или неподдерживаемый запрос, оно должно ответить пакетом типа STALL (см. разд. 3.8).

4.1.2. Стандартные запросы к устройствам

Получение состояния **GET_STATUS**

Запрос GET_STATUS позволяет определить состояние устройства, интерфейса или конечной точки. Запрос имеет следующие параметры:

- ☐ bmRequestType обозначает тип запроса:
 - 10000000b — получить состояние устройства;
 - 10000001b — получить состояние интерфейса;
 - 10000010b — получить состояние конечной точки;
- ☐ bRequest = 0x00;
- ☐ wValue = 0;
- ☐ wIndex — адресант запроса:
 - 0, если запрос обращен к устройству;
 - номер интерфейса, если запрос к интерфейсу;
 - номер конечной точки, если запрос к конечной точке;
- ☐ wLength = 2.

По запросу GET_STATUS устройство возвращает 16-разрядное слово состояния, описывающее текущее состояние устройства, интерфейса или конечной точки.

Слово состояния устройства имеет следующие биты:

- ☐ [15:2] зарезервированы и должны содержать нули;
- ☐ [1] — реакция на сигнал пробуждения от шины USB (remote wakeup):
 - 0 — устройство игнорирует сигнал пробуждения;
 - 1 — устройство реагирует на сигнал пробуждения;
- ☐ [0] — режим питания:
 - 0 — устройство получает питание от шины USB;
 - 1 — устройство получает питание от собственного источника.

Слово состояния интерфейса зарезервировано и содержит нули во всех разрядах.

Разряды *слова состояния конечной точки* имеют следующие значения:

- ☐ [15:1] зарезервированы и должны содержать нули;
- ☐ [0] — признак блокировки (halt) конечной точки:
 - 0 — конечная точка функционирует нормально;
 - 1 — передача данных заблокирована.

Сброс свойства **CLEAR_FEATURE**

Запрос CLEAR_FEATURE используется для запрета свойства или состояния, указываемого значением поля wValue. Запрос имеет следующие параметры:

- ☐ bmRequestType обозначает тип запроса:
 - 00000000b — запретить свойство устройства;
 - 00000001b — запретить свойство интерфейса;
 - 00000010b — запретить свойство конечной точки;
- ☐ bRequest = 1;
- ☐ wValue — код свойства;
- ☐ wIndex — адресант запроса:
 - 0, если запрос обращен к устройству;
 - номер интерфейса, если запрос к интерфейсу;
 - номер конечной точки, если запрос к конечной точке;
- ☐ wLength = 0.

Спецификация USB определяет три кода свойств:

- ❑ 0 — блокировка конечной точки (ENDPOINT_HALT, получатель — конечная точка);
- ❑ 1 — разрешить выполнение сигнала пробуждения (DEVICE_REMOTE_WAKEUP, получатель — устройство);
- ❑ 2 — тестовый режим (TEST_MODE, получатель — устройство).

Передача данных по запросу CLEAR_FEATURE не производится. Сброс состояния ENDPOINT_HALT разблокирует конечную точку; сброс состояния DEVICE_REMOTE_WAKEUP лишает устройство способности реагировать на сигнал пробуждения.

Разрешение свойства **SET_FEATURE**

Запрос SET_FEATURE используется для разрешения свойства или состояния, указываемого значением поля wValue. Запрос имеет следующие параметры:

- ❑ bmRequestType обозначает тип запроса:
 - 00000000b — разрешить свойство устройства;
 - 00000001b — разрешить свойство интерфейса;
 - 00000010b — разрешить свойство конечной точки;
- ❑ bRequest = 0x03;
- ❑ wValue — код свойства;
- ❑ wIndex — адресант запроса:
 - 0, если запрос обращен к устройству;
 - номер интерфейса, если запрос к интерфейсу;
 - номер конечной точки, если запрос к конечной точке;
- ❑ wLength = 0.

Передача данных по запросу SET_FEATURE не производится. Установка состояния ENDPOINT_HALT блокирует конечную точку; установка состояния DEVICE_REMOTE_WAKEUP позволяет устройству реагировать на сигнал пробуждения.

Задание адреса на шине **SET_ADDRESS**

Запрос SET_ADDRESS позволяет присвоить устройству новое значение адреса на шине USB. Запрос имеет следующие параметры:

- ❑ bmRequestType = 00000000b;
- ❑ bRequest = 0x05;
- ❑ wValue — адрес устройства;

- ❑ `wIndex = 0;`
- ❑ `wLength = 0.`

Передача данных при выполнении запроса `SET_ADDRESS` не производится.

Получение дескриптора **GET_DESCRIPTOR**

Запрос `GET_DESCRIPTOR` позволяет получить дескриптор устройства, дескриптор конфигурации или дескриптор строки. Запрос имеет следующие параметры:

- ❑ `bmRequestType = 10000000b, 10000001b` или `10000010b;`
- ❑ `bRequest = 0x06;`
- ❑ `wValue` содержит тип дескриптора в старшем байте (листинг 4.3) и индекс дескриптора в младшем байте (при запросе дескриптора устройства индекс имеет значение 0);
- ❑ `wIndex` равно 0 для дескриптора устройства или конфигурации, или идентификатору языка для дескриптора строки;
- ❑ `wLength` — размер дескриптора в байтах.

Подробнее формат дескриптора и его получение мы рассмотрим в *разд. 4.1.3*. Значение поля `bmRequestType` зависит от типа запрашиваемого дескриптора:

- ❑ `10000000b` — если дескриптор относится к устройству;
- ❑ `10000001b` — если дескриптор относится к интерфейсу;
- ❑ `10000010b` — если дескриптор относится к конечной точке.

Дополнительные подробности мы обсудим в *разд. 8.4.3*.

Передача дескриптора **SET_DESCRIPTOR**

Запрос `SET_DESCRIPTOR` позволяет дополнить существующий или добавить новый дескриптор устройства, конфигурации или строки. Запрос имеет следующие параметры:

- ❑ `bmRequestType = 00000000b, 00000001b` или `00000010b;`
- ❑ `bRequest = 0x07;`
- ❑ `wValue`, `wIndex`, `wLength` имеют те же значения, что и для `GET_DESCRIPTOR`.

В процессе выполнения запроса `SET_DESCRIPTOR` хост передает устройству дескриптор, тип которого определяется параметрами запроса.

Получение кода конфигурации **GET_CONFIGURATION**

По запросу GET_CONFIGURATION устройство выдает код своей текущей конфигурации. Запрос имеет следующие параметры:

- ❑ bmRequestType = 10000000b;
- ❑ bRequest = 0x08;
- ❑ wValue = 0;
- ❑ wIndex = 0;
- ❑ wLength = 1.

При выполнении запроса GET_CONFIGURATION от устройства к хосту передается один байт данных, содержащий код конфигурации устройства.

Задание кода конфигурации **SET_CONFIGURATION**

Запрос SET_CONFIGURATION позволяет задать устройству новую конфигурацию. Запрос имеет следующие параметры:

- ❑ bmRequestType = 00000000b;
- ❑ bRequest = 0x09;
- ❑ wValue — код конфигурации;
- ❑ wIndex = 0;
- ❑ wLength = 1.

При выполнении запроса SET_CONFIGURATION от хоста к устройству передается один байт данных, содержащий код конфигурации устройства, которую требуется установить.

Получение кода настройки интерфейса **GET_INTERFACE**

Запрос GET_INTERFACE позволяет получить код текущей настройки для указанного интерфейса. Запрос имеет следующие параметры:

- ❑ bmRequestType = 10000001b;
- ❑ bRequest = 0x0A;
- ❑ wValue = 0;
- ❑ wIndex — номер интерфейса;
- ❑ wLength = 1.

При выполнении запроса GET_INTERFACE от устройства к хосту передается один байт данных, содержащий код текущего варианта настройки интерфейса.

Задание кода настройки интерфейса **SET_INTERFACE**

Запрос SET_INTERFACE позволяет задать новый вариант настройки для указанного интерфейса. Запрос имеет следующие параметры:

- ❑ bmRequestType = 00000001b;
- ❑ bRequest = 0x0B;
- ❑ wValue — код варианта настройки интерфейса;
- ❑ wIndex — номер интерфейса;
- ❑ wLength = 0.

При выполнении запроса SET_INTERFACE передача данных не производится.

Задание номера кадра синхронизации **SYNC_FRAME**

Запрос SYNC_FRAME используется для задания номера кадра синхронизации. Запрос имеет следующие параметры:

- ❑ bmRequestType = 10000010b;
- ❑ bRequest = 0x0C;
- ❑ wValue = 0;
- ❑ wIndex — номер конечной точки;
- ❑ wLength = 2.

С помощью запроса SYNC_FRAME хост передает заданной конечной точке, работающей в изохронном режиме, 16-разрядное слово данных, содержащее номер кадра, который конечная точка должна использовать для синхронизации передачи.

Обработка стандартных запросов

Листинг 4.4 содержит константы стандартных запросов. Каждый идентификатор является объединением полей bmRequestType и bRequest. Эти константы позволяют легко распознавать запросы.

Листинг 4.4. Константы описания стандартных запросов и их применение

```
#define GET_STATUS_DEVICE      0x8000
#define GET_STATUS_INTERF     0x8100
#define GET_STATUS_ENDPNT     0x8200
#define CLEAR_FEATURE_DEVICE  0x0001
#define CLEAR_FEATURE_INTERF  0x0101
#define CLEAR_FEATURE_ENDPNT  0x0201
#define SET_FEATURE_DEVICE     0x0003
```

```

#define SET_FEATURE_INTERF      0x0103
#define SET_FEATURE_ENDPNT      0x0203
#define SET_ADDRESS             0x0005
#define GET_DESCRIPTOR_DEVICE    0x8006
#define GET_DESCRIPTOR_INTERF    0x8106
#define GET_DESCRIPTOR_ENDPNT    0x8206
#define SET_DESCRIPTOR          0x0007
#define GET_CONFIGURATION        0x8008
#define SET_CONFIGURATION        0x0009
#define GET_INTERFACE           0x810A
#define SET_INTERFACE           0x010B
#define SYNCH_FRAME             0x820C
#define GET_REPORT              0xA101

```

```

switch (wRequest)
{
    case GET_STATUS_DEVICE:
        break;
    case GET_STATUS_INTERF:
        break;
    ... ..
    default:
        break;
}

```

4.1.3. Дескрипторы устройства

Дескриптор устройства (device descriptor) — это структура данных, или форматированный блок информации, который позволяет хосту получить описание устройства. Каждый дескриптор содержит информацию либо об устройстве в целом, либо о его части.

Все USB-устройства должны передавать хабу свои дескрипторы в ответ на стандартный запрос. Это означает, что любое периферийное устройство должно делать две вещи: во-первых, хранить информацию о своих дескрипторах и, во-вторых, пересылать эту информацию в ответ на запрос хаба в определенном формате.

Типы дескрипторов были описаны в предыдущем разделе. Их список приведен в листинге 4.2. Спецификация USB определяет специальную группу дескрипторов, которая должна выдаваться устройством в ответ на стандарт-

ные запросы. Такие дескрипторы называются *стандартными дескрипторами* (standard descriptors).

Дескриптор устройства

Стандартный дескриптор устройства (Standard Device Descriptor) содержит основную информацию об USB-устройстве в целом и обо всех существующих конфигурациях. Устройство может иметь только один такой дескриптор. HS-устройство, содержащее различную информацию для HS- и FS-режимов, должно иметь также уточняющий дескриптор устройства (*см. далее*). Структура стандартного дескриптора показана в табл. 4.2, а соответствующее описание на языках C и Pascal показано в листинге 4.5.

Таблица 4.2. Структура стандартного дескриптора устройства

Смещение	Поле	Размер	Описание
0	bLength	1	Размер дескриптора в байтах
1	bDescriptorType	1	Тип дескриптора (USB_DEVICE_DESCRIPTOR_TYPE)
2	bcdUSB	2	Номер версии спецификации USB в формате BCD
4	bDeviceClass	1	Код класса USB
5	bDeviceSubClass	1	Код подкласса устройства USB
6	bDeviceProtocol	1	Код протокола USB
7	bMaxPacketSize0	1	Максимальный размер пакета для нулевой конечной точки
8	idVendor	2	Идентификатор изготовителя устройства
10	idProduct	2	Идентификатор продукта
12	bcdDevice	2	Номер версии устройства в формате BCD
14	iManufacturer	1	Индекс дескриптора строки, описывающей изготовителя
15	iProduct	1	Индекс дескриптора строки, описывающей продукт
16	iSerialNumber	1	Индекс дескриптора строки, содержащей серийный номер устройства
17	bNumConfigurations	1	Количество возможных конфигураций устройства

Листинг 4.5. Стандартный дескриптор устройства

```
// описание на языке C
typedef struct _USB_DEVICE_DESCRIPTOR {
    UCHAR    bLength;
    UCHAR    bDescriptorType;
    USHORT   bcdUSB;
    UCHAR    bDeviceClass;
    UCHAR    bDeviceSubClass;
    UCHAR    bDeviceProtocol;
    UCHAR    bMaxPacketSize0;
    USHORT   idVendor;
    USHORT   idProduct;
    USHORT   bcdDevice;
    UCHAR    iManufacturer;
    UCHAR    iProduct;
    UCHAR    iSerialNumber;
    UCHAR    bNumConfigurations;
} USB_DEVICE_DESCRIPTOR;
```

```
// описание на языке Pascal
TUsbDeviceDescriptor = packed record
    bLength          : BYTE;
    bDescriptorType  : BYTE;
    bcdUSB           : WORD;
    bDeviceClass     : BYTE;
    bDeviceSubClass  : BYTE;
    bDeviceProtocol  : BYTE;
    bMaxPacketSize0  : BYTE;
    idVendor         : WORD;
    idProduct        : WORD;
    bcdDevice        : WORD;
    iManufacturer    : BYTE;
    iProduct         : BYTE;
    iSerialNumber    : BYTE;
    bNumConfigurations : BYTE;
End;
```


Поля стандартного дескриптора конфигурации подчиняются следующим правилам:

- ☐ размер дескриптора (поле `bLength`) всегда составляет 18 байт;
- ☐ код типа дескриптора (поле `bDescriptorType`) имеет значение 1;
- ☐ номер версии (поле `bcdUSB`) представляется в формате BCD и может принимать следующие значения:
 - 0100H — версия 1.0;
 - 0110H — версия 1.1;
 - 0200H — версия 2.0;
- ☐ HS-устройства должны возвращать значение версии 2.0;
- ☐ поле кода класса (поле `bDeviceClass`) может принимать следующие значения:
 - значение 00H обозначает, что интерфейсы функционируют независимо друг от друга, и каждый из них имеет собственный код класса;
 - значение между 1 и FEH обозначает, что устройство поддерживает различные спецификации для интерфейсов, и интерфейсы не могут функционировать независимо;
 - значение FFH обозначает, что класс устройства определяется изготовителем;
- ☐ код подкласса (поле `bDeviceSubClass`) имеет значение 0;
- ☐ код протокола (поле `bDeviceProtocol`) имеет значение 0;
- ☐ максимальный размер пакета для нулевой конечной точки (поле `bMaxPacketSize0`) составляет 64 байта для HS и 8 байт для других режимов (хотя в общем случае могут использоваться значения 8, 16, 32 и 64);
- ☐ число возможных конфигураций (поле `bNumConfiguration`) описывает число конфигураций только для текущей скорости работы, но не для обеих скоростей.

Идентификатор изготовителя устройства, идентификатор продукта и номер версии используются для подбора драйвера (*см. разд. 10.4*).

Индексы дескрипторов строк используются для получения информации об устройстве в текстовом формате: при передаче запроса на получение дескриптора строки, индекс дескриптора передается в младшем байте параметра `wValue`.

Уточняющий дескриптор устройства

Уточняющий дескриптор устройства (`Device Qualifier Descriptor`) содержит дополнительную информацию о HS-устройстве при его работе на другой

скорости. Например, если устройство работает в FS-режиме, то уточняющий дескриптор вернет информацию об HS-режиме работы, и наоборот. Структура уточняющего дескриптора показана в табл. 4.3.

Таблица 4.3. Структура уточняющего дескриптора устройства

Смещение	Поле	Размер	Описание
0	bLength	1	Размер дескриптора в байтах
1	bDescriptorType	1	Тип дескриптора (DEVICE_QUALIFIER)
2	bcdUSB	2	Номер версии спецификации USB в формате BCD (равно 0200H)
4	bDeviceClass	1	Код класса USB
5	bDeviceSubClass	1	Код подкласса устройства USB
6	bDeviceProtocol	1	Код протокола USB
7	bMaxPacketSize0	1	Максимальный размер пакета для нулевой конечной точки
8	bNumConfigurations	1	Количество дополнительных конфигураций устройства
9	bReserved	1	Зарезервировано, должно быть равно нулю

Поля `idVendor`, `idProduct`, `bcdDevice`, `iManufacturer`, `iProduct`, `iSerialNumber`, присутствующие в стандартном дескрипторе, в дополнительном дескрипторе отсутствуют, т. к. эта информация одинакова для всех скоростей работы.

Дескриптор конфигурации

Стандартный дескриптор конфигурации (Standard Configuration Descriptor) содержит информацию об одной из возможных конфигураций устройства. Структура дескриптора конфигурации показана в табл. 4.4, а соответствующее описание на языках C и Pascal показано в листинге 4.6.

Таблица 4.4. Структура стандартного дескриптора конфигурации

Смещение	Поле	Размер	Описание
0	bLength	1	Размер дескриптора в байтах
1	bDescriptorType	1	Тип дескриптора (USB_CONFIGURATION_DESCRIPTOR_TYPE)

Таблица 4.4 (окончание)

Смещение	Поле	Размер	Описание
2	wTotalLength	2	Общий объем данных (в байтах), возвращаемый для данной конфигурации
4	bNumInterfaces	1	Количество интерфейсов, поддерживаемых данной конфигурацией
5	bConfigurationValue	1	Идентификатор конфигурации, используемый при вызове SET_CONFIGURATION для установки данной конфигурации
6	iConfiguration	1	Индекс дескриптора строки, описывающей данную конфигурацию
7	bmAttributes	1	Характеристики конфигурации
8	MaxPower	1	Код мощности, потребляемой устройством от шины USB

Листинг 4.6. Дескриптор конфигурации

```
// Описание на языке C
typedef struct _USB_CONFIGURATION_DESCRIPTOR {
    UCHAR    bLength;
    UCHAR    bDescriptorType;
    USHORT   wTotalLength;
    UCHAR    bNumInterfaces;
    UCHAR    bConfigurationValue;
    UCHAR    iConfiguration;
    UCHAR    bmAttributes;
    UCHAR    MaxPower;
} USB_CONFIGURATION_DESCRIPTOR;

// Описание на языке Pascal
TUsbConfigurationDescriptor = packed record
    bLength          : BYTE;
    bDescriptorType  : BYTE;
    wTotalLength     : WORD;
    bNumInterfaces   : BYTE;
    bConfigurationValue : BYTE;
    iConfiguration   : BYTE;
```

```
bmAttributes      : BYTE;  
MaxPower          : BYTE;  
End;
```

Поле `bmAttributes` представляет собой битовую маску, имеющую следующие значения:

- ☐ [7] зарезервирован и должен равняться нулю;
- ☐ [6] признак наличия собственного источника питания:
 - 0 — устройство получает питание по шине USB;
 - 1 — устройство имеет собственный источник питания;
- ☐ [5] признак возможности пробуждения устройства по внешнему сигналу:
 - 0 — устройство не имеет такой возможности;
 - 1 — устройство имеет возможность пробуждения;
- ☐ [4:0] зарезервированы и должны содержать нули.

Значение поля `MaxPower` равно максимальному току в миллиамперах, потребляемому устройством от шины USB, деленному на 2.

Устройство может иметь один или несколько дескрипторов конфигурации в соответствии с количеством возможных конфигураций, указанных в стандартном дескрипторе устройства. Каждая конфигурация имеет один или несколько интерфейсов. Каждый интерфейс имеет ноль или несколько конечных точек.

Каждая конфигурация описывается одним стандартным дескриптором, размер которого составляет 9 байт. Поле `bConfigurationValue` является идентификатором конфигурации, описываемой данным дескриптором, и используется при установке конфигурации.

Каждая конфигурация может иметь один или несколько интерфейсов. Количество доступных интерфейсов указывается в поле `bNumInterfaces`. Например, ISDN-устройство может иметь конфигурацию с двумя интерфейсами, каждый из которых предоставляет канал по 64 Кбайт/с, либо конфигурацию с одним интерфейсом, но имеющую канал 128 Кбайт/с.

Устройство может вернуть дескриптор конфигурации с полем `bDescriptionType`, равным `OTHER_SPEED_CONFIGURATION`. Такой дескриптор описывает конфигурацию для HS-режима, если устройство поддерживает этот режим.

Дескриптор интерфейса

Стандартный дескриптор интерфейса (Standard Interface Descriptor) содержит информацию об одном из интерфейсов, доступных при определенной конфигурации устройства. Структура дескриптора интерфейса показана в

табл. 4.5, а соответствующее описание на языках C и Pascal показано в листинге 4.7.

Таблица 4.5. Структура стандартного дескриптора интерфейса

Смещение	Поле	Размер	Описание
0	bLength	1	Размер дескриптора в байтах
1	bDescriptorType	1	Тип дескриптора (USB_INTERFACE_DESCRIPTOR_TYPE)
2	bInterfaceNumber	1	Номер данного интерфейса (нумеруются с 0) в наборе интерфейсов, поддерживаемых в данной конфигурации
3	bAlternateSetting	1	Альтернативный номер интерфейса
4	bNumEndpoints	1	Число конечных точек для этого интерфейса без учета нулевой конечной точки
5	bInterfaceClass	1	Код класса интерфейса
6	bInterfaceSubClass	1	Код подкласса интерфейса
7	bInterfaceProtocol	1	Код протокола
8	iInterface	1	Индекс дескриптора строки, описывающей интерфейс

Листинг 4.7. Дескриптор интерфейса

```
// описание на языке C
typedef struct _USB_INTERFACE_DESCRIPTOR {
    UCHAR bLength;
    UCHAR bDescriptorType;
    UCHAR bInterfaceNumber;
    UCHAR bAlternateSetting;
    UCHAR bNumEndpoints;
    UCHAR bInterfaceClass;
    UCHAR bInterfaceSubClass;
    UCHAR bInterfaceProtocol;
    UCHAR iInterface;
} USB_INTERFACE_DESCRIPTOR;

// описание на языке Pascal
TusbInterfaceDescriptor = packed record
```

```
bLength      : BYTE;
bDescriptorType : BYTE;
bInterfaceNumber : BYTE;
bAlternateSetting : BYTE;
bNumEndpoints : BYTE;
bInterfaceClass : BYTE;
bInterfaceSubClass : BYTE;
bInterfaceProtocol : BYTE;
iInterface    : BYTE;
End;
```

Размер дескриптора интерфейса всегда составляет 9 байт.

При идентификации и нумерации устройств на шине дескриптор интерфейса может использоваться для определения типа устройства по кодам класса, подкласса и протокола.

Дескриптор интерфейса возвращается устройством при выполнении запроса GET_DESCRIPTOR и не может быть запрошен или установлен напрямую вызовами GET_DESCRIPTOR или SET_DESCRIPTOR.

Устройство может иметь альтернативный набор установок (alternate settings), что позволяет изменять настройки устройства после конфигурирования. По умолчанию всегда устанавливаются обычные настройки интерфейса, а с помощью запроса SET_INTERFACE могут быть установлены альтернативные настройки или возвращены настройки по умолчанию.

Если интерфейс использует только нулевую конечную точку, то поле bNumEndpoints должно быть равно нулю.

Дескриптор конечной точки

Стандартный дескриптор конечной точки (Standard Endpoint Descriptor) содержит информацию об одной из конечных точек, доступных при использовании определенного интерфейса. Структура дескриптора конечной точки показана в табл. 4.6, а соответствующее описание на языках C и Pascal показано в листинге 4.8.

Таблица 4.6. Структура стандартного дескриптора конечной точки

Смещение	Поле	Размер	Описание
0	bLength	1	Размер дескриптора в байтах
1	bDescriptorType	1	Тип дескриптора (USB_ENDPOINT_DESCRIPTOR_TYPE)

Таблица 4.6 (окончание)

Смещение	Поле	Размер	Описание
2	bEndpointAddress	1	Код адреса конечной точки
3	bmAttributes	1	Атрибуты конечной точки
4	wMaxPacketSize	2	Максимальный размер пакета для конечной точки
6	bInterval	1	Интервал опроса конечной точки при передаче данных (задается в миллисекундах)

Листинг 4.8. Дескриптор конечной точки

```
// описание на языке C
typedef struct _USB_ENDPOINT_DESCRIPTOR {
    UCHAR    bLength;
    UCHAR    bDescriptorType;
    UCHAR    bEndpointAddress;
    UCHAR    bmAttributes;
    USHORT   wMaxPacketSize;
    UCHAR    bInterval;
} USB_ENDPOINT_DESCRIPTOR, *PUSB_ENDPOINT_DESCRIPTOR;

// описание на языке Pascal
TUsbEndpointDescriptor = packed record
    bLength          : BYTE;
    bDescriptorType  : BYTE;
    bEndpointAddress : BYTE;
    bmAttributes     : BYTE;
    wMaxPacketSize   : WORD;
    bInterval        : BYTE;
End;
```

Размер дескриптора конечной точки составляет 7 байт.

Код адреса bEndpointAddress и байт атрибутов bmAttributes для многих классов периферийных устройств позволяет однозначно определить функциональное назначение конечной точки.

Код адреса `bEndpointAddress` содержит следующие биты:

- ❑ [7] направление передачи (игнорируется для каналов сообщений):
 - 0 — OUT (от хоста);
 - 1 — IN (к хосту);
- ❑ [6:4] зарезервированы и должны содержать нули;
- ❑ [3:0] номер конечной точки.

Байт атрибутов `bmAttributes` содержит следующие биты:

- ❑ [7:6] зарезервированы и должны быть равны нулю;
- ❑ [5:4] тип использования конечной точки:
 - 00 — конечная точка (данные);
 - 01 — конечная точка для явной обратной связи;
 - 10 — конечная точка неявной обратной связи;
 - 11 — зарезервировано;
- ❑ [3:2] тип синхронизации (для изохронных каналов, см. *разд. 3.4*):
 - 00 — нет синхронизации;
 - 01 — асинхронная;
 - 10 — адаптивная;
 - 11 — синхронная;
- ❑ [1:0] тип конечной точки:
 - 00 — канал сообщений;
 - 01 — изохронный канал;
 - 10 — канал передачи данных;
 - 11 — канал прерываний.

Интервал опроса конечной точки (поле `bInterval`) имеет значение только в том случае, если точка используется для передачи данных по прерываниям. Для изохронных конечных точек это поле всегда равно 1. Для остальных типов конечных точек значение этого поля игнорируется.

В HS-режиме для изохронных передач и прерываний биты [12:11] поля `wMaxPacketSize` определяют число транзакций внутри фрейма определяемым значением поля `bInterval` (табл. 4.7).

Таблица 4.7. Соответствие числа транзакций и размера пакетов

Биты [12:11] поля wMaxPacketSize	Максимально допустимый размер пакета (биты [10:0] поля wMaxPacketSize)
00	1—1024
01	513—1024
10	683—1024
11	Не используется, зарезервировано

Дескриптор строки

Дескриптор строки (UNICODE String Descriptor) содержит текст в формате UNICODE. Строка не ограничивается нулем, а длина строки вычисляется вычитанием 2 из размера дескриптора. Структура дескриптора строки показана в табл. 4.8, а соответствующее описание на языках C и Pascal показано в листинге 4.9.

Таблица 4.8. Структура дескриптора строки

Сме- щение	Поле	Размер	Описание
0	bLength	1	Размер дескриптора в байтах ($N + 2$)
1	bDescriptorType	1	Тип дескриптора (USB_STRING_DESCRIPTOR_TYPE)
2	bString	N	Строка символов UNICODE

Листинг 4.9. Дескриптор строки

```
// описание на языке C
typedef struct _USB_STRING_DESCRIPTOR {
    UCHAR bLength;
    UCHAR bDescriptorType;
    WCHAR bString[1];
} USB_STRING_DESCRIPTOR;

// описание на языке Pascal
TUsbStringDescriptor = packed record
    bLength          : BYTE;
```

```
bDescriptorType : BYTE;  
bString          : Array of Byte;  
End;
```

Дескриптор строки является необязательным. Если устройство не поддерживает дескрипторы строк, все ссылки на такие дескрипторы из дескрипторов устройства, конфигурации или интерфейса должны иметь нулевое значение.

Устройство может поддерживать несколько различных языков, поэтому при запросе дескриптора строки нужно задавать идентификатор языка (LANGID).

Строковый индекс 0 для всех языков соответствует дескриптору строки, содержащей массив 16-разрядных идентификаторов всех поддерживаемых языков. Массив идентификаторов не ограничен нулем, а размер массива в байтах вычисляется вычитанием 2 из размера дескриптора (табл. 4.9).

Таблица 4.9. Структура дескриптора идентификатора языков

Сме- щение	Поле	Размер	Описание
0	bLength	1	Размер дескриптора в байтах ($N + 2$)
1	bDescriptorType	1	Тип дескриптора (USB_STRING_DESCRIPTOR_TYPE)
2	wLANGID[0]	2	Идентификатор языка
...	...	...	...
N	wLANGID[x]	2	Идентификатор языка

Специфические дескрипторы

Специфические дескрипторы (Class-Specific Descriptor) могут использоваться в устройствах определенных классов. Например, в *разд. 5.1.2* мы будем описывать дескрипторы, специфические для хабов.

Порядок получения дескрипторов

В процессе нумерации (*см. разд. 10.2.2*) хост с помощью управляющих посылок (*см. разд. 3.3*) запрашивает дескрипторы от устройства в такой последовательности:

- ❑ дескриптор устройства;
- ❑ дескрипторы конфигураций;
- ❑ дескрипторы интерфейсов для конфигурации;
- ❑ дескрипторы интерфейсов всех конечных точек.

Реально при запросе дескриптора конфигурации устройство сразу возвращает последовательность "вложенных" дескрипторов:

- дескриптор конфигурации-1;
 - дескриптор интерфейса-1;
 - ◊ дескриптор конечной точки-1 для интерфейса-1;
 - ◊ дескриптор конечной точки-2 для интерфейса-1;
 - ◊
 - дескриптор интерфейса-2;
 - ◊ дескриптор конечной точки-1 для интерфейса-2;
 -
- дескриптор конфигурации-2.

Общая длина возвращаемых данных заранее не известна. Для определения общей длины возвращаемого списка дескрипторов служит поле `wTotalLength`. Чтобы получить весь список дескрипторов, нужно запросить первые 8 байт дескриптора конфигурации, запомнить значение поля `wTotalLength`, а затем использовать это значение в качестве параметра при повторной подаче запроса. Важно понимать, что такой алгоритм действий выполняется внутри USB-драйвера, а со стороны пользовательской программы получение списка дескрипторов выглядит несколько проще (листинг 4.10).

Листинг 4.10. Получение списка дескрипторов конфигурации

```
// конфигурационный пакет запроса
TSetupPacket = packed record
    bmRequest : UCHAR;
    bRequest  : UCHAR;
    wValue    : Array [1..2] of UCHAR;
    wIndex    : Array [1..2] of UCHAR;
    wLength   : Array [1..2] of UCHAR;
End;

// структура запроса дескриптора
TDescriptorRequest = packed record
    ConnectionIndex : ULONG;
    SetupPacket     : TSetupPacket;
    Data            : Array [1..2048] of Byte;
End;
```

```
procedure TForm1.ShowDeviceDetail(hRoot : THandle; iPort : Integer);
// переменные для выполнения DeviceIoControl
Var Success : LongBool;
    Packet : TDescriptorRequest;
    BytesReturned : Cardinal;

begin
// Получение стандартного дескриптора устройства
ZeroMemory(@Packet, SizeOf(Packet));
Packet.ConnectionIndex      := iPort+1;
Packet.SetupPacket.bmRequest := $80;
Packet.SetupPacket.bRequest  := USB_REQUEST_GET_DESCRIPTOR;
Packet.SetupPacket.wValue [2] := USB_DEVICE_DESCRIPTOR_TYPE;
Packet.SetupPacket.wLength[2] := 1; // Использовать буфер 2 Кбайт

// IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION
Success:= DeviceIoControl(hRoot, GetUSBCtlCode(5),
    @Packet, sizeof(Packet),
    @Packet, sizeof(Packet),
    BytesReturned, nil
);

If not(Success) then begin
    Log(Format('      Ошибка получения информации об устройстве %S',
        [ SysErrorMessage(GetLastError())]));
    Exit;
End;

// отображение дескриптора устройства
DisplayDescriptorInfo(Packet.Data);

// Получение дескрипторов конфигурации
ZeroMemory(@Packet, SizeOf(Packet));
Packet.ConnectionIndex      := iPort+1;
Packet.SetupPacket.bmRequest := $80;
Packet.SetupPacket.bRequest  := USB_REQUEST_GET_DESCRIPTOR;
Packet.SetupPacket.wValue [2] := USB_CONFIGURATION_DESCRIPTOR_TYPE;
Packet.SetupPacket.wLength[2] := 1; // Использовать буфер 2 Кбайт
```

```
// IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION
Success:= DeviceIoControl(hRoot, GetUSBctlCode(5),
    @Packet, sizeof(Packet),
    @Packet, sizeof(Packet),
    BytesReturned, nil
);

If not(Success) then begin
    Log(Format('      Ошибка получения информации об устройстве %S',
        [ SysErrorMessage(GetLastError())]));

    Exit;
End;

// отображение информации о дескрипторах конфигурации
DisplayDescriptorInfo(Packet.Data);
End;
```

Описание функции `DeviceIoControl` можно найти в справочной части книги (см. разд. 16.9). В нашем примере ключевыми параметрами этой функции являются:

- ❑ `hRoot` — дескриптор порта, к которому подключено устройство;
- ❑ `GetUSBctlCode(5)` — код IOCTL-функции (см. разд. 9.3.5);
- ❑ `Packet` — структура типа `TDescriptorRequest`.

Подробности обращения к USB-драйверу мы будем обсуждать в главе 7, а внутреннюю организацию самих драйверов — в главе 9.

Как видно из листинга, в результате выполнения запроса возвращается буфер данных `Packet.Data`. Для разбора и отображения этого буфера можно использовать процедуру `DisplayDescriptorInfo`, код которой показан в листинге 4.11.

Листинг 4.11. Разбор списка дескрипторов

```
procedure TForm1.DisplayDescriptorInfo(Data : Array of byte);
var iData : Integer; lenDescr, typDescr : Byte; PData : Pointer;
begin
    // Проходим по массиву дескрипторов
    // Первый байт - размер дескриптора в байтах
    // Второй байт - тип дескриптора
    iData:= 0;
```

Repeat

```
lenDescr:= Data[iData+0];
typDescr:= Data[iData+1];
PData    := @Data[iData+0];
```

Case typDescr of

\$01: begin // Стандартный дескриптор

With TDeviceDescriptor(PData^) do begin

```
Log(Format('      bcdUSB           =%s', [BCD2Str(bcdUSB) ]));
Log(Format('      bDeviceClass      =%d', [bDeviceClass  ]));
Log(Format('      bDeviceSubClass    =%d', [bDeviceSubClass ]));
Log(Format('      bDeviceProtocol   =%d', [bDeviceProtocol ]));
Log(Format('      bMaxPacketSize0    =%d', [bMaxPacketSize0  ]));
Log(Format('      idVendor          =%d', [idVendor        ]));
Log(Format('      idProduct         =%d', [idProduct       ]));
Log(Format('      bcdDevice          =%s', [BCD2Str(bcdDevice)]));
Log(Format('      iManufacturer      =%d', [iManufacturer  ]));
Log(Format('      iProduct           =%d', [iProduct        ]));
Log(Format('      iSerialNumber      =%d', [iSerialNumber  ]));
Log(Format('      bNumConfigurations=%d', [bNumConfigurations]));
```

End;

End;

\$02: begin // Дескриптор конфигурации

With TusbConfigurationDescriptor(PData^) do begin

```
Log(Format('      iConfiguration=%d', [iConfiguration]));
Log(Format('      bNumInterfaces=%d', [bNumInterfaces]));
Log(Format('      bmAttributes  =%d', [bmAttributes  ]));
Log(Format('      MaxPower(мА)  =%d', [MaxPower*2   ]));
```

End;

End;

\$03: begin

// Дескриптор строки

End;

\$04: begin

// Дескриптор интерфейса

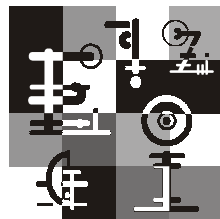
End;

```
$05: begin
    // Дескриптор конечной точки
End;

End; {end of case}
// переходим к следующему дескриптору
iData:= iData + lenDescr;
Until (lenDescr = 0) or (iData > High(Data));
end;
```

Каждый дескриптор отображается в соответствии с содержащимися в нем данными. Для получения доступа к содержимому дескриптора мы используем указатель на текущее положение в буфере данных, приводя его к соответствующему типу. Для перевода BCD-чисел в строку номера версии используется функция `Bcd2Str`, код которой можно найти в приложениях. Полный код этой программы содержится на компакт-диске.

Глава 5



Внутренняя организация хоста и хабов

Хаб — специальное устройство, позволяющее "сжечь" несколько портов одновременно.

5.1. Хабы

Хаб (Hub, другое название — *концентратор*) — устройство, которое обеспечивает дополнительные порты на шине USB. Другими словами, хаб преобразует один порт (*восходящий порт*, Upstream Port) в множество портов (*нисходящие порты*, Downstream Ports). Архитектура допускает соединение нескольких хабов (не более 5) (рис. 5.1).

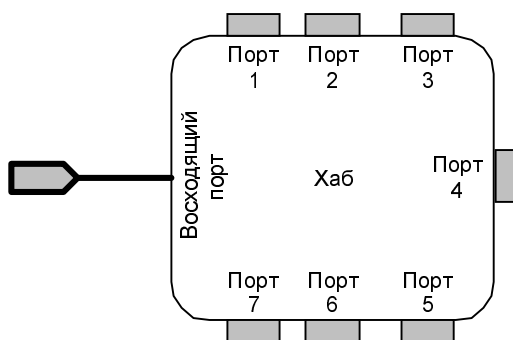


Рис. 5.1. Структурная схема хаба

Основные функции хабов:

- ☐ обеспечение физического подключения устройств к каждому из портов;
- ☐ распознавание подключения и отключения устройств к портам, уведомление порта об изменениях;
- ☐ управление питанием портов, ограничение тока, потребляемого каждым портом;

- ❑ конфигурирование портов;
- ❑ обеспечение изоляции сегментов с низкой скоростью от высокоскоростных;
- ❑ обнаружение ошибок на шине, изолирование неисправных сегментов шины.

Хаб USB 2.0 состоит из контроллера, повторителя и транслятора транзакций.

Контроллер хаба (Hub Controller) содержит регистры для взаимодействия с хостом. Доступ к регистрам осуществляется с помощью специальных команд, которые позволяют конфигурировать хаб, управлять его портами и получать состояние портов.

Повторитель (Hub Repeater) представляет собой управляемый коммутатор, соединяющий выходной порт с входным. Он имеет средства поддержки сброса и приостановки передачи сигналов. В хабе USB 1.1 коммутатор один (FS/LS), а в хабе USB 2.0 добавлен еще один коммутатор для скоростей HS. Спецификация второго коммутатора существенно отличается от первого. Который из коммутаторов используется, зависит от того, на какой скорости (FS или HS) хаб подключается к вышестоящему устройству.

Транслятор транзакций (Transaction Translator) служит для организации расщепленных транзакций, когда хаб подключен на HS, а к его портам подключены устройства или хабы FS/LS. Транслятор транзакций в хабе USB 1.1 отсутствует.

Хаб имеет внутренний генератор синхронизации, который синхронизируется с маркерами кадров (SOF), принимаемыми от восходящего порта. Одной из задач хаба является отслеживание маркеров SOF и выявление пропадания SOF, нарушения времени их появления и обнаружение других ошибок на шине.

Хаб следит за сигналами, генерируемыми устройствами. Неисправное устройство может не вовремя "замолчать" (потерять активность) или, наоборот, что-то "бормотать" (babble). Эти ситуации отслеживает ближайший к устройству хаб и запрещает восходящие передачи от такого устройства не позже, чем по границе кадра. Благодаря бдительности хабов эти ситуации не позволят неисправному устройству заблокировать всю шину.

Каждый из нисходящих (downstream) портов может быть разрешен или запрещен, а также сконфигурирован на высокую, полную или ограниченную скорость обмена. Нисходящие порты хабов имеют множество возможных состояний, из которых можно выделить несколько основных состояний:

- ❑ **Не сконфигурирован** (Not Configured) — состояние портов хаба с неустановленной конфигурацией. На линию выдается сигнал SE0;
- ❑ **Питание отключено** (Powered off) — на порт не подается питание. Выходные буферы переводятся в высокоимпедансное состояние, входные сигналы игнорируются. В это состояние порт переводится по команде кон-

троллера или при обнаружении перегрузки по питанию (потребление превышает заявленное). Если хаб конфигурировался при внешнем питании, то при его пропадании он должен перевести все порты в это состояние. Если хаб конфигурировался при питании от шины, то при переключении его на внешнее питание состояние портов не должно изменяться. Но если в дальнейшем внешнее питание пропадет, все порты должны переводиться в состояние "Питание отключено";

- ❑ **Отсоединен** (Disconnected) — порт не передает сигналы ни в одном направлении, но способен обнаружить подключение устройства. Тогда порт переходит в состояние "Запрещен", а по уровням входных сигналов он определяет скорость подключенного устройства;
- ❑ **Запрещен** (Disabled) — порт передает только сигнал сброса (по команде от контроллера), сигналы от порта (кроме обнаружения отключения) не воспринимаются. Терминация порта соответствует режиму FS/LS. В это состояние порт переводится по команде контроллера, а также при обнаружении ошибки. При обнаружении отключения порт переходит в состояние "Отсоединен", а если отключение обнаружено "спящим" хабом, контроллеру будет послан сигнал "Пробуждение" (Resume);
- ❑ **Разрешен** (Enabled) — порт передает сигналы в обоих направлениях (подключен к коммутатору). По команде контроллера или при обнаружении ошибки кадра порт переходит в состояние "Запрещен", а при обнаружении отключения — в состояние "Отсоединен";
- ❑ **Сброс** (Resetting) — порт по команде от контроллера подает сигнал сброса.
- ❑ **Приостановлен** (Suspended) — порт передает сигнал перевода в состояние останова (*"спящий режим"*), дифференциальные передатчики отключены. Терминация порта соответствует режиму FS/LS (но состояние порта может быть и HS). Если хаб находится в активном состоянии, сигналы через порт не пропускаются ни в одном направлении. Однако "спящий" хаб воспринимает сигналы смены состояния незапрещенных портов, подавая "пробуждающие" сигналы от активизировавшегося устройства даже через цепочку "спящих" хабов.

Состояние каждого порта идентифицируется контроллером хаба с помощью отдельных регистров. Имеется общий регистр, биты которого отражают факт изменения состояния каждого порта. Это позволяет хост-контроллеру быстро узнать состояние хаба, а в случае обнаружения изменений специальными транзакциями уточнить состояние. Хабы могут иметь световые индикаторы состояния нисходящих портов, управляемые автоматически (логикой хаба) или программно (хост-контроллером). Индикатор может представлять собой пару светодиодов — зеленый и желтый (янтарный), или один светодиод с изменяющимся цветом.

Состояние порта представляется следующим образом:

- ☐ не светится — порт не используется;
- ☐ зеленый — нормальная работа;
- ☐ желтый — ошибка;
- ☐ зеленый мигающий — программа требует внимания пользователя (Software Attention);
- ☐ желтый мигающий — аппаратура требует внимания пользователя (Hardware Attention).

Восходящий (upstream) порт хаба конфигурируется и внешне представляется как полноскоростной или высокоскоростной (только для USB 2.0). При подключении порт хаба USB 2.0 обеспечивает терминацию по схеме FS, в режим HS он переводится только по команде контроллера.

5.1.1. Взаимодействие хост-контроллера с хабом

Чтобы получить доступ к устройствам, подключенным к шине USB через хаб, хост должен сконфигурировать хаб и произвести настройку его портов. Процесс настройки хаба хост осуществляет путем определенной последовательности запросов.

Прежде всего, хост должен произвести идентификацию устройства по стандартным дескрипторам (см. разд. 4.1.2). Хаб можно опознать по дескрипторам устройства и интерфейса:

- ☐ в стандартном дескрипторе устройства (см. разд. 4.1.3) поле кода класса устройства содержит значение 09H, поле кода подкласса устройства — значение 00;
- ☐ в стандартном дескрипторе интерфейса (см. разд. 4.1.3) поле количества конечных точек имеет значение 01H, поле класса интерфейса содержит значение 09H, поле подкласса устройства — значение 00H, поле протокола — значение 00H.

Хаб имеет только одну входную конечную точку, которая работает в режиме передачи по прерываниям. Поле значения интервала обслуживания в дескрипторе конечной точки (см. разд. 4.1.3) содержит значение FFH. Время реакции хаба на стандартные запросы не должно превышать 50 мс.

Хаб поддерживает следующие стандартные запросы (см. разд. 4.1.2): GET_STATUS, CLEAR_FEATURE, SET_FEATURE, SET_ADDRESS, GET_DESCRIPTOR, SET_DESCRIPTOR, GET_CONFIGURATION, SET_CONFIGURATION. Реакция на запросы GET_INTERFACE и SET_INTERFACE не определена, т. к. хаб может иметь только один интерфейс. Реакция на запрос SYNC_FRAME не определена, т. к. хаб не может иметь изохронных конечных точек.

5.1.2. Дескриптор хаба

Кроме стандартных дескрипторов, по запросу может быть выдан специфический дескриптор хаба (Hub Descriptor), структура которого приведена в табл. 5.1, а описание на языках C и Pascal — в листинге 5.1.

Таблица 5.1. Структура дескриптора хаба

Смещение	Поле	Размер	Описание
0	bLength	1	Размер дескриптора в байтах
1	bDescriptorType	1	Тип дескриптора (29H)
2	bNbrPorts	1	Количество нисходящих портов
3	wHubCharacteristics	2	Характеристики хаба
5	bPowerOnToPowerGood	1	Время стабилизации напряжения
6	bHubContrCurrent	1	Величина тока (мА), потребляемого контроллером хаба
7	DeviceRemovable	K	Битовая карта подключения съемных устройств к портам
7 + K	PortPwrCtrlMask	K	Маска контроля питания портов

Листинг 5.1. Дескриптор хаба

```
// Описание на языке C
typedef struct _USB_HUB_DESCRIPTOR {
    UCHAR      bDescriptorLength; // Длина этого дескриптора
    UCHAR      bDescriptorType;   // Тип хаба
    UCHAR      bNumberOfPorts;    // Число портов хаба
    USHORT     wHubCharacteristics; // Характеристики
    UCHAR      bPowerOnToPowerGood; // Время стабилизации
    UCHAR      bHubControlCurrent; // Максимальное потребление (мА)
    UCHAR      bRemoveAndPowerMask[64]; // Маска
} USB_HUB_DESCRIPTOR, *PUSB_HUB_DESCRIPTOR;

// Описание на языке Pascal
THubDescriptor = packed record
    bDescriptorLength : Byte; // Длина этого дескриптора
    bDescriptorType   : Byte; // Тип хаба
    bNumberOfPorts    : Byte; // Число портов хаба
```

```
wHubCharacteristics : Word;    // Характеристики
bPowerOnToPowerGood : Byte;    // Время стабилизации
bHubControlCurrent  : Byte;    // Максимальное потребление
bRemoveAndPowerMask : Array [1..64] of Byte; // Маска
End;
```

Поле характеристик хаба (поле `wHubCharacteristics`) состоит из следующих битов:

- ☐ [15:5] зарезервированы;
- ☐ [4:3] режим защиты от перегрузок:
 - 00 — глобальная защита от перегрузки по сумме токов всех портов;
 - 10 — индивидуальная защита для каждого порта;
 - 01 — защита отсутствует;
 - 11 — защита отсутствует;
- ☐ [2] признак составного устройства:
 - 0 — хаб является самостоятельным устройством;
 - 1 — хаб входит в состав периферийного устройства;
- ☐ [1:0] логический режим управления питанием:
 - 00 — подача питания включается одновременно для всех портов;
 - 10 — возможно индивидуальное управление питанием для каждого порта;
 - 01, 11 — у хаба отсутствует схема управления энергией.

Поле `bPowerOnToPowerGood` содержит величину временного интервала от подачи команды включения питания до стабилизации напряжения на выходе порта, заданную с шагом 2 мс.

Битовая маска подключения съемных устройств (поле `DeviceRemovable`) по-битно соответствует каждому из портов:

- ☐ бит 0 зарезервирован;
- ☐ бит 1 соответствует порту 1;
- ☐ бит 2 соответствует порту 2 и т. д.

Если бит имеет значение 0, к порту подключено съемное устройство, если 1 — несъемное.

Последнее поле (`PortPwrCtrlMask`) содержит маску питания портов (каждому порту соответствует один бит маски). В спецификации USB 2.0 это поле не используется и оставлено для совместимости с USB 1.1 (все разряды заполнены единицами).

5.1.3. Запросы хабов

Спецификация USB определяет несколько запросов, специфических для хабов. Эти запросы формируются из кода запроса (табл. 5.2) и селектора свойств (табл. 5.3, 5.4).

Таблица 5.2. Запросы хабов

Код	Запрос	Описание
0	GET_STATUS	Определить состояние устройства
1	CLEAR_FEATURE	Сбросить свойство
2	GET_STATE	Получить состояние устройства
3	SET_FEATURE	Установить свойство
6	GET_DESCRIPTOR	Получить дескриптор
7	SET_DESCRIPTOR	Загрузить дескриптор

Таблица 5.3. Селекторы свойств хабов

Код	Запрос	Описание
0	C_HUB_LOCAL_POWER	Признак изменения состояния встроенного источника питания
1	C_HUB_OVER_CURRENT	Признак изменения состояния индикатора перегрузки по току

Таблица 5.4. Селекторы свойств портов

Код	Запрос	Описание
0	PORT_CONNECTION	К порту подключено устройство
1	PORT_ENABLE	Работа порта разрешена
2	PORT_SUSPEND	Порт находится в режиме ожидания
3	PORT_OVER_CURRENT	Перегрузка по току
4	PORT_RESET	Установлен сигнал сброса
8	PORT_POWER	Питание включено
9	PORT_LOW_SPEED	Порт работает в низкоскоростном режиме
16	C_PORT_CONNECTION	Признак изменения состояния подключения
17	C_PORT_ENABLE	Признак выполнения операции разрешения или запрета работы порта

Таблица 5.4 (окончание)

Код	Запрос	Описание
18	C_PORT_SUSPEND	Признак переключения из состояния ожидания в активный режим или наоборот
19	C_PORT_OVER_CURRENT	Признак изменения состояния индикатора перегрузки по току
20	C_PORT_RESET	Признак того, что сигнал сброса был установлен или снят

5.1.4. Запрос **CLEAR_HUB_FEATURE**

Запрос CLEAR_HUB_FEATURE используется для сброса признака состояния хаба, указанного значением селектора свойств.

Запрос имеет следующие параметры:

- ❑ bmRequestType = 00100000b;
- ❑ bRequest = CLEAR_FEATURE (01H);
- ❑ wValue — селектор свойств хаба (см. табл. 5.3);
- ❑ wIndex = 0;
- ❑ wLength = 0.

Передача данных по этому запросу не производится.

5.1.5. Запрос **CLEAR_PORT_FEATURE**

Запрос CLEAR_PORT_FEATURE используется для сброса признака состояния порта хаба, указанного значением селектора свойств.

Запрос имеет следующие параметры:

- ❑ bmRequestType = 00100011b;
- ❑ bRequest = CLEAR_FEATURE (01H);
- ❑ wValue — селектор свойств порта (см. табл. 5.4);
- ❑ wIndex — номер порта;
- ❑ wLength = 0.

Передача данных по этому запросу не производится.

Запрос CLEAR_PORT_FEATURE допускает использование следующих селекторов: PORT_ENABLE, PORT_SUSPEND, PORT_POWER, C_PORT_CONNECTION, C_PORT_ENABLE, C_PORT_SUSPEND, C_PORT_OVER_CURRENT, C_PORT_RESET.

Сброс свойства `PORT_SUSPEND` вызывает формирование сигнала пробуждения для порта, который находится в режиме ожидания.

После сброса свойства `PORT_ENABLE` работа порта будет запрещена.

Сброс свойства `PORT_POWER` вызывает отключение питания порта.

5.1.6. Запрос ***GET_BUS_STATE***

Запрос `GET_BUS_STATE` используется для диагностики порта с заданным номером.

Запрос имеет следующие параметры:

- ☐ `bmRequestType = 10100011b`;
- ☐ `bRequest = GET_STATE (02H)`;
- ☐ `wValue = 0`;
- ☐ `wIndex` — номер порта;
- ☐ `wLength = 1`.

По этому запросу хаб возвращает один байт, состоящий из следующих битов:

- ☐ [7:2] зарезервированы и равны 0;
- ☐ [1] значение сигнала на линии D+;
- ☐ [0] значение сигнала на линии D-.

5.1.7. Запрос ***GET_HUB_DESCRIPTOR***

Запрос `GET_HUB_DESCRIPTOR` позволяет хосту получить дескриптор хаба.

Запрос имеет следующие параметры:

- ☐ `bmRequestType = 10100000b`;
- ☐ `bRequest = GET_DESCRIPTOR (06H)`;
- ☐ `wValue` содержит тип дескриптора (29H) в старшем байте и индекс дескриптора (00H) в младшем байте;
- ☐ `wIndex = 0`;
- ☐ `wLength` — размер дескриптора в байтах.

По этому запросу хаб возвращает свой дескриптор, структура которого описана в *разд. 5.1.2*.

5.1.8. Запрос ***GET_HUB_STATUS***

Запрос `GET_HUB_STATUS` позволяет определить текущее состояние хаба.

Запрос имеет следующие параметры:

- ☐ `bmRequestType = 10100000b`;

- ❑ `bRequest = GET_STATUS (00H);`
- ❑ `wValue = 0;`
- ❑ `wIndex = 0;`
- ❑ `wLength = 4.`

По этому запросу хаб возвращает 16-разрядное слово состояния `wHubStatus` и 16-разрядное слово индикаторов изменения состояния `wHubChange`.

Слово состояния хаба `wHubStatus` имеет следующую структуру:

- ❑ [15:2] зарезервированы и равны 0;
- ❑ [1] содержит признак перегрузки порта по выходному току:
 - 0 — порт работает нормально;
 - 1 — подключенное к порту устройство потребляет слишком много;
- ❑ [0] содержит признак неисправности встроенного источника питания хаба:
 - 0 — рабочее состояние;
 - 1 — встроенный источник питания выключен и хаб получает питание от шины USB.

Слово индикаторов изменения состояния хаба `wHubChange` имеет следующую структуру:

- ❑ [15:2] зарезервированы и равны 0;
- ❑ [1] содержит признак изменения состояния индикатора перегрузки по току `C_HUB_OVER_CURRENT` (устанавливается в 1, если состояние индикатора перегрузки изменилось);
- ❑ [0] содержит признак изменения состояния встроенного источника питания `C_HUB_LOCAL_POWER` (устанавливается в 1, если состояние питания изменилось).

5.1.9. Запрос **GET_PORT_STATUS**

Запрос `GET_PORT_STATUS` позволяет определить текущее состояние выбранного порта хаба.

Запрос имеет следующие параметры:

- ❑ `bmRequestType = 10100011b;`
- ❑ `bRequest = GET_STATUS (00H);`
- ❑ `wValue = 0;`
- ❑ `wIndex` — номер порта;
- ❑ `wLength = 4.`

По этому запросу хаб возвращает 16-разрядное слово состояния порта `wPortStatus` и 16-разрядное слово индикаторов изменения состояния порта `wPortChange`.

Слово состояния порта `wPortStatus` имеет следующую структуру:

- ☐ [15:10] зарезервированы и равны 0;
- ☐ [9] содержит признак подключения LS-устройства `PORT_LOW_SPEED` (устанавливается в 1, если к порту подключено низкоскоростное устройство);
- ☐ [8] содержит индикатор состояния схемы управления энергией `PORT_POWER`:
 - 0 — питание на устройство не подается;
 - 1 — подача питания разрешена;
- ☐ [7:5] зарезервированы и равны 0;
- ☐ [4] содержит признак активности сигнала сброса `PORT_RESET` (устанавливается в 1, если на шине установлен сигнал сброса);
- ☐ [3] содержит индикатор перегрузки по току `PORT_OVER_CURRENT` (устанавливается в 1, если устройство потребляет от порта слишком большой ток);
- ☐ [2] содержит признак состояния ожидания `PORT_SUSPEND` (устанавливается в 1, если порт находится в режиме ожидания);
- ☐ [1] содержит признак разрешения работы `PORT_ENABLE`:
 - 0 — работа порта запрещена;
 - 1 — работа порта разрешена;
- ☐ [0] содержит признак подключения `PORT_CONNECTION`:
 - 0 — порт свободен;
 - 1 — к порту подключено устройство.

Слово индикаторов изменения состояния порта `wPortChange` имеет следующую структуру:

- ☐ [15:5] зарезервированы и равны 0;
- ☐ [4] содержит индикатор изменения сигнала сброса `C_PORT_RESET` (устанавливается в 1 после установки или снятия сигнала сброса);
- ☐ [3] содержит индикатор изменения сигнала перегрузки по току `C_PORT_OVER_CURRENT` (устанавливается в 1, если состояние сигнала перегрузки изменилось);
- ☐ [2] содержит индикатор изменения состояния ожидания `C_PORT_SUSPEND` (устанавливается в 1, если порт был переключен из состояния ожидания в активный режим работы или наоборот);

- ❑ [1] содержит индикатор изменения признака разрешения работы порта C_PORT_ENABLE (устанавливается в 1, если работа порта была разрешена или запрещена);
- ❑ [0] содержит индикатор изменения состояния подключения C_PORT_CONNECTION (устанавливается в 1, если произошло подключение или отключение устройства).

5.1.10. Запрос **SET_HUB_DESCRIPTOR**

Запрос SET_HUB_DESCRIPTOR позволяет хосту заменить дескриптор хаба.

Запрос имеет следующие параметры:

- ❑ bmRequestType = 00100000b;
- ❑ bRequest = SET_DESCRIPTOR (07H);
- ❑ wValue содержит тип дескриптора (29H) в старшем байте и индекс дескриптора (00H) в младшем байте;
- ❑ wIndex — номер порта;
- ❑ wLength — размер дескриптора в байтах.

По этому запросу хост передает хабу дескриптор, структура которого описана в *разд. 5.1.2*.

5.1.11. Запрос **SET_HUB_FEATURE**

Запрос SET_HUB_FEATURE используется для того, чтобы установить признак состояния хаба, указанный значением селектора свойств.

Запрос имеет следующие параметры:

- ❑ bmRequestType = 00100000b;
- ❑ bRequest = SET_FEATURE (03H);
- ❑ wValue — селектор свойств хаба (см. табл. 5.3);
- ❑ wIndex = 0;
- ❑ wLength = 0.

Передача данных по этому запросу не производится.

5.1.12. Запрос **SET_PORT_FEATURE**

Запрос SET_PORT_FEATURE используется для того, чтобы перевести порт хаба в состояние, указанное значением селектора свойств.

Запрос имеет следующие параметры:

- ❑ bmRequestType = 00100011b;
- ❑ bRequest = SET_FEATURE (03H);

- ❑ `wValue` — селектор свойств порта (см. табл. 5.4);
- ❑ `wIndex` — номер порта;
- ❑ `wLength` = 0.

Передача данных по этому запросу не производится.

Запрос `SET_PORT_FEATURE` допускает использование следующих селекторов: `PORT_ENABLE`, `PORT_SUSPEND`, `PORT_POWER`, `C_PORT_CONNECTION`, `C_PORT_ENABLE`, `C_PORT_SUSPEND`, `C_PORT_OVER_CURRENT`, `C_PORT_RESET`.

После установки свойства `PORT_SUSPEND` порт и подсоединенные к нему устройства переводятся в состояние ожидания.

После установки свойства `PORT_ENABLE` работа порта будет разрешена.

После установки свойства `PORT_POWER` будет включено питание порта.

5.2. Совместная работа устройств с разными скоростями

Спецификация USB 1.x описывает две скорости работы устройств (LS и FS), а спецификация USB 2.0 предусматривает повышение пропускной способности до 480 Мбит/с.

Спецификация USB гарантирует работу USB-устройств с разной скоростью на одной шине. Разумеется, если бы низкоскоростные устройства работали "напрямую", то хост (или хаб) тратил бы значительную часть времени на обмен с этими устройствами, точнее на ожидание маркера SOF от таких устройств.

Для решения проблем совместимости введены два ключевых решения:

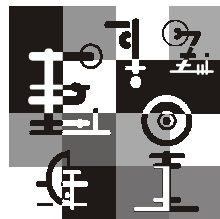
- ❑ (USB 2.0) в состав хаба добавлен *транслятор транзакций*, который буферизирует поступающий с медленного порта кадр, а затем на максимальной скорости передает его хосту, или же буферизирует получаемый на максимальной скорости кадр от хоста, передавая его затем устройству на меньшей, приемлемой для него скорости;
- ❑ хаб USB не транслирует трафик на свои нисходящие порты, к которым подключены низкоскоростные устройства, до тех пор, пока хост-контроллер не передаст специального маркера-преамбулы низкоскоростного обмена (PRE).

Маркер PRE игнорируется всеми устройствами, кроме хабов. Пакетом-преамбулой хост-контроллер гарантирует, что следующий пакет будет им передан на низкой скорости. Этим пакетом будет маркер, определяющий тип транзакции с LS-устройством, а в транзакциях вывода — и пакет данных (перед которым требуется своя преамбула). Хаб разрешает транслиро-

вать на свой нисходящий порт с LS-устройством только один пакет, следующий за преамбулой; по концу пакета (увидев EOP на низкой скорости) он снова запрещает трансляцию. Чтобы хаб успел переключить режим своего приемопередатчика, между преамбулой и последующим пакетом вводится зазор (4 битовых интервала FS). Для ответа LS-устройства никаких преамбул не нужно — хабы способны прозрачно передавать восходящий трафик на обеих скоростях (LS и FS). Хост-контроллер, естественно, должен принимать пакеты и на FS, и на LS. Очевидно, что низкоскоростные транзакции расходуют время кадра весьма неэффективно, но в USB 1.x с этим мирятся ради возможности подключения дешевых устройств и упрощения хабов, которые являются просто повторителями сигналов. Заметим, что маркеры SOF не транслируются на низкоскоростные порты, так что изохронный обмен, для которого они необходимы, для LS-устройств невозможен и не поддерживается.

Хост-контроллер USB 2.0 фактически содержит в себе два контроллера: ЕНС для работы с HS-устройствами и контроллер UHC (или ОНС) для работы с LS-устройствами и FS-устройствами. Корневой хаб имеет равноправные порты, но в процессе конфигурирования в зависимости от свойств подключенного к нему устройства (или хаба) каждый порт соединяется с соответствующим контроллером.

Глава 6



USB без ПК

Если хочешь ты меня полюбить,
Просто так или с USB
И, может быть, мы сразу друг друга поймем,
Если у нас один и тот же разъем.

"Желтая луна", Аквариум

Спецификация USB предполагает четкие отношения между хостом и устройствами, определяя этим модель "главный-подчиненные". Однако очень часто возникают ситуации, когда хотелось бы иметь прямое соединение между устройствами, избегая передачи данных через компьютер. Например, печатать фотоснимки напрямую с цифрового фотоаппарата на принтер или обмениваться музыкальными файлами между CD-плеерами. Еще одна проблема периферии — миниатюрность многих устройств не позволяет использовать стандартные USB-разъемы.

Для решения этих проблем в декабре 2001 года была опубликована новая спецификация, названная OTG (USB On-The-Go, USB "на ходу")¹. Новая спецификация решает следующие задачи:

- ❑ специфицирование миниатюрных разъемов и соответствующих кабелей;
- ❑ дополнение устройств, традиционно играющих роль периферии персональных компьютеров, способностью работать в режиме хоста (point-to-point connection);
- ❑ определение *двухролевого устройства* (dual-role devices), имеющего возможность работать попеременно в режиме хоста и в режиме периферии;
- ❑ ограничение потребления питания от шины до 100 мА для устройств, работающих от батареек.

Спецификация OTG определяет дополнительные разъемы и кабели (см. разд. 2.1).

¹ Полное название спецификации "On-The-Go Supplement to the USB 2.0 Specification"

6.1. Разъемы OTG

Спецификация OTG добавляет 5-контактные вилки mini-A и универсальное 5-контактное гнездо mini-AB (рис. 6.1). Внутри вилки mini-A контакты 4 и 5 электрически соединены, в вилке mini-B контакт 4 свободен. Для облегчения различения разъемов принята цветовая маркировка: разъемы mini-A должны быть белого цвета, mini-B — черного, а гнезда mini-AB — серого.



Рис. 6.1. Дополнительный разъем mini-A

В гнездо mini-AB двухролевого устройства может вставляться как вилка mini-A, так и вилка mini-B. При этом контакт 4 (ID) используется для идентификации типа подключенного устройства.

- ❑ Если контакт 4 (ID) соединен с линией GND (сопротивление менее 10 Ом), то вставлена вилка mini-A. Значит, подключено устройство B; следовательно, двухролевое устройство должно стать хостом.
- ❑ Если контакт 4 (ID) не соединен с линией GND (сопротивление более 10 Ом), то вставлена вилка mini-B. Значит, подключено устройство A; следовательно, двухролевое устройство должно стать периферийным.

6.2. Типы OTG-устройств

Спецификация OTG определяет три типа устройств:

- ❑ *Устройство A (A-Device)* — устройство, в гнездо которого вставлена вилка типа A (или mini-A). Это устройство подает питание на шину и играет роль хоста, по крайней мере, в первое время после подключения к другому устройству. По ходу сеанса связи устройство A может передать функции хоста своему партнеру, а само стать периферийным.
- ❑ *Устройство B (B-Device)* — устройство, в гнездо которого вставлена вилка типа B (или mini-B). Это устройство при подключении к другому устройству играет роль периферийного (ведомого) устройства USB. Если это устройство является двухролевым, то по ходу сеанса связи ему могут быть переданы функции хоста.
- ❑ *Двухролевое устройство (Dual-role device)* — устройство с единственным гнездом типа mini-AB, обеспечивающее питание шины с током не менее

8 мА, поддерживающее FS (дополнительно может поддерживать и HS, а в роли периферийного устройства и LS). Это устройство имеет усеченные возможности хоста, список поддерживаемых периферийных устройств, средства диалога с пользователем. Для управления связью устройство должно поддерживать протоколы запроса сессий и согласования роли хоста.

Протокол запроса сеанса SRP (Session Request Protocol) предназначен для дополнительного энергосбережения: когда устройство-А не нуждается в обмене по шине, оно может снять питание Vbus. При этом устройство-В все-таки может "попросить внимания" — запросить сеанс связи. Здесь сеансом называется интервал времени, в течение которого двухролевое устройство подает допустимое (для работы) напряжение питания. Запрос может выполняться подачей положительных импульсов либо по линии Vbus, либо по сигнальным линиям (D+ или D−). Устройство-В должно использовать оба метода подачи запроса, устройство-А может распознавать любой из них (как удобнее его разработчику).

Протокол согласования роли хоста HNP (Host Negotiation Protocol) позволяет устройству-А и устройству-В поменяться ролями во время сеанса связи (если они оба двухролевые). Протокол может быть инициирован, только если устройство-А пошлет устройству-В специальный разрешающий запрос, предварительно убедившись, что устройство-В поддерживает протокол HNP. Возможность поддержки протоколов HNP и SRP сообщается устройством-В в специальном дескрипторе OTG-устройства.

Устройство-В может запросить управление шиной (стать хостом на время), когда устройство-А прекращает активность (переводит шину в состояние покоя). Для этого устройство-В отключается от шины (отключает свой "подтягивающий" резистор от линии D+). Устройство-А расценивает это как запрос смены роли и подключает свой "подтягивающий" резистор к линии D+. Теперь устройство-В может начинать транзакции, управляя шиной. Когда оно захочет отдать управление шиной, оно прекращает активность и подключает свой "подтягивающий" резистор к линии D+. Устройство-А расценивает это как возврат управления и отключает свой "подтягивающий" резистор от линии D+ — исходные роли, определенные по типу разъема, восстановлены.

6.3. Дескриптор OTG-устройства

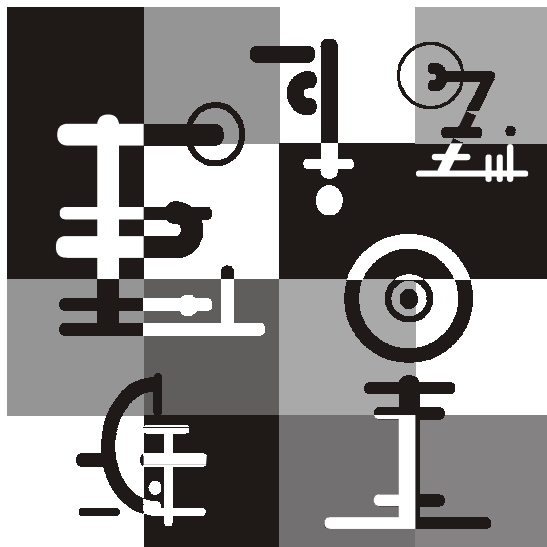
Дескриптор OTG (длина 3, тип 9) должен присутствовать во всех конфигурациях OTG-устройства, он считывается обычным запросом GET_DESCRIPTOR. Дескриптор OTG содержит лишь один байт атрибутов, в котором бит 0 указывает на поддержку SRP, бит 1 — на поддержку HNP (остальные биты нулевые).

Убедившись в поддержке протокола HNP, устройство А еще до выбора конфигурации устройства В должно сообщить свое отношение к HNP. Для этого служат запросы к устройству SET_FEATURE (bmRequestType = 00000000b, bRequest = 3):

- ❑ Set b_hnp_enable (wValue = 3) — устройство А разрешает устройству В запрашивать роль хоста;
- ❑ Set a_hnp_support (wValue = 4) — устройство А только информирует устройство В о том, что оно подключено к порту, поддерживающему HNP, и запрос роли хоста может быть разрешен позже;
- ❑ Set a_alt_hnp_support (wValue = 5) — устройство А информирует устройство В о том, что оно подключено к порту, не поддерживающему HNP, но у устройства А имеется другой порт, на котором HNP поддерживается.

6.4. Интернет-ресурсы к этой главе

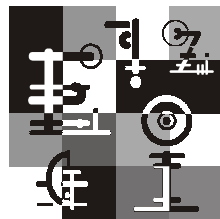
- ❑ USB On-The-Go (OTG), версия 0.9 или USB для маленьких
 - <http://www.ixbt.com/editorial/otg.shtml>
- ❑ Новые контроллеры интерфейса USB для взаимодействия устройств в отсутствие компьютера
 - <http://www.asutp.ru/?p=202209>



ЧАСТЬ III

ПРАКТИКА ПРОГРАММИРОВАНИЯ

Глава 7



Поддержка USB в Windows

Windows: Вы действительно хотите удалить этот файл?

Пользователь: Да!

Windows: А почему?

"Старые" интерфейсы, такие как COM (последовательный коммуникационный порт) и LPT (параллельный порт) имели довольно простой набор функций для работы. Действительно, достаточно было открыть порт с помощью вызова функции `CreateFile` — и можно было читать и передавать данные. Причем открытие порта производилось по имени (листинг 7.1).

Листинг 7.1. Работа со "старыми" интерфейсами

```
// Работа с последовательным интерфейсом в Windows
// Открытие порта по имени
FHandle:= CreateFile('COM1',...);
// Чтение данных с порта
ReadFile(FHandle, Buffer, SizeOf(Buffer), ReallyRead, nil);
// Закрытие порта
CloseHandle(FHandle);

// Работа с последовательным портом в DOS
// Инициализация порта
Port[BaseAdr+3]:= $80;
Port[BaseAdr+0]:= FreqL;
Port[BaseAdr+1]:= FreqH;
Port[BaseAdr+3]:= Params;
// Проверка готовности данных
If (Port[BaseAdr+5] and 1) <> 0 then begin
    // Чтение данных с порта
    B:= Port[BaseAdr+0];
End;
```

С одной стороны, такая простота удобна, но, с другой, — создание надежного протокола требует множества дополнительных усилий как для обеспечения гарантированной доставки данных, так и для обнаружения ошибок обмена. Обеспечение распознавания начала и конца посылок, подсчет и проверка контрольных сумм, повторение недошедших посылок и т. д. — все это программист должен реализовывать самостоятельно. А поддержка Plug and Play и автоматического конфигурирования делает использование таких интерфейсов серьезной проблемой. Интерфейс USB обеспечивает надежную доставку данных, предоставляет множество сервисных функций для конфигурирования устройства, другими словами, реализует весь необходимый набор действительно надежного протокола обмена. Разумеется, за это нужно платить, а именно — реализация USB-протокола очень сложна, поэтому операционная система делит реализацию протокола на уровни (см. разд. 3.2).

7.1. Модель WDM

В Windows 2000/XP драйверная поддержка реализуется на основе WDM (см. главу 9), которая позволяет распределить процесс передачи данных (рис. 7.1).

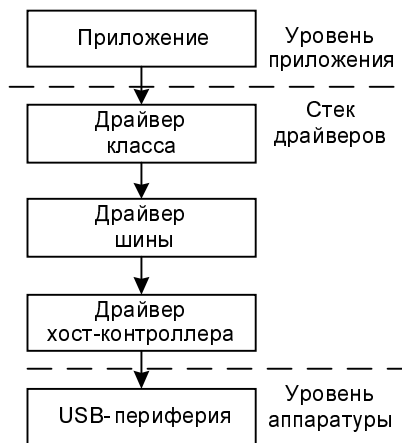


Рис. 7.1. WDM-модель для USB-интерфейса

Два низкоуровневых драйвера входят в поставку Windows, избавляя программиста от множества рутинной работы:

- драйвер хост-контроллера (Host Controller Driver) отвечает за обмен данными с аппаратурой;

❑ драйвер шины (USB Bus Driver) отвечает за управление транзакциями, питанием и распознавание устройств.

С точки зрения прикладного программиста, наибольший интерес представляют драйвер класса (Class Driver) и интерфейс обращения к этому драйверу. Здесь операционная система Windows делает еще один шаг на пути унификации интерфейсов. Все USB-устройства делятся на группы, согласно общим свойствам, выполняемым функциям и требованиям к ресурсам. Для каждой группы устройств Windows предоставляет готовый драйвер, который автоматически устанавливается при обнаружении принадлежности устройства к одной из групп. Таким образом, в большинстве случаев никаких дополнительных драйверов не требуется.

Список основных поддерживаемых на данный момент типов приведен в табл. 7.1.

Таблица 7.1. Список основных поддерживаемых типов устройств (Windows 2000/XP)

Класс	Драйвер	Примечания
Хабы (Hub Device)	hubclass.sys	Хабы
HID-устройства (Human Interface Device)	hidclass.sys	Мышки, джойстики, клавиатуры (см. главу 8)
Аудио (Audio)	sysaudio.sys	Звуковые колонки, виртуальные MIDI-устройства
Устройства хранения данных (Mass Storage Device)	usbstor.sys	Флеш-диски
Принтеры (Printer)	usbprint.sys	Принтеры
Устройства коммуникации (Communication)	mdismp.sys, usb8023.sys, другие	Модемы, сетевые карты

Для работы с нестандартными устройствами можно также воспользоваться одним из системных драйверов. Чаще всего пользуются HID-драйвером, работу с которым мы рассмотрим в *разд. 8.8*. Если же стандартный драйвер не подходит, необходимо писать собственный драйвер поддержки. Структура драйверов обсуждается в *главе 9*, а пример написания драйвера приведен в *главе 14*.

7.2. Взаимодействие с USB-драйвером

Для обращения к драйверам используется функция `DeviceIoControl`. В главе 9 мы будем более подробно обсуждать принцип ее работы, а сейчас посмотрим, как с ее помощью обращаться к драйверу USB.

Функция `DeviceIoControl` должна принимать следующие параметры:

- ☐ дескриптор хоста, порта или устройства, полученный с помощью вызова `CreateFile`;
- ☐ код вызываемой функции;
- ☐ входной буфер и его размер;
- ☐ выходной буфер и его размер;
- ☐ указатель на переменную, в которую будет возвращено число реально переданных (или прочитанных) байтов.

Первый параметр — дескриптор объекта, к которому производится обращение. Листинг 7.2 показывает пример получения дескриптора хоста.

Листинг 7.2. Получение дескриптора хоста

```
var sHost : String; iHost : Integer; hHost : THandle;
    SA : SECURITY_ATTRIBUTES;
begin
    // Формирование структуры SA
    SA.nLength:= SizeOf(SEcurity_ATTRIBUTES);
    SA.lpSecurityDescriptor:= nil;
    SA.bInheritHandle:= False;

    // Номер хоста
    iHost:= 0;
    // Формирование имени хоста
    sHost:= Format('\\.\HCD%d', [iHost]);
    // Получение дескриптора хоста
    hHost:= CreateFile(PChar(@sHost[1]),
        GENERIC_WRITE, FILE_SHARE_WRITE, @SA, OPEN_EXISTING, 0, 0
    );

    // Хост открыт?
    If (hHost <> INVALID_HANDLE_VALUE) then begin
        Log(Format('Хост-контроллер %d успешно открыт', [iHost]));
```

```

    CloseHandle(hHost);
End else begin
    Log(Format('Хост-контроллер <%s> не найден', [sHost]));
End;
End;

```

Для формирования кода функции используется макрос CTL_CODE, код которого показан в листингах 9.14 и 9.16. Все коды стандартных запросов (см. разд. 4.1.2) отсчитываются от константы USB_IOCTL_INDEX. Код функции для формирования кода стандартного запроса показан в листинге 7.3.

Листинг 7.3. Формирование кода стандартного запроса

```

Const
    FILE_DEVICE_USB = $00000022;
    METHOD_BUFFERED = $00000000;
    FILE_ANY_ACCESS = $00000000;
    USB_IOCTL_INDEX = $00FF;

function GetUSBCtlCode(Code : Integer): Integer;
begin
    Result:=
        (FILE_DEVICE_USB shl 16) or
        (FILE_ANY_ACCESS shl 14) or
        ((USB_IOCTL_INDEX + Code) shl 2) or
        METHOD_BUFFERED;
end;

```

Для выполнения стандартного запроса используется структура TSetupPacket (см. листинг 4.1), дополненная необходимой для данного запроса информацией. Например, для получения дескрипторов устройства (стандартного дескриптора или дескриптора конфигурации) используется структура TDescriptorRequest, как показано в листинге 7.4.

Листинг 7.4. Пример стандартного запроса

```

// структура для запроса дескрипторов
TDescriptorRequest = packed record
    ConnectionIndex : ULONG;
    SetupPacket      : TSetupPacket;

```

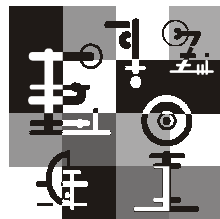
```
Data          : Array [1..2048] of Byte;
End;

// переменные для выполнения DeviceIoControl
Var
    Success : LongBool;
    Packet : TDescriptorRequest;
    BytesReturned : Cardinal;

// Получение стандартного дескриптора устройства
Begin
    ZeroMemory(@Packet, SizeOf(Packet));
    Packet.ConnectionIndex      := iPort+1;
    Packet.SetupPacket.bmRequest := $80;
    Packet.SetupPacket.bRequest  := USB_REQUEST_GET_DESCRIPTOR;
    Packet.SetupPacket.wValue [2] := USB_DEVICE_DESCRIPTOR_TYPE;
    Packet.SetupPacket.wLength[2] := 1; // использовать буфер 2 Кбайта

    // IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION
    Success:= DeviceIoControl(hRoot, GetUSBCtlCode(5),
        @Packet, sizeof(Packet),
        @Packet, sizeof(Packet),
        BytesReturned, nil
    );
End;
```


Глава 8



HID-устройства

Новая акция в нашем магазине:
купите у нас 1000 мегабайт оперативной памяти
и 24 мегабайта вы получите абсолютно бесплатно!

Согласно спецификации, HID (Human Interface Device) — это устройство связи с пользователем:

- ☐ клавиатуры и указатели, например, мышь, трекбол, джойстик;
- ☐ устройства контроля, такие как кнопки управления, переключатели, задвижки;
- ☐ устройства контроля в телефонах, видеомagnetофонах и игровых приставках, например, рулевое управление, игровые педали;
- ☐ устройства, не требующие взаимодействия с человеком, но передающие данные в HID-формате, например, считыватели штрихкода, термометры, вольтметры.

Вообще говоря, HID позволяет написать низкоскоростной двухсторонний обмен с любым устройством, даже не попадающим под жесткое определение устройства ввода. Поскольку Windows 98/2000/XP имеют встроенные HID-драйверы (см. табл. 7.1), то необходимость трудоемкой разработки драйвера отпадает. С точки зрения разработчика программы самого устройства, необходимо поддерживать ряд структур, описывающих HID-интерфейс (см. главу 13), а также обеспечивать обмен по каналу данных прерываний (см. разд. 3.3). Основным ограничением является скорость обмена. Максимально достигаемая скорость составляет 64 Кбит/с, что значительно меньше, чем полная скорость USB-шины — 12 Мбит/с, хотя для многих приложений, например, управления и индикации, вполне достаточно.

8.1. Свойства HID-устройства

Разработка спецификации HID-устройств была основана на следующих требованиях:

- ☐ максимально компактный программный код, требуемый со стороны устройства;

- ❑ возможность ПО пропускать незнакомую информацию, поступающую от устройства;
- ❑ протокол должен быть наращиваемым и устойчивым;
- ❑ устройство должно само описывать свои свойства, позволяя создавать базовое (классовое) программное обеспечение.

HID-устройством может быть любое устройство, способное функционировать согласно правилам, определенным спецификацией:

- ❑ полноскоростное HID-устройство может передавать вплоть до 60 000 байт в секунду, т. е. 64 байта в каждом кадре 1 мс (см. *разд. 3.5*), низкоскоростное — 800 байт в секунду, т. е. 8 байт каждые 10 мс;
- ❑ HID-устройство может устанавливать частоту своего опроса для выяснения, имеет ли устройство новые данные для пересылки;
- ❑ весь обмен с HID-устройством происходит с помощью определенной структуры, которая называется *репортом* (Report). Один репорт может содержать до 65 535 байт данных. Микропрограмма устройства должна содержать *дескриптор репорта* (Report Descriptor), который описывает структуру данных репорта. Репорт имеет достаточно гибкую структуру для описания любого типа устройства и формата передачи данных;
- ❑ HID-устройство должно иметь конечную точку типа Interrupt IN (см. *разд. 3.6*) для выдачи данных в хост. Дополнительно устройство может иметь конечную точку типа Interrupt OUT для получения периодических данных от хоста;
- ❑ HID-устройство должно содержать дескриптор класса и один или более дескрипторов репорта;
- ❑ HID-устройство должно поддерживать специфический для класса управляющий запрос Get_Report, а также опционально поддерживать дополнительный запрос Set_Report;
- ❑ для передачи данных из устройства в хост устройство должно положить данные репорта в буфер соответствующей конечной точки и разрешить передачу. При получении данных от хоста устройство должно разрешить соответствующую конечную точку, а затем, после прихода пакета, забрать данные из буфера.

HID-устройство должно иметь один или более дескрипторов репорта, которые запрашиваются только после того, как хост определил, что подключенное USB-устройство относится к классу HID.

Обмен между хостом и устройством может производиться с помощью трех видов репортов:

- ❑ **Input** и **Output** (входные и выходные репорты) используются для передачи и приема периодических данных, например, нажатий клавиш;

- ❑ **Feature** (специальные репорты) используются там, где очень важно время доставки, например, для установки различных параметров устройства и его инициализации.

Обмен данными с хостом (точнее, с HID-драйвером) производится либо по основному каналу сообщений (канал нулевой конечной точки), либо по каналу прерываний (рис. 8.1).

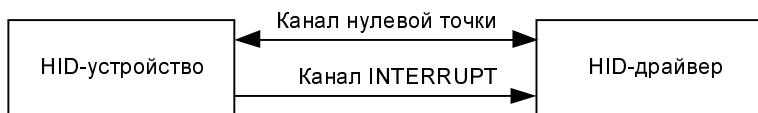


Рис. 8.1. Каналы обмена HID-устройства и драйвера

Канал нулевой точки используется для следующих операций:

- ❑ приема и передачи управляющих посылок;
- ❑ передачи данных с помощью запроса `Get_Report` (будет обсуждаться дальше);
- ❑ приема данных от хоста.

Канал прерываний используется для передачи асинхронных (не требуемых хостом) данных: данные с конечной точки читаются только в том случае, если устройство подтверждает наличие новых данных и необходимость их передачи.

Спецификация HID определяет два типа HID-устройств: устройства, участвующие в начальной загрузке и не участвующие. Первый тип называется загрузочные устройства (*boot device*). К загрузочным устройствам относятся, например, мышь и клавиатура, работа которых начинается с самого начала включения компьютера. Загрузочные устройства должны поддерживать специальные запросы (см. табл. 8.2).

8.2. Порядок обмена данными с HID-устройством

Когда хост запрашивает входной репорт, устройство выдает пакет данных с помощью передачи по прерыванию (т. е. передачи типа `INTERRUPT`, см. *разд. 3.3*). Периодичность генерации таких запросов указывается в дескрипторе конечной точки.

При генерации выходных репортов хост посылает данные в устройство, используя управляющие посылки (`Control Transfers`) или передачу по прерываниям (`Interrupt Transfers`). Возможность проводить передачи по прерыванию

в HID-устройство доступна только начиная с Windows 98 SE, а более ранние версии Windows 98 будут использовать для выходных репортов управляющие передачи. Если HID-устройство не имеет конечной точки с типом Interrupt Output, то драйвер ОС будет использовать управляющие посылки.

Специальные репорты (т. е. репорты типа Feature) имеют направление передачи данных как от хоста к устройству, так и от устройства в хост. Для них всегда используются управляющие посылки. Для того чтобы послать репорт этого типа, хост инициирует запрос Set_Report, предшествующий пакету данных, а далее, в фазе статуса, хост принимает от устройства подтверждение об успешном либо неуспешном принятии данных. Для того чтобы получить специальный репорт, хост инициирует запрос Set_Report, устройство при этом отвечает пакетом данных, а в фазе статуса хост возвращает в устройство информацию об успешно проведенной транзакции. Еще одно преимущество специальных репортов — это возможность задавать каждому репорту его номер (Report ID). При этом у программиста появляется возможность мультиплексировать запросы, если существует необходимость создания интерфейса передачи команд управления и данных через нулевую конечную точку (см. *разд. 13.8*).

8.3. Установка HID-устройства

Для установки устройства Менеджер Устройств операционной системы использует INF-файлы (см. *главу 10.4*) для решения о том, какой драйвер назначить найденному устройству. HID-устройство может использовать встроенный в ОС INF-файл (hiddev.inf для Windows 98 и input.inf для Windows 2000). Альтернативно программист может использовать свои собственные INF-файлы, в которых будет прописана информация о производителе устройства. Преимущество своего INF-файла состоит в отображении понятного названия устройства в окне Менеджера Устройств вместо общего термина "Стандартное устройство".

8.4. Идентификация HID-устройства

Устройство HID класса идентифицируется кодом класса 3 в дескрипторе интерфейса и имеет два специфических дескриптора: HID-дескриптор и дескриптор репорта (на самом деле существует еще физический дескриптор устройства (Physical Descriptor), но мы его рассматривать не будем). HID-дескриптор включается в список дескрипторов при запросе конфигурации. Обмен репортами с устройством возможен только после того, как драйвер определит тип данного устройства и какие интерфейсы оно поддерживает.

8.4.1. Идентификация загрузочных устройств

Идентификация типа устройства производится по полю `bInterfaceSubClass` в дескрипторе интерфейса:

- ☐ 0 — подкласс не указан (не загрузочный интерфейс);
- ☐ 1 — загрузочный интерфейс;
- ☐ 2—255 — зарезервировано.

Таким образом, устройство может иметь одновременно и загрузочный интерфейс, и обычный. Например, USB-клавиатура может работать при загрузке в стандартном режиме (данные обрабатывает BIOS), а при загрузке HID-драйвера добавлять специальные клавиши.

Если устройство определено как загрузочное, то поле `bInterfaceProtocol` в дескрипторе интерфейса обозначает поддерживаемый устройством стандартный протокол:

- ☐ 0 — протокол определяется пользовательским HID-репортом;
- ☐ 1 — клавиатура;
- ☐ 2 — мышь;
- ☐ 3—255 — зарезервировано.

Для незагрузочных устройств поле `bInterfaceProtocol` должно быть равно 0.

Примечание

Если устройство определено как клавиатура или мышь, Windows позволяет открывать устройство с помощью функции `CreateFile`, запрашивать дескрипторы устройства и конечных точек, но чтение данных с помощью функции `ReadFile` блокируется. Единственная возможность обмена данными с такими устройствами — функции `Get_Feature` и `Set_Feature`, конечно, при условии, что устройство поддерживает эти интерфейсы.

8.4.2. Дескриптор конфигурации HID-устройства

По запросу `Get_Descriptor(Configuration)` HID-устройство должно передавать хосту дескриптор конфигурации, все дескрипторы интерфейсов, все дескрипторы конечных точек и все HID-дескрипторы. Ни строковые дескрипторы (String descriptor), ни дескрипторы репортов (Report descriptor) не должны включаться в этот список. HID-дескриптор должен располагаться между дескриптором интерфейса и дескрипторами конечных точек:

Configuration descriptor

Interface descriptor (specifying HID Class)

HID descriptor (associated with above Interface)

Endpoint descriptor (for HID Interrupt Endpoint)

Дескрипторы `HID` и `REPORT` возвращаются по запросу `Get_Descriptor` со специальными кодами запроса (см. далее). Важно понимать, что дескриптор конфигурации относится к устройству, дескриптор интерфейса и `HID` — к интерфейсу, а дескриптор конечной точки — к конечной точке. Это довольно очевидное утверждение оказывается важным при обработке запросов на получение дескрипторов, т. к. поле `bmRequestType` имеет разное значение для каждого из типов. Именно по этой причине в списке объединенных кодов запросов (см. листинг 4.4) присутствуют три разных идентификатора запросов `Get_Descriptor` (см. разд. 4.1.1):

```
#define GET_DESCRIPTOR_DEVICE    0x8006
#define GET_DESCRIPTOR_INTERF    0x8106
#define GET_DESCRIPTOR_ENDPNT    0x8206
```

Однако при обработке запроса эти коды удобно объединить и обрабатывать в одной процедуре:

```
switch (wRequest)
{
... ..
case GET_DESCRIPTOR_DEVICE:
case GET_DESCRIPTOR_INTERF:
case GET_DESCRIPTOR_ENDPNT:
    usb_get_descriptor();
    break;
... ..
}
```

8.4.3. HID-дескриптор

Формат `HID`-дескриптора показан в табл. 8.1, а его описание на языках `C` и `Pascal` — в листинге 8.1.

Таблица 8.1. Структура `HID`-дескриптора

Сме- щение	Поле	Размер	Описание
0	bLength	1	Размер дескриптора в байтах
1	bDescriptorType	1	Тип дескриптора (\$21)
2	bcdHID	2	Версия <code>HID</code>
4	bCountryCode	1	Числовой код страны для локализованных устройств
5	bNumDescriptor	1	Число дескрипторов репортов

Таблица 8.1 (окончание)

Смещение	Поле	Размер	Описание
6	bReportType	1	Номер дескриптора репорта, используемый при вызове Set_Descriptor
7	wReportLength	2	Размер дескриптора репорта
9	bReportType	1	Номер дополнительного дескриптора
10	wReportLength	2	Размер дополнительного дескриптора

Листинг 8.1. HID-дескриптор

```
// Описание на языке C
typedef struct _USB_HID_DESCRIPTOR
{
    UCHAR    bLength;
    UCHAR    bDescriptorType;
    USHORT   bcdHID;
    UCHAR    bCountryCode;
    UCHAR    bNumDescriptors;
    // далее следуют один или несколько дескрипторов
    struct _HID_DESCRIPTOR_DESC_LIST {
        {
            UCHAR    bReportType;
            USHORT   wReportLength;
        } DescriptorList[1];
    } USB_HID_DESCRIPTOR;
// Описание на языке Pascal
Type
THidDescriptorList = packed record
    bReportType      : BYTE;
    wReportLength    : WORD;
End;
TUsbHidDescriptor = packed record
    bLength          : BYTE;
    bDescriptorType  : BYTE;
    bcdHID           : WORD;
    bCountryCode     : BYTE;
```

```
bNumDescriptors : BYTE;  
DescriptorList  : Array of THidDescriptorList;  
End;
```

Первые три поля HID-дескриптора содержат стандартные значения:

- ❑ bLength = 9;
- ❑ bDescriptorType = \$21;
- ❑ bcdHID = \$100.

Поле bCountryCode содержит код языка, если устройство локализовано, и 0 — если нет. Клавиатуры могут использовать это поле для передачи языка клавиш.

HID-дескриптор возвращается по запросу Get_Descriptor(\$21).

8.4.4. Дескриптор репорта

Дескриптор репорта (тип REPORT) не похож на остальные дескрипторы. Он имеет сложную табличную структуру. Длина дескриптора зависит от числа и типа данных, которые он передает.

Содержимое дескриптора репорта складывается из частей, передающих информацию об устройстве. Первая часть каждого слагаемого содержит три поля: тип, тег и размер. Такая структура позволяет, с одной стороны, идентифицировать любую составляющую репорта, а с другой, пропустить неизвестные части, сразу переместившись к следующей составляющей (см. разд .8.5).

Дескриптор репорта имеет код \$22 и, соответственно, возвращается по запросу GetDescriptor(\$22).

8.5. Структура дескриптора репорта

Дескриптор репорта (report descriptor) определяет структуру данных, передаваемых от устройства к хосту и от хоста к устройству. Дескриптор HID состоит из элементов (Items), каждый из которых несет определенную информацию об устройстве.

Элемент может содержать байты данных. Размер данных и структура элемента зависят от *базового типа* (fundamental type) элемента.

8.5.1. Структура элементов репорта

Существуют два основных базовых типа: короткие элементы (short item) и длинные элементы (long item).

Элементы короткого типа

Элементы короткого типа имеют однобайтный префикс, содержащий тип, тег и размер (рис. 8.2.):

- ❑ [7:4] — поле тега (bTag, код функции элемента);
- ❑ [3:2] — тип (bType, см. разд. 8.6):
 - 00 — основной (main);
 - 01 — глобальный (global);
 - 10 — локальный (local);
- ❑ [0:1] — число байт данных в элементе (bSize):
 - 00 — нет байт данных;
 - 01 — 1 байт;
 - 10 — 2 байта;
 - 11 — 4 байта.

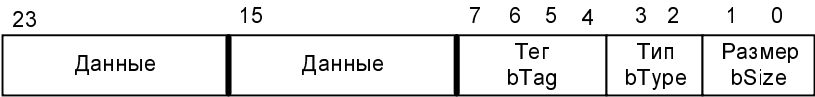


Рис. 8.2. Структура элемента дескриптора репорта

Элементы длинного типа

Элементы длинного типа имеют однобайтный префикс 0xFE, поле bSize всегда равно 2, а длина данных указывается в следующем после префикса байте (рис. 8.3).

