Михаил Гук

ЭНЦИКЛОПЕДИЯ

Наиболее полное и подробное руководство

ШИНЫ PCI, USB и FireWire

ПИТЕР®

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Новосибирск · Ростов-на-Дону · Екатеринбург · Самара
Киев · Харьков · Минск
2005

ББК 32.973.2-04я22
УДК 004.3(03)
Г93

Гук М. Ю.
Г93 Шины PCI, USB и FireWire. Энциклопедия. — СПб.: Питер, 2005. — 540 с.: ил.

ISBN 5-469-00002-8

Книга охватывает шины PCI версий 1.0–3.0, PCI-X версий 1.0 и 2.0, PCI Express, порт AGP 1.0–3.0. Рассмотрены различные конструктивные исполнения шин, включая малогабаритные карты и конструктивы промышленных и инструментальных компьютеров. Шина USB рассмотрена в версиях 1.0–2.0, включая и дополнение OTG (On The Go). Приводится описание интерфейсов и структур данных всех типов хост-контроллеров USB: UHC, OHC и EHC. Описание FireWire охватывает стандарты IEEE 1394-1995, 1394a-2000 и 1394b-2002 с рассмотрением всех возможных типов кабельной шины и физической кросс-шины. Приводятся описания контроллера OHC 1394 и протокола SBP-2.

Книга адресована разработчикам аппаратных и программных средств компьютеров и периферийных устройств.

ББК 32.973.2-04я22
УДК 004.3(03)

ISBN 5-469-00002-8

Краткое содержание

Содержание

От автора

Пауза после выхода к читателям книги «Аппаратные интерфейсы ПК. Энциклопедия» несколько затянулась... Одной из причин тому стала активная деятельность по применению накопленных знаний и поиску новых сведений — разработка периферийных устройств для ПК. В ходе ее углубилось понимание устройства и работы различных интерфейсов, а также идеологии взаимодействия подсистем компьютера. Я с удовольствием делюсь с читателями накопленными знаниями.

Первоначально в данной книге планировалось ограничиться шинами PCI и USB. По этим темам удалось найти много технической информации. Также имеется практический опыт разработок устройств. В том виде, в каком эти темы представлены в данной книге, они вошли в мою книгу «PC Hardware Interfaces: A Developer's Reference by Michael Gook» ISBN, 1-931769-29-X, выпущенную издательством A-List Publishing (партнер БХВ-Петербург) в феврале 2004 г. на английском языке. Эта книга заинтересовала не только англоязычных читателей: в своей книге «Интерфейс USB. Практика использования и программирования» (БХВ-Петербург, осень 2004г.) Павел Агуров приводит довольно обширные «цитаты» (на русском языке, к сожалению, с неточной ссылкой на первоисточник). В Интернете я с удивлением обнаружил перевод этой книги на польский язык.

Позже удалось выйти на источники информации по FireWire (поначалу эта тема казалась слишком платно-закрытой), но, к сожалению, до практического применения в разработках дело еще не дошло.

Книга посвящена самым популярным интерфейсам и начинается с краткого рассмотрения структуры интерфейсов ПК и способов их использования. В ней уделяется значительное внимание вопросам программного взаимодействия с устройствами, без которого они особой ценности в составе ПК и не представляют. Дальнейшие три части книги детально раскрывают различные стороны шин PCI, USB и FireWire. По каждой из шин рассматривается общее устройство и организация, собственно интерфейсы подключения устройств, а также вопросы программного взаимодействия с устройствами. В заключительной главе приводятся практические соображения и рекомендации по выбору интерфейса для подключения устройств собственной разработки.

Плодотворная работа над данной книгой не стала бы возможной без практики в ЦНИИ РТК и общения с коллективами разработчиков нестандартной периферии. Самое весомое достижение на этом поприще автора — успешное создание аппаратно-программного комплекса «ОСЦИГЕН». Комплекс состоит из периферийного устройства аналогового ввода-вывода (осциллограф и генератор), подклю-

ченного к ПК по интерфейсу USB, и его программной поддержки на ПК. В процессе разработки этого «многоэтажного» сооружения (схема, программируемая логика, программа микроконтроллера с интерфейсом USB, драйверы, библиотеки функций и собственно приложение) ряд тезисов данной книги был использован и проверен коллективом разработчиков, возглавляемым автором.