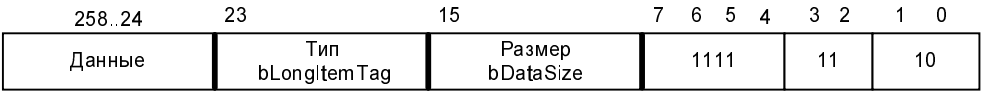


Рис. 8.3. Структура длинного элемента дескриптора репорта

8.5.2. Типы элементов репорта

Спецификация определяет три типа элементов:

- ❑ основные элементы (main items) — определяют группу данных в дескрипторе;

- ☐ глобальные элементы (global items) — определяют данные репорта;
- ☐ локальные элементы (local items) — определяют характеристики конкретных данных.

Основные элементы

Основные элементы определяют или группируют элементы в дескрипторе репорта. Существуют пять элементов основного типа:

- ☐ Input, Output и Feature — определяют поля соответствующих репортов;
- ☐ Collection и End Collection — не определяют поля, но объединяют группу связанных полей внутри репорта.

Биты поля данных для элементов Input, Output и Feature показаны в табл. 8.2 и на рис. 8.4.

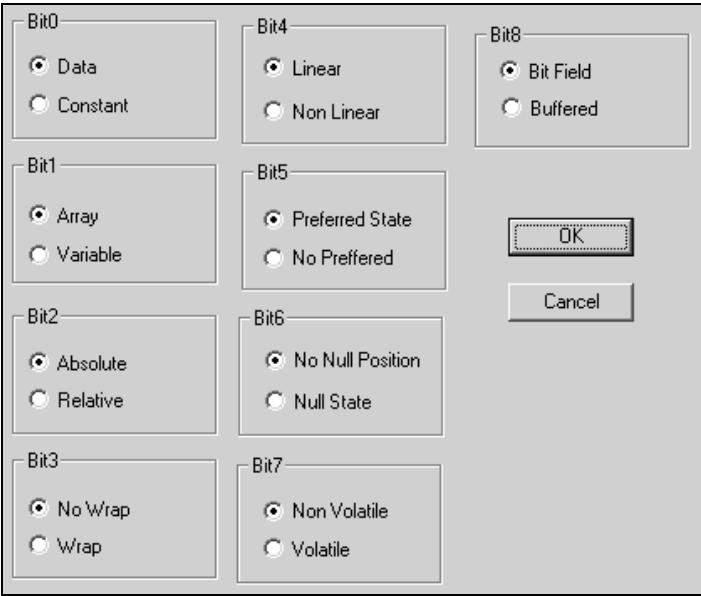


Рис. 8.4. Биты данных элементов Input, Output и Feature

Таблица 8.2. Биты данных элементов Input, Output и Feature

Бит	Если бит равен 0	Если бит равен 1
0	Data	Constant
1	Array	Variable

Таблица 8.2 (окончание)

Бит	Если бит равен 0	Если бит равен 1
2	Absolute	Relative
3	No wrap	Wrap
4	Linear	Non-linear
5	Preferred state	No preferred state
6	No null position	Null state
7	Non-volatile	Volatile
8	Bit Field	Buffered Bytes
9–31	Зарезервировано	

Data/Constant

Data означает, что данные, относящиеся к этой группе, могут изменяться. Constant означает, что данные изменяться не могут (только для чтения).

Array/Variable

Этот бит управляет представлением данных. Например, если клавиатура имеет 8 клавиш, то установка режима Variable будет означать, что клавиатура содержит по одному биту на каждую клавишу. В дескрипторе репорта нужно будет указать, что размер элемента данных (Report Size) равен 1, а число элементов данных (Report Count) равно 8. Установка режима Array будет означать, что каждая клавиша имеет индекс, который передается в репорте, если клавиша нажата. Для восьми клавиш индекс будет кодироваться тремя битами, поэтому поле Report Size должно быть равно 3, а Report Count должно быть равно числу клавиш, которые разрешено нажимать одновременно.

Absolute/Relative

Absolute означает, что значения, переданные в репорте, берутся "как есть". Relative означает, что данные передаются как относительное смещение от предыдущего переданного пакета. Например, джойстик передает абсолютные данные, а мышь — относительные.

No Wrap/Wrap

Wrap означает, что значение "перескочит" через границу, если оно будет продолжать увеличиваться после достижения максимума или уменьшаться после достижения минимума. Этот бит не применяется для Array типа данных.

Linear/Non-linear

Linear означает, что измеренные данные и переданные линейно зависимы. Этот бит не применяется для Array типа данных.

Preferred/No Preferred State

Preferred означает, что устройство имеет специальное состояние, возвращаемое, когда пользователь не взаимодействует с устройством. Например, такое состояние имеет кратковременное нажатие клавиш, и клавиатура возвращает пакет "нет нажатых клавиш", тогда как переключатель не имеет такого состояния, всегда возвращая свое текущее состояние. Этот бит не применяется для Array типа данных.

No Null Position/Null State

Null State означает, что устройство может передавать нулевое состояние, которое может не интерпретироваться корректно, например, не входить в рабочий диапазон данных. No Null Position означает, что устройство не имеет такого состояния и всегда передает корректные и интерпретируемые данные. Этот бит не применяется для Array типа данных.

Non-volatile/Volatile

Этот бит применяется только для OUTPUT и FEATURE репортов. Volatile означает, что устройство может изменить значение данных само, без взаимодействия с хостом. Non-volatile означает, что устройство может изменять значение данных только по требованию хоста. Этот бит не применяется для Array типа данных.

Bit Field/Buffered Bytes

Bit Field означает, что каждый бит или группа бит в байте может представлять отдельную часть данных. Buffered Bytes означает, что данные содержатся в одном или более байт. Значение элемента размера данных (Report Size) для таких данных должно быть равно 8. Этот бит не применяется для Array типа данных.

Более подробное описание можно найти в спецификации HID.

Следует учитывать, что поле данных урезается до последнего ненулевого байта, например:

81 00 – INPUT (Data, Ary, Abs)

81 02 – INPUT (Data, Var, Abs)

82 02 01 – INPUT (Data, Var, Abs, Buf)

Элементы Collection и End Collection могут использоваться в репортах любого типа для объединения связанных элементов в группы. Спецификация оп-

ределяет три типа групп: прикладные (application), физические (physical) и логические (logical). Производители могут определять свои группы.

В табл. 8.3 показано значение поля данных для этих элементов.

Таблица 8.3. Значение поля данных для элементов *Collection* и *End Collection*

Элемент и его префикс	Поле данных	Описание
Collection (0xA1)	0x00	Физическая
	0x01	Прикладная
	0x02	Логическая
	0x03—0x7F	Резерв
	0x80—0xFF	Определяется производителем
End Collection (0xC0)	Нет	Закрывает группу

Прикладная группа объединяет элементы, имеющие одно функциональное назначение. Например, загрузочное устройство может определять две прикладные группы элементов — группу для работы на стадии загрузки и обычную группу. В отдельные группы могут быть объединены репорты INPUT и FEAURE (см. листинг 13.35).

Физическая группа объединяет элементы, представляющие данные геометрической точки (например, позицию указателя мыши, листинг 8.2).

Логическая группа объединяет элементы различного типа в единое целое. Например, могут быть объединены элементы "буфер данных" и "число байт" в этом буфере.

Более детальное описание групп можно найти в спецификации HID, а нам важно знать, что любое описание репорта должно находиться внутри прикладной группы, т. е. внутри элементов Collection (Application) и End Collection.

Глобальные элементы

Глобальные элементы репорта описывают характеристики данных в этом репорте, такие как максимум и минимум величин, размеры и число репортов. Глобальные элементы действуют в пределах всего репорта до следующего глобального элемента.

Спецификация определяет 12 глобальных элементов (табл. 8.4).

Таблица 8.4. Глобальные элементы дескриптора репорта

Элемент	Код*	Описание
Usage Page	000001nn	Назначение данных
Logical Minimum	000101nn	Минимум в логических единицах
Logical Maximum	001001nn	Максимум в логических единицах
Physical Minimum	001101nn	Логический минимум в физических единицах
Physical Maximum	010001nn	Логический максимум в физических единицах
Unit exponent	010101nn	Десятичный порядок единиц
Unit	011001nn	Единицы
Report Size	011101nn	Размер элемента в битах
Report ID	100001nn	Идентификатор репорта
Report Count	100101nn	Число полей данных в репорте
Push	101001nn	Сохраняет копию глобальных настроек в стеке
Pop	101101nn	Восстанавливает глобальные настройки из стека
Резерв	110001nn–111101nn	Резерв

*nn обозначает число байтов, следующих за префиксом.

Наиболее интересными являются элементы Report Count и Report Size, определяющие число байт в репорте и число битов, используемых для каждого элемента данных. Например:

- ❑ два восьмибитовых поля определяются значениями Report Count = 2, Report Size = 8;
- ❑ 10 полей по 4 бита определяются значениями Report Count = 10, Report Size = 4;
- ❑ одно поле размером 16 бит определяется значениями Report Count = 1, Report Size = 16.

Каждый репорт может иметь идентификатор, задаваемый полем Report ID. Мы будем обсуждать использование этого поля в *разд. 13.8*.

Отличие логического минимума и максимума от физического проще всего понять на примере. Пусть устройство передает значение температуры от 0 до 200 °C с сеткой в 2 °C. Физический и логический минимум будет равен 0, а физический максимум — 200. Однако логический максимум будет равен 100.

Элемент `Unit exponent` позволяет определить десятичный порядок значений, описанных в репорте. Допустимы значения от -8 до $+7$ (табл. 8.5). Значение 0 (обозначающее $10^0 = 1$) оставляет данные без изменений. Например, число `1234` и `Unit exponent = 0x0E` преобразуется в `12,34`.

В общем случае для вычисления применяется следующая формула преобразования:

$$\text{Value} = \text{Value_L} \times ((\text{PhMax} - \text{PhMin}) \times (10^{\text{UExp}})) / (\text{LogMax} - \text{LogMin})$$

где

- `Value_L` — значение в логических единицах;
- `PhMax`, `PhMin` — физический максимум и минимум;
- `UExp` — экспонента;
- `LogMax`, `LogMin` — логический максимум и минимум.

Таблица 8.5. Значения элемента `Unit exponent`

Экспонента	0	1	2	3	4	5	6	7
Код	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
Экспонента	-8	-7	-6	-5	-4	-3	-2	-1
Код	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F

Элементы `Push` и `Pop` позволяют сохранить и восстановить глобальные настройки. Часто эти элементы позволяют значительно сократить размер дескриптора за счет сокращения повторных описаний.

Элемент `Usage Page` определяет назначение устройства. Спецификация определяет довольно много назначений (листинг 8.2). Для устройств, не попадающих в указанные типы, определяется тип `Vendor Defined Page`. В зависимости от выбранного назначения интерпретируется назначение полей данных, задаваемое элементом `Usage`.

Листинг 8.2. Значения поля `Usage Page`

<code>USAGE_PAGE (Generic Desktop)</code>	05 01
<code>USAGE_PAGE (Simulation Controls)</code>	05 02
<code>USAGE_PAGE (VR Controls)</code>	05 03
<code>USAGE_PAGE (Sports Controls)</code>	05 04
<code>USAGE_PAGE (Gaming Controls)</code>	05 05
<code>USAGE_PAGE (Keyboard)</code>	05 07
<code>USAGE_PAGE (LEDs)</code>	05 08

USAGE_PAGE (Button)	05 09
USAGE_PAGE (Ordinals)	05 0A
USAGE_PAGE (Telephony Devices)	05 0B
USAGE_PAGE (Consumer Devices)	05 0C
USAGE_PAGE (Digitizers)	05 0D
USAGE_PAGE (Unicode)	05 10
USAGE_PAGE (Alphanumeric Display)	05 14
USAGE_PAGE (Monitor)	05 80
USAGE_PAGE (Monitor Enumerated Values)	05 81
USAGE_PAGE (Monitor Enumerated Values)	05 81
USAGE_PAGE (VESA Virtual Controls)	05 82
USAGE_PAGE (VESA Command)	05 83
USAGE_PAGE (Power Device)	05 84
USAGE_PAGE (Battery System)	05 85
USAGE_PAGE (Vendor Defined Page 1)	06 00 FF

Локальные элементы

Логические элементы характеризуют переключатели, кнопки и другие элементы, состояние которых передается с помощью репорта. Локальные элементы применяются ко всем элементам внутри основных элементов, пока не найдется новое значение, но действие локальных элементов не переходит границу основного элемента.

Наиболее часто используемым является элемент Usage (код 000010nn), обозначающий метод использования данных репорта. Интерпретация кода, заданного в этом элементе, зависит от элемента Usage Page (рис. 8.5).

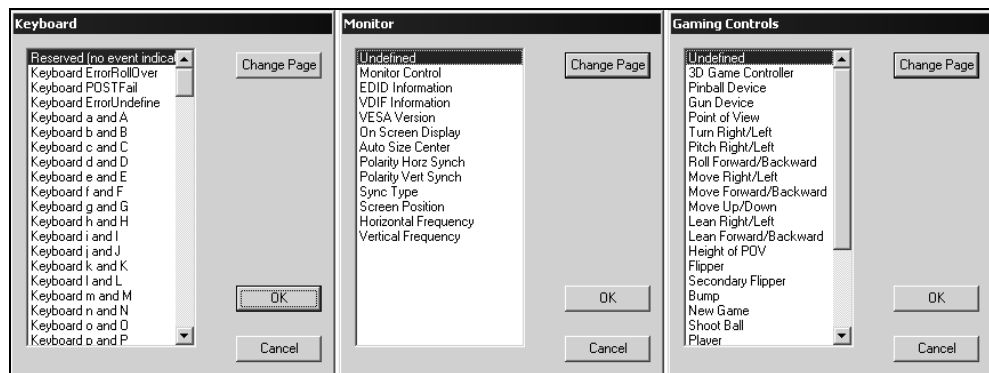


Рис. 8.5. Наборы элементов Usage для разных Usage Page

8.5.3. Примеры дескрипторов

Листинг 8.3 содержит пример дескриптора репорта мыши, листинг 8.4 — клавиатуры, листинг 8.5 — монитора. Использование нестандартного HID-дескриптора мы будем рассматривать в *разд. 13.5* при создании своего HID-устройства.

Листинг 8.3. Дескриптор репорта мыши

```

USAGE_PAGE (Generic Desktop)      05 01
USAGE (Mouse)                     09 02
COLLECTION (Application)          A1 01
    USAGE (Pointer)                09 01
    COLLECTION (Physical)          A1 00
        USAGE_PAGE (Button)        05 09
        USAGE_MINIMUM (Button 1)   19 01
        USAGE_MAXIMUM (Button 3)   29 03
        LOGICAL_MINIMUM (0)        15 00
        LOGICAL_MAXIMUM (1)        25 01
        REPORT_COUNT (3)            95 03
        REPORT_SIZE (1)             75 01
        INPUT (Data,Var,Abs)        81 02
        REPORT_COUNT (1)            95 01
        REPORT_SIZE (5)             75 05
        INPUT (Cnst,Var,Abs)        81 03
        USAGE_PAGE (Generic Desktop) 05 01
        USAGE (X)                   09 30
        USAGE (Y)                   09 31
        LOGICAL_MINIMUM (-127)      15 81
        LOGICAL_MAXIMUM (127)      25 7F
        REPORT_SIZE (8)             75 08
        REPORT_COUNT (2)            95 02
        INPUT (Data,Var,Rel)        81 06
    END_COLLECTION
END_COLLECTION

```

Листинг 8.4. Дескриптор репорта клавиатуры

```

USAGE_PAGE (Generic Desktop)      05 01
USAGE (Keyboard)                  09 06

```

```

COLLECTION (Application)                A1 01
  USAGE_PAGE (Keyboard)                 05 07
  USAGE_MINIMUM (Keyboard LeftControl)  19 E0
  USAGE_MAXIMUM (Keyboard Right GUI)    29 E7
  LOGICAL_MINIMUM (0)                   15 00
  LOGICAL_MAXIMUM (1)                   25 01
  REPORT_SIZE (1)                       75 01
  REPORT_COUNT (8)                      95 08
  INPUT (Data,Var,Abs)                  81 02
  REPORT_COUNT (1)                      95 01
  REPORT_SIZE (8)                       75 08
  INPUT (Cnst,Var,Abs)                  81 03
  REPORT_COUNT (5)                      95 05
  REPORT_SIZE (1)                       75 01
  USAGE_PAGE (LEDs)                     05 08
  USAGE_MINIMUM (Num Lock)              19 01
  USAGE_MAXIMUM (Kana)                  29 05
  OUTPUT (Data,Var,Abs)                 91 02
  REPORT_COUNT (1)                      95 01
  REPORT_SIZE (3)                       75 03
  OUTPUT (Cnst,Var,Abs)                 91 03
  REPORT_COUNT (6)                      95 06
  REPORT_SIZE (8)                       75 08
  LOGICAL_MINIMUM (0)                   15 00
  LOGICAL_MAXIMUM (101)                 25 65
  USAGE_PAGE (Keyboard)                 05 07
  USAGE_MINIMUM (Reserved (no event indicated)) 19 00
  USAGE_MAXIMUM (Keyboard Application)  29 65
  INPUT (Data,Ary,Abs)                  81 00
END_COLLECTION C0

```

Листинг 8.5. Дескриптор репорта монитора

```

USAGE_PAGE (Monitor)                   05 80
USAGE (Monitor Control)                 09 01
COLLECTION (Application)                A1 01
  REPORT_ID (1)                         85 01
  LOGICAL_MINIMUM (0)                   15 00

```

LOGICAL_MAXIMUM (255)	26 FF 00
REPORT_SIZE (8)	75 08
REPORT_COUNT (128)	95 80
USAGE (EDID Information)	09 02
FEATURE (Data,Var,Abs,Buf)	B2 02 01
REPORT_ID (2)	85 02
REPORT_COUNT (243)	95 F3
USAGE (VDIF Information)	09 03
FEATURE (Data,Var,Abs,Buf)	B2 02 01
REPORT_ID (3)	85 03
USAGE_PAGE (VESA Virtual Controls)	05 82
REPORT_COUNT (1)	95 01
REPORT_SIZE (16)	75 10
LOGICAL_MAXIMUM (200)	26 C8 00
USAGE (Brightness)	09 10
FEATURE (Data,Var,Abs)	B1 02
REPORT_ID (4)	85 04
LOGICAL_MAXIMUM (100)	25 64
USAGE (Contrast)	09 12
FEATURE (Data,Var,Abs)	B1 02
REPORT_COUNT (6)	95 06
LOGICAL_MAXIMUM (255)	26 FF 00
USAGE (Video Gain Red)	09 16
USAGE (Video Gain Green)	09 18
USAGE (Video Gain Blue)	09 1A
USAGE (Video Black Level Red)	09 6C
USAGE (Video Black Level Green)	09 6E
USAGE (Video Black Level Blue)	09 70
FEATURE (Data,Var,Abs)	B1 02
REPORT_ID (5)	85 05
LOGICAL_MAXIMUM (127)	25 7F
USAGE (Horizontal Position)	09 20
USAGE (Horizontal Size)	09 22
USAGE (Vertical Position)	09 30
USAGE (Vertical Size)	09 32
USAGE (Trapezoidal Distortion)	09 42
USAGE (Tilt)	09 44
FEATURE (Data,Var,Abs)	B1 02
END_COLLECTION	C0

8.6. Запросы к HID-устройству

HID-устройство должно отвечать на стандартные запросы:

- ❑ `Get_Status` (для типа запроса `Device`);
- ❑ `Set_Address`;
- ❑ `Get_Descriptor` (включая специфические дескрипторы `HID` и `REPORT`);
- ❑ `Get_Configuration`;
- ❑ `Set_Configuration`.

Запрос `Set_Descriptor` позволяет хосту изменять дескрипторы устройства. Обработка этого запроса необязательна.

Кроме стандартных запросов, устройство может обрабатывать специфические HID-запросы. Такие запросы имеют следующие значения полей:

- ❑ поле `bmRequestType` может принимать одно из значений: `10100001B` или `00100001B`;
- ❑ поле `bRequest` задает тип специального запроса:
 - `0x01` — `GET_REPORT`;
 - `0x02` — `GET_IDLE`;
 - `0x03` — `GET_PROTOCOL`;
 - `0x04—0x08` — резерв;
 - `0x09` — `SET_REPORT`;
 - `0x0A` — `SET_IDLE`;
 - `0x0B` — `SET_PROTOCOL`.

Из этого списка обязательным является только запрос `GET_REPORT`¹, а запросы `GET_PROTOCOL` и `SET_PROTOCOL` обрабатываются, если устройство является загрузочным (табл. 8.6).

Таблица 8.6. Специальные запросы HID-устройства

Запрос	Код	Источник данных	Длина данных	Обязателен
<code>GET_REPORT</code>	<code>0x01</code>	Устройство	Длина репорта	Да
<code>GET_IDLE</code>	<code>0x02</code>	Устройство	1	Нет
<code>GET_PROTOCOL</code>	<code>0x03</code>	Устройство	1	Да, для загрузочных устройств

¹ Если используется только передача данных от устройства к хосту, можно обойтись и без обработки `GET_REPORT`.

Таблица 8.6 (окончание)

Запрос	Код	Источник данных	Длина данных	Обязателен
SET_REPORT	0x09	Хост	Длина репорта	Нет
SET_IDLE	0x0A	Хост	0	Нет
SET_PROTOCOL	0x0B	Хост	0	Да, для загрузочных устройств

8.6.1. Запрос *GET_REPORT*

Запрос `GET_REPORT` позволяет хосту принять данные через нулевую конечную точку. Запрос имеет следующие поля:

- ❑ `bmRequestType` = 10100001b;
- ❑ `bRequest` = `GET_REPORT` (код 0x01);
- ❑ `wValue` — тип репорта и идентификатор (Report ID);
- ❑ `wIndex` — номер интерфейса;
- ❑ `wLength` — длина репорта;
- ❑ `Data` — данные репорта.

Старший байт поля `wValue` содержит тип репорта:

- ❑ 01 — `INPUT`;
- ❑ 02 — `OUTPUT`;
- ❑ 03 — `FEATURE`;
- ❑ 04—FF — резерв.

Младший байт поля `wValue` содержит идентификатор репорта (Report ID) или 0, если идентификатор репорта не используется. Обработка этого запроса обязательна для всех HID-устройств.

8.6.2. Запрос *SET_REPORT*

Запрос `SET_REPORT` позволяет хосту передать данные устройству через нулевую конечную точку. Запрос имеет следующие поля:

- ❑ `bmRequestType` = 00100001b;
- ❑ `bRequest` = `SET_REPORT` (код 0x09);
- ❑ `wValue` — тип репорта и идентификатор (Report ID);
- ❑ `wIndex` — номер интерфейса;

- ❑ `wLength` — длина репорта;
- ❑ `Data` — данные репорта.

Значение полей этого запроса такое же, как для запроса `GET_REPORT`. Устройство может игнорировать запросы `SET_REPORT`.

8.6.3. Запрос ***GET_IDLE***

Запрос `GET_IDLE` позволяет хосту прочитать текущее значение длительности для репорта (`idle rate`). Запрос имеет следующие поля:

- ❑ `bmRequestType` = 10100001b;
- ❑ `bRequest` = `GET_IDLE` (код 0x02);
- ❑ `wValue` — идентификатор репорта (`Report ID`) в младшем байте;
- ❑ `wIndex` — номер интерфейса;
- ❑ `wLength` = 1;
- ❑ `Data` — значение длительности (см. разд. 8.6.4).

8.6.4. Запрос ***SET_IDLE***

Запрос `SET_IDLE` позволяет хосту задать значение длительности (`idle rate`) для репортов. Запрос имеет следующие поля:

- ❑ `bmRequestType` = 00100001b;
- ❑ `bRequest` = `SET_IDLE` (код 0x0A);
- ❑ `wValue` — длительность (`Idle duration`) в старшем байте и идентификатор репорта (`Report ID`) в младшем байте;
- ❑ `wIndex` — номер интерфейса;
- ❑ `wLength` = 0.

Величина длительности позволяет управлять передачей репортов в случае отсутствия изменений в передаваемых данных. Нулевое значение длительности означает "бесконечную паузу" (`indefinite`), в этом случае устройство будет передавать репорты только в случае изменений состояния. Ненулевое значение задает длительность интервала с кратностью 4 мс, в течение которого конечная точка отвечает на запросы пакетами `NAK`, если в передаваемых данных нет изменений. Спецификация определяет точность выдерживания интервала $\pm 10\%$.

Если поле `Report ID` равно нулю, то установка производится для всех репортов, иначе — только для репорта с данным идентификатором.

Для клавиатуры рекомендуется устанавливать интервал 500 мс (это будет означать время ожидания перед началом дублирования символов, `repeat`

rate), а для джойстиков и мыши — бесконечный (эти устройства будут передавать данные только в случае изменения состояния).

8.6.5. Запрос **GET_PROTOCOL**

Запрос GET_PROTOCOL позволяет хосту прочитать текущее значение выбранного протокола. Запрос имеет следующие поля:

- ❑ bmRequestType = 10100001b;
- ❑ bRequest = GET_PROTOCOL (код 0x03);
- ❑ wValue = 0;
- ❑ wIndex — интерфейс;
- ❑ wLength = 1;
- ❑ Data — тип протокола:
 - 0 — загрузочный протокол (boot protocol);
 - 1 — обычный протокол (report protocol).

Этот запрос используется только для загрузочных устройств.

8.6.6. Запрос **SET_PROTOCOL**

Запрос SET_PROTOCOL позволяет производить переключение между загрузочным протоколом (boot protocol) и обычным протоколом (report protocol). Запрос имеет следующие поля:

- ❑ bmRequestType = 00100001b;
- ❑ bRequest = SET_PROTOCOL (код 0x0B);
- ❑ wValue — тип протокола:
 - 0 — загрузочный протокол (boot protocol);
 - 1 — обычный протокол (report protocol);
- ❑ wIndex — интерфейс;
- ❑ wLength = 0.

Этот запрос используется только для загрузочных устройств.

8.7. Инструментальные средства

Для того чтобы помочь в написании HID-устройства USB Форум разработал две программы, распространяемые свободно: *HID Descriptor Tool* и *USB Compliance Tool*.

Первая программа (рис. 8.6) позволяет автоматизировать процесс написания дескриптора репорта и проверить его правильность прежде, чем копировать его в устройство.

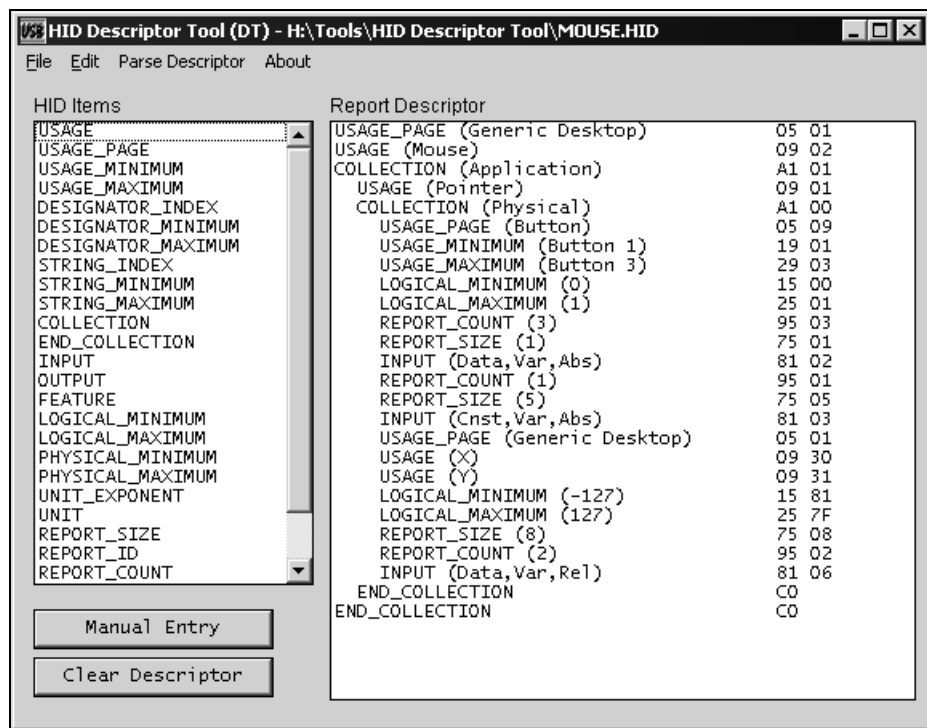


Рис. 8.6. Программа генерации и проверки дескрипторов репортов

Вторая программа представляет собой набор инструментальных средств, позволяющих выполнить серию основных тестов на любом USB-устройстве, плюс содержит дополнительные тесты для HID-устройств, хабов и устройств связи. USB Compliance Tool загружает свой собственный драйвер для испытуемого устройства, с помощью которого можно выбрать и послать в устройство стандартный набор запросов.

8.8. Взаимодействие с HID-драйвером

Любые операции с устройством (не только с HID, но и с любым другим) производятся с помощью дескриптора устройства, получаемого с помощью вызова функции `CreateFile`. Основным параметром этой функции является имя устройства. Получение имени USB-устройства мы будем обсуждать в

главе 10, посвященной Plug and Play, а пока предположим, что у нас есть открытый дескриптор устройства.

Заголовки основных функций, предоставляемых HID-драйвером, приведены в листинге 8.6.

Листинг 8.6. Заголовки функций HID-драйвера (Delphi)

```
// возвращает GUID, связанный с HID
procedure HidD_GetHidGuid(var HidGuid: TGUID) stdcall;

// возвращает атрибуты HID-устройства (идентификаторы
// производителя и продукта)
function HidD_GetAttributes(HidDeviceObject: THandle; var HidAttrs:
THIDDAttributes): LongBool; stdcall;

// возвращает указатель на буфер, содержащий информацию
// о возможностях устройства
function HidD_GetPreparsedData(HidDeviceObject: THandle;
var PreparsedData: PHIDPPreparsedData): LongBool; stdcall;

// возвращает структуру, описывающую возможности устройства
function HidP_GetCaps(PreparsedData: PHIDPPreparsedData;
var Capabilities: THIDPCaps): NTSTATUS; stdcall;

// читает feature-репорт из устройства
function HidD_GetFeature(HidDeviceObject: THandle; var Report;
Size: Integer): LongBool; stdcall;

// передает feature-репорт в устройство
function HidD_SetFeature(HidDeviceObject: THandle; var Report;
Size: Integer): LongBool; stdcall;
```

Обмен данными с устройством можно производить с помощью обычных Windows API функций ReadFile и WriteFile, соответственно для входных и выходных репортов и функций HID API HidD_GetFeature и HidD_SetFeature для специальных репортов (см. разд. 13.6).

Важно

При использовании функции ReadFile пользовательская программа "проваливается" в системный HID-драйвер и будет находиться там до тех пор, пока не получит от HID-устройства запрошенное количество данных. Не помогает даже использование функции ReadFileEx. При написании программы необходимо таким образом разместить вызовы ReadFile, чтобы она не "вешала" основное приложение при ожидании данных с устройства.

HID-функции содержатся в модуле Hid.Dll. Для использования этих функций в Visual Studio нужно подключить модуль hidsdi.h. В Borland Delphi при-

дется либо подключать эти функции вручную (как показано в листинге 8.7), либо использовать готовые классы, например, Hid.Pas из библиотеки JEDI (<http://delphi-jedi.org>).

Листинг 8.7. Подключение HID-функций

```
const
    HidModuleName = 'HID.dll';

procedure HidD_GetHidGuid(var HidGuid: TGUID) stdcall;
{$EXTERNALSYM HidD_GetHidGuid}

function HidD_GetPreparedData(HidDeviceObject: THandle;
var PreparedData: PHIDPPreparedData): LongBool; stdcall;
{$EXTERNALSYM HidD_GetPreparedData}

function HidD_FreePreparedData(PreparedData: PHIDPPreparedData):
LongBool; stdcall;
{$EXTERNALSYM HidD_FreePreparedData}

... ..
procedure HidD_GetHidGuid; external HidModuleName name 'HidD_GetHidGuid';

function HidD_GetPreparedData; external HidModuleName name
'HidD_GetPreparedData';

function HidD_FreePreparedData; external HidModuleName name
'HidD_FreePreparedData';
```

В качестве примера приведем небольшую программу, получающую список HID-устройств и отображающую свойства одного из них (листинг 8.8).

Листинг 8.8. Использование HID-функций

```
unit Unit1;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics,
    Controls, Forms, Dialogs, ExtCtrls, StdCtrls, Buttons;
```

```
type
  TForm1 = class(TForm)
    lbLog: TListBox;
    Panel1: TPanel;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    procedure DisplayHIDInformation(HidName : String);
  public
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

Uses SetupApi, Hid;

// Отображение списка HID-устройств
// и их свойств
procedure TForm1.Button1Click(Sender: TObject);
var HidGuid : TGuid; PnPHandle : HDevInfo;
    DevData: TSPDevInfoData;
    DeviceInterfaceData: TSPDeviceInterfaceData;
    FunctionClassDeviceData: PSPDeviceInterfaceDetailData;
    Success: LongBool;
    DevIndex: DWORD;
    BytesReturned: DWORD;
    HidName : String;
begin
  // ОЧИСТИТЬ ЛОГ
  lbLog.Items.Clear;
  // ПОЛУЧИТЬ GUID для класса HID
  HidD_GetHidGuid(HidGuid);
  // Получаем дескриптор PnP для HID-класса
  PnPHandle := SetupDiGetClassDevs(@HidGuid, nil,
    0, DIGCF_PRESENT or DIGCF_DEVICEINTERFACE);
```

```

// Если ошибка, то выходим
If PnPHandle = Pointer(INVALID_HANDLE_VALUE) then Exit;

Try
    // Индекс текущего устройства
    DevIndex := 0;
    // Цикл по всем устройствам в HID-классе
    Repeat
        DeviceInterfaceData.cbSize := SizeOf(TSPDeviceInterfaceData);

        // Получить информацию об интерфейсах устройства номер DevIndex
        Success := SetupDiEnumDeviceInterfaces(
            PnPHandle, nil, HidGuid, DevIndex, DeviceInterfaceData);
    If Success then begin
        DevData.cbSize := SizeOf(DevData);
        BytesReturned := 0;
        // Получаем подробности об устройстве с интерфейсом
        // DeviceInterfaceData
        // Сначала вызываем с нулевым размером буфера, получаем
        // размер необходимого буфера, потом вызываем повторно,
        // сформировав правильный буфер
        SetupDiGetDeviceInterfaceDetail(PnPHandle,
            @DeviceInterfaceData, nil, 0, BytesReturned, @DevData);
    If (BytesReturned <> 0) and
        (GetLastError = ERROR_INSUFFICIENT_BUFFER) then begin
        // Создаем буфер
        FunctionClassDeviceData := AllocMem(BytesReturned);
        FunctionClassDeviceData.cbSize := 5;
        // Получаем информацию
        If SetupDiGetDeviceInterfaceDetail(PnPHandle,
            @DeviceInterfaceData, FunctionClassDeviceData,
            BytesReturned, BytesReturned, @DevData) then begin
            // Отобразить имя PnP-имя устройства
            HidName:= StrPas(PChar(@FunctionClassDeviceData.DevicePath));
            lbLog.Items.Add(HidName);
            // Отобразить информацию об устройстве
            DisplayHIDInformation(HidName);
        End;
    End;

```

```

    // Освободить буфер
    FreeMem(FunctionClassDeviceData);
End;
End;
// Следующее устройство
Inc(DevIndex);
Until not Success;
Finally
    SetupDiDestroyDeviceInfoList(PnPHandle);
End;
end;

// Отображение информации о HID-устройстве
procedure TForm1.DisplayHIDInformation(HidName : String);
var HidHandle : THandle;
    CanReadWriteAccess : Boolean;
    Attributes : THIDDAttributes;
    NumInputBuffers : Integer;
    Buffer : array [0..253] of WideChar;

    // Получение строки по дескриптору
    Function GetString(StrDescriptor : Byte): WideString;
    var Buffer : array [0..253] of WideChar;
    begin
        Result := 'Ошибка';
        if StrDescriptor <> 0 then
            if HidD_GetIndexedString(HidHandle,
                StrDescriptor, Buffer, SizeOf(Buffer)) then
                Result:= Buffer;
        end;
    end;

begin
    // Сначала пробуем открыть устройство
    // в режиме r/w
    lbLog.Items.Add(' Пробуем открыть HID-устройство...');
    HidHandle:= CreateFile(PChar(@HidName[1]),
        GENERIC_READ or GENERIC_WRITE,
        FILE_SHARE_READ or FILE_SHARE_WRITE,

```

```
        nil,  
        OPEN_EXISTING, 0, 0  
    );  
  
    // Устройство поддерживает запись?  
    CanReadWriteAccess:= HidHandle <> INVALID_HANDLE_VALUE;  
  
    // Если не получилось, пробуем открыть  
    // в режиме только чтения данных  
    If not CanReadWriteAccess then begin  
        HidHandle:= CreateFile(PChar(@HidName[1]),  
                                0,  
                                FILE_SHARE_READ or FILE_SHARE_WRITE,  
                                nil,  
                                OPEN_EXISTING, 0, 0  
        );  
    End else begin  
        lbLog.Items.Add(' Устройство открыто в режиме read/write');  
    End;  
  
    // Если не получилось - ошибка и выход  
    If HidHandle = INVALID_HANDLE_VALUE then begin  
        lbLog.Items.Add(' Ошибка открытия устройства');  
        Exit;  
    End else begin  
        lbLog.Items.Add(' Устройство открыто в режиме read only!');  
    End;  
  
    // Получаем атрибуты устройства  
    Attributes.Size := SizeOf(THIDDAttributes);  
    If HidD_GetAttributes(HidHandle, Attributes) then begin  
        lbLog.Items.Add(  
            Format('   VendorID=%d, ProductID=%d,  
                VersionNumber=%d',[Attributes.VendorID,  
                Attributes.ProductID, Attributes.VersionNumber]));  
    End else begin  
        lbLog.Items.Add(' Ошибка HidD_GetAttributes');  
    End;
```

```
// Получаем число буферов
If HidD_GetNumInputBuffers(HidHandle, NumInputBuffers) then begin
    lbLog.Items.Add(
        Format('    Число входных буферов=%d', [NumInputBuffers]));
End else begin
    lbLog.Items.Add(' Ошибка HidD_GetNumInputBuffers');
End;

// Получаем идентификатор изготовителя
FillChar(Buffer, SizeOf(Buffer), #0);
If HidD_GetManufacturerString(HidHandle, Buffer, SizeOf(Buffer)) then begin
    lbLog.Items.Add(Format('    Производитель=%s', [Buffer]));
End else begin
    lbLog.Items.Add(' Ошибка HidD_GetManufacturerString');
End;

// Получаем идентификатор продукта
FillChar(Buffer, SizeOf(Buffer), #0);
If HidD_GetProductString(HidHandle, Buffer, SizeOf(Buffer)) then begin
    lbLog.Items.Add(Format('    Продукт=%s', [Buffer]));
End else begin
    lbLog.Items.Add(' Ошибка HidD_GetProductString');
End;

// Получаем серийный номер
FillChar(Buffer, SizeOf(Buffer), #0);
If HidD_GetSerialNumberString(HidHandle, Buffer, SizeOf(Buffer)) then begin
    lbLog.Items.Add(Format('    Серийный номер=%s', [Buffer]));
End else begin
    lbLog.Items.Add(' Ошибка HidD_GetSerialNumberString');
End;

// Освободить дескриптор устройства
CloseHandle(HidHandle);
end;

end.
```

Вид формы и результат работы показаны на рис. 8.7.

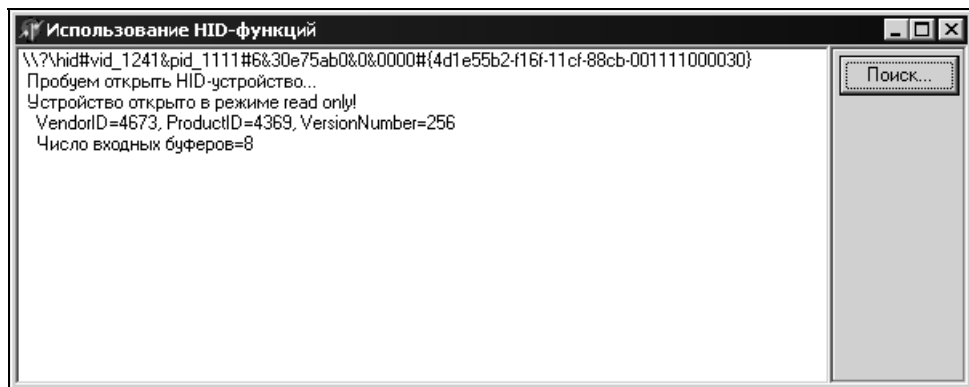


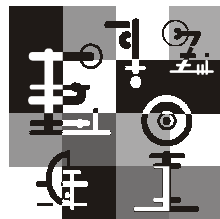
Рис. 8.7. Результат программы, демонстрирующей HID-функции

Заметим, что имя HID-устройства непохоже на обычное имя устройства, как, например, COM1 или LPT. Это имя присваивается менеджером системы Plug and Play и выглядит, например, так:

```
\\?\\hid#vid_1241&pid_1111#6&30e75ab0&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}
```

Более подробно мы будем рассматривать использование HID-функций на практическом примере в *главе 13*.

Глава 9



Введение в WDM

Когда ваш компьютер говорит
"Вставьте диск номер два", не торопитесь,
сначала выньте диск номер один...
даже если вы уверены, что сможете засунуть туда оба.

WDM (Windows Driver Model, драйверная модель Windows) — новая модель архитектуры драйверов, предложенная Microsoft для Windows 2000. Конечно, эта архитектура развивалась, начиная с Windows 3.11, продолжала развиваться в Windows 98 и Windows NT, но по-настоящему полной она стала только в Windows 2000.

С точки зрения WDM, существует три типа драйверов:

- ❑ *драйвер шины* (Bus Driver) — драйвер, обслуживающий контроллер шины, адаптер, мост или любые другие устройства, имеющие дочерние устройства. Драйверы шин нужны для работы системы и, в общем случае, поставляются Microsoft. Для каждого типа шины (PCI, PCMCIA и USB) в системе имеется свой драйвер. Сторонние разработчики создают драйверы для поддержки новых шин, например, для VMEbus, Multibus или Futurebus;
- ❑ *функциональный драйвер* (Function Driver) — основной драйвер устройства, предоставляющий его функциональный интерфейс. Обязателен, кроме тех случаев, когда устройство используется без драйверов (т. е. ввод/вывод осуществляется драйвером шины или драйвером фильтров шины). Функциональный драйвер по определению обладает наиболее полной информацией о своем устройстве. Обычно только этот драйвер имеет доступ к специфическим регистрам устройства;
- ❑ *драйвер фильтра* (Filter Driver) — драйвер, поддерживающий дополнительную функциональность устройства (или существующего драйвера) или изменяющий запросы на ввод/вывод и ответы на них от других драйверов (это часто используется для коррекции устройств, предоставляющих неверную информацию о своих требованиях к аппаратным ресурсам). Такие драйверы необязательны, и их может быть несколько.

Они могут работать как на более высоком уровне, чем функциональный драйвер или драйвер шины, так и на более низком. Обычно эти драйверы предоставляются производителями или независимыми поставщиками оборудования.

В среде WDM один драйвер не может контролировать все аспекты устройства: драйвер шины информирует диспетчера PnP (см. главу 10) об устройствах, подключенных к шине, в то время как функциональный драйвер управляет устройством. Драйверы фильтров низкого уровня позволяют исправлять информацию о требованиях устройства к системным ресурсам, а драйверы фильтров высокого уровня добавляют устройству дополнительную функциональность (например, производят дополнительную защиту клавиатуры).

Примечание

Получить список загруженных в данный момент драйверов можно с помощью утилиты Drivers (файл drivers.exe), как показано на рис. 9.1.

C:>Drivers

ModuleName	Code	Data	Bss	Paged	Init	LinkDate
ntoskrnl.exe	431488	75904	0	1170944	171904	Thu Aug 29 13:03:24 2002
hal.dll	32896	42624	0	28672	14336	Thu Aug 29 12:05:02 2002
KDCOM.DLL	2560	256	0	1280	512	Sat Aug 18 00:49:10 2001
BOOTVID.dll	5632	3584	0	0	512	Sat Aug 18 00:49:09 2001
ACPI.sys	103936	11008	0	40192	4736	Thu Aug 29 12:09:03 2002
WMILIB.SYS	512	0	0	1280	256	Sat Aug 18 01:07:23 2001
...	...	...	...	...	...	...
Cdfs.SYS	6528	640	0	42880	4480	Thu Aug 29 12:58:50 2002
asynmac.sys	8576	1024	0	0	1152	Sat Aug 18 00:55:29 2001
USBSTOR.SYS	6656	128	0	10368	1536	Thu Aug 29 12:32:50 2002
ntdll.dll	466944	20480	0	0	0	Thu Aug 29 14:40:40 2002
Total	5785536	818400	0	4474912	532192	

Рис. 9.1. Получение списка загруженных драйверов

9.1. Драйверные слои

Согласно типам драйверов, существуют три типа объектов:

- ❑ объекты физических устройств (PDO, Physical Device Object);
- ❑ объекты функциональных устройств (FDO, Functional Device Object);
- ❑ объекты фильтров устройств (FiDO, Filter Device Object).

Объекты PDO создаются для каждого физически идентифицируемого элемента аппаратуры, подключенного к шине данных. Объект FDO подразумевает единицу логической функциональности устройства. Объекты фильтров предоставляют дополнительную функциональность.

Начиная с Windows 2000, последовательность загрузки драйверов такая:

1. Во время загрузки ОС производится загрузка шинных драйверов для каждой известной системы шины (список шин составляется при установке ОС и сохраняется в реестре).
2. Вызывается `DriverEntry` (см. разд. 9.3.1), а затем `AddDevice` (см. разд. 9.3.2) каждого шинного драйвера. В `AddDevice` создается FDO для драйвера системной шины. Потом посылают `IRP_MN_START_DEVICE` на созданный FDO.
3. Шинный драйвер составляет список всех устройств, подключенных к шине. Для каждого найденного устройства создается объект PDO.
4. На каждый PDO посылается запрос `IRP_MN_QUERY_DEVICE_RELATION`, в ответ на который шинный драйвер возвращает идентификаторы всех найденных устройств.
5. На эти PDO посылают запрос `IRP_MN_QUERY_ID`, в ответ на который драйвер системной шины сообщает идентификаторы этих устройств.
6. Получив эти идентификаторы, система пытается найти и загрузить драйверы устройств.
7. Найдя драйвер для устройства, система загружает его в память, вызывая его `DriverEntry`. Потом вызывается `AddDevice`, где создается FDO для устройства. Если устройств, управляемых этим драйвером, несколько, то `AddDevice` будет вызвана для каждого устройства. Если в реестре зарегистрированы дополнительные фильтры, то они также загружаются в память. Затем система посылает на FDO запрос `IRP_MN_START_DEVICE`. Кроме того, при необходимости `AddDevice` осуществляет создание символического имени устройства (см. разд. 9.2).
8. Потом посылают на FDO запрос `IRP_MN_QUERY_DEVICE_RELATIONS`. Если устройство само является шиной или держит на себе другие устройства, которыми само не управляет, то для устройств на нем повторяется вся последовательность действий, начиная с пункта 5.

Функция `AddDevice`, вызываемая для каждого FDO, вызывает `IoCreateDevice` и `IoAttachDeviceToDeviceStack`, обеспечивая построение *стека устройств* (device stack). Стек устройств обеспечивает прохождение запросов от пользовательских программ до нижнего (аппаратного) уровня драйверов (рис. 9.2).

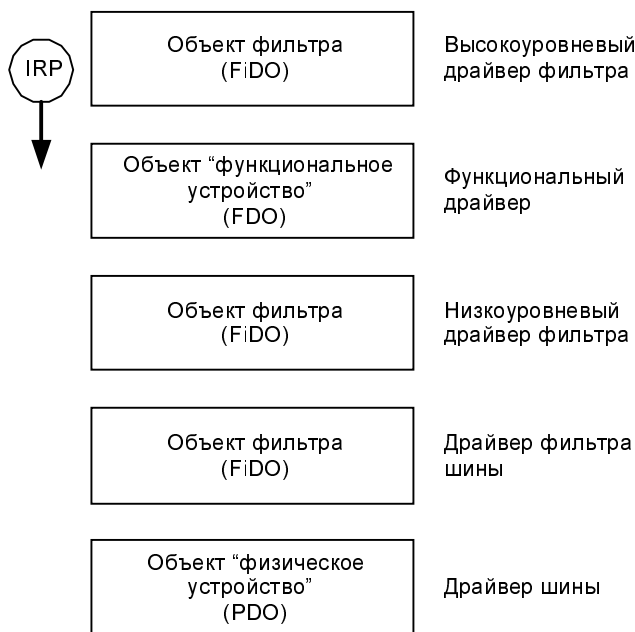


Рис. 9.2. Стек устройств

9.2. Символьные имена устройств

Создавая объект "устройство", драйвер может присвоить ему имя. Тогда он помещается в пространство имен *диспетчера объектов*. Драйвер может позволить определить имя устройства явно или позволить сгенерировать его автоматически. По соглашению имена объектов помещаются в каталог пространства имен `\Device`, недоступный приложениям через Windows API.

Чтобы сделать объект устройства доступным для приложений, драйвер должен создать в каталоге `\??` (до Windows NT4 этот каталог назывался `\DosDevice`, а в Windows XP называется `\GLOBAL??`) символьную ссылку на имя этого объекта в каталоге `\Device` (рис. 9.3). Эта ссылка называется *символьным именем* устройства или *DOS-именем*. Унаследованные драйверы и драйверы логических устройств обычно создают ссылку с общеизвестными

именами (например, F: для устройства чтения компакт-диска \Device\CdRom0 или COM1 для последовательного порта \Device\Serial0). Устройства, создаваемые динамически при работе системы Plug and Play, создают имена, используя GUID, гарантируя глобальную уникальность имени. Получение PnP-имен и работу с ними мы будем обсуждать в *главе 10*.

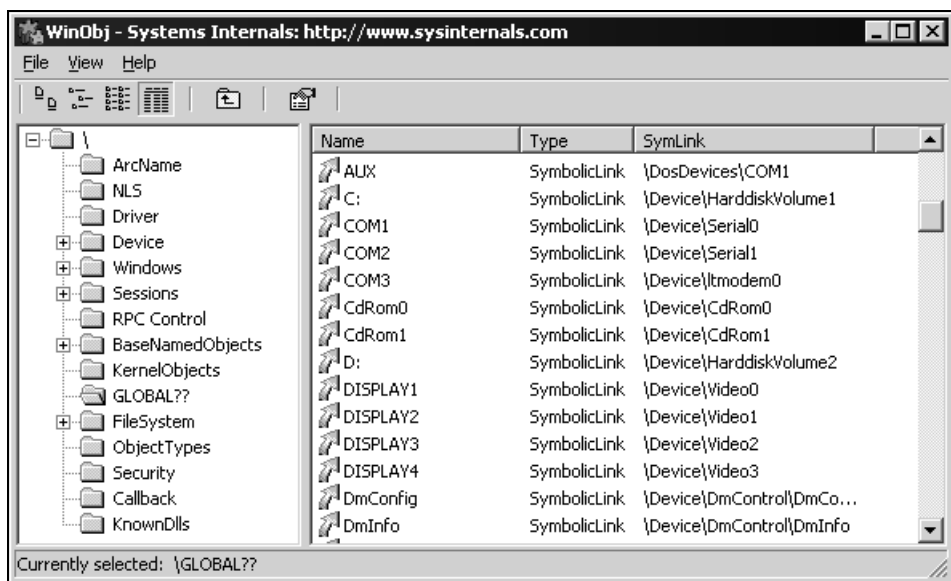


Рис. 9.3. Символьные имена Windows XP

Несколько символьных имен может указывать на одно и то же устройство, однако обратное не верно. Одно имя может ссылаться только на одно устройство. Так, например, последовательный порт \Device\Serial0 может иметь имена COM1 и MyCOM.

Хотя прикладные программы не могут использовать внутренние имена, они могут получать, добавлять и удалять символьные имена для внутренних имен устройств. Эти операции выполняются с помощью функций QueryDosDevice и DefileDosDevice, которые описаны в *главе 16*.

Листинг 9.1 демонстрирует получение списка, добавление и удаление символьных имен.

Листинг 9.1. Работа с DOS-именами устройств

```
unit Unit1;

interface
```

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
Dialogs, StdCtrls, ComCtrls, ExtCtrls;

type

```
TForm1 = class(TForm)
  StatusBar: TStatusBar;
  lbNameList: TListBox;
  Panel1: TPanel;
  Label1: TLabel;
  Label2: TLabel;
  NTDeviceName: TEdit;
  DOSDeviceName: TEdit;
  btnGetName: TButton;
  btnAddName: TButton;
  btnDelName: TButton;
  btnGetList: TButton;
  procedure btnGetNameClick(Sender: TObject);
  procedure btnAddNameClick(Sender: TObject);
  procedure btnDelNameClick(Sender: TObject);
  procedure btnGetListClick(Sender: TObject);
  procedure lbNameListDblClick(Sender: TObject);
private
public
end;
```

var

```
Form1: TForm1;
```

implementation

```
{ $R *.dfm }
```

```
{ Получение NT-имени по DOS-имени }
```

```
procedure TForm1.btnGetNameClick(Sender: TObject);
Var Result : Array [1..MAX_PATH] of Char;
begin
  If (QueryDosDevice(PChar(DOSDeviceName.Text),
```

```
        @Result, MAX_PATH) <> 0) then
    NtDeviceName.Text:= StrPas(@Result)
Else
    NtDeviceName.Text:= 'Устройство не существует';
end;

{Добавить DOS-имя для NT-имени}
procedure TForm1.btnAddNameClick(Sender: TObject);
begin
    If not DefineDosDevice(
        DDD_RAW_TARGET_PATH,
        PChar(DosDeviceName.Text),
        PChar(NtDeviceName.Text))
    then
        StatusBar.Panels[0].Text:= 'Ошибка добавления имени';
end;

{Удалить DOS-имя}
procedure TForm1.btnDelNameClick(Sender: TObject);
Var Result : Array [1..MAX_PATH] of Char;
begin
    {Ищем NT-имя для удаляемого DOS-имени}
    If not(QueryDosDevice(PChar(DOSDeviceName.Text),
        @Result, MAX_PATH) <> 0) then begin
        StatusBar.Panels[0].Text:= 'DOS-имя не определено';
        Exit;
    End;

    {Удаляем DOS-имя}
    If not DefineDosDevice(
        DDD_RAW_TARGET_PATH or
        DDD_REMOVE_DEFINITION or
        DDD_EXACT_MATCH_ON_REMOVE,
        PChar(DosDeviceName.Text),
        PChar(NtDeviceName.Text)
    ) then
        StatusBar.Panels[0].Text:= 'Ошибка удаления имени';
end;
```

```
{Получение всех имен устройства}
procedure TForm1.btnGetListClick(Sender: TObject);
var BufSize : Cardinal; P, PName : Pointer; SName : String;
begin
    {Очищаем предыдущий список}
    lbNameList.Items.Clear;

    {Размер буфера}
    BufSize:= 10240;
    {Распределяем память для буфера}
    GetMem(P, BufSize);
    {Запрашиваем список имен}
    If QueryDosDevice(nil, P, BufSize) <> 0 then begin
        {Цикл по всем именам...}
        PName:= P;
        while (True) do begin
            SName:= StrPas(PName);
            If SName = '' then Break;
            {Добавляем в список}
            lbNameList.Items.Add(SName);
            {Переход к следующему устройству}
            {Сдвигаем указатель на следующую строку}
            PName:= Pointer(LongInt(PName) + Length(SName)+1);
        End;
    End;
    {Освобождаем буфер}
    FreeMem(P);
end;

{Сортировка списка по двойному щелчку}
procedure TForm1.lbNameListDbClick(Sender: TObject);
begin
    lbNameList.Sorted:= True;
    lbNameList.Sorted:= False;
end;

end.
```


Основное диалоговое окно нашей тестовой программы показано на рис. 9.4. Введем в верхнее окно строку COM1 и нажмем кнопку **Получить NT-имя....** В поле **Nt-имя** должна отобразиться строка, похожая на `\Device\Serial0`.

Теперь введем в поле **DOS-имя** какое-нибудь новое имя, например, MyDevice, и нажмем кнопку **Добавить DOS-имя....** Результат мы можем увидеть, нажав кнопку **Получить полный список DOS-имен** или **Получить NT-имя....**

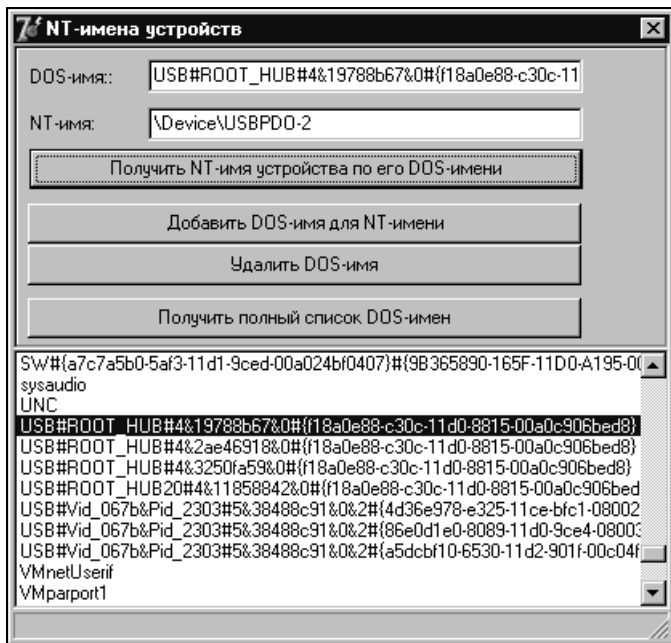


Рис. 9.4. Окно программы работы с именами устройств

В отличие от последовательных портов, имеющих привычные всем имена, USB-компоненты имеют имена, включающие GUID, например, для имени `USB#ROOT_HUB#4&19788b67&0#{f18a0e88-c30c-11d0-8815-00a0c906bed8}`

с помощью нашей программы мы получим NT-имя `\Device\USBPDO-2`.

Для просмотра имен удобно пользоваться программой WinObj (см. разд. 19.4.1).

9.3. Основные процедуры драйвера WDM

Оговорим сразу, что мы рассматриваем именно драйверы модели WDM, а не драйверы Windows 98 или Windows NT. Это важно, т. к. драйверы WDM, с одной стороны, должны содержать дополнительные процедуры для под-

держки PnP, а с другой, более логичны по структуре. Итак, в этом разделе мы перечислим основные процедуры драйвера WDM. Подробное описание процесса создания драйвера не входит в рамки этой книги, поэтому мы ограничимся исключительно практическим интересом, а всех желающих отсылаем к списку литературы [7, 8].

9.3.1. Процедура *DriverEntry*

Процедура *DriverEntry* является "точкой входа" драйвера. Заголовок этой процедуры показан в листинге 9.2.

Листинг 9.2. Заголовок процедуры *DriverEntry*

```
NTSTATUS DriverEntry(
    IN PDRIVER_OBJECT pDriverObject, // Адрес объекта драйвера
    IN PUNICODE_STRING pRegistryPath // Путь в регистре к подразделу драйвера
)
```

Получив от диспетчера в/в указатель на структуру *DRIVER_OBJECT* драйвер должен заполнить в ней следующие поля:

- ❑ *pDriverObject->DriverStartIo* — адрес процедуры *StartIo*, которая необходима для организации обработки очереди необработанных запросов;
- ❑ *pDriverObject->DriverExtension->AddDevice* — адрес процедуры *AddDevice*, которая будет вызываться при инициализации нового устройства;
- ❑ поля в массиве *pDriverObject->MajorFunction[IRP_MJ_xxx]* — драйвер должен регистрировать точки входа в собственные рабочие процедуры (разбирается далее);
- ❑ *pDriverObject->DriverUnload* — адрес процедуры *Unload*, которая будет вызываться перед выгрузкой драйвера.

Пример регистрации рабочих процедур показан в листинге 9.3, а набор констант *IRP_MJ_xxx* из файла *ntddk.h* — в листинге 9.4.

Листинг 9.3. Пример регистрации рабочих процедур драйвера¹

```
// пример из драйвера GiveIOEx
DriverObject->MajorFunction[IRP_MJ_CREATE] = GiveioCreateDispatch;
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = GiveioDeviceControl;
```

¹ Здесь и далее мы будем пользоваться примером драйвера *GiveIoEx* из [1].

Листинг 9.4. Описание констант IRP_MJ_xxx (ntddk.h)

```

#define IRP_MJ_CREATE                0x00
#define IRP_MJ_CREATE_NAMED_PIPE    0x01
#define IRP_MJ_CLOSE                 0x02
#define IRP_MJ_READ                   0x03
#define IRP_MJ_WRITE                 0x04
#define IRP_MJ_QUERY_INFORMATION     0x05
#define IRP_MJ_SET_INFORMATION       0x06
#define IRP_MJ_QUERY_EA              0x07
#define IRP_MJ_SET_EA                0x08
#define IRP_MJ_FLUSH_BUFFERS         0x09
#define IRP_MJ_QUERY_VOLUME_INFORMATION 0x0a
#define IRP_MJ_SET_VOLUME_INFORMATION 0x0b
#define IRP_MJ_DIRECTORY_CONTROL     0x0c
#define IRP_MJ_FILE_SYSTEM_CONTROL   0x0d
#define IRP_MJ_DEVICE_CONTROL        0x0e
#define IRP_MJ_INTERNAL_DEVICE_CONTROL 0x0f
#define IRP_MJ_SHUTDOWN              0x10
#define IRP_MJ_LOCK_CONTROL          0x11
#define IRP_MJ_CLEANUP               0x12
#define IRP_MJ_CREATE_MAILSLLOT      0x13
#define IRP_MJ_QUERY_SECURITY         0x14
#define IRP_MJ_SET_SECURITY           0x15
#define IRP_MJ_POWER                 0x16
#define IRP_MJ_SYSTEM_CONTROL        0x17
#define IRP_MJ_DEVICE_CHANGE         0x18
#define IRP_MJ_QUERY_QUOTA           0x19
#define IRP_MJ_SET_QUOTA             0x1a
#define IRP_MJ_PNP                   0x1b
#define IRP_MJ_PNP_POWER              IRP_MJ_PNP
#define IRP_MJ_MAXIMUM_FUNCTION      0x1b

```

Если драйверу интересен только один тип запросов, то можно воспользоваться константой `IRP_MJ_MAXIMUM_FUNCTION`, как показано в листинге 9.5. Все запросы будут обрабатываться процедурой `IrpHandler`, за исключением запросов `IRP_MJ_DEVICE_CONTROL`, которые будут поступать в процедуру `IrpDeviceControl`.

Листинг 9.5. Обработка одного типа запроса

```
int i;
for (i=0; i < IRP_MJ_MAXIMUM_FUNCTION; i++)
    DriverObject->MajorFunction[i] = IrpHandler;
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = IrpDeviceControl;
```

9.3.2. Процедура *AddDevice*

Основной обязанностью процедуры *AddDevice* (точнее говоря, той процедуры, которая была зарегистрирована в поле *DriverExtension->AddDevice* при вызове *DriverEntry*) является создание объекта устройства с использованием вызова *IoCreateDevice*.

Заголовок процедуры *AddDevice* показан в листинге 9.6, а скелетный пример организации самой рабочей процедуры — в листинге 9.7.

При необходимости подключения к объекту физического устройства (этот объект создается шинным драйвером) драйвер может вызвать функцию *IoAttachDevice*, передав ей указатель, взятый из параметра *PhysicalDeviceObject*.

Листинг 9.6. Заголовок процедуры *AddDevice*

```
NTSTATUS XxxAddDevice(
    IN PDRIVER_OBJECT DriverObject,
    IN PDEVICE_OBJECT PhysicalDeviceObject
);
```

Листинг 9.7. Пример процедуры *AddDevice*

```
NTSTATUS FilterAddDevice(
    IN PDRIVER_OBJECT DriverObject,
    IN PDEVICE_OBJECT PhysicalDeviceObject
)
{
    NTSTATUS                status = STATUS_SUCCESS;
    PDEVICE_OBJECT          deviceObject = NULL;
    PDEVICE_EXTENSION        deviceExtension;
    ULONG                   deviceType = FILE_DEVICE_UNKNOWN;
```

```

// Вызов IoIsWdmVersionAvailable(1, 0x20) возвращает TRUE
// в операционных системах после Windows 2000.
if (!IoIsWdmVersionAvailable(1, 0x20)) {
    deviceObject = IoGetAttachedDeviceReference(PhysicalDeviceObject);
    deviceType = deviceObject->DeviceType;
    ObDereferenceObject(deviceObject);
}

// Создание объекта драйвера-фильтра
status = IoCreateDevice (DriverObject,
    sizeof (DEVICE_EXTENSION),
    NULL, // без имени
    deviceType,
    FILE_DEVICE_SECURE_OPEN,
    FALSE,
    &deviceObject
);

if (!NT_SUCCESS (status)) {
    // выход, если ошибка
    return status;
}

// отладочная информация
DebugPrint (("AddDevice PDO (0x%x) FDO (0x%x)\n",
    PhysicalDeviceObject, deviceObject));

deviceExtension = (PDEVICE_EXTENSION) deviceObject->DeviceExtension;
deviceExtension->NextLowerDriver =
    IoAttachDeviceToDeviceStack(deviceObject, PhysicalDeviceObject);

// Ошибка означает сбой в системе Plug and Play
if(NULL == deviceExtension->NextLowerDriver) {
    IoDeleteDevice(deviceObject);
    return STATUS_UNSUCCESSFUL;
}

deviceObject->Flags |= deviceExtension->NextLowerDriver->Flags &
    (DO_BUFFERED_IO | DO_DIRECT_IO |

```

```

        DO_POWER_PAGABLE

    );

    deviceObject->DeviceType =
        deviceExtension->NextLowerDriver->DeviceType;
    deviceObject->Characteristics =
        deviceExtension->NextLowerDriver->Characteristics;
    deviceExtension->Self = deviceObject;

    // Установка начального состояния фильтра
    INITIALIZE_PNP_STATE(deviceExtension);

    DebugPrint(("AddDevice: %x to %x->%x \n", deviceObject,
        deviceExtension->NextLowerDriver,
        PhysicalDeviceObject));

    deviceObject->Flags &= ~DO_DEVICE_INITIALIZING;
    return STATUS_SUCCESS;
}

```

9.3.3. Процедура *Unload*

Обычно загруженный драйвер остается в системе до перезагрузки. Для того чтобы сделать драйвер выгружаемым, необходимо написать и зарегистрировать процедуру выгрузки *Unload*. Диспетчер в/в произведет вызов этой процедуры в момент ручной либо автоматической выгрузки драйвера.

Заголовок этой процедуры показан в листинге 9.8.

Листинг 9.8. Заголовок процедуры *Unload*

```

VOID GiveioUnload(
    IN PDRIVER_OBJECT pDriverObject // указатель на объект драйвера
)

```

Процедура *Unload* выполняет стандартный набор действий:

1. При необходимости драйвер может сохранять текущие настройки в системном реестре. При последующей загрузке драйвера эти данные могут быть использованы в процедуре *DriverEntry*.

2. Если разрешены прерывания для обслуживаемого устройства, то процедура Unload должна произвести их запрещение и отключение от объекта прерываний.
3. Символьная ссылка должны быть удалена из пространства имен, видимого пользовательскими приложениями. Это выполняется при помощи вызова функции IoDeleteSymbolicLink.
4. Объект драйвера должен быть удален вызовом функции IoDeleteDevice.
5. Если драйвер управляет многокомпонентным контроллером, необходимо повторить шаги 3 и 4 для каждого устройства, подключенного к контроллеру, а затем удалить сам объект контроллера при помощи вызова функции IoDeleteController.
6. Следует выполнить освобождение памяти, выделенной драйверу, во всех типах оперативной памяти.

Драйверы WDM модели выполняют почти все из этих действий в обработчике IRP_MJ_PNP запросов с субкодом IRP_MN_REMOVE.

Листинг 9.9 показывает пример написания процедуры Unload. Обратите внимание на преобразование имени драйвера в кодировку UNICODE.

Листинг 9.9. Пример процедуры Unload

```
// пример из драйвера GiveIOEx
#define DEVICE_NAME_STRING L"giveioex"

VOID GiveioUnload(IN PDRIVER_OBJECT DriverObject)
{
    WCHAR DOSNameBuffer[] = L"\\DosDevices\\" DEVICE_NAME_STRING;
    UNICODE_STRING uniDOSString;

    if (IOPM_local)
        MmFreeNonCachedMemory(IOPM_local, sizeof(IOPM));
    RtlInitUnicodeString(&uniDOSString, DOSNameBuffer);
    IoDeleteSymbolicLink (&uniDOSString);
    IoDeleteDevice(DriverObject->DeviceObject);
}
```

Важно отметить, что процедура Unload не вызывается в момент перезагрузки или выключения системы. При необходимости выполнения действий во время выключения следует делать это в обработчике запросов IRP_MJ_SHUTDOWN, причем объект устройства должен быть с помощью вызова

функции `IoRegisterShutdownNotification` занесен в очередь объектов, получающих уведомление о выключении.

9.3.4. Рабочие процедуры драйвера

Клиенты драйвера (т. е. пользовательские приложения или модули режима ядра) общаются с драйвером с помощью специальных структур данных, называемых *пакетами IRP* (Input/output Request Packet, пакет запроса в/в). При появлении запроса от приложения пользователя диспетчер в/в вызывает соответствующий обработчик драйвера, который был зарегистрирован в массиве `DriverObject→MajorFunction[]`, как показано в листинге 9.2.

Пакеты IRP являются структурами данных переменной длины и состоят из стандартного заголовка, содержащего общую учетную информацию, и одного или нескольких блоков параметров, называемых *ячейкой стека в/в* (I/O stack location). Структура пакета IRP показана на рис. 9.5.

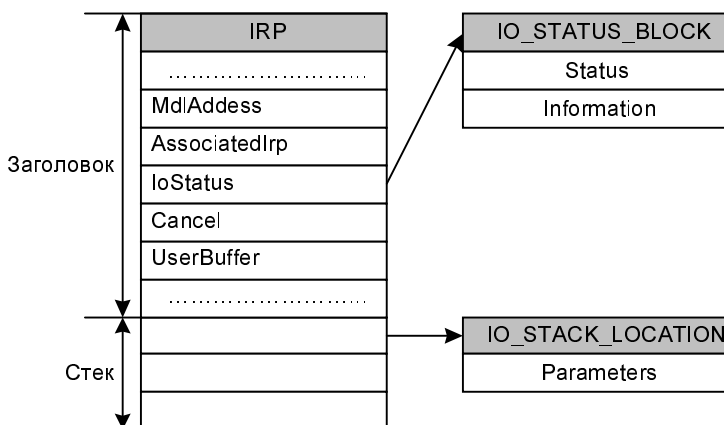


Рис. 9.5. Структура пакета IRP

Заголовок пакета

Описывать все поля заголовка пакета IRP не имеет смысла. Код драйвера может работать со следующими полями:

- ❑ `PMDL MdlAddress` — указатель на MDL-список (Memory Descriptor List, список дескрипторов памяти), если устройство поддерживает прямой в/в;
- ❑ `PVOID AssociatedIrp.SystemBuffer` — указатель на системный буфер для случая, когда устройство поддерживает буферизированный в/в;
- ❑ `IO_STATUS_BLOCK IoStatus` — код состояния (статус) запроса;

- ❑ `BOOLEAN Cancel` — индикатор того, что пакет `IRP` должен быть аннулирован;
- ❑ `PVOID UserBuffer` — адрес пользовательского буфера для в/в.

Поле структуры `IoStatus` фиксирует состояние данной операции в/в: когда драйвер готов завершить обработку пакета `IRP`, он устанавливает поле `IoStatus.Status` в значение `STATUS_xxx`. В поле `IoStatus.Information` записывается 0, если произошла ошибка или другое, определенное операцией в/в значение, чаще всего — число переданных или полученных байт данных (которое может быть равно и нулю).

Ячейки стека в/в

Основное назначение ячеек в/в состоит в хранении параметров запроса на в/в. Диспетчер в/в создает пакет `IRP` с числом ячеек в стеке, равном числу драйверных слоев, участвующих в обработке запроса. Любому драйверу в иерархии разрешен доступ к его собственной ячейке стека. Когда драйвер передает `IRP`-пакет нижнему драйверному уровню, он автоматически перемещает указатель стека в/в пакета таким образом, что он указывает на стековую ячейку для этого драйвера. Когда обработка пакета драйвером нижнего уровня завершена, указатель стека снова возвращается в исходное положение и указывает на ячейку стека для лежащего выше драйвера. Для получения указателя на текущую ячейку существует специальный системный вызов `IoGetCurrentStackLocation`.

Получив указатель на свою ячейку стека, т. е. указатель на структуру `IO_STACK_LOCATION`, драйвер может работать со следующими полями:

- ❑ `UCHAR MajorFunction` — код `IRP_MJ_xxx`, описывающий назначение операции;
- ❑ `UCHAR MinorFunction` — субкод операции;
- ❑ `PDEVICE_OBJECT DeviceObject` — указатель на объект устройства, которому был адресован данный запрос `IRP`;
- ❑ `PFILE_OBJECT FileObject` — файловый объект для данного запроса, если он задан.

В зависимости от значения `MajorFunction` поле `Parameters` представляется по-разному (т. е. оно описано как объединение (`union`)), например:

- ❑ для типа `IRP_MJ_CONTROL` в поле `Parameters` доступны следующие поля:
 - `ULONG OutputBufferLength`
 - `ULONG InputBufferLength`
 - `ULONG IoControlCode`
 - `PVOID Type3InputBuffer`

- ❑ для типов IRP_MJ_READ и IRP_MJ_WRITE в поле Parameters доступны следующие поля:
 - ULONG Length
 - ULONG Key
 - LARGE_INTEGER ByteOffset.

Рабочие процедуры драйвера

Тема книги не позволяет вдаваться в подробности рабочих процедур драйвера (Dispatch Routines). С практической точки зрения достаточно следующей информации:

- ❑ список поддерживаемых драйвером рабочих процедур (т. е. набор обрабатываемых кодов IRP_MJ_xxx) формируется драйвером в процессе выполнения процедуры DriverEntry;
- ❑ перед вызовом DriverEntry диспетчер в/в заполняет весь массив адресом процедуры _IoInvalidDeviceRequest, обеспечивая таким образом корректную обработку неподдерживаемых рабочих процедур;
- ❑ все процедуры драйвера используют один формат параметров и тип вызова, используя прототип, показанный в листинге 9.10.

Листинг 9.10. Прототип процедуры драйвера

NTSTATUS

DispatchProcedurePrototype(

// указатель на объект устройства, для которого

// предназначен IPR-пакет

IN PDEVICE_OBJECT DeviceObject,

// указатель на IRP-пакет

IN PIRP pIrp

)

{

// Возвращаемое значение

NTSTATUS ntStatus = STATUS_SUCCESS;

// Указатель на текущую ячейку стека

PIO_STACK_LOCATION irpSp;

// Длина входного буфера

ULONG inBufLength;

// Длина выходного буфера

ULONG outBufLength;

```

// Указатель на входной буфер
PULONG                LongBuffer;

// Получить указатель на ячейку стека
irpSp                = IoGetCurrentIrpStackLocation( pIrp );
// Длина входного буфера
inBufLength = irpSp->Parameters.DeviceIoControl.InputBufferLength;
// Указатель на входной буфер для метода доступа METHOD_BUFFERED
LongBuffer = (PULONG) pIrp->AssociatedIrp.SystemBuffer;
... ..
return ntStatus;
}

```

Рабочая процедура драйвера возвращает результат типа NTSTATUS:

- ❑ STATUS_SUCCESS — запрос обработан;
- ❑ STATUS_PENDING — ожидается обработка запроса;
- ❑ STATUS_xxx — код ошибки.

Существуют три варианта завершения рабочей процедуры:

- ❑ отклонение запроса — производится в случае, когда рабочая процедура не может обработать запрос, например, вследствие неустранимой ошибки. В этом случае следует выполнить следующие действия (листинг 8.10):
 - в поле `IoStatus.Status` записывается код ошибки;
 - поле `IoStatus.Information` обнуляется;
 - производится вызов `IoCompleteRequest` для завершения обработки запроса;
 - рабочая процедура возвращает тот же код ошибки, который был записан в поле `IoStatus.Status`.
- ❑ завершение работы с запросом. Многие запросы могут быть полностью обработаны без обращения к физическому устройству, за которое отвечает драйвер, например, получение дескриптора устройства или конфигурирование самого драйвера. В случае необходимости завершить обработку запроса следует выполнить следующие шаги (листинг 8.11):
 - записать в поле `IoStatus.Status` код успешного завершения STATUS_SUCCESS;
 - записать в поле `IoStatus.Information` корректное значение (зависит от запроса);

- выполнить вызов `IoCompleteRequest` для завершения обработки запроса;
 - вернуть код успешного завершения `STATUS_SUCCESS`.
- ❑ передача запроса на обработку. В простейшем случае драйвер может сразу завершить обработку запроса, например, в ответ на запрос чтения данных сразу же считать данные с последовательного порта. Однако асинхронный механизм в/в Windows предусматривает и другой метод работы. Рабочая процедура может поместить пакет `IRP` в очередь для последующей обработки и сразу же вернуть сообщение о том, что обработка пакета не была завершена. Для этого следует выполнить следующие шаги (листинг 9.11):
- выполнить вызов `IoMarkIrpPending`, информируя диспетчер в/в о том, что пакет поставлен в очередь на обработку;
 - выполнить вызов `IoStartPacket`, чтобы поместить пакет в системную очередь для последующей обработки процедурой `StartIO`.
 - вернуть код незавершенной обработки пакета `STATUS_PENDING`.

Листинг 9.11. Отклонение запроса рабочей процедурой

```
NTSTATUS
DispatchProcedurePrototype(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP pIrp
)
{
    // Возвращаемое значение
    NTSTATUS ntStatus = STATUS_SUCCESS;

    ... попытка обработки запроса ...

    // Обнаружили ошибку - отклоняем запрос
    // код ошибки - буфер слишком мал
    ntStatus = STATUS_BUFFER_TOO_SMALL;
    // ни одного байта не передано
    pIrp->IoStatus.Information = 0;
    // статус обработки
    pIrp->IoStatus.Status = ntStatus;
    // завершение обработки
    IoCompleteRequest(pIrp, IO_NO_INCREMENT);
```

```
// код завершения  
return ntStatus;  
}
```

Листинг 9.12. Завершение обработки запроса рабочей процедурой

```
NTSTATUS  
DispatchProcedurePrototype(  
    IN PDEVICE_OBJECT DeviceObject,  
    IN PIRP pIrp  
)  
{  
    // Возвращаемое значение  
    NTSTATUS ntStatus = STATUS_SUCCESS;  
  
    ... обработка запроса ...  
  
    // запрос успешно обработан  
    ntStatus = STATUS_SUCCESS;  
    // ни одного байта не передано  
    pIrp->IoStatus.Information = 0;  
    // статус обработки  
    pIrp->IoStatus.Status = ntStatus;  
    // завершение обработки  
    IoCompleteRequest(pIrp, IO_NO_INCREMENT);  
    // код завершения  
    return ntStatus;  
}
```

Листинг 9.13. Постановка запроса в очередь для обработки

```
NTSTATUS  
DispatchProcedurePrototype(  
    IN PDEVICE_OBJECT DeviceObject,  
    IN PIRP pIrp  
)  
{  
    ... предварительная обработка запроса ...  
}
```

```
// поставка запроса в очередь на обработку
IoMarkIrpPending(pIrp);
IoStartPacket(pDeviceObject, pIrp, 0, NULL);
// код завершения
return STATUS_PENDING;
}
```

Следует отметить два важных обстоятельства:

- ❑ вызов `IoCompleteRequest(pIrp,...)` может освободить память, занимаемую собственно пакетом, поэтому оператор `return(pIrp->IoStatus.Status);` может привести к непредсказуемым результатам;
- ❑ после возврата из рабочей процедуры диспетчер в/в завершает все запросы, не помеченные статусом `STATUS_PENDING`, однако не уведомляет об этом вышестоящие в очереди драйверы. Для корректного уведомления драйвер должен вызывать `IoCompleteRequest` в конце обработки запроса.

Набор кодов запросов `IRP_MJ_xxx` и соответствующие им функции `Windows API` пользовательского режима приведен в табл. 9.1. Единственным обязательным для обработки кодом является код `IRP_MJ_CREATE`, генерируемый при вызове `CreateFile`. При необходимости освобождения ресурсов при вызове `CloseHandle` драйвер должен обрабатывать код `IRP_MJ_CLOSE`. Необходимость обработки остальных кодов зависит от функциональности драйвера.

Таблица 9.1. Коды запросов *IRP* и соответствующие функции пользовательского режима

IRP код	Вызов Windows API* или действия
IRP_MJ_CREATE	CreateFile
IRP_MJ_CLEANUP	Очистка ожидающих обработки пакетов IRP при закрытии дескриптора драйвера при обработке вызова CloseHandle
IRP_MJ_CLOSE	CloseHandle
IRP_MJ_READ	ReadFile
IRP_MJ_WRITE	WriteFile
IRP_MJ_DEVICE_CONTROL	DeviceIoControl
IRP_MJ_INTERNAL_DEVICE_CONTROL	Действия по управлению устройством, доступные только для клиентов, работающих в режиме ядра (недоступно для вызовов пользовательского режима)

Таблица 9.1 (окончание)

IRP код	Вызов Windows API* или действия
IRP_MJ_QUERY_INFORMATION	Передача длины файла в ответ на вызов GetFileSize
IRP_MJ_SET_INFORMATION	Установка длины файла по вызову SetFileSize
IRP_MJ_FLUSH_BUFFERS	Запись или очистка служебных буферов при отработке вызовов, например: • FlushFileBuffres • FlushConsoleInputBuffer • PurgeComm
IRP_MJ_SHUTDOWN	Действия, которые нужно выполнить драйверу в процессе подготовки системы к завершению работы
IRP_MJ_PNP	Посылается системой Plug and Play во время нумерации устройств, распределения ресурсов и т. д.
IRP_MJ_DEVICE_CHANGE	Посылается при изменении состава оборудования

* Описание и параметры функций Windows API можно найти в справочной части книги в главе 16.

9.3.5. Обслуживание запросов IOCTL

Как показано в табл. 9.1, набор функций, соответствующих запросам IRP_MJ_ххх, ограничивается набором самих констант. Дополнительные запросы к драйверу формируются с помощью рабочей процедуры драйвера для кода IRP_MJ_DEVICE_CONTROL и, соответственно, функции DeviceIoControl.

Заголовок функции DeviceIoControl показан в листинге 9.14, а более подробное описание дано в справочной части книги, в главе 16.

Листинг 9.14. Заголовок функции DeviceIoControl

```
BOOL DeviceIoControl(
HANDLE hDevice,           // дескриптор драйвера
DWORD dwIoControlCode,    // код операции
LPVOID lpInBuffer,        // входной буфер
```

```
DWORD nInBufferSize,           // размер входного буфера
LPVOID lpOutBuffer,            // выходной буфер
DWORD nOutBufferSize,         // размер выходного буфера
LPDWORD lpBytesReturned,       // число переданных байт
LPOVERLAPPED lpOverlapped      // асинхронная информация
);
```

Дескриптор драйвера получается при вызове функции `CreateFile` или, другими словами, при открытии драйвера. Входной и выходной буферы позволяют обмениваться данными с драйвером.

Наибольший интерес представляет параметр `dwIoControlCode`, передающий в драйвер код выполняемой операции, называемый **IOCTL** (Input/Output Control code, код операции в/в). Коды IOCTL, передаваемые в драйвер, могут быть определены разработчиком драйвера и имеют строго определенный формат (рис. 9.6).

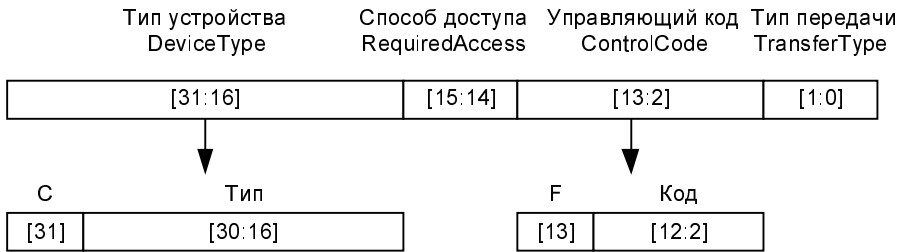


Рис. 9.6. Формат кода IOCTL

Для формирования кода IOCTL в Window DDK существует специальное макроопределение `CTL_CODE`, параметры и пример использования которого показаны в листингах 9.15 и 9.16. Описание параметров этого макроса дано в табл. 9.2. Для пользовательских программ на языке Delphi можно использовать функцию, код которой показан в листинге 9.17.

Листинг 9.15. Макроопределение CTL_CODE

```
#define CTL_CODE( DeviceType, Function, Method, Access ) (\
    ((DeviceType) << 16) | ((Access) << 14) |
    ((Function) << 2) | (Method)\
)
```


Листинг 9.16. Формирование кода IOCTL в драйвере

```
// Номера 32768–65535 зарезервированы для пользователя
#define GIVEIO_TYPE 40000

// Коды функций IOCTL от 0x800 до 0xFFF могут использоваться
#define IOCTL_IOPM_GET_ALL_ACCESS\
    CTL_CODE(GIVEIO_TYPE, 0x900, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_DISK_SET_PARTITION_INFO\
    CTL_CODE(IOCTL_DISK_BASE, 0x008, METHOD_BUFFERED,\
        FILE_READ_DATA | FILE_WRITE_DATA)
```

Листинг 9.17. Формирование кода IOCTL в Delphi

```
Const
    GIVEIO_TYPE      = 40000;
    METHOD_BUFFERED   = 0;
    FILE_ANY_ACCESS   = $0000;

// Функция формирования кода IOCTL
function Get_Ctl_Code(Nr: Integer): Cardinal;
begin
    Result:=
        (GIVEIO_TYPE shl 16) or
        (FILE_ANY_ACCESS shl 14) or
        (Nr shl 2) or
        METHOD_BUFFERED;
end;

// Пример формирования кода для листинга 8.15
Var
    IOCTL_IOPM_GET_ALL_ACCESS: Cardinal;
    ... ..
IOCTL_IOPM_GET_ALL_ACCESS:= Get_Ctl_Code($900);
```

Таблица 9.2. Параметры макроопределения CTL_CODE

Параметр	Описание
DeviceType	Код драйвера: • 0x0000—0x7FFFF — зарезервированы Microsoft • 0x8000—0xFFFF — определяются пользователем
ControlCode	Определяемые драйвером IOCTL-коды: • 0x000—0x7FF — зарезервировано Microsoft • 0x800—0xFFF — определяются пользователем
TransferType	Способ получения доступа к буферу: • 0: METHOD_BUFFERED • 1: METHOD_IN_DIRECT • 2: METHOD_OUT_DIRECT • 3: METHOD_NEITHER
RequiredAccess	Тип доступа: • 0x0000: FILE_ANY_ACCESS • 0x0001: FILE_READ_ACCESS • 0x0002: FILE_WRITE_ACCESS • 0x0003: FILE_READ_ACCESS FILE_WRITE_ACCESS

Заметим, что из табл. 8.2 видно, что флаги C и F, показанные на рис. 9.6 будут равны 0, если код IOCTL является зарезервированным кодом Microsoft.

Более подробное обсуждение значения констант, составляющих код IOCTL, выходит за рамки нашей книги, их можно найти в [8, 9]. Нас же прежде всего интересует способ обработки пользовательских кодов. Листинг 9.18 показывает пример обработки IOCTL-кодов в рабочей процедуре драйвера.

Листинг 9.18. Обработка IOCTL-кода в рабочей процедуре драйвера

```
// в процедуре DriverEntry регистрируем рабочую процедуру драйвера
// для обработки IOCTL-кодов
NTSTATUS DriverEntry(
    IN PDRIVER_OBJECT DriverObject,
    IN PUNICODE_STRING RegistryPath
)
```

```

{
    ... ..
    DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = IOCTL_DeviceControl;
    ... ..
    return STATUS_SUCCESS;
}

// Рабочая процедура драйвера
NTSTATUS
IOCTL_DeviceControl(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP pIrp
)
{
    // результат
    NTSTATUS ntStatus = STATUS_SUCCESS;
    // получить указатель на стек
    PIO_STACK_LOCATION irpSp = IoGetCurrentIrpStackLocation(pIrp);
    // получить код IOCTL
    ULONG ioctlCode = irpSp->Parameters.DeviceIoControl.IoControlCode;
    // размер входного буфера
    ULONG inSize = irpSp->Parameters.DeviceIoControl.InputBufferLength;
    // размер выходного буфера
    ULONG outSize = irpSp->Parameters.DeviceIoControl.OutputBufferLength;

    // обработка кодов IOCTL
    switch (ioctlCode)
    {
        // коды IOCTL_XXX формируются разработчиком драйвера
        case IOCTL_CODE1:
        {
            // проверяем входные параметры
            if ((inSize == 0) || (outSize == 0))
            {
                ntStatus = STATUS_INVALID_PARAMETER;
                break;
            }
            ... .. обработка IOCTL_CODE1 ... ..
            break;
        }
    }
}

```

```

case IOCTL_CODE2:
{
    // проверяем входные параметры
    if (inSize < 100)
    {
        ntStatus = STATUS_BUFFER_TOO_SMALL;
        break;
    }
    ... .. обработка IOCTL_CODE2 ... ..
    break;
}
}
pIrp->IoStatus.Status = ntStatus;
IoCompleteRequest( pIrp, IO_NO_INCREMENT );
return ntStatus;
}

```

Как видно из заголовка функции `DeviceIoControl`, драйверу передается входной и выходной буферы с данными. Драйвер получает указатели на эти данные в полях пакета `IRP`, зависящих от метода доступа, "зашифрованного" в коде номера `IOCTL` (см. табл. 9.2):

- ❑ при использовании метода `METHOD_BUFFERED` диспетчер в/в предоставляет единственный буфер в нестраничной памяти, достаточный для размещения входного и выходного буферов инициатора вызова. Адрес этой области размещается в поле `AssociatedIrp.SystemBuffer` (листинг 9.8). Затем производится копирование входного буфера с данными инициатора запроса в эту область. В поле `UserBuffer` заносится оригинальный адрес буфера для получения данных инициатора запроса. По завершении обработки запроса диспетчер в/в копирует содержимое выделенной области данных в выходной буфер инициатора запроса. Таким образом, драйверу предоставляется один буфер, даже если инициатор запроса указал два разных буфера;
- ❑ при использовании методов `METHOD_IN_DIRECT` и `METHOD_OUT_DIRECT` диспетчер в/в производит фиксацию (lock) выходного буфера инициатора запроса в физической памяти. Затем он производит построение списка дескрипторов памяти для выходного буфера и сохраняет указатель на этот список в поле `MdlAddress` пакета `IRP`. Кроме того, диспетчер в/в выделяет временную область в нестраничном пуле и сохраняет этот адрес в поле `AssociatedIrp.SystemBuffer` пакета `IRP`. Производится копирование содержимого входного буфера инициатора запроса в выделенный системный буфер, а в поле `UserBuffer` производится запись значения

NULL. После этого пакет IRP поступает в вызываемую рабочую процедуру драйвера;

- ❑ при использовании метода METHOD_NEITHER диспетчер в/в помещает адрес входного буфера инициатора запроса в поле `Parameters.DeviceIoControl.Type3InputBuffer` в текущей ячейке стека пакета IRP текущей операции в/в. В поле `UserBuffer` производится запись адреса выходного буфера инициатора запроса, где инициатор ожидает получить результаты выполнения операции. Оба этих адреса указывают в область памяти инициатора запроса.

9.4. Загрузка драйвера и обращение к процедурам драйвера

Теперь, когда мы выяснили, что основная деятельность драйвера производится в рабочих процедурах, разберемся с вопросом, как получить доступ к этим процедурам (при описании мы несколько упростим картину, позволив себе подойти к этому вопросу с практической точки зрения).

9.4.1. Процедура работы с драйвером

Процедура работы с драйвером выглядит следующим образом:

- ❑ загрузка драйвера с помощью вызова `CreateFile` (вызывает также рабочую процедуру `IRP_MJ_CREATE`);
- ❑ вызов либо стандартных рабочих процедур (например, процедуры `IRP_MJ_READ` с помощью функции `ReadFile`, см. табл. 9.1), либо пользовательских процедур, описанных в рабочей процедуре `IRP_MJ_DEVICE_CONTROL` с помощью вызова `DeviceIoControl` (см. *разд. 9.3.5*);
- ❑ закрытие драйвера с помощью вызова `CloseHandle`.

Описание и параметры использованных функций представлены в справочной части книги, в *главе 16*.

Загрузка драйвера производится по имени драйвера или устройства, например, как показано в листинге 9.19.

Листинг 9.19. Загрузка драйвера по имени драйвера или по имени устройства

```
// Загрузка по имени устройства
hComHandle:= CreateFile(
    '\\.\COM1' // передаем имя открываемого порта
    GENERIC_READ or GENERIC_WRITE, // ресурс для чтения и записи
    0, // неразделяемый ресурс
```

```

nil, // Нет атрибутов защиты
OPEN_EXISTING, // вернуть ошибку, если ресурс не существует
FILE_FLAG_OVERLAPPED, // асинхронный режим доступа
0 // Должно быть 0 для COM-портов
);
// Загрузка по имени драйвера
hDevice:= CreateFile('\\\\.\\giveioex',
    GENERIC_READ or GENERIC_WRITE,
    0, nil,
    OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL,
    0
);

```

В качестве имени устройства указывается одно из символьных имен физического устройства (см. *разд. 9.2*). Например, COM1, хотя и кажется именем устройства, на самом деле является символьным именем устройства \\Device\\Serial0. Для того чтобы драйвер был загружен, системе нужно знать физическое расположение файла драйвера. Для этого в специальной ветке реестра содержится запись о драйвере.

9.4.2. Регистрация драйвера

Для того чтобы драйвер мог быть обнаружен системой (т. е. по имени драйвера был найден и загружен физический файл), он должен быть зарегистрирован в специальной ветке реестра.

Регистрация драйвера может производиться с помощью INF-файла (см. *разд. 10.4*), с помощью специальных утилит или программно, с помощью специальной компоненты Windows, называемой SCM-менеджер (Service Control Manager, менеджер управления сервисами).

Регистрация с помощью SCM-менеджера

Преимущество использования SCM-менеджера состоит в том, что его функции позволяют динамически загружать и выгружать драйверы непосредственно из пользовательских программ, не прибегая к использованию Мастера установки оборудования. Таким образом, приложение само может определять время присутствия драйвера в операционной системе. Однако отметим, что не все драйверы могут быть загружены и запущены средствами SCM-менеджера.

Листинг 9.20 показывает регистрацию драйвера с помощью SCM-менеджера, состоящую из следующих действий:

□ регистрация сервиса:

- открытие SCM-менеджера с помощью вызова функции `OpenSCM_Manager` с флагом `SC_MANAGER_ALL_ACCESS`;
- регистрация драйвера как SCM-сервиса с помощью вызова функции `CreateService`;
- закрытие дескриптора сервиса с помощью вызова функции `Close_ServiceHandle`;
- закрытие дескриптора менеджера с помощью вызова функции `Close_ServiceHandle`;

□ старт сервиса:

- открытие SCM-менеджера с помощью вызова функции `OpenSCM_Manager` с флагом `SC_MANAGER_CONNECT`;
- открытие сервиса, соответствующего драйверу, с помощью функции `OpenService` с флагом `SERVICE_START`;
- старт сервиса с помощью функции `StartService`;
- закрытие дескриптора сервиса;
- закрытие дескриптора SCM-менеджера.