Другая работа — чтение лекций по архитектуре вычислительных систем и их интерфейсам на Факультете переподготовки специалистов СПбГПУ, тоже влияет на процесс написания книг. Это занятие отнимает значительное время, но зато заставляет более строго подходить к подаче и структурированию материала.

Для облегчения восприятия в книге используются шрифтовые выделения названий сигналов (Frame#, D+), инструкций, регистров и битов (`SYNC`). Курсивом выделены *ключевые понятия*, а также названия команд (*READ*), пакетов (*Data0*), состояний (*Idle*). Штриховка на рисунках, изображающих назначение регистров и программных структур, означает зарезервированные или не используемые поля.

На все технические вопросы и замечания по книге я готов ответить по электронной почте mgook@stu.neva.ru.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства http://www.piter.com вы найдете подробную информацию о наших книгах.

Введение

Общие вопросы организации интерфейсов

Прежде чем перейти к детальному обсуждению шин, рассмотрим ряд общих вопросов организации соединений в компьютерах.

Структура компьютера

PC-совместимые компьютеры, как и большинство других, построены по классической схеме фон-неймановской машины образца 1945 года. Согласно этой схеме, компьютер состоит из центрального процессора (ЦП, CPU), памяти и устройств ввода/вывода. *Процессор* исполняет программы, находящиеся в памяти; *память* предназначена для хранения программ и данных, доступных процессору; *устройства ввода/вывода* (УВВ) предназначены для связи с внешним миром. Время внесло небольшие коррективы в названия этих «трех китов», и сейчас то, что попадало под название УВВ, называют *периферийными устройствами*. Процессор (один или несколько), память и необходимые элементы, связывающие их между собой и другими устройствами, называют *центральной частью* или *ядром* компьютера (или просто *центром*).

Периферийные устройства (ПУ) — это все программно-доступные компоненты компьютера, не попавшие в его центральную часть. Их можно разделить по назначению на несколько классов:

- устройства хранения данных (устройства внешней памяти) — дисковые (магнитные, оптические, магнитооптические), ленточные (стримеры), твердотельные (карты, модули и устройства USB на флэш-памяти). Эти устройства используются для сохранения информации из памяти на энергонезависимые носители и загрузки этой информации в оперативную память. В каком виде хранится информация на этих устройствах, нам не так уж важно (главное — правильно считать то, что сохранили);
- устройства ввода/вывода, служащие для преобразования информации из внутреннего представления компьютера (биты и байты) в форму, понятную *окружающим*, и обратно. Сюда относятся дисплеи (устройства отображения, то есть вывода), клавиатура и мышь (устройства ввода), принтеры и сканеры, плотте-

ры и дигитайзеры, джойстики, акустические системы и микрофоны, телевизоры и видеокамеры, устройства телеуправления и телеметрии. Любопытно, что в этих парах обычно лидируют устройства вывода, появляющиеся в компьютерах раньше соответствующих устройств ввода. Под *окружающими* подразумевается и человек (и другие биологические объекты), и различные технические устройства (компьютер можно приспособить для управления любым оборудованием, были бы датчики и исполнительные устройства). В какую форму эти устройства преобразуют двоичную информацию — определяется их назначением;

- коммуникационные устройства, служащие для передачи информации между компьютерами и (или) их частями. Сюда относятся модемы (проводные, радио, оптические, инфракрасные...), адаптеры локальных и глобальных сетей. Здесь преобразование формы представления информации нужно только для передачи ее на расстояние.

Главным действием в компьютере является исполнение программного кода центральным процессором, и ЦП должен иметь возможность взаимодействия с ПУ.