Процедура останова и разрегистрации драйвера, показанная в листинге 9.21, является "симметричной" копией процедуры старта:

□ останов сервиса:

- открытие SCM-менеджера с помощью вызова функции `OpenSCM_Manager` с флагом `SC_MANAGER_CONNECT`;
- открытие сервиса, соответствующего драйверу, с помощью функции `OpenService` с флагом `SERVICE_STOP`;
- останов сервиса с помощью функции `ControlService` с флагом `SERVICE_CONTROL_STOP`;
- закрытие дескриптора сервиса;
- закрытие дескриптора SCM-менеджера;

□ разрегистрация сервиса:

- открытие SCM-менеджера с помощью вызова функции `OpenSCM_Manager` с флагом `SC_MANAGER_ALL_ACCESS`;
- открытие сервиса, соответствующего драйверу с помощью функции `OpenService` с флагом `SERVICE_ALL_ACCESS`;

- удаление записи о драйвере с помощью функции DeleteService;
- закрытие дескриптора сервиса;
- закрытие дескриптора SCM-менеджера.

Листинг 9.20. Регистрация драйвера в реестре и старт сервиса

```
Const
    // имя драйвера
    DriverName : PChar= 'giveioex'#0;
    // имя файла драйвера
    FileDriver : String = 'giveioex.sys'#0;

// Процедура регистрации драйвера
Function TGiveIOEx.CreateService(SysPath : String) : Boolean;
var lpServiceArgVectors : PChar;
    hSCMan, hService: SC_HANDLE;
    DriverPath : String;
Begin
    Result:= False;

    {== Создание сервиса ==}
    {Сервис регистрируется в ветке реестра}
    {HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\giveioex}
    hSCMan:= WinSvc.OpenSCManager(
        Nil,          { локальный }
        Nil,          { SERVICES_ACTIVE_DATABASE }
        SC_MANAGER_ALL_ACCESS
    );
    If hSCMan = 0 then Exit;

    {Получаем полное имя к файлу драйвера}
    DriverPath:= SysPath + FileDriver;

    {Создаем сервис (регистрируем драйвер)}
    hService:= WinSvc.CreateService(hSCMan, Drivename, DriverName,
        SERVICE_ALL_ACCESS, SERVICE_KERNEL_DRIVER,
        SERVICE_DEMAND_START, SERVICE_ERROR_NORMAL,
        PChar(@DriverPath[1]),
        nil,nil,nil,nil);
```



```
{Файл не найден или сервис уже зарегистрирован}
If hService = 0 then begin
    MessageDlg(IntToStr(GetLastError), mtError, [mbOK], 0);
    CloseServiceHandle(hSCMan);
    Exit;
End;

{Регистрация успешна}
WinSvc.CloseServiceHandle(hService);
WinSvc.CloseServiceHandle(hSCMan);
Result:= True;
End;

// Старт зарегистрированного сервиса
Function TGiveIOEx.StartService : Boolean;
var lpServiceArgVectors : PChar;
    hSCMan, hService : SC_HANDLE;
    DriverPath : String;
Begin
    Result:= False;

// Старт сервиса
hSCMan := WinSvc.OpenSCManager(nil, nil, SC_MANAGER_CONNECT);
If hSCMan = 0 then Exit;

hService:= WinSvc.OpenService(hSCMan, DriverName, SERVICE_START);
If hService = 0 then begin
    CloseServiceHandle(hSCMan);
    Exit;
End;

lpServiceArgVectors:=nil;
WinSvc.StartService(hService, 0, lpServiceArgVectors);
WinSvc.CloseServiceHandle(hService);
WinSvc.CloseServiceHandle(hSCMan);
Result:= True;
End;
```

Листинг 9.21. Останов сервиса и разрегистрация драйвера

```
// Останов сервиса
Function TGiveIOEx.StopService : Boolean;
var serviceStatus    : TServiceStatus;
    hSCMan, hService : SC_HANDLE;
Begin
    Result:= False;
    {== Остановка сервиса}
    hSCMan:= WinSvc.OpenSCManager(Nil, Nil, SC_MANAGER_CONNECT);
    If hSCMan = 0 then Exit;

    hService:= WinSvc.OpenService(hSCMan, DriverName, SERVICE_STOP);
    If hService = 0 then begin
        WinSvc.CloseServiceHandle(hSCMan);
        Exit;
    End;

    WinSvc.ControlService(hService, SERVICE_CONTROL_STOP, serviceStatus);
    WinSvc.CloseServiceHandle(hService);
    WinSvc.CloseServiceHandle(hSCMan);
    Result:= True;
End;

// Разрегистрация сервиса
Function TGiveIOEx.RemoveService : Boolean;
var serviceStatus    : TServiceStatus;
    hSCMan, hService : SC_HANDLE;
Begin
    Result:= False;
    {== Удаление сервиса из реестра}
    hSCMan:= WinSvc.OpenSCManager(Nil, Nil, SC_MANAGER_ALL_ACCESS);
    If hSCMan = 0 then Exit;

    hService:= WinSvc.OpenService(hSCMan, DriverName, SERVICE_ALL_ACCESS);
    {Ошибка открытия сервиса}
    If hService=0 then begin
        CloseServiceHandle(hSCMan);
        Exit;
    End;
```

```

WinSvc.DeleteService(hService);
WinSvc.CloseServiceHandle(hService);
WinSvc.CloseServiceHandle(hSCMan);
Result:= True;
End;

```

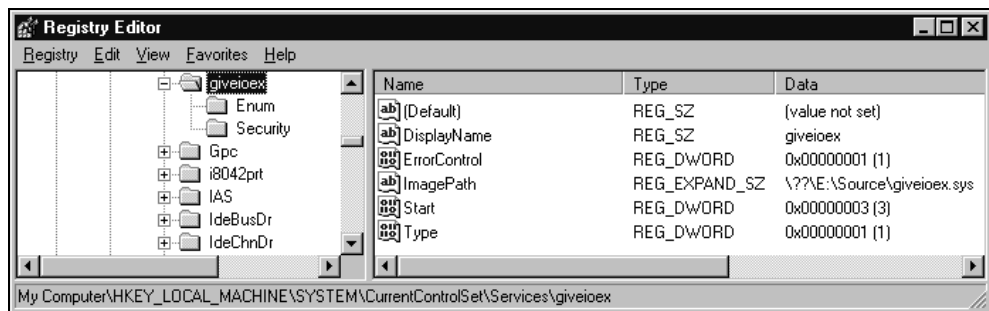


Рис. 9.7. Драйвер зарегистрирован

Рисунок 9.7 показывает результат успешной регистрации драйвера, после которой обращение к драйверу выглядит довольно просто (листинг 9.22).

Параметры драйвера в реестре

При регистрации драйвера в реестре создаются несколько стандартных записей (рис. 9.7).

- ❑ **DisplayName** (тип REG_SZ) — значение этого параметра описывает текст, используемый в служебных программах операционной системы, в частности, в программах панели управления. В случае если данный параметр не указан, используется имя драйвера;
- ❑ **ErrorControl** (тип REG_DWORD) — этот параметр описывает способ обработки ошибки при загрузке или инициализации драйвера. Возможные варианты приведены в табл. 9.3;
- ❑ **ImagePath** (тип REG_EXPAND_SZ) — значение этого параметра описывает полный путь к файлу, содержащему исполняемый код драйвера. По умолчанию значение этого параметра равно %system%\Drivers\имя_драйвера;
- ❑ **Start** (тип REG_DWORD) — значение этого параметра описывает стадию загрузки операционной системы, когда следует загружать драйвер. Возможные варианты приведены в табл. 9.4;

- ❑ **Type** (тип REG_DWORD) — значение этого параметра описывает тип драйвера. Некоторые значения этого параметра приведены в табл. 9.5;
- ❑ параметры подраздела **Enum** — подраздел Enum в ветке описания драйвера присутствует постоянно для драйверов, загруженных с помощью Мастера установки оборудования. Для драйверов, загружаемых при помощи SCM-сервиса, он появляется только после их удачного старта. В этом разделе присутствуют параметры Count (число обнаруженных устройств), NextInstance и параметры 1, 2 и т. д. Параметры 1, 2 и т. д. появляются только для удачно стартовавших драйверов, а их значения указывают на ветку HKLM\System\CurrentControlSet\Enum (где отражаются все когда-либо удачно стартовавшие драйверы).

Таблица 9.3. Значения параметра ErrorControl

Значение	Символьное имя	Описание
0x00	SERVICE_ERROR_IGNORE	Ошибки игнорируются, загрузка продолжается без уведомлений об ошибках в данном драйвере
0x01	SERVICE_ERROR_NORMAL	Ошибки игнорируются, но сообщения об ошибках выводятся, при этом загрузка продолжается
0x02	SERVICE_ERROR_SEVERE	Порядок загрузки нарушается и начинается заново с использованием набора параметров последней успешной загрузки, а если он уже используется, то ошибка игнорируется
0x03	SERVICE_ERROR_CRITICAL	Порядок загрузки нарушается и начинается заново с использованием набора параметров последней успешной загрузки, а если он уже используется, то загрузка прерывается и выводится сообщение об ошибке

Таблица 9.4. Значения параметра Start

Значение	Символьное имя	Описание
0x00	SERVICE_BOOT_START	Драйвер запускается загрузчиком ОС
0x01	SERVICE_SYSTEM_START	Драйвер запускается на стадии загрузки компонентов ядра ОС
0x02	SERVICE_AUTO_START	Драйвер будет запущен средствами SCM-менеджера после загрузки компонентов ядра ОС

Таблица 9.4 (окончание)

Значение	Символьное имя	Описание
0x03	SERVICE_DEMAND_START	Драйвер запущен пользовательским приложением при помощи средств SCM-менеджера
0x04	SERVICE_DISABLED	Драйвер не может быть запущен

Таблица 9.5. Значения параметра Type

Значение	Символьное имя	Описание
0x01	SERVICE_KERNEL_DRIVER	Драйвер режима ядра
0x02	SERVICE_FILE_SYSTEM_DRIVER	Драйвер файловой системы
0x04	SERVICE_ADAPTER	Драйвер адаптера

9.4.3. Обращение к рабочим процедурам

Обращение к стандартным рабочим процедурам производится с помощью стандартных функций Windows API, а к пользовательским — с помощью вызова функции DeviceIoControl.

Соответствие рабочих процедур и функций Windows API приведено в табл. 9.1. Например, вызов рабочей процедуры с кодом IRP_MJ_READ производится с помощью вызова функции ReadFile.

Вызов функции DeviceIoControl позволяет обращаться к рабочей процедуре IRP_MJ_DEVICE_CONTROL. Различение пользовательских процедур производится с помощью подкода (sub code), передаваемого при вызове DeviceIoControl (см. разд. 9.2.5).

Листинг 9.22 демонстрирует вызов пользовательской рабочей процедуры с кодом \$900.

Листинг 9.22. Загрузка и обращение к рабочим процедурам драйвера

```
Procedure TGiveIOEx.GiveIoForProcess(dwProcessId : Cardinal);
var hDevice : SC_HANDLE; Result : Cardinal;
begin
    // Загрузить драйвер с именем giveioex
    hDevice:= CreateFile('\\.\giveioex',
        GENERIC_READ or GENERIC_WRITE,
```

```

0,nil,
OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL,
0
);
// Обращение к рабочей процедуре с кодом IRP_MJ_DEVICE_CONTROL
// и подкодом $900
DeviceIoControl(hDevice, Get_Ctl_Code($900),
    @dwProcessId, SizeOf(dwProcessId), nil, 0, Result, nil);
// Закрытие драйвера
CloseHandle(hDevice);
end;
```

9.4.4. Хранение драйвера внутри исполняемого файла

В заключение раздела сделаем небольшое замечание. В приведенном ранее случае для работы программы требуется два файла — файл программы и файл драйвера. Для удобства установки и распространения программы можно использовать специальный ресурсный файл, позволяющий "встроить" файл драйвера в файл программы, и распространять один файл вместо двух. Для этого нужно в любом редакторе ресурсов создать бинарный ресурс, загрузив его из файла драйвера. Полученный файл ресурса (он имеет расширение `res`) с помощью директивы `{$R имя_файла.res}` включают в модуль программы, содержащий обращение к драйверу, а затем выполняют следующие шаги (листинг 9.23):

1. Создать файл драйвера.
2. Записать в созданный файл содержимое ресурса.
3. Работать, как обычно, с созданным файлом драйвера.
4. Удалить файл драйвера.

Таким образом, динамическое создание и уничтожение файла драйвера будет производиться прозрачно для программы.

Листинг 9.23. Динамическое создание файла драйвера

```

// Подключение файла ресурсов
{$R GiveIoEx.RES}

// получение пути к системному каталогу
function GetSystemDir : String;
```

```
var Buffer: array[0..1023] of Char;
begin
  SetString(Result, Buffer, GetSystemDirectory(Buffer, SizeOf(Buffer)));
end;

// получение имени драйвера
function GetDriverPath : String;
begin
  Result:= GetSystemDir + '\giveioex.sys';
end;

// создание файла драйвера из ресурса
Function TGiveIOEx.CreateFileFromResource : Boolean;
Var S: TFileStream; Rsrc: HRSRC; Res: THandle;
    Data: Pointer;
Begin
  Result := False;
  // 101 - номер ресурса в ресурсном файле
  Rsrc := FindResource(HInstance, MakeIntResource(101), RT_RCDATA);
  If Rsrc = 0 then Exit;

  Res:= LoadResource(HInstance, Rsrc);
  Try
    Data := LockResource(Res);
    If Data <> nil then
      Try
        S:= TFileStream.Create(GetDriverPath, fmCreate);
        Try
          S.WriteBuffer(Data^, SizeOfResource(HInstance, Rsrc));
        Finally
          S.Free;
        End;
        Result:= True;
      Finally
        UnlockResource(Res);
      End;
    Finally
      FreeResource(Res);
    End;
  End;
End;
```

```
// удаление файла драйвера
Procedure TGiveIOEx.RemoveFile;
var DriverPath : String;
Begin
  DriverPath:= GetDriverPath;
  Windows.DeleteFile(PChar(DriverPath));
End;
```

Естественно, такая процедура возможна только для динамической загрузки драйвера с помощью SCM-менеджера.

9.5. Инструменты создания драйверов

Процесс написания драйверов достаточно сложен и трудоемок, и, конечно, на рынке программного обеспечения появились программы, облегчающие написание и тестирование драйверов.

9.5.1. NuMega Driver Studio

Этот программный комплекс включается помощником, интегрирующимся со средой разработки Microsoft Visual Sdutio. Последовательно отвечая на вопросы помощника, можно получить работоспособный скелет драйвера. Для компиляции полученного кода требуются библиотеки и классы NuMega и Microsoft DDK.

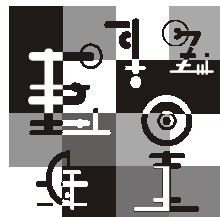
9.5.2. Jungo WinDriver

Пакет предназначен для разработки драйверов устройств, использующих стандарты PCI, Compact PCI, USB, ISA, ISA PnP, EISA и работающих под управлением операционных систем Windows 9X/ME/NT/2000. Позволяет обращаться к физической памяти, портам, устанавливать собственные обработчики аппаратных прерываний. Не требует наличия Windows DDK и программирования на уровне ядра. Используется графическая оболочка для диагностики оборудования и автоматической генерации кода на языках C/C++ или Pascal (Delphi).

9.5.3. Jungo KernelDriver

Пакет предназначен для разработки драйверов устройств, использующих стандарты PCI, Compact PCI, USB, ISA, ISA PnP, EISA и работающих на уровне ядра под управлением операционных систем Windows 9X/ME/NT/2000. Обеспечивает более высокую производительность, чем WinDriver. Требуется наличия Windows DDK. Используется графическая оболочка для диагностики оборудования и автоматической генерации кода.

Глава 10



Спецификация PnP для USB

Windows обнаружила перемещение указателя мыши.
Изменения вступят в силу после перезагрузки системы.

10.1. Общие сведения о системе Plug and Play

Появление огромного числа моделей периферийных устройств привело к очевидной невозможности их ручного конфигурирования. Выбор портов, номеров аппаратных прерываний, ячеек памяти... А ведь все параметры надо выставить корректно, исключая конфликты оборудования. При этом следует учитывать, что некоторые устройства умеют "подвинуться" на другие номера прерываний, а некоторые — требуют фиксированного номера. Другими словами, очевидна необходимость автоматического конфигурирования устройств.

10.1.1. Задачи и функции Plug and Play

Протокол Plug and Play (дословно, "подключил и играй") позволяет достаточно просто подключать новое оборудование. Перед началом работы система (BIOS при начальной загрузке, Windows при запуске) опрашивает устройства, узнает их требования к системным ресурсам и пытается бесконфликтно разделить ресурсы между устройствами. Если это не удастся, конфликтующие устройства будут работать некорректно. В этом случае необходимо вручную внести корректировки в настройки. Многие устройства, например, не поддерживают самостоятельное переключение диапазонов используемых ресурсов, но могут настраиваться с помощью специальных переключателей. Спецификация Plug and Play предусматривает также "горячее" подключение устройств, т. е. подключение во время работы.

Итак, основными функциями системы PnP (это сокращенное обозначение Plug and Play) являются:

- ☐ определение подключаемых устройств;
- ☐ идентификация устройств, уведомление ОС об их появлении;

- ❑ определение отключения устройства и уведомление об этом ОС;
- ❑ автоматическое конфигурирование устройств без вмешательства пользователя.

Для того чтобы устройство смогло поддерживать спецификацию PnP, должно быть выполнено несколько условий:

- ❑ устройство должно уметь выполнять программное конфигурирование. Должна существовать возможность установки портов ввода/вывода, задействованных прерываний и ресурсов памяти с помощью программного конфигурирования, исключая механическое конфигурирование с помощью переключателей;
- ❑ устройство и шина, к которому оно подключается, должны информировать систему о подключении, отключении или изменении конфигурации устройств;
- ❑ необходимые драйверы должны устанавливаться автоматически (за исключением запроса о местоположении нужных драйверов). Для этого устройство должно сообщать системе всю необходимую информацию;
- ❑ желательно, чтобы устройство поддерживало "горячее" подключение/отключение, т. е. подключение устройств без выключения компьютера. Естественно, это справедливо только для устройств, подключаемых к соответствующим шинам.

Первые попытки реализации PnP были сделаны в Windows 95. В Windows 98 разработка была продолжена, а в Windows 2000/XP/2003 "горячее" подключение устройств считается штатным режимом работы.

10.1.2. Запуск процедуры PnP

Windows производит краткий опрос наличия устройств при старте. Существует возможность запустить этот процесс вручную, по необходимости.

Например, в Windows 98, необходимо зайти в меню **Пуск** → **Настройка** → **Панель управления** и выбрать иконку **Установка оборудования**. После этого будет показано диалоговое окно, похожий на рис. 10.1 (вид диалогового окна может немного меняться в зависимости от версии Windows). Нажатие на кнопку **Далее** (точнее говоря, надо два раза нажать **Далее**, ответив утвердительно на предупреждающее диалоговое окно) запускает процедуру быстрого поиска новых устройств. Те же действия можно выполнить, нажав правую кнопку мыши на иконке **Мой компьютер**.

В Windows 2000/XP активизировать процедуру PnP-опроса можно и из окна **Device Manager** (Менеджер устройств), нажав кнопку **Scan for hardware changes** (Обнаружение изменений аппаратуры) (рис. 10.2).

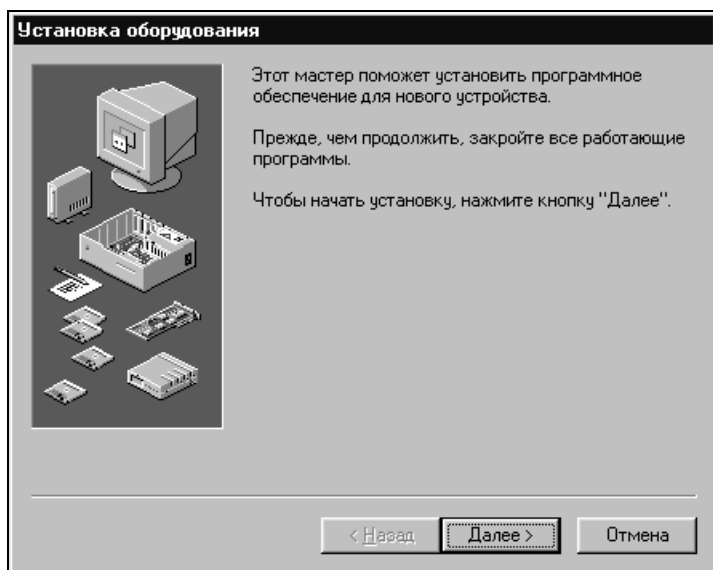


Рис. 10.1. Диалоговое окно поиска нового оборудования (Windows 98)

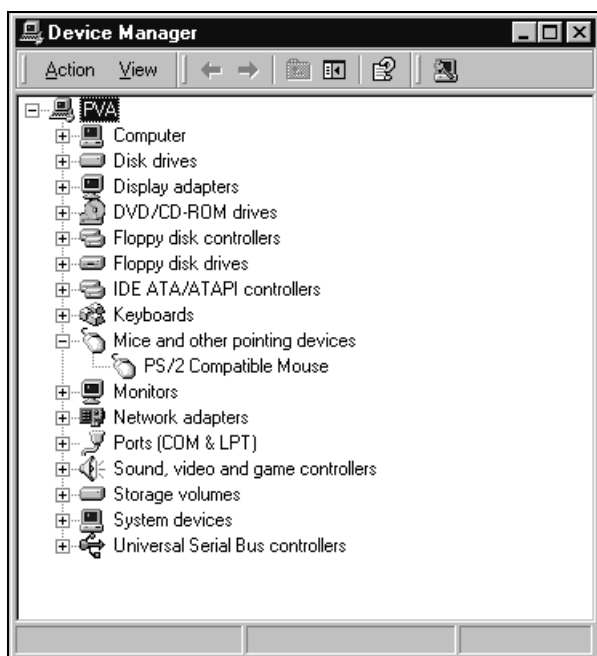


Рис. 10.2. Окно менеджера устройств (Windows 2000)

Полезно

В Windows 2000/XP можно сделать иконку на рабочем столе для быстрого запуска Менеджера устройств. Для этого надо создать ярлык к файлу devmgmt.msc, расположенному в каталоге %WINDOSW%\system32.

В Windows 2000/XP диалоговое окно добавления нового оборудования можно вызвать и программными средствами, как показано в листинге 10.1.

Листинг 10.1. Запуск диалогового окна добавления нового оборудования (Windows 2000/XP)

```
type
  TCplApplet = function(
    hwndCpl: HWND;
    uMsg: DWORD; lParam1, lParam2: Longint
  ): Longint; stdcall;

// по нажатию кнопки
procedure TForm1.Button1Click(Sender: TObject);
var
  APMModule : THandle;
  Applet    : TCplApplet;
begin
  // Загрузка CPL-библиотеки
  APMModule:= LoadLibrary('hdwwiz.cpl');
  // Если ошибка загрузки - выход
  if APMModule <= HINSTANCE_ERROR then Exit;
  // Точка входа
  Applet:= TCplApplet(GetProcAddress(APModule, 'CplApplet'));
  // Передать сообщение CPL_DBLCLK - "запустить по двойному щелчку"
  Applet(0, 5 {= CPL_DBLCLK}, 0, 0);
  // Освободить ссылку на библиотеку
  FreeLibrary(APModule);
end;
```

10.1.3. Программные компоненты PnP

Поддержка PnP в операционной системе осуществляется следующими компонентами (рис. 10.3):

- ❑ *PnP-менеджер* — состоит из двух частей: одна работает в режиме ядра, другая — в пользовательском режиме. Часть, работающая в режиме ядра,

взаимодействует с аппаратурой и другими программными компонентами, функционирующими в режиме ядра, обеспечивая правильным определением и конфигурированием аппаратуры. Часть, работающая в пользовательском режиме, взаимодействует с компонентами пользовательского интерфейса, позволяя диалоговым программам делать запросы и изменять конфигурацию установленного PnP программного обеспечения;

- ❑ *Менеджер управления энергопитанием* (Power Manager) — определяет и обрабатывает события энергосбережения;
- ❑ *Системный реестр Windows* (Windows System Registry) — является базой данных установленного аппаратного и программного обеспечения, поддерживающего спецификацию PnP. Содержимое реестра помогает драйверам и другим компонентам при определении ресурсов, используемых любым конкретным устройством;
- ❑ *INF-файлы* — каждое устройство должно быть полностью описано файлом, который используется при установке управляемого им драйвера. INF-файл содержит всю необходимую информацию о драйверах, действиях, необходимых при установке и удалении устройства, ключах системного реестра и т. д.;
- ❑ *драйверы для PnP-устройств* — их можно разделить на две категории: драйверы WDM и NT. Последние являются "унаследованными" от Windows NT и могут не полностью поддерживать архитектуру WDM. Драйверы WDM по определению полностью соответствуют требованиям взаимодействия по правилам PnP.

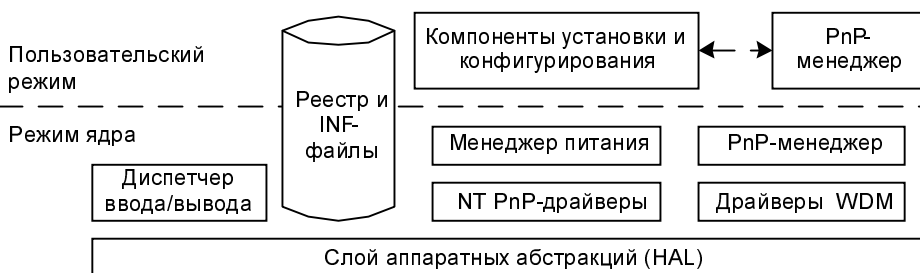


Рис. 10.3. Программные PnP-компоненты Windows 2000/XP

10.2. Plug and Play для USB

Согласно спецификации USB, любое USB-устройство должно соответствовать спецификации PnP. Шина USB поддерживает динамическое подклю-

чение и отключение устройств "по определению". Нумерация устройств шины является постоянным процессом, отслеживающим изменения физической топологии.

10.2.1. Конфигурирование устройств USB

При начальном подключении или после сброса производится начальное конфигурирование. Хаб определяет подключение и отключение устройств к своим портам и сообщает состояние портов при запросе от хоста. Хост разрешает работу порта и адресуется к устройству через канал управления, используя нулевой порт.

Хост определяет, является ли новое подключенное устройство хабом или функцией, и назначает ему *уникальный адрес USB*. Хост создает канал управления с этим устройством, используя назначенный адрес и нулевой номер конечной точки.

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 заинтересованному ПО.

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.

10.2.2. Нумерация устройств USB

Нумерация устройств (Enumeration), подключенных к шине, осуществляется динамически по мере их подключения (или включения их питания) без какого-либо вмешательства пользователя или клиентского ПО. Процедура нумерации выполняется следующим образом.

1. **Включение устройства.** Пользователь подключает устройство к порту, или подается питание на устройство, уже подключенное к порту. Устройство может подключаться к корневому или любому другому хабу. Хаб подает питание на порт и устройство переходит в состояние *Питание подано* (Powered).
2. **Хаб определяет подключение устройства.** Хаб, производящий постоянный мониторинг каждого порта, определяет, что к порту подключено устройство (см. разд. 18.1.7). Определив подключение устройства, хаб продолжает подавать питание на устройство, но пока не передает данные, т. к. устройство еще не готово их принимать.

3. **Хаб информирует хост о новом устройстве.** Хаб, к которому подключилось устройство, информирует хост о смене состояния своего порта ответом на опрос состояния. Каждый хаб имеет специальное прерывание (точнее, конечно, канал типа Interrupt) для передачи таких уведомлений. Когда хост узнает о подключении нового устройства, он посылает запрос GET_STATUS для получения дополнительной информации.
4. **Хаб проверяет режим устройства.** Хаб проверяет, является ли устройство низкоскоростным или полноскоростным, и отправляет эту информацию в ответ на запрос GET_STATUS. Спецификация USB 1.x позволяет хабу производить определение скоростного режима и после сброса, но USB 2.0 требует знания режима до сброса.
5. **Хаб подает устройству сигнал сброса.** Когда хост узнает о появлении нового устройства, хост-контроллер посылает хабу запрос SET_FEATURE, который указывает хабу произвести сброс устройства. Хаб производит сброс только того порта, к которому подключено новое устройство, другие хабы и порты шины не затрагиваются.
6. **Хост определяет возможность работы устройства в режиме HS.**
7. **Хаб устанавливает соединение между устройством и шиной.** Хост проверяет, что сброс устройства произведен. Для этого хост посылает запрос GET_STATUS. Если устройство не отвечает, хост повторяет запрос. Состояние устройства после сброса называется *Основным состоянием* (Default state). В этом состоянии регистры устройства сброшены, а устройство готово к обмену по нулевому каналу.
8. **Хост определяет конфигурацию нулевой точки.** Хост посылает запрос GET_DESCRIPTOR для того, чтобы узнать размер максимального пакета для Основного канала. Хост посылает запрос по адресу 0 конечной точке номер 0. Так как в один момент времени хост будет работать только с одним (обнаруженным) устройством, то на этот запрос откликнется только одно устройство, даже если к шине подключено их несколько. Устройство отвечает восьмибайтовым дескриптором, в котором содержится максимальный размер пакета, поддерживаемый конечной точкой 0.
9. **Хост назначает устройству уникальный адрес,** посылая запрос SET_ADDRESS. Устройство посылает хосту подтверждение и переходит в состояние *Адресовано* (Addressed). С этого момента любой обмен с устройством возможен только по этому адресу. Адрес устройства верен до отключения устройства, во время следующего включения устройство может получить другой адрес.
10. **Хост считывает конфигурацию устройства,** включая заявленный потребляемый ток от шины. Хост посылает запрос GET_DESCRIPTOR по новому адресу. Дескриптор, посылаемый устройством, содержит максимальный

размер пакета для нулевой конечной точки, число поддерживаемых устройством конфигураций и другую информацию об устройстве.

11. **Хост ищет и загружает драйвер устройства.** После того как хост узнал всю информацию об устройстве, он ищет наиболее подходящий драйвер и загружает его. При поиске драйвера Windows проверяет поля Vendor, Product ID и Release Number в INF-файлах. Если такой INF-файл не найден, Windows пытается найти драйвер, согласно классу, подклассу и типу протокола, полученным от устройства.
12. **Драйвер выбирает конфигурацию.** Драйвер посылает запрос SET_CONFIGURATION. Многие устройства имеют только одну возможную конфигурацию, но если устройство поддерживает несколько конфигураций, драйвер выберет либо первую, либо базовую, либо попросит пользователя выбрать нужную конфигурацию. Исходя из полученной информации, хост конфигурирует все имеющиеся конечные точки данного устройства, которое переводится в состояние *Сконфигурировано* (Configured). Теперь хаб позволяет устройству потреблять от шины полный ток, заявленный в конфигурации. Устройство готово.

Когда устройство отключается от шины, хаб уведомляет об этом хост и работа порта запрещается, а хост обновляет свою текущую топологическую информацию.

10.2.3. PnP-идентификаторы устройств USB

Каждое устройство, спроектированное по спецификации PnP, должно иметь идентификатор, который однозначно определяет модель данного устройства. Этот идентификатор должен быть предоставлен шинному аппаратному обеспечению (и, соответственно, шинному драйверу) по поступлении запроса. Секция описания модели (см. разд. 10.4.5) содержит поле hw_id, играющее роль идентификатора модели.

Идентификатор устройства должен иметь строго определенный для данного класса устройств формат. Для устройств USB идентификатор имеет следующий формат:

```
USB\Vid_vvvv&Pid_dddd&Rev_rr
```

Здесь vvvv — идентификатор поставщика (поле idVendor дескриптора устройства, см. разд. 4.1.3), зарегистрированный в Комитете USB производителей; dddd — идентификатор, присвоенный производителем данной модели устройства (поле idProduct); rr — номер версии разработки. Все эти поля вводятся как шестнадцатеричные числа.

В INF-файле допустимо указывать усеченные варианты идентификаторов, например:

```
USB\Vid_vvvv&Pid_dddd
```

```
USB\Class_cc&SubClass_ss_Prot_pp
```


USB\Class_cc&SubClass_ss

USB\Class_cc

Здесь *cc* — код базового класса из полученного дескриптора устройства или дескриптора интерфейса данного USB-устройства; *ss* — код подкласса; *pp* — идентификатор протокола.

Примеры USB-идентификаторов:

❑ USB\VID_040A&PID_0100 — цифровая USB-камера Kodak;

❑ USB\ROOT_HUB20 — USB-хаб;

❑ USB\VID_067B&PID_2303 — USB-мобильный телефон.

10.3. Получение списка USB-устройств

Для получения списка USB-устройств используются *функции инсталляции* (Setup API Function). Эти функции содержатся в динамической библиотеке SetupAPI.Dll. Объем нашей книги не позволяет привести описание всех функций (всего их около 600), поэтому мы опишем только необходимые для работы.

Если в C++ вызов функций Setup API проблем не вызывает, то в Delphi придется подключать функции библиотеки вручную (как показано в листинге 10.2) либо использовать готовые классы, например, SetupApi.Pas из библиотеки JEDI (<http://delphi-jedi.org>).

Листинг 10.2. Подключение функций SetupAPI.DLL

```
// имя динамической библиотеки
const
    SetupApiModuleName = 'SetupApi.dll';
// описание внешних функций
function SetupDiGetClassDevs;
    external SetupApiModuleName name 'SetupDiGetClassDevsA';
function SetupDiGetClassDevs;
    external SetupApiModuleName name 'SetupDiGetClassDevsA';
function SetupDiEnumDeviceInfo;
    external SetupApiModuleName name 'SetupDiEnumDeviceInfo';
function SetupDiGetDeviceRegistryProperty;
    external SetupApiModuleName name 'SetupDiGetDeviceRegistryPropertyA';
function SetupDiDestroyDeviceInfoList;
    external SetupApiModuleName name 'SetupDiDestroyDeviceInfoList';
```

Для получения списка устройств используется такая последовательность действий:

1. Получение дескриптора класса устройств (дескриптор либо корневого класса, либо класса с конкретным GUID) с помощью вызова функции `SetupDiGetClassDevs`.
2. Вызов функции `SetupDiEnumDeviceInfo` для получения описателя очередного устройства.
3. Вызов функции `SetupDiGetDeviceRegistryProperty` для получения информации об устройстве.
4. Повторение шагов 2—3 до тех пор, пока функция `SetupDiEnumDeviceInfo` не вернет `False`.

Для получения списка всех устройств функции `SetupDiGetClassDevs` вместо идентификатора класса передается `nil`, а в поле флагов записывается `DIGCF_ALLCLASSES`, а для получения дескриптора конкретного класса в эту функцию передается идентификатор нужного класса (см. табл. 10.1).

Для примера мы создадим небольшую программу, позволяющую получить либо список всех устройств в системе, либо устройств одного из следующих классов:

- ☐ видеоадаптеры;
- ☐ USB-устройства;
- ☐ HID-устройства.

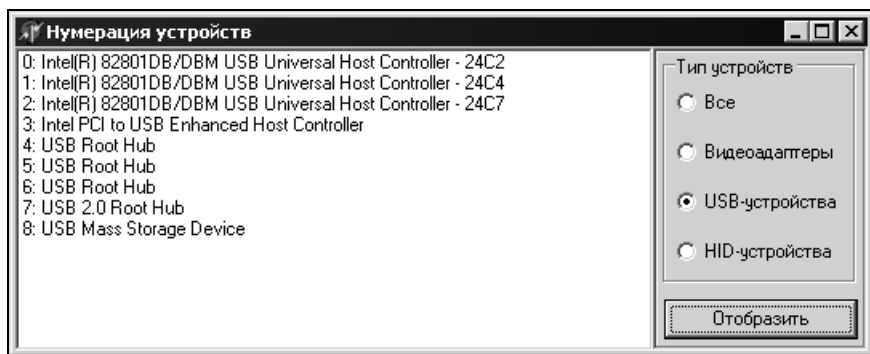


Рис. 10.4. Программа получения списка устройств

Получение списка видеоадаптеров сделано исключительно для упрощения тестирования (USB-устройства могут отсутствовать, а видеоадаптер есть всегда). Отметим также, что классы USB-устройств и HID-устройств различа-

ются. К первому классу относятся хост-контроллер, корневой хаб, флеш-диски и т. д., а устройства HID-класса мы будем рассматривать в *главе 8*.

Наша тестовая программа будет состоять из одной формы, показанной на рис. 10.4, а исходный код приведен в листинге 10.3.

Листинг 10.3. Получение списка устройств

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    Panel1: TPanel;
    rgDeviceType: TRadioGroup;
    Button2: TButton;
    procedure Button2Click(Sender: TObject);
  private
  public
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

Uses SetupApi;

procedure TForm1.Button2Click(Sender: TObject);
var
  Guid : TGUID;
  PnPHandle: HDevInfo;
```

```

DeviceInfoData : SP_DEVINFO_DATA;
DeviceInterfaceData: SP_DEVICE_INTERFACE_DATA;
i : Cardinal;
Success: LongBool;
DataT, buffersize: Cardinal;
buffer : PByte;

Const
    // GUID класса "видеоадаптеры"
    DisplayGuid : TGUID = '{4d36e968-e325-11ce-bfc1-08002be10318}';
    // GUID класса "HID-устройства"
    HidGuid      : TGUID = '{745a17a0-74d3-11d0-b6fe-00a0c90f57da}';
    // GUID класса "USB-устройства"
    USBGuid      : TGUID = '{36FC9E60-C465-11CF-8056-444553540000}';

begin
    // очистка лога
    ListBox1.Clear;

    If rgDeviceType.ItemIndex = 0 then begin // Все устройства
        PnPHandle:= SetupDiGetClassDevs(nil, nil, 0,
            DIGCF_PRESENT or DIGCF_ALLCLASSES);

    End else begin
        // Устройства конкретного класса
        Case rgDeviceType.ItemIndex of
            1: Guid:= DisplayGuid; // Все видеоадаптеры
            2: Guid:= USBGuid;      // Все USB-устройства
            3: Guid:= HidGuid;      // Все HID-устройства
        End;
        PnPHandle:= SetupDiGetClassDevs(@Guid, nil, 0, DIGCF_PRESENT);
    End;

    // Цикл по всем устройствам класса
    Try
        i:= 0;
        DeviceInfoData.cbSize:= SizeOf(SP_DEVINFO_DATA);
        DeviceInterfaceData.cbSize := SizeOf(SP_DEVICE_INTERFACE_DATA);
        Repeat
            // Получаем очередное устройство
            Success:= SetupDiEnumDeviceInfo(PnPHandle, i, DeviceInfoData);

```

```
If Success then begin
    buffer:= nil;
    buffersize:= 0;
    // Получаем информацию об устройстве
    while not SetupDiGetDeviceRegistryProperty
        (
            PnPHandle,
            DeviceInfoData,
            SPDRP_DEVICEDESC,
            DataT,
            buffer,
            buffersize,
            buffersize
        )
    do begin
        if (GetLastError() = ERROR_INSUFFICIENT_BUFFER) then begin
            if (buffer <> nil) then FreeMem(buffer);
            buffer:= AllocMem(buffersize);
        end else begin
            break;
        end;
    end;
    // Отобразить информацию об устройстве
    ListBox1.Items.Add(Format('%d: %s',[i, StrPas(PChar(buffer))]]));
    if (buffer <> nil) then FreeMem(buffer);
End;
Inc(i);
Application.ProcessMessages;
until not Success;

Finally
    // Освободить дескриптор класса
    SetupDiDestroyDeviceInfoList(PnPHandle);
End;
end;
end.
```

Немного странный цикл при вызове функции `SetupDiGetDeviceRegistryProperty` объясняется довольно просто: первый вызов этой функции производится с нулевым размером буфера (`bufferSize := 0`), поэтому функция возвращает ошибку "недостаточный размер буфера" (`ERROR_INSUFFICIENT_BUFFER`) и нужный размер буфера в переменной `bufferSize`. На втором проходе функции передается буфер нужного размера (см. разд. 4.1.3). Результат выполнения программы виден на рис. 10.4.

10.4. INF-файл

Для установки драйвера устройства Windows ищет специальный файл с расширением `inf`. Этот файл содержит всю информацию о действиях, которые необходимо произвести для установки драйвера: какие файлы нужно перенести и зарегистрировать в системе, какие ветки реестра необходимо создать и т. д. Кроме того, этот же файл содержит информацию о действиях, которые нужно произвести при удалении устройства из системы, а также дополнительную информацию о производителе устройства.

Структура INF-файла полностью совпадает с обычным INI-файлом: файл состоит из секций и некоторого набора ключей в них.

Важно

В операционных системах Windows 9x размер INF-файла не может превосходить 64 Кбайт. Для Windows NT/2000/XP ограничений нет. Максимальная длина любого поля в INF-файле составляет 512 символов.

Конечно же, описывать все поля INF-файла мы не будем. Для этого потребовалась бы книга в несколько раз больше этой. Мы постарались выбрать тот минимум полей и их значений, который будет нам необходим при создании и установке драйвера устройства.

10.4.1. Структура INF-файла

Инсталляционный INF-файл поделен на секции, каждая из которых начинается с идентификатора (имени секции), заключенного в квадратные скобки. Часть секций является обязательной, присутствие других секций зависит от назначения драйвера. Порядок следования секций не играет роли, важно лишь, чтобы секции имели корректные имена и были правильно соотнесены в перекрестных ссылках. Секция продолжается до начала следующей секции или до обнаружения конца файла.

Имя секции не должно содержать более 28 символов для Windows 9x и более 255 символов для Windows NT/2000/XP. Имя секции может содержать про-

белы, если ссылка на такую секцию заключена в кавычки, однако лучше ограничиваться именами без пробелов.

Записи внутри каждой секции описывают некоторые действия либо ссылаются на другие секции. Запись в секции представляет строку формата `entry = value[, value[, value...]]`

где `entry` является ключевым словом, либо маркером (ссылкой на значение; такие ссылки заключаются в знаки `%...%`). В операционных системах Windows 9x все запятые должны присутствовать в количестве, указанном в документации, а в секциях Windows NT замыкающие перечисление запятые можно опускать, если сами значения опущены.

Символ "точка с запятой" означает в следующей за ним позиции начало комментариев, которые продолжаются до конца строки. Исключение составляют строки, в которых символ "точка с запятой", заключен в кавычки. Комментарии не принимаются в рассмотрение при анализе файла.

При необходимости продолжить запись на следующей строке в последней позиции текущей строки ставится знак "обратный слэш" (`\`).

10.4.2. Секция *Version*

Секция `Version` содержит основную информацию о INF-файле. Рассмотрим значения ключей этой секции:

- ❑ `Signature = "signature-name"` — описывает версию Windows, для которой применим этот INF-файл. Значение ключа не зависит от регистра, но должно точно соответствовать указанной ниже записи строк (знаки "\$" в начале и конце строки обязательны).
 - `"$windows NT$"` — операционные системы NT;
 - `"$windows 95$"` — системы Windows 9x;
 - `"$chicago$"` — любая версия Window;
- ❑ `Class = class-name` — описывает тип устройства. Значение может быть одной из стандартных строк. Следует отметить, что если значение `Class` из INF-файла не совпадает со значением, переданным устройством, INF-файл все равно будет считаться правильным, а система установит устройство согласно значению из INF-файла.

Если указывается новый тип устройства, должен быть указан ключ `ClassGuid`, содержащий GUID нового типа устройства (см. табл. 10.1). Длина значения ключа `Class` не может превышать 32 символа;
- ❑ `ClassGuid = {nnnnnnnnn-nnnn-nnnn-nnnn-nnnnnnnnnnnnn}` — описывает GUID типа устройства (табл. 10.1). Обязательно указывается для нового типа устройства;

- ❑ `Provider = %INF-creator%` — содержит имя компании-разработчика INF-файла;
- ❑ `DriverVer = mm/dd/yyyy[,x.y.v.z]` — это информация о версии драйвера, устанавливаемого INF-файлом. Поле `mm/dd/yyyy` указывает дату драйвера, а необязательные параметры `x, y, v, z` — версию драйвера.

Если версия драйвера указывается, то каждая цифра должна быть в диапазоне от 0 до 65 535. Следует учитывать, что в версиях Windows ME, 2000 и XP это значение не учитывается при установке драйвера и служит только для отображения версии в Менеджере устройств (Device Manager). А в Windows XP SP1, Windows Server 2003 и более поздних версиях это значение учитывается программой установки драйвера.

При установке или переустановке драйвера Windows сравнивает дату уже установленного драйвера и всех подходящих INF-файлов и выбирает наиболее новый (кроме Windows 98/ME, которые не учитывают дату при установке драйверов). Если ключ `DriverVer` не указан, система считает дату как 00/00/0000. Таким образом, файлы без этого ключа автоматически считаются наиболее старыми.

Таблица 10.1. Некоторые значения ключей `Class` и `ClassGuid` секции `Version`

Имя	Описание	GUID
HIDClass	HID устройства	{745a17a0-74d3-11d0-b6fe-00a0c90f57da}
Ports	Порты (COM и LPT)	{4d36e978-e325-11ce-bfc1-08002be10318}
Image	Цифровые камеры, сканеры и т. д.	{6bdd1fc6-810f-11d0-bec7-08002be2092f}
Multifunction	Многофункциональные устройства, такие, как PCMCIA-модемы или сетевые адаптеры	{4d36e971-e325-11ce-bfc1-08002be10318}
System	Системные устройства	{4d36e97d-e325-11ce-bfc1-08002be10318}
USB	Хост-контроллер USB, хабы, но не USB-периферия	{36fc9e60-c465-11cf-8056-444553540000}
NoDriver	Драйвер отсутствует	{4d36e976-e325-11ce-bfc1-08002be10318}
Unknown	Другие устройства	{4d36e97e-e325-11ce-bfc1-08002be10318}

10.4.3. Секция *Manufacturer*

Секция *Manufacturer* описывает производителя одного или более устройств, устанавливаемых INF-файлом.

Секция имеет следующий формат:

```
[Manufacturer]
manufacturer-identifier
[manufacturer-identifier]
[manufacturer-identifier]
... ..
```

Каждый ключ секции содержит информацию, описывающую одну модель устройства, и должен располагаться на отдельной строке. Допустимо использование одного из следующих форматов:

1. *Manufacturer-name*
2. *%strkey%=models-section-name*
3. *%strkey%=models-section-name[,TargetOSVersion][,TargetOSVersion]*

В случае использования первого формата INF-файл должен содержать секцию с таким же именем (см. разд. 10.4.5). Например:

; пример из файла DECPSMW4.INF

```
[Manufacturer]
"Digital"
[Digital]

"Digital DEClaser 5100/Net"=D5100_MS.SPD,Digital_DEClaser_5100/Net
```

Второй формат аналогичен первому, но позволяет использовать строки для перевода, указываемые в секции *String*. Например:

; пример из файла CXPDPFPCI.INF

```
[Manufacturer]
%String1%=DIGI
[DIGI]
%String2%=CxpPCI1ST.Install,MF\digipci1st_dev0
[Strings]
String1="Digi International"
String2="Digi DataFire PCI 1 S/T (CXP)"
```

Естественно, в секции *Strings* должны быть определены все используемые ссылки *%strkey%*.

Третий формат описания доступен только, начиная с версии Windows XP. Он позволяет указывать список версий и типов Windows, для которых предназначен данный INF-файл. Программа установки будет выбирать тот драйвер, который наиболее близко подходит к перечисленным описаниям.

Формат строки TargetOSVersion следующий:

```
NT[Architecture][.OSMajorVersion][.OSMinorVersion][.ProductType]
[.SuiteMask]]]
```

Кратко рассмотрим поля этой строки:

- ☐ идентификатор NT указывает, что это описание поддерживается только версиями Windows семейства NT;
- ☐ поле Architecture описывает аппаратную платформу. Если указывается, должен быть или x86, или ia64;
- ☐ поле OSMajorVersion задает старший номер версии операционной системы. Для Windows XP это номер 5;
- ☐ поле OSMinorVersion задает младший номер версии операционной системы. Для Windows XP это номер 1;
- ☐ поле ProductType может быть одной из констант VER_NT_xxx, определенных в winnt.h, например:
 - 0x00000001 = VER_NT_WORKSTATION;
 - 0x00000002 = VER_NT_DOMAIN_CONTROLLER;
 - 0x00000003 = VER_NT_SERVER;
- ☐ поле SuiteMask — это маска из значений констант VER_SUITE_xxxx, определенных в winnt.h, например,
 - 0x00000001 = VER_SUITE_SMALLBUSINESS;
 - 0x00000002 = VER_SUITE_ENTERPRISE;
 - 0x00000004 = VER_SUITE_BACKOFFICE;
 - 0x00000008 = VER_SUITE_COMMUNICATIONS;
 - 0x00000010 = VER_SUITE_TERMINAL;
 - 0x00000020 = VER_SUITE_SMALLBUSINESS_RESTRICTED;
 - 0x00000040 = VER_SUITE_EMBEDDEDNT;
 - 0x00000080 = VER_SUITE_DATACENTER;
 - 0x00000100 = VER_SUITE_SINGLEUSERS;
 - 0x00000200 = VER_SUITE_PERSONAL;
 - 0x00000400 = VER_SUITE_SERVERAPPLIANCE.

Если ключи секции Manufacturer содержат определения версий ОС, то и платформы, и секции, на которые ссылаются указанные ключи, должны содержать те же определения, например:

```
[Manufacturer]
```

```
%MyName% = MyName,NTx86.5.1
```

```
... ..
```

```
[MyName]
%MyDev% = InstallA,hwid
... ..
[MyName.NTx86.5.1]
%MyDev% = InstallB,hwid
... ..
[InstallA.ntx86]      ; Windows 2000 (NT4-x86 будет также пытаться
                      ; читать эту секцию)
... ..
[InstallA]            ; Win98/WinME (Win95 также будет пытаться
                      ; читать эту секцию)
... ..
[InstallB]            ; Windows XP и позже, и только x86
... ..
```

10.4.4. Секция *DestinationDirs*

Секция *DestinationDirs* указывает один или несколько каталогов для копирования, удаления и переименования файлов.

Эта секция необходима в INF-файле, если он содержит либо ключ *CopyFiles*, либо ссылку на секции *CopyFiles*, *DelFiles* или *RenFiles*.

Каждый каталог описывается на отдельной строке секции и имеет формат:

```
[DestinationDirs]
[DefaultDestDir=dirid[,subdir]]
[file-list-section=dirid[,subdir]] ...
```

Ключ *DefaultDescDir*

Ключ *DefaultDescDir* указывает каталог по умолчанию для копирования, удаления и переименования файлов, которые не описаны в списке *file-list-section*.

Ключи *file-list-section*

Ключи *file-list-section* перечисляют имена файлов и их каталоги, если нужно установить каталоги, отличные от *DefaultDescDir*.

Ключ *dirid*

Ключ *dirid* указывает каталог, в котором будут находиться указанный файл или файлы. Это может быть или абсолютный путь, или номер (идентификатор) каталога. Абсолютный путь может ссылаться на секцию *String*.

Идентификатор каталога представляет собой целое число. Значения в диапазоне от -1 до 32 767 зарезервированы (табл. 10.2). Кроме того, следует учитывать, что в целях совместимости с Windows 98/ME значение 65 535 приравнивается к -1 .

Таблица 10.2. Таблица значение *dirid*

Значение	Описание
1	<i>SourceDrive:\pathname</i> (указывает каталог, из которого был установлен INF-файл)
10	Windows-каталог, т. е. %windir%
11	Системный каталог, т. е. %windir%\system32 для NT-систем, и %windir%\system для Windows 9x/ME
12	Каталог драйверов, т. е. %windir%\system32\drivers для NT-систем и %windir%\system\IoSubsys для Windows 9x/ME
17	Каталог INF-файлов, т. е. %windir%\INF
18	Каталог файлов помощи, т. е. Help directory %windir%\HELP
20	Каталог шрифтов
24	Системный диск, т. е. если Windows установлена в папку C:\winnt, то это будет "C:\"
30	Корневой каталог загрузочного диска (для NT-систем не обязательно совпадает с ID24)
50	Системный каталог для NT-систем, т. е. %windir%\system (только для NT)
53	Каталог пользовательского профиля
54	Каталог, где расположены файлы <i>ntldr.exe</i> и <i>osloader.exe</i> (для NT-систем)
-1	Абсолютный путь
16406	<i>All Users\Start Menu</i>
16407	<i>All Users\Start Menu\Programs</i>
16408	<i>All Users\Start Menu\Programs\Startup</i>
16409	<i>All Users\Desktop</i>
16415	<i>All Users\Favorites</i>
16419	<i>All Users\Application Data</i>
16422	<i>Program Files</i>

Таблица 10.2 (окончание)

Значение	Описание
16427	<i>Program Files\Common</i>
16429	<i>All Users\Templates</i>
16430	<i>All Users\Documents</i>

Ключ *subdir*

Ключ *subdir* обозначает каталог относительно *dirid*, например,

; пример из файла **ICW97.INF**

[DestinationDirs]

CopyHELP = 18 ; LDID_HELP

CopySYS = 11 ; LDID_SYS

CopyINF = 17 ; LDID_INF

DeleteICW2 = 24,%ProgramFiles%\%OLD_ICWDIR%

Обратите внимание, что подкаталог в *dirid* указывается через запятую, а не через слэш!

10.4.5. Секция описания модели

Секции описания моделей должны соответствовать ссылкам на модели из секции *Manufacturer* (см. *разд. 10.4.3*). Формат строк этой секции имеет вид:

Device-description=install-section-name,hw-id[,compatible-id...]

- ❑ *device-description* — это любая уникальная строка описания или ссылка на строку в секции *String*;
- ❑ *install-section-name* содержит имя секции, описывающей устройство;
- ❑ *hw-id* содержит строку, соответствующую *Hardware ID* в идентификаторе PnP, и может иметь один из форматов, перечисленных далее:
 - *Enumerator\device-id* — обычный формат для устройства, устанавливаемого по спецификации PnP. Например, *SERENUM\PVA0001*;
 - **device-id* — указывает, что устройство поддерживается различными сервисами. Например, **PNP0F01* идентифицирует Microsoft-мышь, которая имеет совместимый идентификатор *SERENUM\PNP0F01*;
 - *Device-class-id* — спецификатор шины, как он описан в аппаратной спецификации;
- ❑ в поле *compatible-id* содержатся один или несколько разделенных запятой типов устройств, совместимых с *hw-id*.

Пусть, например, секция `Manufacturer` ссылается на секцию `CompanyName`, которая содержит описание устройств, подключаемых к последовательному порту (SERENUM) и имеющих идентификаторы `PVA0001` и `PVA0000`. Для таких устройств описание драйвера будет находиться в секции `MyDevice_SECTION`:

```
[Manufacturer]
%Mfg%=CompanyName
[CompanyName]
%MyDeviceStr%=MyDevice_SECTION, SERENUM\PVA0001, PVA0000
[MyDevice_SECTION]
CopyFiles= MyDevice.CopyFiles
AddReg=MyDevice.AddReg
DelReg=MyDevice.DelReg
LogConfig=MyDevice.Config
```

10.4.6. Секция *xxx.AddReg* и *xxx.DelReg*

Эти секции описывают действия с реестром при установке и удалении устройства. Ссылка на имена секций дается из секции описания модели (см. разд. 10.4.5).

Формат строк этой секции имеет вид:

`Reg-root, [subkey], [value-entry-name], [flags], [value]`

- ☐ `reg-root` — идентификатор корневой ветки реестра. Может быть одним из обозначений:
 - `HKCR` = `HKEY_CLASSES_ROOT`;
 - `HKCU` = `HKEY_CURRENT_USER`;
 - `HKLM` = `HKEY_LOCAL_MACHINE`;
 - `HKU` = `HKEY_USERS`.
- ☐ `subkey` — содержит имя ветки реестра относительно `reg-root` или ссылку на строку секции `String`;
- ☐ `value-entry-name` — содержит имя ключа, создаваемого в реестре в ветке `reg-root\subkey`. Может быть или строкой, заключенной в кавычки, или ссылкой на значение из секции `String`;
- ☐ `flags` — необязательное, обозначает тип и свойства добавляемого значения. Значение представляет собой маску из констант `FLG_xxx` (табл. 10.3). Мы приведем самые существенные из этих констант.

Таблица 10.3. Таблица флагов секции AddReg (DelReg)

Флаг секции	Описание флага
\$00000001 FLG_ADDREG_BINVALUETYPE	Создать как "Бинарные данные"
\$00000010 FLG_ADDREG_KEYONLY	Создать subkey, но игнорировать value-entry-name и его значение
\$00000020 FLG_ADDREG_OVERWRITEONLY	Переписать значение value-entry-name, если оно существует. Если такого значения еще нет, не создавать
\$00001000 FLG_ADDREG_64BITKEY	Создание 64-битного значения (Windows XP и выше)
\$00000000 FLG_ADDREG_TYPE_SZ	Создание значения типа REG_SZ. Этот флаг является значением по умолчанию, если ключ flags не указан
\$00010000 FLG_ADDREG_TYPE_MULTI_SZ	Создание значения типа REG_MULTI_SZ. Для таких значений не требуется код NULL для завершения строки
\$00020000 FLG_ADDREG_TYPE_EXPAND_SZ	Добавление значения типа REG_EXPAND_SZ
\$00010001 FLG_ADDREG_TYPE_DWORD	Добавление значения типа REG_DWORD
\$00020001 FLG_ADDREG_TYPE_NONE	Добавление значения типа REG_NONE (только Windows 2000)

❑ value содержит значение ключа реестра reg-root\subkey\value-enentry-name и должно соответствовать типу, указанному флагами flags.

; пример из файла TAPI.INF

[add.reg]

```
HKU,Software\Microsoft\Windows\CurrentVersion\Telephony,,,  
HKU,Software\Microsoft\Windows\CurrentVersion\Telephony\HandoffPriorities,  
"RequestMakeCall",2,"DIALER.EXE"  
HKLM,Software\Microsoft\Windows\CurrentVersion\Telephony\Locations,"NextID",3,01,00,00,00  
HKLM,Software\Microsoft\Windows\CurrentVersion\RunOnce,TapiSetup2,,"tapiupr.exe"
```

10.4.7. Секция *xxx.LogConfig*

Секция *xxx.LogConfig* описывает системные ресурсы, требуемые драйвером. Ссылка на имя этой секции дается из секции описания модели (см. *разд. 10.4.5*).

В этой секции могут указываться ключи (мы снова выбрали наиболее используемые ключи):

- ❑ *DMAConfig* — описание ресурсов DMA;
- ❑ *IOConfig* — описание ресурсов портов;
- ❑ *MemConfig* — описание ресурсов памяти;
- ❑ *IRQConfig* — описание ресурсов IRQ.

Пример:

```
[MyDevice1.LogConfig]
DMAConfig=0,1
IOConfig=3bc-3be(3ff::),378-37a(3ff::),278-27a(3ff::)
[MyDevice2.LogConfig]
IOConfig=8@100-3ff%fff8(3ff::)
MemConfig=4000@C8000-EFFFF%FFFFC000
[MyDevice3.LogConfig]
IOConfig=110-11F(3FF::),130-13F(3FF::),150-15F(3FF::)
IRQConfig=3,4
MemConfig=4000@C0000-DFFFF%FFFFC000
```

Подробное описание всех ключей можно найти в MSDN.

10.4.8. Секция *xxx.CopyFiles*

Секция *xxx.CopyFiles* описывает операции над файлами, которые требуются для работы драйвера. Ссылка на имя этой секции дается из секции описания модели (см. *разд. 10.4.5*).

Секция содержит или имена файлов, или ссылку на секцию с именами файлов.

Каждое имя файла задается строкой вида:

```
Dest-file-name[,source-file-name][,temp-file-name][,flag]
```

Поле *dest-file-name* задает имя конечного файла. Если секция *source-file-name* не указана, то задает и имя исходного файла.

Поле *source-file-name* задает имя исходного файла. Если для операции копирования файлов имена исходного и конечного файлов одинаковы, то это поле можно пропустить.

Поле temp-file-name задает имя промежуточного файла, которое будет использовано, если конечный файл занят в данный момент времени. Используется только в Windows 9x, т. к. Windows NT и выше создают временные файлы автоматически. Операция над файлом будет выполнена при перезагрузке Windows.

Поле flag — необязательное поле, представленное в шестнадцатеричном или десятичном виде. Является маской одного или нескольких системных флагов COPYFLG_xxx (табл. 10.4).

Таблица 10.4. Таблица флагов секции CopyFiles

Флаги секции CopyFiles	Описание флагов
\$00000400 COPYFLG_REPLACEONLY	Копировать файл только в том случае, если он уже существует в конечном каталоге
\$00000800 COPYFLG_NODECOMP	Копировать исходный файл без распаковки
\$00000008 COPYFLG_FORCE_FILE_IN_USE	Копировать файл в файл temp и использовать его только после рестарта системы (эмуляция поведения занятости файла)
\$00000010 COPYFLG_NO_OVERWRITE	Не заменять файл, если он уже существует. Этот флаг не может использоваться с другими флагами
\$00001000 COPYFLG_REPLACE_BOOT_FILE	После выполнения операции требовать перезагрузки системы
\$00000020 COPYFLG_NO_VERSION_DIALOG	Не переписывать файл в конечном каталоге, если существующий файл более новый, чем копируемый
\$00000004 COPYFLG_NOVERSIONCHECK	Игнорировать версию файла. Переписывать конечный файл независимо от его версии
\$00000002 COPYFLG_NOSKIP	Не позволять пользователю отказаться от копирования файла

10.4.9. Секция Strings

Секция Strings используется для создания INF-файлов на нескольких языках. Другие секции могут ссылаться на ключи этой секции с помощью знаков %. Строка, заключенная в знаки %, означает подстановку значения переменной из этой секции.

Значения полей этой секции могут быть обычными строками или строками, заключенными в кавычки, если они содержат существенные пробелы до или после строки.

Для NT-платформы секция Strings может содержать индекс языка, для которого указываются строки. Индекс представляет собой объединения констант LANG_xxx и SUBLANG_xxx. Например,

```
[Strings]           ; Обычный набор строк
DiskName="My Excellent Software"
LocaleSubDir="English"

[Strings.0407]      ; Набор строк для немецкого языка (0407)
DiskName="Meine ausgezeichnete Software"
LocaleSubDir="German"

[Strings.0419]      ; набор строк для русского языка (0419)
[Strings.0422]      ; набор строк для украинского языка (0422)
```

10.4.10. Связи секций

Собрав все перекрестные ссылки, мы получим картину связей между основными секциями.

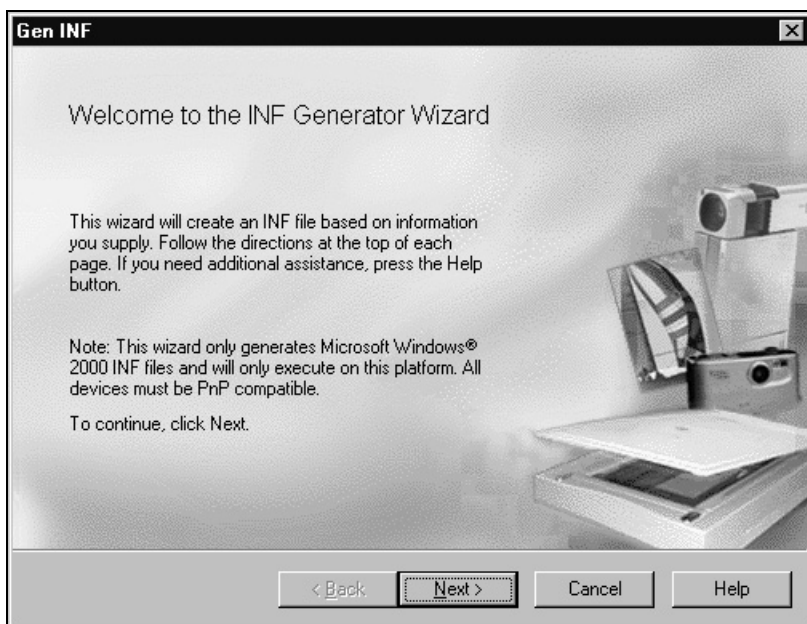


Рис. 10.5. Связи основных секций INF-файла

Итак, секция `Version` является обязательной в INF-файле. Секция `Manufacturer` содержит ссылку на секцию описания модели, которая в свою очередь содержит ссылки на секции для конкретных устройств с заданными серийными номерами. Секция описания устройства содержит ссылки на секции `CopyFiles`, `AddReg`, `DelReg` и `LogConfig`. Кроме того, любая секция может содержать ссылку на строки из секции `Strings` (рис. 10.5).

10.4.11. Создание и тестирование INF-файлов

В комплект Windows DDK входит довольно удобная утилита для создания INF-файлов, которая называется `GetInf` (рис. 10.6). Найти ее можно в каталоге `%DDK%\tools\geninf\x86\` в Windows XP DDK или `%DDK%\tools\` в Windows 2000 DDK.

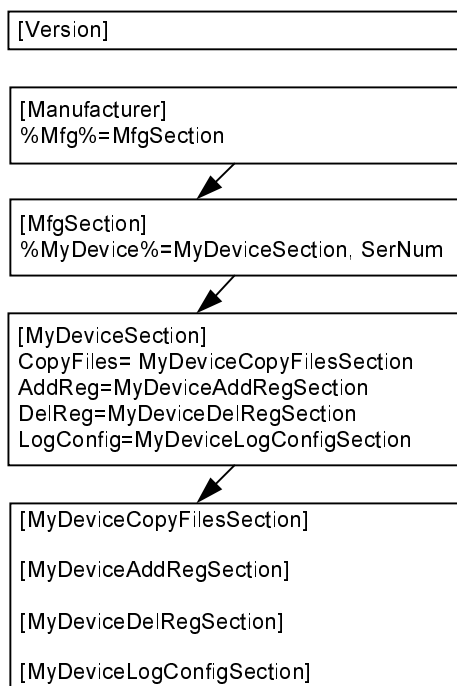


Рис. 10.6. Утилита генерации INF-файлов (Windows XP DDK)

Последовательно отвечая на вопросы утилиты (рис. 10.7), довольно легко получить вполне приемлемый вариант INF-файла.

Для проверки правильности написания INF-файла в Windows DDK предусмотрена утилита `ChkInf`. Для ее работы требуется установка интерпретатора

ActivePerl, загрузить который можно с сайта **www.activestate.com**. Утилита выдает результат в формате HTML.

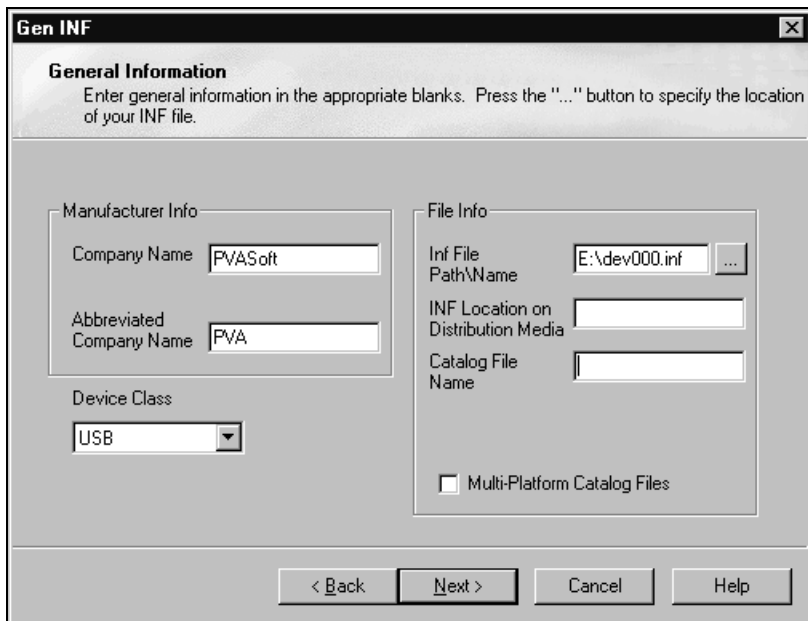


Рис. 10.7. Одно из диалоговых окон помощника создания INF-файла

10.4.12. Установка устройств с помощью INF-файла

В *разд. 1.7* мы уже рассматривали процедуру установки USB-устройств. В этом разделе мы опишем общие этапы этого процесса. Итак, обнаружив подключение нового устройства, диспетчер PnP выполняет следующие шаги:

1. PnP-менеджер режима ядра уведомляет PnP-менеджер пользовательского режима об обнаружении нового устройства со специфическими идентификаторами PnP (код производителя, модель, версия и т. д.).
2. PnP-менеджер пользовательского режима составляет список возможно подходящих драйверов, проверяя, в частности, системный каталог с доступными INF-файлами.
3. Если подходящий INF-файл не обнаружен, система откладывает все последующие действия до момента, пока в систему войдет пользователь с достаточным уровнем привилегий. Этому пользователю предлагается

диалоговое окно Мастера Установки (Add Hardware Wizard). Пользователь должен указать месторасположение подходящих INF-файлов.

4. При обнаружении подходящего INF-файла производится его обработка: выполняется копирование указанных файлов драйвера, модификация реестра и т. д.
5. На основе директив INF-файла PnP-менеджер режима ядра загружает все фильтр-драйверы нижнего уровня, затем функциональный драйвер и, наконец, верхние фильтр-драйверы, предназначенные для обслуживания нового устройства. Драйверу, находящемуся на вершине стека, направляются PnP-запросы (IRP-пакеты с кодом IRP_MJ_PNP), включая IRP_MN_START_DEVICE.

10.5. Ветки реестра для USB

Ветка реестра (рис. 10.8)

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\PCI

содержит имена USB-контроллеров, например,

VEN_1002&DEV_4742&SUBSYS_00000000&REV_5C

Каждая из этих веток имен содержит ветку, имя которой похоже на такое:

3&225b1d41&0&0008

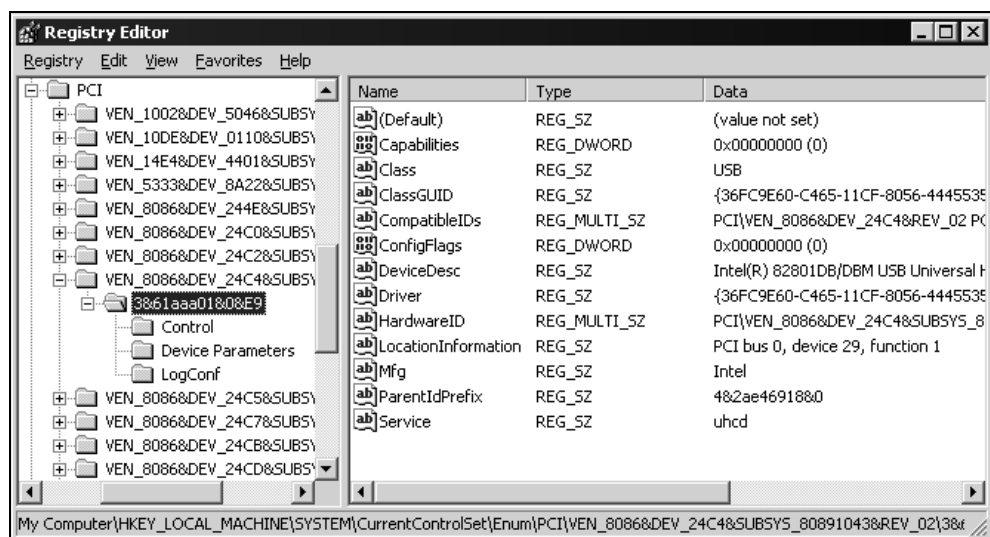


Рис. 10.8. Ветка реестра USB-контроллеров

Наиболее интересны следующие ключи этой ветки:

- ❑ DeviceDesc — описание контроллера;
- ❑ HardwareID — полное аппаратное имя;
- ❑ Mfg — имя производителя.

Ветка реестра

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB

содержит ключи, описывающие USB-устройства, когда-либо присутствующие в системе. Отметим, что к этим устройствам относятся также корневые хабы (рис. 10.9), но не относятся устройства, имеющие другой идентификатор класса (например, для класса USBSTOR создается другая ветка реестра).

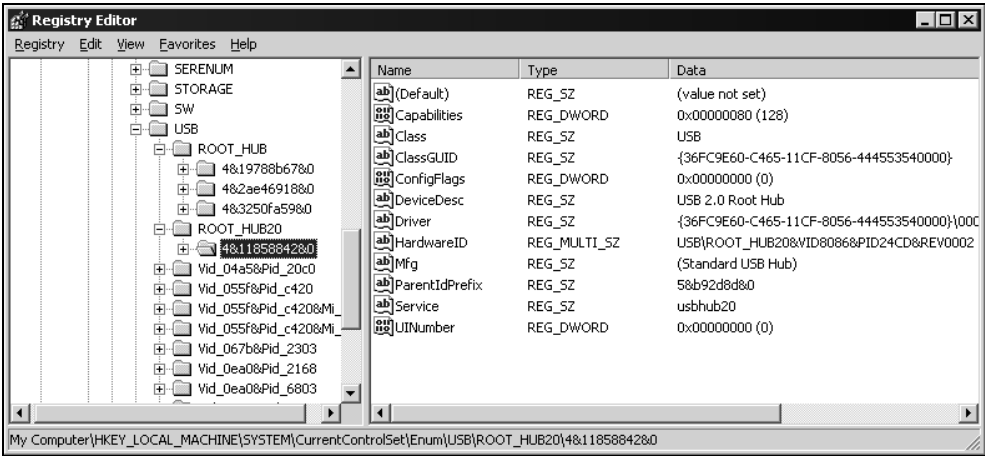
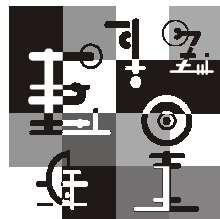


Рис. 10.9. Ветка реестра корневых хабов

Ветка Device Parameters содержит ключ SymbolicName, значение которого равно символьному имени (см. разд. 9.2) соответствующего устройства.

Глава 11



Функции BIOS

Перепрошьем Ваш BIOS так,
что родная мать не узнает!

Объявление

Одним из аргументов в пользу использования COM-интерфейсов является поддержка COM на уровне BIOS. Использование функций BIOS может потребоваться при программировании в DOS, например, при программировании промышленного компьютера (контроллера). Такие компьютеры имеют встроенный клон DOS, практически полностью совместимый с обычным DOS. В этой главе мы опишем функции BIOS для работы с USB.

11.1. Сервис BIOS 1AH

Для работы с PCI-шиной (и, соответственно, с шиной USB) используются следующие функции В1Н прерывания 1AH:

- ❑ B101H — определение наличия PCI BIOS;
- ❑ B102H — поиск PCI-устройства по идентификаторам устройства и производителя;
- ❑ B103H — поиск PCI-устройства по коду класса;
- ❑ B108H — чтение регистра конфигурации (байт, Byte);
- ❑ B109H — чтение регистра конфигурации (слово, Word);
- ❑ B10AH — чтение регистра конфигурации (двойное слово, DWord);
- ❑ B10BH — запись регистра конфигурации (байт, Byte);
- ❑ B10CH — запись регистра конфигурации (слово, Word);
- ❑ B10DH — запись регистра конфигурации (двойное слово, DWord).

Основное назначение этих функций — работа с конфигурационным пространством, общий вид которого показан на рис. 11.1. Наиболее интересными полями являются поля, позволяющие определить тип устройства:

- ❑ vendor ID (Word, смещение 0) — код фирмы-производителя устройства (см. приложение 3);

- ❑ Device ID (Word, смещение 2) — код устройства;
- ❑ Class Code (DWord, смещение 8) — код класса устройства (табл. 11.1).

Device ID		Vendor ID		00H
Status		Code		04H
Class Code			Revision ID	08H
BITS	Header Type	Latency Timer	Cache Line Size	0CH
Определяется типом устройства				
Определяется пользователем				

Рис. 11.1. Конфигурационное пространство устройства PCI

Для вызова конкретной функции ее номер задается в регистре AX. Для операций чтения и записи регистров номер шины (контроллера) задается в регистре BH. В зависимости от функции могут указываться входные параметры в регистрах BH, AL, CH, CL и др. При успешном выполнении функции флаг переноса CF сбрасывается в 0, в случае ошибки — устанавливается в 1. После выполнения функций регистры EAX, EBX, ECX, EDX и флаги могут измениться.

Следует отметить, что выполнение описанных функций сервиса 1AH может потребовать до 1 Кбайт стека.

11.1.1. Функция B101H — определение наличия PCI BIOS

Функция B101H позволяет проверить наличие поддержки PCI-функций. Входных параметров у этой функции нет, необходимо только занести в регистр AX номер функции.

На выходе заполняются следующие регистры:

- ❑ AH = 00H, если существует поддержка PCI;
- ❑ EDX = 20494350H (символы " PCI" в обратном порядке, в старшем байте записан пробел);
- ❑ EDI = адрес точки входа в защищенном режиме;
- ❑ CS = дескриптор защищенного режима (ring 0) для доступа к полному адресному пространству;
- ❑ AL = аппаратные характеристики PCI;

- ❑ BH = старшая часть номера версии PCI в формате BCD;
- ❑ BL = младшая часть номера версии PCI в формате BCD;
- ❑ CL = номер последней PCI-шины в системе (нумерация начинается с нуля);
- ❑ CF — результат выполнения:
 - 0 — функция выполнена успешно;
 - 1 — ошибка.

Присутствие PCI BIOS соответствует трем условиям: функция B101H выполнялась успешно, в регистре AH записан 0, в регистре EDX записана правильная сигнатура.

11.1.2. Функция B102H — поиск PCI-устройства по идентификаторам устройства и производителя

Для поиска PCI-устройства заданного типа используется функция B102H.

Входными регистрами этой функции являются:

- ❑ AX = B102H, номер функции;
- ❑ CX — идентификатор устройства (0—65535);
- ❑ DX — идентификатор изготовителя (0—65534);
- ❑ SI — порядковый номер устройства заданного типа (начиная с 0).

После выполнения будут заполнены следующие регистры:

- ❑ BH — номер шины, к которой подключено устройство (0—255);
- ❑ BL — номер устройства и функции:
 - [7:3] номер устройства;
 - [2:0] номер функции;
- ❑ AH — код возврата:
 - 00H — выполнение успешно;
 - 83H — ошибочное значение DX;
 - 86H — устройство не найдено;
- ❑ CF — результат выполнения:
 - 0 — функция выполнена успешно;
 - 1 — ошибка.

Для поиска всех устройств выбранного типа в регистр SI заносится 0 и производится последовательный вызов функции поиска. При каждом вызове значение SI инкрементируется. Выполнение производится до получения кода ошибки 86H.

11.1.3. Функция B103H — поиск PCI-устройства по коду класса

Для поиска PCI-устройства заданного класса используется функция B103H.

Входными регистрами этой функции являются:

- AX = B103H, номер функции;
- ECX — код класса в битах [23:0];
- SI — порядковый номер устройства заданного типа (начиная с 0).

После выполнения будут заполнены следующие регистры:

- BH — номер шины, к которой подключено устройство (0—255);
- BL — номер устройства и функции:
 - [7:3] номер устройства;
 - [2:0] номер функции;
- AH — код возврата:
 - 00H — выполнение успешно;
 - 86H — устройство не найдено;
- CF — результат выполнения:
 - 0 — функция выполнена успешно;
 - 1 — ошибка.

Для поиска всех устройств выбранного типа в регистр SI заносится 0 и производится последовательный вызов функции поиска. При каждом вызове значение SI инкрементируется. Выполнение производится до получения кода ошибки 86H.

Код класса содержится в 24 младших битах регистра ECX:

- [23:16] базовый класс;
- [15:8] подкласс;
- [7:0] интерфейс.

Значения некоторых кодов приведены в табл. 11.1.

Таблица 11.1. Некоторые коды классов PCI-устройств

Базовый класс	Подкласс	Интерфейс	Описание
01H	00H	00H	SCSI-контроллер
01H	02H	00H	Контроллер дискового
02H	00H	00H	Контроллер Ethernet
06H	00H	00H	Мост хоста
80H	00H	00H	Контроллер прерываний 8259
0CH	03H	00H	Устройство USB спецификации UHC
0CH	03H	10H	Устройство USB спецификации OHC
0CH	03H	80H	Устройство USB без определенного программного интерфейса
0CH	03H	FEH	Устройство USB (не хост-контроллер)

11.1.4. Функция B108H — чтение регистра конфигурации (Byte)

Для чтения байтового конфигурационного регистра заданного устройства используется функция B108H.

Входными регистрами этой функции являются:

- AX = B108H, номер функции;
- BH — номер шины, к которой подключено устройство (0—255);
- BL — номер устройства и функции:
 - [7:3] номер устройства;
 - [2:0] номер функции;
- DI — порядковый номер регистра (0—255).

После выполнения будут заполнены следующие регистры:

- CL — считанный байт;
- AH — код возврата:
 - 00H — выполнение успешно;
 - 87H — ошибочный номер регистра;
- CF — результат выполнения:
 - 0 — функция выполнена успешно;
 - 1 — ошибка.

11.1.5. Функция B109H — чтение регистра конфигурации (Word)

Для чтения двухбайтового конфигурационного регистра заданного устройства используется функция B109H.

Входными регистрами этой функции являются:

- ☐ AX = B109H, номер функции;
- ☐ BH — номер шины, к которой подключено устройство (0—255);
- ☐ BL — номер устройства и функции:
 - [7:3] номер устройства;
 - [2:0] номер функции;
- ☐ DI — смещение регистра в конфигурационном пространстве (0, 2, 4, ..., 254).

После выполнения будут заполнены следующие регистры:

- ☐ CX — считанное слово;
- ☐ AH — код возврата:
 - 00H — выполнение успешно;
 - 87H — ошибочный номер регистра;
- ☐ CF — результат выполнения:
 - 0 — функция выполнена успешно;
 - 1 — ошибка.

11.1.6. Функция B10AH — чтение регистра конфигурации (DWord)

Для чтения двойного слова из конфигурационного пространства заданного устройства используется функция B10AH.

Входными регистрами этой функции являются:

- ☐ AX = B10AH, номер функции;
- ☐ BH — номер шины, к которой подключено устройство (0—255);
- ☐ BL — номер устройства и функции:
 - [7:3] номер устройства;
 - [2:0] номер функции;
- ☐ DI — смещение регистра в конфигурационном пространстве (0, 4, 8, ..., 252).

После выполнения будут заполнены следующие регистры:

- ❑ ECX — считанное слово;
- ❑ AH — код возврата:
 - 00H — выполнение успешно;
 - 87H — ошибочный номер регистра;
- ❑ CF — результат выполнения:
 - 0 — функция выполнена успешно;
 - 1 — ошибка.

11.1.7. Функция B10BH — запись регистра конфигурации (Byte)

Для записи байтового конфигурационного регистра заданного устройства используется функция B10BH.

Входными регистрами этой функции являются:

- ❑ AX = B10BH, номер функции;
- ❑ BH — номер шины, к которой подключено устройство (0—255);
- ❑ BL — номер устройства и функции:
 - [7:3] номер устройства;
 - [2:0] номер функции;
- ❑ DI — порядковый номер регистра (0—255);
- ❑ CL — записываемый байт.

После выполнения будут заполнены следующие регистры:

- ❑ AH — код возврата:
 - 00H — выполнение успешно;
 - 87H — ошибочный номер регистра;
- ❑ CF — результат выполнения:
 - 0 — функция выполнена успешно;
 - 1 — ошибка.

11.1.8. Функция B10CH — запись регистра конфигурации (Word)

Для записи двухбайтового конфигурационного регистра заданного устройства используется функция B10CH.

Входными регистрами этой функции являются:

- ❑ AX = B10BH, номер функции;
- ❑ BH — номер шины, к которой подключено устройство (0—255);
- ❑ BL — номер устройства и функции:
 - [7:3] номер устройства;
 - [2:0] номер функции;
- ❑ DI — смещение регистра в конфигурационном пространстве (0, 4, ..., 254);
- ❑ CX — записываемое слово.

После выполнения будут заполнены следующие регистры:

- ❑ AH — код возврата:
 - 00H — выполнение успешно;
 - 87H — ошибочный номер регистра;
- ❑ CF — результат выполнения:
 - 0 — функция выполнена успешно;
 - 1 — ошибка.

11.1.9. Функция B10DH — запись регистра конфигурации (DWord)

Для записи двойного слова в конфигурационное пространство заданного устройства используется функция B10DH.

Входными регистрами этой функции являются:

- ❑ AX = B10BH, номер функции;
- ❑ BH — номер шины, к которой подключено устройство (0—255);
- ❑ BL — номер устройства и функции:
 - [7:3] номер устройства;
 - [2:0] номер функции;
- ❑ DI — смещение регистра в конфигурационном пространстве (0, 8, ..., 252);
- ❑ ECX — записываемое двойное слово.

После выполнения будут заполнены следующие регистры:

- ❑ AH — код возврата:
 - 00H — выполнение успешно;
 - 87H — ошибочный номер регистра;

- CF — результат выполнения:
- 0 — функция выполнена успешно;
 - 1 — ошибка.

11.2. Пример использования

Листинг 11.1 показывает модуль, содержащий функции сервиса PCI BIOS. Модуль написан на языке Pascal. Разумеется, все программы, его использующие, будут работать только в DOS или Windows 9x (Windows 2000 и Windows XP блокируют вызов прерывания 1AH даже из DOS-приложений).

Листинг 11.1. Сервис PCI BIOS (DOS)

```
{ Использование функций B1H сервиса 1AH}
unit PCI;

INTERFACE

{ Определение поддержки PCI BIOS}
Function DetectPCIBios : Boolean;

{ Поиск устройства по идентификаторам
  { производителя и устройства      }
Function DetectPCIDevice(
    DeviceID : Word;
    VendorID : Word;
    Index    : Word;
    var BusNumber      : Byte;
    var DevFuncNumber : Byte
): Byte;

{ Поиск устройства по коду класса }
Function DetectPCIDeviceI(
    ClassCode : Longint;
    Index     : Word;
    var BusNumber      : Byte;
    var DevFuncNumber : Byte
): Byte;
```

```
{ Чтение регистра конфигурации (Byte)}
```

```
Function ReadPCIRegisterByte(  
    RegisterNumber : Word;  
    BusNumber      : Byte;  
    DevFuncNumber  : Byte;  
    var Result     : Byte  
): Byte;
```

```
{ Чтение регистра конфигурации (Word)}
```

```
Function ReadPCIRegisterWord(  
    RegisterNumber : Word;  
    BusNumber      : Byte;  
    DevFuncNumber  : Byte;  
    var Result     : Word  
): Word;
```

```
{ Чтение регистра конфигурации (DWord)}
```

```
Function ReadPCIRegisterDWord(  
    RegisterNumber : Word;  
    BusNumber      : Byte;  
    DevFuncNumber  : Byte;  
    var Result     : Longint  
): Longint;
```

```
{ Запись регистра конфигурации (Byte)}
```

```
Function WritePCIRegisterByte(  
    RegisterNumber : Word;  
    BusNumber      : Byte;  
    DevFuncNumber  : Byte;  
    Input          : Byte  
): Byte;
```

```
{ Запись регистра конфигурации (Word)}
```

```
Function WritePCIRegisterWord(  
    RegisterNumber : Word;  
    BusNumber      : Byte;  
    DevFuncNumber  : Byte;  
    Input          : Word  
): Word;
```



```
{ Запись регистра конфигурации (Dword)}
```

```
Function WritePCIRegisterDWord(
    RegisterNumber : Word;
    BusNumber      : Byte;
    DevFuncNumber  : Byte;
    Input          : Longint
): Byte;
```

IMPLEMENTATION

```
Function DetectPCIBios : Boolean; assembler;
```

```
Asm
```

```
    Mov ax, 0b101h
```

```
    Int 1ah
```

```
    Jc @nopicbios
```

```
    Mov ax, 1
```

```
    Ret
```

```
@nopicbios:
```

```
    Xor ax, ax
```

```
    Ret
```

```
End;
```

```
Function DetectPCIDevice(
```

```
    DeviceID : Word;
```

```
    VendorID : Word;
```

```
    Index    : Word;
```

```
    var BusNumber      : Byte;
```

```
    var DevFuncNumber : Byte
```

```
): Byte;
```

```
Var _Result, bn, fn : Byte;
```

```
Begin
```

```
    Asm
```

```
        mov ax, 0b102h
```

```
        mov cx, DeviceID
```

```
        mov dx, VendorID
```

```
        mov si, Index
```

```
        int 1ah
```

```
        mov _Result, ah
```

```

    mov bn, bh
    mov fn, bl
End;

BusNumber      := bn;
DevFuncNumber := fn;
DetectPCIdDevice := _Result;
End;

Function DetectPCIdDeviceI(
    ClassCode : Longint;
    Index      : Word;
    var BusNumber      : Byte;
    var DevFuncNumber : Byte
): Byte;
Var _Result, bn, fn : Byte;
Begin
    Asm
        mov ax, 0b103h
        {MOV ECX, ClassCode}
        db 66h; mov cx, word ptr ClassCode
        mov si, Index
        int 1ah
        mov _Result, ah
        mov bn, bh
        mov fn, bl
    End;

    BusNumber      := bn;
    DevFuncNumber := fn;
    DetectPCIdDeviceI := _Result;
End;

Function ReadPCIdRegisterByte(
    RegisterNumber : Word;
    BusNumber      : Byte;
    DevFuncNumber  : Byte;
    var Result      : Byte
): Byte;

```

```
Var _Result : Byte; Res : Byte;
Begin
  Asm
    mov AX, 0B108h
    mov BH, BusNumber
    mov BL, DevFuncNumber
    mov DI, RegisterNumber
    int 1Ah
    mov Res, cl
    mov _Result, ah
  End;
  Result:= Res;
  ReadPCIRegisterByte:= _Result;
End;
```

```
Function ReadPCIRegisterWord(
  RegisterNumber : Word;
  BusNumber      : Byte;
  DevFuncNumber  : Byte;
  var Result     : Word
): Byte;
```

```
Var _Result : Byte; Res : Word;
Begin
  Asm
    mov AX, 0B109h
    mov BH, BusNumber
    mov BL, DevFuncNumber
    mov DI, RegisterNumber
    int 1Ah
    mov Res, cx
    mov _Result, ah
  End;
  Result:= Res;
  ReadPCIRegisterWord:= _Result;
End;
```

```
Function ReadPCIRegisterDWord(
  RegisterNumber : Word;
  BusNumber      : Byte;
```

```
        DevFuncNumber : Byte;
        var Result      : Longint
    ): Byte;
Var _Result : Byte; Res : Longint;
Begin
    Asm
        mov AX, 0B10ah
        mov BH, BusNumber
        mov BL, DevFuncNumber
        mov DI, RegisterNumber
        int 1Ah
        { MOV RES, ECX }
        db 66h; mov word ptr Res, cx
        mov _Result, ah
    End;
    Result:= Res;
    ReadPCIRegisterDword:= _Result;
end;
```

```
Function WritePCIRegisterByte(
    RegisterNumber : Word;
    BusNumber      : Byte;
    DevFuncNumber  : Byte;
    Input          : Byte
): Byte;
Var _Result : Byte;
Begin
    Asm
        mov AX, 0B10bh
        mov BH, BusNumber
        mov BL, DevFuncNumber
        mov DI, RegisterNumber
        mov CL, Input
        int 1Ah
        mov _Result, ah
    End;
    WritePCIRegisterByte:= _Result;
end;
```

```
Function WritePCIRegisterWord(  
    RegisterNumber : Word;  
    BusNumber      : Byte;  
    DevFuncNumber  : Byte;  
    Input          : Word  
    ): Byte;  
Var _Result : Byte;  
Begin  
    Asm  
        mov AX, 0B10ch  
        mov BH, BusNumber  
        mov BL, DevFuncNumber  
        mov DI, RegisterNumber  
        mov CX, Input  
        int 1Ah  
        mov _Result, ah  
    End;  
    WritePCIRegisterWord:= _Result;  
end;  
  
Function WritePCIRegisterDWord(  
    RegisterNumber : Word;  
    BusNumber      : Byte;  
    DevFuncNumber  : Byte;  
    Input          : Longint  
    ): Byte;  
Var _Result : Byte;  
Begin  
    Asm  
        mov AX, 0B10dh  
        mov BH, BusNumber  
        mov BL, DevFuncNumber  
        mov DI, RegisterNumber  
        {MOV ECX, Input}  
        db 66h; mov CX, word ptr Input  
        int 1Ah  
        mov _Result, ah  
    End;
```

```
WritePCIRegisterDWord:= _Result;  
end;  
  
END.
```

Листинг 11.2 показывает пример использования модуля PCI для поиска USB-контроллеров по идентификаторам производителя и устройства, а листинг 11.3 — для поиска USB-контроллеров по коду класса.

Листинг 11.2. Поиск USB-контроллеров по идентификаторам устройства и производителя

```
Uses PCI, StrFunc;  
  
Procedure DetectUSBController;  
var  
    DeviceID, VendorID : Word;  
    Description : String;  
    IDIndex      : Integer;  
    DevIndex     : Word;  
    BusNumber, DevFuncNumber : Byte;  
    Found        : Boolean;  
  
begin  
    IDIndex:=0;  
    Repeat  
        Case IDIndex of  
            0: begin  
                DeviceId:=$7020; VendorId:=$8086;  
                Description:='Intel 82371SB USB controller';  
            end;  
            1: begin  
                DeviceId:=$7112; VendorId:=$8086;  
                Description:='Intel PIIX4 USB controller';  
            end;  
            2: begin  
                DeviceId:=$3104; VendorId:=$1106;  
                Description:='VIA AMD-645 EHCI USB v2';  
            end;  
        end;  
    until Found;
```

```

3: begin
    DeviceId:=$3038; VendorId:=$1106;
    Description:='VIA AMD-645 UHC USB';
end;
4: begin
    DeviceId:=$A0F8; VendorID:=$1045;
    Description:='Opti 82C750 (Vendetta) USB controller';
end;
5: begin
    DeviceId:=$C861; VendorID:=$1045;
    Description:='Opti 82C861/871 USB controller';
end;
Else IDIndex:= -1;
End;

If IDIndex = -1 then Break;

DevIndex:= 0;
Repeat
    Found:= DetectPCIDevice(DeviceId, VendorId,
                           DevIndex, BusNumber, DevFuncNumber) = 0;
If Found then begin
    Writeln(DevIndex, ': ', Description);
    Writeln(' DeviceID      : ', Long2Hex(DeviceId));
    Writeln(' VendorID      : ', Long2Hex(VendorId));
    Writeln(' BusNumber      : ', BusNumber);
    Writeln(' DevFuncNumber : ', DevFuncNumber);
End;
    Inc(DevIndex);
Until not(Found);

    Inc(IDIndex);
Until (False);
end;

Begin
If DetectPCIBios then begin
    WriteLn('PICBIOS поддерживается');
End else begin

```

```

WriteLn('PICBIOS не поддерживается');
Exit;
End;

DetectUSBController;
End.

```

Листинг 11.3. Поиск USB-контроллеров по коду класса

```

Uses PCI, StrFunc;

Procedure DetectUSBDevice(ClassCode : Longint);
Var
  Index      : Word;
  BusNumber, DevFuncNumber : Byte;
  Found      : Byte;
  DeviceID, VendorID : Word;
Begin
  Index:= 0;
  While (True) do begin
    Found:= DetectPCIDeviceI(ClassCode, Index,
                           BusNumber, DevFuncNumber);
    If Found = 0 then begin
      WriteLn('-----');
      Writeln(' Index      : ', Index      );
      Writeln(' BusNumber   : ', BusNumber );
      Writeln(' DevFuncNumber : ', DevFuncNumber);

      If ReadPCIRRegisterWord(0,
        BusNumber, DevFuncNumber, VendorID) = 0 then begin
        Writeln(' VendorID   : ', Long2Hex(VendorID));
      End;

      If ReadPCIRRegisterWord(2,
        BusNumber, DevFuncNumber, DeviceID) = 0 then begin
        Writeln(' DeviceID   : ', Long2Hex(DeviceID));
      End;
    end;
  end;
End;

```



```
End else begin
    Break;
End;
Inc(Index);
End;
End;

Begin
If DetectPCIBios then begin
    WriteLn('PICBIOS поддерживается');
End else begin
    WriteLn('PICBIOS не поддерживается');
    Exit;
End;