Для обращения к периферийным устройствам в процессорах x86, применяемых в PC-совместимых компьютерах, специально выделено *пространство ввода/вывода* (I/O Space), отдельное от памяти. Размер пространства адресов ввода/вывода 64 Кбайт; в этой области могут располагаться регистры периферийных устройств разрядностью 1, 2 или 4 байт, и для обращения к ним имеется несколько специальных инструкций процессора (`IN`, `OUT`, `INS` и `OUTS`). Регистры ПУ могут отображаться и в пространстве памяти — областях, свободных от оперативной и постоянной памяти. Заметим, что обособление пространства ввода/вывода используется далеко не во всех архитектурах процессоров. В любом случае разные регистры разных устройств не должны пересекаться по адресам в своих пространствах — в этом состоит требование *бесконфликтного распределения ресурсов* по адресам.

В плане взаимодействия с остальными компонентами компьютера процессор ничего не умеет, кроме как обращаться к ячейке (читать или писать байт, слово, двойное слово) пространства памяти или пространства ввода/вывода, а также реагировать на аппаратные прерывания. Таким образом, любое периферийное устройство должно представляться процессору набором регистров (ячеек) и, возможно, служить источником прерываний.

Иерархия и организация подключений

Компоненты компьютера соединяются друг с другом иерархией средств подключения, наверху которой стоят *интерфейсы системного уровня подключения*. Эта группа интерфейсов характерна тем, что в их транзакциях фигурируют физические адреса пространства памяти и (если есть) пространства ввода/вывода. Группа связанных между собой интерфейсов системного уровня образует логическую

системную шину компьютера. Системную шину образуют следующие физические интерфейсы:

- шина подключения центрального процессора (или нескольких процессоров в сложных системах) — FSB (Front Side Bus, фасадная шина)[1];
- шина подключения контроллеров памяти, оперативной и постоянной. Собственно шина памяти (Memory Bus) системной уже не является, поскольку в ней фигурируют не системные адреса, а адреса физических банков памяти;
- шины ввода/вывода, обеспечивающие связь между центральной частью компьютера и периферийными устройствами.

Типичные представители шин ввода/вывода в IBM PC — шина ISA (отмирающая) и шина PCI (развивающаяся в PCI-X и далее). Через шины ввода/вывода проходят все обращения ЦП к периферии. К шинам ввода/вывода подключаются *контроллеры* и *адаптеры*[2] периферийных устройств или их интерфейсов. Часть ПУ совмещена со своими контроллерами (адаптерами), как, например, сетевой адаптер Ethernet, подключенный к шине PCI. Другие же ПУ подключаются к своим контроллерам через промежуточные *периферийные интерфейсы*, являющиеся нижним уровнем иерархии подключений. Периферийные интерфейсы — самые разнообразные из всех аппаратных интерфейсов. К периферии, подключаемой через промежуточные интерфейсы, относится большинство устройств хранения (дисковые, ленточные), устройств ввода/вывода (дисплеи, клавиатуры, мыши, принтеры, плоттеры), ряд коммуникационных устройств (внешние модемы). Для взаимодействия с ПУ процессор обращается к регистрам контроллера, «представляющего интересы» подключенных к нему устройств.

По назначению периферийные интерфейсы можно разделить на специализированные и универсальные, выделенные и разделяемые:

- *специализированные* интерфейсы ориентированы на подключение устройств определенного узкого класса, и в них используются сугубо специфические протоколы передачи информации. Примеры — популярнейший интерфейс мониторов VGA, интерфейс накопителя на гибких дисках, традиционные интерфейсы клавиатуры и мыши, IDE/ATA и ряд других;
- *универсальные* интерфейсы имеют более широкое назначение, их протоколы обеспечивают доставку данных, не привязываясь к специфике передаваемой информации. Примеры — коммуникационные порты (COM), SCSI, USB, FireWire;
- *выделенные* интерфейсы позволяют подключить к одному порту (точке подключения) адаптера (контроллера) лишь одно устройство; число подключаемых устройств ограничено числом портов. Примеры — COM-порт, LPT-порт в стандартном режиме, интерфейс VGA-монитора, порт AGP, Serial SCSI;
- *разделяемые* интерфейсы позволяют подключить к одному порту адаптера множество устройств. Варианты физического подключения разнообразны: шина (жесткая, как ISA или PCI; кабельная шина SCSI и IDE/ATA), цепочка (daisy

[1] Это понятие в ряде источников отождествляют с системной шиной, но в данной книге будем пользоваться более широким толкованием понятия «системная шина».

[2] Контроллер отличается от адаптера более высоким уровнем «интеллекта»

chain) устройств (SCSI, LPT-порт в стандарте IEEE 1284.3), логическая шина на хабах (USB) или встроенных повторителях (IEEE 1394 FireWire).

Параллельные и последовательные интерфейсы

Для компьютеров и связанных с ним устройств наиболее распространенной является задача передачи дискретных данных, и, как правило, в значительных количествах (не один бит). Самый распространенный способ представления данных сигналами — двоичный: например, условно высокому (выше порога) уровню напряжения соответствует логическая единица, низкому — логический ноль (возможно и обратное представление). Для того чтобы передавать группу битов, используются два основных подхода к организации интерфейса:

- *параллельный интерфейс* — для каждого бита передаваемой группы используется своя сигнальная линия (обычно с двоичным представлением), и все биты группы передаются одновременно за один квант времени. Примеры: параллельный порт подключения принтера (LPT-порт, 8 бит), интерфейс ATA/ATAPI (16 бит), SCSI (8 или 16 бит), шина PCI (32 или 64 бита);
- *последовательный интерфейс* — используется лишь одна сигнальная линия, и биты группы передаются друг за другом по очереди; на каждый из них отводится свой квант времени (битовый интервал). Примеры: последовательный коммуникационный порт (COM-порт), последовательные шины USB и FireWire, PCI Express, интерфейсы локальных и глобальных сетей.

На первый взгляд организация параллельного интерфейса проще и нагляднее и этот интерфейс обеспечивает более быструю передачу данных, поскольку биты передаются сразу пачками. Очевидный недостаток параллельного интерфейса — большое количество проводов и контактов разъемов в соединительном кабеле (по крайней мере по одному на каждый бит). Отсюда громоздкость и дороговизна кабелей и интерфейсных цепей устройств, с которой мирятся ради вожделенной скорости. У последовательного интерфейса приемопередающие узлы функционально сложнее, зато кабели и разъемы гораздо проще и дешевле. Понятно, что на большие расстояния тянуть многопроводные кабели параллельных интерфейсов неразумно (и невозможно), здесь гораздо уместнее последовательные интерфейсы.

Теперь подробнее разберемся со скоростью передачи данных. Очевидно, что она равна числу бит, передаваемых за квант времени, деленному на продолжительность кванта. Для простоты можно оперировать *тактовой частотой интерфейса* — величиной, обратной длительности кванта. Это понятие естественно для синхронных интерфейсов, у которых имеется сигнал синхронизации (clock), определяющий возможные моменты возникновения всех событий (смены состояния). Для асинхронных интерфейсов можно воспользоваться эквивалентной тактовой частотой — величиной, обратной минимальной продолжительности одного состояния интерфейса. Теперь можно сказать, что максимальная (пиковая) скорость передачи данных равна произведению тактовой частоты на разрядность интерфейса. У последовательного интерфейса разрядность 1 бит, у параллельного она соответ-

ствует числу параллельных сигнальных цепей передачи битов данных. Остаются вопросы о достижимой тактовой частоте и разрядности. И для последовательного, и для параллельного интерфейсов максимальная тактовая частота определяется достижимым (при разумной цене и затратах энергии) быстродействием приемопередающих цепей устройств и частотными свойствами кабелей. Здесь уже очевидны выгоды последовательного интерфейса: для него, в отличие от параллельного интерфейса, затраты на построение высокоскоростных элементов не приходится умножать на разрядность.