WriteLn(' Контроллеры (UHC USB)');
DetectUSBDevice($0C0300);
WriteLn(' Контроллеры (OHC USB)');
DetectUSBDevice($0C0310);
WriteLn(' Контроллеры (EHCI USB v2)');
DetectUSBDevice($0C0320);
End.
```

Листинги 11.2 и 11.3 производят поиск контроллеров, а для поиска устройств, подключенных к контроллерам, необходимо проверить состояние портов контроллера, обратившись к регистрам "состояние порта 0" и "состояние порта 1" (см. табл. 18.2). Пример поиска устройств на шине USB приведен в листинге 11.4. Следует помнить, что данный пример использует карту регистров UHC и, например, для контроллеров EHCI работать не будет. Для удобства использования мы сделали отдельный модуль и класс для работы с USB-контроллерами (листинг 11.5).

Листинг 11.4. Поиск устройств на USB-шине

```
Uses PCI, StrFunc, Crt, Util, ContList;

Var i, p : Byte;
    USBControllerList : PUSBControllerList;

Begin
    ClrScr;
```

```
If DetectPCIBios then begin
  WriteLn('PICBIOS поддерживается');
End else begin
  WriteLn('PICBIOS не поддерживается');
  Exit;
End;

USBControllerList:= New(PUSBControllerList, Init);
USBControllerList^.MakeList;

WriteLn('Найдено ', USBControllerList^.GetCount, ' USB-контроллеров');

{ Поиск устройств по контроллерам }
For i:= 1 to USBControllerList^.GetCount do begin
  With USBControllerList^.GetAt(i)^ do begin
    { Если контроллер не сконфигурирован - не обрабатываем }
    If not(IsConfigured) then Continue;

    ResetController (True ); { сброс }
    ResetController (False); { снять сигнал сброса }
    EnableController(True ); { активизировать контроллер }

    For p:= 1 to 2 do begin
      If PortConnected(p) then begin
        WriteLn('Найдено устройство: контр=', i, ' порт ', p);
        If IsLowSpeedDevice(p) then
          WriteLn(' - найденное устройство LowSpeed');
        End;
      End;

      EnableController(False); { остановить контроллер }
    End;
  End;

  Dispose(USBControllerList, Done);
End.
```

Листинг 11.5. Класс `TUSBControllerList`

```
Unit ContList;
```

```
INTERFACE
```

```
Uses Crt, PCI;
```

```
{ USB-контроллер }
```

```
Type
```

```
  PUSBController = ^TUSBController;
```

```
  TUSBController = object
```

```
    Public
```

```
      Index              : Word;
```

```
      BusNumber          : Byte;
```

```
      DevFuncNumber      : Byte;
```

```
      BaseAddress        : LongInt;
```

```
      IRQNumber          : Byte;
```

```
    Public
```

```
      Procedure ResetController (Reset : Boolean);
```

```
      Procedure EnableController(Enable : Boolean);
```

```
      Function  IsConfigured : Boolean;
```

```
      Procedure CommandStop;
```

```
      Procedure CommandRun;
```

```
      Procedure SetFrameNumber(Number : Word);
```

```
      Procedure SetFrameListAddress(FrameListAdr : LongInt);
```

```
    Public
```

```
      Function  GetPortBaseAddr(PortIndex : Byte) : Word;
```

```
      Function  GetPortState(PortIndex : Byte) : Word;
```

```
      Function  PortConnected(PortIndex : Byte): Boolean;
```

```
      Function  IsLowSpeedDevice(PortIndex : Byte): Boolean;
```

```
      Procedure EnablePort(PortIndex : Byte; Enable : Boolean);
```

```
End;
```

```
{ Максимальное число контроллеров }
```

```
Const
```

```
  MaxControllerIndex = 6;
```

```

{ Список контроллеров }
Type
  PUSBControllerList = ^TUSBControllerList;
  TUSBControllerList = object
    Private
      USBContrArr : Array [1..MaxControllerIndex] of PUSBController;
      USBContrCnt : Byte;
    Private
      Procedure DetectUSBDevice(ClassCode : Longint);
    Public
      Constructor Init;
      Destructor Done; virtual;
    Public
      Procedure MakeList;
      Function GetCount : Integer;
      Function GetAt(Index : Integer) : PUSBController;
  End;

```

IMPLEMENTATION

```

{=====}
{                               TUSBController                               }
{=====}
Procedure TUSBController.ResetController(Reset : Boolean);
Var Signal : Word;
Begin
  If Reset then Signal:= 4 else Signal:= 0;
  PortW[BaseAddress]:= Signal;
  Delay(10);
End;

Procedure TUSBController.EnableController(Enable : Boolean);
Var Signal : Word;
Begin
  If Enable then Signal:= 1 else Signal:= 0;
  PortW[BaseAddress]:= Signal;
End;

```

```
Function TUSBController.IsConfigured : Boolean;
Begin
  IsConfigured:= (BaseAddress <> -1);
End;

Function TUSBController.GetPortBaseAddr(PortIndex : Byte) : Word;
Begin
  If PortIndex = 1
  Then GetPortBaseAddr:= BaseAddress + $10
  Else GetPortBaseAddr:= BaseAddress + $12;
End;

Function TUSBController.GetPortState(PortIndex : Byte) : Word;
Begin
  GetPortState:= PortW[GetPortBaseAddr(PortIndex)];
End;

Function TUSBController.PortConnected(PortIndex : Byte): Boolean;
Begin
  PortConnected:= (GetPortState(PortIndex) and $0001) <> 0;
End;

Function TUSBController.IsLowSpeedDevice(PortIndex : Byte): Boolean;
Begin
  IsLowSpeedDevice:= (GetPortState(PortIndex) and $0100) <> 0;
End;

Procedure TUSBController.EnablePort(PortIndex : Byte; Enable : Boolean);
Var PortAddr : Word;
Begin
  PortAddr:= GetPortBaseAddr(PortIndex);
  If Enable
  Then PortW[PortAddr]:= PortW[PortAddr] or $04
  Else PortW[PortAddr]:= PortW[PortAddr] and $FB;
End;

procedure TUSBController.CommandRun;
begin
```

```

PortW[BaseAddress]:= PortW[BaseAddress] or 1;
end;

procedure TUSBController.CommandStop;
begin
  PortW[BaseAddress]:= PortW[BaseAddress] and $FE;
end;

Procedure TUSBController.SetFrameNumber(Number : Word);
Begin
  PortW[BaseAddress+6]:= Number;
End;

Procedure TUSBController.SetFrameListAddress(FrameListAdr : LongInt);
Begin
  Asm
    mov dx, BaseAddress
    add dx, 08h
    db 66h; mov ax, word ptr FrameListAdr { mov eax, ... }
    db 66h; out dx,ax                     { out dx,eax }
  End;
End;

{=====}
{          TUSBControllerList          }
{=====}

Constructor TUSBControllerList.Init;
Var Index : Integer;
Begin
  USBContrCnt:= 0;
  For Index:= 1 to MaxControllerIndex do
    USBContrArr[Index]:= nil;
  End;

Destructor TUSBControllerList.Done;
Var Index : Integer;
Begin
  For Index:= 1 to MaxControllerIndex do
    If USBContrArr[Index] <> nil then

```

```

    Dispose(USBContrArr[Index]);
End;

Procedure TUSBControllerList.MakeList;
Begin
    DetectUSBDevice($0C0300); {UHC USB}
    {DetectUSBDevice($0C0310); {OHC USB}
    {== EHCI имеет другую карту адресов!}
    {DetectUSBDevice($0C0320); {EHCI USB}
End;

Function TUSBControllerList.GetCount : Integer;
Begin
    GetCount:= USBContrCnt;
End;

{ Поиск USB-контроллеров }
Procedure TUSBControllerList.DetectUSBDevice(ClassCode : Longint);
Var
    Index      : Word;
    BusNumber, DevFuncNumber, IRQNumber : Byte;
    BaseAddress : LongInt;
    Found : Byte;
    DeviceID, VendorID : Word;
Begin
    Index:= 0;
    While (True) do begin
        Found:= DetectPCIddeviceI(ClassCode, Index,
            BusNumber, DevFuncNumber);
        { Если найден очередной контроллер}
        { Добавить его в список контроллеров}
        If Found = PCI_SUCCESS then begin
            { Добавить новую запись в список }
            Inc(USBContrCnt);
            USBContrArr[USBContrCnt]:= New(PUSBController);

            { Заполняем свойства записи }
            USBContrArr[USBContrCnt]^..Index      := Index;

```

```

USBContrArr[USBContrCnt]^BusNumber      := BusNumber;
USBContrArr[USBContrCnt]^DevFuncNumber:= DevFuncNumber;

{20H - смещение двойного слова,}
{хранящего базовый адрес контроллера}
If ReadPCIRegisterDWord($20,
    BusNumber, DevFuncNumber, BaseAddress) = PCI_SUCCESS then begin
    {обнулить 5 младших бит}
    BaseAddress:= BaseAddress and $0FFE0;
    {сохранить базовый адрес в таблице}
    USBContrArr[USBContrCnt]^BaseAddress:= BaseAddress;
End else begin
    { ошибка чтения базового адреса }
    USBContrArr[USBContrCnt]^BaseAddress:= -1;
End;

{3CH - смещение байта, хранящего }
{номер прерывания IRQ}
If ReadPCIRegisterByte($3C,
    BusNumber, DevFuncNumber, IRQNumber) = PCI_SUCCESS then begin
    USBContrArr[USBContrCnt]^IRQNumber:= IRQNumber;
End;

If USBContrCnt > MaxControllerIndex then Exit;
End else begin
    Break;
End;
Inc(Index);
End;
End;

Function TUSBControllerList.GetAt(Index : Integer) : PUSBController;
Begin
    GetAt:= USBContrArr[Index];
End;

END.
```


В этом листинге мы несколько поменяли процедуру DetectUSBDevice: найденные в системе контроллеры сохраняются в специальном массиве USBContrArr класса TUSBControllerList (мы обрабатываем первые 6 контроллеров). После обнаружения контроллеров производится проверка состояния портов для каждого из них. Установленный в единицу бит 0 порта состояния обозначает, что к порту подключено устройство, а установленный в единицу бит 8 показывает, что это низкоскоростное устройство (см. разд. 18.1.7).

Для чтения данных с найденного устройства нам придется выполнить всю ту работу, которую в Windows берет на себя USB-драйвер: сформировать список дескрипторов, передать его устройству и дождаться ответа. К сожалению, эта задача является довольно сложной. Во-первых, для обращения к устройству (точнее будет сказать — порту, к которому подключено устройство) используются регистры хост-контроллера, поэтому следует учитывать конкретную спецификацию контроллера (см. разд. 1.6). Во-вторых, при передаче блока данных необходимо указать контроллеру линейный адрес памяти, что при использовании драйверов-расширителей, не так очевидно, как кажется. Самый простой способ решения второй проблемы — перевод процессора в режим линейной адресации, как это сделано в [6], но такое решение годится, пожалуй, лишь для демонстрационных целей. Объем книги не позволяет привести полный пример решения для всех случаев и всех типов контроллеров. Мы приведем общую схему решения для контроллера UCH. Регистры этого контроллера описываются в главе 18.

Листинг 11.6 показывает основные типы, используемые для работы.

Листинг 11.6. Типы для работы с контроллером UCH

```
Type { Список кадров }
  PFrameList = ^TFrameList;
  TFrameList = Array [1..1024] of LongInt;

Type { Запрос }
  TUSBRequest = packed record
    bmRequestType: Byte;
    bRequest      : Byte;
    wValue        : Word;
    wIndex        : Word;
    wLength       : Word;
  End;

Type { Заголовок очереди и дескриптор передачи }
  TTD_Array = Array [1..8*64] of LongInt;
```

```

TQueueHeadPointerArray = Packed record
  QH_Descriptor : Array [1..8] of LongInt;
  TD_Array      : TTD_Array;
End;
PQueueHead = ^TQueueHead;
TQueueHead = TQueueHeadPointerArray;

```

Для поиска устройства мы будем использовать класс `TUSBControllerList`. Для простоты мы будем работать с первым найденным устройством, однако очень просто изменить этот код для поиска нужного устройства (листинг 11.7).

Листинг 11.7. Поиск первого USB-устройства

```

Procedure GetFirstDevice;
Begin
  USB_BaseAddr:= 0;
  For USB_HostIndex:= 1 to USBControllerList^.GetCount do
    With USBControllerList^.GetAt(USB_HostIndex)^ do
      For USB_PortNum:= 1 to 2 do
        If PortConnected(USB_PortNum) then begin
          USB_BaseAddr:= BaseAddress;
          USB_PortReg := GetPortBaseAddr(USB_PortNum);
          Exit;
        End;
      End;
    End;
  End;

```

Переменные `USB_HostIndex`, `USB_PortNum`, `USB_BaseAddr`, `USB_PortReg` будут хранить соответственно номер контроллера, номер порта, базовый адрес контроллера и адрес порта.

Формирование запроса производится с помощью типа `TUSBRequest`. В листинге 11.8 показано формирование запроса `SET_ADDRESS`.

Листинг 11.8. Формирование запроса SET_ADDRESS

```

Var
  SetAddressDesc : TUSBRequest;

With SetAddressDesc do begin
  bmRequestType:= 0;

```

```

bRequest      := 5;
wValue        := 1;
wIndex        := 0;
wLength       := 0;
End;

```

Для передачи запроса устройству нам придется выполнить следующие действия:

- ☐ получить линейный адрес запроса;
- ☐ сформировать заголовок очереди и получить его линейный адрес;
- ☐ сформировать список кадров и получить его линейный адрес;
- ☐ добавить в список кадров записи, соответствующие транзакциям нужного запроса (см. разд. 3.10.3);
- ☐ разрешить работу контроллера и порта.

Как мы уже говорили, основной проблемой является получение линейного адреса. В простейшем случае линейный адрес вычисляется, как показано в листинге 11.9, однако при использовании драйверов *Nimem* или *Emm386* такой код работать не будет.

Листинг 11.9. Вычисление линейного адреса

```

Function GetPtrBase(P : Pointer) : Longint;
Begin
  GetPtrBase:= Longint(Seg(P^)) shl 4 + Longint(Ofs(P^));
End;

```

Общая схема передачи запроса показана в листинге 11.10.

Листинг 11.10. Общая схема передачи запроса

```

{ Вычислить линейный адрес блока команды }
Addr_SetAddress:= GetLinearAddr(@SetAddressDesc);

{ Отводим память под заголовок очереди }
{ Используем GetMem или другой способ }
GetMem(QueueHead, 1024*10);
{ Заполняем заголовок очереди }
With QueueHead^ do begin
  QH_Descriptor[1]:= 3;

```

```

QH_Descriptor[2]:= 0;
QH_Descriptor[3]:= 0;
QH_Descriptor[4]:= 0;
QH_Descriptor[5]:= 0;
QH_Descriptor[6]:= 0;
QH_Descriptor[7]:= 0;
QH_Descriptor[8]:= 0;
{TD_Array : TTD_Array;}
End;

{ Вычисляем линейный адрес заголовков }
Addr_QH:= GetLinearAddr(@QueueHead^.QH_Descriptor);
Addr_TD_Array:= GetLinearAddr(@QueueHead^.TD_Array);

{ Базовый адрес памяти (храним для освобождения) }
GetMem(FrameListHandle, 8192);
FillChar(FrameListHandle^, 8192, #0);
{ Линейный адрес }
FrameListAddr:= GetLinearAddr(FrameListHandle);
{ Выравненный адрес }
FrameListBaseAddr:=
  LongInt(FrameListAddr + 4096) and $fffff000;
{ Указатель на выровненный адрес }
FrameList:= GetBasePtr(FrameListAddr);
{ Заполняем }
For i:= 1 to 1024 do begin
  FrameList^[i]:= Addr_QH or $2;
End;

{ Класс контроллеров для поиска устройства }
USBControllerList:= New(PUSBControllerList, Init);
USBControllerList^.MakeList;
WriteLn('Найдено контроллеров:', USBControllerList^.GetCount);

{ Ищем первое устройство }
GetFirstDevice;

{ Если не нашли - выход }
If USB_BaseAddr = 0 then begin
  WriteLn('Устройства не найдены');

```

```

    Dispose(USBControllerList, Done);
    Exit;
End;

WriteLn('Найдено устройство:');
WriteLn(' номер контр-ра:', USB_HostIndex);
WriteLn(' базовый адрес :', USB_BaseAddr );
WriteLn(' порт           :', USB_PortNum );
WriteLn(' адрес порта    :', USB_PortReg );

{ DD[Base+8], - базовый адрес списка кадров }
{ Заносим адрес списка кадров }
PortW[USB_BaseAddr+ 8]:= FrameListBaseAddr and $0000FFFF;
PortW[USB_BaseAddr+10]:= FrameListBaseAddr shr 16;

{ сброс номера кадра USB }
PortW[USB_BaseAddr+6]:= 0;

{ Включить хост-контроллер }
PortW[USB_BaseAddr]:= 1;

{ Разблокировать порт }
PortW[USB_PortReg]:= $E;

{ Определить и запомнить тип устройства}
If USBControl-
lerList^.GetAt(USB_HostIndex)^.IsLowSpeedDevice(USB_PortNum)
Then ShDevType:= $4000000
Else ShDevType:= $0000000;

{ После сброса устройство имеет нулевой номер}
ShFuncNum:= 0;

{ Выполнение транзакции для запроса SET_ADDRESS}
while not(KeyPressed) do begin
{ Подать команду "Set Address"}
i:= 0;
With QueueHead^ do begin
    TD_Array[i+1]:= Addr_TD_Array + 32;
    TD_Array[i+2]:= ShDevType or $00800000;

```

```
TD_Array[i+3]:= ShFuncNum or $00E0002D;
TD_Array[i+4]:= Addr_SetAddress;
TD_Array[i+5]:= 0;
TD_Array[i+6]:= 0;
TD_Array[i+7]:= 0;
TD_Array[i+8]:= 0;
End;

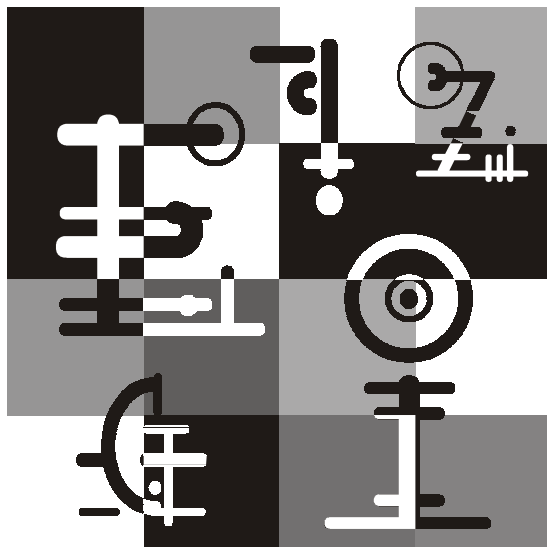
i:= 8;
With QueueHead^ do begin
  TD_Array[i+1]:= 1;
  TD_Array[i+2]:= ShDevType or $00800000;
  TD_Array[i+3]:= ShFuncNum or $FFE80069;
  TD_Array[i+4]:= 0;
  TD_Array[i+5]:= 0;
  TD_Array[i+6]:= 0;
  TD_Array[i+7]:= 0;
  TD_Array[i+8]:= 0;
End;

{Записать кадр в заголовок очереди }
QueueHead^.QH_Descriptor[2]:= Addr_TD_Array;

{Ждать выполнения транзакции }
  Delay(10000);
End;

{ Выключить хост-контроллер}
PortW[USB_BaseAddr]:= 0;

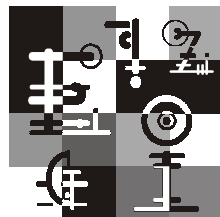
{ Освободить список контроллеров }
Dispose(USBControllerList, Done);
```



ЧАСТЬ IV

СОЗДАНИЕ USB-УСТРОЙСТВ

Глава 12



USB-периферия

Гадание на микросхемах, откусывая ножки, —
Любит? Не любит?

В этой главе мы дадим краткий обзор основных микросхем, используемых для организации USB-интерфейса. Микросхемы можно разделить на следующие группы:

- ☐ преобразователи интерфейса:
 - COM в USB;
 - LPT в USB;
 - конвертеры других интерфейсов;
- ☐ микроконтроллеры с USB-интерфейсом:
 - микроконтроллеры на основе ядра 8051;
 - другие микроконтроллеры;
- ☐ микросхемы хабов;
- ☐ микросхемы OTG.

Такое деление довольно условно, например, микроконтроллер AT43USB320A включает в себя хаб, а микросхема FT2232BM является программируемым преобразователем интерфейса.

Среди множества производителей микросхем можно выделить следующих:

- ☐ Atmel (www.atmel.com);
- ☐ Cypress (www.cypress.com);
- ☐ Cygnal¹ (www.silabs.com);
- ☐ Fairchild Semiconductor (www.fairchildsemi.com);
- ☐ FTDI (www.ftdichip.com);

¹ С декабря 2003 фирма Cygnal приобретена Silicon Laboratories.

- ☐ Intel (www.intel.com);
- ☐ Microchip (www.microchip.com);
- ☐ Motorola (e-www.motorola.com);
- ☐ Philips (www.semiconductors.philips.com);
- ☐ Texas Instruments (www.texasinstruments.com);
- ☐ Trans Dimension (www.transdimension.com).

Конечно, на сегодняшний день производится огромное число USB-микросхем. Подробный обзор потребовал бы книги в несколько раз большего объема, поэтому наиболее подробно мы будем рассматривать только микросхемы, доступные на российском рынке и не требующие дополнительных инструментов (специальных программаторов или ассемблеров).

Важно

При описании микросхем мы будем использовать часто употребляемые обозначения, такие как UART, SRAM, FLASH, АЦП и т. д. Объем книги не позволяет привести расшифровку этих обозначений, и мы надеемся, что читатель, готовящийся к созданию своего USB-устройства, обладает достаточными знаниями в этой области.

12.1. Микросхемы Atmel

Корпорация Atmel основана в 1984 году и является в настоящее время признанным мировым лидером в областях разработки, производства и маркетинга современных электронных компонентов, включая логические микросхемы с расширенными функциональными возможностями, микросхемы энергонезависимой памяти, а также интегральные схемы для радиочастотного диапазона и для смешанной обработки сигналов.

12.1.1. Микроконтроллеры с архитектурой MSC-51

В табл. 12.1 мы привели список 8-разрядных микроконтроллеров с архитектурой MSC-51. Наиболее простой контроллер — AT89C5131 — имеет два 10-битных АЦП, а остальные контроллеры имеют расширенную функциональность:

- ☐ AT89C51SND1 имеет встроенный MP3-декодер;
- ☐ AT85C5122 и AT89C5122 имеют интерфейс для чтения смарт-карт.

Таблица 12.1. 8-разрядные микроконтроллеры Atmel с архитектурой MCS-51

Тип	Питание, В	МГц	I/O	FLASH, Кбайт	SRAM, Кбайт	Интерфейс	АЦП, бит
AT89C5131	3,0—3,6	40	34 18	32	1,25+4 EEPROM	UART, USB, SPI	
AT89C5132	2,7—3,3	40	44 38	64	2,5	UART, USB, SPI, I2S	2x10
AT89C51SND1	2,7—3,3	20	44	64	2,25	UART, IDE, USB 1.1, SPI, I2S, MP3 Decoder	1x10
AT85C5122	3,6—5,5	16	46 13	32 CRAM	0,75	UART, USB, SPI, Smart Card	
AT89C5122	3,6—5,5	16	46 13	32	0,75	UART, USB, SPI, Smart Card	

Микроконтроллер AT89C5131 имеет следующие характеристики (более подробно мы будем рассматривать этот контроллер в *главах 13 и 14*).

❑ Ядро 80C52X2 (6 тактов на инструкцию):

- максимальная частота ядра 40 МГц;
- двойной указатель данных;
- полнодуплексный улучшенный UART (EUART);
- три 16-разрядных таймера-счетчика: T0, T1 и T2;
- 256 байт сверхоперативной памяти.

❑ 32 Кбайт встроенной флэш-памяти с внутрисхемным программированием через USB или UART.

❑ 4 Кбайт EEPROM для загрузочного сектора (3 Кбайт) и данных (1 Кбайт).

❑ 1 Кбайт встроенного расширенного ОЗУ (XRAM).

❑ USB 1.1 и USB 2.0 FS модуль с прерыванием на завершение передачи:

- конечная точка 0 для управления передачей: 32-байтный буфер FIFO;
- 6 программируемых конечных точек с направлениями ввода и вывода и с режимами передачи данных (Bulk), прерываний (Interrupt) и изохронный (Isochronous):
 - ◊ конечные точки 1, 2, 3: 32-байтный буфер FIFO;

- ◇ конечные точки 4, 5: размер буфера FIFO 2×64-байта с двойной буферизацией (режим Ping-pong);
 - ◇ конечная точка 6: 2×512-байтный буфер FIFO с двойной буферизацией (режим Ping-pong);
 - прерывания по приостановке/возобновлению;
 - сброс при подаче питания и сброс USB-шины;
 - генерация 48 МГц для полноскоростного функционирования шины;
 - отключение от USB-шины по запросу микроконтроллера;
 - 5-канальный программируемый счетный массив (PCA) с 16-разрядным счетчиком, быстродействующим выходом, сравнением/захватом фронтов, функциями ШИМ и сторожевого таймера.
 - Программируемый сторожевой таймер (однократно разрешает после сброса): от 50 мс до 6 с при 4 МГц.
 - Интерфейс подключения клавиатуры с генерацией прерывания на порте P1 (8 разрядов).
 - SPI-интерфейс.
 - 34 линии ввода/вывода.
 - 4 вывода для подключения светодиода с программируемым источником тока : 2—6—10 мА.
 - 4-уровневая система прерываний с приоритетами (11 источников).
 - Режимы холостого хода и экономичный.
 - Встроенный генератор 0: 32 МГц с аналоговой схемой ФАПЧ для синтеза 48 МГц.
 - Стабилизатор напряжения и выход опорного источника : 3,3 В, 4 мА.
 - Низкий диапазон напряжения источника питания:
 - 3,0 В—3,6 В;
 - максимальный рабочий ток 30 мА (при 40 МГц);
 - потребление 100 мкА в экономичном режиме.
 - Диапазон напряжения питания USB (недоступно в первой версии):
 - 3,6 В—5,5 В;
 - максимальный рабочий ток 30 мА (при 40 МГц);
 - ток потребления в экономичном режиме 200 мкА.
 - Коммерческий и промышленный температурные диапазоны.
 - Корпуса: PLCC52, VQFP64, MLF48, SO28.
- Ориентировочная стоимость этого микропроцессора \$9.

12.1.2. Контроллеры хабов

Контроллеры хабов представлены двумя микросхемами:

- ❑ AT43301 — контроллер LS/FS-хаба 1—4 с общим управлением питанием нисходящих портов;
- ❑ AT43312A — контроллер LS/FS-хаба 1—4 с индивидуальным управлением питанием нисходящих портов.

12.1.3. Микропроцессоры-хабы с ядром AVR

Микропроцессоры-хабы Atmel с ядром AVR представлены довольно широким модельным рядом:

- ❑ AT43320A — микроконтроллер на ядре AVR. Имеет встроенные USB-функцию и хаб с 4 внешними нисходящими портами, работающие в LS/FS-режимах, 512 байт ОЗУ, 32×8 регистров общего назначения, 32 программируемых вывода, последовательный и SPI-интерфейсы. Функция имеет 3 конечные точки с буферами FIFO размером 8 байт. Для нисходящих портов хаба предусмотрено индивидуальное управление питанием;
- ❑ AT43321 — контроллер клавиатуры на ядре AVR. Имеет встроенные USB-функцию и хаб с 4 внешними нисходящими портами, работающими в LS/FS-режимах, 512 байт ОЗУ, 16 Кбайт ПЗУ, 32×8 регистров общего назначения, 20 программируемых выводов, последовательный и SPI-интерфейсы. Функция имеет 3 конечные точки. Для нисходящих портов хаба предусмотрено индивидуальное управление питанием;
- ❑ AT43324 — микроконтроллер на ядре AVR. Имеет встроенные USB-функцию и хаб с 2 внешними нисходящими портами, работающие в LS/FS-режимах, 512 байт ОЗУ, 16 Кбайт ПЗУ, 32×8 регистров общего назначения, 34 программируемых вывода. Клавиатурная матрица может иметь размер 18×8. Контроллер имеет 4 выхода для подключения светодиодов. Функция имеет 3 конечные точки. Для нисходящих портов хаба предусмотрено индивидуальное управление питанием;
- ❑ AT43355 — микроконтроллер на ядре AVR. Имеет встроенные USB-функцию и хаб с 2 внешними нисходящими портами, работающие в LS/FS-режимах, 1 Кбайт ОЗУ, 24 Кбайт ПЗУ, 32×8 регистров общего назначения, 27 программируемых выводов, последовательный и SPI-интерфейсы, 12-канальный 10-разрядный АЦП. Функция имеет 1 управляющую конечную точку и 3 программируемых конечных точки с буферами FIFO размером 64/64/8 байт.

12.1.4. Другие микросхемы Atmel

Еще одна микросхема — AT76C711 — представляет собой контроллер моста на основе микроконтроллера AVR, между полноскоростной шиной USB и быстрым последовательным асинхронным интерфейсом. Эта микросхема имеет следующие характеристики:

- ☐ 8-разрядный AVR-микроконтроллер;
- ☐ тактовая частота 24 МГц;
- ☐ напряжение питания 3 В;
- ☐ потребление в активном режиме 50 мА;
- ☐ потребление в спящем режиме 0,2 мА;
- ☐ программируемый UART с 16-разрядными FIFO на стороне приема (максимальная производительность до 921 Кбод);
- ☐ программируемый SPI-интерфейс;
- ☐ полноскоростной USB-контроллер;
- ☐ встроенная SRAM данных емкостью 2 Кбит;
- ☐ встроенная двухпортовая RAM емкостью 2 Кбит, для сегментации и перекomпоновки пересылаемых между USB и UART интерфейсами пакетов;
- ☐ внутрисистемная SRAM кодов программы емкостью 128 Кбайт и организацией 8 К × 16;
- ☐ встроенная ROM загрузчика для загрузки программ во встроенную SRAM программ как через USB, так и через SPI интерфейс;
- ☐ одна USB конечная точка управления;
- ☐ пять программируемых конечных точек USB (до 64 байтов) с FIFO с двойным буферированием для взаимных пересылок;
- ☐ один 8-разрядный таймер-счетчик;
- ☐ один 16-разрядный таймер-счетчик;
- ☐ внутренние и внешние источники прерывания;
- ☐ программируемый сторожевой таймер;
- ☐ независимый бод-генератор UART.

Контроллер AT76C711 может быть использован в применениях, периферия которых поддерживает быстрые синхронные или асинхронные пересылки данных между ведущим устройством или другой периферией по высокоскоростным последовательным каналам типа USB:

- ☐ соединение сетевых интерфейсных карт (NIC, Network Interface Cards) с головными системами;
- ☐ беспроводная связь;

- ☐ мосты между микроконтроллерами и последовательными интерфейсами различных типов;
- ☐ мосты USB — UART;
- ☐ мосты USB — IrDA;
- ☐ мосты IrDA — UART;
- ☐ адаптация пакетов сетевых пакетных протоколов к требованиям USB.

Еще одна интересная микросхема — защищенный микроконтроллер AT90SC6464C-USB-I, оснащенный полноскоростным USB-интерфейсом (корпус PQFP44). Основные характеристики:

- ☐ не требует применения внешнего тактового генератора;
- ☐ базируется на RISC-микроконтроллере secureAVR;
- ☐ 128 Кбайт встроенной энергонезависимой памяти;
- ☐ 64 Кбайт встроенной флеш-памяти и столько же памяти типа EEPROM;
- ☐ два интерфейса USB 2.0 вместе со стандартным интерфейсом ISO 7816;
- ☐ мощные криптографические возможности.

Микроконтроллер AT90SC6464C-USB-I ориентирован на технологию eToken, использующуюся в защищенных приложениях на базе персональных компьютеров. Кроме того, он может быть встроен в периферийную аппаратуру, телевизионные приставки, модемы, карманные компьютеры, различные устройства обеспечения защиты авторских прав и другое оборудование. Везде, где требуется защита данных, этот микроконтроллер может стать высоконадежным и недорогим решением. Микроконтроллер способен обеспечивать безопасность транзакций, шифрование электронной почты, защиту программного обеспечения, файлов и т. п.

12.2. Микросхемы Cygnal

В этом разделе мы расскажем о некоторых микросхемах компании Cygnal. Компания Cygnal Integrated Products Inc. была учреждена в 1999 году, а с 10 декабря 2003 году она является подразделением компании Silicon Laboratories.

Множество статей по применению микросхем Cygnal можно найти на странице <http://www.premier-electric.com/products/cygnal/appnotes/b4fd3b74b2616767.html>.

12.2.1. Микропроцессоры C8051F320 и C8051F321

Основные технические характеристики микропроцессоров C8051F320 и C8051F321 приведены ниже:

- ☐ 8051-совместимое ядро CIP-51 производительностью до 25 MIPS;
- ☐ поддержка протокола USB 2.0;

- ❑ 16 Кбайт флэш-памяти программ секторами по 512 байт, каждый из которых может конфигурироваться как память программ или данных;
- ❑ 1 Кбайт + 256 байт ОЗУ + 1 Кбайт FIFO памяти USB;
- ❑ 25(21) цифровых входа/выхода (в микроконтроллерах F320 и F321 соответственно), конфигурируемых через встроенную коммутационную матрицу CROSSBAR и совместимых с пятивольтовой периферией без дополнительных внешних преобразователей уровней;
- ❑ расширенный обработчик до 16 источников прерываний;
- ❑ четыре 16-битных таймера общего применения;
- ❑ программируемая 16-битная счетная матрица с пятью модулями захвата/сравнения и возможностью организации на ее базе ШИМ-генератора;
- ❑ встроенные прецизионный супервизор напряжения питания и двуправленный сигнал сброса, который может использоваться как системный сброс для других устройств схемы;
- ❑ встроенный JTAG-интерфейс последовательного программирования Flash-памяти и внутрисхемный отладчик программ в режимах: пошаговом, с заданием точек остановки или реального времени;
- ❑ два встроенных компаратора напряжения с программируемыми гистерезисом и временем срабатывания, конфигурируемые как источники прерывания или сброса (ток потребления одного компаратора менее 0,5 мкА);
- ❑ встроенный датчик температуры +3 °C;
- ❑ быстродействующий АЦП (17-канальный в F320, 13-канальный в F321) разрешением 10 бит и производительностью 200 тысяч преобразований в секунду. Функционально АЦП содержит два встроенных аналоговых мультиплексора и может работать как в дифференциальном, так и интегральном режиме преобразований. Опорное напряжение АЦП может задаваться внутренним источником V_{REF} , напряжением питания микроконтроллера или внешним выводом;
- ❑ АЦП имеет функцию программируемого "оконного детектора". В регистрах микроконтроллера программируется нижнее и верхнее значение напряжения, которое необходимо отслеживать. Если напряжение на входе АЦП выйдет за пределы заданных пороговых значений, генерируется соответствующее прерывание. Таким образом, функция "оконный детектор" значительно экономит ресурсы процессора, избавляя программу от необходимости "рутинного" циклического опроса;
- ❑ максимальная погрешность АЦП составляет +1 LSB (младший разряд отсчета);
- ❑ аппаратно-встроенные интерфейсы SMBusT/I2CT, SPIT и UART;

- ❑ USB контроллер содержит:
 - универсальный последовательный контроллер (SIE);
 - FIFO-буфер на 1 Кбайт;
 - интегрированный приемопередатчик, не требующий при подключении внешних пассивных компонентов;
 - схему восстановления частоты и внутренний генератор, позволяющий контроллеру USB работать в полноскоростном и низкоскоростном режимах;
- ❑ встроенный стабилизатор напряжения на 3 В с током нагрузки до 100 мА, позволяющий запитывать микроконтроллер непосредственно от USB-шины. Кроме этого, стабилизатор имеет программное управление и внешний выход нагрузки, который может использоваться как напряжение питания других 3-вольтовых компонентов схемы;
- ❑ напряжение питания ядра микроконтроллера от 2,7 до 3,6 В. Токи потребления для различных режимов работы составляют величины от менее чем 0,1 мкА до 25 мА.

Для программирования и отладки микроконтроллеров планируется использовать специализированный комплект разработки C8051F320DK-E, включающий интегрированные среды Sygnal IDE и Keil uVision (www.keil.com) для разработки и программирования на С и ассемблере аппаратные средства отладки (плата эмулятора и устройство преобразования RS232-JTAG), набор соединителей и сетевой блок питания.

Стоимость микросхем C8051F320 и C8051F321 составляет приблизительно \$50, однако комплект разработки стоит порядка \$1500.

12.2.2. Другие микросхемы Sygnal

Список некоторых USB-микросхем и их основные характеристики представлены в табл. 12.2. Цена на эти микросхемы варьируются от \$50 до \$250 без учета отладочного комплекта и программатора, цена которых составляет от \$650 до \$1000.

Таблица 12.2. Микросхемы Sygnal

Тип [MIPS] ¹	FLASH, (Кбайт)	RAM (байт)	В/в ²	Шины	Тай-меров	АЦП ³	ЦАП ³
C8051F000 [20]	32	256	32	UART, SMBus, SPI	4	8×12	2×12
C8051F001 [20]	32	256	16	UART, SMBus, SPI	4	8×12	2×12

Таблица 12.2 (продолжение)

Тип [MIPS]¹	FLASH, (Кбайт)	RAM (байт)	В/В²	Шины	Тай- меров	АЦП³	ЦАП³
C8051F002 [20]	32	256	8	UART, SMBus, SPI	4	4×12	2×12
C8051F005	32	2304	32	UART, SMBus, SPI	4	8×12	2×12
C8051F006	32	2304	16	UART, SMBus, SPI	4	8×12	2×12
C8051F007	32	2304	8	UART, SMBus, SPI	4	4×12	2×12
C8051F010 [20]	32	256	32	UART, SMBus, SPI	4	8×10	2×12
C8051F011 [20]	32	256	16	UART, SMBus, SPI	4	8×10	2×12
C8051F012 [20]	32	256	8	UART, SMBus, SPI	4	4×10	2×12
C8051F015	32	2304	32	UART, SMBus, SPI	4	8×10	2×12
C8051F016	32	2304	16	UART, SMBus, SPI	4	8×10	2×12
C8051F017	32	2304	8	UART, SMBus, SPI	4	4×10	2×12
C8051F018	16	1280	32	UART, SMBus, SPI	4	8×10	нет
C8051F019	16	1280	16	UART, SMBus, SPI	4	8×10	нет
C8051F020	64	4352*	64	2 UARTs, SMBus, SPI	5	8×12	2×12
C8051F021	64	4352*	32	2 UARTs, SMBus, SPI	5	8×12	2×12
C8051F022	64	4352*	64	2 UARTs, SMBus, SPI	5	8×10	2×12
C8051F023	64	4352*	32	2 UARTs, SMBus, SPI	5	8×10	2×12
C8051F040	64	4352*	64	CAN2.0B, 2 UARTs, SMBus, SPI	5	13×12	2×12
C8051F041	64	4352*	32	CAN2.0B, 2 UARTs, SMBus, SPI	5	13×12	2×12
C8051F042	64	4352*	64	CAN2.0B, 2 UARTs, SMBus, SPI	5	13×10	2×12

Таблица 12.2 (продолжение)

Тип [MIPS] ¹	FLASH, (Кбайт)	RAM (байт)	В/В ²	Шины	Тай- меров	АЦП ³	ЦАП ³
C8051F043	64	4352*	32	CAN2.0B, 2 UARTs, SMBus, SPI	5	13×10	2×12
C8051F060	64	4352*	59	CAN2.0B, 2 UARTs, SMBus, SPI	5	2×16	2×12
C8051F061	64	4352*	24	CAN2.0B, 2 UARTs, SMBus, SPI	5	2×16	2×12
C8051F062	64	4352*	59	CAN2.0B, 2 UARTs, SMBus, SPI	5	2×16	2×12
C8051F063	64	4352*	24	CAN2.0B, 2 UARTs, SMBus, SPI	5	2×16	2×12
C8051F120 [100]	128	8448*	64	2 UARTs, SMBus, SPI	5	8×12	2×12
C8051F121 [100]	128	8448*	32	2 UARTs, SMBus, SPI	5	8×12	2×12
C8051F122 [100]	128	8448*	64	2 UARTs, SMBus, SPI	5	8×10	2×12
C8051F123 [100]	128	8448*	32	2 UARTs, SMBus, SPI	5	8×10	2×12
C8051F124 [50]	128	8448*	64	2 UARTs, SMBus, SPI	5	8×12	2×12
C8051F125 [50]	128	8448*	32	2 UARTs, SMBus, SPI	5	8×12	2×12
C8051F126 [50]	128	8448*	64	2 UARTs, SMBus, SPI	5	8×10	2×12
C8051F127 [50]	128	8448*	32	2 UARTs, SMBus, SPI	5	8×10	2×12
C8051F206	8	1280	32	UART, SPI	3	32×12	нет
C8051F220	8	256	32	UART, SPI	3	32×8	нет
C8051F221	8	256	22	UART, SPI	3	32×8	нет
C8051F226	8	1280	32	UART, SPI	3	32×8	нет

Таблица 12.2 (окончание)

Тип [MIPS] ¹	FLASH, (Кбайт)	RAM (байт)	В/В ²	Шины	Тай- меров	АЦП ³	ЦАП ³
C8051F230	8	256	32	UART, SPI	3	нет	нет
C8051F231	8	256	22	UART, SPI	3	нет	нет
C8051F236	8	1280	32	UART, SPI	3	нет	нет
C8051F300	8	256	8	UART, SMBus	3	8×8	нет
C8051F301	8	256	8	UART, SMBus	3	нет	нет
C8051F302	8	256	8	UART, SMBus	3	8×8	нет
C8051F303	8	256	8	UART, SMBus	3	нет	нет
C8051F304	4	256	8	UART, SMBus	3	нет	нет
C8051F305	2	256	8	UART, SMBus	3	нет	нет
C8051F310	16	1280	29	UART, SMBus, SPI	4	21×10	нет
C8051F311	16	1280	25	UART, SMBus, SPI	4	17×10	нет
C8051F320	16	2304	25	USB 2.0, UART, SMBus, SPI	4	18×10	нет
C8051F321	16	2304	21	USB 2.0, UART, SMBus, SPI	4	13×10	нет
C8051F330	8	768	17	UART, SMBus, SPI	4	16×10	1×10
C8051F331	8	768	17	UART, SMBus, SPI	4	нет	нет

1 — операций в секунду (если не указано, то 25); 2 — число линий ввода/вывода; 3 — число каналов, разрядность; * — возможность подключения внешней памяти.

12.3. Микросхемы FTDI

Компания Future Technology Devices International (FTDI) была создана в начале 1990 года в Великобритании Фредом Дартсом (Fred Dart). Профиль компании — разработка и производство микросхем преобразователей интерфейсов (USB-UART, USB-FIFO и др.), а также программного обеспечения (драйверы, утилиты) и отладочных средств для разработки преобразователей интерфейсов.

12.3.1. Микросхемы FT232AM и FT232BM

Микросхемы FT232AM и FT232BM представляют собой однокристалльный асинхронный двунаправленный преобразователь USB в последовательный интерфейс (RS-232, RS-422 или RS-485). Микросхема FT232BM является расширением FT232AM, но при этом сохраняет полную совместимость по контактам и функциональности. Мы будем рассматривать FT232BM (рис. 12.1).



Рис. 12.1. Микросхема FT232BM

Микросхема FT232BM включает в себя: USB-приемопередатчик, UART-контроллер и буферы, стабилизатор напряжения, умножитель частоты и другие функциональные узлы, которые делают ее готовым решением для быстрой и недорогой модернизации системы с COM-портом для работы с интерфейсом USB. Эту микросхему можно устанавливать в USB-модемах, переходниках COM-USB, сканерах штрихкода, измерительной аппаратуре — фактически в любых устройствах, ранее использовавших сравнительно медленные RS-интерфейсы. Она 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 драйверы создают впечатление, что обмен идет через обычный COM-порт.

Микросхема FT232BM имеет следующие характеристики (более подробно мы будем рассматривать эту микросхему в *главе 15*):

- ❑ простой и дешевый преобразователь интерфейса USB в последовательный RS-интерфейс;
- ❑ поддержка сигналов управления потоком и модема;
- ❑ UART с поддержкой 7- или 8-битовых данных, 1 или 2 стоп-бит;
- ❑ контроль четности: чет, нечет, маркер, паритет, нет контроля;
- ❑ встроенная поддержка ошибок приема-передачи и обрыва линии;
- ❑ передача данных на скорости до 3 Мбит (TTL);
- ❑ передача данных на скорости до 1 Мбит (RS-232);

- ☐ передача данных на скорости до 3 Мбит (RS-422, RS-485);
- ☐ приемный буфер размером 384 байта, буфер передатчика размером 128 байт для повышения пропускной способности;
- ☐ управляемый тайм-аут буфера приема (настраивается от 1 до 255 мс, что позволяет гибко настраивать быстродействие устройства при передаче коротких пакетов данных);
- ☐ полная поддержка управления потоком X_{ON}/X_{OFF} ;
- ☐ встроенная поддержка специальных символов (символов, при передаче которых вызывается обработчик прерывания);
- ☐ автоматическое переключение линий для RS-485;
- ☐ поддержка пробуждения и засыпания для USB-интерфейса с помощью выводов SLEEP# и RI#;
- ☐ поддержка питания от шины с помощью вывода PWREN#;
- ☐ встроенный преобразователь сигналов UART и управляющих сигналов для согласования с 5- и 3,3-вольтовой логикой;
- ☐ встроенный умножитель частоты от 6 МГц до 48 МГц;
- ☐ встроенный регулятор 3,3 В для USB;
- ☐ автоматический сброс (Reset) при подаче питания;
- ☐ поддержка всех типов передач данных (передача пакетов данных, управляющих пакетов, передача данных по прерыванию и изохронных данных);
- ☐ поддержка протоколов хост-контроллера UHCI, OHCI, EHCI;
- ☐ совместимость с USB 1.1 и USB 2.0;
- ☐ передача идентификаторов VIP, PID, Serial Number и Product;
- ☐ программируемая EEPROM-память через USB-интерфейс;
- ☐ аппаратно поддерживаются EEPROM с протоколом Microwire (93C46, 93C56, 93C66);
- ☐ поддержка операционных систем Windows 98, Windows 2000/ME/XP, Windows CE, MAC OS 8, MAC OS 9, MAC OS X, Linux 2.40 и выше.

Включение в схему FT232BM требует минимального количества дополнительных внешних компонентов (см. главу 15).

Стоимость микросхемы FT232BM составляет примерно \$4,5.

12.3.2. Микросхемы FT245AM и FT245BM

Микросхемы FT245AM и FT245BM представляют собой преобразователь USB в параллельный интерфейс. FT245BM является расширением FT245AM (рис. 12.2). По своим характеристикам FT245BM полностью аналогична

FT232BM, отличие заключается в поддержке 8-битного параллельного интерфейса вместо последовательного. Параллельный интерфейс представлен 1-байтным портом шины данных, сигналами чтения и записи (RD/WR, TXE/RXE).



Рис. 12.2. Микросхема FT245AM

Основные характеристики FT245BM:

- ☐ однокристалльный двунаправленный преобразователь USB-FIFO;
- ☐ скорость обмена до 8 Мбит/с;
- ☐ буфер приема данных 384 байта с программируемым тайм-аутом по приему, буфер передачи — 128 байт;
- ☐ совместимость со спецификациями USB 1.1 и USB 2.0;
- ☐ совместимость с интерфейсами хост-контроллеров UHCI/OHCI/EHCI;
- ☐ напряжение питания от 4,4 В до 5,25 В и интегрированный стабилизатор напряжения 3,3 В;
- ☐ встроенная схема формирования сигнала "Сброс";
- ☐ встроенный умножитель частоты от 6 МГц до 48 МГц;
- ☐ возможность программирования микросхем EEPROM с протоколом Microwire под управлением USB;
- ☐ встроенный преобразователь уровней FIFO и управляющих сигналов для управления 5 В и 3,3 В логикой.

Стоимость микросхемы FT245BM составляет примерно \$4,5.

12.3.3. Микросхема FT2232BM

Микросхема FT2232C — новая разработка FTDI, появившаяся на рынке в 2004 году. Она представляет собой аналог FT232BM и FT245BM, но имеет возможность работы с параллельным и последовательным интерфейсами в зависимости от выбранной конфигурации.

Основные характеристики FT2232L:

- ❑ однокристалльный двунаправленный преобразователь USB в FIFO или UART;
- ❑ двояный буфер приема данных 384 байта с программируемым тайм-аутом по приему, буфер передачи — 128 байт;
- ❑ совместимость со спецификациями USB 1.1 и USB 2.0;
- ❑ совместимость с интерфейсами хост-контроллеров UHCI/OHCI/ЕHCI;
- ❑ напряжение питания от 4,4 В до 5,25 В и интегрированный стабилизатор напряжения 3,3 В.
- ❑ возможность программирования микросхем EEPROM с протоколом Microwire под управлением USB;
- ❑ встроенный преобразователь уровней и управляющих сигналов для управления 5 В и 3,3 В логикой.

12.3.4. Микросхема FT8U100AX

Микросхема FT8U100AX является однокристалльным USB хаб-контроллером. Она поддерживает 7 нижних и 1 верхний USB-порт по спецификации USB 1.1 и интерфейсы хост-контроллеров UHCI/OHCI.

FT8U100AX имеет встроенное 8-битное микропроцессорное ядро с 256 байтами ОЗУ, работающее от внешнего генератора на частоте 48 МГц. Программа FT8U100AX должна размещаться во внешней памяти.

Кроме USB, FT8U100AX имеет порты для подключения внешней периферии: последовательный порт RS-232, порт PS/2-клавиатуры, порт PS/2-мыши (двухкнопочная, трехкнопочная или с колесиком) и порт инфракрасного канала передачи данных. Кроме этого, есть встроенный последовательный двухпроводной интерфейс с поддержкой режимов ведущий-подчиненный (master/slave).

В FT8U100AX есть специальные выводы для подключения светодиодов индикации статуса USB. Светодиоды многофункциональные и отображают текущий трафик, режим HS/LS, включение защиты по току или помехам периферии, состояния активности, сброса или приостановки систем.

На каждом из подключенных портов USB осуществляется автоматический контроль напряжения питания и перегрузки.

Программное обеспечение для FT8U100AX можно свободно скачать с официального сайта FTDI (www.ftdichip.com).

Микросхема использует напряжение питания 3,3 В и имеет 100-выводной PQFP тип корпуса.

Стоимость такой микросхемы составляет примерно \$5,6.

12.3.5. Отладочные комплекты и модули

Для отладки могут использоваться модули DLP-USB232M и DLP-USB245M, которые выполняют преобразование потока данных USB 1.1 FS в поток асинхронных последовательных данных с уровнями 3,3 В/5 В и скоростью до 3 Мбит/с или в поток параллельных данных с уровнями 3,3 В/5 В и скоростью до 8 Мбит/с соответственно. Стоимость такой платы составляет около \$30.

Конструктивно модули DLP представляют собой миниатюрную четырехслойную плату, вставляемую в колодку DIP24 и имеющую разъем USB (рис. 12.3).

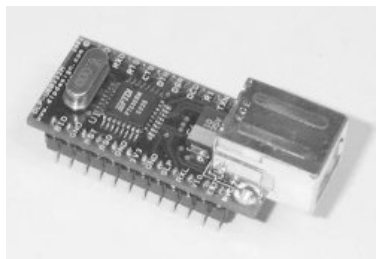


Рис. 12.3. Модуль DLP-USB232M для отладки FT232BM

Модули DLP выполнены на базе микросхем FT8U232BM и FT8245BM, которые аппаратным образом реализуют поддержку протокола USB и содержат FIFO-буферы на прием и передачу данных. Для задания USB-настроек модули содержат программируемую через USB EEPROM-память.

Драйверы под Windows 98/2000/ME/XP и примеры программирования можно загрузить с фирменного сайта www.ftdichip.com.

Кроме этих комплектов существуют и другие отладочные платы:

- ☐ специальный комплект DLP-USB1 (на основе FT245AM);
- ☐ компактный модуль USB MOD-02 (на основе FT245AM);
- ☐ модуль DLP-245PB, содержащий помимо FT245BM еще и микроконтроллер PIC16F877;
- ☐ модуль DLP-245SY, содержащий FT245BM и Scenix SX48.

Платы эмуляторов DPL-EvalP и DPL-EvalS с установленными микроконтроллерами PIC16F870 и Scenix SX28 и температурными датчиками DS18S20. Платы позволяют выполнять разработки и демонстрируют различные аспекты применения компонентов USB: измерение температуры, соединение с USB-хостом персонального компьютера, мониторинг питания схемы во время работы USB. Кроме этого, память программ установленных

на платах микроконтроллеров может быть перепрограммирована с использованием кабеля-адаптера DLP-FLASH.

Для отладки FT2232C может использоваться модуль DLP-USB2232M. Существует также модуль MORPH-IC, представляющий собой простое и гибкое устройство ввода/вывода цифровых сигналов. Обеспечивается поддержка отладочных средств Altera и конфигурирование FPGA через USB. Модуль имеет 36 двунаправленных выводов, 4 входа, 1 выход, 8 разделяемых выводов, 576 логических элементов и 1,5 Кбайт ОЗУ.

12.3.6. Драйверы

Драйверы виртуального COM-порта (VCP, Virtual Communication Port) организуют в системе последовательный порт (в дополнение к существующим аппаратным), и переадресуют все обращения к нему в прямые запросы непосредственно оборудованию. Программное обеспечение взаимодействует с USB-устройствами через стандартные вызовы VCOMM API Windows или с помощью любых распространенных коммуникационных компонентов. Существует две разновидности VCP-драйверов для Windows:

- ❑ драйверы с поддержкой PnP обычно используются для организации универсальных конвертеров USB в RS-232, модемов и других применений, отвечающих спецификации Windows RS-232 Plug and Play (<http://www.institute-rt.ru/ftdi/P8002104.ZIP>);
- ❑ драйверы без поддержки PnP применяются во встраиваемых системах и устройствах, в которых периферия, подключенная к FT232AM, не задействует интерфейс Windows Plug and Play для загрузки своего собственного программного обеспечения (<http://www.institute-rt.ru/ftdi/N8002101.zip>).

Драйверы для Linux созданы сторонними разработчиками и включены в ядро начиная с версии 2.4. Более подробная информация содержится на сайте <http://ftdi-usb-sio.sourceforge.net>. Драйверы для Apple Mac OS X можно загрузить с сайта компании "Премьер-Электрик" (<http://www.premier-electric.com/files/Ftdi/Soft/FTDIUSBSerialDriver105.pkg.hqx>).

Процедура установки VCP-драйвера в Windows ничем не отличается от установки драйвера любого другого устройства. Все файлы из архива, в котором поставляется драйвер, необходимо переписать на дискету или в специально созданную папку на жестком диске. Далее, подключив к USB преобразователь интерфейса (или другое устройство на микросхемах FT8U232AM, FT8U245AM), откройте окно **Установка/Удаление оборудования** (Add/Remove Hardware) и следуйте указаниям Мастера установки. Чтобы убедиться в успешной установке драйверов, откройте вкладку **Менеджер устройств** (Device Manager) в окне **Свойства системы** (System Properties) и найдите в списке USB High Speed Serial Converter. Если ничего похожего там нет, процедуру инсталляции стоит повторить еще раз.

После успешной установки драйверов в пункте USB High Speed Serial Converter появится устройство USB Serial port (COMx), где x — номер виртуального последовательного порта. Основные параметры COMx идентичны параметрам и настройкам стандартного последовательного порта. Можно изменить скорость работы UART, число бит в слове, режим проверки четности, длину стоп-бита, способ управления потоком. Единственное отличие — возможность выбрать или изменить номер порта x в окне **Дополнительные настройки порта** (Advanced Port Settings). В качестве инструмента программирования виртуального COM-порта для Windows 98 можно использовать семейство стандартных функций VCOMM API.

Драйверы прямого доступа D2XX (<http://www.institute-rt.ru/ftdi/D10606.zip>) для Windows предлагают альтернативное решение и позволяют приложениям взаимодействовать с FT232AM и FT245AM непосредственно, используя специализированную DLL вместо функций работы с портами VCP. D2XX-интерфейс включает в себя низкоуровневый драйвер Windows WDM, управляющий устройствами через стек USB Windows, и динамическую библиотеку для взаимодействия с приложениями (написанными на C++, Delphi, Visual Basic и т. д.). Установочный INF-файл, программа для удаления драйверов и пользовательская документация дополняют пакет поставки.

Загрузить примеры программ можно по следующим адресам:

- ❑ C++ Builder — <http://www.institute-rt.ru/ftdi/D2XXAPP.ZIP>;
- ❑ Delphi — <http://www.institute-rt.ru/ftdi/d2xxappl.zip>;
- ❑ Visual C++ — <http://www.institute-rt.ru/ftdi/dlpvcc2.zip>;
- ❑ Visual Basic — http://www.institute-rt.ru/ftdi/demo_vb6.zip.

12.3.7. Дополнительные утилиты

Утилиты FT232AM, FT245AM и FT8U100AX прошиваются с помощью EEPROM 93C46. Параметры, которые могут быть изменены:

- ❑ идентификатор USB VID и PID;
- ❑ кодовая строка (сигнатура) производителя;
- ❑ серийный номер устройства.

Специальные утилиты позволяют настраивать устройства "на лету" — т. е. в конечное изделие запаивается чистая EEPROM, загружается с помощью этих утилит и сразу тестируется:

- ❑ FTD2XXST — утилита для инициализации и отладки устройств на базе FT232AM и FT245AM (<http://www.institute-rt.ru/ftdi/FTD2XXST.ZIP>). Она основана на последних драйверах D2XX и работает на платформах Windows 98, Windows ME и Windows 2000;

- ❑ **E2PROG** — программа для работы с FT8U100AX (<http://www.institute-rt.ru/ftdi/e2prog.zip>). Запускается под Windows 98 и задействует предварительно установленный драйвер VCP.

12.3.8. Другие модули

Модуль Н.Т.Н.USB EASY Tap (стоимость примерно \$49) является мостом между шиной USB и сетью RS-485, построенной на витой паре и позволяющей предавать данные на расстояния до 1200 м.

12.4. Микросхемы Intel

Из линейки USB-микросхем Intel мы опишем несколько:

- ❑ **8x931Ax** — микроконтроллер с архитектурой MSC-51. Имеет встроенную USB-функцию, работающую в LS/Fs-режимах, 256 байт ОЗУ, 0 или 8 Кбайт ПЗУ, 8×4 регистра общего назначения, 32 программируемых вывода, последовательный интерфейс, интерфейс управления клавиатурой. Функция имеет 3 конечные точки с буферами FIFO размером соответственно 8, 16 и 8 байт;
- ❑ **8x931Hx** — микроконтроллер с архитектурой MSC-51. Имеет встроенную USB-функцию и хаб с 4 внешними нисходящими портами, работающими в LS/Fs-режимах, 256 байт ОЗУ, 0 или 8 Кбайт ПЗУ, 8×4 регистра общего назначения, 32 программируемых вывода, последовательный интерфейс, интерфейс управления клавиатурой. Функция имеет 3 конечные точки с буферами FIFO размером соответственно 8, 16, и 8 байт;
- ❑ **8x930Ax** — микроконтроллер с архитектурой MSC-251. Имеет встроенную USB-функцию, работающую в LS/Fs-режимах, 1024 байта ОЗУ, 0, 8 или 16 Кбайт ПЗУ, 40 регистров общего назначения, 32 программируемых вывода, последовательный интерфейс. Функция имеет 4 (или 6) конечных точек с буферами FIFO размером соответственно 16, 1024, 16, 16 (или 16, 256, 32, 32, 32, 16) байт.
- ❑ **8x930Hx** — микроконтроллер с архитектурой MSC-251. Имеет встроенную USB-функцию и хаб с 4 внешними нисходящими портами, работающими в LS/Fs-режимах, 1024 байта ОЗУ, 0, 8 или 16 Кбайт ПЗУ, 40 регистров общего назначения, 32 программируемых вывода, последовательный интерфейс. Функция имеет 4 конечные точки с буферами FIFO размером соответственно 16, 1024, 16, 16 байт.

Основным отличием микросхем семейства 8x930 и 8x931 является то, что в основе приборов семейства 8x931 использован микроконтроллер с архитектурой MCS-51, а в основе приборов семейства 8x930 — микроконтроллер с архитектурой MCS-251, которая, по утверждению специалистов фирмы, обеспечивает производительность в 15 раз бóльшую, чем архитектура MCS-51.

С точки зрения спецификации USB микросхемы семейств 8х930 и 8х931 являются USB-устройствами, которые обслуживаются как функции, обеспечивающие интерфейс с периферией, и как хабы (8х930Нх и 8х931Нх), обеспечивающие USB-порты для дополнительной периферии (рис. 12.4). Микросхемы семейства 8х931Нх оснащены встроенным интерфейсом управления клавиатурой.

Таблица 12.3 содержит основные характеристики микросхем этого семейства.

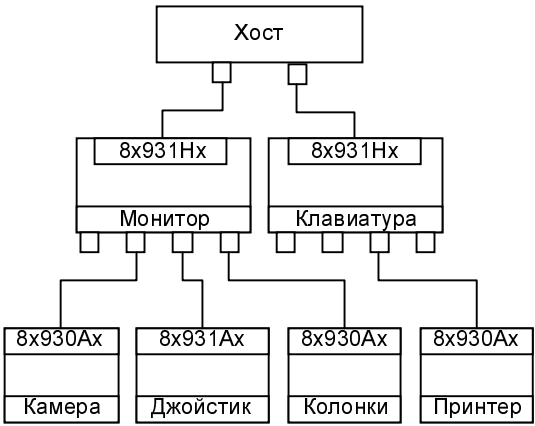


Рис. 12.4. Архитектура использования микросхем семейства 8х930 и 8х931

Таблица 12.3. Сравнительная характеристика микросхем Intel

	8х931Нх	8х931Ах	8х930Нх	8х930Ах
Встроенная ROM, Кбайт	0 или 8	0 или 8	0, 8 или 16	0, 8 или 16
Встроенная RAM, байт	256	256	1024	1024
Встроенная периферия				
Таймеры/счетчики	3	3	3	3
Последовательный порт I/O	Есть	Есть	Есть	Есть
РСА, аппаратный сторожевой таймер	Нет	Нет	Есть	Есть
Совместимость кодов с микроконтроллерами MCS R 51	Есть	Есть	Есть	Есть
Совместимость кодов с микроконтроллерами MCS R 251	Нет	Нет	Есть	Есть
Интерфейс клавиатуры	Есть	Есть	Нет	Нет

Таблица 12.3 (продолжение)

	8x931Hx	8x931Ax	8x930Hx	8x930Ax
USB-характеристики				
Совместимость с USB спецификациями	1.0	1.0	1.0	1.0
Встроенные USB приемопередатчики	Есть	Есть	Есть	Есть
Автоматическое управление FIFO приема/передачи	Есть	Есть	Есть	Есть
Тактовая частота (кварц/PLL), МГц	12	12	12	12
Производительность USB на полной скорости, Мбит/с	12	12	12	12
Режим пониженной тактовой частоты	Есть	Есть	Есть	Есть
Режимы остановки/возобновления	Есть	Есть	Есть	Есть
USB векторы прерывания (хаба, функции и остановки/возобновления)	Есть	Есть	Есть	Есть
Разделение сигналов сброса	Есть	Есть	Нет	Есть
USB-функции				
Количество конечных точек функций	3	3	4	4 (или 6)
Объем FIFO приема/передачи, байт				
Конечная точка 0	8	8	16	16
Конечная точка 1	16	16	0–1024	0–1024 (или 256)
Конечная точка 2	8	8	16	16 (или 32)
Конечная точка 3	—	—	16	16 (или 32)
Конечная точка 4	—	—	—	(32)
Конечная точка 5	—	—	—	(16)
Возможности USB-хаба				
Встроенный нисходящий порт	USB порт 1	—	USB-порт 4	—

Таблица 12.3 (окончание)

	8х931Нх	8х931Ах	8х930Нх	8х930Ах
Возможности USB-хаба				
Производительность USB на полной скорости, Мбит/с	12	—	12	—
Внешние нисходящие порты	4 (USB-порты 2,3,4,5)	—	4 (USB-порты 1,2,3,5)	—
Производительность USB-хаба (полная/малая скорости), Мбит/с	12 /1,5	—	12 /1,5	—
Емкость FIFO приема/передачи конечной точки хаба 0, байт	8	—	16	—
Емкость буфера регистра передачи данных конечной точки хаба 1, байт	1	—	1	—
Ядро микроконтроллера				
Архитектура	MCSR 51 (аккумуляторная)	MCSR 51 (аккумуляторная)	MCSR 251 (регистровая)	MCSR 251 (регистровая)
Адресное пространство				
Память программ/данных, Кбайт	64/64	64/64	256 (единое адресное пространство)	256 (единое адресное пространство)
Внешняя шина (мультиплексируемая)				
Адрес, разрядов	16	16	16, 17 или 18	16, 17 или 18
Данные, разрядов	8	8	8	8
Количество регистров	8	8	40	40
Параллельные порты I/O	4	4	4	4
Напряжение питания, В	от 4,4 до 5,25	от 4,15 до 5,25	от 4,0 до 5,25	от 4,0 до 5,25
Диапазон рабочих температур	от 0 °С до 70 °С	от 0 °С до 70 °С	от 0 °С до 70 °С	от 0 °С до 70 °С
Корпуса	64SDIP, 64QFP, 68PLCC	64SDIP, 64QFP, 68PLCC	64SDIP, 68PLCC	68PLCC

12.5. Микросхемы Microchip

В этом разделе мы приведем краткие описания нескольких микросхем компании Microchip:

- ❑ **PIC16C745** — микроконтроллер с архитектурой PIC. Имеет встроенную USB-функцию, работающую в LS-режиме, 256 байт ОЗУ, 14 336 байт ПЗУ, 22 программируемых вывода, последовательный интерфейс, 5-канальный 8-битный АЦП;
- ❑ **PIC16C765** — микроконтроллер с архитектурой PIC. Имеет встроенную USB-функцию, работающую в LS-режиме, 256 байт ОЗУ, 14 336 байт ПЗУ, 33 программируемых вывода, последовательный интерфейс, 8-канальный 8-битный АЦП;
- ❑ **PIC18F2450** — микроконтроллер с архитектурой PIC. Имеет встроенную USB-функцию, работающую в LS/FS-режиме, 1536 байт ОЗУ, 16 384 байт ПЗУ, 19 программируемых выводов, последовательный и SPI-интерфейсы, 5-канальный 10-битный АЦП. Функция имеет 8 конечных точек;
- ❑ **PIC18F2550** — микроконтроллер с архитектурой PIC. Имеет встроенную USB-функцию, работающую в LS/FS-режиме, 1536 байт ОЗУ, 32 768 байт ПЗУ, 19 программируемых выводов, последовательный, CAN- и SPI-интерфейсы, 5-канальный 10-битный АЦП. Функция имеет 8 конечных точек;
- ❑ **PIC18F4450** — микроконтроллер с архитектурой PIC. Имеет встроенную USB-функцию, работающую в LS/FS-режиме, 1536 байт ОЗУ, 16 384 байт ПЗУ, 34 программируемых вывода, последовательный, CAN- и SPI-интерфейсы, 8-канальный 10-битный АЦП. Функция имеет 8 конечных точек;
- ❑ **PIC18F4550** — микроконтроллер с архитектурой PIC. Имеет встроенную USB-функцию, работающую в LS/FS-режиме, 1536 байт ОЗУ, 32 768 байт ПЗУ, 34 программируемых вывода, последовательный, CAN- и SPI-интерфейсы, 8-канальный 10-битный АЦП. Функция имеет 8 конечных точек.

12.6. Микросхемы Motorola

Микросхемы для USB производства компании Motorola представляют собой достаточно специализированные периферийные микросхемы:

- ❑ **RD68HC08USBHKEYBD** — клавиатурный интерфейс с хабом;
- ❑ **RD68HC908DDCICP** — интерфейс для DDC-интерфейса (VGA-портов мониторов и т. п.);

- ❑ **RD68HC908USB** — интерфейс к принтерам, сканерам, устройствам чтения штрихкодов и т. п.;
- ❑ **RD68HC908USBMKEYBD** — интерфейс USB/PS2-клавиатуры;
- ❑ **RD68HC908USBMSE** — интерфейс оптической USB-мыши;
- ❑ **RD68HC908USBSKEY** — интерфейс защитного ключа;
- ❑ **RD68HC908WOMK** — интерфейс оптической USB-мыши и мультимедийной клавиатуры.

12.7. Микросхемы Philips

Отличительная черта микросхем Philips — наличие большого числа собственных технологий, например:

- ❑ **SoftConnect** — подключение к USB сопровождается подачей на вывод D+ (для высокоскоростных USB-устройств) высокого напряжения через нагрузочный резистор с сопротивлением 1,5 кОм. Микросхемы оснащаются встроенными резисторами, но по умолчанию они не подключены. Подключение организуется командой, формируемой системным микроконтроллером, что позволяет микроконтроллеру завершить последовательность инициализации прежде, чем будет разрешено подсоединение к USB;
- ❑ **GoodLink** — индикация правильного подсоединения USB. В процессе пересчета светодиодный индикатор будет кратковременно вспыхивать в соответствии с трафиком. После того как PDIUSB12 будет пересчитан и сконфигурирован LED, индикатор будет светиться непрерывно. Последующие успешные пересылки (подтвержденные) *в* и *из* приборов будут подтверждаться кратковременным выключением LED. В процессе останова (suspend) LED будет находиться в выключенном состоянии;
- ❑ **PSIE** — Philips Serial Interface Engine — механизм последовательного интерфейса реализует нижний уровень протокола USB. SIE реализован полностью аппаратно, что способствует обеспечению высокого быстродействия и исключает необходимость вмешательства микропрограммных средств. К функциям этого блока относятся: распознавание синхронизирующих посылок, параллельные и последовательные преобразования, введение и выведение битов заполнения, проверка и генерация CRC, проверка и генерация PID, распознавание адреса, оценка и генерация квитирования.

12.7.1. Микросхемы USB

Некоторые микросхемы производства Philips представлены в табл. 12.4.

Таблица 12.4. Микросхемы USB производства Philips

Название	Функциональное назначение	Соответствие спецификациям	Производительность, Мбит/с	Корпуса
PDIUSBP11A	Аналоговый USB-приемопередатчик	USB 1.1	12 и 1,5	SO-14, SSOP-14, TSSOP-14
PDIUSBD11	Интерфейс USB с последовательной шиной	USB 1.1, I2C, SMBus 1.0	1	DIP-16, SO-16
PDIUSBD12	Интерфейс USB с параллельной шиной	USB 1.1	2	SO-28, TSSOP-28
ISP1181	Интерфейс USB с параллельной шиной, 16 конечных точек	USB 1.1	12	TSSOP-48

Микросхема PDIUSBP11A

Аналоговый приемопередатчик PDIUSBP11A разработан для организации интерфейса с физическим уровнем USB. Выходы VPO и VMO этой микросхемы для механизма последовательного интерфейса (SIE, Serial Interface Enginer) управляются ведущим устройством, декодирующим стробируемые входы (VP и VM). Использование механизма последовательного интерфейса совместно с USB-приемопередатчиком позволяет разработчикам реализовать гибкое совместимое с USB устройство с использованием широкодоступных логических компонентов.

Микросхема PDIUSBD11

Микросхема PDIUSBD11 предназначена для использования в не очень сложных компьютерных мониторах, клавиатурах и других устройствах, имеющих быстродействующую последовательную шину I2C (1 Мбит/с). Малое потребление позволяет использовать функцию хаба с питанием от шины. Аппаратная и программная совместимость с прибором PDIUSBH11A обеспечивает гибкость проектирования, позволяя реализовывать периферию с хабом или без хаба.

Микросхема PDIUSB12

Микросхема PDIUSB12 с встроенной функцией USB предназначена для организации связи с микроконтроллером системы посредством быстрого (2 Мбит/с) стандартного параллельного интерфейса и поддерживает локальные DMA-пересылки. Эту микросхему можно использовать в большинстве периферийных устройств, таких как принтеры, сканеры, устройства внешней памяти и цифровые фотокамеры.

Микросхема ISP1181

Микросхема ISP1181 является высокопроизводительным прибором интерфейса полноскоростной USB с параллельной шиной, ориентированным на широкий диапазон периферии — от цифровых камер до кабельных модемов. Его высокая производительность обеспечивается: быстрым параллельным интерфейсом с производительностью 12 Мбит/с (цикл чтение-запись 85 нс), 16-разрядными параллельными портами в/в, 16 конечными точками (из которых 14 — программируемые) и объемная FIFO-память. Микросхема ISP1181 поддерживает также локальные DMA-пересылки, увеличивающие скорость обработки данных.

12.7.2. Хабы

Список микросхем хабов, выпускаемых фирмой Philips, приведен в табл. 12.5.

Таблица 12.5. Микросхемы хабов производства Philips

Название	USB	Кол-во нисхо- дящих портов	Кол-во встроен- ных функ- ций	Последо- вательный интерфейс	Напряже- ние пита- ния, ядро (I/O), В	Потребле- ние, мА	Корпус
PDIUSBH11	1.0	4	1	I2C	3,3 (5)	100	32SDIP, 32SO
PDIUSBH11A	1.0	4	3	I2C, SMBus	3,3 (5)	100	32SDIP, 32SO
PDIUSBH12	1.0	2	3	I2C, SMBus	3,3 (5)	100	28SDIP, 28SO
ISP1122	1.1	2–5 ¹	нет	I2C	от 4,0 до 5,5	95	32SDIP, 32SO, 32LQFP
ISP1123	1.1	2–5 ¹	1	I2C	от 4,0 до 5,5	95	32SDIP, 32SO, 32LQFP

Таблица 12.5 (окончание)

Название	USB	Кол-во нисходящих портов	Кол-во встроенных функций	Последовательный интерфейс	Напряжение питания, ядро (I/O), В	Потребление, мА	Корпус
ISP1130	1.1	2	3 ²	I2C	от 4,0 до 5,5	Нет данных	56SDIP, 56SSOP

1 — конфигурируется; 2 — одна для работы с клавиатурой.

Микросхема PDIUSBH11

Микросхема PDIUSBH11 является комбинированным USB-хабом (хаб со встроенной функцией), соответствующим требованиям спецификаций USB 1.0 и последовательного интерфейса I2C. Кроме того, микросхема совместима со спецификациями USB HID и Monitor Control Class. Встроенная функция PDIUSBH11 выглядит как PORT1 ведущей системы и четыре нисходящих порта, имеющие номера с 2 по 5.

Основными применениями этой микросхемы являются мониторы компьютеров и клавиатуры.

Основные характеристики:

- ☐ четыре нисходящих порта с пакетным соединением;
- ☐ встроенная функция с двумя конечными точками (управления и прерывания);
- ☐ встроенная память FIFO для хаба и встроенной функции;
- ☐ автоматическая обработка протокола;
- ☐ гибкий ведомый интерфейс I2C с производительностью 100 Кбит/с;
- ☐ обеспечение программного управления монитором;
- ☐ напряжение питания 3,3 В с устойчивыми к 5 В I/O (100 мА).

Микросхемы PDIUSBH11A и PDIUSBH12

USB-хабы PDIUSBH11A и PDIUSBH12 являются оптимизированными по функциям и стоимости USB-хабами второго поколения с четырьмя (PDIUSBH11A) и двумя (PDIUSBH12) нисходящими портами и тремя встроенными функциями (комбинированные хабы), соответствующими спецификациям USB 1.0, спецификациям последовательных интерфейсов I2C и SMBus. Они полностью совместимы со спецификациями HID и Monitor Control Class.

Микросхемы PDIUSBH11A и PDIUSBH12 обратно совместимы с аппаратными и программными средствами PDIUSBH11, что позволяет модернизировать уже выпускаемую продукцию. Они могут быть применены в мониторах компьютеров, клавиатурах и др., использующих архитектуру I2C или SMBus.

Основные возможности:

- ☐ совместимость со спецификацией USB 1.0;
- ☐ совместимость с требованиями по управлению питанием ACPI, OnNOW и USB;
- ☐ поддержка спецификаций USB HID и Monitor Control Class;
- ☐ поддержка спецификации System Management Bus Specification 1.0;
- ☐ четыре (PDIUSBH11A) и два (PDIUSBH12) нисходящих порта с по пакетным соединением и автоматическим определением скорости передачи;
- ☐ поддержка до трех встроенных функций;
- ☐ полная аппаратная реализация низкого уровня протокола USB, FIFO-память и приемопередатчики;
- ☐ автоматическая обработка USB-протокола;
- ☐ высокоскоростной интерфейс I2C (до 1 Мбит/с);
- ☐ аппаратная и программная совместимость с прибором PDIUSBH11;
- ☐ программное подсоединение к шине USB;
- ☐ мигающая индикация трафика USB нисходящего соединения;
- ☐ низкочастотный кварцевый генератор (12 МГц);
- ☐ программирование тактовой частоты;
- ☐ возможность питания по шине с очень малым собственным потреблением;
- ☐ один источник питания 3,3 В с устойчивым к напряжению 5 В I/O;
- ☐ полностью сканируемая схема с надежностью перекрытия ошибок свыше 99%, обеспечивающая высокую надежность;
- ☐ встроенная защита от воздействия электростатического электричества с уровнем свыше 8 КВ;
- ☐ поставляется в 32-выводных (PDIUSBH11A) и 28-выводных (PDIUSBH12) корпусах DIP и SO.

12.7.3. Другие микросхемы Philips

Дополнительно можно выделить микросхемы, предназначенные для работы с аудиоприборами (табл. 12.6), предназначенные для организации работы подключаемых по шине USB-устройств, таких как: микрофоны, наушники

и динамики, телефоны и автоответчики, мониторы и другие, связываемые по шине USB бытовые аудиоустройства. Эти микросхемы реализуют все функции аналоговой и цифровой обработки звука и поддерживают стандарты USB для HID- и аудиоустройств.

Таблица 12.6. USB-аудиомикросхемы фирмы Philips

Название	Назначение	Напряжение питания ядра (периферии), В	Потребление, мА	Корпуса
UDA1325	Однокристалльный стереоаудиокодек	3,3 (5)	60	SDIP42, QFP64
UDA1335H	Стереoaудиосистема записи и воспроизведения	3,3 (5)	60	QFP64
UDA1321	Стереoaудиосистема адаптивного 20-разрядного цифро-аналогового преобразования (DAC)	3,3	50	SO28, SDIP32, QFP64
UDA1331H	Стереoaудиосистема периферии аудиовоспроизведения	3,3	50	QFP64

12.8. Микросхемы Texas Instruments

Микросхемы Texas Instruments отличаются многофункциональностью (табл. 12.7), например, объединением хаба и микроконтроллера.

Таблица 12.7. Микросхемы фирмы Texas Instruments

Название	Описание
TUSB5052	USB-хаб с микроконтроллером 8052 и 2 UART и I2C
TUSB3210	Микроконтроллер семейства MCS-51 с USB и I2C, 3,3 В напряжения питания, 8 Кбайт RAM программ с загрузчиком и быстродействием 4 MIPS
TUSB2136	2-портовый хаб, микроконтроллер семейства MCS-51, интерфейсы USB 1.1 и I2C
TUSB2036	3-портовый USB-хаб, LS/FS-режимы, индивидуальное управление питанием нисходящих портов
TUSB2046B	4-портовый USB-хаб
TUSB2077A	7-портовый USB-хаб

Таблица 12.7 (окончание)

Название	Описание
TUSB5152	USB-мост с микроконтроллером 8052 и 2 UART, I2C, IEEE-1284
TUSB3410	USB-мост между USB и UART согласно спецификации USB 2.0 и OTG

Микросхема TUSB5052

TUSB5052 — контроллер для согласования USB с двумя последовательными портами и конфигурируемым хабом.

Общие особенности:

- ☐ поддержка спецификации USB 1.1;
- ☐ скорость передачи данных 12 Мбит/с через USB;
- ☐ поддержка приостановки и возобновления работы USB;
- ☐ 5 нисходящих портов (FS/LS);
- ☐ поддержка 8 входных и 8 выходных конечных точек;
- ☐ расширенный UART (см. ниже);
- ☐ интегрированный микроконтроллер 8052:
 - 256 байт ОЗУ для хранения данных;
 - 6 Кбайт ПЗУ (с загрузкой через USB и I2C);
 - 16 Кбайт ОЗУ для хранения программного кода с загрузкой от ведущего устройства или через I2C порт;
 - 2 Кбайт ОЗУ для буферов данных и блоков описания конечных точек (EDB);
 - два порта в/в (порты 1 и 3 микропроцессора 8052);
 - ведущий контроллер I2C;
- ☐ поддержка интерфейса внешнего микроконтроллера;
- ☐ встроенный четырехканальный DMA-контроллер для передачи данных (bulk) через USB/UART;
- ☐ работа от кварцевого резонатора 6 МГц. Встроенная схема генерации 48/24 МГц и 7,384615 МГц для встроенного генератора скорости;
- ☐ режим экономии (Power-down mode);
- ☐ 100-выводной корпус TQFP;
- ☐ работа от источника 3,3 В/5 В.

Расширенный UART подразумевает следующую функциональность:

- ☐ аппаратное и программное управление потоком:
 - программируемые коды Xon/Xoff;
 - программируемые режимы автоматического переключения RTS/DTR и CTS/DSR;
- ☐ автоматическое управление приемопередатчиками шины RS-485 с эхом и без эха;
- ☐ скорость передачи данных от 50 до 460,8 Кбод;
- ☐ размер символа 5, 6, 7 или 8 бит;
- ☐ управление контролем четности;
- ☐ генерация 1, 1,5 или 2 стоп-бит;
- ☐ генерация и детектирование обрыва линии;
- ☐ возможность внутреннего тестирования и обратной связи;
- ☐ функции управления линиями модема (CTS, RTS, DSR, DTR, RI и DCD);
- ☐ возможность внутренней диагностики:
 - обратная связь для контроля повреждения линии связи;
 - обрыв, паритет, переполнение, ошибка кадра.

Микросхема TUSB2136

TUSB2136 — USB-хаб с контроллером клавиатуры и микропроцессором 8052.

Основные характеристики TUSB2136:

- ☐ несколько устройств, определенных одним программным кодом, в одной микросхеме (до 16 устройств в микросхеме);
- ☐ поддержка спецификации USB 1.1;
- ☐ скорость передачи данных 1,5 и 12 Мбит/с через USB-интерфейс;
- ☐ поддержка приостановки, возобновления работы и удаленного пробуждения USB;
- ☐ встроенный двухпортовый хаб с индивидуальным управлением питанием на каждом порту;
- ☐ встроенный контроллер клавиатуры;
- ☐ встроенный микроконтроллер 8052:
 - 256 Кбайт ОЗУ для хранения данных;
 - 8 Кбайт ОЗУ для хранения кода программы, загружаемой от ведущего устройства или через I2C порт;

- 512 Кбайт ОЗУ для буферов данных и блоков описания конечных точек (EDB);
 - 4 порта в/в общего назначения (0,1, 2 и 3);
 - ведущий контроллер I2C-шины для доступа к внешним подчиненным устройствам;
 - сторожевой таймер.
- ☐ функционирование от кварцевого резонатора частотой 12 МГц;
 - ☐ встроенная схема автоподстройки частоты генерирует частоты 48/12 МГц;
 - ☐ поддержка до 4 входных и 4 выходных конечных точек;
 - ☐ режим пониженного энергопотребления;
 - ☐ питание 3,3 В;
 - ☐ 64-выводной TQFP-корпус.

Микросхема TUSB3410

TUSB3410 — мост между USB и UART. Микросхема содержит всю необходимую для подключения к хосту логику и микропроцессор семейства 8052. Микросхема может использоваться для подключения по спецификации OTG.

Общие особенности:

- ☐ поддержка спецификации USB 2.0;
- ☐ поддержка режима передачи FS;
- ☐ поддержка приостановки, возобновления работы и удаленного пробуждения USB;
- ☐ возможность питания USB от шины или от внешнего источника;
- ☐ поддерживает 3 входные и 3 выходные конечные точки;
- ☐ встроенный микроконтроллер серии 8052;
- ☐ поддержка DMA для передачи массивов данных;
- ☐ расширенный UART;
- ☐ 32-выводной LQFP-корпус.

12.9. Микросхемы Trans Dimension

Среди микросхем производства Trans Dimension можно выделить три категории микросхем:

- ☐ хост-контроллеры: TD242LP, TD243, UHP112, UHC124;
- ☐ периферийные контроллеры: TD242LP, TD243, DC2003, DC2013;
- ☐ микросхемы OTG: TD242LP, TD243.

Отметим, что многофункциональная микросхема TD242LP принадлежит всем трем категориям. Это одна из самых компактных микросхем такой категории (она имеет 64-контактный корпус Micro BGA и размер 5×5 мм), потребляющая в процессе работы ток лишь 20 мА.

Основные характеристики TD242LP:

- ❑ имеет два порта, один из которых всегда работает в режиме хоста, а второй может быть хостом, периферийным портом или OTG-портом;
- ❑ поддержка спецификаций USB 2.0 FS/LS и OTG;
- ❑ поддержка протокола USB на аппаратном уровне;
- ❑ высокая производительность;
- ❑ быстрый 16-разрядный доступ к памяти (25 Мбит/с для 16-битных данных);
- ❑ простое питание 3,3 В, специальный контакт для подключения внешнего питания 5 В;
- ❑ от 2,5 до 3,3 В для в/в и интерфейсов;
- ❑ низкое потребление;
- ❑ маленькие размеры;
- ❑ драйверная поддержка для многих операционных систем и систем реального времени (WinCE, Linux, VxWorks, Nucleus, LynxOS, QNX, pSOS, PowerTV, SMX, AMX, ThreadX, VRTX, ITRON, Symbian OS, MS DOS и многих других);
- ❑ 64-контактный корпус TF-BGA (5×5 мм) или LQFP (10×10 мм).

Несколько слов про другие микросхемы. Микросхема UHP112 является мостом PCI-USB, а UHC114 — простым и дешевым решением для встроенных хост-контроллеров. Микросхемы DC2003 и DC2013 поддерживают высокоскоростной обмен (480 Мбит/с), при этом первый контроллер имеет встроенный IDE-интерфейс, а второй — DMA-интерфейс для не-PCI-шины.

12.10. Микросхемы защиты питания

Микросхемы защиты питания USB представлены в табл. 12.8. Их подробное описание можно найти на сайтах производителей.

Таблица 12.8. Микросхемы защиты питания

Название	Описание
Микросхемы Texas Instruments (www.texasinstruments.com)	
SN75240	Микросхема защиты последовательного интерфейса USB с двумя каналами

Таблица 12.8 (окончание)

Название	Описание
Микросхемы Texas Instruments (www.texasinstruments.com)	
SN65240	Универсальная микросхема защиты последовательного интерфейса с двумя каналами
SN65220	Универсальная микросхема защиты последовательного интерфейса с одним каналом
Микросхемы STMicroelectronics (www.st.com)	
USBUF01W6	Микросхема защиты входных портов USB (2 электромагнитных высокочастотных фильтра и защита от статики)
USB6B1	Микросхема защиты портов USB, RS-485, RS-423 (2 линии в/в и питание)

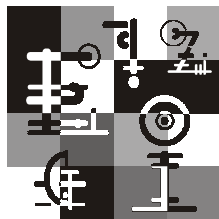
12.11. Интернет-ресурсы к этой главе

- ☐ Philips Semiconductors. Шинные устройства
 - <http://www.atel.ru/philips08.htm>
- ☐ Контроллеры USB фирмы Cypress
 - <http://sub.chipdoc.ru/html.cgi/txt/ic/Cypress/usb/index.htm?fid=26>
- ☐ Низкий по стоимости USB 2.0 трансивер от CYPRESS
 - <http://www.macro-peterburg.ru/page.phtml?parentid=62838&pageid=170968>
- ☐ AT76C711 — контроллер моста, на основе микроконтроллера AVR, между полноскоростной шиной USB и быстрым последовательным асинхронным интерфейсом
 - http://www.gaw.ru/html.cgi/txt/ic/Atmel/standart_app/usb/at76c711.htm
- ☐ AT89C5131 — 8-разрядный флэш-микроконтроллер с полноскоростным USB-портом
 - <http://www.gaw.ru/html.cgi/txt/ic/Atmel/micros/mcs51/at89c5131.htm>
- ☐ AT83C51SND1C, AT89C51SND1C — однокристалльный микроконтроллер с MP3-декодером и человеко-машинным интерфейсом
 - <http://www.gaw.ru/html.cgi/txt/ic/Atmel/micros/mcs51/at89c51snd1.htm>
- ☐ AT83C5132, AT89C5132 — USB-микроконтроллер с 64 Кбайт ПЗУ или флэш-памяти
 - <http://www.gaw.ru/html.cgi/txt/ic/Atmel/micros/mcs51/at89c5132.htm>

- ❑ 8-разрядные КМОП Flash микроконтроллеры семейства MCS-51
 - <http://www.ineltek.ru/html.cgi/txt/ic/Atmel/micros/mcs51/about.htm>
- ❑ Микроконтроллеры семейства AT8xC5132
 - <http://www.ineltek.ru/html.cgi/txt/ic/Atmel/micros/mcs51/at89c5131.htm>
- ❑ Микроконтроллеры Atmel семейства AT8xC5132
 - <http://www.gaw.ru/html.cgi/txt/ic/Atmel/micros/mcs51/at89c5132/usb.htm>
- ❑ Отладочные комплекты FTDI
 - <http://www.dlpdesign.com/usb/usb232.html>
- ❑ FTDI. Комплекты разработки
 - <http://www.premier-electric.com/products/FTDI/KITS/>
- ❑ Микроконтроллеры Cygnal со встроенными USB-контроллером и стабилизатором напряжения
 - <http://www.premier-electric.com/briefs/publications/c447ac04d46486ae.html>
- ❑ Обзор микросхем Cygnal
 - http://www.eltis.kiev.ua/cl_index.htm
- ❑ Комментарии по применению микросхем Cygnal, спецификации
 - <http://www.premier-electric.com/products/cygnal/appnotes/b4fd3b74b2616767.html>
- ❑ Использование хабов в отрасли USB (хабы Philips)
 - <http://www.gaw.ru/html.cgi/txt/ic/Philips/usb.hub.htm>
- ❑ Совместимые с USB аудиоприборы фирмы Philips
 - <http://www.gaw.ru/html.cgi/txt/ic/Philips/usb2.htm>
- ❑ Приборы USB-интерфейса фирмы Philips
 - <http://www.gaw.ru/html.cgi/txt/ic/Philips/usb.htm>
- ❑ Интерфейс USB: описание и основы устройств сопряжения
 - <http://elhelp.h1.ru/sprlist/usb.htm>
- ❑ Семейство микроконтроллеров Intel
 - <http://dfe3300.karelia.ru/koi/posob/micropr/intel.htm>
- ❑ USB-микроконтроллеры PC периферии фирмы Intel
 - <http://www.gaw.ru/html.cgi/doc/intel/usb.htm>
- ❑ TUSB2136. Разветвитель универсальной последовательной шины USB с контроллером клавиатуры

- http://www.gaw.ru/html.cgi/txt/ic/Texas_Instruments/interfaces/usb/TUSB2136.htm
- ☐ USB-микросхемы RainBow
 - <http://www.rtcs.ru/search/search.exe?q=USB>
- ☐ Микросхема TD242LP
 - <http://www.transdimension.com/www/products/semiconductors/td242lp/index.html>
- ☐ Semiconductors from TransDimension: USB Controllers
 - <http://www.transdimension.com/www/products/semiconductors/index.html>
- ☐ Motorola: USB Connectivity Reference Designs
 - <http://e-www.motorola.com/webapp/sps/site/overview.jsp?nodeId=03t3ZGNG0S7066>
- ☐ Компания Atmel разработала защищенный USB-микроконтроллер
 - <http://www.asutp.ru/?p=201810>
- ☐ Преобразователь интерфейса USB-RS232 (FTDI)
 - <http://www.cqham.ru/usb-rs232.htm>
- ☐ Универсальный программный продукт для разработки устройств с интерфейсом USB 2.0
 - <http://www.macro-peterburg.ru/page.phtml?parentid=62838&pageid=171176>

Глава 13



HID-устройство на основе Atmel AT89C5131

Лучший способ научиться программированию —
взять дизассемблер и посмотреть, как это делают другие.

Мы выбрали микропроцессор Atmel AT89C5131 по нескольким причинам. Во-первых, это недорогой, но достаточно быстродействующий процессор с широко известным ядром 8051, имеющий 6 конечных точек (см. *разд. 12.1.1*). Во-вторых, для реализации схемы требуется минимум дополнительной обвязки. Немаловажно и наличие бесплатного ассемблера, компилятора языка C, программатора и драйверов для Windows/Linux. Удобна возможность программирования процессора не по SPI, а "напрямую" по USB-каналу. Ну, и, наконец, на сегодня, это почти единственный микропроцессор, доступный без заказа.

Конечно, объем книги не позволит нам привести полное описание этого процессора и, тем более, языка C и ассемблера. Полное описание процессора и дополнительную информацию о нем можно найти на сайте Atmel (www.atmel.com). В нашей книге мы ограничимся тем минимумом информации, который потребуется для реализации простого устройства.

13.1. Структурная схема AT89C5131

Контроллер AT89C5131 содержит специальный аппаратный модуль, который позволяет микропроцессору обеспечивать обмен данными по USB-интерфейсу (рис. 13.1). Для этого необходимы опорные синхроимпульсы с частотой 48 МГц, которые вырабатываются контроллером синхронизации. Эти синхроимпульсы используются для формирования 12 МГц тактовых импульсов из принятого дифференциального потока данных USB и передачи данных на высокой скорости, соответствующей требованиям к USB-устройствам. Формирование синхроимпульсов выполняется цифровой сис-

темой ФАПЧ (DPLL, Digital Phase Locked Loop). Коэффициент деления задается битами USBCDx регистра USBCLK.

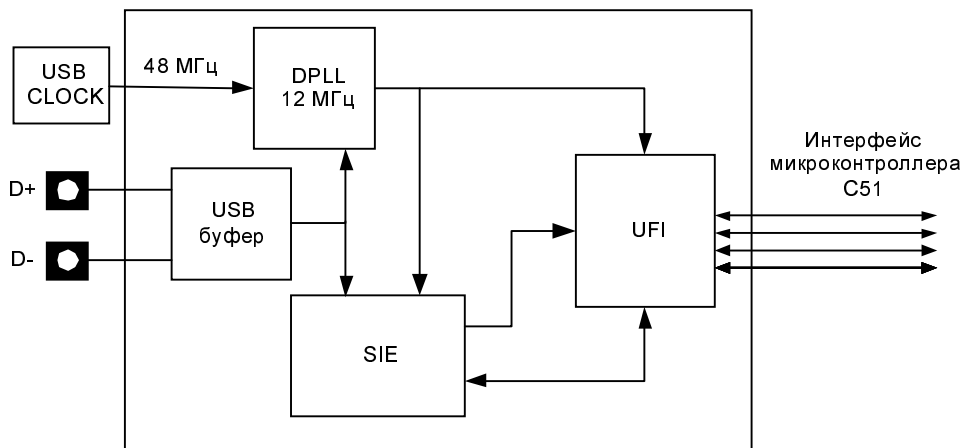


Рис. 13.1. Структурная схема USB-модуля в микропроцессоре AT89C5131

Блок последовательного интерфейса (SIE, Serial Interface Engine) выполняет следующие функции:

- ☐ NRZI-кодирование и декодирование данных;
- ☐ вставку и извлечение бита;
- ☐ формирование битов проверки на четность (CRC-кодирование и декодирование);
- ☐ автоматическое формирование сигналов ACK и NACK;
- ☐ идентификацию типа передатчика;
- ☐ контроль адресов;
- ☐ восстановление синхроимпульсов (при помощи DPLL).

Функциональный интерфейсный модуль (UFI, Universal Function Interface) обеспечивает интерфейс между микропроцессорами и SIE. Он управляет обменом на пакетном уровне с минимальными программными затратами, выполняющими запись и считывание FIFO-буфера конечной точки.

Для экономии места остальные функциональные составляющие, показанные на рис. 13.2 (источники прерываний, USB-регистры и т. д.), мы рассмотрим при описании регистров микропроцессора, необходимых нам для работы.

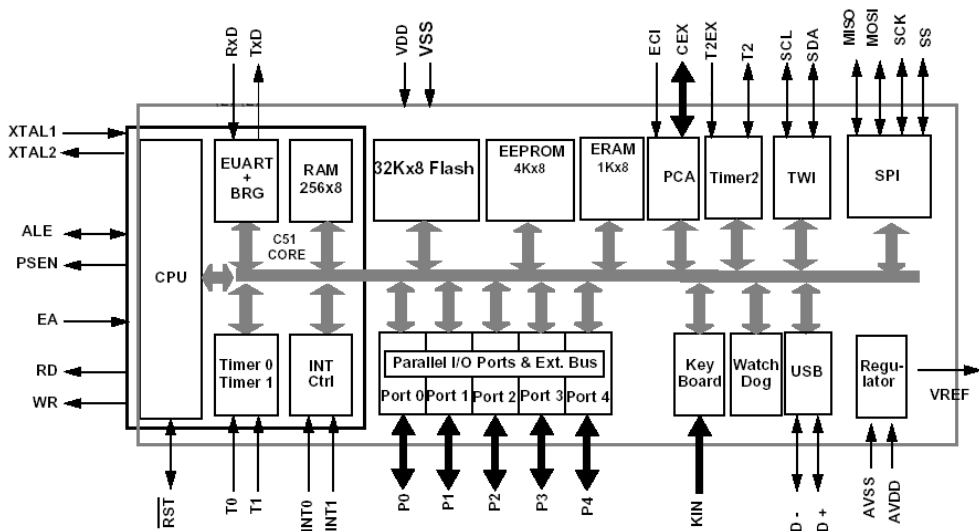


Рис. 13.2. Структурная схема AT89C5131

13.2. USB-регистры AT89C5131

Микроконтроллер AT89C5131 содержит несколько ключевых регистров, с помощью которых производится конфигурирование и обмен по интерфейсу USB. Обычные регистры, присутствующие во всех контроллерах семейства 8051, мы не описываем.

13.2.1. Регистр *USBCON*

Регистр USBCON (байт, адрес 0xBC) — основной управляющий регистр USB-модуля. После сброса регистр принимает значение 0000000b.

Регистр содержит следующие биты:

- ☐ [7] **USBE** — бит включения модуля USB. Установка бита включает USB-контроллер. Сброс бита отключает и сбрасывает USB-контроллер;
- ☐ [6] **SUSPCLK** — бит приостановки синхронизации USB. Установка бита отключает вход 48 МГц синхроимпульсов (продолжение детектирования все еще возможно). Сброс включает вход 48 МГц синхроимпульсов;
- ☐ [5] **SDRMWUP** — бит передачи удаленного пробуждения. Устанавливается для вызова внешнего прерывания USB-контроллера при удаленном пробуждении. Резюме исходящего потока передается, если только бит **RMWUPE** установлен, все USB-синхроимпульсы активизированы и USB-шина на-

ходила в состоянии приостановки (состояние SUSPEND) не менее 5 мс. Сбрасывается программно;

- ❑ [4] зарезервирован. Всегда считывается как 0. Не пытайтесь установить этот бит;
- ❑ [3] UPRSM — резюме исходящего потока (только чтение). Устанавливается аппаратно после установки бита SDRMWUP, если бит RMWUPE был также установлен. Сбрасывается аппаратно после передачи резюме исходящего потока;
- ❑ [2] RMWUPE — бит разрешения удаленного пробуждения. Устанавливается для разрешения запроса резюме исходящего потока ведущего устройства. Сбрасывается после отображения резюме исходящего потока в RSMINPR. Замечание: не устанавливайте этот бит, если у ведущего устройства для прибора не установлена функция DEVICE_REMOTE_WAKEUP;
- ❑ [1] CONFG — конфигурационный бит. Устанавливается после корректной обработки запроса SET_CONFIGURATION с ненулевым значением. Сбрасывается программно после получения запроса SET_CONFIGURATION с нулевым значением. Сбрасывается аппаратно при аппаратном сбросе или после обнаружения сброса на шине USB;
- ❑ [0] FADDEN — бит разрешения функции адресации. Устанавливается программным обеспечением прибора после успешного завершения транзакции SET_ADDRESS. В последующем он не должен сбрасываться программно. Сбрасывается аппаратно при аппаратном сбросе или после обнаружения сброса на шине USB. Когда этот бит сброшен, используется функция адресации по умолчанию (нулевая конечная точка).

Листинг 13.1 показывает описания и макросы для работы с регистром USBCON.

Листинг 13.1. Регистр USBCON

```
// Описание регистра
sfr USBCON = 0xBC;

// Описание констант для доступа к битам регистра
#define MSK_USBE      0x80    // Бит включения модуля USB
#define MSK_SUSPCLK   0x40    // Бит приостановки синхронизации USB
#define MSK_SDRMWUP   0x20    // Бит передачи удаленного пробуждения
#define MSK_DETACH    0x10    // Отключение от линии
#define MSK_UPRSM     0x08    // Бит резюме исходящего потока (r/o)
#define MSK_RMWUPE    0x04    // Бит разрешения удаленного пробуждения
#define MSK_CONFG     0x02    // Конфигурационный бит
#define MSK_FADDEN    0x01    // Бит разрешения функции адресации
```



```
// Включение/выключение модуля USB
#define Usb_enable()          (USBCON |= MSK_USBE)
#define Usb_disable()        (USBCON &= ~MSK_USBE)

// Подключение/отключение от линии
#define Usb_detach()          (USBCON |= MSK_DETACH)
#define Usb_attach()          (USBCON &= ~MSK_DETACH)

// Конфигурационный бит
#define Usb_set_CONFIG()      (USBCON |= MSK_CONFIG)
#define Usb_clear_CONFIG()    (USBCON &= ~MSK_CONFIG)

// Бит разрешения функции адресации
#define Usb_set_FADDEN()      (USBCON |= MSK_FADDEN)
#define Usb_clear_FADDEN()    (USBCON &= ~MSK_FADDEN)

// Бит приостановки синхронизации USB
#define Usb_set_suspend_clock() (USBCON |= MSK_SUSPCLK)
#define Usb_clear_suspend_clock() (USBCON &= ~MSK_SUSPCLK)
```

13.2.2. Регистр *USBADDR*

Регистр USBADDR (байт, адрес 0xC6) — регистр USB-адреса. После сброса регистр принимает значение 0000000b.

Регистр содержит следующие биты:

[7] FEN — бит активизации функции. Устанавливается для активизации функции. Программное обеспечение прибора установит этот бит после приема сброса USB и примет участие в текущем конфигурационном процессе с установленным по умолчанию адресом (FEN сбросится в 0);

[6:0] UADD6:UADD0 — биты USB-адреса. Эти биты содержат заданный по умолчанию адрес после включения питания или сброса USB-шины. Запись их состояния произойдет после принятия программным обеспечением прибора запроса SET_ADDRESS.

Листинг 13.2 показывает описания и макросы для работы с регистром USBADR.

Листинг 13.2. Регистр USBADDR

```
// Описание регистра
sfr USBADDR = 0xC6;

// Описание констант для доступа к битам регистра
#define MSK_FEN      0x80
```

```
// Конфигурирование USB-адреса
#define Usb_configure_address(x)      (USBADDR = (0x80 | x))
#define Usb_address()                (USBADDR & 0x7F)
```

13.2.3. Регистр *USBINT*

Регистр *USBINT* (байт, адрес 0xBD) — регистр флагов основных USB-прерываний. После сброса регистр принимает значение 0000000b.

Регистр содержит следующие биты:

- ❑ [7:6] зарезервированы — всегда считываются как 0. Не пытайтесь установить эти биты;
- ❑ [5] *WUPCPU* — флаг прерывания пробуждения ЦП. Устанавливается аппаратно, когда находящийся в режиме *SUSPEND* USB-контроллер перезапускается сигналом активности USB-шины (но не резюме исходящего потока). Установка этого бита вызывает USB-прерывание, когда установлен бит *EWUPCPU* в регистре *USBIEN*. Сбрасывается программно после переключения всех USB-синхроимпульсов;
- ❑ [4] *EORINT* — флаг прерывания окончания сброса. Устанавливается аппаратно при обнаружении USB-контроллером окончания сброса. Установка этого бита вызывает USB-прерывание, когда установлен бит *EEORINT* в регистре *USBIEN*. Сбрасывается программно;
- ❑ [3] *SOFINT* — флаг прерывания при обнаружении начала кадра. Устанавливается аппаратно после приема пакета начала кадра *SOF*. Установка этого бита вызывает USB-прерывание, когда установлен бит *ESOFINT* в регистре *USBIEN*. Сбрасывается программно;
- ❑ [2:1] зарезервированы. Всегда считываются как 0. Не пытайтесь установить эти биты;
- ❑ [0] *SPINT* — флаг прерывания по приостановке. Устанавливается аппаратно при обнаружении USB-приостановки (шина не занята в течение трех кадровых периодов, т. е. состояние *J* в течение 3 мс). Установка этого бита вызывает USB-прерывание, когда установлен бит *ESPIINT* в регистре *USBIEN*. Сбрасывается программно.

Листинг 13.3 показывает описания и макросы для работы с регистром *USBINT*.

Листинг 13.3. Регистр *USBINT*

```
// Описание регистра
sfr USBINT = 0xBD;
```

```
// Описание констант для доступа к битам регистра
#define MSK_SPINT      0x01 // Флаг прерывания при приостановке
#define MSK_SOFINT     0x08 // Флаг прер-я при обнаружении начала кадра
#define MSK_EORINT     0x10 // Флаг прерывания окончания сброса
#define MSK_WUPCPU     0x20 // Флаг прерывания пробуждения ЦП
// Флаг прерывания окончания сброса
#define Usb_clear_reset()      (USBINT &= ~MSK_EORINT)
#define Usb_reset()           (USBINT & MSK_EORINT)
// Флаг прерывания пробуждения ЦП
#define Usb_clear_resume()     (USBINT &= ~MSK_WUPCPU)
#define Usb_resume()          (USBINT & MSK_WUPCPU)
// Флаг прерывания при обнаружении начала кадра
#define Usb_clear_sof()       (USBINT &= ~MSK_SOFINT)
#define Usb_sof()             (USBINT & MSK_SOFINT)
// Флаг прерывания при приостановке
#define Usb_clear_suspend()   (USBINT &= ~MSK_SPINT)
#define Usb_suspend()         (USBINT & MSK_SPINT)
```

13.2.4. Регистр *USBIEN*

Регистр USBIEN (байт, адрес 0xBE) — регистр разрешений основных USB-прерываний. После сброса регистр принимает значение 0000000b.

Регистр содержит следующие биты:

- ❑ [7:6] зарезервированы — всегда считываются как 0. Не пытайтесь установить эти биты;
- ❑ [5] EWUPCPU — бит разрешения прерывания при пробуждении ЦП. Установка этого бита разрешает прерывание при пробуждении ЦП. Сброс бита запрещает прерывание при пробуждении ЦП;
- ❑ [4] EEORINT — бит разрешения прерывания по окончании сброса. Установка этого бита разрешает прерывание по окончании сброса. Сброс этого бита запрещает прерывание по окончании сброса;
- ❑ [3] ESOFINT — бит разрешения прерывания при обнаружении начала кадра. Установка этого бита разрешает прерывание при обнаружении начала кадра. Сброс этого бита запрещает прерывание при обнаружении начала кадра;
- ❑ [2:1] зарезервированы — всегда считываются как 0. Не пытайтесь установить эти биты;

- ❑ [0] ESPINT — бит разрешения прерывания при обнаружении приостановки. Установка этого бита разрешает прерывание при обнаружении приостановки. Сброс этого бита запрещает прерывание при обнаружении приостановки.

Листинг 13.4 показывает описания и макросы для работы с регистром USBIEN.

Листинг 13.4. Регистр USBIEN

```
// Описание регистра
sfr USBIEN = 0xBE;

// Описание констант для доступа к битам регистра
#define MSK_ESPINT      0x01
#define MSK_ESOFINT     0x08
#define MSK_EEORINT     0x10
#define MSK_EWUPCPU     0x20

// Макросы для доступа к битам регистра
#define Usb_enable_reset_int()      (USBIEN |= MSK_EEORINT)
#define Usb_enable_resume_int()     (USBIEN |= MSK_EWUPCPU)
#define Usb_enable_sof_int()        (USBIEN |= MSK_ESOFINT)
#define Usb_enable_suspend_int()    (USBIEN |= MSK_ESPINT)
#define Usb_disable_reset_int()     (USBIEN &= ~MSK_EEORINT)
#define Usb_disable_resume_int()    (USBIEN &= ~MSK_EWUPCPU)
#define Usb_disable_sof_int()       (USBIEN &= ~MSK_ESOFINT)
#define Usb_disable_suspend_int()   (USBIEN &= ~MSK_ESPINT)
```

13.2.5. Регистр UEPNUM

Регистр UEPNUM (байт, адрес 0xC7) — регистр номера USB конечной точки. После сброса регистр принимает значение 00000000b.

Регистр содержит следующие биты:

- ❑ [7:2] зарезервированы — всегда считываются как 0. Не пытайтесь установить эти биты;
- ❑ [1:0] EPNUM1:EPNUM0 — биты номера конечной точки. Задают номер конечной точки, к которой будет происходить обращение при считывании и записи регистров UEPSTAX, UEPDATX, UBUCTLX или UEPCONX.

Листинг 13.5 показывает описания и макросы для работы с регистром UEPNUM.

Листинг 13.5. Регистр UEPNUM

```
// Описание регистра
sfr UEPNUM = 0xC7;
// Макрос выбора номера конечной точки
#define Usb_select_ep(e)          (UEPNUM = e)
```

13.2.6. Регистр *UEPCONX*

Регистр UEPCONX (байт, адрес 0xD4) — управляющий регистр конечной USB-точки, номер которой задан в регистре UEPNUM. После сброса регистр принимает значение 00000000b.

Регистр содержит следующие биты:

- ❑ [7] EREN — бит активизации конечной точки. Установка бита включает конечную точку в соответствии с конфигурацией прибора. Нулевая конечная точка всегда активизируется после аппаратного сброса или сброса шины USB и участвует в конфигурации прибора. Сброс бита отключает конечную точку в соответствии с конфигурацией прибора;
- ❑ [6:4] зарезервированы — всегда считываются как 0. Не пытайтесь установить эти биты;
- ❑ [3] DTGL — бит изменения статуса данных (только чтение). Устанавливается аппаратно при приеме пакета DATA1. Сбрасывается аппаратно при приеме пакета DATA0;
- ❑ [2] EPDIR — бит установки направленности конечных точек. Установка бита устанавливает пакетные, прерывающие и изохронные конечные точки в режим приема. Сброс бита устанавливает пакетные, прерывающие и изохронные конечные точки в режим передачи. Бит не влияет на управляющие конечные точки;
- ❑ [1:0] ERTYPE1:ERTYPE0 — биты установки типа конечных точек (для нулевой конечной точки всегда должен быть установлен "управляющий" тип):
 - 00 — управляющая конечная точка;
 - 01 — изохронная конечная точка;
 - 10 — пакетная конечная точка;
 - 11 — прерывающая конечная точка.

Заметим, что если новый пакет данных принимается до изменения состояния бита DTGL с 0 на 1 или с 1 на 0, то возможна потеря пакета. Когда это происходит у пакетной конечной точки, встроенное программное обеспече-

ние прибора должно предположить, что ведущее устройство повторило передачу правильно принятого пакета, т. к. ведущее устройство не приняло правильного подтверждения (ACK), после этого программа должна отказаться от передачи нового пакета (конечная точка сбрасывается к DATA0 только при конфигурировании). У конечных точек, являющихся источником прерывания, переключение данных управляется так же, как и при использовании пакетных конечных точек. В управляющих конечных точках каждая транзакция SETUP начинается с DATA0, и переключение данных, в этом случае, используется, как и у пакетных конечных точек, до тех пор, пока не закончится стадия DATA (при контроле записи посылки); стадия STATUS завершает передачу DATA1 (при контроле считывания посылки). Для изохронных конечных точек встроенное программное обеспечение прибора будет восстанавливать каждый новый пакет данных и может игнорировать этот бит.

Листинг 13.6 показывает описания и макросы для работы с регистром UEPCONX.

Листинг 13.6. Регистр UEPCONX

```
// Описание регистра
sfr UEPCONX = 0xD4;

// типы конечных точек
#define CONTROL          0x00
#define ISOCHRONOUS_OUT  0x01
#define BULK_OUT         0x02
#define INTERRUPT_OUT    0x03
#define ISOCHRONOUS_IN   0x05
#define BULK_IN          0x06
#define INTERRUPT_IN     0x07

// Описание констант для доступа к битам регистра
#define MSK_EPEN         0x80 /* включение конечной точки */

// Конфигурирование типа конечной точки
#define Usb_configure_ep_type(x)    (UEPCONX = x)
```

13.2.7. Регистр UEPSTAX

Регистр UEPSTAX (байт, адрес 0xCE) — регистр управления и статуса конечной USB-точки, номер которой задан в регистре UEPNUM. После сброса регистр принимает значение 00000000b.

Регистр содержит следующие биты:

- ❑ [7] DIR — бит управления направлением конечной точки. Этот бит учитывается только тогда, когда конечной точке присвоен тип контрольной. Должен быть установлен для стадии данных. Для остальных случаев должен быть сброшен. Этот бит должен быть установлен при RXSETUP-прерывании до изменения состояния любого другого бита. Также он определяет фазу статуса (IN для проверки записи и OUT для контроля чтения). Этот бит должен быть сброшен для стадии статуса контрольной исходящей транзакции;
- ❑ [6] зарезервирован — всегда считывается как 0. Не пытайтесь установить этот бит;
- ❑ [5] STALLRQ — бит запроса остановки установления связи. Установка бита приведет к посылке ответа STALL (остановки) ведущему устройству для следующей установки связи. В противном случае этот бит должен быть сброшен;
- ❑ [4] TXRDY — управляющий бит готовности передачи пакета. Бит должен быть установлен после записи пакета в FIFO-буфер конечной точки для передачи IN-данных. Данные должны записываться в FIFO-буфер конечной точки только после сброса этого бита. Установка этого бита без записи данных в FIFO-буфер приведет к посылке пакета нулевой длины, который рекомендуется в общем случае и может потребоваться для обозначения передачи в тех случаях, когда длина последнего пакета данных равна максимальной (например, для контрольного считывания передачи). Сбрасывается аппаратно сразу после посылки пакета изохронной конечной точкой или после получения контрольной, пакетной или прерывающей конечной точкой подтверждения от ведущего устройства;
- ❑ [3] STLCRC — флаг прерывания при приостановке посылки и флаг прерывания при обнаружении CRC-ошибки:
 - для контрольных, пакетных и прерывающих конечных точек он устанавливается аппаратно после посылки при помощи STALLRQ-запроса приостановки установления связи (после этого произойдет прерывание конечной точки, если оно разрешено в регистре UEP1EN) и сбрасывается аппаратно после приема пакета SETUP;
 - для изохронных конечных точек этот бит устанавливается аппаратно при обнаружении ошибки в принятых данных (CRC-ошибка в принятых данных), после этого произойдет прерывание конечной точки, если оно разрешено в регистре UEP1EN, и сбрасывается аппаратно после приема неповрежденных данных;
- ❑ [2] RXSETUP — флаг прерывания при получении пакета SETUP. Устанавливается аппаратно после получения допустимого пакета SETUP от ведущего устройства. После этого произойдет прерывание, если оно разре-

шено в регистре UEP1EN. Сбрасывается программно после считывания SETUP-данных из FIFO-буфера конечной точки;

- ❑ [1] RXOUT — флаг прерывания при принятии исходящих данных. Устанавливается аппаратно после принятия исходящего пакета. После этого произойдет прерывание, если оно разрешено в регистре UEP1EN и все текущие исходящие пакеты отклонены до сброса этого бита. Однако в управляющих конечных точках ранее принятая SETUP-транзакция может переписать содержимое FIFO-буфера конечной точки, даже если был принят пакет данных при установленном этом флаге. Сбрасывается программно после считывания исходящих данных из FIFO-буфера конечной точки;
- ❑ [0] TXCMP — флаг прерывания по окончании передачи входящих данных. Устанавливается аппаратно после передачи входящего пакета изохронной точкой или после получения подтверждения приема (ACK) от ведущего устройства контрольной, пакетной или прерывающей конечной точкой. После этого произойдет прерывание, если оно разрешено в регистре UEP1EN. Сбрасывается программно перед следующей установкой бита TXRDY.

Листинг 13.7 показывает описания и макросы для работы с регистром UEPSTAX.

Листинг 13.7. Регистр UEPSTAX

```
// Описание регистра
sfr UEPSTAX = 0xCE;

// Описание констант для доступа к битам регистра
// Флаг прерывания по окончании передачи входящих данных
#define MSK_TXCMPL      0x01
// Флаг прерывания при принятии исходящих данных
#define MSK_RXOUT       0x02
#define MSK_RXOUTB0     0x02
// Флаг прерывания при получении пакета SETUP
#define MSK_RXSETUP      0x04
// Флаг прерывания при приостановке посылки
// Флаг прерывания при обнаружении CRC ошибки
#define MSK_STALLED      0x08
// Управляющий бит готовности передачи пакета
#define MSK_TXRDY        0x10
// Бит запроса остановки установления связи
#define MSK_STALLRQ      0x20
#define MSK_RXOUTB1      0x40
```



```
// Бит управления направлением конечной точки
#define MSK_DIR          0x80

// Макросы для работы с регистром UEPSTAX
#define Usb_set_stall_request()      (UEPSTAX |= MSK_STALLRQ)
#define Usb_clear_stall_request()    (UEPSTAX &= ~MSK_STALLRQ)
#define Usb_clear_stalled()          (UEPSTAX &= ~MSK_STALLED)
#define Usb_stall_sent()              (UEPSTAX & MSK_STALLED)
#define Usb_stall_requested()         (UEPSTAX & MSK_STALLRQ)
#define Usb_clear_rx_setup()          (UEPSTAX &= ~MSK_RXSETUP)
#define Usb_setup_received()          (UEPSTAX & MSK_RXSETUP)
#define Usb_clear_DIR()               (UEPSTAX &= ~MSK_DIR)
#define Usb_set_DIR()                 (UEPSTAX |= MSK_DIR)
#define Usb_set_tx_ready()            (UEPSTAX |= MSK_TXRDY)
#define Usb_clear_tx_ready()          (UEPSTAX &= ~MSK_TXRDY)
#define Usb_clear_tx_complete()       (UEPSTAX &= ~MSK_TXCMPL)
#define Usb_tx_complete()             (UEPSTAX & MSK_TXCMPL)
#define Usb_tx_ready()                (UEPSTAX & MSK_TXRDY)
#define Usb_clear_rx()                (UEPSTAX &= ~MSK_RXOUT)
#define Usb_clear_rx_bank0()          (UEPSTAX &= ~MSK_RXOUTB0)
#define Usb_clear_rx_bank1()          (UEPSTAX &= ~MSK_RXOUTB1)
#define Usb_rx_complete()             (UEPSTAX & MSK_RXOUTB0B1)
```

13.2.8. Регистр *UEPRST*

Регистр UEPST (байт, адрес 0xD5) — регистр сброса конечной точки. После сброса регистр принимает значение 00000000b. Для сброса FIFO-буфера конечной точки установите и сбросьте соответствующий бит перед началом любой операции до аппаратного сброса или при получении сброса USB-шины.

Регистр содержит следующие биты:

- ☐ [7] зарезервирован — всегда считываются как 0. Не пытайтесь установить этот бит;
- ☐ [6] EP6RST — сброс FIFO-буфера шестой конечной точки;
- ☐ [5] EP5RST — сброс FIFO-буфера пятой конечной точки;
- ☐ [4] EP4RST — сброс FIFO-буфера четвертой конечной точки;
- ☐ [3] EP3RST — сброс FIFO-буфера третьей конечной точки;
- ☐ [2] EP2RST — сброс FIFO-буфера второй конечной точки;
- ☐ [1] EP1RST — сброс FIFO-буфера первой конечной точки;
- ☐ [0] EP0RST — сброс FIFO-буфера нулевой конечной точки.

Листинг 13.8 показывает описания и макросы для работы с регистром UEPREST.

Листинг 13.8. Регистр UEPREST

```
// Описание регистра
sfr UEPREST = 0xD5;

// Описание констант для доступа к битам регистра
#define MSK_EP6RST    0x40
#define MSK_EP5RST    0x20
#define MSK_EP4RST    0x10
#define MSK_EP3RST    0x08
#define MSK_EP2RST    0x04
#define MSK_EP1RST    0x02
#define MSK_EP0RST    0x01

// Макрос для сброса конечной точки
#define Usb_reset_endpoint(ep_num) UEPREST=0x01<<ep_num;UEPREST=0x00
```

13.2.9. Регистр UEPINT

Регистр UEPINT (байт, адрес 0xF8) — регистр прерываний конечных USB-точек (только чтение). После сброса регистр принимает значение 00000000b.

Флаг прерывания от конечной точки устанавливается аппаратно после установки прерывания в регистре UEPSTAX и если прерывание от этой конечной точки разрешено в регистре UEP1EN. Флаг должен быть сброшен программно.

Регистр содержит следующие биты:

- ☐ [7] зарезервирован — всегда считываются как 0. Не пытайтесь установить этот бит;
- ☐ [6] EP6INT — флаг прерывания от шестой конечной точки;
- ☐ [5] EP5INT — флаг прерывания от пятой конечной точки;
- ☐ [4] EP4INT — флаг прерывания от четвертой конечной точки;
- ☐ [3] EP3INT — флаг прерывания от третьей конечной точки;
- ☐ [2] EP2INT — флаг прерывания от второй конечной точки;
- ☐ [1] EP1INT — флаг прерывания от первой конечной точки;
- ☐ [0] EP0INT — флаг прерывания от нулевой конечной точки.

Листинг 13.9 показывает описания и макросы для работы с регистром UEPREST.

Листинг 13.9. Регистр UEPREST

```
// Описание регистра
sfr UEPINT = 0xF8;
// Проверка наличия прерывания от конечных точек
#define Usb_endpoint_interrupt()    (UEPINT != 0)
```

13.2.10. Регистр UEPIEN

Регистр UEPIEN (байт, адрес 0xC2) — регистр разрешений прерываний конечных USB-точек. После сброса регистр принимает значение 00000000b.

Для разрешения прерывания от конечной точки необходимо установить соответствующий бит, а для запрещения — сбросить.

Регистр содержит следующие биты:

- ☐ [7] зарезервирован — всегда считывается как 0. Не пытайтесь установить этот бит;
- ☐ [6] EP6INTE — бит разрешения прерывания шестой конечной точки;
- ☐ [5] EP5INTE — бит разрешения прерывания пятой конечной точки;
- ☐ [4] EP4INTE — бит разрешения прерывания четвертой конечной точки;
- ☐ [3] EP3INTE — бит разрешения прерывания третьей конечной точки;
- ☐ [2] EP2INTE — бит разрешения прерывания второй конечной точки;
- ☐ [1] EP1INTE — бит разрешения прерывания первой конечной точки;
- ☐ [0] EP0INTE — бит разрешения прерывания нулевой конечной точки.

Листинг 13.10 показывает описания и макросы для работы с регистром UEPIEN.

Листинг 13.10. Регистр UEPIEN

```
// Описание регистра
sfr UEPINT = 0xC2;
// Описание констант для доступа к битам регистра
#define MSK_EP6INTE    0x40
#define MSK_EP5INTE    0x20
#define MSK_EP4INTE    0x10
```

```
#define MSK_EP3INTE    0x08
#define MSK_EP2INTE    0x04
#define MSK_EP1INTE    0x02
#define MSK_EP0INTE    0x01

// Разрешение/запрещение прерываний от конечных точек
#define Usb_enable_ep_int(e)      (UEPIEN |= (0x01 << e))
#define Usb_disable_ep_int(e)    (UEPIEN &= ~(0x01 << e))
```

13.2.11. Регистр *UEPDATA*X

Регистр UEPDATA (байт, адрес 0xCF) — регистр данных FIFO-буфера конечной USB-точки, номер которой задан в регистре UEPNUM. После сброса состояние регистра не регламентировано.

Листинг 13.11 показывает описания и макросы для работы с регистром UEPDATA.

Листинг 13.11. Регистр UEPDATA

```
// Описание регистра
sfr    UEPDATA    = 0xCF;

// Чтение данных с конечной точки
#define Usb_read_byte()      (UEPDATA)

// Передача данных в конечную точку
#define Usb_write_byte(x)    (UEPDATA = x)
```

13.2.12. Регистр *UBUCTL*X

Регистр UBUCTLX (байт, адрес 0xE2) — регистр счетчика байтов конечной USB-точки, номер которой задан в регистре UEPNUM. После сброса регистр принимает значение 00000000b.

Регистр содержит следующие биты:

- ☐ [7] зарезервирован — всегда считывается как 0. Не пытайтесь установить этот бит;
- ☐ [6:0] ВУСТ6:ВУСТ0 — счетчик байтов принятых пакетов данных. Значение этого счетчика равно количеству байтов данных, принятых после получения идентификатора (PID) данных.

Листинг 13.12 показывает описания и макросы для работы с регистром UBUCTLX.

Листинг 13.12. Регистр UBYCTLX

```
// Описание регистра
sfr UBYCTLX = 0xE2;
// Получение счетчика
#define Usb_get_nb_byte ()      (UBYCTLX)
```

13.2.13. Регистр UFNUML

Регистр UFNUML (байт, адрес 0xBA) — регистр младших битов номера USB-кадра (только чтение). После сброса регистр принимает значение 0x00. Регистр содержит младшие 8 бит 11-битного номера кадра.

13.2.14. Регистр UFNUMH

Регистр UFNUMH (байт, адрес 0xBB) — регистр старших битов номера USB-кадра (только чтение). После сброса регистр принимает значение 0x00.

Регистр содержит следующие биты:

- ❑ [7:6] зарезервированы — всегда считываются как 0. Не пытайтесь установить эти биты;
- ❑ [5] CRCOK — бит отсутствия CRC-ошибки принятого номера кадра. Устанавливается аппаратно после принятия неповрежденного номера кадра в стартовом или кадровом пакете. Обновляется после каждого принятия стартового или кадрового пакета;
- ❑ [4] CRCERR — бит наличия CRC-ошибки принятого номера кадра. Устанавливается аппаратно после принятия поврежденного номера кадра в стартовом или кадровом пакете. Обновляется после каждого принятия стартового или кадрового пакета;
- ❑ [3] зарезервирован — всегда считывается как 0. Не пытайтесь установить этот бит;
- ❑ [2:0] FNUM10:FNUM8 — номер кадра. Старшие 3 бита 11-битного номера кадра. Они доступны в последнем принятом SOF-пакете. Биты FNUM не изменяются, если принят поврежденный SOF.

13.3. Схемотехника AT89C5131

На рис. 13.3 показана схема расположения выводов AT89C5131 в 52-контактном корпусе.

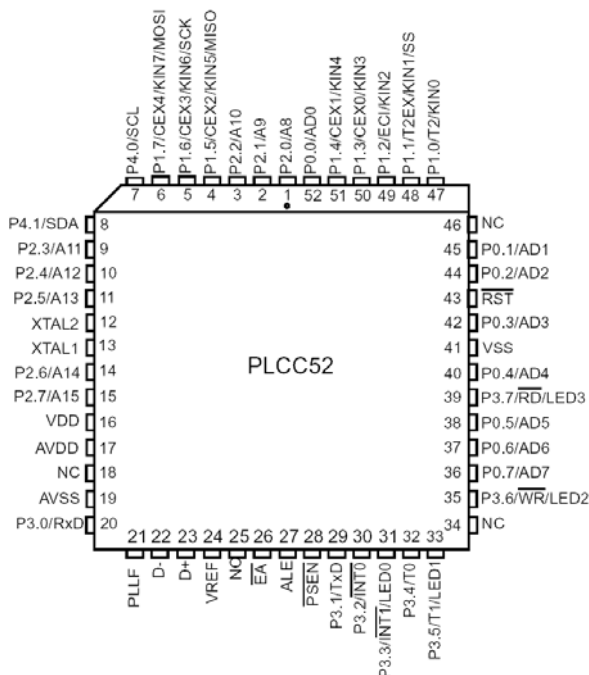


Рис. 13.3. Схема расположения выводов AT89C5131

Схема включения AT89C5131 очень проста (рис. 13.4).

Файлы схемы для PCAD можно найти на прилагаемом к книге компакт-диске.

13.4. Инструменты программирования

Для программирования микропроцессора необходимо несколько инструментов:

□ обязательные:

- компилятор языка C, ассемблер и линковщик;
- программатор;

□ необязательные:

- отладчик;
- эмулятор.

Для написания драйвера со стороны компьютера необходимы соответствующие инструменты (см. разд. 9.5).

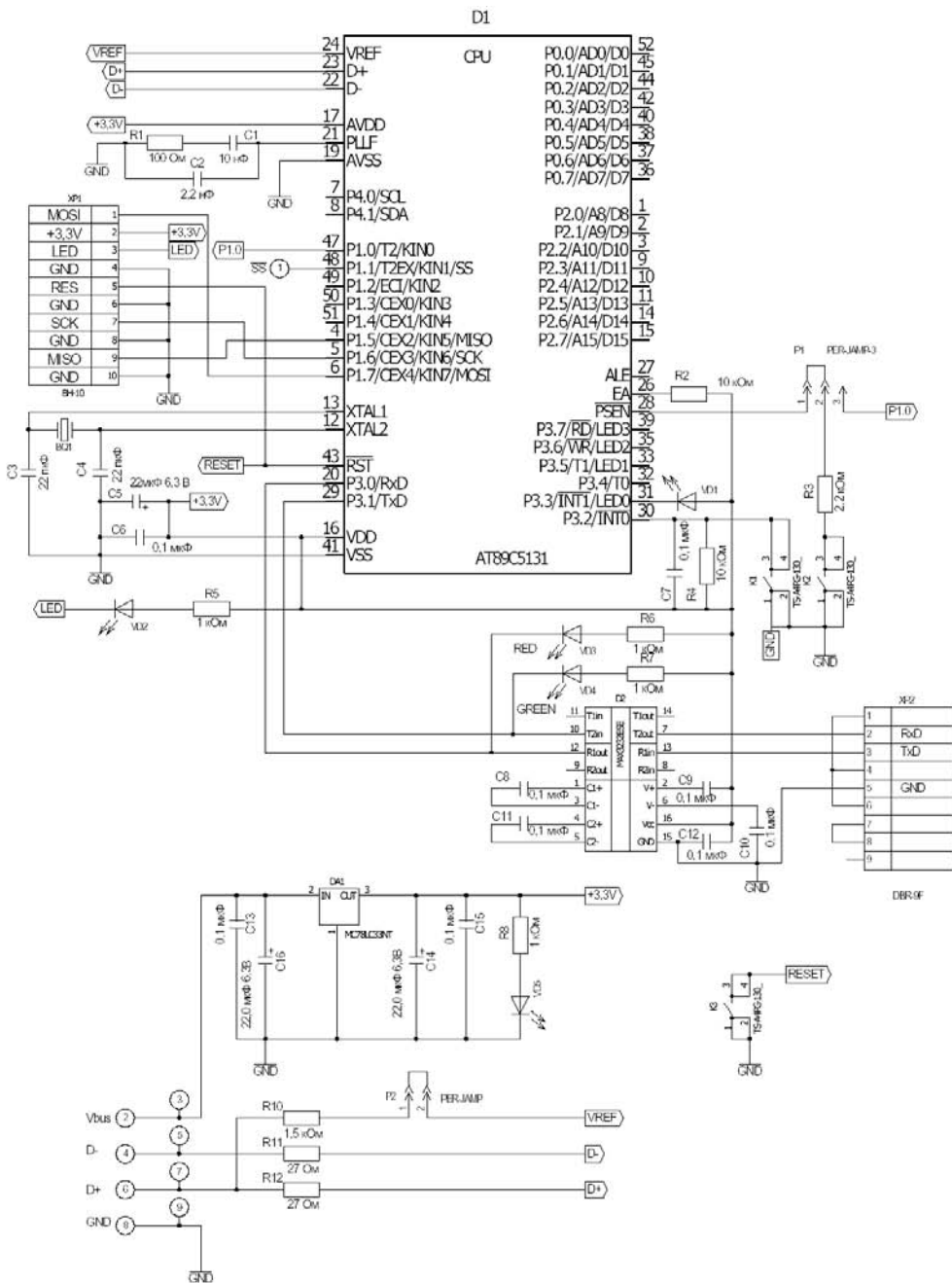


Рис. 13.4. Схема тестовой платы для AT89C5131

13.4.1. Компилятор

Для микропроцессора AT89C5131 существуют несколько компиляторов для языка C и ассемблера (IAR, Keil и др.). Заголовочные файлы (т. е. файлы описания регистров) можно загрузить с сайта Atmel:

http://www.atmel.com/dyn/resources/prod_documents/c51_include_files.zip

Компиляторы могут включать среду разработки для Windows, отладчик и другие утилиты. В нашей книге мы будем пользоваться старомодным компилятором IAR для DOS. Он не имеет красивой графической оболочки и работает из командной строки, однако нам этот "недостаток" представляется довольно удобным.

Стандартный проект IAR C состоит из нескольких файлов:

- ❑ CSTARTUP.S03 — файл кода загрузчика и описания основных сегментов;
- ❑ xxx.c51 — собственно файл программы;
- ❑ xxx.h — один или несколько заголовочных файлов;
- ❑ CL8051S.R03 — библиотека стандартных функций для 8051;
- ❑ LNK8051.XCL — командный файл настроек линковщика.

Для компиляции программы используется командный файл make.bat, код которого показан в листинге 13.13.

Листинг 13.13. Командный файл для компиляции проекта

```
@Echo off
Cls
// Удаляем файлы ошибок
If exist err.txt del err.txt
If exist errs.txt del errs.txt
// Компилируем CSTARTUP.S03
If exist LIB\CSTARTUP.r03 del LIB\CSTARTUP.r03
Bin\A8051.exe CSTARTUP.S03 LIB\LST\CSTARTUP.lst
If not exist CSTARTUP.r03 goto m_exit
copy CSTARTUP.r03 LIB\CSTARTUP.r03
del CSTARTUP.r03

// Компилируем основной файл
Bin\ICC8051.EXE test.c51 -l LIB\LST\test.LST -xDFT -a LIB\ASM\test.A51 -g
-C -ms -q -e> err.txt
If exist LIB\test.r03 del LIB\test.r03
If not exist test.r03 goto m_exit
```



```
copy test.r03 LIB\test.r03
del test.r03

// Линкуем
Bin\XLink.exe -f lnk8051.xcl > errs.txt
```

```
:m_exit
If exist err.txt type err.txt
If exist errs.txt type errs.txt
```

Как видно из листинга, проект содержит следующую структуру каталогов:

- ❑ BIN — программы компиляторов и линковщика;
- ❑ INC — стандартные заголовочные файлы и библиотеки;
- ❑ LIB — скомпилированные библиотеки для линковки;
- ❑ LIB\ASM — скомпилированный ASM-код;
- ❑ LIB\LST — листинги модулей.

Два последних каталога используются для отладки кода. Полные коды командных файлов, файлов LNK8051.XCL и CSTARTUP.S03 можно найти на прилагаемом компакт-диске.

13.4.2. Программатор

Программатор предназначен для "заливки" программы в память микропроцессора. Существует несколько способов программирования, различающихся типом соединения, например, возможно программирование по интерфейсу SPI или по интерфейсу USB. Последний способ представляется наиболее удобным, т. к. не требует дополнительных разъемов, проводов и т. п.

Программатор состоит из трех составляющих: программы программатора, драйвера и кабеля для подключения. В случае использования USB применяется обычный USB-кабель.

Нам удалось найти две программы, позволяющие производить программирование AT89C5131 по каналу USB.

Программатор FLIP

Программа FLIP (рис. 13.5) является стандартным программатором, предоставляемым Atmel для работы со своими микропроцессорами.



Рис. 13.5. Программатор FLIP

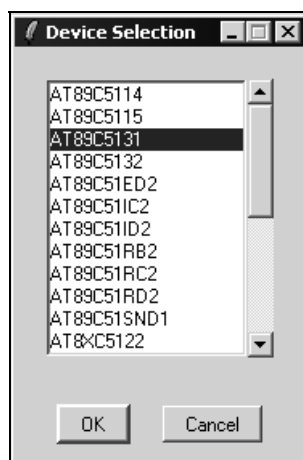


Рис. 13.6. Выбор типа микропроцессора в программе FLIP

Следует обратить внимание на версию программы. Для корректной работы с USB необходима версия не ниже 2.2.0 с обязательным обновлением AtIsp.dll от 15 марта 2004 г. (к сожалению никаких версий внутри файла не прописано, поэтому узнать, установлено ли обновление, можно только косвенно по размеру файла — 331 Кбайт).

К достоинствам программы можно отнести поддержку довольно большого числа микропроцессоров (рис. 13.6) и наличие довольно большого числа функций:

- ☐ очистка, проверка, чтение и запись EEPROM- и FLASH-памяти;
- ☐ возможность ручного редактирования буферов памяти (рис. 13.7);

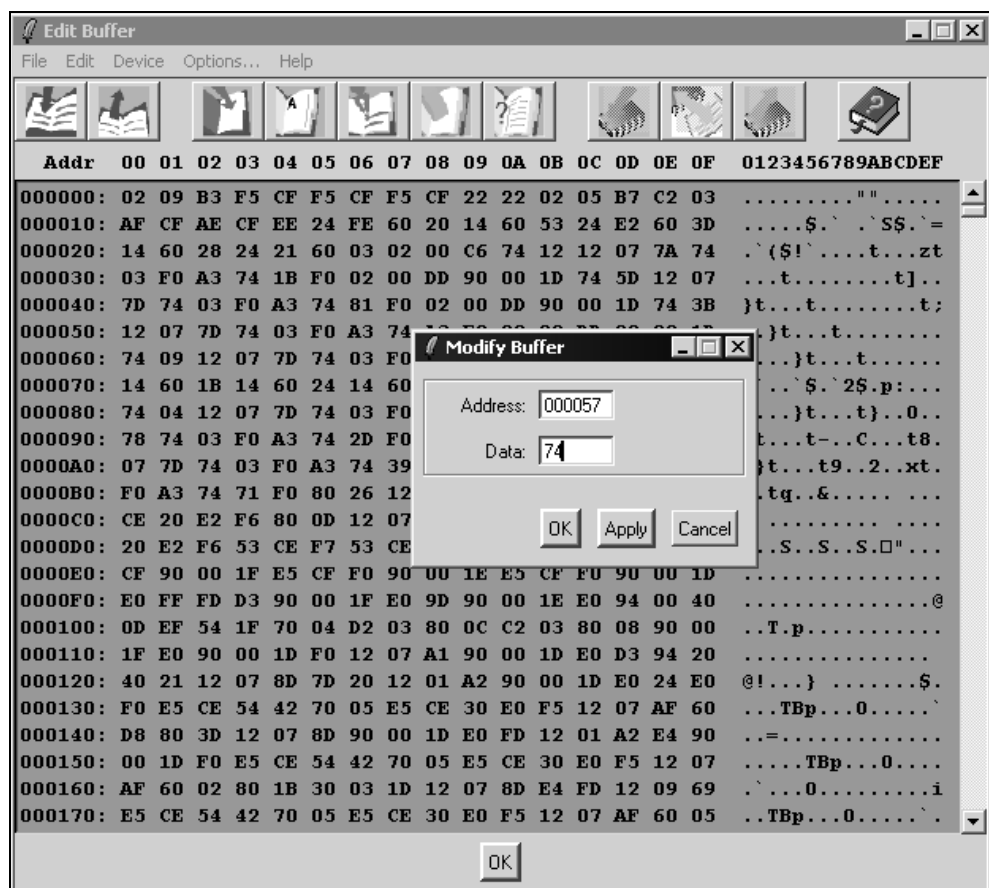


Рис. 13.7. Редактирование буферов памяти в программе FLIP

- ❑ возможность загрузки и сохранения буферов в HEX-формат;
- ❑ отслеживание изменений в последнем загруженном файле (программа предлагает перезагрузить файл заново, если он изменился).

К недостаткам мы бы отнесли некоторую запутанность интерфейса, весьма краткий файл подсказки и необходимость каждый раз открывать USB-соединение, если устройство перезапустили (а это стандартная последовательность действий: загружаем программу, проверяем, исправляем, снова загружаем).

После инсталляции программы обязательно требуется выбрать в меню пункт **Start→Programs→FLIP x.y.z→Install USB Driver**. Эта команда копирует в системный каталог необходимые файлы драйверов и INF-файлы. После подключения устройства (и запуска стартового загрузчика, *см. далее*) оно должно появиться в списке устройств в ветке Jungo (рис. 13.8). Наличие драйверов Jungo (*см. разд. 9.5*) связано с тем, что USB-драйверы для Atmel предоставлены разработчиками WinDriver (рис. 13.9).



Рис. 13.8. Новое устройство — загрузчик AT89C5131

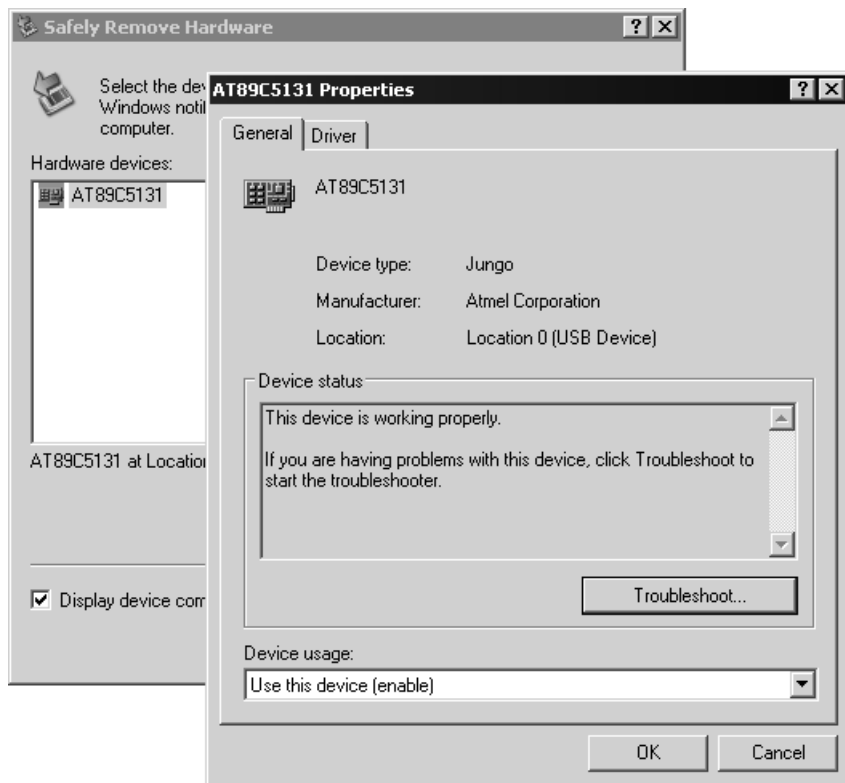


Рис. 13.9. Драйвер Jungo для USB

Программатор ER-Tronik

Еще один программатор удалось найти на немецком сайте <http://www.er-tronik.de>. Эта программа предназначена только для программирования микропроцессора AT89C5131 и только по интерфейсу USB (рис. 13.10). Конечно, узкая функциональность является существенным минусом программы, но, с другой стороны, дает, как нам кажется, и существенные плюсы. Простой и понятный интерфейс показался очень удобным:

- ❑ три кнопки в правой части представляют весь требуемый для работы набор функций (READ — прочитать, ERASE — очистить, WRITE — записать);
- ❑ совершенно очевидный редактор кода (в отличие от FLIP, где для редактирования требуется двойной щелчок на байте, и открывается отдельное окно для редактирования);
- ❑ два отдельных окна для FLASH и EEPROM не требуют переключения между типами памяти, а настройки в правой части позволяют работать как с одним типом памяти, так и с обоими одновременно;

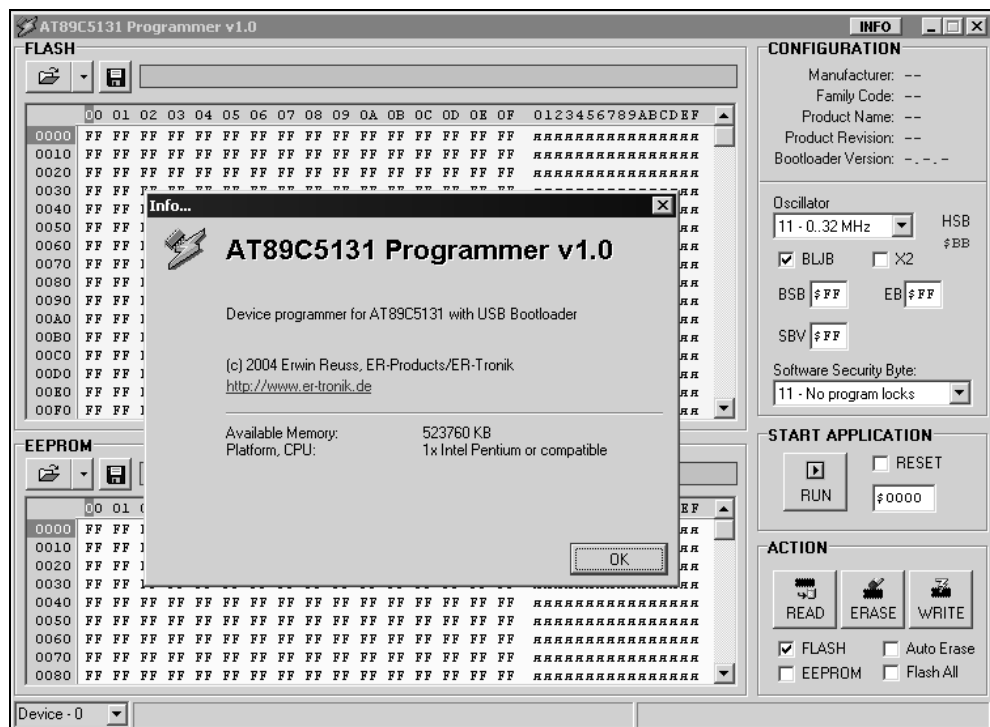


Рис. 13.10. Программатор ER-Tronik

- особенно удобным показалась возможность загружать содержимое памяти из файла, список которых сохраняется в специальном меню, позволяя очень быстро загрузить файл еще раз (рис. 13.11).

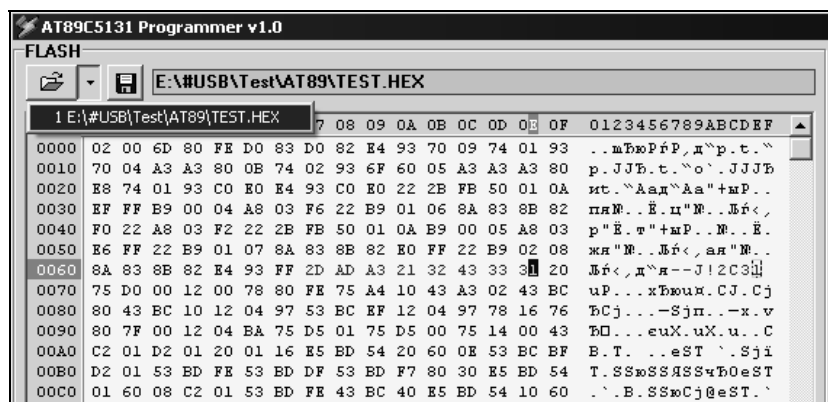


Рис. 13.11. Удобство и простота — преимущества ER-Tronik

К минусам можно отнести, пожалуй, только отсутствие документации и USB-драйвер, при установке которого выдается сообщение о возможной несовместимости с Windows XP. Впрочем, никаких проблем при работе в Windows XP обнаружено не было (рис. 13.12).

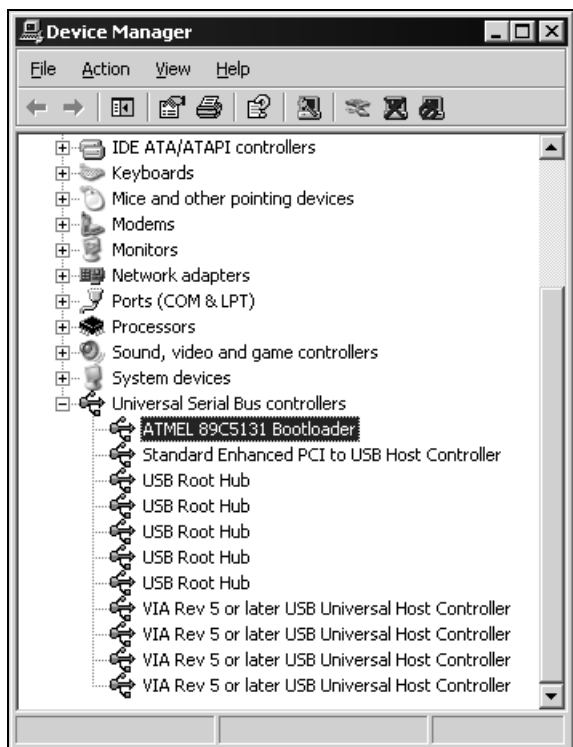


Рис. 13.12. Драйвер USB программатора ER-Tronik

Загрузка программы в процессор

Упрощенно говоря, внутри процессора существует два загрузчика: пользовательский и аппаратный (HBL, Hardware BootLoader). Пользовательский загрузчик позволяет запускать программы, записанные в память процессора, а аппаратный позволяет осуществить запись самой программы.

Очевидная последовательность действий такова: стартовать аппаратный загрузчик, записать программу в микропроцессор, стартовать записанную программу.

Для старта HBL требуется выполнить такую последовательность действий:

1. Отключить прибор от USB-шины, разомкнув перемычку P2 (линия VREF).
2. Удерживая кнопки K3 (линия Reset) и K2 (линия PSEN) подключить прибор к USB-шине, замкнув перемычку P2.
3. Отпустить кнопку K3.
4. Отпустить кнопку K2.

Если HBL успешно стартовал, Windows обнаружит новое устройство со следующими характеристиками:

- ☐ Vendor ID = 03EB;
- ☐ Product ID = 2FFD;
- ☐ при установке драйверов FLIP новое устройство будет иметь имя "Jungo AT89C5131" (см. рис. 13.8), а при установке драйверов ER-Tronik — имя "ATMEL 89C5131 Bootloader" (см. рис. 13.12).

13.5. Программа для микропроцессора

Программа, записываемая в микропроцессор, должна выполнять следующие действия:

- ☐ инициализировать USB-интерфейс и внутренние переменные;
- ☐ производить обмен по USB-шине, включая процедуру нумерации и обработку запросов от системы Plug and Play;
- ☐ производить выполнение других операций, необходимых программисту.

Для простоты понимания код нашей программы мы будем реализовывать в несколько этапов. Сначала мы сделаем простую программу, позволяющую работать с нулевой конечной точкой. Затем мы добавим обработку строковых дескрипторов, что позволит Windows более детально "рассказать" о подключенном устройстве. Затем мы добавим еще одну конечную точку для передачи данных и научимся работать с HID-интерфейсом.

13.5.1. Первая версия программы для AT89C5131

В листинге 13.14 приведена функция main (основная функция программы на языке C), реализующая основной алгоритм программы микропроцессора.

Листинг 13.14. Функция main программы для AT89C5131

```
/* описание регистров AT89C5131 */  
#include "INC\i5131.h"
```



```
/* описание констант для доступа к регистрам */
```

```
#include "INC\ext_5131.h"
```

```
/* описание типов */
```

```
#include "types.h"
```

```
/* макросы для управления регистрами */
```

```
#include "reg_macr.h"
```

```
/* описание USB-дескрипторов */
```

```
#include "usb_enum.h"
```

```
/* заголовки функций */
```

```
#include "test.h"
```

```
/* функции для работы с USB */
```

```
#include "usb_func.h"
```

```
/* ===== */
```

```
/* MAIN - основная функция программы */
```

```
/* ===== */
```

```
void main()
```

```
{
```

```
    /* выполнить инициализацию USB */
```

```
    usb_init();
```

```
    /* устройство подключено к шине */
```

```
    usb_connected = TRUE;
```

```
    /* основной цикл программы */
```

```
    for (;;) 
```

```
    {
```

```
        /* если устройство отключено от шины */
```

```
        if (!usb_connected)
```

```
        {
```

```
            /* если получен сигнал побудки */
```

```
            if (Usb_resume())
```

```
            {
```

```
                /* установить флаг активности */
```

```
                usb_connected = TRUE;
```

```
                /* сброс режима SUSPEND */
```

```
                Usb_clear_suspend_clock();
```

```
                Usb_clear_suspend();
```

```
                Usb_clear_resume();
```

```
                Usb_clear_sof();
```

```
            }
```

```
/* если устройство подключено к шине */
} else {
    /* если получен сигнал "засыпания" */
    if (Usb_suspend())
    {
        usb_connected = FALSE;
        Usb_clear_suspend();
        Usb_set_suspend_clock();
    }

    /* если получен сигнал сброса */
    if (Usb_reset())
    {
        Usb_clear_reset();
    }

    /* сигнал SOF */
    if (Usb_sof())
    {
        Usb_clear_sof();
    }

    /* обнаружено прерывание от конечной точки */
    if (Usb_endpoint_interrupt())
    {
        /* переключиться на 0 конечную точку */
        Usb_select_ep(EP_CONTROL);
        /* если получен пакет SETUP (см. разд. 4.1.1) */
        if (Usb_setup_received())
        {
            /* начать процесс нумерации */
            usb_enumeration_process();
        }
    }

    // переключиться на первую конечную точку
    // переключиться на вторую конечную точку
    //.....
}
} /* end for ;; */
}
```

В начале работы программы вызывается функция `usb_init` (ее код показан в листинге 13.15). Эта функция достаточно проста и заключается в выполнении такой последовательности действий:

1. Инициализация внутреннего генератора (его функции описаны в документации на микросхему).
2. Включение USB-интерфейса.
3. Инициирование процесса нумерации (для этого устройство отключается от шины и не ранее чем через 5 мс подключается).
4. Конфигурирование нулевой конечной точки (разумеется, нулевая точка конфигурируется как управляющая).
5. Сброс нулевой конечной точки (Reset).
6. Разрешение прерываний от нулевой конечной точки (при этом прерывание как таковое не вызывается, т. к. бит IE сброшен, но соответствующие биты в регистре `UEPINT` выставляются).
7. Инициализация других переменных.

После выполнения этих операций устройство готово к конфигурированию через канал нулевой конечной точки.

Листинг 13.15. Функция инициализации USB

```
// Описание делителей PLL в зависимости от используемого кварца
#define PLL_3MHz          0xF0
#define PLL_4MHz          0xC0
#define PLL_6MHz          0x70
#define PLL_8MHz          0x50
#define PLL_12MHz         0x30
#define PLL_16MHz         0x20
#define PLL_18MHz         0x72
#define PLL_20MHz         0xB4
#define PLL_24MHz         0x10
#define PLL_32MHz         0x21
#define PLL_40MHz         0xB9

// Функция инициализации USB
void usb_init()
{
    /* конфигурирование синхронизатора */
    PLL_set_div(PLL_24MHz); /* используется кварц 24 МГц */
    PLL_enable();           /* включение генератора */
}
```

```
/* Включение USB */
usb_enable();

/* Отключиться-подключиться для начала нумерации */
usb_detach();
delay5();
usb_attach();
delay5();

/* Конфигурирование нулевой конечной точки */
usb_configure_endpoint(0, CONTROL|MSK_EPEN);

/* Сброс нулевой конечной точки */
usb_reset_endpoint(0);

/* Разрешение прерываний от нулевой конечной точки */
usb_enable_ep_int(0);

/* Инициализация переменных */
usb_configuration_nb = 0x00; /* номер активной конфигурации */
}
```

После инициализации программа циклически выполняет обработку сигналов побудки, прерываний, сброса и т. д. При этом контролируется возникновение прерываний от конечных точек с помощью проверки регистра UEPINT. Конечно, основной цикл можно заменить на пустышку `for (;;) { }`, а всю работу перенести в обработчики прерываний, но, как нам кажется, линейная структура программы более читабельна и понятна.

Принцип обработки USB-сигналов легко понять из листинга, а мы займемся процедурой нумерации `usb_enumeration_process` (листинг 13.16).

Листинг 13.16. Процедура нумерации

```
void usb_enumeration_process()
{
    /* Выбрать нулевую конечную точку */
    usb_select_ep(0);
    /* Обработка запроса */
    usb_read_request();
}
```

Со стороны микроконтроллера процедура нумерации заключается в обработке стандартных запросов. Для формирования корректного ответа с FIFO-буфера нулевой конечной точки последовательно считываются все поля и данные запроса, согласно структуре, описанной в *разд. 4.1.1*, и в соответствии с типом запроса, выполняются действия (листинг 13.17).

Листинг 13.17. Чтение и анализ запроса

```
/* коды запросов */
#define GET_STATUS_DEVICE      0x8000
#define GET_STATUS_INTERF      0x8100
#define GET_STATUS_ENDPNT      0x8200
#define CLEAR_FEATURE_DEVICE   0x0001
#define CLEAR_FEATURE_INTERF    0x0101
#define CLEAR_FEATURE_ENDPNT   0x0201
#define SET_FEATURE_DEVICE      0x0003
#define SET_FEATURE_INTERF      0x0103
#define SET_FEATURE_ENDPNT      0x0203
#define SET_ADDRESS             0x0005
#define GET_DESCRIPTOR_DEVICE   0x8006
#define GET_DESCRIPTOR_INTERF   0x8106
#define GET_DESCRIPTOR_ENDPNT   0x8206
#define SET_DESCRIPTOR          0x0007
#define GET_CONFIGURATION       0x8008
#define SET_CONFIGURATION       0x0009
#define GET_INTERFACE           0x810A
#define SET_INTERFACE           0x010B
#define SYNCH_FRAME             0x820C
#define GET_REPORT              0xA101

// Чтение и анализ запроса
void usb_read_request()
{
    data uint16 wRequest;

    /* чтение bmRequestType */
    ((byte*)&wRequest)[0] = Usb_read_byte();
    /* чтение bRequest */
    ((byte*)&wRequest)[1] = Usb_read_byte();
```

```
switch (wRequest)
{
    case GET_STATUS_DEVICE:
        usb_get_status_device();
        break;

    case GET_STATUS_INTERF:
        usb_get_status_interface();
        break;

    case GET_STATUS_ENDPNT:
        usb_get_status_endpoint();
        break;

    case SET_ADDRESS:
        usb_set_address();
        break;

    case GET_DESCRIPTOR_DEVICE:
    case GET_DESCRIPTOR_INTERF:
    case GET_DESCRIPTOR_ENDPNT:
        usb_get_descriptor();
        break;

    case GET_CONFIGURATION:
        usb_get_configuration();
        break;

    case SET_CONFIGURATION:
        usb_set_configuration();
        break;

    default:
        STALL();
        break;
}
```

Важно отметить, что если запрос не поддерживается устройством, то для него обязательно должна быть выполнена процедура отклонения `STALL()`, как, например, в ветке `default`. Код этой процедуры приведен в листинге 13.18.

Листинг 13.18. Процедура отклонения запроса

```
void STALL(){
    Usb_clear_rx_setup();
    Usb_set_stall_request();
    while (!Usb_stall_sent());
    Usb_clear_stall_request();
    Usb_clear_stalled();
    Usb_clear_DIR();
}
```

Самый сложный обработчик — обработчик запроса `GET_DESCRIPTOR`. Для обработки этого запроса следует выполнить следующие действия:

1. Прочитать запрос, согласно структуре, описанной в *разд. 4.1.1*, определить тип запрашиваемого дескриптора (`descriptor_type`).
2. В зависимости от типа запрашиваемого дескриптора получить указатель на нужную структуру (`pbuffer`) и размер передаваемого блока данных (`data_to_transfer`).
3. Разделить передаваемые данные на блоки размером максимального пакета (константа `EP_CONTROL_LENGTH`).
4. Произвести поблочную передачу данных хосту.

Код функции, производящей обработку запроса `GET_DESCRIPTOR`, приведен в листинге 13.19.

Листинг 13.19. Обработка запроса `GET_DESCRIPTOR`

```
void usb_get_descriptor(){
    data byte    data_to_transfer;
    data uint16  wLength;
    data byte    descriptor_type;
    data byte    string_type;
    bit          zlp;
    data byte    *pbuffer;

    zlp = FALSE;
```

```
string_type      = Usb_read_byte(); /* wValue */
descriptor_type = Usb_read_byte(); /* wValue */

/* анализ типа запрашиваемого дескриптора */
switch (descriptor_type)
{
    case DEVICE: /* дескриптор устройства */
    {
        data_to_transfer = sizeof(usb_device_descriptor);
        pBuffer = &(usb_device_descriptor.bLength);
        break;
    }

    case CONFIGURATION: /* дескриптор конфигурации */
    {
        data_to_transfer = sizeof(usb_configuration);
        pBuffer = &(usb_configuration.cfg.bLength);
        break;
    }

    default: /* отклонение запросов других дескрипторов */
    {
        STALL();
        return;
    }
}

/* поле wIndex, не используется */
ACC = Usb_read_byte();
ACC = Usb_read_byte();

/* Чтение требуемого числа байт */
/* (поле wLength) */
((byte*)&wLength)[1] = Usb_read_byte();
((byte*)&wLength)[0] = Usb_read_byte();

/* Требуется больше чем есть? */
if (wLength > data_to_transfer)
{
    /* Если число байт пакета кратно размеру пакета, */
```



```

/* то пакет завершения формируем специально */
if ((data_to_transfer % EP_CONTROL_LENGTH) == 0) { zlp = TRUE; }
else { zlp = FALSE; }
}
else
{
/* посылаем только требуемое число байт */
/* иногда это меньше, чем есть реально */
/* деление на блоки не требуется */
data_to_transfer = (byte)wLength;
}

/* сброс флага приема данных */
Usb_clear_rx_setup() ;
/* переключение направления нулевой точки */
Usb_set_DIR();

/* шлем, пока хватает данных на формирование полного пакета */
/* остаток пакета досылаем потом (если надо) */
while (data_to_transfer > EP_CONTROL_LENGTH)
{
/* передача буфера максимальной длины*/
pbuffer = usb_send_ep0_packet(pbuffer, EP_CONTROL_LENGTH);
/* сдвигаем указатель */
data_to_transfer -= EP_CONTROL_LENGTH;

/* ждем завершения передачи */
while (((Usb_rx_complete())) && (!(Usb_tx_complete())));
Usb_clear_tx_complete();
if ((Usb_rx_complete())) /* если передача прервана хостом */
{
Usb_clear_tx_ready(); /* завершить передачу */
Usb_clear_rx();
return;
}
}
}

/* посылаем остаточный пакет данных */
/* если такой пакет неполный, то он является */

```

```
/* пакетом завершения передачи */
pbuffer = usb_send_ep0_packet(pbuffer, data_to_transfer);
data_to_transfer = 0;
while ((!(Usb_rx_complete())) && (!(Usb_tx_complete())));
Usb_clear_tx_complete();
if ((Usb_rx_complete())) /* если передача прервана хостом */
{
    Usb_clear_tx_ready();
    Usb_clear_rx();
    return;
}

/* при необходимости (если все пакеты полные) */
/* формируем пакет завершения специально - посылкой */
/* пакета нулевой длины */
if (zlp == TRUE)
{
    usb_send_ep0_packet(pbuffer, 0);
    while ((!(Usb_rx_complete())) && (!(Usb_tx_complete())));
    Usb_clear_tx_complete();
    if ((Usb_rx_complete())) /* если передача прервана хостом */
    {
        Usb_clear_tx_ready();
        Usb_clear_rx();
        return;
    }
}

/* дождаться завершения передачи */
while ((!(Usb_rx_complete())) && (!(Usb_setup_received())));

if (Usb_setup_received())
{
    return;
}

if (Usb_rx_complete())
{

```

```
    Usb_clear_DIR(); /* переключение направления 0 точки */
    Usb_clear_rx();
}
}
```

Собственно сама функция передачи пакета (листинг 13.20) выглядит достаточно очевидно, и, наверно, дополнительно комментирования не требует. Структуры дескрипторов мы разберем чуть позже.

Листинг 13.20. Передача пакета на нулевую конечную точку

```
byte * usb_send_ep0_packet(byte* tbuf, byte data_length)
{
    data int i;
    data byte b;

    Usb_select_ep(0);

    /* цикл передачи пакета заданной длины */
    for (i=0; i<data_length; i++)
    {
        b = *tbuf;
        Usb_write_byte(b); /* передача байта */
        tbuf++; /* следующий байт */
    }
    Usb_set_tx_ready();
    return tbuf;
}
```

Функции обработки остальных запросов выглядят значительно проще. Обработка запроса GET_CONFIGURATION сводится к передаче номера текущей конфигурации (листинг 13.21), сохраненного в переменной usb_configuration_nb. Этот номер сохраняется при обработке запроса SET_CONFIGURATION (листинг 13.22).

Листинг 13.21. Обработка запроса GET_CONFIGURATION

```
void usb_get_configuration(){
    Usb_clear_rx_setup();
    Usb_set_DIR();
}
```

```
    Usb_write_byte(usb_configuration_nb);

    END_OK();
}
```

Кроме сохранения номера текущей конфигурации в обработчике SET_CONFIGURATION должна производиться инициализация других конечных точек, если они существуют (листинг 13.22).

Листинг 13.22. Обработка запроса SET_CONFIGURATION

```
void usb_set_configuration(){
    data byte configuration_number;

    /* читать номер конфигурации */
    configuration_number = Usb_read_byte();
    Usb_clear_DIR();
    Usb_clear_rx_setup();

    /* если выбрана доступная конфигурация */
    if (configuration_number <= CONF_NB)
    {
        /* сохранить номер конфигурации */
        usb_configuration_nb = configuration_number;
    }
    else
    {
        /* ошибочный запрос — отклоняем */
        STALL();
        return;
    }

    Usb_set_tx_ready();
    while (!Usb_tx_complete());
    Usb_clear_tx_complete();

    /* Конфигурирование других конечных точек */
    /* если они существуют */
    usb_ep_init();
}
```

Запрос GET_STATUS позволяет определить состояние устройства, интерфейса или конечной точки (см. разд. 4.1.2). Состояние, возвращаемое по запросу GET_STATUS, представляет собой два байта, один из которых возвращает нужную информацию, а второй зарезервирован и равен нулю. Состояние устройства определяет его реакцию на сигнал побудки и тип питания, состояние интерфейса зарезервировано и всегда равно 0, а состояние конечной точки пока игнорируется. Код обработчиков запроса GET_STATUS показан в листинге 13.23.

Листинг 13.23. Обработка запроса GET_STATUS

```
// Обработка GET_STATUS_DEVICE
void usb_get_status_device(){
    Usb_clear_rx_setup();
    Usb_set_DIR();

    /*
        [1]= 0: устройство игнорирует сигнал побудки
        [0]= 0: устройство получает питание от шины USB
    */
    Usb_write_byte(0x00);
    /* зарезервирован, всегда 0 */
    Usb_write_byte(0x00);

    END_OK();
}

// Обработка GET_STATUS_INTERF
void usb_get_status_interface(){
    Usb_clear_rx_setup();
    Usb_set_DIR();

    /* зарезервировано, всегда 0 */
    Usb_write_byte(0x00);
    Usb_write_byte(0x00);

    END_OK();
}
```

```
// Обработка GET_STATUS_ENDPNT
void usb_get_status_endpoint(){
    STALL(); // нет конечных точек, игнорируем
}
```

С помощью запроса SET_ADDRESS хост передает устройству его адрес на шине USB. Начиная с момента получения этого запроса, устройство может отвечать хосту, только если хост обращается по этому адресу. На получение своего адреса устройство должно ответить установкой флага "устройство адресовано", как показано в листинге 13.24.

Листинг 13.24. Обработка запроса SET_ADDRESS

```
void usb_set_address(){
    data byte add;

    /* прочитать и сохранить адрес */
    add = Usb_read_byte();
    Usb_clear_rx_setup();
    Usb_set_tx_ready();

    /* выставить флаг "устройство адресовано" */
    Usb_set_FADDEN();
    while (!(Usb_tx_complete()));
    Usb_clear_tx_complete();

    /* установить полученный адрес */
    Usb_configure_address(add);
}
```

Еще одна функция, которую мы использовали для завершения передачи данных хосту, показана в листинге 13.25.

Листинг 13.25. Завершение передачи данных хосту

```
void END_OK(){
    Usb_set_tx_ready();
    while ((!(Usb_tx_complete())) || (Usb_setup_received()));
    Usb_clear_tx_complete();
    while ((!(Usb_rx_complete())) || (Usb_setup_received()));
}
```

```
    Usb_clear_rx();  
    Usb_clear_DIR();  
}
```

Теперь снова вернемся к листингу 13.19 и рассмотрим вопрос формирования дескрипторов. Листинг 13.26 показывает структуры, используемые для формирования дескриптора устройства (структура `usb_st_device_descriptor`), дескриптора конфигурации (структура `usb_st_configuration_descriptor`) и дескриптора интерфейса (структура `usb_st_interface_descriptor`). Назначение полей полностью соответствует структурам, описанным в *разд. 4.1.3*.

Листинг 13.26. Структуры дескрипторов для языка C AT89C5131

```
// Структура дескриптора устройства  
struct usb_st_device_descriptor  
{  
    byte    bLength;  
    byte    bDescriptorType;  
    uint16  bscUSB;  
    byte    bDeviceClass;  
    byte    bDeviceSubClass;  
    byte    bDeviceProtocol;  
    byte    bMaxPacketSize0;  
    uint16  idVendor;  
    uint16  idProduct;  
    uint16  bcdDevice;  
    byte    iManufacturer;  
    byte    iProduct;  
    byte    iSerialNumber;  
    byte    bNumConfigurations;  
};  
  
// Структура дескриптора конфигурации  
struct usb_st_configuration_descriptor  
{  
    byte    bLength;  
    byte    bDescriptorType;  
    uint16  wTotalLength;  
    byte    bNumInterfaces;  
    byte    bConfigurationValue;
```

```

byte    iConfiguration;
byte    bmAttributes;
byte    MaxPower;
};
// Структура дескриптора интерфейса
struct usb_st_interface_descriptor
{
    byte bLength;
    byte bDescriptorType;
    byte bInterfaceNumber;
    byte bAlternateSetting;
    byte bNumEndpoints;
    byte bInterfaceClass;
    byte bInterfaceSubClass;
    byte bInterfaceProtocol;
    byte iInterface;
};

```

Сами дескрипторы мы будем хранить прямо в коде программы, описав переменные со спецификатором code (листинг 13.27).

Важно

При описании дескрипторов следует помнить, что двухбайтовые числа должны описываться "наоборот", т. е. сначала младший байт, затем старший. Например, двухбайтовое поле длины дескриптора, размером 25 байт будет записано как 0x1900. Для удобства записи можно использовать специальный макрос wSWAP:

```
#define wSWAP(x) (((x)>>8)&0x00FF)|(((x)<<8)&0xFF00))
```

Листинг 13.27. Описание дескрипторов устройства и конфигурации

```

/* Типы дескрипторов */
#define DEVICE            0x01
#define CONFIGURATION    0x02
#define STRING            0x03
#define INTERFACE        0x04
#define ENDPOINT         0x05

/* Типы питания устройства */
#define USB_CONFIG_BUSPOWERED    0x80

```



```

#define USB_CONFIG_SELFPOWERED    0x40
#define USB_CONFIG_REMOTEWAKEUP   0x20

/* Константы, составляющие дескриптор устройства */
#define USB_SPECIFICATION          0x1001
#define USB_SPECIFICATION          0x1001
#define DEVICE_CLASS                0x02
#define DEVICE_SUB_CLASS            0
#define DEVICE_PROTOCOL             0
#define EP_CONTROL_LENGTH           32
#define VENDOR_ID                   0xEB03 /* Atmel = 03EBh */
#define PRODUCT_ID                  0x0921
#define RELEASE_NUMBER              0x0000

/* Дескриптор устройства */
code struct usb_st_device_descriptor usb_device_descriptor =
{
    sizeof(usb_device_descriptor),
    DEVICE,                /* тип дескриптора, =1          */
    USB_SPECIFICATION,     /* спецификация                */
    DEVICE_CLASS,          /* класс устройства            */
    DEVICE_SUB_CLASS,
    DEVICE_PROTOCOL,       /* протокол USB                */
    EP_CONTROL_LENGTH,     /* размер пакета 0-й точки     */
    VENDOR_ID,             /* ID производителя и устройства */
    PRODUCT_ID,
    RELEASE_NUMBER,        /* версия устройства           */
    0,                     /* дескр. строки изготовителя  */
    0,                     /* дескр. строки продукта      */
    0,                     /* дескр. строки серийного номера */
    1                      /* число конфигураций          */
};

/* Константы, составляющие дескриптор конфигурации */
#define CONF_LENGTH              wSWAP(18) /* 9+9= 18 байт */
#define CONF_NB                  1
#define CONF_ATTRIBUTES          USB_CONFIG_BUSPOWERED
#define MAX_POWER                50 /* = 100 мА */

```

```

/* Константы для дескриптора интерфейса */
#define INTERFACE0_CLASS      0xFF
#define INTERFACE0_SUB_CLASS  0x00
#define INTERFACE0_PROTOCOL   0xFF

/* Полный дескриптор конфигурации */
code struct
{
    struct usb_st_configuration_descriptor  cfg;
    struct usb_st_interface_descriptor     ifc;
}
usb_configuration =
{
/* CONFIGURATION */
    { 9,
        CONFIGURATION, /* =2 */
        CONF_LENGTH,   /* длина всех дескрипторов */
        1,              /* число интерфейсов */
        CONF_NB,        /* номер конфигурации */
        0,              /* дескр. строки конфигур. */
        CONF_ATTRIBUTES, /* атрибуты */
        MAX_POWER       /* максимальный ток */
    },
/* INTERFACE 0 */
    { 9,
        INTERFACE,      /* =4 */
        0,              /* 0-й интерфейс */
        0,              /* номер альтернативного инт. */
        0,              /* только 0-я конечная точка */
        INTERFACE0_CLASS,
        INTERFACE0_SUB_CLASS,
        INTERFACE0_PROTOCOL,
        0               /* дескр. строки интерфейса */
    }
};

```

Как уже говорилось (см. *разд. 4.1.3*), по запросу дескриптора конфигурации возвращаются дескрипторы конфигурации интерфейсов и конечных точек. В нашем случае конечных точек (кроме нулевой) нет, поэтому мы будем

возвращать только два дескриптора. Следует обратить внимание, что поле `wTotalLength` (константа `CONF_LENGTH`) содержит длину всех дескрипторов, возвращаемых по запросу дескриптора конфигурации (в нашем случае возвращаются два дескриптора, общая длина которых $9 + 9 = 18$ байт).

Теперь остается собрать весь код, откомпилировать (полный исходный код и полученный HEX-файл вы можете найти на компакт-диске) и загрузить в микропроцессор. После подачи сигнала сброса Windows должен обнаружить новое устройство (рис. 13.13) и начать поиск и установку драйверов (рис. 13.14).



Рис. 13.13. Windows обнаружила новое устройство



Рис. 13.14. Windows производит поиск и установку драйверов

13.5.2. Добавляем строковые дескрипторы

У нашего устройства есть один недостаток, от которого мы постараемся избавиться в этом разделе: при обнаружении нового устройства Windows может сообщить только общую информацию об устройстве и обобщенное название — USB Device. Конечно, нам хотелось бы, чтобы Windows сообщал, какое именно устройство найдено.

Для решения этой проблемы нам надо добавить обработку запросов строковых дескрипторов или, другими словами, выполнить следующие действия:

1. В описании дескриптора устройства `usb_device_descriptor` поля индексов дескрипторов строк изготовителя (поле `iManufacturer`), продукта (поле `iProduct`) и серийного номера (поле `iSerialNumber`) надо заменить на *уникальные* (в пределах устройства) номера дескрипторов строк (листинг 13.28).
2. Добавить описание структур самих дескрипторов и дескриптора идентификатора языка (листинг 13.29).
3. В функцию `usb_get_descriptor` добавить обработку запроса типа дескриптора `STRING` (листинг 13.30).

Листинг 13.28. Дескриптор устройства со строками идентификации

```
#define LANG_ID            0x00
#define MAN_INDEX         0x01
#define PRD_INDEX         0x02
#define SRN_INDEX         0x03

code struct usb_st_device_descriptor usb_device_descriptor =
{
    ... ..
    PRODUCT_ID,
    RELEASE_NUMBER,      /* версия устройства      */
    MAN_INDEX,           /* дескр. строки изготовителя */
    PRD_INDEX,           /* дескр. строки продукта   */
    SRN_INDEX,           /* дескр. строки серийного ном. */
    1                    /* число конфигураций      */
};
```

Листинг 13.29. Описание дескрипторов строк и языка

```

/* Макрос для создания дескриптора строки */
#define structSTRING_DESCRIPTOR(NAME, LEN) struct NAME \
    { \
        byte    bLength; \
        byte    bDescriptorType; \
        uint16  wstring[LEN]; \
    }

/* Дескриптор строки производителя */
#define USB_MANUFACTURER_NAME {'P'<<8, 'V'<<8, 'A'<<8, \
                                'S'<<8, 'o'<<8, 'f'<<8, 't'<<8}

#define USB_MN_LENGTH          7

code structSTRING_DESCRIPTOR(usb_st_manufacturer, USB_MN_LENGTH)
usb_manufacturer =
    { sizeof(usb_manufacturer), STRING, USB_MANUFACTURER_NAME };

/* Дескриптор строки продукта */
#define USB_PRODUCT_NAME      {'P'<<8, 'V'<<8, 'A'<<8, \
                                'S'<<8, 'o'<<8, 'f'<<8, 't'<<8, \
                                ' ' <<8, \
                                't'<<8, 'e'<<8, 's'<<8, 't'<<8, \
                                ' ' <<8, \
                                'b'<<8, 'o'<<8, 'a'<<8, 'r'<<8, 'd'<<8}

#define USB_PN_LENGTH          18

code structSTRING_DESCRIPTOR(usb_st_product, USB_PN_LENGTH)
usb_product =
    { sizeof(usb_product), STRING, USB_PRODUCT_NAME };

/* Дескриптор строки серийного номера */
#define USB_SERIAL_NUMBER      {'1'<<8, '.'<<8, '0'<<8, '.'<<8, '0'<<8}
#define USB_SN_LENGTH          5

code structSTRING_DESCRIPTOR(usb_st_serial_number, USB_SN_LENGTH)
usb_serial_number =
    { sizeof(usb_serial_number), STRING, USB_SERIAL_NUMBER };

/* Дескриптор строки идентификатора языка */
#define LANGUAGE_ID            0x0904

```

```
code struct STRING_DESCRIPTOR(usb_st_language_descriptor, 1)
usb_language =
{ sizeof(usb_language), STRING, LANGUAGE_ID };
```

Листинг 13.30. Обработка запроса дескриптора строки

```
void usb_get_descriptor(){
... ..
/* тип строкового дескриптора */
string_type = Usb_read_byte();
/* тип дескриптора */
descriptor_type = Usb_read_byte();

switch (descriptor_type)
{
    case GET_STATUS:
... ..
... ..
    case STRING:
    {
        switch (string_type)
        {
            case LANG_ID: /* дескриптор языка */
            {
                data_to_transfer = sizeof (usb_language);
                pBuffer = &(usb_language.bLength);
                break;
            }
            case MAN_INDEX: /* дескриптор изготовителя */
            {
                data_to_transfer = sizeof (usb_manufacturer);
                pBuffer = &(usb_manufacturer.bLength);
                break;
            }
            case PRD_INDEX: /* дескриптор продукта */
            {
                data_to_transfer = sizeof (usb_product);
                pBuffer = &(usb_product.bLength);
```

```

        break;
    }
    case SRN_INDEX: /* дескриптор серийного номера */
    {
        data_to_transfer = sizeof (usb_serial_number);
        pBuffer = &(usb_serial_number.bLength);
        break;
    }
    default: /* неподдерживаемый дескриптор */
    {
        STALL();
        return;
    }
}
break;
}

default:
... ..
}

```

Загрузив новую программу можно увидеть, что Windows выдает значительно более богатую информацию о новом устройстве (рис. 13.15, 13.16).



Рис. 13.15. Windows показывает имя обнаруженного устройства

Следует отметить, что при поиске INF-файла эти имена не используются, а поиск подходящего драйвера производится по идентификаторам продукта (Product ID) и производителя (Vendor ID).

В заключение раздела еще одно замечание. При описании строковых дескрипторов мы использовали немного странную запись

```
#define USB_PRODUCT_NAME    {'P'<<8, 'V'<<8, 'A'<<8, 'S'<<8, ...
```



Рис. 13.16. Использование имени устройства при установке драйверов

Необходимость в сдвиге каждого символа строки на 8 битов влево связана с форматом дескриптора — строка дескриптора должна состоять из символов UNICODE, т. е. двухбайтовых символов. С помощью UNICODE-строк можно передавать не только английские, но и русские строки. Проще всего сделать это, набрав нужный текст в редакторе Word и сохранив его в файл типа Кодированный текст. Затем символы этого файла можно просмотреть в любом шестнадцатеричном редакторе (например, в Far по клавише <F4> в режиме просмотра) и записать в обратном порядке следования байт.

Например, строка "Тестовая плата" будет кодироваться следующим образом:

```
0000000000:  FEFF 0422 0435 0441 | 0442 043E 0432 0430   Тестовая
0000000010:  044F 0020 043F 043B | 0430 0442 0430 000D   плата
0000000020:  000A                                     |
```

Первые два байта (0xFEFF) и последние четыре (0x000D, 0x000A) являются заголовком файла и символами перевода строки, и мы их отбрасываем, а остальные записываем в виде последовательности слов с обратным порядком байт.

```
#define USB_PRODUCT_NAME {0x2204, 0x3504, 0x4104, 0x4204, 0x3E04, 0x3204,
0x3004, 0x4F04, 0x2000, 0x3F04, 0x3B04, 0x3004, 0x4204, 0x3004}
```

Запуск программы с новым дескриптором показывает, что мы старались не зря (рис. 13.17).

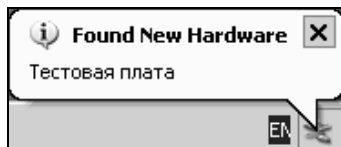


Рис. 13.17. Устройство с русским именем

13.5.3. Добавление конечных точек

Для добавления конечных точек необходимо добавить дескрипторы этих точек в дескриптор конфигурации, не забыв, кроме того, изменить поле `bNumEndpoints`.

Листинг 13.31 показывает описание структуры дескриптора конечной точки, а листинг 13.32 — дескриптор конфигурации с двумя конечными точками.

Листинг 13.31. Структура дескриптора конечной точки для AT89C5131

```
struct usb_st_endpoint_descriptor
{
    byte    bLength;
    byte    bDescriptorType;
    byte    bEndpointAddress;
    byte    bmAttributes;
    uint16  wMaxPacketSize;
    byte    bInterval;
};
```

Листинг 13.32. Дескриптор конфигурации с двумя конечными точками

```
#define EP_0_ADDRESS      (0|CONTROL)

/* первая конечная точка */
#define EP_1_CONFIG        (BULK|EP_CONFIG_IN) /* конфигурация */
#define EP_1_ADDRESS      (2|EP_DIRECT_IN)    /* адрес */
#define EP_1_ATTRIBUTES    BULK                /* атрибуты */

/* вторая конечная точка */
#define EP_2_CONFIG        (BULK|EP_CONFIG_OUT) /* конфигурация */
#define EP_2_ADDRESS      (2|EP_DIRECT_OUT)    /* адрес */
```

```

#define EP_2_ATTRIBUTES    BULK                                /* атрибуты    */
code struct
{
    struct usb_st_configuration_descriptor    cfg;
    struct usb_st_interface_descriptor       ifc;
    struct usb_st_endpoint_descriptor        ep1;
    struct usb_st_endpoint_descriptor        ep2;
}
usb_configuration =
{
/* CONFIGURATION */
    { 9,
    .....
    },
/* INTERFACE 0 */
    { 9,
    .....
        2,                /* конечные точки (кроме 0) */
    .....
    },
/* Дескриптор первой конечной точки */
    {
        7,
        ENDPOINT,          /* дескриптор ENDPOINT      */
        EP_1_ADDRESS,      /* номер конечной точки     */
        EP_1_ATTRIBUTES,   /* атрибуты конечной точки  */
        wSWAP(64),         /* максимальный размер пакета */
        0,                 /* частота опроса           */
    },
/* Дескриптор второй конечной точки */
    {
        7,
        ENDPOINT,          /* дескриптор ENDPOINT      */
        EP_2_ADDRESS,      /* номер конечной точки     */
        EP_2_ATTRIBUTES,   /* атрибуты конечной точки  */
        wSWAP(64),         /* максимальный размер пакета */
        0,                 /* частота опроса           */
    }
};

```

Мы специально скрыли уже известные нам поля дескриптора, выделив только изменившиеся. Как мы описывали в *разд. 4.1.3*, главными полями дескриптора конечной точки являются поля номера и атрибутов. Для формирования значения этих полей используются константы, показанные в листинге 13.33.

Листинг 13.33. Константы для описания конечных точек

```
/* Типы конечных точек */
#define CONTROL          0x00
#define ISOCHRONOUS      0x01
#define BULK             0x02
#define INTERRUPT        0x03
/* для вычисления номера конечной точки */
#define EP_DIRECT_OUT     0x00
#define EP_DIRECT_IN     0x80
/* для вычисления конфигурации точки */
#define EP_CONFIG_OUT    0x00
#define EP_CONFIG_IN     0x04
```

Такое несколько усложненное описание адреса конечной точки связано со структурой регистра UEPCONX, в котором для описания типа точки используются биты [1:0], а бит 2 используется для указания направления передачи, структурой поля bmAttributes, в котором для описания типа используются биты [1:0] и структурой номера конечной точки, в котором направление конечной точки задается битом 7. Так, для формирования описания конечной точки типа BULK с направлением передачи IN мы используем следующие константы:

```
// Конфигурация – биты [1:0] и 2
#define EP_1_CONFIG      (BULK | EP_CONFIG_IN)
// Номер конечной точки – биты [3:0] и 7
#define EP_1_ADDRESS     (2 | EP_DIRECT_IN)
// Атрибуты – биты [1:0]
#define EP_1_ATTRIBUTES  BULK
```

Инициализация всех конечных точек, кроме нулевой, производится после получения запроса SET_CONFIGURATION (листинг 13.34).

Листинг 13.34. Инициализация конечных точек

```
// Обработка запроса SET_CONFIGURATION
void usb_set_configuration(){
... ..
```

```

    usb_ep_init();
}
// Конфигурирование конечных точек, кроме нулевой
void usb_ep_init()
{
    // конфигурирование
    usb_configure_endpoint(1, EP_1_CONFIG | MSK_EPEN);
    // сброс
    usb_reset_endpoint(1);
    // разрешение прерываний от конечной точки
    Usb_enable_ep_int(1);

    usb_configure_endpoint(2, EP_2_CONFIG | MSK_EPEN);
    usb_reset_endpoint(2);
    Usb_enable_ep_int(2);
}

```

13.5.4. Создание HID-устройства

Теперь, когда понятно, как передавать дескрипторы и конфигурировать конечные точки, мы готовы к созданию HID-устройства (см. главу 8). Для этого необходимо выполнить следующие действия (минимальные требования к HID-устройству мы приводили в *разд. 8.1*):

- ❑ создать дескриптор репорта (см. *разд. 8.5*) либо вручную, либо с помощью одной из утилит (см. *разд. 8.7*);
- ❑ добавить в структуру `usb_configuration` HID-дескриптор (тип `usb_st_hid_descriptor`);
- ❑ добавить описание конечной точки типа `INTERRUPT IN`;
- ❑ добавить код для передачи данных в хост через конечную точку;
- ❑ в код функции `usb_get_descriptor` добавить обработку запроса дескриптора `HID` (код `0x21`) и дескриптора `REPORT` (код `0x22`).

Описание новых дескрипторов приведено в листинге 13.35.

Листинг 13.35. Дескрипторы HID-устройства

```

#define CONF_LENGTH      wSWAP(34)
#define CONF_NB          1

```

```

#define CONF_ATTRIBUTES      USB_CONFIG_BUSPOWERED
#define MAX_POWER           50 /* = 100 mA */

/* Константы для интерфейса 0 */
#define INTERFACE0_CLASS     0x03 /* HID */
#define INTERFACE0_SUB_CLASS 0xFF
#define INTERFACE0_PROTOCOL 0xFF

/* Размер дескриптора репорта */
#define SIZE_OF_REPORT       23

code struct
{
    struct usb_st_configuration_descriptor  cfg;
    struct usb_st_interface_descriptor     ifc;
    struct usb_st_hid_descriptor           hid;
    struct usb_st_endpoint_descriptor      ep1;
}
usb_configuration =
{
/* CONFIGURATION */
    { 9,
    ... ..
    },
/* INTERFACE 0 */
    { 9,
        INTERFACE,      /* =4 */
        0,              /* 0-й интерфейс */
        0,              /* номер альтернативного инт. */
        1,              /* 1 конечная точка (кроме 0) */
        INTERFACE0_CLASS,
        INTERFACE0_SUB_CLASS,
        INTERFACE0_PROTOCOL,
        0                /* дескр. строки интерфейса */
    },
/* Дескриптор HID */
    { 9,
        HID,            /* дескриптор HID */

```

```

0x0001,          /* Версия HID      */
0,              /* Числовой код страны для локал. устройств */
1,              /* Число дескрипторов репортов */
REPORT,         /* Номер дескриптора репорта */
wSWAP(SIZE_OF_REPORT) /* Размер дескриптора репорта */
},
/* Дескриптор конечной точки */
{
    7,
    ENDPOINT,      /* дескриптор ENDPOINT */
    0x81,          /* номер конечной точки */
    0x03,          /* атрибуты конечной точки */
    wSWAP(8),      /* максимальный размер пакета */
    0              /* частота опроса */
}
};

```

```

code struct{
    byte rep[SIZE_OF_REPORT];
}
HIDReport =
    /* HID Report */
    {
        0x06, 0x00, 0xff, /* USAGE_PAGE (Generic Desktop) */
        0x09, 0x01,      /* USAGE (Vendor Usage 1) */
        0xa1, 0x01,      /* COLLECTION (Application) */
        0x19, 0x01,      /* USAGE_MINIMUM (Vendor Usage 1) */
        0x29, 0x01,      /* USAGE_MAXIMUM (Vendor Usage 1) */
        0x15, 0x00,      /* LOGICAL_MINIMUM (0) */
        0x26, 0xff, 0x00, /* LOGICAL_MAXIMUM (255) */
        0x75, 0x08,      /* REPORT_SIZE (8) */
        0x95, 0x07,      /* REPORT_COUNT(7) */
        0x81, 0x02,      /* INPUT (Data,Var,Abs) */
        0xc0             /* END_COLLECTION */
    };

```

Комментарии к дескриптору репорта мы оставили такими, как они были сгенерированы утилитой HID Descriptor Tool. Передача дескрипторов HID и

REPORT ничем не отличается от передачи других дескрипторов (листинг 13.36), следует только помнить, что эти дескрипторы относятся к интерфейсу и, поэтому запрашиваются с помощью кода GET_DESCRIPTOR_INTERF.

Листинг 13.36. Передача дескрипторов HID и REPORT

```
void usb_get_descriptor(){
    ... ..
    switch (descriptor_type)
    {
        ... ..
        case HID:
        {
            data_to_transfer = sizeof(usb_configuration.hid);
            pBuffer = &(usb_configuration.hid.bLength);
            break;
        }
        case REPORT:
        {
            data_to_transfer = SIZE_OF_REPORT;
            pBuffer = &(HIDReport.rep[0]);
            break;
        }
        ... ..
    }
}
```

Скомпилируем и загрузим нашу новую программу в тестовое устройство. В Windows XP не потребуется даже перезагрузки — драйверы для HID-устройства будут установлены автоматически. Windows 98 также устанавливает драйверы автоматически, но для их старта необходимо подать устройству сигнал сброса. Рисунок 13.18 показывает обнаруженное HID-устройство в списке устройств системы.

Отметим, что устройство как бы состоит из двух частей: HID-интерфейса (устройство USB Human Interface Device, мы описали его, указав тип 3 в дескрипторе интерфейса) и пользовательского интерфейса (устройство HID-compliant device, тип которого мы описали в дескрипторе репорта). Вся работа с устройством (чтение конфигурации, прием и передача данных) производится со второй частью устройства.

Однако мало создать HID-устройство, нужно еще научиться производить обмен данными.

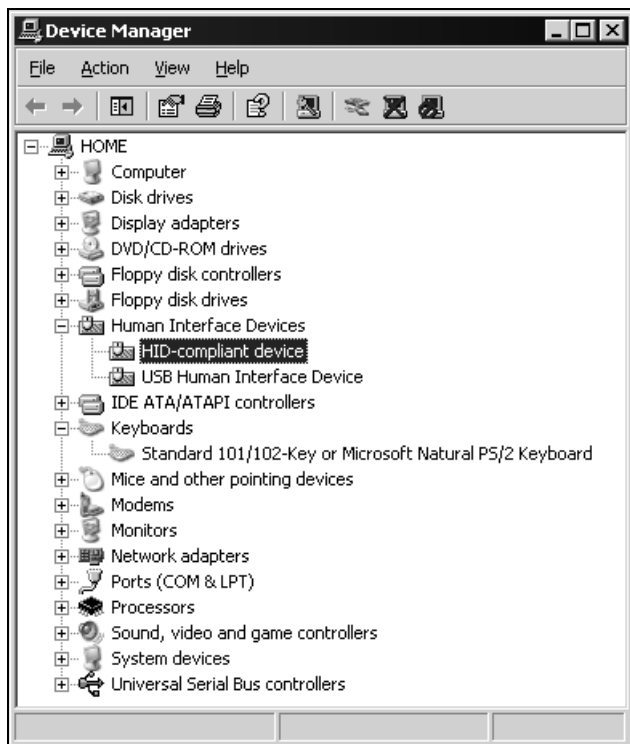


Рис. 13.18. HID-устройство обнаружено и успешно установлено

13.5.5. Обмен данными с HID-устройством

Как мы описывали в *разд. 8.2*, обмен с HID-устройством может производиться несколькими методами. Соответственно, для чтения и записи данных используются разные функции.

Использование INPUT-репортов

Репорты типа INPUT используются в случае, когда время доставки и обработки данных не критично. Чтение данных со стороны приложения производится с помощью обычных функций ReadFile или ReadFileEx (еще один способ чтения для Windows XP данных мы рассмотрим в *разд. 13.7*).

Передача данных через конечную точку состоит из двух операций: заполнения FIFO-буфера этой конечной точки и собственно передачи (листинг 13.37). Заполнять буфер можно в любом месте программы, а передача данных должна производиться при получении запроса от хоста. Для HID-

устройств важно заполнить FIFO-буфер данными, соответствующими структуре, декларированной в дескрипторе репорта (в дескрипторе конечной точки описывается максимальное число байт передачи, а в дескрипторе репорта — структура самих данных).

Листинг 13.37. Передача данных через конечную точку

```
void main()
{
    ... ..
    /* основной цикл программы */
    for (;;)
    {
        Fill_EP1_FIFO();
        ... ..
        /* обнаружено прерывание от конечной точки */
        if (Usb_endpoint_interrupt())
        {
            ... ..
            /* переключиться на 1 конечную точку */
            Usb_select_ep(1);
            if (Usb_tx_complete()){
                Usb_clear_tx_complete();
                end_point1_ready = FALSE;
            }
        }
    }
} /* end for ;; */
}

void Fill_EP1_FIFO()
{
    /* устройство не готово к передаче данных */
    if ((usb_configuration_nb == 0) || (Usb_suspend()))
        return;

    if (!end_point1_ready) /* предыдущий пакет отправлен */
    {
        Usb_select_ep(1);
```

```
    Usb_write_byte(0x01);
    Usb_write_byte(0x02);
    Usb_write_byte(point1_state);
    Usb_write_byte(0x04);
    Usb_write_byte(0x05);
    Usb_write_byte(0x06);
    Usb_write_byte(0x07);
    Usb_set_tx_ready();

    point1_state++;
    end_point1_ready = TRUE;
    return;
}
}
```

Флаг `end_point1_ready` позволяет заполнять буфер новыми данными только после передачи предыдущего пакета. Переменная `point1_state` является простым счетчиком отправленных пакетов, введенная исключительно для того, чтобы данные не были статическими и хоть немного менялись.

Важно понимать еще одну тонкость работы HID-устройства в случае использования INPUT-репортов. Добавим в функцию передачи данных переключение светодиодного индикатора, подключенного к контакту P3.3:

```
Usb_select_ep(1);
if (Usb_tx_complete()){
    Usb_clear_tx_complete();
    end_point1_ready = FALSE;
    P3.3=p_test; p_test=!p_test;
}
```

Подключив устройство, мы увидим, что обмен данными происходит постоянно, хотя мы со стороны Windows никакие данные не запрашивали! Запустим USB Monitor. Легко увидеть, что HID-драйвер Windows читает данные с устройства каждый раз, когда устройство сообщает о появлении нового пакета данных, т. е. каждый цикл программы. Что же произойдет, если данных не будет? Добавим еще одну строку, позволяющую управлять появлением новых пакетов:

```
void Fill_EP1_FIFO()
{
    /* устройство не готово к передаче данных */
}
```

```

if ((usb_configuration_nb == 0) || (Usb_suspend()))
    return;
if (P3.2 == 0)
    return;
... ..
}

```

Теперь, нажимая кнопку, соответствующую контакту P3.2, мы будем прерывать генерацию новых пакетов. Разумеется, при отсутствии новых пакетов HID-драйвер не производит чтения данных с устройства, однако, как мы покажем в следующем разделе, функция чтения данных `ReadFile` "зависает" до тех пор, пока не сможет получить требуемый пакет.

Таким образом, при использовании INPUT-репортов предпочтительнее генерировать пакеты данных всегда (еще лучше делать это не в цикле программы, а по прерыванию), а при отсутствии новых данных либо посылать предыдущие, либо специальным флагом отмечать, что данные отсутствуют (в Windows XP можно использовать другой механизм чтения данных, см. разд. 13.7).

Оптимальный вариант — выделение чтения данных в отдельный поток и использование асинхронных функций (листинг 13.38)¹.

Листинг 13.38. Использование асинхронного чтения

```

ReadOL    : TOverLapped; {структура для асинхронного чтения}
ReadFlag  : Boolean;

... ..

HidHandle:= CreateFile(PChar(@HidName[1]),
    GENERIC_READ or GENERIC_WRITE,
    FILE_SHARE_READ or FILE_SHARE_WRITE,
    nil,
    OPEN_EXISTING,
    FILE_FLAG_OVERLAPPED, // открыть для асинхронного доступа
    0
);

... ..

// инициализация буфера
FillChar(InputReport, SizeOf(InputReport), #0);

```

¹ Более подробно о создании потоков асинхронного чтения можно прочитать в [1].

```

// создание события для асинхронного чтения
FillChar(ReadOL, SizeOf(ReadOL), 0);
ReadOL.hEvent:= CreateEvent(nil, True, True, nil);

ReadFlag:= False;
// если не удалось прочитать сразу - операция еще длится,
// или произошла ошибка
If not ReadFile(HidHandle, InputReport,
    Capabilities.InputReportByteLength, BytesRead, @ReadOL) then begin
    // если операция еще длится, то ждем
    If GetLastError() = ERROR_IO_PENDING then begin
        // ждем 2 сек, если не получилось, выходим
        If WaitForSingleObject(ReadOL.hEvent, 2000) = WAIT_OBJECT_0 then
            ReadFlag:= True;
        End else begin
            lbLog.Items.Add(' Ошибка ReadFile');
        End;
    End else begin
        ReadFlag:= True; // прочитали данные сразу
    End;

    // если данные прочитаны...
    If ReadFlag then begin
        ... ..
    End;
    ... ..
    // Освободить дескрипторы
    CloseHandle(ReadOL.hEvent);
    CloseHandle(HidHandle);

```

Программу чтения данных мы будем разбирать в *разд. 13.6*.

Использование FEATURE-репортов

Второй способ передачи данных, который мы рассмотрим, заключается в использовании репортов типа FEATURE. Репорты этого типа удобно использовать, когда важно время доставки данных.

Для описания репорта типа FEATURE достаточно изменить описание дескриптора REPORT (листинг 13.39).

Листинг 13.39. Дескриптор FEATURE REPORT

```
code struct{
    byte  rep[SIZE_OF_REPORT];
}
HIDReport =
    /* HID Report */
    {
        0x06, 0x00, 0xff, /* USAGE_PAGE (Generic Desktop) */
        0x09, 0x01,      /* USAGE (Vendor Usage 1) */
        0xa1, 0x01,      /* COLLECTION (Application) */
        0x19, 0x01,      /*  USAGE_MINIMUM (Vendor Usage 1) */
        0x29, 0x01,      /*  USAGE_MAXIMUM (Vendor Usage 1) */
        0x15, 0x00,      /*  LOGICAL_MINIMUM (0) */
        0x26, 0xff, 0x00, /*  LOGICAL_MAXIMUM (255) */
        0x75, 0x08,      /*  REPORT_SIZE (8) */
        0x95, 0x07,      /*  REPORT_COUNT(7) */
        0xb1, 0x02,      /*  FEATURE (Data, Var, Abs) */
        0xc0             /*  END_COLLECTION */
    };
```

Как видно, FEATURE-репорт отличается от INPUT одним байтом, однако, способ его обработки отличается существенно.

Обработка FEATURE-запросов производится при получении запроса GET_REPORT (код запроса 0xA101). Код обработчика показан в листинге 13.40.

Листинг 13.40. Обработка запроса GET_REPORT

```
void usb_read_request()
{
    data uint16 wRequest;

    /* чтение bmRequestType */
    ((byte*)&wRequest)[0] = Usb_read_byte();
    /* чтение bRequest */
    ((byte*)&wRequest)[1] = Usb_read_byte();

    switch (wRequest)
    {
        ... .. .
```

```
case GET_REPORT:
    hid_get_report();
    break;

default:
    STALL();
    break;
}
}

void hid_get_report()
{
    // если конфигурация еще не установлена, не отвечаем
    if (usb_configuration_nb == 0){
        STALL();
        return;
    }

    // индикатор получения запроса (светодиод)
    P3.3=p_test; p_test=!p_test;

    // переключение на передачу
    Usb_clear_rx_setup();
    Usb_set_DIR();
    // передача пакета
    Usb_write_byte(0x11);
    Usb_write_byte(0x12);
    Usb_write_byte(point1_state);
    Usb_write_byte(0x14);
    Usb_write_byte(0x15);
    Usb_write_byte(0x16);
    Usb_write_byte(0x17);
    point1_state++;

    // подтверждение передачи и переключение на прием
    END_OK();
}
```

Программу чтения FEATURE-репортов мы будем рассматривать в следующем разделе, объединив ее с программой чтения других репортов. В заключение отметим, что чтение данных производится не постоянно, а по запросу со стороны хоста, поэтому индикатор P3.3 будет изменять свое состояние после каждого чтения. Однако, как и в случае с INPUT-репортами, если устройство не ответит на запрос, программа зависнет до получения данных или ответа STALL.

13.6. Чтение репортов в Windows

В этом разделе мы создадим небольшую программу для чтения данных с HID-устройства, воспользовавшись информацией *главы 8*. Мы специально не приводили код этой программы в предыдущих разделах, хотя и подразумевали наличие такой программы. Мы объединим программу чтения разных типов репортов, сделав одну универсальную программу.

Для чтения данных с устройства нужно получить его дескриптор с помощью функции `CreateFile`. В отличие от, например, последовательных портов HID-устройства не имеют простых и понятных имен (`COM1`, `COM2` и т. д.). Для получения имени устройства нужно воспользоваться функциями модуля `SetupApi` и выполнить следующие действия:

1. Получить GUID класса HID с помощью вызова `HidD_GetHidGuid`.
2. Получить дескриптор PnP для HID-класса с помощью вызова `SetupDiGetClassDevs`.
3. Произвести цикл по всем устройствам HID-класса, вызывая `SetupDiGetDeviceInterfaceDetail`, и найти нужное устройство (для простоты мы будем производить чтение данных с каждого найденного устройства).

Соответствующий код показан в листинге 13.41.

Листинг 13.41. Поиск HID-устройств

```
// Отображение списка HID-устройств и их свойств
procedure TForm1.Button1Click(Sender: TObject);
var HidGuid : TGuid; PnPHandle : HDevInfo;
    DevData: TSPDevInfoData;
    DeviceInterfaceData: TSPDeviceInterfaceData;
    FunctionClassDeviceData: PSPDeviceInterfaceDetailData;
    Success: LongBool;
    DevIndex: DWORD;
```

```

    BytesReturned: DWORD;
    HidName : String;
begin
    // Очистить лог
    lbLog.Items.Clear;
    // Получить GUID для класса HID
    HidD_GetHidGuid(HidGuid);
    // Получаем дескриптор PnP для HID-класса
    PnPHandle:= SetupDiGetClassDevs(@HidGuid, nil, 0,
        DIGCF_PRESENT or DIGCF_DEVICEINTERFACE);
    // Если ошибка, то выходим
    If PnPHandle = Pointer(INVALID_HANDLE_VALUE) then Exit;

Try
    // Индекс текущего устройства
    DevIndex := 0;
    // Цикл по всем устройствам в HID-классе
    Repeat
        DeviceInterfaceData.cbSize := SizeOf(TSPDeviceInterfaceData);
        // Получить информацию об интерфейсах устройства номер DevIndex
        Success := SetupDiEnumDeviceInterfaces(PnPHandle, nil,
            HidGuid, DevIndex, DeviceInterfaceData);
    If Success then begin
        DevData.cbSize := SizeOf(DevData);
        BytesReturned := 0;
        // Получаем подробности об устройстве с
        // интерфейсом DeviceInterfaceData
        // Сначала вызываем с нулевым размером буфера,
        // получаем размер необходимого буфера, потом
        // вызываем повторно, сформировав правильный буфер
        SetupDiGetDeviceInterfaceDetail(PnPHandle, @DeviceInterfaceData,
            nil, 0, BytesReturned, @DevData);
        If (BytesReturned <> 0) and
            (GetLastError = ERROR_INSUFFICIENT_BUFFER) then begin
            // Создаем буфер
            FunctionClassDeviceData := AllocMem(BytesReturned);
            FunctionClassDeviceData.cbSize := 5;
            // Получаем информацию
            If SetupDiGetDeviceInterfaceDetail(PnPHandle, @DeviceInterfaceData,

```



```
FunctionClassDeviceData, BytesReturned,
BytesReturned, @DevData) then begin
// Отобразить PnP-имя устройства
HidName:= StrPas(PChar(@FunctionClassDeviceData.DevicePath));
lbLog.Items.Add(HidName);
// Чтение репортов
ReadHIDReports(HidName);
End;
// Освободить буфер
FreeMem(FunctionClassDeviceData);
End;
End;
// Следующее устройство
Inc(DevIndex);
Until not Success;
Finally
SetupDiDestroyDeviceInfoList(PnPHandle);
End;
end;
```

Компонент `lbLog` типа `TListBox` служит для отображения сообщений.

Получив имя устройства (переменная `HidName`), мы вызываем функцию `ReadHIDReports(HidName)`, задачей которой является чтение данных с найденного устройства.

Так как мы ничего не знаем про свойства найденного устройства, то сначала пробуем открыть устройство для чтения и записи, а если это не получается, то просто для чтения. С успешно открытым устройством можно работать. Вначале мы получим атрибуты устройства, вызвав функцию `HidD_GetAttributes`. Это даст нам информацию, записанную в дескрипторе устройства: идентификаторы производителя (Vendor ID), устройства (Product ID) и версию программы в устройстве (Version Number). Для получения строковых значений идентификаторов (если они описаны в дескрипторе) можно воспользоваться функциями `HidD_GetManufacturerString`, `HidD_GetProductString` и `HidD_GetSerialNumberString`. В принципе, именно в этот момент времени можно произвести проверку свойств устройства и, если его идентификаторы не совпадают с нужными, перейти к следующему устройству.

Для чтения данных мы должны определить, какие репорты поддерживает устройство. Для этого мы используем функцию `HidP_GetCaps`, возвращающую размеры `INPUT`, `OUTPUT` и `FEATURE`-репортов. Если размер

соответствующего репорта равен нулю, устройство не поддерживает репорты этого типа. Перед вызовом `HidP_GetCaps` требуется вызов `HidD_GetPreparedData`, создающий специальный буфер типа `THIDPPreparedData`.

Для чтения INPUT-репорта вызывается обычная функция `ReadFile` (в Windows 2000/XP можно использовать `ReadFileEx`). Для чтения FEATURE-репортов используется функция `HidD_GetFeature`.

Полный код функции чтения репортов показан в листинге 13.42.

Листинг 13.42. Чтение репортов

```
// Чтение репортов
procedure TForm1.ReadHIDReports(HidName : String);
var HidHandle : THandle;
    CanReadWriteAccess : Boolean;
    Attributes : THIDAttributes;
    NumInputBuffers : Integer;
    Buffer : array [0..253] of WideChar;
    InputReport : Array [0..255] of Byte;
    i : Integer; S : String;
    BytesRead : Cardinal;
    Capabilities : HIDP_CAPS;
    PreparedData: PHIDPPreparedData;
begin
    // Сначала пробуем открыть устройство
    // в режиме r/w
    lbLog.Items.Add(' Пробуем открыть HID-устройство...');
    HidHandle:= CreateFile(PChar(@HidName[1]),
        GENERIC_READ or GENERIC_WRITE,
        FILE_SHARE_READ or FILE_SHARE_WRITE,
        nil,
        OPEN_EXISTING,
        0,
        0
    );

    // Устройство поддерживает запись?
    CanReadWriteAccess:= HidHandle <> INVALID_HANDLE_VALUE;
```

```
// Если не получилось, пробуем открыть
// в режиме только чтения данных
If not CanReadWriteAccess then begin
    HidHandle:= CreateFile(PChar(@HidName[1]),
        0,
        FILE_SHARE_READ or FILE_SHARE_WRITE,
        nil,
        OPEN_EXISTING, 0, 0
    );
End else begin
    lbLog.Items.Add(' Устройство открыто в режиме read/write');
End;

// Если не получилось - ошибка и выход
If HidHandle = INVALID_HANDLE_VALUE then begin
    lbLog.Items.Add(' Ошибка открытия устройства');
    Exit;
End else begin
    lbLog.Items.Add(' Устройство открыто в режиме read only!');
End;

// Получаем атрибуты устройства
Attributes.Size := SizeOf(THIDDAttributes);
If HidD_GetAttributes(HidHandle, Attributes) then begin
    lbLog.Items.Add(Format(
        ' VendorID=%d, ProductID=%d, VersionNumber=%d',
        [Attributes.VendorID, Attributes.ProductID,
        Attributes.VersionNumber]
    ));
End else begin
    lbLog.Items.Add(' Ошибка HidD_GetAttributes');
End;

// Получаем число буферов
If HidD_GetNumInputBuffers(HidHandle, NumInputBuffers) then begin
    lbLog.Items.Add(Format(
        ' Число входных буферов=%d', [NumInputBuffers]));
End else begin
```

```
lbLog.Items.Add(' Ошибка HidD_GetNumInputBuffers');
End;

// Получаем идентификатор изготовителя
FillChar(Buffer, SizeOf(Buffer), #0);
If HidD_GetManufacturerString(HidHandle,
                               Buffer, SizeOf(Buffer)) then begin
    lbLog.Items.Add(Format('    Производитель=%s', [Buffer]));
End else begin
    lbLog.Items.Add(' Ошибка HidD_GetManufacturerString');
End;

// Получаем идентификатор продукта
FillChar(Buffer, SizeOf(Buffer), #0);
If HidD_GetProductString(HidHandle, Buffer, SizeOf(Buffer)) then begin
    lbLog.Items.Add(Format('    Продукт=%s', [Buffer]));
End else begin
    lbLog.Items.Add(' Ошибка HidD_GetProductString');
End;

// Получаем серийный номер
FillChar(Buffer, SizeOf(Buffer), #0);
If HidD_GetSerialNumberString(
                               HidHandle, Buffer, SizeOf(Buffer)) then begin
    lbLog.Items.Add(Format('    Серийный номер=%s', [Buffer]));
End else begin
    lbLog.Items.Add(' Ошибка HidD_GetSerialNumberString');
End;

If HidD_GetPreparsedData(HidHandle, PreparsedData) then begin
    HidP_GetCaps(PreparsedData, Capabilities);
    lbLog.Items.Add(Format(
        ' UsagePage                =%x', [Capabilities.UsagePage]
    ));
    lbLog.Items.Add(Format(
        ' InputReportByteLength    =%d', [Capabilities.InputReportByteLength]
    ));
    lbLog.Items.Add(Format(
```

```

    ' OutputReportByteLength =%d',[Capabilities.OutputReportByteLength]
));
lbLog.Items.Add(Format(
    ' FeatureReportByteLength=%d',[Capabilities.FeatureReportByteLength]
));
End else begin
    lbLog.Items.Add(' Ошибка HidD_GetPreparedData');
End;

If Capabilities.FeatureReportByteLength > 0 then begin
    FillChar(InputReport, SizeOf(InputReport), #0);
// InputReport[1]:= 1;
    If HidD_GetFeature(HidHandle, InputReport,
        Capabilities.FeatureReportByteLength) then begin
        // отображение полученных данных
        S:= '';
        For i:= 1 to Capabilities.FeatureReportByteLength-1 do begin
            S:= S + ' ' + IntToHex(InputReport[i], 2);
        End;
        lbLog.Items.Add(' Прочитано: ' + S);
    End else begin
        lbLog.Items.Add(
            ' Ошибка HidD_GetFeature ('+SysErrorMessage(GetLastError)+'')');
    End;
End else begin
    lbLog.Items.Add(' FeatureReport не поддерживается');
End;

If Capabilities.InputReportByteLength > 0 then begin
    FillChar(InputReport, SizeOf(InputReport), #0);
    If ReadFile(HidHandle, InputReport,
        Capabilities.InputReportByteLength, BytesRead, nil) then begin
        // отображение полученных данных
        S:= '';
        For i:= 1 to BytesRead do begin
            S:= S + ' ' + IntToHex(InputReport[i], 2);
        End;
        lbLog.Items.Add(' Прочитано: ' + S);
    End;
End;

```

```
End else begin
    lbLog.Items.Add(
        ' Ошибка ReadFile ('+SysErrorMessage(GetLastError)+') ');
End;
End else begin
    lbLog.Items.Add(' InputReport не поддерживается');
End;

// Освободить блок PreparedData
HidD_FreePreparedData(PreparedData);
// Освободить дескриптор устройства
CloseHandle(HidHandle);
end;
```

Заметим, что после вызова `HidD_GetPreparedData` полученный буфер необходимо освободить с помощью функции `HidD_FreePreparedData`.

Запустив программу (ее полный код можно найти на компакт-диске к книге), мы увидим информацию, показанную на рис. 13.19.

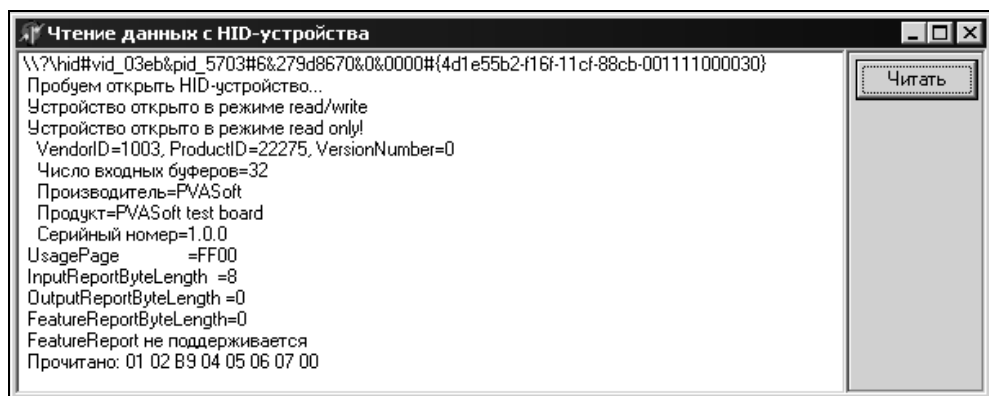


Рис. 13.19. Окно чтения INPUT-репорта с HID-устройства

Внимательно посмотрев на рисунок, можно заметить, что функция `hidP_GetCaps` вернула размер буфера на единицу больше, чем мы описывали в дескрипторе устройства. Это связано с тем, что в первом байте буфера передается номер репорта (Report ID). Более подробно этот вопрос мы рассмотрим в следующих разделах.

В заключение приведем код функции `GetHIDString` для получения строки по ее идентификатору (листинг 13.43).

Листинг 13.43. Получение строки по идентификатору

```
// Получение строки по идентификатору
Function GetHIDString(StrDescriptor : Byte): WideString;
var Buffer : array [0..253] of WideChar;
begin
    Result := 'Ошибка';
    if StrDescriptor <> 0 then
        if HidD_GetIndexedString(HidHandle, StrDescriptor,
            Buffer, SizeOf(Buffer)) then
            Result:= Buffer;
end;
```

13.7. Дополнительные функции Windows XP

Как мы уже говорили, для чтения INPUT-репортов используется функция `ReadFile` (или `ReadFileEx`). Windows XP предоставляет еще одну функцию для чтения репортов типа INPUT — `HidD_GetInputReport`. Заголовок этой функции показан в листинге 13.44.

Листинг 13.44. Функция `HidD_GetInputReport`

```
{IFDEF WINXP}
function HidD_GetInputReport(
    HidDeviceObject : THandle; // дескриптор устройства
    Buffer           : Pointer; // буфер
    BufferLength     : ULONG    // размер буфера
): LongBool; stdcall;
{$ENDIF}
```

Так же как и для функции `HidD_GetFeature`, размер буфера должен быть на 1 больше, чем описано в дескрипторе устройства, а нулевой байт — содержать идентификатор репорта (более подробно мы обсудим идентификаторы репортов в следующем разделе).

При использовании `HidD_GetInputReport` хост посылает устройству запрос `GET_REPORT` с типом запроса `INPUT`, т. о. для корректного ответа необходи-

мо обрабатывать этот запрос, как мы это делали в листинге 13.39. Пример вызова `HidD_GetInputReport` приведен в листинге 13.45, а полный пример можно найти на компакт-диске к книге.

Листинг 13.45. Использование `HidD_GetInputReport`

```
// Только Windows XP!
If Capabilities.InputReportByteLength > 0 then begin
    // Инициализация буфера
    FillChar(InputReport, SizeOf(InputReport), #0);
    // Получение данных
    If HidD_GetInputReport(HidHandle, @InputReport,
        Capabilities.InputReportByteLength) then begin
        // Отображение полученных данных
        S:= '';
        For i:= 0 to Capabilities.InputReportByteLength-1 do begin
            S:= S + ' ' + IntToHex(InputReport[i], 2);
        End;
        lbLog.Items.Add(' Прочитано: ' + S);
    End else begin
        // Ошибка
        lbLog.Items.Add(
            ' Ошибка HidD_GetInputReport ('+SysErrorMessage(GetLastError)+') ');
    End;
End;
```

13.8. Устройство с несколькими репортами

Как мы уже говорили, спецификация HID позволяет создавать устройства с несколькими репортами одновременно. В этом разделе мы покажем несколько связанных с этим тонкостей.

Итак, добавим в наше устройство (листинг 13.38) еще один репорт типа FEATURE. Получившийся дескриптор показан в листинге 13.46. Первый репорт будет иметь идентификатор 1 и содержать 7 байт, а второй — идентификатор 2 и содержать 4 байта.

Листинг 13.46. Дескриптор с двумя FEATURE-репортами

```

#define SIZE_OF_REPORT      47

code struct{
    byte rep[SIZE_OF_REPORT];
}

HIDReport =
    /* HID Report */
    {
        0x06, 0x00, 0xff, /* USAGE_PAGE (Generic Desktop) */

        0x09, 0x01,      /* USAGE (Vendor Usage 1) */
        0xa1, 0x01,      /* COLLECTION (Application) */
        0x19, 0x01,      /*   USAGE_MINIMUM (Vendor Usage 1) */
        0x29, 0x01,      /*   USAGE_MAXIMUM (Vendor Usage 1) */
        0x15, 0x00,      /*   LOGICAL_MINIMUM (0) */
        0x26, 0xff, 0x00, /*   LOGICAL_MAXIMUM (255) */
        0x85, 0x01,      /*   REPORT_ID (1) */
        0x75, 0x08,      /*   REPORT_SIZE (8) */
        0x95, 0x07,      /*   REPORT_COUNT(7) */
        0xB1, 0x02,      /*   FEATURE (Data,Var,Abs) */
        0xc0             /* END_COLLECTION */

        ,

        0x09, 0x01,      /* USAGE (Vendor Usage 1) */
        0xa1, 0x01,      /* COLLECTION (Application) */
        0x19, 0x01,      /*   USAGE_MINIMUM (Vendor Usage 1) */
        0x29, 0x01,      /*   USAGE_MAXIMUM (Vendor Usage 1) */
        0x15, 0x00,      /*   LOGICAL_MINIMUM (0) */
        0x26, 0xff, 0x00, /*   LOGICAL_MAXIMUM (255) */
        0x85, 0x02,      /*   REPORT_ID (2) */
        0x75, 0x08,      /*   REPORT_SIZE (8) */
        0x95, 0x04,      /*   REPORT_COUNT(4) */
        0xB1, 0x02,      /*   FEATURE (Data,Var,Abs) */
        0xc0             /* END_COLLECTION */
    };

```

При правильной передаче репорта нам нужно будет различать, какой именно репорт запросил хост. Для этого мы изменим функцию `hid_get_report`, как показано в листинге 13.47.

Листинг 13.47. Передача одного из двух репортов

```
void hid_get_report()
{
    data byte bReportType;
    data byte bReportID;

    /* Если устройство еще не сконфигурировано — ответ STALL */
    if (usb_configuration_nb == 0){
        STALL();
        return;
    }

    /* Индикатор чтения */
    P3.3=p_test; p_test=!p_test;

    /* Считать идентификатор репорта */
    bReportID = Usb_read_byte();
    /* Считать тип репорта */
    bReportType = Usb_read_byte();

    /* Переключение на передачу */
    Usb_clear_rx_setup();
    Usb_set_DIR();

    switch (bReportID)
    {
    case 1:
    {
        /* Передача первого репорта */
        Usb_write_byte(0x01);
        Usb_write_byte(0x02);
        Usb_write_byte(point1_state);
        Usb_write_byte(0x04);
        Usb_write_byte(0x05);
        Usb_write_byte(0x06);
        Usb_write_byte(0x07);
        break;
    }
    }
```

```
/* Передача второго репорта */  
case 2:  
{  
    Usb_write_byte(0x11);  
    Usb_write_byte(0x12);  
    Usb_write_byte(point1_state);  
    Usb_write_byte(0x14);  
    break;  
}  
/* Такой репорт не поддерживается */  
default:  
    STALL();  
    return;  
}  
point1_state++;  
// Подтверждение и переключение на прием  
END_OK();  
}
```

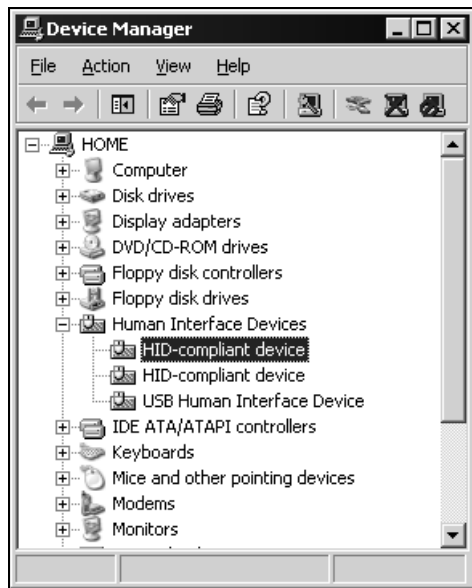


Рис. 13.20. Обнаружено устройство с двумя репортами

Подключив наше новое устройство к шине, мы увидим, что Windows обнаружил два HID-устройства (рис. 13.20), однако прочитать с них данные с помощью нашей программы не удастся.

Для чтения репортов, имеющих идентификатор, необходимо передавать его в первом байте буфера репорта. Скажем сразу, что сделать это для INPUT-репортов можно только в Windows XP при использовании функции `HidD_GetInputReport`, а для FEATURE-репортов нужно использовать функцию `HidD_GetFeature` (листинг 13.48).

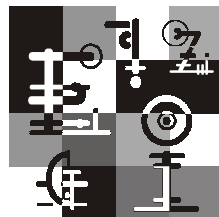
Листинг 13.48. Использование `HidD_GetFeature` для запроса репорта по номеру

```
// Инициализация буфера
FillChar(InputReport, SizeOf(InputReport), #0);
// Определение номера репорта
If Capabilities.FeatureReportByteLength = 8 then InputReport[0] := 1
  Else InputReport[0] := 2;
// Получение репорта
If HidD_GetFeature(HidHandle, InputReport,
  Capabilities.FeatureReportByteLength) then begin
  // Отображение данных
End else begin
  // Ошибка
End;
Посмотрим на строку
If Capabilities.FeatureReportByteLength = 8
  then InputReport[0] := 1
  else InputReport[0] := 2;
```

Здесь находится первый подводный камень. Если до сих пор мы никак не были привязаны к характеристикам устройства, то здесь мы вынуждены воспользоваться знаниями о соответствии длины репорта и его номере. Собственно эта проверка и означает выбор одного из двух "устройств", обнаруженных Windows при использовании двойного репорта.

И еще одно замечание. Спецификация HID запрещает создание двух репортов с одинаковыми идентификаторами (даже разных типов). Устройства, имеющие некорректное описание репорта, работать не будут.

Глава 14



Создание USB-устройства на основе ATMEL AT89C5131

Не HID-ом единым жив человек.

Использование HID-устройств избавляет от написания драйвера, однако часто ограничения HID не позволяют воспользоваться этими удобствами. Значит, нужно писать свой драйвер.

14.1. Не-HID-устройство

В предыдущей главе мы уже сделали USB-устройство, которое затем преобразовали согласно спецификации HID. Для нужд этой главы мы остановимся на обычном USB-устройстве. В листинге 14.1 приведены дескрипторы нашего устройства.

Листинг 14.1. Дескрипторы устройства

```
#define USB_SPECIFICATION      0x0100
#define DEVICE_CLASS           0x00
#define DEVICE_SUB_CLASS       0x00
#define DEVICE_PROTOCOL        0x00
#define EP_CONTROL_LENGTH      8
#define VENDOR_ID              0xEB03    /* Atmel vendor ID = 03EBh */
#define PRODUCT_ID             0x358
#define RELEASE_NUMBER         0x0000

#define LANG_ID                0x00
#define MAN_INDEX              0x01
#define PRD_INDEX              0x02
#define SRN_INDEX              0x03
```

```

code struct usb_st_device_descriptor usb_device_descriptor =
{
    sizeof(usb_device_descriptor),
    DEVICE,                /* =1 */
    USB_SPECIFICATION,    /* спецификация */
    DEVICE_CLASS,          /* класс устройства */
    DEVICE_SUB_CLASS,
    DEVICE_PROTOCOL,      /* протокол USB */
    EP_CONTROL_LENGTH,    /* размер пакета 0-й точки */
    VENDOR_ID,            /* ID производителя и устройства */
    PRODUCT_ID,
    RELEASE_NUMBER,       /* версия устройства */
    MAN_INDEX,            /* дескр. строки изготовителя */
    PRD_INDEX,            /* дескр. строки продукта */
    SRN_INDEX,            /* дескр. строки серийного ном. */
    1                      /* число конфигураций */
};

#define CONF_LENGTH        wSWAP(25)
#define CONF_NB            1
#define CONF_ATTRIBUTES    USB_CONFIG_BUSPOWERED
#define MAX_POWER          50 /* = 100 mA */

/* INTERFACE 0 DESCRIPTOR */
#define INTERFACE0_CLASS    0x00
#define INTERFACE0_SUB_CLASS 0xFF
#define INTERFACE0_PROTOCOL 0xFF

/* первая конечная точка */
#define EP_1_CONFIG        (INTERRUPT|EP_CONFIG_IN) /* конфигурация */
#define EP_1_ADDRESS       (1|EP_DIRECT_IN)         /* адрес */
#define EP_1_ATTRIBUTES    INTERRUPT                /* атрибуты */

code struct
{
    struct usb_st_configuration_descriptor cfg;
    struct usb_st_interface_descriptor ifc;
    struct usb_st_endpoint_descriptor ep1;
}

```

```

usb_configuration =
{
/* CONFIGURATION */
{ 9,
  CONFIGURATION,      /* =2 */
  CONF_LENGTH,        /* длина всех дескрипторов */
  1,                  /* число интерфейсов */
  CONF_NB,            /* номер конфигурации */
  0,                  /* дескр. строки конфигур. */
  CONF_ATTRIBUTES,    /* атрибуты */
  MAX_POWER           /* 100 mA */
},
/* INTERFACE 0 */
{ 9,
  INTERFACE,          /* =4 */
  0,                  /* 0-й интерфейс */
  0,                  /* номер альтернативного инт. */
  1,                  /* 1 конечная точка (кроме 0) */
  INTERFACE0_CLASS,
  INTERFACE0_SUB_CLASS,
  INTERFACE0_PROTOCOL,
  0                   /* дескр. строки интерфейса */
},
/* Дескриптор первой конечной точки */
{
  7,
  ENDPOINT,           /* дескриптор ENDPOINT */
  EP_1_ADDRESS,       /* номер конечной точки */
  EP_1_ATTRIBUTES,    /* атрибуты конечной точки */
  wSWAP(8),           /* максимальный размер пакета */
  1                   /* частота опроса */
}
};

```

Как видно, устройство имеет один интерфейс и одну конечную точку (кроме нулевой) типа INTERRUPT с направлением передачи IN, т. е. от устройства к хосту. Размер пакета данных, передаваемых через эту точку, равен 8. Ради экономии места мы не привели строковые дескрипторы (они описаны в листинге 13.29).

Данные, которые мы будем передавать, состоят из 8 байт, один из которых мы будем увеличивать при каждой передаче данных (листинг 14.2).

Листинг 14.2. Структура передаваемых данных

```
if (!end_point1_ready) /* предыдущий пакет отправлен */
{
    Usb_select_ep(1);
    Usb_write_byte(55);
    Usb_write_byte(point1_state); point1_state++;
    Usb_write_byte(0x52);
    Usb_write_byte(0x54);
    Usb_write_byte(0x55);
    Usb_write_byte(0x56);
    Usb_write_byte(0x57);
    Usb_write_byte(0x58);
    Usb_set_tx_ready();

    end_point1_ready = TRUE;
}
```

Все это мы уже делали в *главе 13*, поэтому дополнительных комментариев давать не будем.

14.2. Создание драйвера с помощью Driver Studio

В *разд. 9.5* мы описали несколько инструментов, с помощью которых можно быстро создать драйвер USB-устройства. В этом разделе мы рассмотрим процесс создания драйвера с помощью NuMega Driver Studio.

Для работы нам потребуются установленные Microsoft Visual Studio 6.0 и Microsoft DDK (далее мы будем пользоваться сокращениями VS и DDK соответственно).

Установка Driver Studio (сокращенно DS) довольно проста. После инсталляции в VS появляется дополнительное меню со следующими кнопками (рис. 14.1):

- ☐ запуск помощника создания драйвера;
- ☐ запуск помощника создания сетевого драйвера;

- ☐ изменение переменных окружения;
- ☐ компиляция с помощью утилиты BULD из DDK.

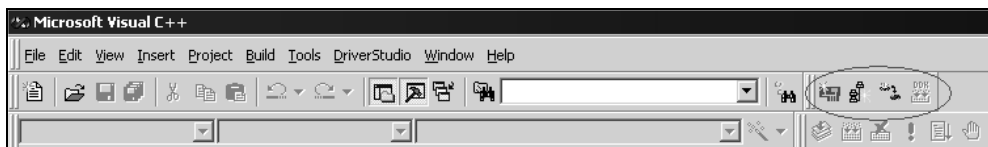


Рис. 14.1. Меню Driver Studio в MS Visual Studio

Сразу отметим, что для компиляции наших драйверов можно использовать либо компиляцию из среды VS, либо воспользоваться командными файлами `make_XP.bat` и `make2000.bat`. Эти файлы можно найти на прилагаемом к книге компакт-диске.

В случае возникновения проблем с компиляцией (особенно в Windows 2000) проверьте правильность переменных окружения. Должны существовать следующие переменные (мы приводим только основные):

- ☐ `BASEDIR` — путь к папке DDK, например, `F:\WINXPDDK`;
- ☐ `DRIVERWORKS` — путь к папке DS в коротком формате, например, `F:\PROGRA~1\NuMega\SOFTIC~1\DRIVER~3`;
- ☐ `MSDevDir` — путь к папке MSDev, например, `F:\VStudio\Common\MSDev98`.

Настройку переменных DS можно провести прямо из среды VS, нажав третью кнопку в меню (рис. 14.2).

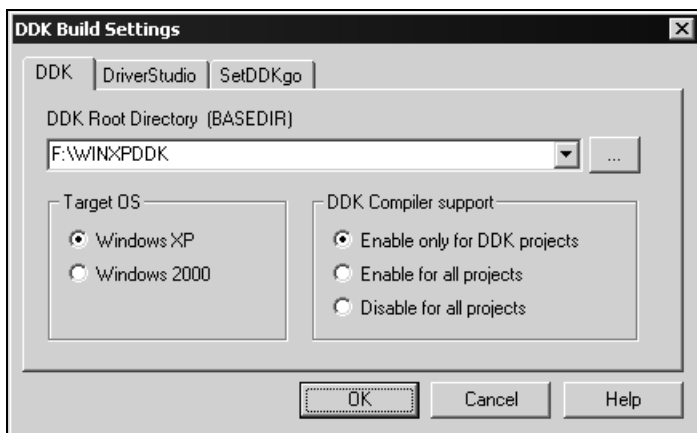


Рис. 14.2. Настройка переменных Driver Studio

После установки потребуется скомпилировать библиотеки DS (вообще говоря, при первой компиляции драйвера эти библиотеки должны бы компилироваться автоматически, но в некоторых случаях этого не происходит). Сделать это можно из командной строки с помощью командного файла `bldlib.bat`, находящегося в папке `DriverWorks`. Для компиляции нужно указать два параметра — тип компиляции и тип библиотек:

❑ компиляция DEBUG-версии: `bldlib.bat checked wdm`

❑ компиляции RELEASE-версии: `bldlib.bat free wdm`

Откомпилировать библиотеку можно и с помощью файла проекта `VdwLibs.dsw`, находящегося в папке `DriverWorks\source`.

После компиляции в папке `DriverWorks\lib\i386` должен появиться файл `vdw_wdm.lib`.

14.2.1. Несколько слов о библиотеке Driver Studio

Библиотека классов DS представляет собой надстройку над чистым WDM API. Это избавляет программиста от использования довольно запутанных низкоуровневых функций, позволяя, тем не менее, выполнять любые необходимые операции.

В нашей книге мы не сможем описать все классы DS, поэтому мы ограничимся только теми, которые потребуются нам для написания USB-драйвера.

Класс KDriver

Класс `KDriver` является базовым классом драйвера. Его задача — предоставить стандартные базовые функции драйвера (такие как `DriverEntry`, `AddDevice`, `Unload` и т. д.). Для этого класса неважно, к какому оборудованию обращается драйвер. Для управления оборудованием создается экземпляр класса `KDevice`.

Для реализации специфических драйверов библиотека предоставляет несколько наследников `KDriver` (рис. 14.3).



Рис. 14.3. Наследование классов `KDriver`

Класс *KDevice*

Класс *KDevice* является базовым классом объекта устройства. Когда драйвер получает запрос *IRP*, объект *KDriver* не обрабатывает запрос самостоятельно. Он передает его объекту *KDevice*, который отвечает за непосредственную связь с оборудованием.

Сам по себе класс *KDevice* используется редко. Чаще используют один из его потомков (рис. 14.4), например, *KPnPDevice*. Каждый драйвер должен иметь хотя бы один объект устройства.

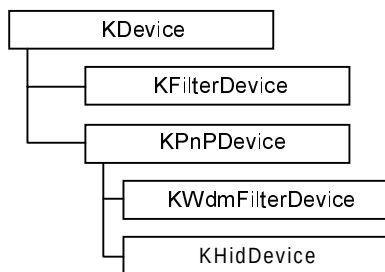


Рис. 14.4. Наследование классов *KDevice*

Объект устройства содержит виртуальные методы для обработки запросов на чтение (*ReadFile*), запись (*WriteFile*) или выполнение специальных функций (*DeviceIoControl*). Конкретный экземпляр объекта устройства должен перекрыть эти методы для придания объекту нужной функциональности.

Класс *KIrp*

Класс *KIrp* представляет собой оболочку для структуры пакета запроса в/в *IRP* (см. разд. 9.3). Как мы увидим далее, создание пакета запроса с помощью этого класса значительно проще, чем создание структуры *IRP* напрямую.

Класс *KRegistryKey*

Класс *KRegistryKey* позволяет обращаться к данным драйвера, сохраненным в реестре. Данные сохраняются в ветке

`HKLM\SYSTEM\CurrentControlSet\Services\<имя драйвера>\Parameters\`

Перечисление секций и данных доступно с помощью функций *EnumerateSubkey* и *EnumerateValue*. Для доступа к конкретным данным класс имеет методы *QueryValue* (чтение), *WriteValue* (запись), *DeleteValue* (удаление).

Класс *KLowerDevice*

Класс *KLowerDevice* является базовым классом для устройства нижнего уровня. Библиотека предоставляет несколько наследников этого класса, реализующих интерфейсы устройств нижнего уровня (рис. 14.5).

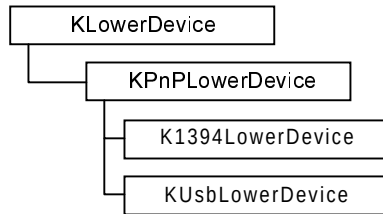


Рис. 14.5. Наследование классов *KLowerDevice*

Класс *KUsbLowerDevice* предоставляет основные функции для обращения к USB-интерфейсу (листинг 14.3).

Листинг 14.3. Некоторые функции *KUsbLowerDevice*

```

class KUsbLowerDevice : public KPNPLowerDevice
{
public:
    // Функция выбора конфигурации
    AC_STATUS ActivateConfiguration(
        UCHAR ConfigurationValue, // номер конфигурации
        ULONG MaxConfigSize=1024 // максим. размер конфигурации
    );

    // Получение строкового дескриптора
    NTSTATUS GetStringDescriptor(
        UCHAR Index,           // номер строки
        PWCHAR pStr,           // буфер для строки
        UCHAR MaxLen,          // размер буфера
        SHORT LangId = 0x109 // код языка
    );

    // Получение счетчика кадров
    ULONG GetCurrentFrameNumber(void);

    // Передача запроса SET_FEATURE
  
```

```

NTSTATUS SetFeature(
    USHORT Feature, // код функции
    PIO_COMPLETION_ROUTINE CompletionRoutine=NULL,
    PVOID Context=NULL
);

// Передача запроса CLEAR_FEATURE
NTSTATUS ClearFeature(
    USHORT Feature, // код функции
    PIO_COMPLETION_ROUTINE CompletionRoutine=NULL,
    PVOID Context=NULL
);

// передача пакета запроса в/в драйверу нижнего уровня
NTSTATUS SubmitUrb(
    PURB pUrb,
    PIO_COMPLETION_ROUTINE CompletionRoutine = NULL,
    PVOID Context=NULL,
    ULONG mSecTimeOut=0
);

```

Параметр `CompletionRoutine` позволяет указать адрес функции, которая будет вызвана при завершении операции. Это позволяет организовать асинхронное выполнение операций в/в.

Классы USB

Мы уже описали несколько классов, относящихся к USB: класс `KhidMiniDriver` (наследник `KDriver`), класс `KhidDevice` (наследник `KDevice`) и класс `KUsbLowerDevice` (наследник `KLowerDevice`). Эти классы предоставляют предопределенную функциональность. Конечно же, библиотека `DS` предоставляет классы для реализации пользовательских USB-интерфейсов.

Объект класса `KUsbInterface` предоставляет функции для работы с USB-интерфейсами. Драйвер может создавать столько интерфейсов, сколько их описано в дескрипторе конфигурации.

Объект класса `KUsbPipe` предоставляет функции для работы с конечными точками. Драйвер должен создавать конечные точки только с параметрами, как они описаны в дескрипторе конфигурации. Создание несуществующих конечных точек приведет к краху системы.

Принцип работы с этими классами следующий:

❑ в конструкторе экземпляра `KDriver` создается экземпляр класса `KPnpDevice`;

- ❑ в конструкторе экземпляра `KPnpDevice` создаются:
 - экземпляр `m_Lower` класса `KUsbLowerDevice`, предоставляющий доступ к функциям низкого уровня;
 - один или несколько экземпляров класса интерфейса `KUsbInterface`;
 - при необходимости создаются экземпляры классов конечных точек `KUsbPipe`;
- ❑ в функции драйвера `OnStartDevice` выполняется активизация одной из конфигураций с помощью вызова `m_Lower.ActivateConfiguration()`;
- ❑ при получении запроса в/в, например, запроса чтения, производятся следующие действия:
 - с помощью методов конечной точки создается и инициализируется пакет `PURB`, например, вызовом `BuildBulkTransfer`, `BuildControlTransfer` или `BuildInterruptTransfer`;
 - созданный запрос передается драйверу нижнего уровня с помощью вызова метода конечной точки `SubmitUrb`;
 - полученные данные (сохраненные в пакете `URB`) передаются пользовательской программе, инициировавшей запрос.

14.2.2. Другие классы Driver Studio

Для общности приведем несколько классов, которые могут оказаться полезными при разработке драйвера:

- ❑ `KIoRegister` — порт в/в. Этот класс позволяет читать и записывать в порт 8, 16, и 32-битные значения;
- ❑ `KIoRange` — диапазон адресов в/в. Практически представляет собой массив экземпляров класса `KIoRegister`;
- ❑ `KMemoryRegister` представляет собой отдельную ячейку памяти в адресном пространстве устройства;
- ❑ `KMemoryRange` представляет собой диапазон адресов памяти;
- ❑ `KEvent` — объект синхронизации работы потоков;
- ❑ `KTimer` — объект таймера;
- ❑ `KSystemThread` — класс для создания нового потока в драйвере;
- ❑ `KList` — класс списка;
- ❑ `KFile` — класс для работы с файлами;
- ❑ `KUnicode` — класс для работы со строками.

14.2.3. Создание шаблона драйвера с помощью Driver Studio

Итак, для создания драйвера все готово. Нажимаем первую кнопку меню...

Шаг 1. Задание имени и пути проекта

На первом шаге создания проекта необходимо задать имя проекта и путь, по которому будут располагаться файлы драйвера (рис. 14.6).

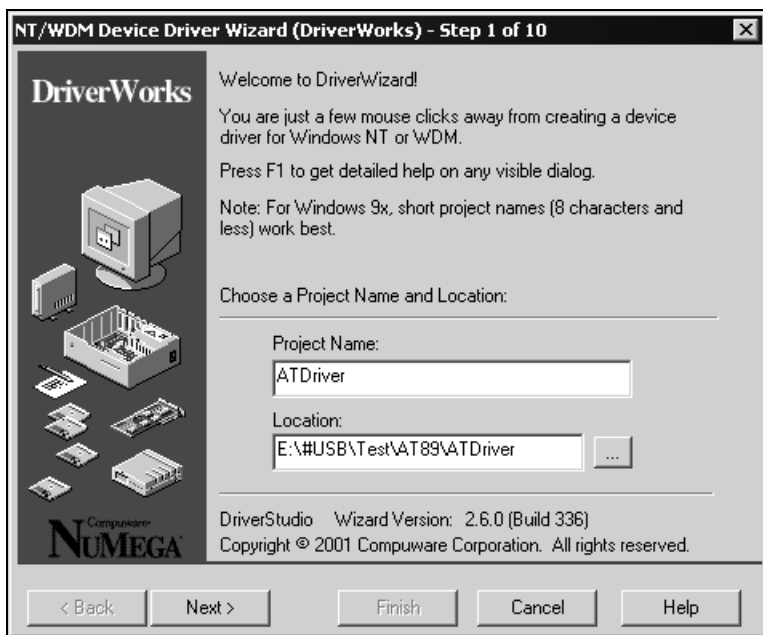


Рис. 14.6. Первый шаг создания драйвера

Шаг 2. Выбор архитектуры драйвера

На втором шаге необходимо выбрать архитектуру создаваемого драйвера: Windows NT 4.0 или WDM (рис. 14.7). Первый вариант уже практически не используется, поэтому мы выбираем модель WDM.

Шаг 3. Выбор шины

На третьем шаге необходимо выбрать шину, на которой располагается устройство, для которого создается драйвер (рис. 14.8). В нашем случае это USB (заметим, что если драйвер создается для последовательного или параллельного интерфейса, то следует выбрать **None**, т. к. обращение к таким портам производится напрямую, без дополнительного драйвера шины).



Рис. 14.7. Второй шаг создания драйвера

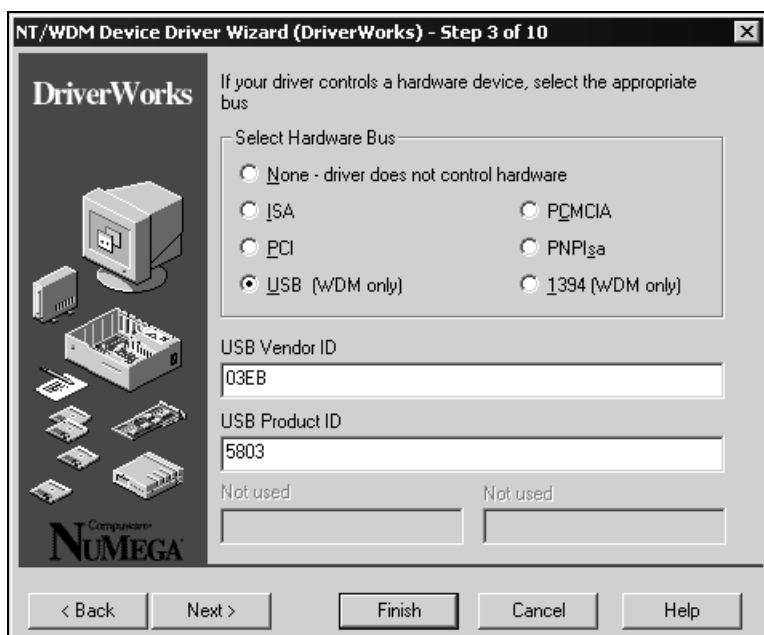


Рис. 14.8. Третий шаг создания драйвера

Для USB-устройств следует указать следующие параметры: идентификатор производителя (USB Vendor ID) и идентификатор продукта (USB Product ID). Конечно, следует указывать те же идентификаторы, что и в дескрипторе устройства.

Заметим, что указанные значения идентификаторов используются только для формирования INF-файла, их можно легко изменить и после генерации файлов драйвера. Так, в нашем случае в INF-файл будет записана строка:

```
[Manufacturer]
```

```
%MfgName%=Mfg0
```

```
[Mfg0]
```

```
%DeviceDesc%=ATDriver_DDI, USB\VID_03EB&PID_5803
```

Шаг 4. Задание набора конечных точек

На четвертом шаге необходимо задать набор конечных точек, поддерживаемых устройством (рис. 14.9). Свойства конечных точек должны в точности совпадать со свойствами, описанными в дескрипторе конфигурации, иначе запуск драйвера приведет к краху системы.

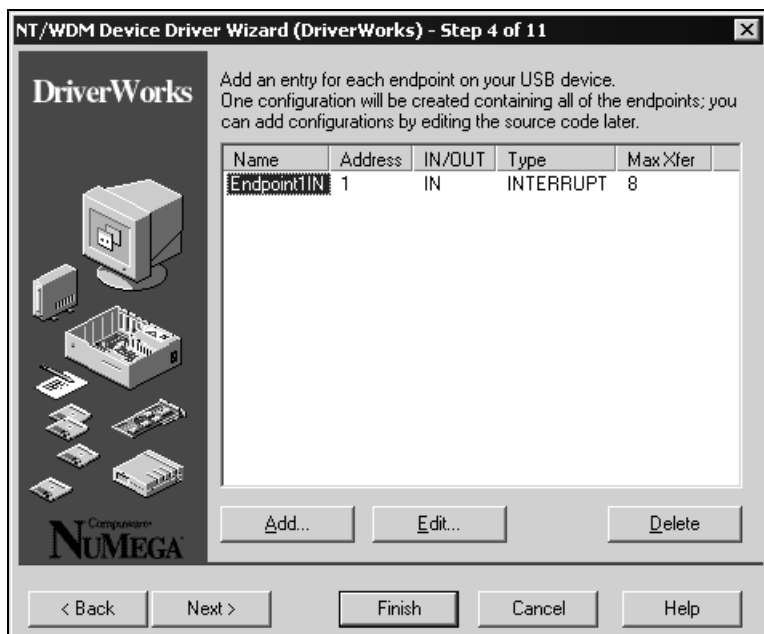


Рис. 14.9. Четвертый шаг создания драйвера

Для добавления новой конечной точки следует нажать кнопку **Add** и заполнить требуемые поля свойств конечной точки (рис. 14.10).



Рис. 14.10. Добавление конечной точки

Кнопка **Suggest** в диалоговом окне добавления конечной точки позволяет автоматически сгенерировать имя переменной для конечной точки.

Шаг 5. Задание имени класса и файла

На пятом шаге (рис. 14.11) необходимо задать имена для присвоения файлу C++, который содержит класс драйвера (**File Name**), и самому классу драйвера (**Driver Class**). Эти имена создаются на основе имени проекта и обычно их оставляют без изменений.

Шаг 6. Выбор функций драйвера

На шестом шаге необходимо выбрать функции, которые будет поддерживать драйвер (рис. 14.12). Возможные варианты функциональности:

- ☐ **Read** — драйвер будет обрабатывать запросы на чтение;
- ☐ **Write** — драйвер будет обрабатывать запросы на запись;
- ☐ **Flush** — драйвер поддерживает функцию сброса буферов;
- ☐ **Device Control** — драйвер будет поддерживать пользовательские запросы с помощью функции `DeviceIoControl`;

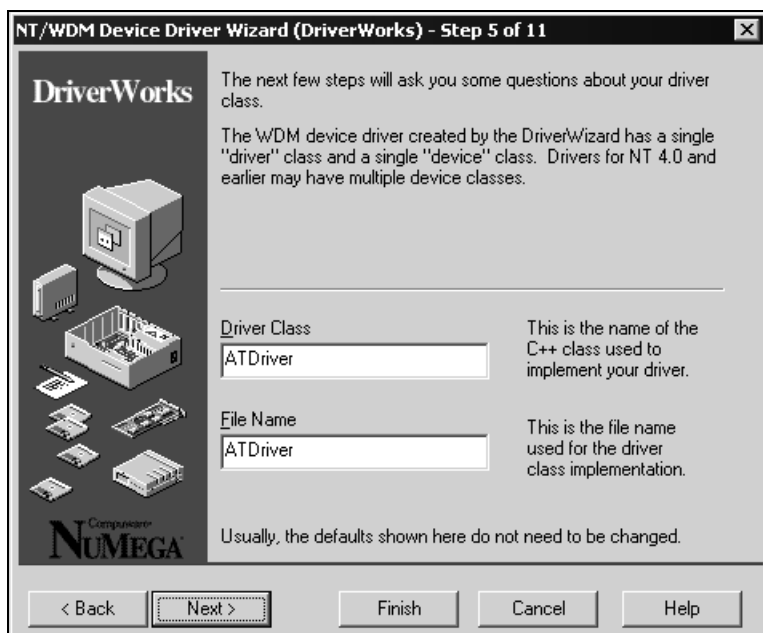


Рис. 14.11. Пятый шаг создания драйвера

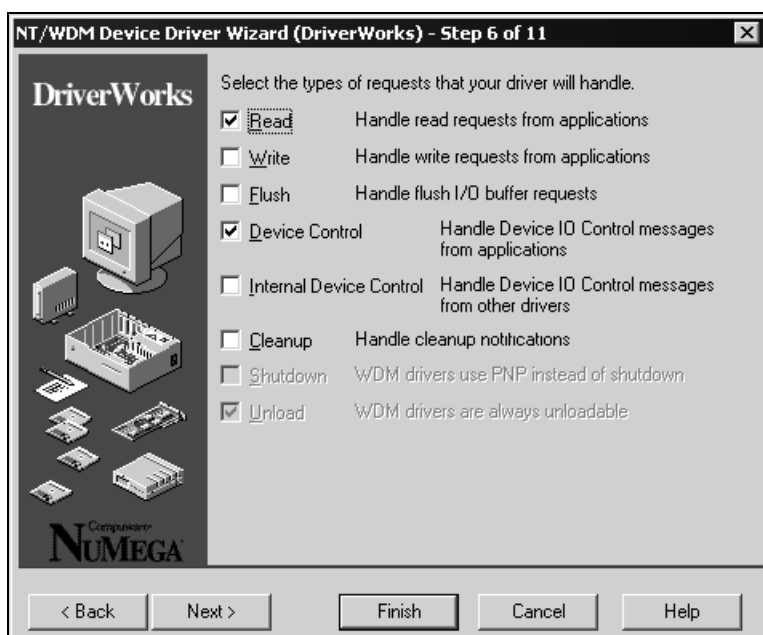


Рис. 14.12. Шестой шаг создания драйвера

☐ **Internal Device Control** — драйвер будет обрабатывать запросы от других драйверов;

☐ **Cleanup** — драйвер будет обрабатывать запросы на очистку буфера обмена.

Для нашего драйвера мы оставим только обработку функций чтения и пользовательских запросов.

Шаг 7. Выбор способа обработки запросов

На седьмом шаге необходимо выбрать способ обработки запросов (рис. 14.13). Опция **Select queuing method** выбирает, каким образом будут буферизироваться запросы на в/в:

☐ **None** — запросы не буферизируются в очереди;

☐ **DriverManaged** — драйвер содержит одну или более одной очереди, в которой сохраняются запросы на в/в, пришедшие от других драйверов или системы;

☐ **SystemManaged** — драйвер использует только одну очередь сообщений.

Также надо выбрать, будут ли буферизироваться запросы на чтение и запись.

Для USB-драйверов выбор не предоставляется, поэтому просто пропускаем этот шаг.



Рис. 14.13. Седьмой шаг создания драйвера

Шаг 8. Создание сохраняемых параметров драйвера

На восьмом шаге можно задать параметры, которые драйвер будет загружать из реестра Windows при старте (рис. 14.14).

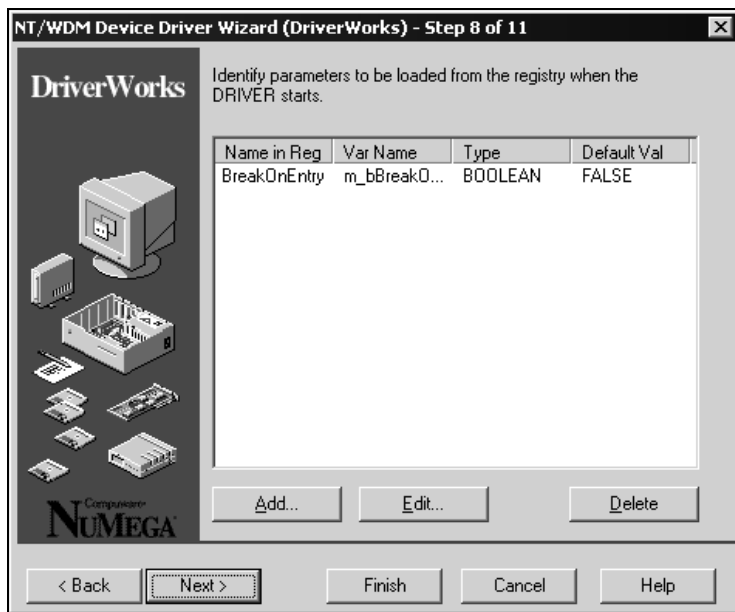


Рис. 14.14. Восьмой шаг создания драйвера

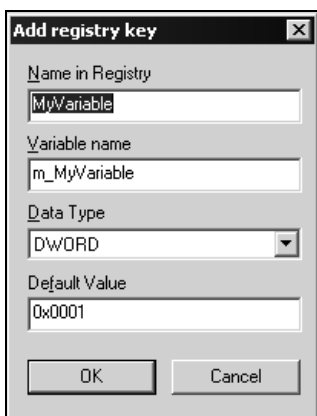


Рис. 14.15. Добавление нового сохраняемого параметра

Для добавления нового параметра нужно нажать кнопку **Add**. При этом задается параметр реестра, имя переменной, тип параметра и его значение по

умолчанию (рис. 14.15). Запись и считывание параметров из реестра позволяют драйверу задавать какие-либо конфигурационные параметры, сохранять данные, необходимые для его запуска или работы.

Шаг 9. Свойства драйвера

Девятый шаг создания драйвера состоит из нескольких частей. В верхней части окна мастера задается имя класса драйвера. С помощью кнопки **Rename** можно задать другое имя.

Вкладка **Interface** (рис. 14.16) позволяет задать имя драйвера (как он будет открываться с помощью функции CreateFile) и способ обращения к драйверу — по символьному имени (см. *разд. 9.2*) или по уникальному идентификатору (GUID).

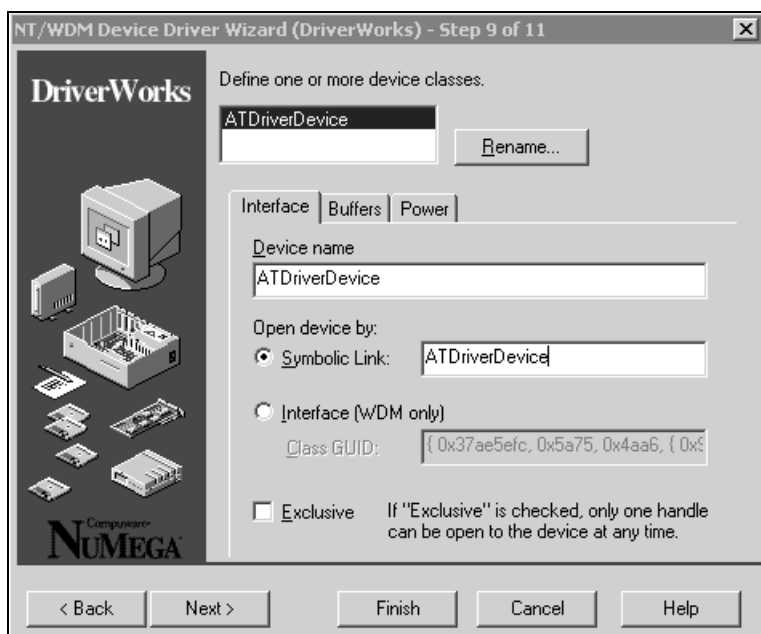


Рис. 14.16. Вкладка **Interface** девятого шага создания драйвера

Вкладка **Buffers** (рис. 14.17) позволяет выбрать способ передачи буферов памяти. Опция **Buffered** будет создавать промежуточные буферы внутри драйвера, а затем копировать данные в буфер пользовательской программы, а опция **Direct** заставит драйвер работать с буферами пользовательской программы напрямую. Подробности доступа к буферам можно найти в *разд. 9.3*.

Вкладка **Power** (рис. 14.18) позволяет выбрать способ управления энергопотреблением.

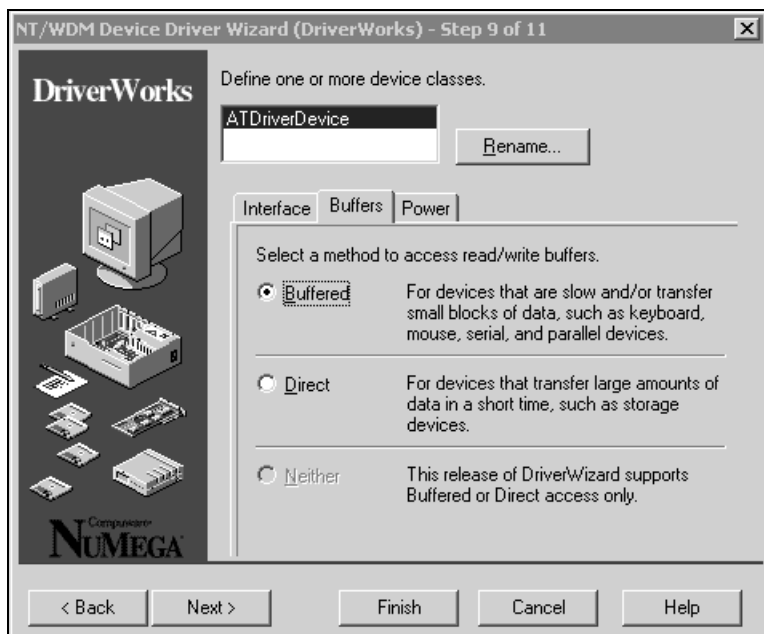


Рис. 14.17. Вкладка **Buffers** девятого шага создания драйвера



Рис. 14.18. Вкладка **Power** девятого шага создания драйвера

Шаг 10. Задание кодов IOCTL

На десятом шаге задаются коды функции DeviceIoControl (рис. 14.19), если, конечно, была выбрана поддержка этой функции на 6 шаге.

Кнопка **Add** позволяет добавить код пользовательской функции (рис. 14.20).

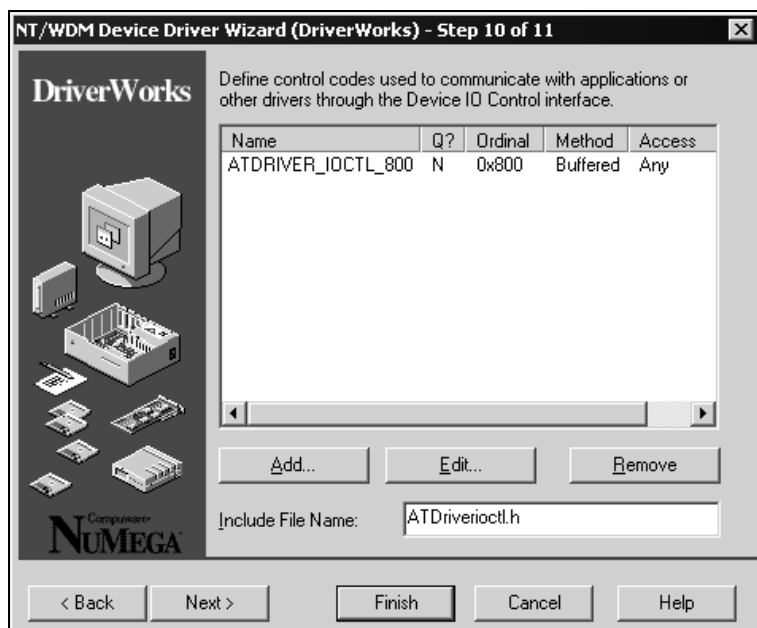


Рис. 14.19. Десятый шаг создания драйвера

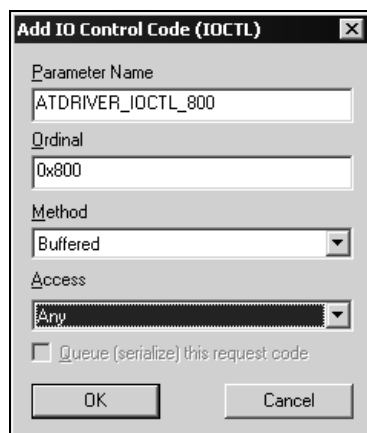


Рис. 14.20. Добавление кода пользовательской функции

Шаг 11. Дополнительные настройки

На последнем шаге создания драйвера (рис. 14.21) выбирается, нужно ли создавать тестовое приложение для драйвера (**Create test console application**) и дополнительные настройки отладки и создания лога событий.

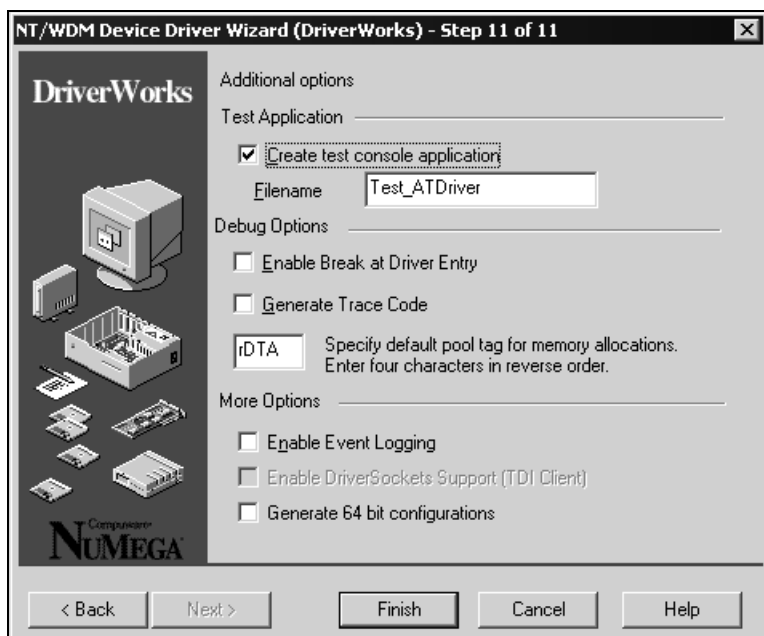


Рис. 14.21. Последний шаг создания драйвера

Вообще говоря, тестовое приложение нам не нужно, т. к. для создания приложения мы будем использовать Delphi, но для простейшего тестирования драйвера можно использовать и приложение C++.

14.2.4. Доработка шаблона драйвера

Получившийся драйвер является заготовкой для настоящего USB-драйвера и требует небольшой доработки. Точнее, требуется выполнить следующий набор действий:

1. Дописать реализацию функций в/в.
2. Сгенерированный драйвер содержит только один интерфейс, поэтому при необходимости нужно создать нужное число экземпляров класса KusbInterface.

3. По умолчанию активизируется конфигурация номер 1; если устройство имеет другие настройки или несколько конфигураций, нужно обеспечить их функционирование.
4. Дописать реализацию функций IOCTL.

14.2.5. Базовые методы класса устройства

Листинг 14.4 показывает код класса ATDriver (мы позволили себе несколько сократить листинг, выделив только важные места кода). Обратим внимание на переменную `m_Unit`. При загрузке драйвера (функция `DriverEntry`) эта переменная обнуляется, а при каждом вызове `AddDevice` увеличивается на единицу. Это позволяет создавать экземпляры класса `ATDriverDevice` с уникальными в пределах системы именами. Первое устройство, для которого будет загружен драйвер, будет иметь имя `ATDriverDevice0`, второе `ATDriverDevice1` и т. д.

Листинг 14.4. Код класса ATDriver (с сокращениями)

```
#define VDW_MAIN
#include <vdw.h>
#include <kusb.h>
#include "ATDriver.h"
#include "ATDriverDevice.h"

#pragma hdrstop("ATDriver.pch")

// Generated by DriverWizard version DriverStudio 2.6.0 (Build 336)
//
// ATDriver::DriverEntry
// Вызывается при загрузке драйвера
//
NTSTATUS ATDriver::DriverEntry(PUNICODE_STRING RegistryPath)
{
    m_Unit = 0;
    return STATUS_SUCCESS;
}

// End INIT section
//
#pragma code_seg()
```

```
// ATDriver::AddDevice
// Вызывается при подключении устройства
////////////////////////////////////
NTSTATUS ATDriver::AddDevice(PDEVICE_OBJECT Pdo)
{
    ATDriverDevice * pDevice = new (
        static_cast<PCWSTR>(KUnitizedName(L"ATDriverDevice", m_Unit)),
        FILE_DEVICE_UNKNOWN,
        static_cast<PCWSTR>(KUnitizedName(L"ATDriverDevice", m_Unit)),
        0,
        DO_BUFFERED_IO
    )
    ATDriverDevice(Pdo, m_Unit);

    m_Unit++;
    DbgPrint("Unit number is %d", m_Unit);

    return status;
}
```

Собственно работа с устройством сосредоточена в методах класса `ATDriverDevice`. Как мы описывали в *разд. 14.2.1*, в конструкторе этого класса создаются экземпляры объектов драйвера нижнего уровня (`m_Lower`), интерфейса (`m_Interface`) и конечной точки (`m_Endpoint1IN`). При необходимости здесь же можно создать другие интерфейсы и конечные точки, если они поддерживаются устройством. Листинг 14.5 показывает конструктор класса нашего устройства (мы позволили себе сократить проверки для повышения читабельности кода).

Листинг 14.5. Конструктор класса `ATDriverDevice`

```
ATDriverDevice::ATDriverDevice(PDEVICE_OBJECT Pdo, ULONG Unit) :
    KPnpDevice(Pdo, NULL)
{
    DbgPrint("ATDriverDevice::ATDriverDevice START");
    NTSTATUS status;

    DbgPrint("Unit number is %d", Unit);
    // Запомнить номер устройства
    m_Unit = Unit;
```

```

// Создание объекта драйвера нижнего уровня
status = m_Lower.Initialize(this, Pdo);
// Создание объекта интерфейса
status = m_Interface.Initialize(
    m_Lower, // Объект драйвера нижнего уровня
    0,       // Номер интерфейса
    1,       // Номер конфигурации
    0        // Номер альтернативного интерфейса
);
// Создание конечной точки
status = m_Endpoint1IN.Initialize(m_Lower, 0x81 /* 8 */);
// Информирование драйвера нижнего уровня о создании
// нового интерфейса
SetLowerDevice(&m_Lower);
// Инициализация политики PnP
SetPnpPolicy();
}

```

Важно

По умолчанию конструктор конечной точки принимает три параметра: указатель на объект драйвера нижнего уровня, номер конечной точки и максимальный размер пакета. Последний параметр нужно самостоятельно удалить из конструктора, т. к. размер пакета будет указываться при создании IRP-пакета.

Следующий важный метод — `OnStartDevice`, вызываемый при начале работы устройства. Код этого метода показан в листинге 14.6. Если устройство поддерживает несколько конфигураций, можно изменить номер активной конфигурации.

Листинг 14.6. Метод `ATDriverDevice::OnStartDevice`

```

NTSTATUS ATDriverDevice::OnStartDevice(KIrp I)
{
    NTSTATUS status = STATUS_UNSUCCESSFUL;
    AC_STATUS acStatus = AC_SUCCESS;

    I.Information() = 0;

    // Активизация конфигурации 1
    acStatus = m_Lower.ActivateConfiguration(

```

```

        1    // Номер конфигурации
    );

    if (acStatus == AC_SUCCESS)
    {
        // Конфигурация установлена успешно
        status = STATUS_SUCCESS;
    }
    if (acStatus == AC_FAILED_TO_OPEN_PIPE_OBJECT)
    {
        // Конфигурация установлена, но созданных конечных
        // точек в ней не существует
        status = STATUS_SUCCESS;
    }

    return status;
}

```

Все остальные методы, кроме метода чтения данных (Read) и обработчика пользовательской функции (ATDRIVER_IOCTL_800_Handler), мы оставляем без изменений. Метод Read мы разберем в следующем разделе, а метод пользовательских функций мы уже рассматривали в *разд. 9.3.5* и повторяться не будем. Его можно использовать, например, для чтения строковых дескрипторов или других целей.

14.2.6. Реализация чтения данных

Метод Read вызывается, когда пользовательское приложение вызывает функцию ReadFile. Код, генерируемый по умолчанию, не производит никаких действий, возвращая пакет нулевой длины.

Нашей задачей является создание пакета URB для организации чтения данных с конечной точки m_Endpoint1IN. Листинг 14.7 показывает реализацию метода Read для чтения данных с конечной точки типа INTERRUPT.

Листинг 14.7. Метод Read для конечной точки типа INTERRUPT

```

NTSTATUS ATDriverDevice::Read(KIrp I)
{
    // Если запрошено 0 байт, то всегда возвращаем успех
    if (I.ReadSize() == 0)

```

```

{
    I.Information() = 0;
    return I.PnpComplete(this, STATUS_SUCCESS);
}
NTSTATUS status = STATUS_SUCCESS;

// Указатель на буфер для чтения данных
PVOID pBuffer = I.BufferedReadDest();
// Число запрошенных для чтения байт
ULONG dwTotalSize = I.ReadSize(CURRENT);
// Будет сохранять число реально прочитанных байт
ULONG dwBytesRead = 0;
// Если конечная точка не активна, то чтение невозможно
if (!m_Endpoint1IN.IsOpen())
{
    I.Information() = 0;
    return I.PnpComplete(this, STATUS_INSUFFICIENT_RESOURCES);
}
// Максимальное число байтов для выбранной конечной точки
ULONG dwMaxSize = m_Endpoint1IN.MaximumTransferSize();
// Если запрошено больше, чем можно прочитать, то
// ограничим запрос максимальным размером
if (dwTotalSize > dwMaxSize) dwTotalSize = dwMaxSize;
// Создает запрос для чтения данных типа INTERRUPT
PURB pUrb = m_Endpoint1IN.BuildInterruptTransfer(
    pBuffer,          // Буфер для чтения
    dwTotalSize,      // Сколько данных нужно прочитать
    TRUE              // разрешаем использование коротких пакетов
);
pUrb->UrbBulkOrInterruptTransfer.TransferFlags =
    (USBD_TRANSFER_DIRECTION_IN | USBD_SHORT_TRANSFER_OK);
// Передаем сформированный запрос драйверу нижнего уровня
status = m_Endpoint1IN.SubmitUrb(pUrb);
// Возвращаем реально прочитанное число байт
dwBytesRead = pUrb->UrbBulkOrInterruptTransfer.TransferBufferLength;
// Освободить пакет
delete pUrb;
// Вернуть реально прочитанное число байт
I.Information() = dwBytesRead;

```

```
// Подтвердить успешное завершение операции  
return I.PnpComplete(this, status);  
}
```

14.2.7. Установка драйвера

Скомпилировав драйвер, мы получим файл ATDriver.sys.

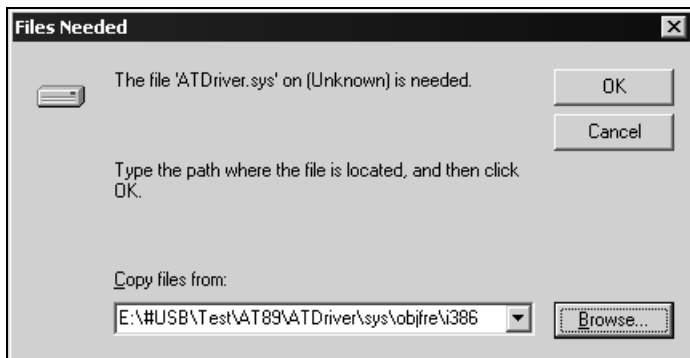


Рис. 14.22. Указание файла драйвера



Рис. 14.23. Драйвер обнаружен

Теперь, подключив наше устройство, нужно установить этот драйвер, указав путь к файлу ATDriver.inf и файлу самого драйвера (рис. 14.22).

Если все прошло успешно, система обнаружит драйвер и установит его (рис. 14.23). Теперь можно заняться созданием программы для чтения данных.

14.2.8. Программа чтения данных

В *главе 9* мы достаточно подробно разобрали методы загрузки и обращения к драйверам, поэтому дополнительных комментариев к листингу 14.8 не требуется.

Листинг 14.8. Модуль для загрузки и обращения к драйверу ATDriverDevice

```
unit DrvLoader;

interface

Uses Windows, SysUtils;

type
  TAT89DriverLoader = class
  private
    hDevice : THandle; // хранить дескриптор драйвера
  public
    Constructor Create;
    Destructor Destroy; override;
  public
    // Регистрация драйвера
    Function CreateService(SysPath : String) : Boolean;
    // Разрегистрация драйвера
    Function RemoveService : Boolean;
    // Старт драйвера
    Function StartService : Boolean;
    // Останов драйвера
    Function StopService : Boolean;
  public
    // Открытие драйвера
    Function Open(DevIndex : Integer) : Boolean;
    // Закрытие драйвера
    Procedure Close;
```



```
public
// Вызов функции DeviceIoControl
Function Func1 : Cardinal;
// Вызов функции чтения
Function Read : String;
End;

implementation

Uses WinSvc, Dialogs;

// Конструктор
Constructor TAT89DriverLoader.Create;
Begin
    Inherited Create;
    hDevice:= INVALID_HANDLE_VALUE;
End;

// Деструктор
Destructor TAT89DriverLoader.Destroy;
Begin
    Close;
    Inherited Destroy;
End;

// Описание драйвера
Const
    // Имя драйвера
    DriverName : PChar= 'ATDriverDevice'#0;
    // Имя файла драйвера
    FileDriver : String = 'ATDriver.sys'#0;
    // Полное имя драйвера
    DriverPath : String = '\\.\ATDriverDevice';

Function TAT89DriverLoader.CreateService(SysPath : String) : Boolean;
var hSCMan, hService: SC_HANDLE;
    DriverPath : String;
Begin
    Result:= False;
```

```
hSCMan:= WinSvc.OpenSCManager(  
    Nil,          { local }  
    Nil,          { SERVICES_ACTIVE_DATABASE }  
    SC_MANAGER_ALL_ACCESS  
);  
If hSCMan = 0 then Exit;  
  
// Получаем полное имя к файлу драйвера  
// Если драйвер находится в текущем каталоге,  
// где запускается программа, то не добавляем SysPath  
DriverPath:= {SysPath + }FileDriver;  
  
// Создаем сервис (регистрируем драйвер)  
hService:= WinSvc.CreateService(hSCMan, Drivename, DriverName,  
    SERVICE_ALL_ACCESS, SERVICE_KERNEL_DRIVER,  
    SERVICE_DEMAND_START, SERVICE_ERROR_NORMAL,  
    PChar(@DriverPath[1]),  
    nil,nil,nil,nil,nil);  
// Файл не найден или сервис уже зарегистрирован  
If hService = 0 then begin  
    MessageDlg(SysErrorMessage(GetLastError), mtError, [mbOK], 0);  
    CloseServiceHandle(hSCMan);  
    Exit;  
End;  
  
// Регистрация успешна  
WinSvc.CloseServiceHandle(hService);  
WinSvc.CloseServiceHandle(hSCMan);  
Result:= True;  
End;  
  
// Удаление регистрации драйвера  
Function TAT89DriverLoader.RemoveService : Boolean;  
var hSCMan, hService : SC_HANDLE;  
Begin  
    Result:= False;  
    // Удаление сервиса из реестра  
    hSCMan:= WinSvc.OpenSCManager(Nil,Nil,SC_MANAGER_ALL_ACCESS);
```

```
If hSCMan = 0 then Exit;
hService:= WinSvc.OpenService(hSCMan, DriverName, SERVICE_ALL_ACCESS);
// Ошибка открытия сервиса
If hService=0 then begin
    CloseServiceHandle(hSCMan);
    Exit;
End;
// Удалить
WinSvc.DeleteService(hService);
WinSvc.CloseServiceHandle(hService);
WinSvc.CloseServiceHandle(hSCMan);
Result:= True;
End;
// Старт сервиса драйвера
Function TAT89DriverLoader.StartService : Boolean;
var lpServiceArgVectors : PChar;
    hSCMan, hService : SC_HANDLE;
Begin
    Result:= False;

    // Старт зарегистрированного сервиса
    hSCMan := WinSvc.OpenSCManager(nil, nil, SC_MANAGER_CONNECT);
    If hSCMan = 0 then Exit;

    hService:= WinSvc.OpenService(hSCMan, DriverName, SERVICE_START);
    If hService = 0 then begin
        CloseServiceHandle(hSCMan);
        Exit;
    End;

    lpServiceArgVectors:=nil;
    WinSvc.StartService(hService, 0, lpServiceArgVectors);
    WinSvc.CloseServiceHandle(hService);
    WinSvc.CloseServiceHandle(hSCMan);
    Result:= True;
End;

// Останов сервиса драйвера
Function TAT89DriverLoader.StopService : Boolean;
```

```
var serviceStatus    : TServiceStatus;
    hSCMan, hService : SC_HANDLE;

Begin
    Result:= False;
    // Остановка сервиса
    hSCMan:= WinSvc.OpenSCManager(nil, nil, SC_MANAGER_CONNECT);
    If hSCMan = 0 then Exit;
    hService:= WinSvc.OpenService(hSCMan, DriverName, SERVICE_STOP);
    If hService = 0 then begin
        WinSvc.CloseServiceHandle(hSCMan);
        Exit;
    End;
    // Останов
    WinSvc.ControlService(hService, SERVICE_CONTROL_STOP, serviceStatus);
    WinSvc.CloseServiceHandle(hService);
    WinSvc.CloseServiceHandle(hSCMan);
    Result:= True;
End;

// Открытие драйвера. Параметр DevIndex передает номер
// устройства (согласно параметру m_Unit в листинге 14.4)
Function TAT89DriverLoader.Open(DevIndex : Integer) : Boolean;
Var DevName : String;
Begin
    If DevIndex <> -1 then DevName:= DriverPath + IntToStr(DevIndex)
    Else DevName:= DriverPath;

    hDevice:= CreateFile(@DevName[1],
        GENERIC_READ or GENERIC_WRITE,
        0,nil,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        0
    );

    Result:= hDevice <> INVALID_HANDLE_VALUE;
    If not Result then begin
        MessageDlg(SysErrorMessage(GetLastError), mtError, [mbOK], 0);
    End;
End;
```

```
// Закрыть дескриптор драйвера
Procedure TAT89DriverLoader.Close;
Begin
  If hDevice <> INVALID_HANDLE_VALUE then
    CloseHandle(hDevice);
  hDevice:= INVALID_HANDLE_VALUE;
End;

// Параметры вызова DeviceIoControl
Const
  FILE_DEVICE_UNKNOWN = $22;
  METHOD_BUFFERED       = 0;
  FILE_ANY_ACCESS       = $0000;

// Формирование CTL-кода
function Get_Ctl_Code(Nr: Integer): Integer;
begin
  Result:=
    (FILE_DEVICE_UNKNOWN shl 16) or
    (FILE_ANY_ACCESS      shl 14) or
    (Nr                    shl  2) or
    METHOD_BUFFERED;
end;

// Выполнение функции DeviceIoControl
Function TAT89DriverLoader.Func1 : Cardinal;
Var Res : Cardinal;
Begin
  If hDevice = INVALID_HANDLE_VALUE then Exit;
  DeviceIoControl(hDevice, Get_Ctl_Code($800),...);
End;

// Функция чтения 8 байт
Function TAT89DriverLoader.Read : String;
Var A : Array[1..8] of Byte;
    ReadBytes : Cardinal;
    i : Integer;
Begin
  // Инициализация буфера
  For i:= 1 to 8 do A[i]:= 0;
```

```

Result:= '';
If hDevice = INVALID_HANDLE_VALUE then Exit;

// Чтение
If not ReadFile(hDevice, A, 8, ReadBytes, nil) then
  MessageDlg(SysErrorMessage(GetLastError), mtError, [mbOK], 0);

// Шестнадцатеричная строка
For i:= 1 to 8 do
  Result:= Result + IntToHex(A[i],2)+' ';
End;

end.

```

Код тестовой программы, использующей модуль DrvLoader, показан в листинге 14.9, а вид формы и результат работы — на рис. 14.24 (для контроля мы запустили программу DebugView, показывающую отладочные сообщения драйвера).

Листинг 14.9. Тестовая программа чтения данных с USB-устройства

```

unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms,
  Dialogs, StdCtrls, DrvLoader, ExtCtrls, Spin;

type
  TForm1 = class(TForm)
    btnLoad: TButton;
    btnUnload: TButton;
    Bevel1: TBevel;
    btnOpen: TButton;
    btnClose: TButton;
    Bevel2: TBevel;
    btnFunc1: TButton;
    EOutput: TEdit;

```

```
    Button1: TButton;
    SpinEdit1: TSpinEdit;
    procedure btnLoadClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure btnUnloadClick(Sender: TObject);
    procedure btnOpenClick(Sender: TObject);
    procedure btnCloseClick(Sender: TObject);
    procedure btnFunc1Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
private
public
    Driver : TAT89DriverLoader;
end;

var
    Form1: TForm1;

implementation

{$R *.dfm}

// Создание компонента TDriverLoader
procedure TForm1.FormCreate(Sender: TObject);
begin
    Driver:= TAT89DriverLoader.Create;
end;

// Уничтожение компонента TDriverLoader
procedure TForm1.FormDestroy(Sender: TObject);
begin
    Driver.Free;
end;

// Регистрация сервиса в реестре и старт
procedure TForm1.btnLoadClick(Sender: TObject);
begin
    Driver.CreateService(ExtractFilePath(ParamStr(0)));
    Driver.StartService;
end;
```

```
// Останов сервиса и удаление из реестра
procedure TForm1.btnUnloadClick(Sender: TObject);
begin
    Driver.StopService;
    Driver.RemoveService;
end;

// Открытие драйвера
procedure TForm1.btnOpenClick(Sender: TObject);
begin
    If Driver.Open(SpinEdit1.Value)
    then MessageDlg('Драйвер открыт успешно', mtConfirmation, [mbOK], 0)
    else MessageDlg('Ошибка открытия драйвера', mtError, [mbOK], 0);
end;

// Закрытие драйвера
procedure TForm1.btnCloseClick(Sender: TObject);
begin
    Driver.Close;
end;

// Выполнение DeviceIoControl($800)
procedure TForm1.btnFunc1Click(Sender: TObject);
begin
    EOutput.Text:= IntToStr(Driver.Func1);
end;

// Чтение данных
procedure TForm1.Button1Click(Sender: TObject);
begin
    EOutput.Text:= Driver.Read;
end;

end.
```

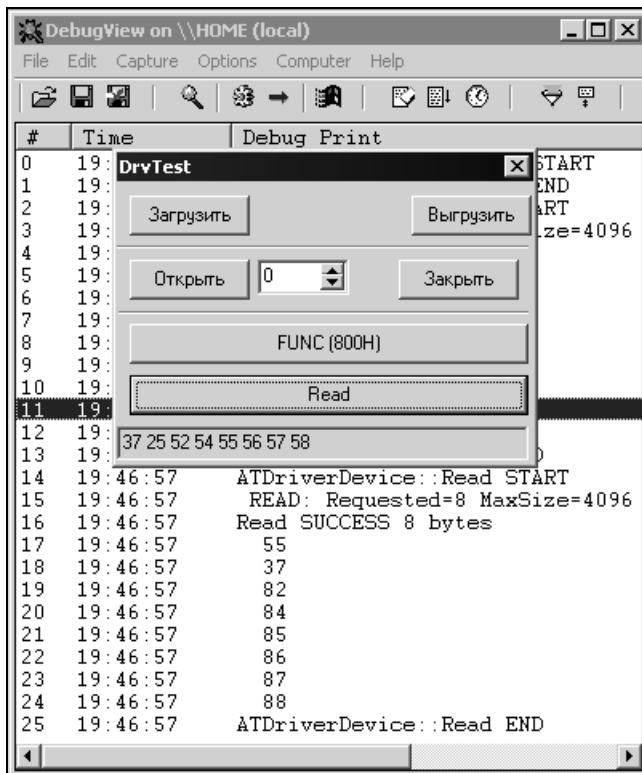



Рис. 14.24. Результат чтения данных с USB-устройства

14.2.9. Чтение данных с конечных точек других типов

В предыдущем разделе мы пользовались конечной точкой типа INTERRUPT. При необходимости можно легко заменить тип конечной точки. Листинг 14.10 показывает константы для дескриптора конфигурации конечной точки типа BULK, а листинг 14.11 — изменения в коде драйвера.

Листинг 14.10. Константы для конечной точки типа BULK

```
/* первая конечная точка */
#define EP_1_CONFIG      (BULK|EP_CONFIG_IN) /* конфигурация */
#define EP_1_ADDRESS     (1|EP_DIRECT_IN)    /* адрес */
#define EP_1_ATTRIBUTES  BULK                 /* атрибуты */
```

Листинг 14.11. Изменение в коде драйвера для чтения конечной точки типа BULK

```
NTSTATUS ATDriverDevice::Read(KIrp I)
{
    .....
    PURB pUrb = m_Endpoint1IN.BuildBulkTransfer(
        pBuffer,          // Буфер
        dwTotalSize,      // Сколько байт читать
        TRUE,             // Направление (TRUE = IN)
        NULL,
        FALSE,            // Разрешить короткие чтения
        NULL
    );
    .....
    pUrb->UrbBulkOrInterruptTransfer.TransferFlags =
        (USB_D_TRANSFER_DIRECTION_IN | USB_D_SHORT_TRANSFER_OK);
    .....
    dwBytesRead = pUrb->UrbBulkOrInterruptTransfer.TransferBufferLength;
    .....
}
```

Для использования конечных точек типа CONTROL достаточно изменить метод формирования пакета запроса:

```
PURB pUrb = m_Endpoint1IN.BuildControlTransfer(
    pBuffer,          // Буфер
    dwTotalSize,      // Сколько данных читать?
    TRUE             // Направление (TRUE = IN)
);
```

14.2.10. "Чистый" USB-драйвер

Использование Driver Studio избавило нас от множества рутинной работы, однако не стоит забывать, что эта программа стоит довольно существенных денег, в то время как использование Windows DDK доступно всем разработчикам.

Основную структуру WDM-драйвера мы уже описали в *главе 9*. В этом разделе мы приведем схему разработки такого драйвера для шины USB.

Общий "план" драйвера показан на рис. 14.25.

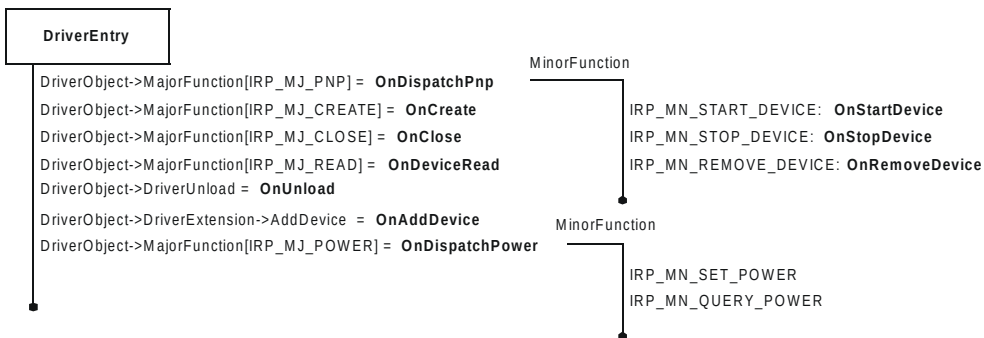


Рис. 14.25. Общий "план" USB-драйвера

Основными обработчиками драйвера являются:

- ❑ **DriverEntry** — точка входа в драйвер;
- ❑ **OnAddDevice** — вызывается при добавлении нового устройства;
- ❑ **OnCreate** — вызывается при создании устройства;
- ❑ **OnClose** — вызывается при закрытии устройства;
- ❑ **OnUnload** — вызывается при выгрузке драйвера;
- ❑ **OnDispatchPower** — диспетчер управления энергопотреблением;
- ❑ **OnDispatchPnp** — диспетчер для обработки командами PnP:
 - **OnStartDevice** — старт устройства;
 - **OnStopDevice** — останов устройства;
 - **OnRemoveDevice** — удаление устройства;
- ❑ **OnDeviceRead** — вызывается при обработке функции ReadFile.

Регистрация основных обработчиков производится в процедуре **DriverEntry** (листинг 14.12).

Листинг 14.12. Регистрация обработчиков USB-драйвера

```

NTSTATUS
DriverEntry(
    IN PDRIVER_OBJECT DriverObject,
    IN PUNICODE_STRING RegistryPath
)
{
    NTSTATUS ntStatus = STATUS_SUCCESS;
  
```

```
// Основные обработчики драйвера
DriverObject->MajorFunction[IRP_MJ_CREATE]= OnCreate;
DriverObject->MajorFunction[IRP_MJ_CLOSE ]= OnClose;
DriverObject->MajorFunction[IRP_MJ_READ  ]= OnDeviceRead;
DriverObject->MajorFunction[IRP_MJ_PNP   ]= OnDispatchPnp;
DriverObject->MajorFunction[IRP_MJ_POWER ]= OnDispatchPower;
DriverObject->DriverUnload                = OnUnload;
DriverObject->DriverExtension->AddDevice  = OnAddDevice;

return ntStatus;
}
```

В отличие от NT-драйвера, в WDM-драйвере создание объекта устройства и регистрация символьных имен должна выполняться внутри процедуры OnAddDevice (листинг 14.13).

Листинг 14.13. Функция OnAddDevice USB-драйвера

```
NTSTATUS OnAddDevice(
    IN PDRIVER_OBJECT DriverObject,
    IN PDEVICE_OBJECT PhysicalDeviceObject
)
{
    NTSTATUS          ntStatus = STATUS_SUCCESS;
    PDEVICE_OBJECT    deviceObject = NULL;
    PDEVICE_EXTENSION pdx;

    WCHAR NameBuffer[] = L"\\Device\\" DEVICE_NAME_STRING;
    WCHAR DOSNameBuffer[] = L"\\DosDevices\\" DEVICE_NAME_STRING;
    UNICODE_STRING uniNameString, uniDOSString;

    // Создание буферов для имен
    RtlInitUnicodeString(&uniNameString, NameBuffer);
    RtlInitUnicodeString(&uniDOSString, DOSNameBuffer);
    DbgPrint("UniName=%s DosName=%s", uniDOSString, uniNameString);

    // Инициализация объекта драйвера
    ntStatus = IoCreateDevice(
        DriverObject,
```

```

        sizeof (DEVICE_EXTENSION),
        &uniNameString,
        FILE_DEVICE_UNKNOWN,
        0,
        FALSE,
        &deviceObject
    );

    // Создание символического имени драйвера
    ntStatus = IoCreateSymbolicLink(&uniDOSString, &uniNameString);

    // Инициализация блока данных объекта устройства
    DbgPrint("Init device extension");
    pdx = (PDEVICE_EXTENSION) (deviceObject->DeviceExtension);
    pdx->OpenHandles = 0;

    // Драйвер будет использовать прямой в/в для запросов
    // чтения и записи
    deviceObject->Flags |= DO_DIRECT_IO;

    // Сохраняем ссылку на драйвер нижнего уровня. Ему мы будем
    // пересылать запросы на в/в
    pdx->StackDeviceObject =
        IoAttachDeviceToDeviceStack(deviceObject, PhysicalDeviceObject);

    return ntStatus;
}

```

В WDM предусмотрен довольно удобный механизм хранения данных, относящихся к конкретному экземпляру драйвера: при создании объекта устройства в функцию `IoCreateDevice` передается размер блока пользовательских данных (`DeviceExtension`). В любом другом обработчике можно легко получить указатель на этот буфер с помощью следующего кода:

```
PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION )fdo->DeviceExtension;
```

Внутренняя структура этих данных — забота программиста, драйвер только лишь создает блок памяти нужного размера. В нашем примере мы используем описание, показанное в листинге 14.14.

Листинг 14.14. Структура DEVICE_EXTENSION

```
typedef struct _DEVICE_EXTENSION
{
    // Объект устройства в IRP-стеке
    PDEVICE_OBJECT StackDeviceObject;
    // Число устройств для этого драйвера
    ULONG          OpenHandles;
    // TRUE, если устройство стартовано
    BOOLEAN        Started;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;
```

Обработчики `OnCreate` и `OnClose` в общем случае могут не выполнять никаких специальных действий за исключением управления счетчиком копий драйвера (листинг 14.15).

Листинг 14.15. Процедуры OnCreate и OnClose

```
NTSTATUS OnCreate(
    IN PDEVICE_OBJECT fdo,
    IN PIRP Irp
)
{
    NTSTATUS ntStatus;
    PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION )fdo->DeviceExtension;

    // счетчик открытых устройств
    pdx->OpenHandles++;

    Irp->IoStatus.Status      = STATUS_SUCCESS;
    Irp->IoStatus.Information = 0;
    ntStatus = Irp->IoStatus.Status;
    IoCompleteRequest(Irp, IO_NO_INCREMENT);
    return ntStatus;
}

NTSTATUS OnClose(
    IN PDEVICE_OBJECT fdo,
```

```

    IN PIRP Irp
    )
{
    NTSTATUS ntStatus;
    PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION )fdo->DeviceExtension;

    // счетчик открытых устройств
    pdx->OpenHandles--;

    Irp->IoStatus.Status = STATUS_SUCCESS;
    Irp->IoStatus.Information = 0;
    ntStatus = Irp->IoStatus.Status;
    IoCompleteRequest (Irp, IO_NO_INCREMENT);
    return ntStatus;
}

```

Обработчик `OnUnload` может выполнять некоторые действия по освобождению памяти, буферов и т. д., а диспетчер энергопотребления `OnDispatchPower` используется только в случае, если устройство поддерживает соответствующие интерфейсы. В нашем случае эти обработчики не выполняют никаких специальных действий (листинг 14.16).

Листинг 14.16. Обработчики `OnUnload` и `OnDispatchPower`

```

VOID OnUnload(
    IN PDRIVER_OBJECT DriverObject
    )
{
}

NTSTATUS OnDispatchPower(
    IN PDEVICE_OBJECT fdo,
    IN PIRP           Irp
    )
{
    PIO_STACK_LOCATION irpStack, nextStack;
    PDEVICE_EXTENSION pdx = fdo->DeviceExtension;
    NTSTATUS ntStatus;
}

```

```

Irp->IoStatus.Status = STATUS_SUCCESS;
Irp->IoStatus.Information = 0;
irpStack = IoGetCurrentIrpStackLocation(Irp);

nextStack = IoGetNextIrpStackLocation(Irp);
RtlCopyMemory(nextStack, irpStack, sizeof(IO_STACK_LOCATION));

PoStartNextPowerIrp(Irp);
ntStatus = PoCallDriver(pdx->StackDeviceObject, Irp);

if (ntStatus == STATUS_PENDING)
{
    IoMarkIrpPending(Irp);
}

return ntStatus;
}

```

Основную работу по поддержке PnP производит процедура-диспетчер `OnDispatchPnp`, распределяющая вызовы в соответствии с кодом `MinorFunction` (листинг 14.17).

Листинг 14.17. Диспетчер PnP-запросов `OnDispatchPnp`

```

NTSTATUS OnDispatchPnp(
    IN PDEVICE_OBJECT fdo,
    IN PIRP           Irp
)
{
    PIO_STACK_LOCATION irpStack;
    PDEVICE_EXTENSION pdx = fdo->DeviceExtension;
    ULONG fcn;
    NTSTATUS ntStatus;

    /* Получаем текущую позицию в стеке драйверов */
    irpStack = IoGetCurrentIrpStackLocation (Irp);
    /* Номер функции */
    fcn = irpStack->MinorFunction;

```



```
switch (fcn)
{
    /* Обработка старта устройства */
    case IRP_MN_START_DEVICE:
    {
        ntStatus = OnStartDevice(fdo);
        if (ntStatus == STATUS_SUCCESS)
        {
            pdx->Started = TRUE;
        }
        break;
    }
    /* Обработка останова устройства */
    case IRP_MN_STOP_DEVICE:
    {
        // Сначала передаем запрос драйверу ниже по стеку
        IoSkipCurrentIrpStackLocation(Irp);
        IoCallDriver(pdx->StackDeviceObject, Irp);
        // Отрабатываем останов нашего устройства
        ntStatus = OnStopDevice(fdo);
        break;
    }
    /* Удаление устройства из системы */
    case IRP_MN_REMOVE_DEVICE:
    {
        ntStatus = OnRemoveDevice(fdo, Irp);
        break;
    }
    // Все остальные запросы передаем драйверу дальше по стеку
    default:
    {
        IoSkipCurrentIrpStackLocation(Irp);
        ntStatus = IoCallDriver(pdx->StackDeviceObject, Irp);
    }
}

return ntStatus;
}
```

Для непосредственного выполнения запросов к устройству необходимо выполнить следующие действия:

1. Выделить блок нестраничной памяти (non paged memory) с помощью вызова `ExAllocatePool`.
2. Сформировать запрос с помощью одной из функций формирования запросов (например, `UsbBuildGetDescriptorRequest`, `UsbBuildSelectConfigurationRequest`, `UsbBuildInterruptOrBulkTransferRequest` и т. п.).
3. Передать запрос USB-D-драйверу с помощью функции `DoCallUSBD` (листинг 14.18).
4. Обработать результат запроса.
5. Освободить память.

В случае асинхронного вызова два последних действия переносятся в функцию, вызываемую при завершении выполнения операции.

Листинг 14.18. Передача запроса USB-D-драйверу

```
NTSTATUS DoCallUSBD(
    IN PDEVICE_OBJECT fdo,
    IN PURB Urb
)
{
    NTSTATUS ntStatus, status = STATUS_SUCCESS;
    PDEVICE_EXTENSION pdx;
    PIRP irp;
    KEVENT event;
    IO_STATUS_BLOCK ioStatus;
    PIO_STACK_LOCATION nextStack;

    pdx = fdo->DeviceExtension;

    // объект синхронизации
    KeInitializeEvent(&event, NotificationEvent, FALSE);

    irp = IoBuildDeviceIoControlRequest(
        IOCTL_INTERNAL_USB_SUBMIT_URB,
        pdx->StackDeviceObject,
        NULL,
        0,
```

```
        NULL,  
        0,  
        TRUE, /* INTERNAL */  
        &event,  
        &ioStatus  
    );  
  
    // получение следующей позиции в стеке драйверов  
    nextStack = IoGetNextIrpStackLocation(irp);  
  
    // Формирование параметров для вызова  
    nextStack->Parameters.Others.Argument1 = Urb;  
    // Вызов  
    ntStatus = IoCallDriver(pdx->StackDeviceObject, irp);  
  
    // Если запрос еще выполняется...  
    if (ntStatus == STATUS_PENDING)  
    {  
        status = KeWaitForSingleObject(  
            &event,  
            Suspended,  
            KernelMode,  
            FALSE,  
            NULL);  
    } else {  
        ioStatus.Status = ntStatus;  
    }  
    // Обработка результата запроса  
    ntStatus = ioStatus.Status;  
  
    if (NT_SUCCESS(ntStatus))  
    {  
        if (!(USB_SUCCESS(Urb->UrbHeader.Status)))  
            ntStatus = STATUS_UNSUCCESSFUL;  
    }  
  
    return ntStatus;  
}
```

Обработчик `OnStartDevice` может содержать код, работающий с нулевой конечной точкой, код получения дескрипторов и код конфигурирования устройства (листинг 14.19).

Листинг 14.19. Обработчик `OnStartDevice`

```
NTSTATUS OnStartDevice(
    IN PDEVICE_OBJECT fdo
)
{
    PDEVICE_EXTENSION pdx;
    NTSTATUS ntStatus;
    USB_DEVICE_DESCRIPTOR deviceDescriptor = NULL;
    PURB urb;
    ULONG size;

    pdx = fdo->DeviceExtension;
    urb = ExAllocatePool( NonPagedPool,
        sizeof(struct _URB_CONTROL_DESCRIPTOR_REQUEST));

    size = sizeof(USB_DEVICE_DESCRIPTOR);
    deviceDescriptor = ExAllocatePool(NonPagedPool, size);

    UsbBuildGetDescriptorRequest(
        urb,
        (USHORT) sizeof (struct _URB_CONTROL_DESCRIPTOR_REQUEST),
        USB_DEVICE_DESCRIPTOR_TYPE,
        0,
        0,
        deviceDescriptor,
        NULL,
        size,
        NULL
    );

    // Передача запроса на выполнение
    ntStatus = DoCallUSBD(fdo, urb);
}
```

```
// Отображение дескриптора
if (NT_SUCCESS(ntStatus)) {
    DbgPrint("Device Descriptor:");
    DbgPrint("bLength          %d ", deviceDescriptor->bLength);
    DbgPrint("bDescriptorType      0x%x", deviceDescriptor->bDescriptorType);
    DbgPrint("bcdUSB                0x%x", deviceDescriptor->bcdUSB);
    DbgPrint("bDeviceClass           0x%x", deviceDescriptor->bDeviceClass);
    DbgPrint("bDeviceSubClass        0x%x", deviceDescriptor->bDeviceSubClass);
    DbgPrint("bDeviceProtocol        0x%x", deviceDescriptor->bDeviceProtocol);
    DbgPrint("bMaxPacketSize0        0x%x", deviceDescriptor->bMaxPacketSize0);
    DbgPrint("idVendor                0x%x", deviceDescriptor->idVendor);
    DbgPrint("idProduct              0x%x", deviceDescriptor->idProduct);
    DbgPrint("bcdDevice              0x%x", deviceDescriptor->bcdDevice);
    DbgPrint("iManufacturer          0x%x", deviceDescriptor->iManufacturer);
    DbgPrint("iProduct                0x%x", deviceDescriptor->iProduct);
    DbgPrint("iSerialNumber          0x%x", deviceDescriptor->iSerialNumber);
}

// Освободить занятую память
ExFreePool(deviceDescriptor);
ExFreePool(urb);

if (NT_SUCCESS(ntStatus)) {
    // Конфигурируем устройство
    ntStatus = OnConfigureDevice(fdo);
}

return ntStatus;
}
```

Конфигурирование устройства (процедура OnConfigureDevice) состоит из следующих действий (листинг 14.20):

- ❑ передача запроса GET_CONFIGURATION с минимальным размером буфера (для получения нужного размера буфера);
- ❑ получение полного дескриптора;
- ❑ создание конечных точек;
- ❑ конфигурирование конечных точек.

Листинг 14.20. Конфигурирование устройства

```

NTSTATUS
OnConfigureDevice(
    IN  PDEVICE_OBJECT fdo
)
{
    PDEVICE_EXTENSION pdx;
    NTSTATUS ntStatus;
    PURB urb = NULL;
    ULONG size;
    USB_CONFIGURATION_DESCRIPTOR configurationDescriptor = NULL;
    UCHAR alternateSetting, MyInterfaceNumber;
    USB_INTERFACE_INFORMATION interfaceObject;
    USB_INTERFACE_LIST_ENTRY  interfaceList;

    pdx = fdo->DeviceExtension;

// Память для URB
    urb = ExAllocatePool(NonPagedPool,
                        sizeof(struct _URB_CONTROL_DESCRIPTOR_REQUEST));

// Получить только дескриптор конфигурации
    size = sizeof(USB_CONFIGURATION_DESCRIPTOR) + 16;
    configurationDescriptor = ExAllocatePool(NonPagedPool, size);

    UsbBuildGetDescriptorRequest(urb,
        (USHORT) sizeof (struct _URB_CONTROL_DESCRIPTOR_REQUEST),
        USB_CONFIGURATION_DESCRIPTOR_TYPE,
        0,
        0,
        configurationDescriptor,
        NULL,
        sizeof (USB_CONFIGURATION_DESCRIPTOR),
        NULL
    );

    ntStatus = DoCallUSBD(fdo, urb);
}

```

```
// Определение нужного размера буфера
size = configurationDescriptor->wTotalLength + 16;

// Освободить старый буфер и отвести новый, нужного размера
ExFreePool(configurationDescriptor);
configurationDescriptor = NULL;
configurationDescriptor = ExAllocatePool(NonPagedPool, size);

// Получение полного дескриптора конфигурации
UsbBuildGetDescriptorRequest(urb,
    (USHORT) sizeof (struct _URB_CONTROL_DESCRIPTOR_REQUEST),
    USB_CONFIGURATION_DESCRIPTOR_TYPE,
    0,
    0,
    configurationDescriptor,
    NULL,
    size,
    NULL
);
ntStatus = DoCallUSBD(fdo, urb);

// Получение конфигурации
interfaceList.InterfaceDescriptor =
    USBD_ParseConfigurationDescriptorEx(
        ConfigurationDescriptor,
        ConfigurationDescriptor,
        -1, -1, -1, -1, -1
    );

// Создание запроса для получения конфигурации
urb = USBD_CreateConfigurationRequestEx(
    ConfigurationDescriptor, &interfaceList
);

// Получение указателя на буфер описания интерфейса
interfaceObject =
    (PUSBD_INTERFACE_INFORMATION)
    (&(urb->UrbSelectConfiguration.Interface));
```

```
// Конфигурирование конечных точек интерфейса
for (j=0; j<interfaceList[0].InterfaceDescriptor->bNumEndpoints; j++)
{
    PUSBD_PIPE_INFORMATION pipe;
    pipe = &interfaceObject->Pipes[j];

    pipe->MaximumTransferSize = 1024; // установить при необходимости

    DbgPrint("PipeType 0x%x\n", pipe->PipeType);
    DbgPrint("EndpointAddress 0x%x\n", pipe ->EndpointAddress);
    DbgPrint("MaxPacketSize 0x%x\n", pipe ->MaximumPacketSize);
    DbgPrint("Interval 0x%x\n", pipe ->Interval);
    DbgPrint("Handle 0x%x\n", pipe ->PipeHandle);
    DbgPrint("MaximumTransferSize 0x%x\n", pipe ->MaximumTransferSize);
}

ntStatus = DoCallUSBD(fdo, urb);

ExFreePool(urb);
ExFreePool(configurationDescriptor);

return ntStatus;
}
```

В обработчике OnStopDevice производится передача запроса SET_CONFIGURATION с нулевым номером конфигурации, т. е. сброс конфигурации (листинг 14.21).

Листинг 14.21. Обработчик OnStopDevice

```
NTSTATUS OnStopDevice(
    IN PDEVICE_OBJECT fdo
)
{
    PDEVICE_EXTENSION pdx;
    NTSTATUS ntStatus = STATUS_SUCCESS;
    PURB urb;
    ULONG size;

    pdx = fdo->DeviceExtension;
```



```

size= sizeof(struct _URB_SELECT_CONFIGURATION);
urb = ExAllocatePool(NonPagedPool, size);

if (urb)
{
    NTSTATUS status;

    UsbBuildSelectConfigurationRequest(urb, (USHORT) size, NULL);
    status = DoCallUSBD(fdo, urb);
    ExFreePool(urb);
} else {
    ntStatus = STATUS_NO_MEMORY;
}

return ntStatus;
}

```

Обработчик `OnRemoveDevice` содержит код, "обратный" коду `OnAddDevice`. В этом обработчике удаляется символьное имя устройства и освобождается объект устройства (листинг 14.22).

Листинг 14.22. Обработчик `OnRemoveDevice`

```

NTSTATUS OnRemoveDevice(
    IN PDEVICE_OBJECT fdo,
    IN PIRP Irp
)
{
    NTSTATUS ntStatus;
    PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
    ULONG i;

    WCHAR NameBuffer[] = L"\\Device\\" DEVICE_NAME_STRING;
    WCHAR DOSNameBuffer[] = L"\\DosDevices\\" DEVICE_NAME_STRING;
    UNICODE_STRING uniNameString, uniDOSString;

    // Создание буферов для имен
    RtlInitUnicodeString(&uniNameString, NameBuffer);

```

```

RtlInitUnicodeString(&uniDOSString , DOSNameBuffer);
// Удаление символического имени драйвера
ntStatus = IoDeleteSymbolicLink(&uniNameString);

IoDetachDevice(pdx->StackDeviceObject);
IoDeleteDevice(fdo);

IoSkipCurrentIrpStackLocation(Irp);
ntStatus = IoCallDriver(pdx->StackDeviceObject, Irp);

return ntStatus;
}

```

Чтение данных производится в обработчике OnDeviceRead (листинг 14.23).

Листинг 14.23. Чтение данных с устройства

```

NTSTATUS OnDeviceRead(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP pIrp
)
{
    NTSTATUS          ntStatus    = STATUS_SUCCESS;
    USB_PIPE_INFORMATION pipeInfo = NULL;
    USB_PIPE_HANDLE    pipeHandle = NULL;
    PURB               urb        = NULL;
    ULONG              urbSize    = 0;

    pipeInfo = ... ; // одна из конечных точек
    pipeHandle = pipeInfo->PipeHandle;

    urbSize = sizeof(struct _URB_BULK_OR_INTERRUPT_TRANSFER);

    urb = ExAllocatePool(NonPagedPool, urbSize);

    transferFlags = USB_SHORT_TRANSFER_OK;
    if (USB_ENDPOINT_DIRECTION_IN(pipeInfo->EndpointAddress))
        transferFlags |= USB_TRANSFER_DIRECTION_IN;
}

```

```
// Формирование запроса
UsbBuildInterruptOrBulkTransferRequest(
    urb,
    (USHORT) urbSize,
    pipeHandle,
    NULL,
    Irp->MdlAddress,
    bufferLength,
    transferFlags,
    NULL
);

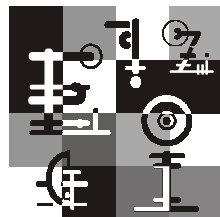
// Выполнение запроса
ntStatus = DoCallUSBDFdo, urb);

if (NT_SUCCESS(ntStatus))
{
    Irp->IoStatus.Information =
        urb->UrbBulkOrInterruptTransfer.TransferBufferLength;
}

// Освободить память
ExFreePool(urb);

// Вернуть результат запроса
return ntStatus;
}
```

Глава 15



Использование микросхем FTDI

Должен ли я отказаться от хорошего обеда лишь потому, что не понимаю процесса пищеварения?

О. Хэвисайд

Компания FTDI предоставляет очень удобный способ подключения интерфейса USB, позволяющий практически ничего не знать о внутренней организации USB. Микросхемы FT232 и FT245 представляют собой преобразователи USB в последовательный и параллельный интерфейсы соответственно. Специальные драйверы, доступные для различных операционных систем (см. разд. 12.3.6), организуют в системе обычный последовательный порт.

15.1. Функциональная схема FT232BM

На рис. 15.1 показана функциональная схема FT232BM. Ее основа — приемопередатчики обоих интерфейсов. Блок UART снабжен полным набором сигнальных цепей стандарта RS-232, приемопередатчик USB — всего двумя информационными выводами USBDP и USBDM, образующими двунаправленный канал передачи данных. Блок последовательного контроллера SIE (SIE, Serial Interface Engine) 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 хост-контроллера (компьютера). Через него же управляют режимом работы UART. Предусмотрены два FIFO-буфера промежуточного хранения данных на прием и на передачу. Управление FIFO возложено на соответствующий контроллер.

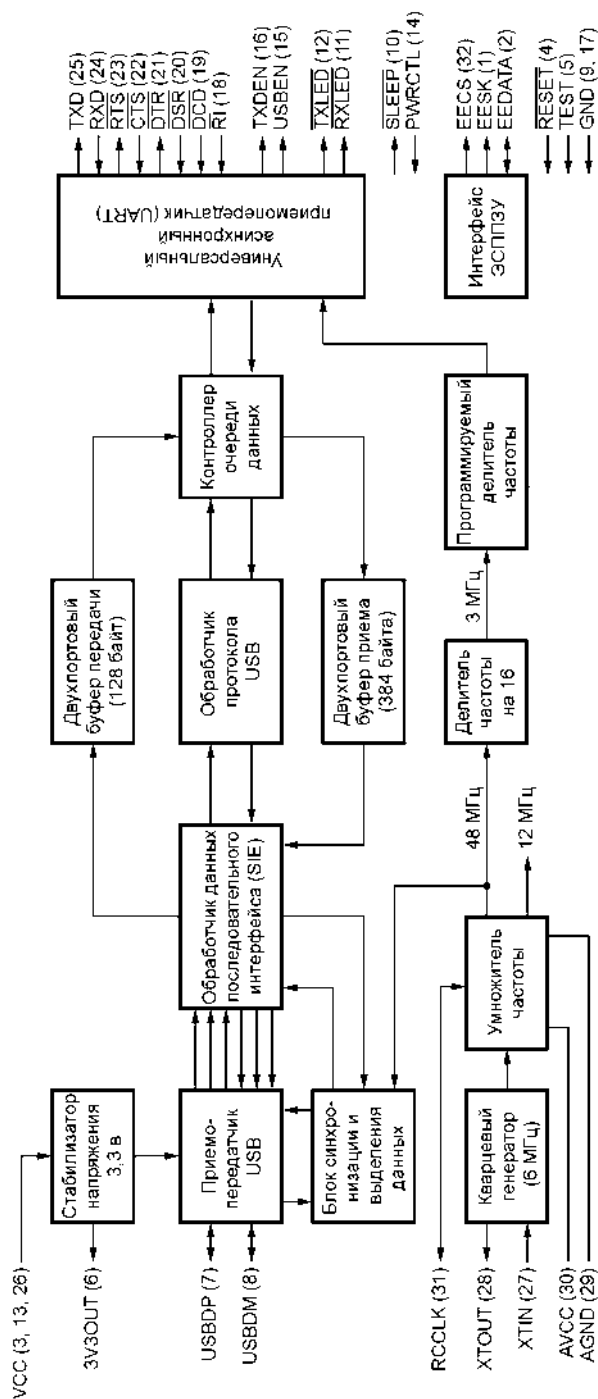


Рис. 15.1. Функциональная схема FT232BM

Задающий генератор микросхемы работает от внешнего кварцевого или керамического резонатора на 6 МГц. Далее его частоту умножают на 8 (до 48 МГц). Тактовую частоту UART получают из 48 МГц в два приема: делением на 16, затем — до нужного значения с помощью программируемого делителя. Контроллер UART может работать со скоростью от 300 Бод до 2 МБод, однако фактически достижимая максимальная скорость зависит от применяемой совместно с FT232BM микросхемы-преобразователя уровней интерфейсных сигналов.

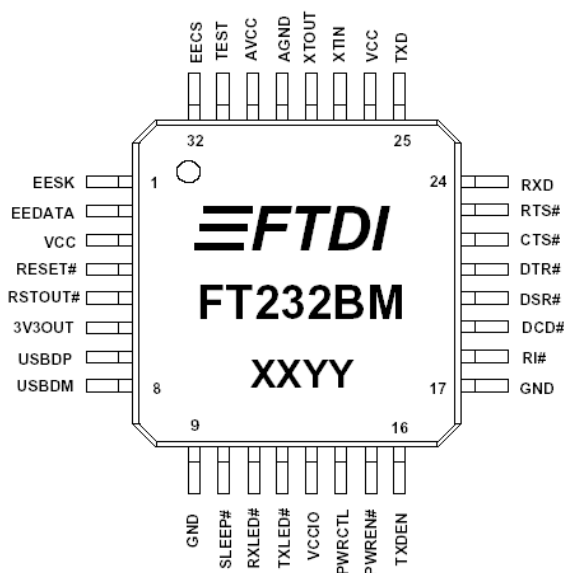


Рис. 15.2. Расположение выводов микросхемы FT232BM

Выводы EECS, EESK, EEDATA (рис. 15.2) микросхемы FT232BM 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 связано несколько устройств на микросхемах FT232BM. Особенно важен серийный номер, т. к. программный драйвер полагается на его уникальность, ассоциируя тот или иной виртуальный COM-порт с конкретным устройством. Если ПЗУ отсутствует, к компьютеру можно подключить только одно образующее виртуальный COM-порт устройство.

Низким уровнем на входе RESET микросхему FT232BM приводят в исходное состояние. К выводу RCCLK должна быть присоединена RC-цепь, поддерживающая начало работы микросхемы на время, достаточное для "раскачки" кварцевого резонатора, подключенного к выводам XTIN, XTOUT. Вход TEST используют только в отладочном режиме. При обычной работе он должен быть соединен с общим проводом (GND).

Имеется несколько вспомогательных выходов. Высокий уровень на выходе USBEN сигнализирует о завершении процесса инициализации микросхемы по USB. Если некоторое время обмена данными не происходит, микросхема автоматически переходит в "спящий режим", о чем свидетельствует низкий уровень на выходе SLEEP. Аналогичные уровни на выходах TXLED и RXLED показывают, что идет соответственно передача или прием данных. Сигнал с выхода TXDEN предназначен для управления приемопередатчиком интерфейса RS-485. Его уровень — высокий, когда по линии TXD идет передача данных.

Напряжение питания микросхемы FT232BM (VCC) — 4,4...5,25 В, потребляемый ток — не более 50 мА в рабочем и 250 мкА в спящем режиме. Если микросхему питают напряжением, поступающим по USB, ее вывод 14 (PWRCTL) необходимо соединить с общим проводом (GND), если устройство имеет собственный источник питания — с цепью VCC. Логические выходы микросхемы рассчитаны на ток до 4 мА (вытекающий) и до 8 мА (втекающий).

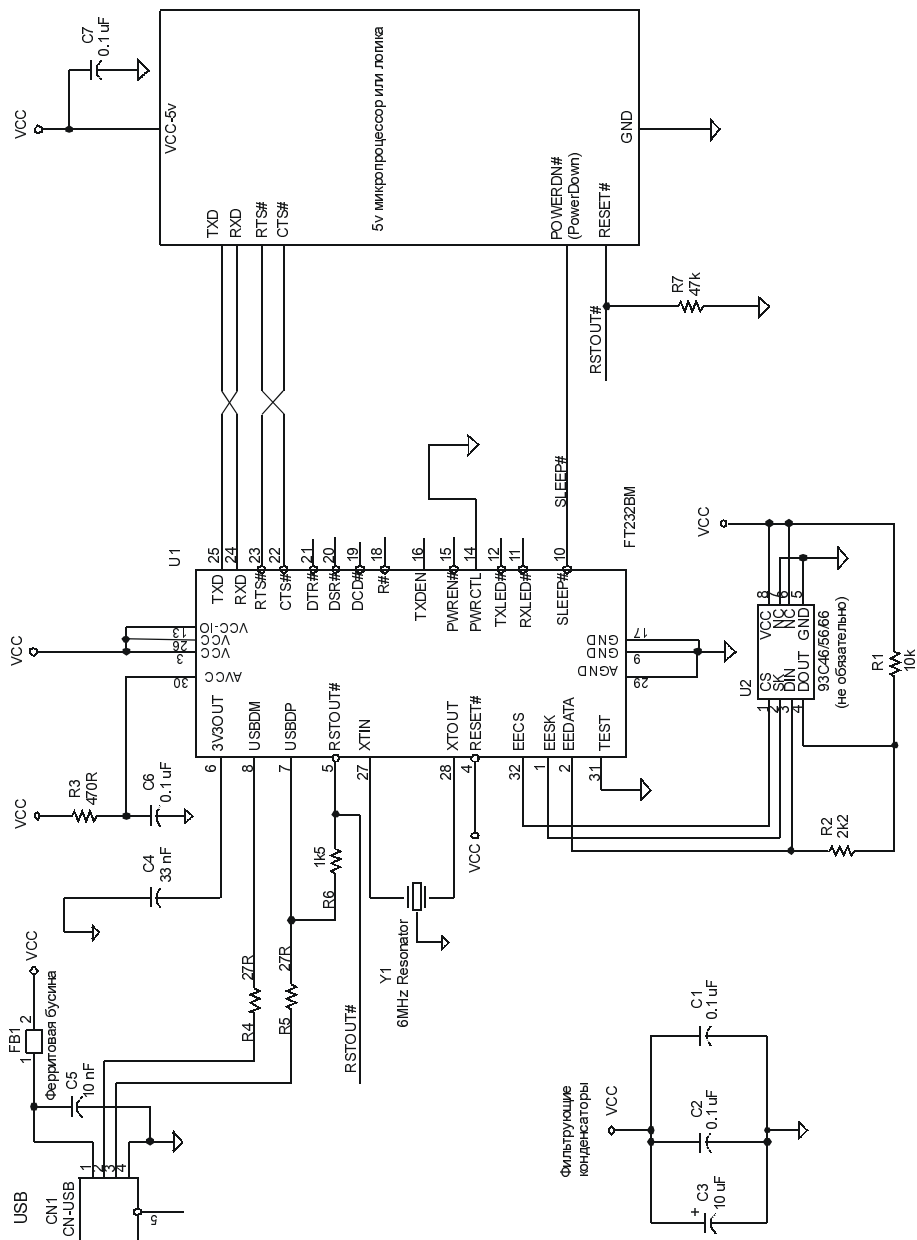
15.2. Схемотехника FT232BM

Документ FT232BM Designers Guide, доступный на сайте компании FTDI, содержит несколько примеров использования этих микросхем. Мы приведем только две из них. Первая схема (рис. 15.3) показывает подключение FT232BM с питанием от USB-шины и выходом на 5-вольтовый приемопередатчик (логика или микропроцессор).

На рис. 15.4 показана схема подключения FT232BM с внешним питанием.

15.3. Функции D2XX

Драйверы виртуального последовательного порта (см. разд. 12.3.6) позволяют работать с USB-интерфейсом через обычные функции COM-портов. Существует и альтернативное решение. Динамическая библиотека FTD2XX.dll (см. разд. 12.3.6) предоставляет набор функций для прямого взаимодействия с микросхемами FTDI (листинг 15.1).



15.3. Схема включения FT232 с питанием от шины

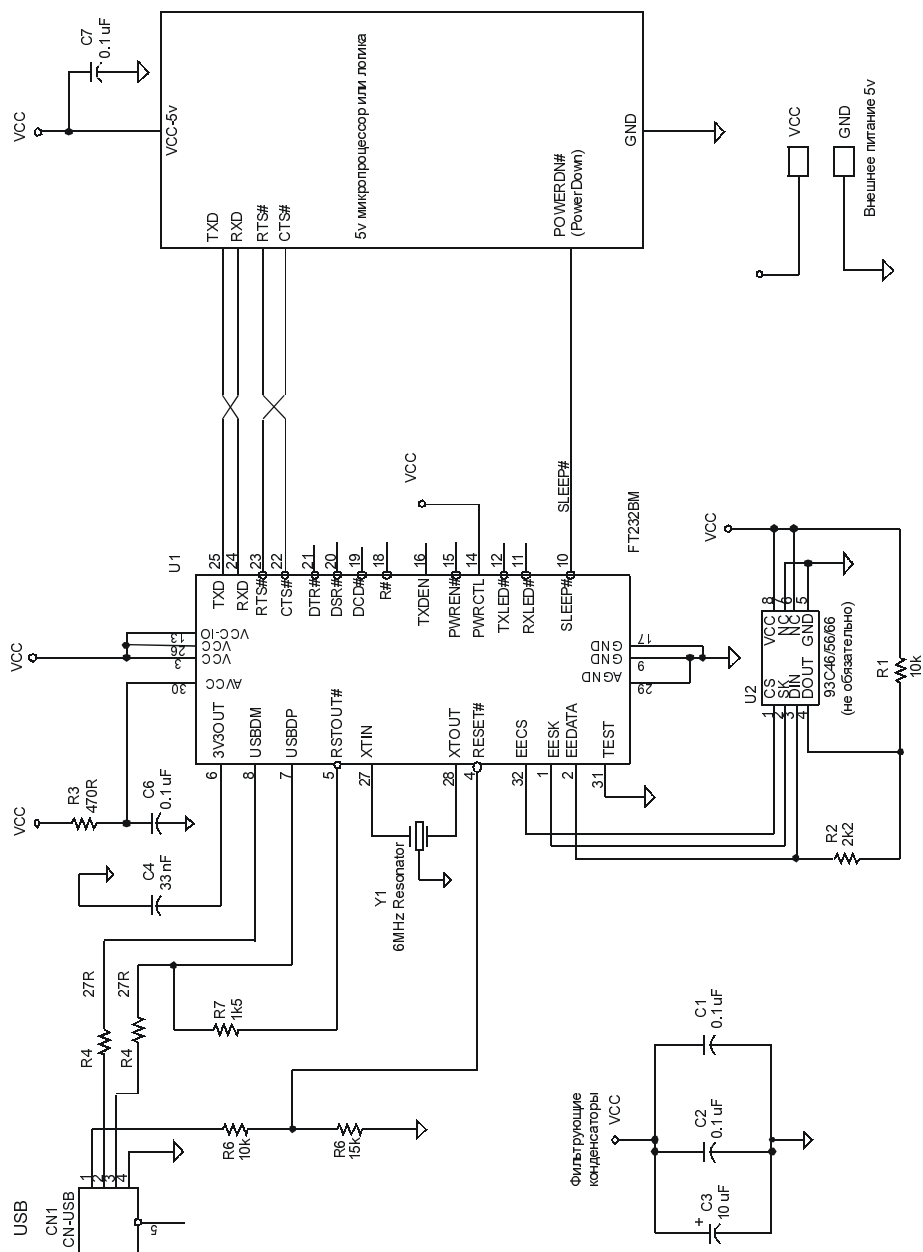


Рис. 15.4. Схема включения FT232 с внешним питанием

Листинг 15.1. Функции D2XX

```
// Открыть устройство
function FT_Open(PVDevice:Integer; ftHandle:Pointer ) : FT_Result ;
stdcall ; External FT_DLL_Name name 'FT_Open';

// Закрыть устройство
function FT_Close(ftHandle:Dword) : FT_Result ; stdcall ; External
FT_DLL_Name name 'FT_Close';

// Чтение блока данных
function FT_Read(ftHandle:Dword; FTInBuf : Pointer; BufferSize : LongInt;
ResultPtr : Pointer ) : FT_Result ; stdcall ; External FT_DLL_Name name
'FT_Read';

// Передача блока данных
function FT_Write(ftHandle:Dword; FTOutBuf : Pointer; BufferSize : Long-
Int; ResultPtr : Pointer ) : FT_Result ; stdcall ; External FT_DLL_Name
name 'FT_Write';

// Установка скорости обмена
function FT_SetBaudRate(ftHandle:Dword;BaudRate:Dword) : FT_Result ;
stdcall ; External FT_DLL_Name name 'FT_SetBaudRate';

function
// Установка характеристик линии обмена (длина байта, число стоп-бит
// и четность)
FT_SetDataCharacteristics(ftHandle:Dword;WordLength,StopBits,Parity:Byte)
: FT_Result ; stdcall ; External FT_DLL_Name name
'FT_SetDataCharacteristics';

// Управление линией Xon/Xoff
function
FT_SetFlowControl(ftHandle:Dword;FlowControl:Word;XonChar,XoffChar:Byte)
: FT_Result ; stdcall ; External FT_DLL_Name name 'FT_SetFlowControl';

// Сброс устройства
function FT_ResetDevice(ftHandle:Dword) : FT_Result ; stdcall ; External
FT_DLL_Name name 'FT_ResetDevice';

// Установка DTR в 1
function FT_SetDtr(ftHandle:Dword) : FT_Result ; stdcall ; External
FT_DLL_Name name 'FT_SetDtr';

// Установка DTR в 0
function FT_ClrDtr(ftHandle:Dword) : FT_Result ; stdcall ; External
FT_DLL_Name name 'FT_ClrDtr';

// Установка RTS в 1
function FT_SetRts(ftHandle:Dword) : FT_Result ; stdcall ; External
FT_DLL_Name name 'FT_SetRts';

// Установка RTS в 0
function FT_ClrRts(ftHandle:Dword) : FT_Result ; stdcall ; External
FT_DLL_Name name 'FT_ClrRts';
```

```
// Получение состояния линий модема
function FT_GetModemStatus(ftHandle:Dword;ModemStatus:Pointer) :
FT_Result ; stdcall ; External FT_DLL_Name name 'FT_GetModemStatus';
function
// Установка специальных символов
FT_SetChars(ftHandle:Dword;EventChar,EventCharEnabled,ErrorChar,ErrorChar
Enabled : Byte) : FT_Result ; stdcall ; External FT_DLL_Name name
'FT_SetChars';
// Очистка буферов
function FT_Purge(ftHandle:Dword;Mask:Dword) : FT_Result ; stdcall ;
External FT_DLL_Name name 'FT_Purge';
// Установка тайм-аутов
function FT_SetTimeouts(ftHandle:Dword;ReadTimeout,WriteTimeout:Dword) :
FT_Result ; stdcall ; External FT_DLL_Name name 'FT_SetTimeouts';
// Получение размера очереди
function FT_GetQueueStatus(ftHandle:Dword;RxBytes:Pointer) : FT_Result ;
stdcall ; External FT_DLL_Name name 'FT_GetQueueStatus';
// Получение числа устройств на линии
function FT_GetNumDevices(pvArg1:Pointer;pvArg2:Pointer;dwFlags:Dword) :
FT_Result ; stdcall ; External FT_DLL_Name name 'FT_ListDevices';
// Получение списка устройств
function FT_ListDevices(pvArg1:Dword;pvArg2:Pointer;dwFlags:Dword) :
FT_Result ; stdcall ; External FT_DLL_Name name 'FT_ListDevices';
// Расширенная функция открытия устройства
function FT_OpenEx(pvArg1:Pointer;dwFlags:Dword;ftHandle:Pointer) :
FT_Result ; stdcall ; External FT_DLL_Name name 'FT_OpenEx';
```

Все функции возвращают одну из констант (тип `FT_Result` описан просто как `Integer`):

<code>FT_OK</code>	<code>= 0</code>	// успешное выполнение функции
<code>FT_INVALID_HANDLE</code>	<code>= 1</code>	// ошибочный дескриптор устройства
<code>FT_DEVICE_NOT_FOUND</code>	<code>= 2</code>	// устройство не найдено
<code>FT_DEVICE_NOT_OPENED</code>	<code>= 3</code>	// устройство не открыто
<code>FT_IO_ERROR</code>	<code>= 4</code>	// ошибка в/в
<code>FT_INSUFFICIENT_RESOURCES</code>	<code>= 5</code>	// недостаточно ресурсов
<code>FT_INVALID_PARAMETER</code>	<code>= 6</code>	// ошибочные параметры
<code>FT_SUCCESS</code>	<code>= FT_OK</code>	// успешное выполнение функции

Набор функций практически ничем не отличается от функций последовательного порта. Исключение составляют функции, позволяющие работать со списком устройств (`FT_GetNumDevices`, `FT_ListDevices`, `FT_OpenEx`).

Эти функции используют флаги, задающие тип поиска:

FT_LIST_NUMBER_ONLY	= \$80000000	// только по номерам
FT_LIST_BY_INDEX	= \$40000000	// по индексу
FT_LIST_ALL	= \$20000000	// по любому условию
FT_OPEN_BY_SERIAL_NUMBER	= 1	// по серийному номеру
FT_OPEN_BY_DESCRIPTION	= 2	// по описанию

15.4. Переход от COM к USB

Схема преобразователя COM в USB, предоставленная институтом радиотехники (www.institute-rt.ru), показана на рис. 15.5.

С его помощью многие устройства, снабженные интерфейсом RS-232, можно соединить с компьютером по USB. Преобразователь подключают к компьютеру (или хабу) с помощью USB-вилки типа A (CN1), снабженной соединительным кабелем длиной 1,5 м. Увеличивать длину сверх названной не следует, это приведет к сбоям в работе USB.

15.4.1. Описание схемы преобразователя

Микросхема U3 FT8U232AM включена по стандартной схеме, рекомендованной изготовителем. Узел на транзисторе Q1 в момент подачи питания (подключения преобразователя к сети USB) формирует импульс, приводящий микросхему U3 в исходное состояние. Напряжение питания поступает на узлы преобразователя через фильтры FB1 и FB2 — обычные провода с надетыми на них ферритовыми шайбами.

Цепь R5C10 создает задержку на время запуска генератора на резонаторе Y1, в качестве которого можно применять импортный HC49U, отечественный РК415 и др. Если резонатор двухвыводной и не содержит встроенных конденсаторов, для надежного запуска генератора возможно придется установить внешние конденсаторы емкостью 10...20 пФ.

Микросхема U1 содержит приемники и передатчики интерфейсных сигналов, отвечающие стандарту RS-232, а также преобразователи напряжения 5 ± 10 В, необходимые для их работы. Указанная на схеме микросхема SP213ЕНСА (Sipex) обеспечивает скорость обмена данными до 460 Кбод. Если достаточно скорости 115 Кбод, указанную микросхему можно заменить на SP213ECA той же фирмы, MAX213CAI (Maxim) или ADM213EARS (Analog Devices).

Микросхема U1 93C46, как уже было сказано, не обязательна. Если ее решено установить, то необходимо предварительно запрограммировать, воспользовавшись рекомендациями, имеющимися в приложении к описанию микросхемы FT8U245.

Этот документ и много другой полезной технической и справочной информации можно найти на сайте компании FTDI (www.ftdichip.com). Вся документация, необходимая для изготовления платы преобразователя в заводских условиях, размещена на FTP-сервере <ftp://ftp.paguo.ru/pub/2002/07/usb-rs232>.

15.4.2. Установка скорости обмена

Информация о значениях коэффициента деления тактовой частоты программируемым делителем микросхемы FT8U245AM, необходимых для получения той или иной скорости обмена данными, содержится в файле `ftdiport.inf`, сопровождающем драйвер. Изменяя эти значения, можно достичь нестандартных значений скорости работы UART. Однако чаще их приходится изменять, чтобы учесть, например, отклонение частоты кварцевого резонатора от номинальных 6 МГц.

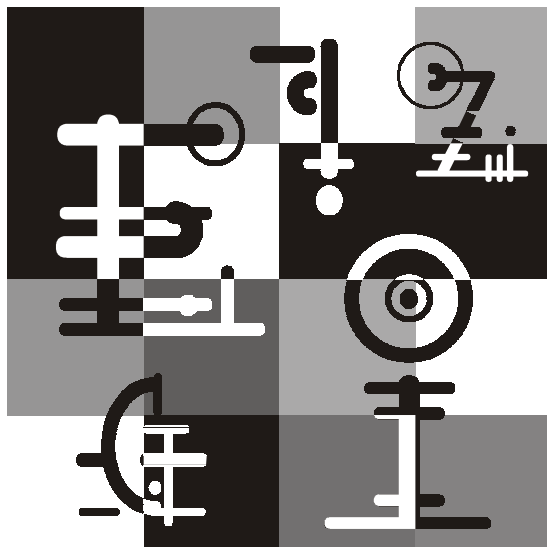
Чтобы рассчитать нужное значение коэффициента деления, число, вдвое меньшее частоты кварцевого резонатора (Гц), делят на требуемую скорость передачи (Бод). Частное округляют до ближайшего числа с дробной частью 0,125, 0,25, 0,5 или до целого числа. Полученное значение необходимо преобразовать в 16-разрядный двоичный код. В 14 младших разрядов кода (D0—D13) заносят целую часть коэффициента, а в старшие (D14, D15) — дробную в соответствии с таблицей. Этот код затем преобразуют в двухбайтное шестнадцатеричное число.

Работая в системе Windows 98, в разделе [FtdiPort232.HW.AddReg] упомянутого выше файла `ftdiport.inf` найдите строку

```
HKR, , ConfigData, 1, 01, 00, 3F, 3F, 10, 27, 88, 13, C4, 09, E2, 04, 71, 02, 38, 41, 9C, 80, 4E, C0, 34, 00, 1A, 00, 0D, 00, 06, 40, 03, 80, 00, 00, 00, 00
```

Младший байт каждого коэффициента записан первым, за ним следует старший. Например, последовательность E2,04 соответствует числу 4E2H. Внося нужные изменения, отредактированным файлом заменяют исходный.

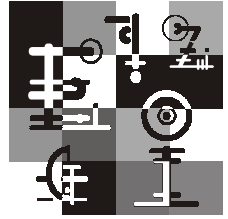
Работая в системе Windows 2000, аналогичным образом редактируют такую же строку в разделе [FtdiPort232.NT.HW.AddReg] того же файла.



ЧАСТЬ V

СПРАВОЧНИК

Глава 16



Базовые функции Windows

Рассмотрим базовые функции Windows, такие как открытие и закрытие объекта, чтение и передача данных, ожидание сигнального состояния объектов и др.

16.1. Функции *CreateFile* и *CloseHandle*: открытие и закрытие объекта

Функция *CreateFile* открывает объект, а функция *CloseHandle* — закрывает. Объектом может являться файл, драйвер, порт, устройство и т. д. Объект может быть открыт в режиме разделения или эксклюзивно. В этом случае попытка открыть ресурс еще раз, завершится с ошибкой. Дескриптор, полученный после вызова *CreateFile*, должен быть закрыт после использования вызовом *CloseHandle*.

Формат заголовков *CreateFile* и *CloseHandle* на языке C имеет следующий вид:

```
HANDLE CreateFile(  
    LPCTSTR lpFileName,           // имя объекта  
    DWORD   dwDesiredAccess,      // способ доступа  
    DWORD   dwShareMode,          // тип совместного доступа  
    LP_SECURITY_ATTRIBUTES lpSA,  // атрибуты защиты  
    DWORD   dwCreationDisposition, // параметры создания  
    DWORD   dwFlagsAndAttributes, // атрибуты  
    HANDLE  hTemplateFile         // дескриптор template-файла  
);  
  
BOOL CloseHandle(  
    HANDLE hObject                // дескриптор объекта  
);
```


Формат заголовков `CreateFile` и `CloseHandle` на языке Delphi имеет следующий вид:

```
function CreateFile(  
    lpFileName: PChar;           // имя объекта  
    dwDesiredAccess,             // способ доступа  
    dwShareMode: DWORD;         // тип совместного доступа  
    lpSA : PSecurityAttributes; // атрибуты защиты  
    dwCreationDisposition,       // параметры создания  
    dwFlagsAndAttributes: DWORD; // атрибуты  
    hTemplateFile: THandle       // дескриптор template-файла  
): THandle;  
function CloseHandle(  
    hObject: THandle             // дескриптор объекта  
): BOOL;
```

16.1.1. Дополнительные сведения

Для нотации языка C строка, передаваемая в функцию `CreateFile`, должна иметь дублированные знаки "\", например, `\\.\COM1` выглядеть как `\\\\.\COM1`.

16.1.2. Возвращаемое значение

Если функция `CreateFile` выполнена успешно, возвращается дескриптор открытого ресурса. Этот дескриптор используется для доступа к открытому ресурсу. Если при открытии ресурса произошла ошибка, функция возвращает значение `INVALID_HANDLE_VALUE`, а подробности можно узнать, вызвав функцию `GetLastError`.

После использования дескриптор должен быть освобожден вызовом функции `CloseHandle`.

Функция `CloseHandle` возвращает ненулевое значение, если закрытие дескриптора выполнено успешно, и возвращает 0, если произошла ошибка. Расширенную информацию об ошибке можно получить с помощью вызова `GetLastError`.

16.1.3. Пример вызова

Листинг 16.1 показывает структуру программы, использующей функции `CreateFile/CloseHandle` для доступа к коммуникационному порту COM1.

Листинг 16.1. Использование функций CreateFile и CloseHandle

```

{Переменная для хранения дескриптора порта}
var
  ComHandle : THandle;

{Открыть порт}
ComHandle:= CreateFile('\\\\.\\COM1',
                      GENERIC_READ or GENERIC_WRITE,
                      0,
                      nil,
                      OPEN_EXISTING,
                      FILE_ATTRIBUTE_NORMAL or FILE_FLAG_OVERLAPPED,
                      0

                      );

{Проверить результат}
if ComHandle = INVALID_HANDLE_VALUE then begin
  {Ошибка открытия порта, функция GetLastError вернет код ошибки}
  Exit;
end;
{ ... порт открыт успешно ...}
{ ... использование порта через дескриптор ComHandle ...}
{Закрытие порта}
CloseHandle(ComHandle);

```

16.2. Функция *ReadFile*: чтение данных

Функция *ReadFile* производит синхронное или асинхронное чтение данных.

Формат заголовка *ReadFile* на языке C имеет следующий вид:

```

BOOL ReadFile(
    HANDLE          hHandle,      // дескриптор, полученный от CreateFile
    LPVOID          lpBuffer,     // буфер для чтения
    DWORD           nNBTR,        // число байт для чтения
    LPDWORD         nNBR,         // реально прочитанное число байт
    LPOVERLAPPED    lpOverlapped // параметры асинхронного чтения
);

```

Формат заголовка `ReadFile` на языке Delphi имеет следующий вид:

```
function ReadFile(  
    hFile: THandle;           // дескриптор, полученный от CreateFile  
    var Buffer;               // буфер для чтения  
    nNumberOfBytesToRead: DWORD; // число байт для чтения  
    var lpNumberOfBytesRead: DWORD; // реально прочитанное число байт  
    lpOverlapped: Poverlapped // параметры асинхронного чтения  
): BOOL;
```

Первый параметр передает дескриптор объекта, полученный с помощью функции `CreateFile`. Указатель на буфер для чтения данных задается с помощью второго параметра, а размер этого буфера в байтах — с помощью третьего.

Параметр `nNBR` передает указатель на переменную типа `DWORD`, в которую возвращено реально прочитанное число байт. В Windows NT/2000/XP этот параметр не может быть `NULL`, если `lpOverlapped` равно `NULL`, и может быть `NULL`, если параметр `lpOverlapped` ненулевой. В Windows 95/98/ME этот параметр не может быть нулевым. Для получения количества байт, прочитанных в асинхронном режиме, может использоваться функция `GetOverlappedResult`.

Последний параметр передает настройки для асинхронного чтения данных. Если объект был открыт с параметром `FILE_FLAG_OVERLAPPED`, этот параметр обязательно должен указывать на правильную структуру типа `OVERLAPPED`, если же объект был открыт без использования `FILE_FLAG_OVERLAPPED`, то этот указатель обязательно должен быть `NULL`.

16.2.1. Дополнительные сведения

В Windows NT/2000/XP можно использовать функцию `ReadFileEx` для асинхронного чтения данных (см. *разд. 16.4*).

16.2.2. Возвращаемое значение

Функция завершается, если прочитано необходимое количество байтов или произошла ошибка. Если чтение прошло успешно, возвращается ненулевое значение.

В случае ошибки возвращается 0, а код ошибки можно получить с помощью вызова `GetLastError`.

16.2.3. Пример вызова

Листинг 16.2 показывает пример использования функции `ReadFile` для синхронного чтения данных.

Листинг 16.2. Пример использования функции ReadFile

```

var
  ComHandle : THandle;
  CurrentState : TComStat;
  CodeError : Cardinal;
  PData : Pointer;
  AvaibleBytes, RealRead : Cardinal;
begin
  ComHandle:= CreateFile(...);
  ... ..
  {Возвращает структуру состояния порта и код ошибок}
  ClearCommError(ComHandle, CodeError, @CurrentState);
  { Число полученных, но еще не прочитанных байт}
  AvaibleBytes:= CurrentState.cbInQue;
  { Проверка числа доступных байт}
  If AvaibleBytes > 0 then begin
    GetMem(PData, AvaibleBytes);
    If ReadFile(ComHandle, PData^, AvaibleBytes, RealRead, nil) then begin
      {Реально прочитано RealRead байт}
    End;
    FreeMem(PData);
  End;
  ... ..
  CloseHandle(ComHandle);
end;

```

16.3. Функция *WriteFile*: передача данных

Функция *WriteFile* производит синхронную или асинхронную запись данных в файл (порт, драйвер).

Формат заголовка *WriteFile* на языке C имеет следующий вид:

```

BOOL WriteFile(
  HANDLE          hHandle,      // дескриптор, полученный от CreateFile
  LPCVOID         lpBuffer,     // буфер данных
  DWORD           nNBW,         // длина буфера
  LPDWORD         pNBW,         // реально отправленное число байт
  LPOVERLAPPED    lpOverlapped // параметры асинхронной записи
);

```

Формат заголовка `WriteFile` на языке Delphi имеет следующий вид:

```
function WriteFile(  
    hFile      : THandle;    // дескриптор, полученный от CreateFile  
    const Buffer;             // буфер данных  
    nNBWTW     : DWORD;      // длина буфера  
    var lpNBW   : DWORD;      // реально отправленное число байт  
    lpOverlapped: POverlapped // параметры асинхронной записи  
): BOOL;
```

Первый параметр передает дескриптор объекта, полученный с помощью функции `CreateFile`. Указатель на буфер данных для записи задается с помощью второго параметра, а размер этого буфера в байтах — с помощью третьего.

Параметр `nNBW` задает указатель на переменную типа `DWORD`, в которую будет записано реально переданное число байтов. В Windows 95/98/ME этот параметр не может быть нулевым. В Windows NT/2000/XP этот параметр может быть нулевым, если задан указатель на параметры асинхронной записи `lpOverlapped`, и не может быть нулевым, если задается синхронная запись, т. е. указатель `lpOverlapped` нулевой.

16.3.1. Дополнительные сведения

В Windows NT/2000/XP можно использовать функцию `WriteFileEx` для асинхронной записи данных (см. разд. 16.5).

16.3.2. Возвращаемое значение

Если выполнение успешно, функция `WriteFile` возвращает ненулевое значение. Если функция завершилась с ошибкой, она возвращает нулевое значение, а код ошибки можно узнать с помощью вызова `GetLastError`.

16.3.3. Пример вызова

Листинг 16.3 показывает пример использования функции `WriteFile` для синхронной записи данных в коммуникационный порт, а листинг 16.4 — пример асинхронной записи.

Листинг 16.3. Пример использования функции `WriteFile` (синхронная запись)

```
var  
    FComPortHandle : THandle;  
    DataPtr : Pointer;  
    nToWrite, nWrite : Integer;
```

```

FComPortHandle:= CreateFile(...);
nToWrite:= количество передаваемых байт
GetMem(DataPtr, nToWrite);
... заполняем буфер данными ...
WriteFile(FComPortHandle, DataPtr^, nToWrite, nWrite, nil);
FreeMem(DataPtr);
CloseHandle(FComPortHandle);

```

Листинг 16.4. Пример использования функции WriteFile (асинхронная запись)

```

var
  FComHandle : THandle;
  AsyncPtr   : PAsync;
  BytesTrans : DWORD;

FComHandle:= CreateFile(...,
                        FILE_ATTRIBUTE_NORMAL or FILE_FLAG_OVERLAPPED,
                        0
                      );
{Создание асинхронных параметров}
New(AsyncPtr);
With AsyncPtr^ do begin
  Kind := 0; { 0 – write, 1 – read }
  GetMem(Data, Count);
  Move(Buffer, Data^, Count);
  Size := Count;
end;
{Передача данных}
WriteFile(FComHandle, Buffer, Count, BytesTrans, @AsyncPtr^.Overlapped);
{Освобождение памяти}
Dispose(AsyncPtr);
CloseHandle(FComHandle);

```

16.4. Функция *ReadFileEx*: APC-чтение данных

Функция `ReadFileEx` осуществляет асинхронное чтение данных. Работа этой функции похожа на вызов обычной функции `ReadFile` в режиме асинхронного чтения, но `ReadFileEx` позволяет программе выполнять другие дейст-

вия во время чтения данных. При завершении чтения будет вызвана специальная callback-процедура.

В Windows 95/98/ME эта функция не может быть использована для чтения данных из COM-порта.

Формат заголовка ReadFileEx на языке C имеет следующий вид:

```
BOOL ReadFileEx(
    HANDLE      hFile,           // дескриптор объекта
    LPVOID      lpBuffer,       // буфер для данных
    DWORD       dwBufLen,       // число байт для чтения
    LPOVERLAPPED lpOverlapped,  // параметры асинх. чтения
    // callback-процедура, выполняемая по завершении чтения
    LP_OVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
);
```

Формат заголовка ReadFileEx на языке Delphi имеет следующий вид:

```
function ReadFileEx(
    hFile          : THandle;      // дескриптор объекта
    lpBuffer       : Pointer;      // буфер для данных
    nNumberOfBytesToRead: DWORD;    // число байт для чтения
    lpOverlapped   : POverlapped;  // параметры асинх. чтения
    // callback-процедура, выполняемая по завершении чтения
    lpCompletionRoutine : TPROverlappedCompletionRoutine
): BOOL;
```

Все параметры кроме последнего совпадают с параметрами функции ReadFile. Последний параметр задает адрес процедуры, которая будет выполнена по завершении чтения данных. Эта процедура должна иметь тип TPROverlappedCompletionRoutine, описываемый следующим образом:

```
VOID CALLBACK FileIOCompletionRoutine(
    DWORD      dwErrorCode,           // код ошибки
    DWORD      dwNumberOfBytesTransferred, // число прочитанных байт
    LPOVERLAPPED lpOverlapped        // асинхронная структура
);
```

В Delphi тип этой процедуры описан как обычный указатель, без спецификации параметров:

```
type
    TPROverlappedCompletionRoutine = TFarProc;
```

На самом деле формат заголовка этой процедуры в Delphi должен быть такой, как показан в листинге 16.5.

Параметры callback-процедуры имеют следующий смысл:

- ❑ `dwErrorCode` принимает значение 0, если операция успешна;
- ❑ `dwNumberOfBytesTransferred` равен числу прочитанных байтов или 0, если функция завершена с ошибкой;
- ❑ `lpOverlapped` передает структуру асинхронного чтения.

16.4.1. Возвращаемое значение

При успешном завершении функция `ReadFileEx` возвращает ненулевое значение, а при ошибочном — ноль, при этом код ошибки можно получить с помощью вызова `GetLastError`.

16.4.2. Дополнительные сведения

Для завершения всех асинхронных операций может использоваться функция `CancelIo`.

Функция `ReadFileEx` игнорирует параметр `hEvent` в структуре `lpOverlapped` и он может использоваться программой.

16.4.3. Пример вызова

Листинг 16.5 показывает пример использования `ReadFileEx`. Обратите внимание, что процедура, вызываемая при завершении операции, должна иметь спецификатор `stdcall`.

Листинг 16.5. Пример использования функции `ReadFileEx`

```
Procedure TReadThread.Execute;  
Var ReadOL : TOverLapped; {структура для асинхронного чтения}  
  
  {Callback-процедура, вызываемая при получении байта}  
  Procedure OnCompletionRead(  
    dwErrorCode, dwNumberOfBytesTransferred : Cardinal;  
    var lpOverlapped : TOverlapped  
  ); stdcall;  
  begin  
    end;  
Begin  
  With FOwner do  
    While (not Terminated) and Connected do begin {пока порт открыт}
```



```

{Запуск операции асинхронного чтения}
ReadFileEx(FHandle, @FByte, 1, @ReadOL, @OnCompletionRead);
{Ожидание завершения операции}
SleepEx(INFINITE, True);
{Сюда мы попадем, только когда байт будет принят}
Synchronize(DoReadByte);
End;
End;

```

16.5. Функция *WriteFileEx*: APC-передача данных

Функция `WriteFileEx` производит асинхронную запись данных. Работа этой функции похожа на вызов функции `WriteFile` в режиме асинхронной записи, но `WriteFileEx` позволяет программе выполнять другие действия во время записи (передачи) данных. При завершении записи будет вызвана специальная callback-процедура.

В Windows 95/98/ME эта функция не может быть использована для чтения данных из COM-порта.

Формат заголовка `WriteFileEx` на языке C имеет следующий вид:

```

BOOL WriteFileEx(
    HANDLE      hFile,           // дескриптор объекта
    LPCVOID     lpBuffer,        // буфер для данных
    DWORD       dwBufLen,        // число байтов для записи
    LPOVERLAPPED lpOverlapped,   // параметры асинх. записи
    // callback-процедура, выполняемая по завершении записи
    LP_OVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
);

```

Формат заголовка `WriteFileEx` на языке Delphi имеет следующий вид:

```

function WriteFileEx(
    hFile      : THandle;           // дескриптор объекта
    lpBuffer   : Pointer;           // буфер для данных
    nNumberOfBytesToWrite: DWORD;    // число байт для записи
    const lpOverlapped : TOverlapped; // параметры асинх. записи
    lpCompletionRoutine: FARPROC
): BOOL;

```

Все параметры этой функции совпадают с параметрами функций `WriteFile` и `ReadFileEx`.

16.5.1. Возвращаемое значение

При успешном завершении функция `WriteFileEx` возвращает ненулевое значение, а при ошибочном — ноль. Код ошибки можно получить с помощью вызова `GetLastError`.

16.5.2. Пример вызова

Листинг 16.6 показывает пример использования `WriteFileEx`. Обратите внимание на спецификатор `stdcall` функции обратного вызова `OnCompletionWrite`.

Листинг 16.6. Пример использования функции `WriteFileEx`

```
Function TComPort.WriteByte(const B : Byte) : Boolean;
Var WriteOL : TOverlapped; {структура для асинхронной записи}
    {callback-процедура, вызываемая после завершения передачи}
    Procedure OnCompletionWrite(
        dwErrorCode, dwNumberOfBytesTransferred : Cardinal;
        var lpOverlapped : TOverlapped
    ); stdcall;
begin
    MessageBeep(0);
end;

Begin
    Result:= False;
    {создание события для асинхронной записи}
    FillChar(WriteOL, SizeOf(WriteOL), 0);
    WriteOL.hEvent:= CreateEvent(nil, True, True, nil);
    {асинхронная отправка байта}
    WriteFileEx(FHandle, @B, 1, WriteOL, @OnCompletionWrite);
    SleepEx(INFINITE, True);
    {освобождение дескриптора события}
    CloseHandle(WriteOL.hEvent);
End;
```

16.6. Функция *WaitForSingleObject*: ожидание сигнального состояния объекта

Функция `WaitForSingleObject` ожидает сигнального состояния объекта синхронизации. Функция завершается в двух случаях:

- ❑ объект перешел в сигнальное состояние;
- ❑ по завершении тайм-аута ожидания.

Формат заголовка на языке C имеет вид:

```
DWORD WaitForSingleObject(  
    HANDLE    hHandle,          // дескриптор объекта  
    DWORD     dwMilliseconds    // пауза ожидания сигнального состояния  
);
```

Формат заголовка на языке Delphi имеет вид:

```
function WaitForSingleObject(  
    hHandle      : THandle; // дескриптор объекта  
    dwMilliseconds: DWORD    // пауза ожидания сигнального состояния  
): DWORD;
```

Первый параметр передает дескриптор объекта. Пауза ожидания `dwMilliseconds` задается в миллисекундах. Специальная константа `INFINITE` задает неограниченное время ожидания.

При необходимости ожидания сигнального состояния одного из нескольких объектов или всех одновременно следует воспользоваться функцией `WaitForMultipleObjects`, а не объединять `WaitForSingleObject` с помощью `OR` или `AND`.

16.6.1. Возвращаемое значение

При успешном выполнении функция `WaitForSingleObject` возвращает значение, указывающее на состояние объекта. При ошибке возвращает `WAIT_FAILED`. В табл. 16.1 приводятся возможные результаты функции.

Таблица 16.1. Результаты функции *WaitForSingleObject*

Код	Описание
WAIT_ABANDONED	Используется для мьютексов. Возвращается в том случае, когда объект не был освобожден, хотя поток, его создавший, уже завершен. В нашей книге мы не используем мьютексы
WAIT_OBJECT_0	Объект перешел в сигнальное состояние

Таблица 16.1 (окончание)

Код	Описание
WAIT_TIMEOUT	Тайм-аут завершен, а сигнальное состояние не достигнуто. Естественно, в случае INFINITE такой результат невозможен
WAIT_FAILED	Ошибка вызова WaitForSingleObject

16.7. Функция *WaitForMultipleObjects*: ожидание сигнального состояния объектов

При необходимости ожидания сигнального состояния одного из нескольких объектов одновременно следует воспользоваться функцией `waitForMultipleObjects`, а не объединять `WaitForSingleObject` с помощью оператора `OR`. Эта функция ожидает сигнального состояния одного или всех вместе объектов синхронизации. Функция завершается в двух случаях:

- ❑ один или все объекты перешли в сигнальное состояние;
- ❑ по завершении тайм-аута ожидания.

Формат заголовка на языке C имеет вид:

```
DWORD WaitForMultipleObjects(  
    DWORD nCount,           // число ожидаемых объектов  
    const HANDLE * lpHandles, // массив дескрипторов объектов  
    BOOL bWaitAll           // флаг "ожидать все объекты"  
    DWORD dwMilliseconds    // пауза ожидания сигнального состояния  
);
```

Формат заголовка на языке Delphi имеет вид:

```
function WaitForMultipleObjects(  
    nCount      : DWORD;           // число ожидаемых объектов  
    lpHandles   : PWOHandleArray; // массив дескрипторов объектов  
    bwaitAll    : BOOL;           // флаг "ожидать все объекты"  
    dwMilliseconds: DWORD         // пауза ожидания сигнального состояния  
): DWORD;
```

Первый параметр передает число объектов синхронизации, дескрипторы которых передаются во втором параметре. Если третий параметр `TRUE`, функция будет ожидать сигнального состояния всех объектов, иначе — хотя бы одного из них. Четвертый параметр задает паузу ожидания так же, как в функции `WaitForSingleObject`. Пауза ожидания задается в миллисекундах. Специальная константа `INFINITE` задает неограниченное время ожидания.

16.7.1. Возвращаемое значение

При успешном выполнении функция `WaitForMultipleObjects` возвращает значение, указывающее на состояние объектов. При ошибке возвращает `WAIT_FAILED`. В табл. 16.2 приводятся возможные результаты функции.

Таблица 16.2. Результат функции `WaitForMultipleObjects`

Код	Описание
От <code>WAIT_ABANDONED_0</code> до <code>(WAIT_ABANDONED_0+nCount-1)</code>	Если <code>bwaitAll = TRUE</code> , возвращаемое значение показывает, что один из объектов — неосвобожденный мьютекс. Если <code>bwaitAll = FALSE</code> , то результат функции минус константа <code>WAIT_ABANDONED_0</code> будет равен индексу неосвобожденного мьютекса в массиве <code>lpHandles</code>
От <code>WAIT_OBJECT_0</code> до <code>(WAIT_OBJECT_0+nCount-1)</code>	Если <code>bwaitAll = TRUE</code> , то все объекты перешли в сигнальное состояние. Если <code>FALSE</code> , то результат функции минус константа <code>WAIT_OBJECT_0</code> будет равен индексу объекта в массиве <code>lpHandles</code> , первым перешедшего в сигнальное состояние
<code>WAIT_TIMEOUT</code>	Тайм-аут завершен, а сигнальное состояние ни одного объекта не достигнуто. Естественно, в случае <code>INFINITE</code> такой результат невозможен
<code>WAIT_FAILED</code>	Ошибка вызова <code>WaitForSingleObject</code>

16.8. Функция *GetOverlappedResult*: результат асинхронной операции

Функция `GetOverlappedResult` возвращает результат асинхронной операции с файлом или коммуникационным устройством.

Формат заголовка на языке C имеет вид:

```
BOOL GetOverlappedResult(  
    HANDLE          hHandle,          // дескрипторы объекта (файла или порта)  
    LPOVERLAPPED    lpOverlapped,    // структура асинхронного вызова  
    LPDWORD         lpNBT,           // число переданных или прочитанных байтов  
    BOOL            bwait             // флаг ожидания  
);
```

Формат заголовка на языке Delphi имеет вид:

```
function GetOverlappedResult(  
    hFile          : THandle; // дескрипторов объекта (файла или порта)  
    const lpOverlapped: TOverlapped; // структура асинхронного вызова  
    var   lpNBT      : DWORD; // число переданных или прочитанных байтов  
    bWait           : BOOL    // флаг ожидания  
): BOOL;
```

Первый параметр передает дескриптор файла или коммуникационного устройства. Перед вызовом этот дескриптор должен быть связан с асинхронной операцией, вызовом `ReadFile`, `WriteFile`, `DeviceIoControl` или `WaitCommEvent`.

Второй параметр передает структуру `OVERLAPPED`, которая использовалась при начале асинхронной операции.

Третий параметр возвращает число байтов, реально прочитанных или записанных в результате операции. Этот параметр не используется для коммуникационных портов.

Если параметр `bWait` равен `TRUE`, то функция не завершится, пока операция не будет завершена. Если `FALSE` и операция продолжается, то функция вернет `FALSE`, а вызов `GetLastError` вернет `ERROR_IO_INCOMPETE`.

В Windows 95/98/ME, если параметр `bWait` равен `TRUE`, то поле `hEvent` в структуре `OVERLAPPED` не может быть `NULL`.

16.8.1. Возвращаемое значение

При успешном выполнении функции возвращается ненулевое значение.

16.9. Функция *DeviceIoControl*: прямое управление драйвером

Функция `DeviceIoControl` посылает команду, задаваемую кодом `dwIoCode`, напрямую драйверу устройства с дескриптором `hDevice`, указывая ему выполнить определенные действия.

Для коммуникационных устройств дескриптор можно получить с помощью вызова `CreateFile` с обязательным использованием флага асинхронных операций `FILE_FLAG_OVERLAPPED`.

Формат заголовка на языке C имеет вид:

```
BOOL DeviceIoControl(  
    HANDLE    hDevice, // дескрипторов устройства
```

```

DWORD      dwIoCode,      // код выполняемой функции
LPVOID      lpInBuffer,    // указатель на входные данные
DWORD      dwInBufSize,    // размер входного буфера
LPVOID      lpOutBuffer,   // указатель на выходной буфер
DWORD      nOutBufSize,    // размер выходного буфера
LPDWORD     lpBytesRetn,    // требуемый размер выходного буфера
LPOVERLAPPED lpOverlapped // структура асинхронного вызова
);

```

Формат заголовка на языке Delphi имеет вид:

```

function DeviceIoControl(
    hDevice      : THandle; // дескрипторов устройства
    dwIoControlCode : DWORD; // код выполняемой функции
    lpInBuffer    : Pointer; // указатель на входные данные
    nInBufferSize : DWORD;   // размер входного буфера
    lpOutBuffer   : Pointer; // указатель на выходной буфер
    nOutBufferSize : DWORD;   // размер выходного буфера
    var lpBytesRetn : DWORD;   // требуемый размер выходного буфера
    lpOverlapped: Poverlapped // структура асинхронного вызова
): BOOL;

```

Параметр `lpInBuffer` передает входные данные, если они необходимы для выполнения операции. Может быть передано `NULL`, если для выполнения операции не требуется дополнительной информации. Размер передаваемых входных данных задается параметром `dwInBufSize`.

Если функция, задаваемая кодом `dwIoCode`, должна возвращать данные, то для этих данных передается указатель на буфер `lpOutBuffer` и его размер `nOutBufSize`. Если функция не возвращает данные, параметр `lpOutBuffer` должен быть установлен в `NULL`, а `nOutBufSize` в 0.

Последний параметр возвращает реальный размер данных, сохраненных в `lpOutBufSize`. Если буфер слишком маленький, функция завершится с ошибкой, а вызов `GetLastError` вернет `ERROR_INSUFFICIENT_BUFFER`. Значение `lpBytesRetn` в этом случае будет равно 0. Некоторые драйверы, если буфер мал для принятия полного пакета данных, возвращают только часть данных. В этом случае `GetLastError` возвращает значение `ERROR_MORE_DATA`, а `lpBytesRetn` равно числу полученных байтов. Приложение должно снова вызвать `DeviceIoControl` с теми же параметрами, указав новый стартовый адрес.

Если `lpOverlapped` равно `NULL`, то `lpByteRetn` не должно быть `NULL`. Даже если функция не возвращает данные и `lpOutBuffer` равно `NULL`, вызов

DeviceIoControl будет использовать lpBytesRetn. Однако после таких операций значение lpBytesRetn бессмысленно.

Если lpOverlapped не равно NULL, то lpByteRetn может быть NULL. Если этот параметр не NULL и операция возвращает данные, то lpBytesRetn не имеет смысла, пока вызванная асинхронная операция не завершится. Для получения результата и числа возвращенных байтов используется вызов GetOverlappedResult. Если hDevice связан с портом ввода/вывода, то получить число возвращенных байтов можно с помощью функции GetQueuedCompletionStatus.

Если поле lpOverlapped не NULL, то оно должно содержать указатель на структуру OVERLAPPED. Если hDevice был открыт без использования флага FILE_FLAG_OVERLAPPED, то этот параметр будет игнорироваться. Иначе операция будет расцениваться как асинхронная. Структура lpOverlapped в этом случае должна содержать правильный дескриптор объекта события, иначе функция завершится с непредсказуемой ошибкой.

Для асинхронных операций функция DeviceIoControl завершается сразу же, а сигнальный объект-событие будет сигнализировать о завершении операции. В синхронном режиме функция не завершится, пока не завершится операция, или завершится по ошибке.

Важно отметить, что в Windows 95/98/ME эту функцию можно применять только к дескрипторам виртуальных драйверов. Например, для открытия системного VxD-драйвера надо передавать в функцию CreateFile имя \\.\vwin32.

16.9.1. Возвращаемое значение

При успешном выполнении функция DeviceIoControl возвращает ненулевое значение. В случае ошибки вернет 0, а вызов GetLastError вернет код ошибки.

16.10. Функция *QueryDosDevice*: получение имени устройства по его DOS-имени

Функция QueryDosDevice возвращает информацию о DOS-имени устройства. Функция может получить текущее значение NT-имени для данного DOS-имени или вернуть список всех DOS-имен системы.

Формат заголовка на языке C имеет вид:

```
DWORD QueryDosDevice(  
    LPCTSTR pName,      // DOS-имя устройства  
    LPTSTR  pResult,     // буфер для результата
```



```
    DWORD    dwMax,      // размер буфера pResult
);
```

Формат заголовка на языке Delphi имеет вид:

```
function QueryDosDevice(
    lpDeviceName : PChar; // DOS-имя устройства
    lpTargetPath : PChar; // буфер для результата
    ucchMax      : DWORD  // размер буфера pResult
): DWORD;
```

Первый параметр передает DOS-имя устройства, например, COM1 или C:. В этом случае в буфер pResult будет записано внутреннее имя устройства. Если же pName нулевой, то в буфер будут записаны все DOS-имена, зарегистрированные в системе. Каждое имя — строка, завершающаяся нуль-символом. Признак завершения таблицы имен — двойной нуль-символ (т. е. пустая строка).

Второй параметр передает буфер для сохранения результата. Размер буфера передается в параметре dwMax. Если размер буфера мал, то результат выполнения зависит от версии операционной системы (*см. далее*).

16.10.1. Возвращаемое значение

При успешном выполнении функция QueryDosDevice возвращает число символов, сохраненных в буфере. В случае ошибки вернет 0, а вызов GetLastError вернет код ошибки.

Если буфер имеет недостаточный размер, функция вернет 0, а вызов GetLastError вернет код ERROR_INSUFFICIENT_BUFFER. В Windows NT/2000 если буфер мал, функция запишет в буфер столько данных, на сколько хватит его объема.

16.10.2. Пример вызова

Листинг 16.14 показывает получение списка всех имен в системе с помощью вызова QueryDosDevice.

Листинг 16.14. Пример вызова функции QueryDosDevice

```
{Получение всех имен устройства}
procedure TForm1.btnGetListClick(Sender: TObject);
var BufSize : Cardinal; P, PName : Pointer; SName : String;
begin
```

```

{Очищаем предыдущий список}
lbNameList.Items.Clear;

{Размер буфера}
BufSize:= 10240;
{Распределяем память для буфера}
GetMem(P, BufSize);
{Запрашиваем список имен}
If QueryDosDevice(nil, P, BufSize) <> 0 then begin
    {Цикл по всем именам...}
    PName:= P;
    While (True) do begin
        SName:= StrPas(PName);
        If SName = '' then Break;
        {Добавляем в список}
        lbNameList.Items.Add(SName);
        {Переход к следующему устройству}
        {Сдвигаем указатель на следующую строку}
        PName:= Pointer(LongInt(PName) + Length(SName)+1);
    End;
End;
{Освобождаем буфер}
FreeMem(P);
end;

```

16.11. Функция *DefineDosDevice*: операции с DOS-именем устройства

Функция `DefineDosDevice` определяет, переопределяет или удаляет DOS-имя из системы. Выполняемая операция зависит от параметра `dwFlags`. Его возможные значения приведены в табл. 11.3.

Формат заголовка на языке C имеет вид:

```

BOOL DefineDosDevice(
    DWORD   dwFlags,    // код операции
    LPCTSTR pName,      // DOS-имя устройства или его префикс
    LPCTSTR pDevice,    // NT-имя устройства
);

```

Формат заголовка на языке Delphi имеет вид:

```
function DefineDosDevice(  
    dwFlags      : DWORD; // код операции  
    lpDeviceName,      // DOS-имя устройства или его префикс  
    lpTargetPath : PChar  // NT-имя устройства  
): BOOL;
```

Первый параметр задает код операции (табл. 16.3). Второй параметр передает DOS-имя устройства, а третий — внутреннее имя устройства. Для выполнения этой функции требуются права администратора системы.

16.11.1. Возвращаемое значение

При успешном выполнении функция DefineDosDevice возвращает ненулевое значение. В случае ошибки функция вернет 0, а вызов GetLastError вернет код ошибки.

16.11.2. Пример вызова

Листинг 16.15 показывает добавление имени в систему с помощью использования функции DefineDosDevice.

Листинг 16.15. Пример вызова DefineDosDevice

```
{Добавить DOS-имя для NT-имени}  
procedure TForm1.btnAddNameClick(Sender: TObject);  
begin  
    If not DefineDosDevice(  
        DDD_RAW_TARGET_PATH,  
        PChar(DosDeviceName.Text),  
        PChar(NtDeviceName.Text))  
    then  
        StatusBar.Panels[0].Text:= 'Ошибка добавления имени';  
end;
```

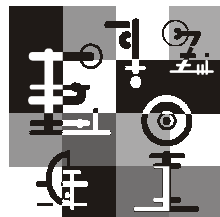
Таблица 16.3. Коды операций функции DefineDosDevice

Значение	Описание
DDD_RAW_TARGET_PATH	При указании этого параметра функция не конвертирует значение rDevice в имя устройства, а берет указанное значение имени "как есть"

Таблица 16.3 (окончание)

Значение	Описание
DDD_REMOVE_DEFINITION	Удаление указанного DOS-имени для выбранного устройства. Функция просматривает все имена, зарегистрированные для устройства, и удаляет те, префиксы которых совпадают с указанной в <code>pName</code> строкой. Если существует несколько таких имен для устройства, функция удалит только первое из них Если параметр <code>pDevice</code> нулевой или указывает на строку нулевой длины, то функция удалит первое подходящее имя из списка имен системы
DDD_EXACT_MATCH_ON_REMOVE	Указывается вместе с <code>DDD_REMOVE_DEFINITION</code> для гарантирования, что функция не удалит лишнего. Функция будет удалять имя только при полном совпадении имени из списка имен и указанного в <code>pName</code>

Глава 17



Функции *HID API*

Объем книги не позволяет привести описание всех функций HID. Полный список HID-функций можно найти в файле `Hid.pas` на компакт-диске.

Почти все HID-функции возвращают результат типа `LongBool`. При успешном выполнении функции возвращают `True`.

17.1. Функция *HidD_Hello*: проверка библиотеки

Функция `HidD_Hello` генерирует тестовую строку. Функция может использоваться для проверки наличия HID-библиотеки.

Формат заголовка на языке Delphi имеет вид:

```
function HidD_Hello(Buffer: PChar; BufferLength: ULONG): ULONG;
```

Первый параметр задает адрес буфера для строки, а второй — его размер.

17.2. Функция *HidD_GetHidGuid*: получение GUID

Функция `HidD_GetHidGuid` возвращает GUID для HID-класса. Это значение используется для поиска устройств этого класса.

Формат заголовка на языке Delphi имеет вид:

```
procedure HidD_GetHidGuid(var HidGuid: TGUID);
```

Единственный параметр передает переменную для результата.

Пример использования этой функции показан в листинге 17.1.

Листинг 17.1. Пример использования функции `HidD_GetHidGuid`

```
var  
    HidGuid : TGUID;
```

```
// Получить GUID для класса HID
HidD_GetHidGuid(HidGuid);
```

17.3. Функция *HidD_GetPreparedData*: создание описателя устройства

Функция `HidD_GetPreparedData` подготавливает буфер типа `THIDPPreparedData`, который используется для работы некоторых других HID-функций. После использования буфер должен быть освобожден с помощью вызова `HidD_FreePreparedData`.

Формат заголовка на языке Delphi имеет вид:

```
function HidD_GetPreparedData(HidDeviceObject: THandle;
    var PreparedData: PHIDPPreparedData): LongBool;
```

Первый параметр передает дескриптор устройства, полученный с помощью вызова `CreateFile`, а второй — указатель на буфер `THIDPPreparedData`.

Пример использования этой функции показан в листинге 17.2.

Листинг 17.2. Пример использования функции `HidD_GetPreparedData`

```
var
    PreparedData: PHIDPPreparedData;

If HidD_GetPreparedData(HidHandle, PreparedData) then begin
    // использование PreparedData
    ... ..
    // Освободить блок PreparedData
    HidD_FreePreparedData(PreparedData);
End;
```

17.4. Функция *HidD_FreePreparedData*: освобождение описателя устройства

Функция `HidD_FreePreparedData` освобождает буфер, созданный функцией `HidD_GetPreparedData`.

Формат заголовка на языке Delphi имеет вид:

```
function HidD_FreePreparedData(
    PreparedData: PHIDPPreparedData): LongBool;
```

Пример использования этой функции показан в листинге 17.2.

17.5. Функция *HidD_GetFeature*: получение FEATURE-репорта

Функция `HidD_GetFeature` позволяет получить FEATURE-репорт от HID-устройства.

Формат заголовка на языке Delphi имеет вид:

```
function HidD_GetFeature(HidDeviceObject: THandle;  
    var Report; Size: Integer): LongBool;
```

Первый параметр передает дескриптор устройства, полученный с помощью вызова `CreateFile`, второй — указатель на буфер для репорта, третий — размер буфера. Размер буфера должен быть на единицу больше, чем размер репорта, описанный в дескрипторе HID-устройства¹. Его можно получить с помощью функции `HidP_GetCaps`.

Если используется несколько репортов, то в первом байте буфера должен передаваться идентификатор репорта (Report ID).

Пример использования этой функции показан в листинге 17.3.

Листинг 17.3. Пример использования функции *HidD_GetFeature*

```
var  
    FeatureReport : Array [0..255] of Byte;  
  
If Capabilities.FeatureReportByteLength > 0 then begin  
    FillChar(Feature, SizeOf(Feature), #0);  
    // Feature [1]:= 1; идентификатор репорта  
    If HidD_GetFeature(HidHandle, Feature,  
        Capabilities.FeatureReportByteLength) then begin  
        // использование репорта  
    End;  
End;
```

17.6. Функция *HidD_SetFeature*: передача FEATURE-репорта

Функция `HidD_SetFeature` используется для передачи FEATURE-репорта. Размер передаваемого буфера должен быть на единицу больше, чем размер,

¹ Спецификация USB, также как и Windows, использует понятие "дескриптор". Следует различать дескриптор, возвращаемый функцией `CreateFile` (тип `THandle`) и дескриптор USB-устройства.

описанный в дескрипторе устройства. В первом байте буфера передается идентификатор репорта (Report ID), если он используется, и 0 — если нет.

Формат заголовка на языке Delphi имеет вид:

```
function HidD_SetFeature(HidDeviceObject: THandle;  
                        var Report; Size: Integer): LongBool;
```

Параметры этой функции совпадают с параметрами функции HidD_GetFeature.

17.7. Функция *HidD_GetNumInputBuffers*: получение числа буферов

Функция HidD_GetNumInputBuffers возвращает количество репортов, сохраняемое в кольцевом буфере драйвера.

Формат заголовка на языке Delphi имеет вид:

```
function HidD_GetNumInputBuffers(HidDeviceObject: THandle;  
                                var NumBufs: Integer): LongBool;
```

Первый параметр передает дескриптор устройства, полученный с помощью вызова CreateFile, второй — переменную для возвращения результата.

17.8. Функция *HidD_SetNumInputBuffers*: установка числа буферов

Функция HidD_SetNumInputBuffers устанавливает количество репортов, сохраняемое в кольцевом буфере драйвера.

Формат заголовка на языке Delphi имеет вид:

```
function HidD_SetNumInputBuffers(HidDeviceObject: THandle;  
                                NumBufs: Integer): LongBool;
```

Параметры этой функции совпадают с параметрами функции HidD_GetNumInputBuffers.

17.9. Функция *HidD_GetAttributes*: получение атрибутов устройства

Функция HidD_GetAttributes возвращает атрибуты устройства, описываемые структурой THIDDAttributes (листинг 17.4).

Листинг 17.4. Описание структуры THIDDAtributes

```
PHIDDAtributes = ^THIDDAtributes;
THIDDAtributes = record
    Size:          ULONG; // размер структуры
    VendorID:      Word;   // идентификатор производителя
    ProductID:     Word;   // идентификатор продукта
    VersionNumber: Word;   // версия
    // Могут присутствовать дополнительные поля
end;
```

Формат заголовка на языке Delphi имеет вид:

```
function HidD_GetAttributes(HidDeviceObject: THandle;
                           var HidAttrs: THIDDAtributes): LongBool;
```

Первый параметр передает дескриптор устройства, полученный с помощью вызова CreateFile, второй — буфер для возвращения результата. Поле Size структуры THIDDAtributes должно быть заполнено перед вызовом функции.

Пример использования этой функции показан в листинге 17.5.

Листинг 17.5. Пример использования функции HidD_GetAttributes

```
Var
    Attributes : THIDDAtributes;

Attributes.Size := SizeOf(THIDDAtributes);
If HidD_GetAttributes(HidHandle, Attributes) then begin
    // Функция выполнена успешно
End;
```

17.10. Функция *HidD_GetManufacturerString*: получение строки производителя

Функция HidD_GetManufacturerString возвращает строку производителя, индекс которой описан в дескрипторе конфигурации устройства.

Формат заголовка на языке Delphi имеет вид:

```
function HidD_GetManufacturerString(HidDeviceObject: THandle;
                                    Buffer: PWideChar; BufferLength: Integer): LongBool;
```

Первый параметр передает дескриптор устройства, полученный с помощью вызова `CreateFile`, второй — буфер для строки, третий — размер буфера.

Пример использования этой функции показан в листинге 17.6.

Листинг 17.6. Пример использования функции `HidD_GetManufacturerString`

```
Var
  Buffer : array [0..253] of WideChar;
  S : String;

FillChar(Buffer, SizeOf(Buffer), #0);
If HidD_GetManufacturerString(HidHandle, Buffer, SizeOf(Buffer)) then
begin
  S:= Format('    Производитель=%s', [Buffer]);
End;
```

17.11. Функция *HidD_GetProductString*: получение строки продукта

Функция `HidD_GetProductString` возвращает строку продукта, индекс которой описан в дескрипторе конфигурации устройства.

Формат заголовка на языке Delphi имеет вид:

```
function HidD_GetProductString(HidDeviceObject: THandle;
  Buffer: PWideChar; BufferLength: Integer): LongBool;
```

Первый параметр передает дескриптор устройства, полученный с помощью вызова `CreateFile`, второй — буфер для строки, третий — размер буфера.

17.12. Функция *HidD_GetSerialNumberString*: получение строки серийного номера

Функция `HidD_GetSerialNumberString` возвращает строку серийного номера, индекс которой описан в дескрипторе конфигурации устройства.

Формат заголовка на языке Delphi имеет вид:

```
function HidD_GetSerialNumberString(HidDeviceObject: THandle;
  Buffer: PWideChar; BufferLength: Integer): LongBool;
```

Первый параметр передает дескриптор устройства, полученный с помощью вызова `CreateFile`, второй — буфер для строки, третий — размер буфера.

17.13. Функция *HidD_GetIndexedString*: получение строки по индексу

Функция `HidD_GetIndexedString` возвращает строку по индексу.

Формат заголовка на языке Delphi имеет вид:

```
function HidD_GetIndexedString(HidDeviceObject: THandle;  
    Index: Integer; Buffer: PWideChar;  
    BufferLength: Integer): LongBool;
```

Первый параметр передает дескриптор устройства, полученный с помощью вызова `CreateFile`, второй — индекс запрашиваемой строки, третий — буфер для строки, четвертый — размер буфера.

Пример использования этой функции показан в листинге 17.7.

Листинг 17.7. Пример использования функции *HidD_GetIndexedString*

```
Function GetString(StrDescriptor : Byte): WideString;  
var Buffer : array [0..253] of WideChar;  
begin  
    Result := 'Ошибка';  
    if StrDescriptor <> 0 then  
        if HidD_GetIndexedString(HidHandle, StrDescriptor,  
            Buffer, SizeOf(Buffer)) then  
            Result:= Buffer;  
end;
```

17.14. Функция *HidD_GetInputReport*: получение INPUT-репорта

Функция `HidD_GetInputReport` позволяет получить INPUT-репорт. Эта функция доступна только начиная с Windows XP.

Формат заголовка на языке Delphi имеет вид:

```
function HidD_GetInputReport(HidDeviceObject: THandle;  
    Buffer: Pointer; BufferLength: ULONG): LongBool;
```

Первый параметр передает дескриптор устройства, полученный с помощью вызова `CreateFile`, второй — буфер для репорта, третий — размер буфера. Размер буфера может быть получен с помощью вызова функции `HidP_GetCaps`.

Пример использования этой функции показан в листинге 17.8.

Листинг 17.8. Пример использования функции `HidD_GetInputReport`

```
var
    InputReport : Array [0..255] of Byte;

If HidD_GetInputReport(HidHandle, @InputReport,
    Capabilities.InputReportByteLength) then begin
    // Репорт получен успешно
End;
```

17.15. Функция *HidD_SetOutputReport*: передача OUTPUT-репорта

Функция `HidD_SetOutputReport` позволяет передать OUTPUT-репорт. Эта функция доступна только начиная с Windows XP.

Формат заголовка на языке Delphi имеет вид:

```
function HidD_SetOutputReport(HidDeviceObject: THandle;
    Buffer: Pointer; BufferLength: ULONG): LongBool;
```

Первый параметр передает дескриптор устройства, полученный с помощью вызова `CreateFile`, второй — буфер репорта, третий — размер буфера.

17.16. Функция *HidP_GetCaps*: получение свойств устройства

Функция `HidP_GetCaps` позволяет получить свойства устройства. Функция возвращает структуру `THIDPCaps` (листинг 17.9).

Формат заголовка на языке Delphi имеет вид:

```
function HidP_GetCaps(PreparsedData: PHIDPPreparsedData;
    var Capabilities: THIDPCaps): NTSTATUS;
```

Листинг 17.9. Структура свойств устройства

```
PTHIDPCaps = ^ THIDPCaps;
THIDPCaps = record
    Usage:                TUsage; // значение Usage
    UsagePage:            TUsage; // значение Usage Page
```

```

InputReportByteLength:    Word;    // размер INPUT-репорта
OutputReportByteLength:   Word;    // размер OUTPUT-репорта
FeatureReportByteLength:  Word;    // размер FEATURE-репорта
Reserved:                 array [0..16] of Word;
// Специфические HID-значения
NumberLinkCollectionNodes: Word;
NumberInputButtonCaps:    Word;
NumberInputValueCaps:     Word;
NumberInputDataIndices:   Word;
NumberOutputButtonCaps:   Word;
NumberOutputValueCaps:    Word;
NumberOutputDataIndices:  Word;
NumberFeatureButtonCaps:  Word;
NumberFeatureValueCaps:   Word;
NumberFeatureDataIndices: Word;
end;
```

Пример использования этой функции показан в листинге 17.10.

Листинг 17.10. Пример использования функции HidP_GetCaps

```

Var
  Capabilities : HIDP_CAPS;
  PreparedData: PHIDPPreparedData;

If HidD_GetPreparedData(HidHandle, PreparedData) then begin
  HidP_GetCaps(PreparedData, Capabilities);
  // Capabilities.UsagePage
  // Capabilities.InputReportByteLength
  // Capabilities.OutputReportByteLength
  // Capabilities.FeatureReportByteLength
End;
```

17.17. Функция *HidP_MaxDataListLength*: получение размеров репортов

Функция *HidP_MaxDataListLength* позволяет получить размер репорта, определенного параметром *ReportType*.

Формат заголовка на языке Delphi имеет вид:

```
function HidP_MaxDataListLength(ReportType: DWORD;  
                                PreparedData: PHIDPPreparedData): ULONG;
```

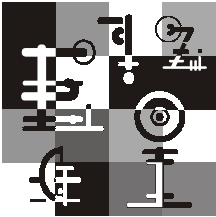
Первый параметр передает тип репорта:

```
HidP_Input    = 0;
```

```
HidP_Output   = 1;
```

```
HidP_Feature  = 2;
```

Второй параметр передает указатель на структуру THIDPPreparedData, получаемую вызовом HidP_GetCaps.



Хост-контроллер UCH

Интерфейс контроллера UHC описан в документе Universal Host Controller Interface (UHCI) Design Guide, версия 1.1 которого вышла в 1996 г. Это FS/LS хост-контроллер, который большую часть забот по планированию транзакций перекладывает на ПО — драйвер контроллера UHC (UHCD).

18.1. Регистры управления хост-контроллером

Управление хост-контроллером драйвер интерфейса USB производит через специальные регистры, которые принято разделять на две группы:

- ❑ группа конфигурационных регистров PCI (USB PCI Configuration Registers);
- ❑ группа регистров пространства ввода/вывода (USB Host Controller IO Space Registers).

Группа конфигурационных регистров показана в табл. 18.1. Более подробного описания мы давать не будем, т. к. эти регистры в пользовательских программах не используются.

Таблица 18.1. Конфигурационные регистры хост-контроллера USB

Смещение	Обозначение	Название	Доступ
00—01H	VID	Идентификатор производителя (Vendor Identification)	RO
02—03H	DID	Идентификатор устройства (Device Identification)	RO
04—05H	PCICMD	Регистр команд (PCI Command)	R/W
06—07H	PCISTS	Регистр статуса (PCI Device Status)	R/W
08H	RID	Ревизия (Revision Identification)	RO
09—0BH	CLASSC	Код класса (Class Code)	RO

Таблица 18.1 (окончание)

Смещение	Обозначение	Название	Доступ
0CH	—	Резерв	—
0DH	MLT	Задержка таймера (Latency Timer)	R/W
0EH	HEDT	Тип заголовка (Header Type)	RO
0F—1FH	—	Резерв	—
20—23H	USBBA	Базовый регистр в/в (USB IO Space Base Address)	R/W
24—3BH	—	Резерв	—
3CH	INTLN	Линия прерывания (Interrupt Line)	R/W
3DH	INTPN	Ножка прерывания (Interrupt Pin)	RO
3E—5FhH	—	Резерв	—
60H	SBRNUM	Версия последовательной шины (Serial Bus Release Number)	RO
61—BFH	—	Резерв	—
C0—C1H	LEGSUP	Поддержка совместимости (Legacy Support)	R/W
C2—FFH	—	Резерв	RO

Доступ к регистрам в/в хост-контроллера осуществляется через группу портов в/в, базовый адрес которой задан в конфигурационном регистре USBBA (табл. 18.2).

Таблица 18.2. Регистры в/в хост-контроллера

Смещение от базового адреса	Обозначение	Название	Доступ
00—01H	USBCMD	Регистр команд USB (USB Command)	R/W2*
02—03H	USBSTS	Регистр состояния USB (USB Status)	R/WC
04—05H	USBINTR	Регистр управления прерываниями (USB Interrupt Enable)	R/W
06—07H	FRNUM	Регистр номера кадра (Frame Number)	R/W2*
08—0BH	FLBASEADD	Регистр базового адреса кадра (Frame List Base Address)	R/W
0CH	SOFMOD	Регистр модификатора начала кадра (Start Of Frame Modify)	R/W

Таблица 18.2 (окончание)

Смещение от базового адреса	Обозначение	Название	Доступ
10—11H	PORTSC0	Регистр состояния и управления порта 0 (Port 0 Status and Control)	R/WC2*
12—13H	PORTSC1	Регистр состояния и управления порта 1 (Port 1 Status and Control)	R/WC2*

* Эти регистры имеют тип DWORD. Запись байта в эти регистры приведет к непредсказуемым последствиям.

18.1.1. Регистр команды USB (USBCMD)

Адрес: Base + (00H—01H).

Значение по умолчанию: 0000H.

Атрибуты: R/W.

Размер: 16 бит.

Регистр команды USB (USBCMD, USB Command Register) предназначен для передачи команд хост-контроллеру и доступен как для записи, так и для чтения. Контроллер начинает выполнение команды сразу же после того, как она записана в регистр.

Назначение битов этого регистра следующее:

- ☐ [15:8] зарезервированы;
- ☐ [7] **размер пакета** (MAXP, Max Packet) — задает максимальный размер пакета завершения кадра: 0—32 байта, 1—64 байта;
- ☐ [6] **флаг завершения конфигурации** (CF, Configure Flag) — флаг завершения конфигурирования контроллера. Данный разряд может быть установлен в единицу программным обеспечением после завершения процесса конфигурирования хост-контроллера. Этот флаг используется только программным обеспечением и на работу самого контроллера не влияет;
- ☐ [5] **включение режима отладки** (SWDBG, Software Debug) — запись единицы в этот разряд включает режим отладки (debug mode), запись нуля — выключает. Включение режима отладки доступно только при сброшенном бите RS, т. е. только в приостановленном режиме контроллера. В режиме отладки контроллер останавливается после выполнения каждой транзакции и сбрасывает бит RS. Возобновление работы контроллера происходит после установки RS в единицу (программным путем);

- [4] **общий выход из режима ожидания** (FGR, Force Global Resume) — запись единицы в данный разряд выводит хост-контроллер и подключенные к нему устройства из режима ожидания. Устанавливать данный разряд может не только программное обеспечение, но и сам хост-контроллер — при обнаружении подключения или отключения устройства во время пребывания устройства в режиме ожидания. Снять сигнал "пробуждения" можно по прошествии не менее 20 мс после его установки, записав в данный разряд ноль. Переключение в ноль этого разряда приводит к посылке сигнала EOP по шине;
- [3] **переключение в глобальный режим ожидания** (EGSM, Enter Global Suspend Mode) — запись единицы в данный разряд вызывает переключение хост-контроллера и всех подключенных к нему устройств в режим ожидания. В этом режиме не выполняются никакие транзакции, но контроллер может принимать сигналы удаленной побудки. Перед установкой в единицу этого бита необходимо остановить контроллер, сбросив бит RS. При выходе из режима ожидания данный бит сбрасывается в ноль программным обеспечением после сброса в ноль бита FGR;
- [2] **глобальный сброс** (GRESET, Global Reset) — запись единицы в данный разряд вызывает общий сброс хост-контроллера и всех подключенных к нему устройств. Снять сигнал сброса можно по прошествии не менее 10 мс после его установки, записав в данный разряд ноль;
- [1] **сброс хост-контроллера** (HRESET, Host Controller Reset) — запись единицы в данный разряд приводит к сбросу внутренних регистров хост-контроллера: обнуляется механизм обнаружения подключения и отключения устройств, блокируется работа портов контроллера. В результате происходит программное отключение подключенных к хосту устройств, биты 1 и 3 в регистрах состояния портов контроллера устанавливаются в единицу, а биты 0 и 8 сбрасываются. После завершения процесса сброса контроллер самостоятельно сбрасывает бит HRESET и разрешает обнаружение подсоединенных устройств, что приводит к соответствующему изменению битов 0 и 8 в регистрах состояния портов;
- [0] **запуск/останов** (RS, Run/Stop) — запись единицы в данный разряд активизирует работу контроллера (контроллер приступает к обработке и передаче данных), а запись нуля приводит к немедленной остановке контроллера и прекращению всех выполняемых операций. Контроллер сам может сбрасывать данный разряд в ноль в случае возникновения серьезных ошибок и сбоев. Этот бит используется для режима отладки (см. описание бита SWDBG в *разд. 18.1.1*).

18.1.2. Регистр состояния USB (USBSTS)

Адрес: Base + (02H—03H).

Значение по умолчанию: 0000H.

Атрибуты: R/WC.

Размер: 16 бит.

Регистр состояния USB (USBSTS, USB Status Register) отражает текущее состояние хост-контроллера. Регистр доступен для чтения и сброса.

Назначение битов этого регистра следующее:

- ☐ [15:6] зарезервированы;
- ☐ [5] **признак останова контроллера** (HC Halted) — устанавливается в единицу после либо сброса в ноль бита RS в регистре команды USB, либо программно или аппаратно (в режиме отладки или при обнаружении внутренней ошибки контроллера);
- ☐ [4] **признак внутренней ошибки контроллера** (Host Controller Process Error) — устанавливается контроллером при обнаружении ошибки функционирования. При обнаружении ошибки сбрасывается значение бита RS в регистре команды USB и вызывается прерывание;
- ☐ [3] **признак системной ошибки** (Host System Error) — устанавливается в единицу при возникновении сбоев в процессе передачи данных по шине PCI (PCI Parity Error, PCI Master Abort, PCI Target Abort). При обнаружении ошибки сбрасывается значение бита RS в регистре команды USB и вызывается прерывание;
- ☐ [2] **сигнал пробуждения** (Resume Detect) — признак поступления на шину сигнала удаленной побудки от устройства USB;
- ☐ [1] **признак прерывания по ошибке транзакции** (USB Error Interrupt) — устанавливается в единицу, если транзакция завершилась с ошибкой;
- ☐ [0] **признак USB-прерывания** (USBINT, USB Interrupt) — устанавливается в единицу контроллером при возникновении запроса прерывания по завершении транзакции (при установленном бите IOC в дескрипторе передачи) или при обнаружении короткого пакета (размер пакета меньше заданной в дескрипторе величины).

18.1.3. Регистр управления прерываниями (USBINTR)

Адрес: Base + (04H—05H).

Значение по умолчанию: 0000H.

Атрибуты: R/W.

Размер: 16 бит.

Регистр управления прерываниями (USBINTR, USB Interrupt Enable Register) позволяет разрешать или запрещать генерацию прерываний различных типов. Регистр USBINTR доступен и для чтения, и для записи.

Назначение битов этого регистра следующее:

- ❑ [15:4] зарезервированы;
- ❑ [3] **прерывание по короткому пакету** (Short Packet Interrupt Enable) — управление прерыванием по обнаружению короткого пакета: 0 — прерывание запрещено, 1 — разрешено;
- ❑ [2] **прерывание по завершении транзакции** (IOC Enable, Interrupt On Complete Enable) — управление прерыванием по завершении транзакции: 0 — прерывание запрещено, 1 — разрешено;
- ❑ [1] **прерывание по сигналу побудки** (Resume Interrupt Enable) — управление прерыванием по сигналу пробуждения: 0 — прерывание запрещено, 1 — разрешено;
- ❑ [0] **прерывание по тайм-ауту и ошибке CRC** (Timeout/CRC Interrupt Enable) — управление прерыванием по тайм-ауту и обнаружению ошибок CRC: 0 — прерывание запрещено, 1 — разрешено.

По умолчанию все прерывания запрещены (за исключением прерываний по сбоям самого контроллера, которые не маскируются).

18.1.4. Регистр номера кадра (*FRNUM*)

Адрес: Base + (06H—07H).

Значение по умолчанию: 0000H.

Атрибуты: R/W2.

Размер: 16 бит.

Регистр номера кадра (FRNUM, Frame Number Register) содержит текущий номер кадра USB. Регистр доступен для чтения в любой момент времени, а запись возможна только в том случае, если контроллер остановлен (бит RS в регистре команды USB сброшен в ноль).

Назначение битов этого регистра следующее:

- ❑ [15:11] зарезервированы;
- ❑ [10:0] **номер кадра/индекс кадра** (Frame Number/Frame List Current Index) — эти биты содержат текущий номер кадра, который передается в начале кадра в пакете SOF. Значение этих битов увеличивается на единицу после завершения каждого кадра, а после достижения значения 7FFH регистр обнуляется. Кроме того, разряды 0—9 используются при формировании индекса текущего элемента в списке кадров (соответствуют разрядам 2—11 индекса).

18.1.5. Регистр базового адреса кадра (**FLBASEADD**)

Адрес: Base + (08H—0BH).

Значение по умолчанию: не определено.

Атрибуты: R/W.

Размер: 32 бита.

Регистр базового адреса кадра (FLBASEADD, Frame List Base Address Register) содержит начальный (абсолютный) адрес списка кадров в оперативной памяти. Регистр доступен для записи и чтения. Базовый адрес должен быть выровнен на границу 4 Кбайт.

Назначение битов этого регистра следующее.

☐ [31:12] **базовый адрес** (Base Address);

☐ [11:0] зарезервированы, должны быть равны нулю.

Контроллер формирует указатель на текущий элемент списка кадров путем комбинирования сдвинутых влево на два разряда битов 0—9 из регистра номера кадра и битов 12—31 из регистра базового адреса. Разряды 0 и 1 указателя всегда равны нулю (указатель выравнивается на границу двойного слова). Количество указателей в списке кадров равно 1024, а размер списка составляет 4 Кбайт.

18.1.6. Регистр модификатора начала кадра (**SOFMOD**)

Адрес: Base + 0CH.

Значение по умолчанию: 40H (соответствует 1 кГц).

Атрибуты: R/W.

Размер: 8 бит.

Регистр модификатора начала кадра (SOFMOD, Start Of Frame Modify Register) служит для подстройки частоты кадров USB с целью обеспечения синхронизации всех устройств системы при работе в режиме реального времени.

Назначение битов этого регистра следующее:

☐ [7] зарезервирован;

☐ [6:0] **регистр SOFTV** (SOF Timing Value).

Значение SOFTV складывается с числом 11 936, в результате чего формируется делитель частоты кварцевого резонатора генератора тактовой частоты. Частота кварцевого резонатора составляет 12 МГц, поэтому значение по умолчанию (40H) соответствует 1 кГц. Изменяя значение модификатора от 0 до 127, можно осуществить подстройку частоты кадров USB в пределах $\pm 0,5\%$.

18.1.7. Регистр состояния и управления порта (PORTSC)

Адрес: $\text{Base} + (10\text{H} - 11\text{H}) - \text{PORTSC0}$.

$\text{Base} + (12\text{H} - 13\text{H}) - \text{PORTSC1}$.

Значение по умолчанию: 0080H.

Атрибуты: R/W2.

Размер: 16 бит.

Регистр состояния и управления порта (PORTSC, Port Status and Control Register) позволяет контролировать режим работы хост-контроллера. Регистры PORTSC0 и PORTSC1 доступны для записи и чтения. Запись данных может производиться только словом.

Назначение битов этого регистра следующее:

- ❑ [15:13] зарезервированы;
- ❑ [12] **признак режима ожидания** (Suspend) — устанавливается в единицу, когда порт находится в режиме ожидания. Бит доступен для чтения и записи и может использоваться для программного перевода порта в режим ожидания. Этот бит не может быть установлен в единицу, если установлен глобальный режим ожидания (бит 3 в регистре USBCMD). Этот бит вместе с битом 2 определяет состояние хаба:
 - [12,2] = x0 — выключен;
 - [12,2] = 01 — включен;
 - [12,2] = 11 — режим ожидания.

В режиме ожидания любые транзакции и передачи данных на нисходящий порт блокируются. Исключение составляют команды сброса (глобальный сброс или сброс порта). При установке режима ожидания этот бит устанавливается не сразу, а после завершения текущей активной транзакции;

- ❑ [11:10] зарезервированы;
- ❑ [9] **сброс порта** (Port Reset) — бит сброса порта устанавливается в единицу при подаче команды сброса и находится в этом состоянии до тех пор, пока процедура сброса не будет завершена;
- ❑ [8] **признак LS-устройства** (Low Speed Device Attached) — этот бит доступен только для чтения и устанавливается в единицу, если к порту подключено низкоскоростное устройство;
- ❑ [7] зарезервирован, всегда имеет значение 1;
- ❑ [6] **признак наличия сигнала пробуждения** (Resume Detect) — этот бит доступен для чтения и установки. Хост-контроллер устанавливает этот бит в единицу при обнаружении сигнала пробуждения; программное обеспече-

ние устанавливает этот бит для формирования сигнала пробуждения. Если этот бит имеет значение 1, запись нуля приводит к посылке низкоскоростного сигнала EOP, но бит остается в единице до окончания EOP;

- ❑ [5:4] **состояние линии** (Line Status) — бит 4 отражает текущее состояние линии D+, а бит 5 — состояние линии D-. Эти биты доступны только для чтения;
- ❑ [3] **признак изменения активности порта** (Port Enable/Disable Change) — значение этого бита равно нулю, если состояние порта не изменялось и равно единице, если произошло включение или отключение порта. Запись единицы в этот бит сбрасывает его в ноль;
- ❑ [2] **включение/выключение порта** (Port Enable/Disable) — значение 1 включает работу порта, а 0 — выключает. Данный разряд доступен как для чтения, так и для записи. Запись нуля (блокировка порта) может выполняться как программным обеспечением, так и хост-контроллером (при возникновении сбоя в работе порта), а запись единицы (включение порта) — только программным обеспечением. Состояние данного разряда не изменится, пока не изменится реальное состояние порта (возможна задержка срабатывания);
- ❑ [1] **признак изменения статуса подключения** (Connect Status Change) — устанавливается в единицу при любых изменениях текущего статуса подключения. Этот разряд доступен для чтения и сброса (запись единицы в данный разряд сбрасывает его в ноль);
- ❑ [0] **текущий статус подключения** (Current Connect Status) — данный разряд доступен только для чтения и служит для определения наличия подключения USB-устройств к данному порту (0 — к порту ничего не подключено, 1 — к порту подключено USB-устройство).

18.2. Структуры данных хост-контроллера UCH

18.2.1. Список кадров

Список кадров (Frame List) — это массив, содержащий 1024 указателя. Каждый элемент массива занимает 32 бита, т. о. весь массив занимает 4 Кбайт. Начальный адрес списка хранится в регистре базового адреса списка кадров FLBASEADD (он должен быть выровнен на границу 4 Кбайт).

Разряды элемента списка имеют следующее назначение:

- ❑ [31:4] биты 4–31 указателя кадров (FLP, Frame List Pointer);
- ❑ [3:2] зарезервированы и должны иметь значение 0;

- ❑ [1] тип структуры данных (QH/TD Select), адрес которой содержится в указателе кадра;
 - 0 — дескриптор передачи (TD, Transfer Descriptor);
 - 1 — заголовок очереди (QH, Queue Head);
- ❑ [0] признак завершающего кадра (Terminate). Значение 0 этого бита означает, что указатель кадра является правильным и содержит адрес заголовка очереди или дескриптора передачи. Значение 1 означает, что указатель не несет информации и обрабатываться не должен.

18.2.2. Дескриптор передачи

Дескриптор передачи (TD, Transfer Descriptor) описывает параметры транзакции, запрашиваемой клиентом. Дескриптор должен быть выровнен на границу 16 байт. Дескрипторы всех четырех типов передач имеют одинаковую структуру.

Каждый дескриптор имеет размер 32 байта и состоит из двух частей: младшие четыре двойных слова (DWORD) занимает область данных хост-контроллера, а старшие четыре двойных слова — область данных программного обеспечения, которая не обрабатывается и на функционирование хост-контроллера не влияет.

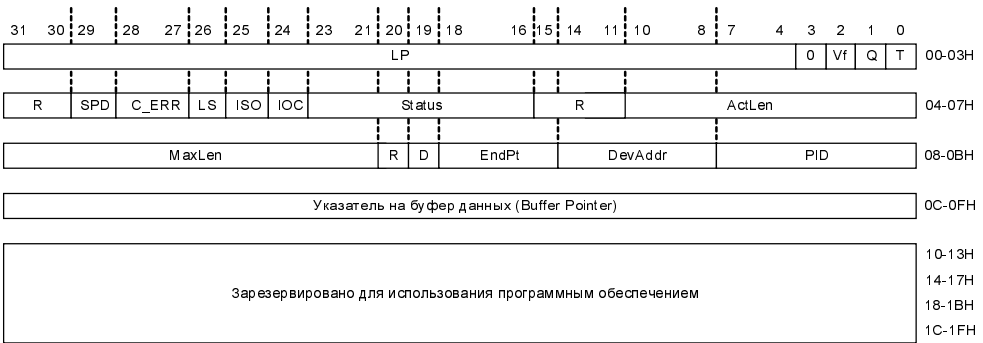


Рис. 18.1. Структура дескриптора передачи

На рис. 18.1 показана структура дескриптора передачи, состоящего из следующих полей:

- ❑ двойное слово 0 (00–03H) — указатель на следующий элемент списка дескрипторов (TD Link Pointer):
 - [31:4] **указатель на следующий элемент списка дескрипторов (LP, Link Pointer)**;

- [3] зарезервирован, при записи должен быть равен 0;
 - [2] **порядок обработки очередей** (Vf, Depth/Breadth Select) — значение 0 означает обработку "в ширину", а 1 — "в глубину"; при обработке "в ширину" при завершении обработки данного дескриптора контроллер переключится на обработку следующей очереди, а при обработке "в глубину" будет обрабатываться следующая транзакция в текущей очереди;
 - [1] **тип структуры данных** (Q, QH/TD Select) — значение 0 означает дескриптор передачи, а 1 — заголовок очереди;
 - [0] **признак последнего элемента списка** (T, Terminate) — если этот бит равен 0, то в поле указателя содержится адрес следующего элемента списка, а если 1, то данный элемент является последним в списке и поле указателя не должно обрабатываться;
- двойное слово 1 (04–07H) — слово управления и состояния (TD Control and Status):
- [31:30] зарезервированы и должны содержать нули;
 - [29] **разрешение приема укороченного пакета данных** (SPD, Short Packet Detect):
 - ◊ 0 — прием запрещен;
 - ◊ 1 — если длина принимаемого пакета меньше заданной, дескриптор передачи становится неактивным, заголовок очереди не изменяется и (по окончании кадра) устанавливается бит USBINT в регистре состояния и вырабатывается прерывание (если оно разрешено);
 - [28:27] **счетчик ошибок** (C_ERR, Number of Error) — работает на вычитание (его значение уменьшается на единицу после каждой неудачной попытки выполнения транзакции):
 - ◊ 00 — нет лимита ошибок;
 - ◊ 10 — допускается одна ошибка;
 - ◊ 01 — допускаются две ошибки;
 - ◊ 11 — допускаются три ошибки.
- После исчерпания лимита ошибок транзакция становится неактивной, и устанавливается признак сбоя (бит 22);
- [26] **тип устройства** (LS, Low Speed Device):
 - ◊ 0 — полноскоростное устройство;
 - ◊ 1 — низкоскоростное устройство;

- [25] **признак дескриптора изохронной передачи** (ISO, Isochronous Select):
 - ◇ 1 — дескриптор изохронной передачи (после выполнения изохронные дескрипторы отмечаются как неактивные независимо от результата выполнения транзакции);
 - ◇ 0 — дескриптор другого типа передачи;
- [24] **управление сигналом прерывания по завершении кадра** (IOC, Interrupt on Complete) — при значении этого бита 0 прерывание не вырабатывается, а при значении 1 по завершении кадра, в котором выполнялась обработка данного дескриптора, вызывается прерывание;
- [23:16] состояние процесса выполнения команды (Status):
 - ◇ [23] **признак активного дескриптора** (Active) — устанавливается в единицу программным обеспечением при включении дескриптора в очередь и сбрасывается хост-контроллером после завершения связанной с данным дескриптором транзакции или после обнаружения фатальной ошибки при ее выполнении;
 - ◇ [22] **признак сбоя при выполнении транзакции** (Stalled) — устанавливается в единицу при обнаружении серьезной ошибки в процессе выполнения транзакции; при установке этого бита контроллер одновременно сбрасывает бит 23;
 - ◇ [21] **признак ошибки в буфере данных** (Data Buffer Error) — устанавливается в 1 при переполнении буфера в процессе приема или опустошении буфера в процессе передачи данных;
 - ◇ [20] **признак обнаружения перекрестных помех** (Babble Detected) — устанавливается в единицу при обнаружении помех при выполнении транзакции;
 - ◇ [19] **признак отказа от транзакции** (NAK Received) — устанавливается в единицу, если контроллер получил сигнал NAK при выполнении транзакции;
 - ◇ [18] **признак обнаружения ошибки тайм-аута или ошибки CRC** (CRC/Time out Error) — устанавливается в единицу, если устройство не отвечает на запрос или при выполнении транзакции обнаружено несовпадение контрольной суммы;
 - ◇ [17] **признак ошибки NRZI** (Bitsuff Error) — устанавливается в единицу, если в принятой последовательности битов подряд следует более шести единиц;
 - ◇ [16] зарезервирован и должен иметь значение 0;
- [15:11] зарезервированы и должны содержать нули;

- [10:0] **объем данных, переданных в результате транзакции** (ActLen, Actual Length) — значение этого поля на единицу меньше количества переданных байтов;
- двойное слово 2 (04–07H) — маркер дескриптора передачи (TD Token):
 - [31:21] **объем передаваемых данных** в байтах минус единица (MaxLen, Maximum Length):
 - ◊ 000H — 1 байт;
 - ◊ 4FFH — 1280 байт;
 - ◊ 7FFH — пустой пакет;
 - ◊ 500H–7FЕH — недопустимые значения.
 - [20] зарезервирован и должен содержать ноль.
 - [19] **переключатель синхронизации данных** (D, Data Toggle):
 - ◊ 0 — DATA0;
 - ◊ 1 — DATA1;
 - [18:15] **номер конечной точки** (EndPt, Endpoint);
 - [14:8] **адрес устройства** (DevAddr, Device Address);
 - [7:0] **идентификатор пакета** (PID, Packet Identification):
 - ◊ 2DH — SETUP;
 - ◊ 69H — IN;
 - ◊ E1H — OUT.
- двойное слово 3 (0C–0FH) — указатель на буфер данных (TD Buffer Pointer);
- двойные слова 4–7 (10–1FH) область данных программного обеспечения.

18.2.3. Заголовок очереди

Заголовок очереди (QH, Queue Head) — это специальная структура данных, предназначенная для создания очередей, используемых при передачах типов Control, Bulk и Interrupt. Заголовок очереди должен быть выровнен на границу 16 байт.

На рис. 18.2 показана структура заголовка очереди, состоящего из 32-разрядных слов:

- двойное слово 0 (байты 00–03H) — указатель на следующий элемент "горизонтального списка":
 - [31:4] **биты 4–31 указателя на следующий элемент "горизонтального списка"** (QHLP, Queue Head Link Pointer); младшие четыре разряда указателя содержат нули;

- [3] зарезервирован и должен содержать ноль;
- [2] зарезервирован (R, Reserved), не влияет на работу;

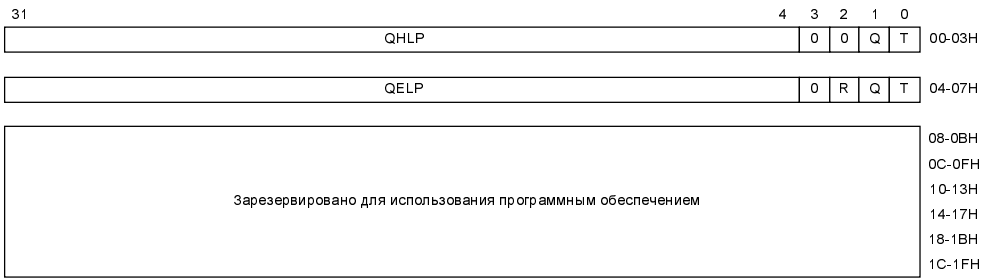


Рис. 18.2. Структура заголовка очереди

- [1] **тип структуры данных** (Q, QH/TD Select), адрес которой находится в указателе:
 - ◊ 0 — дескриптор передачи;
 - ◊ 1 — заголовок очереди;
 - [0] **признак последнего заголовка в списке** (T, Terminate):
 - ◊ 0 — указатель содержит адрес следующего заголовка;
 - ◊ 1 — данный элемент является последним в "горизонтальном списке" и поле указателя не должно обрабатываться контроллером;
- двойное слово 1 (байты 04—07H) — указатель на первый элемент очереди:
- [31:4] **биты 4—31 указателя на следующий элемент очереди** (QELP, Queue Element Link Pointer) — младшие четыре разряда указателя содержат нули;
 - [3:2] зарезервированы и должны содержать нули;
 - [1] **тип структуры данных** (Q, QH/TD Select), адрес которой находится в указателе:
 - ◊ 0 — дескриптор передачи;
 - ◊ 1 — заголовок очереди;
 - [0] **признак последнего элемента очереди** (T, Terminate):
 - ◊ 0 — указатель содержит адрес следующего элемента;
 - ◊ 1 — данный элемент является последним в очереди, и поле указателя не должно обрабатываться контроллером;
- двойные слова 2—7 зарезервированы для использования программным обеспечением.

18.3. Обработка списка дескрипторов UCH

Порядок выполнения запросов определяется *планом обработки дескрипторов* (Schedule Layout). В начале выполнения очередного кадра хост-контроллер получает из списка кадров указатель на список дескрипторов, который должен быть обработан в данном кадре.

Список дескрипторов формируется согласно определенным правилам (рис. 18.3). Каждый элемент списка может быть действительным или недействительным. Действительный элемент списка кадров должен содержать указатель на дескриптор передачи или очередь заголовка. В нормальном режиме работы контроллера все элементы списка должны быть действительными.

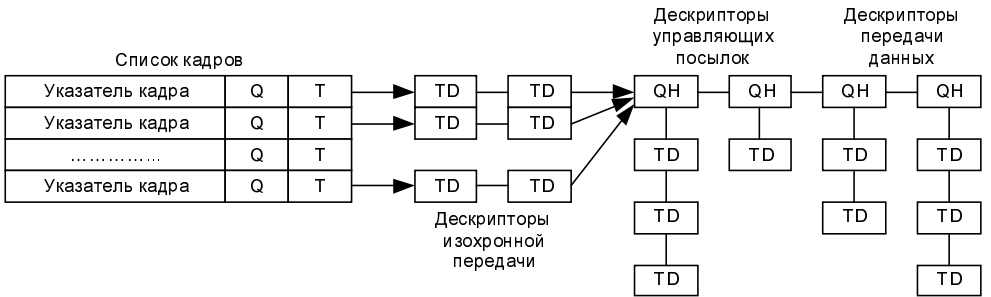


Рис. 18.3. Пример списка дескрипторов

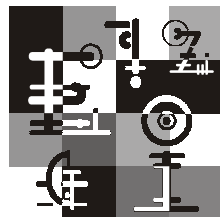
В первую очередь должны быть обработаны дескрипторы изохронной передачи, поэтому такие дескрипторы размещаются последовательно друг за другом в начале списка. Далее обрабатывается список заголовков, в начале которого размещены заголовки очередей дескрипторов для передачи по прерываниям, затем следуют заголовки очередей дескрипторов управляющих посылок. В конце списка находятся заголовки очередей дескрипторов передачи данных. Список дескрипторов изохронной передачи в каждом кадре свой, а список заголовков очередей общий для всех кадров.

Каждый заголовок очереди указывает на первый из находящихся в очереди дескрипторов передачи. Обычно список заголовков очередей обрабатывается по горизонтали (в ширину): контроллер извлекает из заголовка первый дескриптор в очереди, обрабатывает его и переходит к следующему заголовку очереди. При необходимости можно установить режим обработки очереди по вертикали: контроллер в этом случае вначале обработает всю очередь и только потом перейдет к заголовку следующей очереди.

Если выполнение операции, заданной дескриптором передачи, не завершено в текущем кадре (например, данные не готовы для передачи), в следующем кадре операция повторяется. Когда операция завершена, дескриптор передачи помечается как обслуженный и удаляется из очереди: контроллер извлекает из него указатель на следующий дескриптор и переписывает его в заголовок очереди. Область, зарезервированная в дескрипторе передачи для программного обеспечения, в первую очередь предназначена для "сборки мусора": из обслуженных и ненужных дескрипторов можно сформировать очередь с целью повторного использования занимаемых ими участков памяти.

Каждая очередь дескрипторов передачи обычно формируется прикладной программой для работы с определенной функцией или конечной точкой функции. Если программа работает с несколькими конечными точками или в системе параллельно выполняется несколько прикладных программ, в списке будут присутствовать несколько очередей дескрипторов.

Глава 19



Инструменты

В *разд. 9.5* мы уже рассматривали инструменты создания драйверов, а в этой главе мы опишем некоторые инструменты, облегчающие работу.

19.1. Средства Microsoft Visual Studio

Поставка Microsoft Visual Studio 6 включает несколько программ, которые могут оказаться полезными при разработке USB-устройств и драйверов.

19.1.1. Depends

Программа Depends (рис. 19.1) позволяет посмотреть внутреннее содержание DLL-модулей: импортируемые и экспортируемые функции, использование других модулей, версии модулей и т. д.

19.1.2. Error Lookup

Программа Error Lookup (рис. 19.2) отображает строковое значение ошибки по номеру ошибки.

19.1.3. GuidGen

Программа GuidGen (рис. 19.3) позволяет сгенерировать уникальный идентификатор GUID для ключа в реестре или для идентификатора драйвера (устройства).

Более простая консольная программа UUIDGEN создает один идентификатор при каждом запуске.

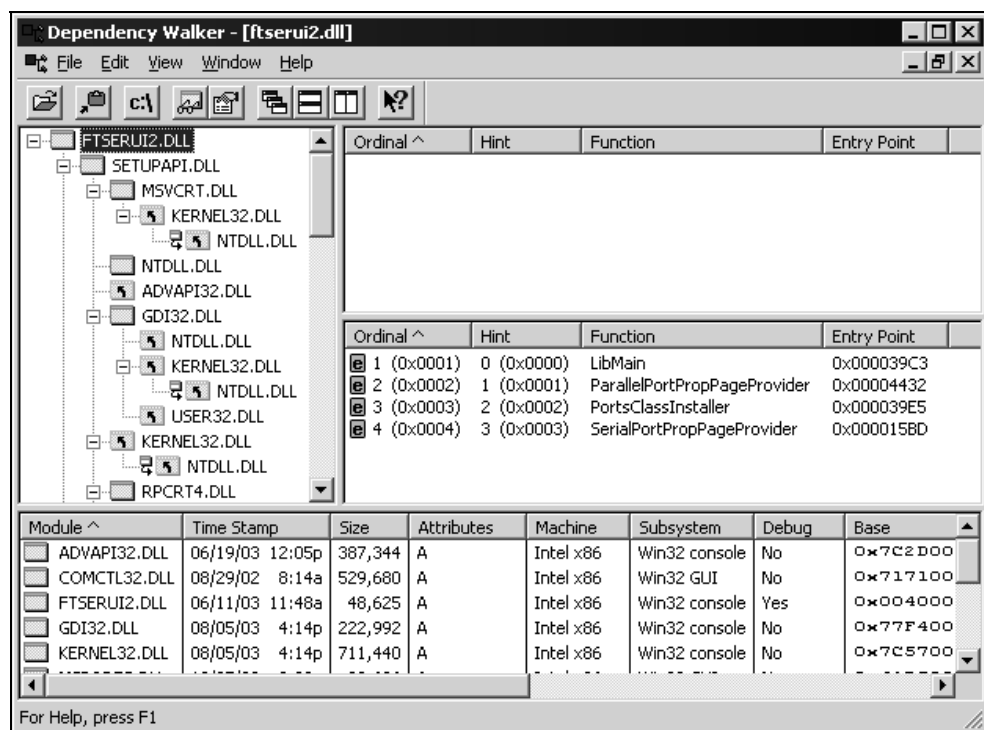


Рис. 19.1. Программа Depends

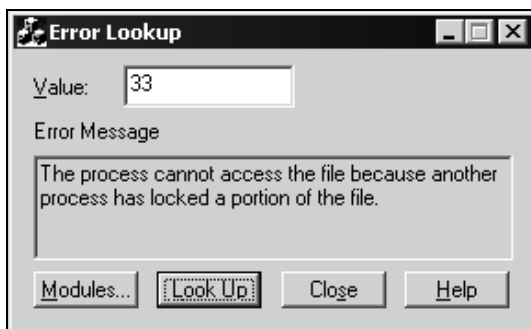


Рис. 19.2. Программа Error Lookup

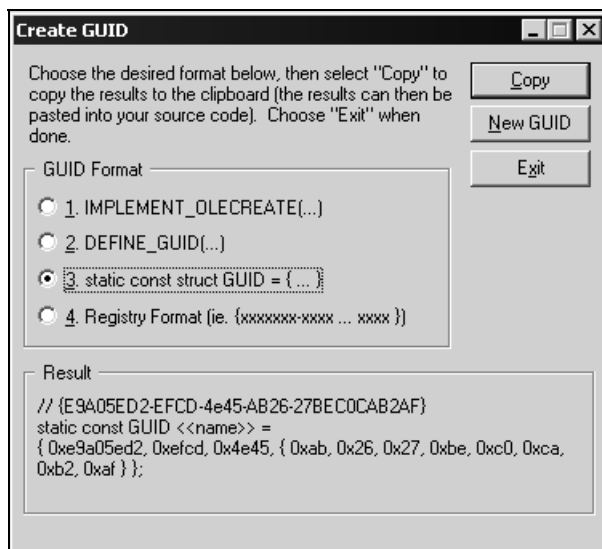


Рис. 19.3. Программа GuidGen

19.2. Средства Microsoft DDK

Среди утилит, входящих в состав Microsoft DDK (Driver Development Kit, комплект разработчика драйверов), есть утилиты, которые помогут в работе с USB-интерфейсом и созданием INF-файлов.

19.2.1. DeviceTree

Программа DeviceTree (рис. 19.4) позволяет отобразить дерево драйверов и соответствующих устройств. Отображение дерева устройств производится с двух точек зрения: с точки зрения принадлежности объектов устройств драйверам (режим D) и с точки зрения взаимной подчиненности объектов устройств при выполнении нумерации устройств.

Для каждого драйвера отображаются список обрабатываемых кодов (рабочих процедур), размер, атрибуты драйвера и множество других параметров. Однако полнота информации оборачивается другой стороной — при построении дерева устройств программа может привести к сбою или аварийной перезагрузке компьютера (о чем программа честно предупреждает при старте).

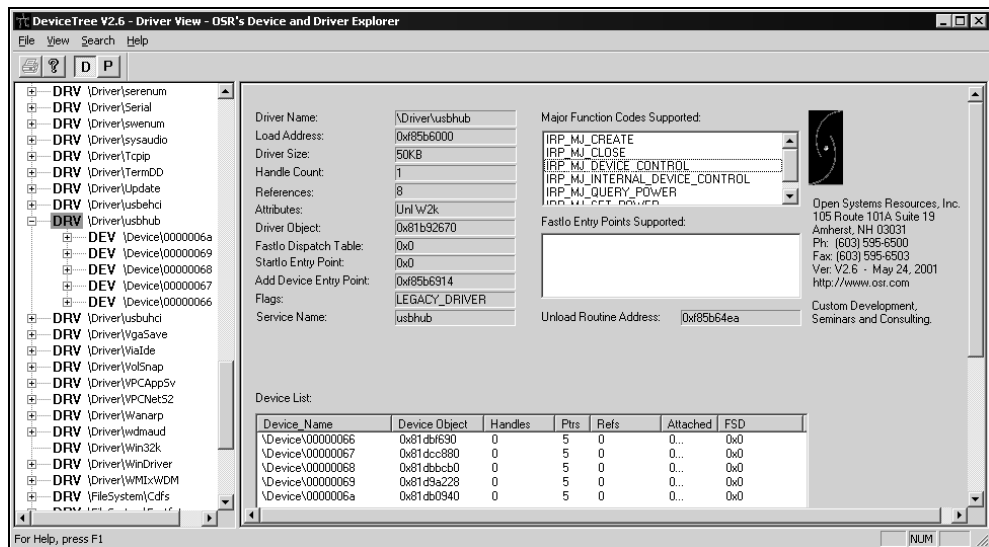


Рис. 19.4. Программа DeviceTree

19.2.2. DevCon

Консольная программа DevCon позволяет получить информацию о системе. Ниже приведены примеры использования этой программы с различными ключами.

Ключ *classes*

Ключ *classes* отображает список всех зарегистрированных классов системы (листинг 19.1).

Листинг 19.1. Программа Devcon с ключом *classes*

```
// Windows XP. Список приводится с сокращениями.
>F:\WINXPDDK\tools\devcon\i386\devcon.exe classes
Listing 49 setup class(es).
WCEUSBS      : Windows CE USB Devices
USB          : Universal Serial Bus controllers
CDROM        : DVD/CD-ROM drives
Computer     : Computer
DiskDrive    : Disk drives
Display      : Display adapters
```

fdc	: Floppy disk controllers
hdc	: IDE ATA/ATAPI controllers
Keyboard	: Keyboards
MEDIA	: Sound, video and game controllers
Modem	: Modems
Monitor	: Monitors
Mouse	: Mice and other pointing devices
MTD	: PCMCIA and Flash memory devices
Ports	: Ports (COM & LPT)
Printer	: Printers
System	: System devices
Unknown	: Other devices
FloppyDisk	: Floppy disk drives
Processor	: Processors
HIDClass	: Human Interface Devices
LegacyDriver	: Non-Plug and Play Drivers

Ключ *driverfiles*

Ключ *driverfiles* отображает список драйверов, соответствующих выбранному классу устройств или весь список драйверов (опция "*"). Пример вызова показан в листинге 19.2.

Листинг 19.2. Программа Devcon с ключом *driverfiles*

```
// Windows XP. Список приводится с сокращениями.
>F:\WINXPDDK\tools\devcon\i386\devcon.exe driverfiles *
ACPI\AUTHENTICAMD_-X86_FAMILY_6_MODEL_10\_0
    Name: AMD Athlon(tm) XP 2200+
    Driver installed from f:\winxp\inf\cpu.inf [Processor_Inst]. 1
file(s) used by driver:
    F:\WINXP\System32\DRIVERS\processr.sys
ACPI\FIXEDBUTTON\2&DABA3FF&0
    Name: ACPI Fixed Feature Button
    Driver installed from f:\winxp\inf\machine.inf [NO_DRV]. No files
used by driver.
ACPI\PNP0000\3&61AAA01&0
    Name: Programmable interrupt controller
    Driver installed from f:\winxp\inf\machine.inf [NO_DRV_PIC]. No files
used by driver.
```

ACPI\PNP0100\3&61AAA01&0

Name: System timer

Driver installed from f:\winxp\inf\machine.inf [NO_DRV_X]. No files used by driver.

ACPI\PNP0200\3&61AAA01&0

Name: Direct memory access controller

Driver installed from f:\winxp\inf\machine.inf [NO_DRV_X]. No files used by driver.

ACPI\PNP0303\3&61AAA01&0

Name: Standard 101/102-Key or Microsoft Natural PS/2 Keyboard

Driver installed from f:\winxp\inf\keyboard.inf [STANDARD_Inst]. 2 file(s) used by driver:

F:\WINXP\System32\DRIVERS\i8042prt.sys

F:\WINXP\System32\DRIVERS\kbdclass.sys

ACPI\PNP0400\1

Name: Printer Port (LPT1)

Driver installed from f:\winxp\inf\msports.inf [LptPort]. 1 file(s) used by driver:

Ключ *hwids*

Ключ *hwids* отображает список аппаратных идентификаторов устройств системы (листинг 19.3).

Листинг 19.3. Программа Devcon с ключом *hwids*

```
// Windows XP. Список приводится с сокращениями.
>F:\WINXPDDK\tools\devcon\i386\devcon.exe hwids *
HID\VID_1241&PID_1111\6&25B17C15&0&0000
  Name: HID-compliant mouse
  Hardware ID's:
    HID\Vid_1241&Pid_1111&Rev_0100
    HID\Vid_1241&Pid_1111
    HID_DEVICE_SYSTEM_MOUSE
    HID_DEVICE_UP:0001_U:0002
    HID_DEVICE
PCI\VEN_1106&DEV_3038&SUBSYS_30381106&REV_81\3&61AAA01&0&80
  Name: VIA Rev 5 or later USB Universal Host Controller
  Hardware ID's:
    PCI\VEN_1106&DEV_3038&SUBSYS_30381106&REV_81
```

```
PCI\VEN_1106&DEV_3038&SUBSYS_30381106
PCI\VEN_1106&DEV_3038&CC_0C0300
PCI\VEN_1106&DEV_3038&CC_0C03
```

Compatible ID's:

```
PCI\VEN_1106&DEV_3038&REV_81
PCI\VEN_1106&DEV_3038
PCI\VEN_1106&CC_0C0300
PCI\VEN_1106&CC_0C03
PCI\VEN_1106
PCI\CC_0C0300
PCI\CC_0C03
```

```
USB\ROOT_HUB\4&319D8414&0
```

Name: USB Root Hub

Hardware ID's:

```
USB\ROOT_HUB&VID1106&PID3038&REV0081
USB\ROOT_HUB&VID1106&PID3038
USB\ROOT_HUB
```

Ключ *rescan*

Ключ *rescan* указывает системе начать поиск новых устройств. В отличие от аналогичной процедуры, запускаемой из менеджера устройств, в данном случае никаких оконных сообщений не отображается.

Ключ *stack*

Ключ *stack* собирает информацию обо всех устройствах в стеке выбранного класса (листинг 19.4).

Листинг 19.4. Программа Devcon с ключом *stack*

```
// Windows XP. Список приводится с сокращениями.
>F:\WINXPDDK\tools\devcon\i386\devcon.exe stack *USB*
USB\ROOT_HUB\4&1EE3D36F&0
Name: USB Root Hub
Setup Class: {36FC9E60-C465-11CF-8056-444553540000} USB
Class upper filters:
    hhdsusbh
Controlling service:
    ushub
```

USB\ROOT_HUB\4&319D8414&0

Name: USB Root Hub

Setup Class: {36FC9E60-C465-11CF-8056-444553540000} USB

Class upper filters:

hhdusbh

Controlling service:

usbhub

USB\ROOT_HUB\4&350B3C2C&0

Name: USB Root Hub

Setup Class: {36FC9E60-C465-11CF-8056-444553540000} USB

Class upper filters:

hhdusbh

Controlling service:

usbhub

USB\ROOT_HUB\4&C2A22CA&0

Name: USB Root Hub

Setup Class: {36FC9E60-C465-11CF-8056-444553540000} USB

Class upper filters:

hhdusbh

Controlling service:

usbhub

USB\ROOT_HUB2\4&205A5C46&0

Name: USB Root Hub

Setup Class: {36FC9E60-C465-11CF-8056-444553540000} USB

Class upper filters:

hhdusbh

Controlling service:

usbhub

5 matching device(s) found.

Ключ *status*

Ключ *status* отображает состояние драйверов системы (листинг 19.5).

Листинг 19.5. Программа Devcon с ключом *status*

```
// Windows XP. Список приводится с сокращениями.  
>F:\WINXPDDK\tools\devcon\i386\devcon.exe status *
```

PCI\VEN_1106&DEV_3038&SUBSYS_30381106&REV_81\3&61AAA01&0&80

Name: VIA Rev 5 or later USB Universal Host Controller

Driver is running.

HID\VID_1241&PID_1111\6&25B17C15&0&0000

Name: HID-compliant mouse

Driver is running.

ACPI\PNP0000\3&61AAA01&0

Name: Programmable interrupt controller

Device is currently stopped.

19.2.3. ChkInf и GenInf

Программа GenInf (рис. 19.5) позволяет сгенерировать INF-файл, последовательно отвечая на вопросы помощника, а программа ChkInf — проверить правильность INF-файла. Мы рассматривали эти программы в *разд. 10.4.11*.

The screenshot shows the 'Gen INF' application window. The title bar reads 'Gen INF'. The main area is titled 'General Information' and contains the instruction: 'Enter general information in the appropriate blanks. Press the "..." button to specify the location of your INF file.' Below this, there are two main sections: 'Manufacturer Info' and 'File Info'. The 'Manufacturer Info' section includes fields for 'Company Name', 'Abbreviated Company Name', and a 'Device Class' dropdown menu currently set to 'USB'. The 'File Info' section includes fields for 'Inf File Path\Name' (with a browse button), 'INF Location on Distribution Media', and 'Catalog File Name'. There is also an unchecked checkbox for 'Multi-Platform Catalog Files'. At the bottom of the window are four buttons: '< Back', 'Next >', 'Cancel', and 'Help'.

Рис. 19.5. Программа GenInf

19.3. Средства CompuWare Corporation

В поставку NuMega SoftICE Driver Suite (теперь CompuWare Corporation) кроме известного отладчика SoftICE входит несколько полезных утилит.

19.3.1. Monitor

Программа Monitor (рис. 19.6) позволяет динамически загружать, запускать, останавливать и выгружать драйверы, а также производить мониторинг загрузки драйверов.

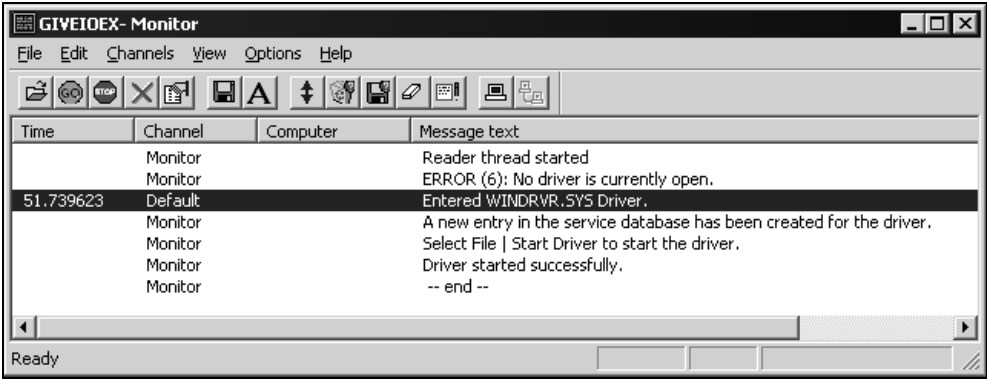


Рис. 19.6. Программа Monitor

19.3.2. SymLink

Программа SymLink (рис. 19.7) отображает список символьных имен, зарегистрированных в системе.

19.3.3. EzDriverInstaller

Программа EzDriverInstaller (рис. 19.8) позволяет запустить или остановить драйвер, записанный в выбранном INF-файле.

19.3.4. WdmSniff

Программа WdmSniff (рис. 19.9) позволяет производить мониторинг IRP-пакетов практически любого драйвера (рис. 19.10).

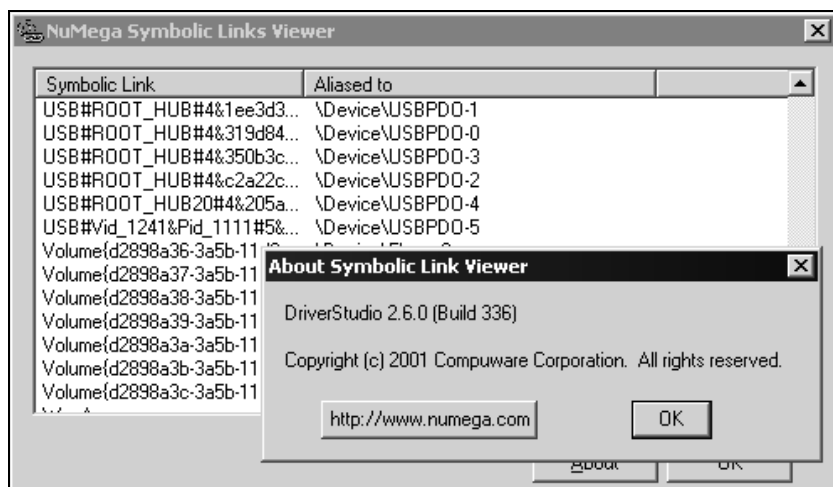


Рис. 19.7. Программа SymLink

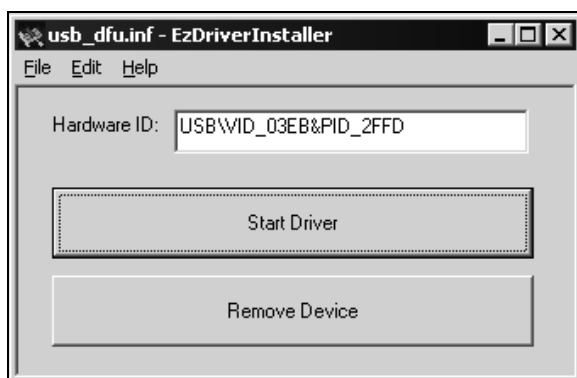


Рис. 19.8. Программа EzDriverInstaller

19.4. Средства SysInternals

19.4.1. WinObj

Программа WinObj (рис. 19.11) отображает список символьных имен, зарегистрированных в системе.

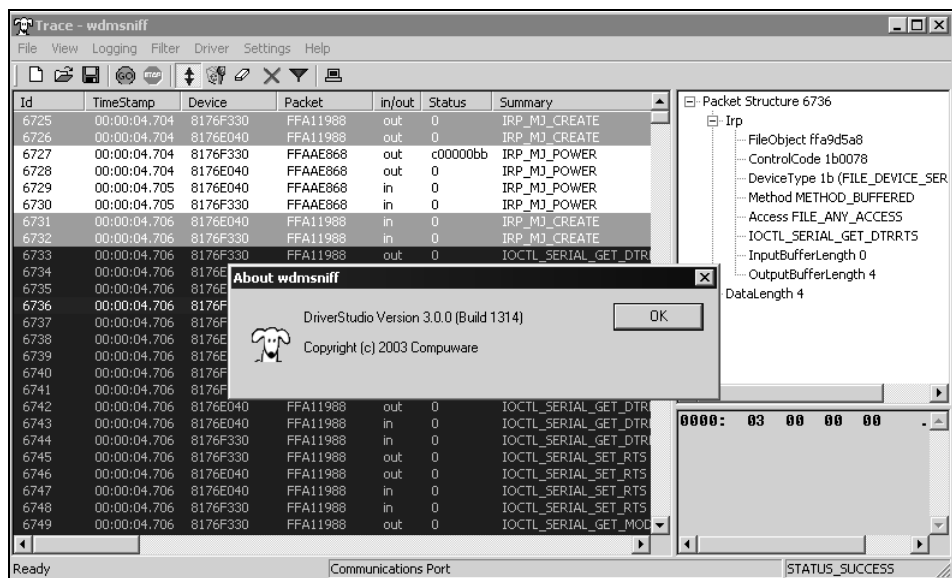


Рис. 19.9. Программа WdmSniff

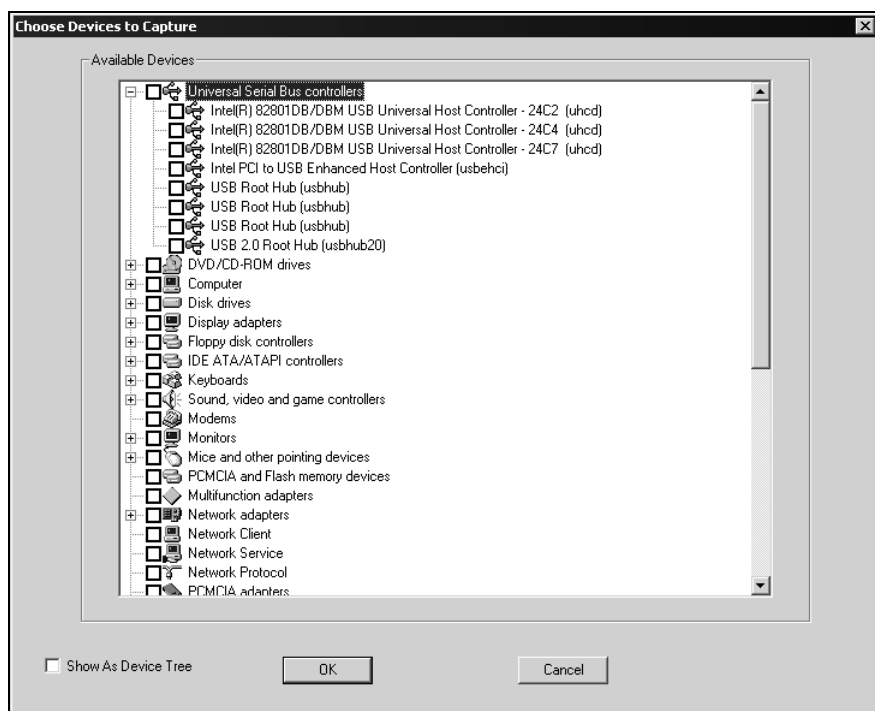


Рис. 19.10. Выбор устройств в программе WdmSniff

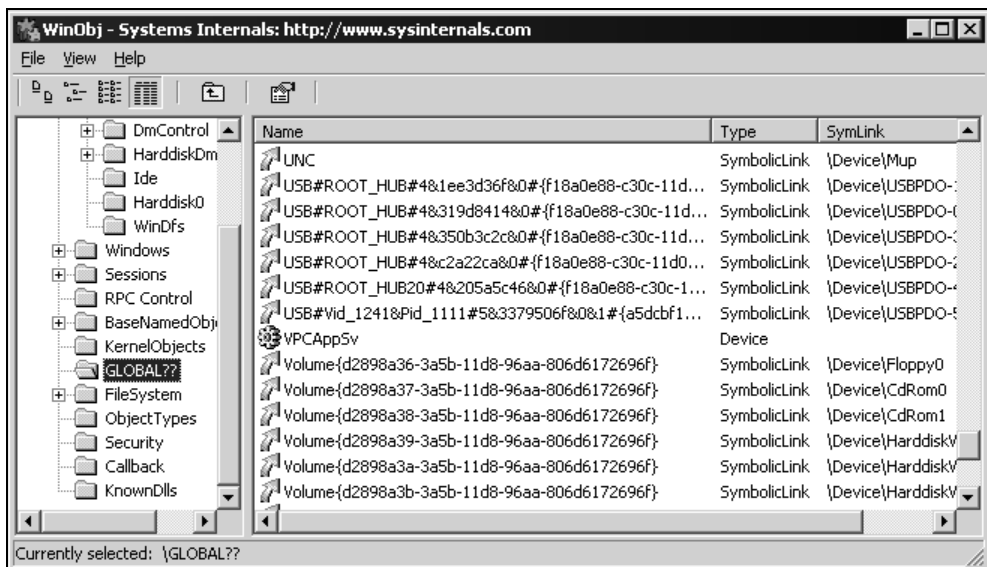


Рис. 19.11. Программа WinObj

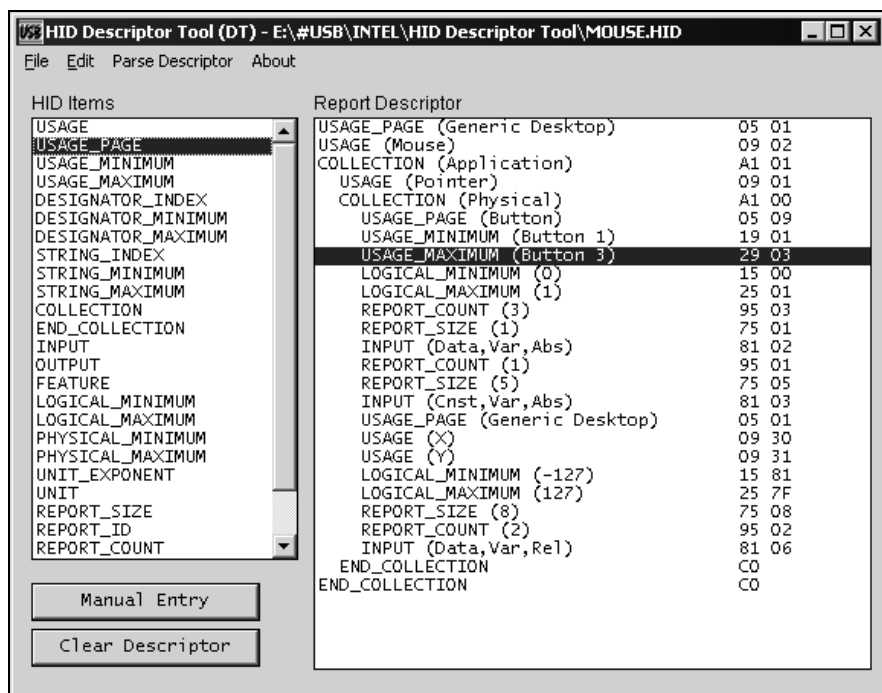


Рис. 19.12. Программа HID Descriptor Tool

19.5. Средства USB Forum

USB Forum (USB-IF, USB Implementers Forum, Inc.) — организация, утверждающая и разрабатывающая стандарты шины USB. Сайт этой организации (www.usb.org) содержит несколько удобных утилит, доступных для свободного использования. Кроме того, на сайте доступен форум разработчиков, на котором можно получить множество полезной информации и квалифицированную помощь.

19.5.1. HID Descriptor Tool

Программа HID Descriptor Tool (рис. 19.12) позволяет просто и достаточно удобно создавать и редактировать дескрипторы репортов для HID-устройств. Созданные дескрипторы можно сохранить в специальном файле с расширением `hid` для последующего редактирования, а также сохранить в виде фрагмента на языке ассемблера или C (листинг 19.6). Это незаменимый инструмент при разработке HID-устройств.

Листинг 19.6. Сохранение дескриптора репорта в разных форматах

```
// Сохранение в формате заголовочного файла C
// E:\#USB\INTEL\HID Descriptor Tool\MOUSE.HID.h
char ReportDescriptor[50] = {
    0x05, 0x01,           // USAGE_PAGE (Generic Desktop)
    0x09, 0x02,           // USAGE (Mouse)
    0xa1, 0x01,           // COLLECTION (Application)
    0x09, 0x01,           //     USAGE (Pointer)
    0xa1, 0x00,           //     COLLECTION (Physical)
    0x05, 0x09,           //         USAGE_PAGE (Button)
    0x19, 0x01,           //         USAGE_MINIMUM (Button 1)
    0x29, 0x03,           //         USAGE_MAXIMUM (Button 3)
    0x15, 0x00,           //         LOGICAL_MINIMUM (0)
    0x25, 0x01,           //         LOGICAL_MAXIMUM (1)
    0x95, 0x03,           //         REPORT_COUNT (3)
    0x75, 0x01,           //         REPORT_SIZE (1)
    0x81, 0x02,           //         INPUT (Data,Var,Abs)
    0x95, 0x01,           //         REPORT_COUNT (1)
    0x75, 0x05,           //         REPORT_SIZE (5)
    0x81, 0x03,           //         INPUT (Cnst,Var,Abs)
    0x05, 0x01,           //     USAGE_PAGE (Generic Desktop)
```

```

0x09, 0x30,      //  USAGE (X)
0x09, 0x31,      //  USAGE (Y)
0x15, 0x81,      //  LOGICAL_MINIMUM (-127)
0x25, 0x7f,      //  LOGICAL_MAXIMUM (127)
0x75, 0x08,      //  REPORT_SIZE (8)
0x95, 0x02,      //  REPORT_COUNT (2)
0x81, 0x06,      //  INPUT (Data,Var,Rel)
0xc0,            //  END_COLLECTION
0xc0            //  END_COLLECTION

```

```
};
```

```
// Сохранение в виде данных ассемблера
```

```

db 5h, 1h      ; USAGE_PAGE (Generic Desktop)
db 9h, 2h      ; USAGE (Mouse)
db a1h, 1h     ; COLLECTION (Application)
db 9h, 1h      ;  USAGE (Pointer)
db a1h, 0h     ;  COLLECTION (Physical)
db 5h, 9h      ;    USAGE_PAGE (Button)
db 19h, 1h     ;    USAGE_MINIMUM (Button 1)
db 29h, 3h     ;    USAGE_MAXIMUM (Button 3)
db 15h, 0h     ;    LOGICAL_MINIMUM (0)
db 25h, 1h     ;    LOGICAL_MAXIMUM (1)
db 95h, 3h     ;    REPORT_COUNT (3)
db 75h, 1h     ;    REPORT_SIZE (1)
db 81h, 2h     ;    INPUT (Data,Var,Abs)
db 95h, 1h     ;    REPORT_COUNT (1)
db 75h, 5h     ;    REPORT_SIZE (5)
db 81h, 3h     ;    INPUT (Cnst,Var,Abs)
db 5h, 1h      ;    USAGE_PAGE (Generic Desktop)
db 9h, 30h     ;    USAGE (X)
db 9h, 31h     ;    USAGE (Y)
db 15h, 81h    ;    LOGICAL_MINIMUM (-127)
db 25h, 7fh    ;    LOGICAL_MAXIMUM (127)
db 75h, 8h     ;    REPORT_SIZE (8)
db 95h, 2h     ;    REPORT_COUNT (2)
db 81h, 6h     ;    INPUT (Data,Var,Rel)
db c0h         ;    END_COLLECTION
db c0h         ; END_COLLECTION

```

19.6. Средства HDD Software

Программа USB Monitor (рис. 19.13) разработана компанией HDD Software (www.hhdsoftware.com) и является, пожалуй, наиболее мощной программой мониторинга USB-пакетов. Программа имеет простой и удобный интерфейс.

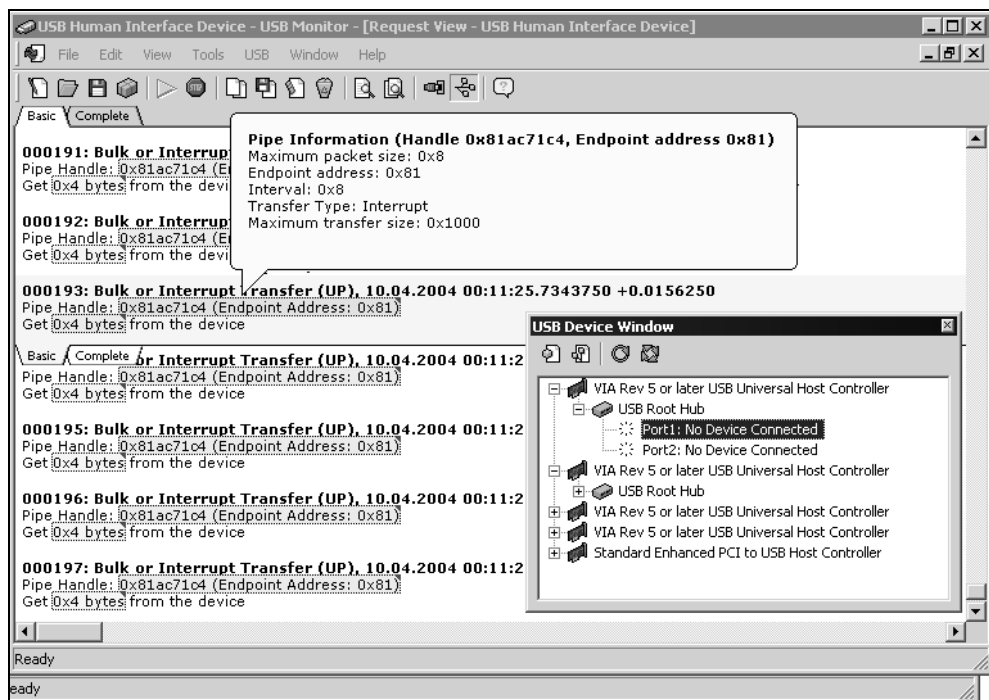


Рис. 19.13. Программа USB Monitor

19.7. Средства Sourceforge

Для мониторинга USB-трафика можно использовать программу SnoopPro (рис. 19.14), разрабатываемую в рамках проектов с открытым исходным кодом. Программа доступна на сайте sourceforge.net/projects/usbsnoop.

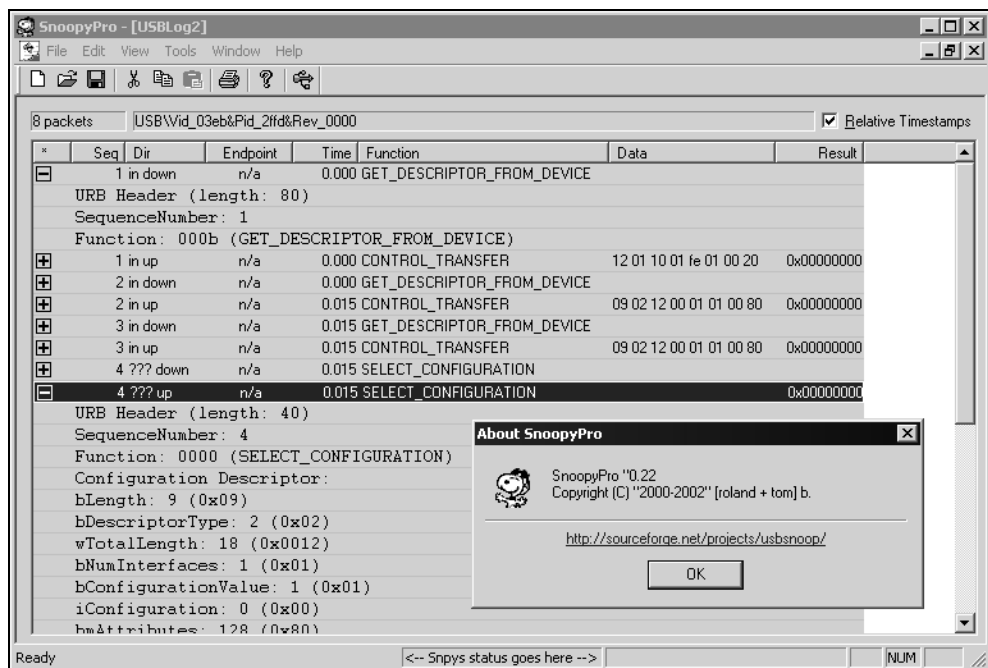
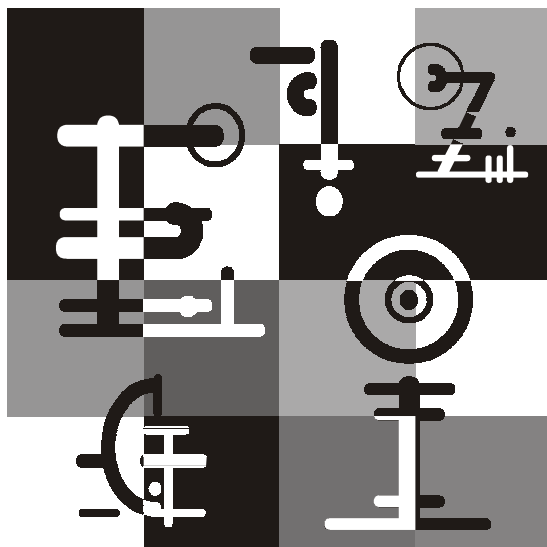
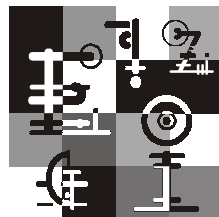


Рис. 19.14. Программа SnoopyPro



ПРИЛОЖЕНИЯ

Приложение 1



Дополнительные функции

Для преобразования BCD-чисел в строку номера версии мы используем очень простую функцию, код которой показан в листинге П1.1

Листинг П1.1. Преобразование BCD-числа в строку

```
function BCD2Str(Value : Word) : String;
begin
    Result:=
        IntToHex(Value shr 8, 2) +    // старшая часть версии
        '.' +                          // разделитель
        IntToHex(Value and $00FF, 2); // младшая часть версии
end;
```

В DOS-программах мы пользовались функциями преобразования числа в шестнадцатеричную строку. Ради экономии места мы не приводили эти функции в исходных листингах, но собрали такие функции в листинге П1.2.

Листинг П1.2. Дополнительные функции преобразования

```
Unit StrFunc;

INTERFACE

Function Byte2Hex(B: Byte):String;
Function Long2Hex(B: Longint):String;
Function Pointer2Hex(P : Pointer):String;

IMPLEMENTATION
```

```
Function Byte2Hex(B: Byte):String;  
Const hStr : String ='0123456789ABCDEF';  
Begin  
    Byte2Hex:= hStr[(B div 16)+1]+ hStr[(B mod 16)+1];  
End;  
  
Function Long2Hex(B: Longint):String;  
Begin  
    Long2Hex:=  
        Byte2Hex((B and $0FF000000) shr 24) +  
        Byte2Hex((B and $000FF0000) shr 16) +  
        Byte2Hex((B and $00000FF00) shr 8) +  
        Byte2Hex((B and $0000000FF) shr 0);  
End;  
  
Function Pointer2Hex(P : Pointer):String;  
Begin  
    Pointer2Hex:= Long2Hex(Seg(P^))+':'+Long2Hex(Ofs(P^));  
End;  
  
END.
```

Приложение 2

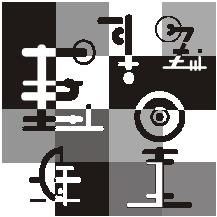


Таблица идентификаторов языков (LangID)

Список идентификаторов содержится в спецификации USB_LANGID. Актуальный на данный момент список приведен в табл. П2.1.

Таблица П2.1. Таблица идентификаторов языков

ID	Язык	ID	Язык	ID	Язык
0x041c	Albanian	0x3409	English (Philippines)	0x0446	Punjabi.
0x0401	Arabic (Saudi Arabia)	0x0425	Estonian	0x0418	Romanian
0x0801	Arabic (Iraq)	0x0438	Faeroese	0x0419	Russian
0x0c01	Arabic (Egypt)	0x0429	Farsi	0x044f	Sanskrit.
0x1001	Arabic (Libya)	0x040b	Finnish	0x0c1a	Serbian (Cyrillic)
0x1401	Arabic (Algeria)	0x040c	French (Standard)	0x081a	Serbian (Latin)
0x1801	Arabic (Morocco)	0x080c	French (Belgian)	0x0459	Sindhi
0x1c01	Arabic (Tunisia)	0x0c0c	French (Canadian)	0x041b	Slovak
0x2001	Arabic (Oman)	0x100c	French (Switzerland)	0x0424	Slovenian
0x2401	Arabic (Yemen)	0x140c	French (Luxembourg)	0x040a	Spanish (Traditional Sort)
0x2801	Arabic (Syria)	0x180c	French (Monaco)	0x080a	Spanish (Mexican)

Таблица П2.1 (продолжение)

ID	Язык	ID	Язык	ID	Язык
0x2c01	Arabic (Jordan)	0x0437	Georgian.	0x0c0a	Spanish (Modern Sort)
0x3001	Arabic (Lebanon)	0x0407	German (Standard)	0x100a	Spanish (Guatemala)
0x3401	Arabic (Kuwait)	0x0807	German (Switzerland)	0x140a	Spanish (Costa Rica)
0x3801	Arabic (U.A.E.)	0x0c07	German (Austria)	0x180a	Spanish (Panama)
0x3c01	Arabic (Bahrain)	0x1007	German (Luxembourg)	0x1c0a	Spanish (Dominican Republic)
0x4001	Arabic (Qatar)	0x1407	German (Liechtenstein)	0x200a	Spanish (Venezuela)
0x042b	Armenian.	0x0408	Greek	0x240a	Spanish (Colombia)
0x044d	Assamese.	0x0447	Gujarati.	0x280a	Spanish (Peru)
0x042c	Azeri (Latin)	0x040d	Hebrew	0x2c0a	Spanish (Argentina)
0x082c	Azeri (Cyrillic)	0x0439	Hindi.	0x300a	Spanish (Ecuador)
0x042d	Basque	0x040e	Hungarian	0x340a	Spanish (Chile)
0x0423	Belarusian	0x040f	Icelandic	0x380a	Spanish (Uruguay)
0x0445	Bengali.	0x0421	Indonesian	0x3c0a	Spanish (Paraguay)
0x0402	Bulgarian	0x0410	Italian (Standard)	0x400a	Spanish (Bolivia)
0x0455	Burmese	0x0810	Italian (Switzerland)	0x440a	Spanish (El Salvador)
0x0403	Catalan	0x0411	Japanese	0x480a	Spanish (Honduras)

Таблица П2.1 (продолжение)

ID	Язык	ID	Язык	ID	Язык
0x0404	Chinese (Taiwan)	0x044b	Kannada.	0x4c0a	Spanish (Nicaragua)
0x0804	Chinese (PRC)	0x0860	Kashmiri (India)	0x500a	Spanish (Puerto Rico)
0x0c04	Chinese (Hong Kong SAR, PRC)	0x043f	Kazakh	0x0430	Sutu
0x1004	Chinese (Singapore)	0x0457	Konkani.	0x0441	Swahili (Kenya)
0x1404	Chinese (Macau SAR)	0x0412	Korean	0x041d	Swedish
0x041a	Croatian	0x0812	Korean (Johab)	0x081d	Swedish (Finland)
0x0405	Czech	0x0426	Latvian	0x0449	Tamil.
0x0406	Danish	0x0427	Lithuanian	0x0444	Tatar (Tatarstan)
0x0413	Dutch (Netherlands)	0x0827	Lithuanian (Classic)	0x044a	Telugu.
0x0813	Dutch (Belgium)	0x042f	Macedonian	0x041e	Thai
0x0409	English (United States)	0x043e	Malay (Malaysian)	0x041f	Turkish
0x0809	English (United Kingdom)	0x083e	Malay (Brunei Darussalam)	0x0422	Ukrainian
0x0c09	English (Australian)	0x044c	Malayalam.	0x0420	Urdu (Pakistan)
0x1009	English (Canadian)	0x0458	Manipuri	0x0820	Urdu (India)
0x1409	English (New Zealand)	0x044e	Marathi.	0x0443	Uzbek (Latin)
0x1809	English (Ireland)	0x0861	Nepali (India).	0x0843	Uzbek (Cyrillic)
0x1c09	English (South Africa)	0x0414	Norwegian (Bokmal)	0x042a	Vietnamese

Таблица П2.1 (окончание)

ID	Язык	ID	Язык	ID	Язык
0x2009	English (Jamaica)	0x0814	Norwegian (Nynorsk)	0x04ff	HID (Usage Data Descriptor)
0x2409	English (Caribbean)	0x0448	Oriya.	0xf0ff	HID (Vendor Defined 1)
0x2809	English (Belize)	0x0415	Polish	0xf4ff	HID (Vendor Defined 2)
0x2c09	English (Trinidad)	0x0416	Portuguese (Brazil)	0xf8ff	HID (Vendor Defined 3)
0x3009	English (Zimbabwe)	0x0816	Portuguese (Standard)	0xfcff	HID (Vendor Defined 4)

Приложение 3

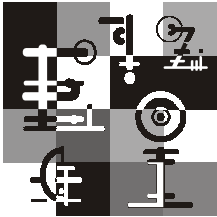


Таблица кодов производителей (Vendor ID, Device ID)

В табл. ПЗ.1 приведены значения некоторых производителей (Vendor ID) и классов устройств (Device ID).

Таблица ПЗ.1. Некоторые коды Vendor ID и Device ID

Vendor ID	Компания	Device ID	Устройство
0E11	Compaq Computer Corp.		
		7020	USB Controller
		A0F8	USB Open Host Controller
100B	National Semiconductor		
		0012	USB Controller
1008	Epson		
1014	IBM		
100A	Phoenix Technologies		
101C	Western Digital		
1022	Advanced Micro Devices (AMD)		
		7404	AMD 755 PCI to USB Open Host Controller
		740C	AMD 756 USB Controller
1023	Trident Microsystems		
1025	Acer Labs Incorporated (ALI)		
		5237	M5237 PCI USB Host Controller
1028	Dell Computer Corp		

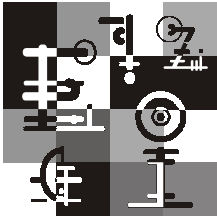
Таблица ПЗ.1 (продолжение)

Vendor ID	Компания	Device ID	Устройство
1029	Siemens Nixdorf AG		
102B	Matrox Graphics Inc		
1032	Compaq		
1033	NEC Electronics Hong Kong		
		0035	uPD9210FGC-7EA USB Host Controller
1039	Silicon Integrated Systems (SiS)		
		7001	SiS5571 USB Host Controller
103C	Hewlett-Packard Company		
1043	Asustek Computer Inc		
1045	OPTi Inc		
		A0F8	82C750 PCI USB Controller
		C861	82C861 FireLink PCI-to-USB Bridge
104C	Texas Instruments (TI)		
104D	Sony Corp		
106B	Apple Computer Inc		
1075	Advanced Integration Research		
1076	Chaintech Computer Co Ltd		
1095	CMD Technology Inc		
		0670	USB0670 USB Controller
		0673	USB0673 USB Controller
10B7	3COM Corp		
10B9	Acer Labs Incorporated (ALI)		
		5237	ALI M5237 USB Host Controller
1106	VIA Technologies Inc		
		3038	VT83C572 USB Controller
		0571	VIA AMD-645 USB controller

Таблица ПЗ.1 (окончание)

Vendor ID	Компания	Device ID	Устройство
1114	Atmel Corp		
1121	Zilog		
1131	Philips Semiconductors		
11C0	HEWLETT PACKARD		
11C1	AT&T Microelectronics (Lucent)		
		5801	USB Open Host Controller
8086	Intel Corporation		
		2412	82801AA USB Controller
		2422	82801AB USB Controller
		2428	82801AB Hub Interface-to-PCI Bridge
		2442	82801BA USB Controller
		2444	82801BA USB Controller
		244E	82801BA Hub Interface to PCI Bridge
		7020	82371SB PIIX3 USB Host Controller (Triton II)
		7112	82371AB PIIX4 USB Interface
		719A	82443MX USB Universal Host Controller
		7602	82372FB PCI to USB Universal Host Controller

Приложение 4



Описание компакт-диска

Папки	Описание	Разделы
\AT89C5131\AT89—BULK_IN	Использование конечной точки типа BULK. Программа для микроконтроллера AT89C5131, драйвер для Windows и тестовая программа	13.5
\AT89C5131\AT89—INT_IN	Использование конечной точки типа INTERRUPT. Программа для микроконтроллера AT89C5131, драйвер для Windows и тестовая программа	13.5
\AT89C5131\AT89—HID-read	HID-устройство. Программа для микроконтроллера AT89C5131 и тестовая программа	13.5
\AT89C5131\Scheme	Схема тестовой платы для AT89C5131	13.3
\AT89C5131\AT89—HID-feature	HID-устройство с FEATURE-репортом. Программа для микроконтроллера AT89C5131 и тестовая программа	13.5
\AT89C5131\AT89—HID-2input	HID-устройство с двумя INPUT-репортами. Программа для микроконтроллера AT89C5131 и тестовая программа.	13.8
\AT89C5131\AT89-HID-2feature	HID-устройство с двумя FEATURE-репортами. Программа для микроконтроллера AT89C5131 и тестовая программа	13.8
\CONTLIST\	Класс TUSBController и его использование для получения списка USB-устройств в DOS	11.2
\CRCCalc\	Пример вычисления контрольных сумм	3.9
\DeviceName\	Пример работы с символьными именами	9.2
\Enumerator\	Поиск и нумерация устройств в Windows	10.3

(окончание)

Папки	Описание	Разделы
\FullEnumeration\	Поиск USB-устройств и чтение их дескрипторов и характеристик	10.3
\HIDTest\	Пример чтения дескрипторов и характеристик HID-устройства	8.8
\StartAddHardware\	Запуск процедуры поиска нового оборудования	10.1.2
\USB_in_DOS\	Работа с USB в DOS (INT 1AH)	11.2
\WDMTest\	Пример WDM-драйвера и обращения к нему	9.3, 9.4

Литература

1. Axelson Jan. USB Complete. — Lakeview Research, 2001.
2. John Hyde. USB Design by Example. A Practical Guide to Building I/O Devices. — Intel Press, 2001.
3. Агуров П. В. Последовательные интерфейсы. Практика программирования. — СПб.: БХВ-Петербург, 2004.
4. Гук М. Аппаратные средства IBM PC. Энциклопедия. — СПб.: Питер, 2002.
5. Гук М. Аппаратные интерфейсы IBM PC. Энциклопедия. — СПб.: Питер, 2002.
6. Кулаков В. Программирование на аппаратном уровне: специальный справочник. 2-е изд. — СПб.: Питер, 2003.
7. Солдатов В. П. Программирование драйверов Windows. — М.: Бином, 2004.
8. Соломон Д., Руссанович М. Внутреннее устройство Windows 2000. Структура и алгоритмы работы компонентов Windows 2000 и NTFS 5. — СПб.: Питер, Microsoft Press, 2001.
9. Рихтер Д. Windows для профессионалов — СПб.: Питер, Microsoft Press, 1999.
10. Фролов А. В., Фролов Г. В. Защищенный режим процессоров Intel 80286/80386/80486. — М.: Диалог-МИФИ, 1993.

Предметный указатель

A, B

ACK 92
BIOS 38
Boot Device 151

C

Control Read 93
Control Write 93
CRC 83, 85

D

Descriptor:
 configuration 109
 device 105
 endpoint 113
 hub 127
 interface 111
 qualifier 108
 report 150
 standard device 106
 string 116

F

FDO 183
FiDO 183

H

HID 149, 377
HNP 139
Hub 123

I

IN 83
IRP 66, 196

L, N

LANGID 117
NAK 92
NRZI 57

O

OTG 137
OUT 83

P

PDO 183
PID 81
PING 83
Pipe 79
Plug and Play:
 процедуры 222
 спецификация 35
 функции 221

S, W

SETUP 83
Setup Packet 96
SOF 83
SRP 139
STALL 92
SYNC 81
WDM 144

Д

Дескриптор:

- HID 154, 377
- OTG 139
- интерфейса 111
- конечной точки 113, 374
- конфигурации 109
- отображение 120
- порядок получения 117
- репорта 150, 156
- специфический 117
- стандартный 106
- строки 116, 369
- тип 98
- устройства 105
- уточняющий 108
- хаба 126, 127
- хоста (получение) 146

Драйвер:

- получение списка 182
- фильтра 181
- функциональный 181
- шины 181

З

Запрос:

- a_alt_hnp_support 140
- a_hnp_support 140
- b_hnp_enable 140
- CLEAR_FEATURE 100
- CLEAR_HUB_FEATURE 130
- CLEAR_PORT_FEATURE 130
- GET_BUS_STATE 131
- GET_CONFIGURATION 103, 360
- GET_DESCRIPTOR 102, 139, 153, 356
- GET_HUB_DESCRIPTOR 131
- GET_HUS_STATUS 131
- GET_IDLE 168, 170
- GET_INTERFACE 103
- GET_PORT_STATUS 132
- GET_PROTOCOL 168, 171
- GET_REPORT 150, 168, 169, 386

- GET_STATUS 99, 362
- IRP_MJ_xxx 203
- SET_ADDRESS 101, 363
- SET_CONFIGURATION 103, 361
- SET_DESCRIPTOR 102
- SET_FEATURE 101, 140
- SET_HUB_DESCRIPTOR 134
- SET_HUB_FEATURE 134
- SET_IDLE 168, 170
- SET_INTERFACE 104
- SET_PORT_FEATURE 134
- SET_PROTOCOL 168, 171
- SET_REPORT 150, 168, 169
- SYNC_FRAME 77, 104
- обработка 104
- стандартный 97
- формирование 147
- хаба 129

И

Идентификатор пакета 81

Идентификация:

- HID 152
- устройств 58

Иконка USB 14, 46

Интерфейс 81, 111

Интерфейс USB:

- архитектура 16
- возможности 12
- история создания 12
- логическая архитектура 17
- логические уровни 65
- механизм передачи данных 32
- обозначение 14
- ограничения 45
- питание 59
- свойства 12
- физическая архитектура 16
- физический интерфейс 55

К

Кабель:

- длина 53
- типы 52

требования 51
 цветовая гамма 55
Кадр 32, 77
Канал 32, 79
Канал сообщений 80
Конечная точка 32, 78
 атрибуты 115
 блокировка 101
 максимальное число точек 79
 определение 17
Контроллер хаба 124
Контрольная сумма 83, 85
Концентратор 18
Корневой хаб 16

М

Маркер. См. Пакет
Микрокадр 77

Н

Нумерация 117, 353

О

Обратная связь:
 неявная 75
 согласование частот 75
 явная 76
Основной канал сообщений 80

П

Пакет 81
 Data0 84
 Data1 84
 Data2 84
 IN 83
 IRP 66, 196
 MData 84
 OUT 83
 PING 83
 PRE 135
 SETUP 83, 91, 96

SOF 74, 77, 83
SPLIT 84
квитирование 92
маркер 90
подтверждение 84

Передачи:

 данных с подтверждением 91
 изохронные 72, 91, 95
 массивов данных 71
 по прерываниям 71, 95
 прием с подтверждением 91
 приоритеты 72
 управляющие 71, 91

Питание:

 от шины 59
 собственное 60

Побудка 19, 101

Повторитель 124

Поле:

 bmRequestType 97

Полоса пропускания 67

Порт 18

 восходящий 18, 123
 нисходящий 18, 123

Посылка:

 без данных 93
 запись данных 93
 чтение данных 93

Поток 79

Преобразователь интерфейса 22, 31

Прерывания 32

Приостановка 19

Процедура:

 AddDevice 192
 DriverEntry 190
 Unload 194
 рабочие процедуры 198

Р

Разъемы 53, 138

Регистр AT89C5131

 UBYCTLX 337

 UEPCONX 330

(окончание рубрики см. на с. 552)

Регистр AT89C5131 (окончание):

UEPDATX 337
UEPIEN 336
UEPINT 335
UEPNUM 329
UEPRST 334
UEPSTAX 331
UFNUMH 338
UFNUML 338
USBADDR 326
USBCON 324
USBIEN 328
USBINT 327

Режимы передачи данных 34

Ренумерация 59

Репорт:

FEATURE 385
INPUT 381
номер 152
определение 150

С

Сетевое соединение 30

Символьное имя 184

Слово состояния:

интерфейса 100
конечной точки 100
устройства 100

Состояние:

Connect 57
Disabled 20
Disconnect 57
Disconnected 19
Enabled 20
EOP 57
Idle State 57
J 57
K 57
Powered Off 19
Reset 57
Resume State 57
SOP 57
Suspended 20

Стаффинг 58

Структура:

OVERLAPPED 474

Т

Транзакции 90

планирование 68

транслятор 124, 135

У

Удаленная побудка 61

Устранение проблем 41

Устройства:

звуковые колонки 24
измерительная техника 28
клавиатура 21
конфигурирование 35
модем 23
монитор 21
мышь 21
преобразователь интерфейса 22
примеры 20
сканер 23
флеш-диски 25

Устройство 18

HID 149, 377
адаптивное 75
асинхронное 74
двухролевое 137, 138
загрузочное 151, 153
логическое 18
свойства 18
синхронное 75
типы 145
устройство А 138
устройство В 138

Ф

Файл INF 152

структура 234

Фрейм 32

Функции 18

BIOS 1AH 251
CloseHandle 471
CreateFile 143, 172, 471
DefineDosDevice 185, 489
DeviceIoControl 120, 146, 203, 485
GetOverlappedResult 474, 484
HidD_FreePreparsedData 395, 493
HidD_GetAttributes 173, 390, 495
HidD_GetFeature 173, 391, 401, 494
HidD_GetHidGuid 173, 388, 492
HidD_GetIndexedString 396, 498
HidD_GetInputReport 396, 401, 498
HidD_GetManufacturerString
390, 496
HidD_GetNumInputBuffers 495
HidD_GetPreparsedData 173,
391, 493
HidD_GetProductString 390, 497
HidD_GetSerialNumberString 497
HidD_Hello 492
HidD_SetFeature 173, 494
HidD_SetNumInputBuffers 495
HidD_SetOutputReport 499
HidP_GetCaps 173, 390, 499
HidP_MaxDataListLength 500

QueryDosDevice 185, 487
ReadFile 391, 473
ReadFileEx 474, 477
SetupDiGetClassDevs 388
SetupDiGetDeviceInterfaceDetail
388
WaitForMultipleObjects 483
WaitForSingleObject 482
WriteFile 475
WriteFileEx 476, 480

X

Хаб 28, 123
 корневой 16, 18
 определение 16, 18
 свойства 19
Хост:
 обязанности 20
 определение 16
Хост-контроллер 18

Э

Энергопотребление 60, 111