В *параллельном* интерфейсе существует явление *перекоса* (skew), существенно влияющее на достижимый предел тактовой частоты. Суть его в том, что сигналы, одновременно выставленные на одной стороне интерфейсного кабеля, доходят до другого конца не одновременно из-за разброса характеристик цепей. На время прохождения влияет длина проводов, свойства изоляции, соединительных элементов и т. п. Очевидно, что перекос (разница во времени прибытия) сигналов разных битов должен быть существенно меньше кванта времени, иначе биты будут искажаться (путаться с одноименными битами предшествующих и последующих посылок). Вполне понятно, что перекос ограничивает и допустимую длину интерфейсных кабелей: при одной и той же относительной погрешности скорости распространения сигналов на большей длине набегает и больший перекос. Перекос сдерживает и увеличение разрядности интерфейса: чем больше используется параллельных цепей, тем труднее добиться их идентичности. Из-за этого даже приходится «широкий» (многоразрядный) интерфейс разбивать на несколько «узких» групп, для каждой из которых используются свои управляющие сигналы. В 90-х годах в схемотехнике приемопередающих узлов стали осваиваться частоты в сотни мегагерц и выше, то есть длительность кванта стала измеряться единицами наносекунд. Достичь соизмеримо малого перекоса можно лишь в пределах жестких компактных конструкций (печатная плата), а для связи отдельных устройств кабелями длиной в десятки сантиметров пришлось остановиться на частотах, не превышающих десятков мегагерц. Для того чтобы ориентироваться в числах, отметим, что за 1 нс сигнал пробегает по электрическому проводнику порядка 20–25 см. Наносекунда — это период сигнала с частотой 1 ГГц.

Для повышения пропускной способности параллельных интерфейсов с середины 90-х годов стали применять *двойную синхронизацию* DDR (Dual Data Rate). Ее идея заключается в выравнивании частот переключения информационных сигнальных линий и линий стробирования (синхронизации). В «классическом» варианте данные информационных линий воспринимаются только по одному перепаду (фронту или спаду) синхросигнала, что удваивает частоту переключения линии синхросигнала относительно линий данных. При двойной синхронизации данные воспринимаются и по фронту, и по спаду, так что частота смены состояний всех линий выравнивается, что при одних и тех же физических параметрах кабеля и интерфейсных схем позволяет удвоить пропускную способность. Волна этих модернизаций началась с интерфейса ATA (режимы UltraDMA) и прокатилась уже и по SCSI (Ultra160 и выше), и по памяти (DDR SDRAM). Кроме того, на высоких

частотах применяется *синхронизация от источника данных* (Source Synchronous transfer): сигнал синхронизации, по которому определяются моменты переключения или действительности (валидности) данных, вырабатывается самим источником данных. Это позволяет точнее совмещать по времени данные и синхронизующие импульсы, поскольку они распространяются по интерфейсу параллельно в одном направлении. Альтернатива — *синхронизация от общего источника* (common clock) — не выдерживает высоких частот переключения, поскольку здесь в разных (пространственных) точках временные соотношения между сигналами данных и сигналами синхронизации будут различными.

Повышение частоты переключений интерфейсных сигналов, как правило, сопровождается *понижением уровней* сигналов, формируемых интерфейсными схемами. Эта тенденция объясняется энергетическими соображениями: повышение частоты означает уменьшение времени, отводимого на переключения сигналов. Чем выше амплитуда сигнала, тем выше должна быть скорость нарастания сигнала и, следовательно, выходной ток передатчика. Повышение выходного тока (импульсного!) нежелательно по разным причинам: большие перекрестные помехи в параллельном интерфейсе, необходимость применения мощных выходных формирователей, повышенное тепловыделение. Тенденцию снижения напряжения можно проследить на примере порта AGP (3,3/1,5/0,8 В), шин PCI/PCI-X (5/3,3/1,5 В), SCSI, шин памяти и процессоров.

В *последовательном* интерфейсе явления перекоса отсутствуют, так что повышать тактовую частоту можно вплоть до предела возможностей приемопередающих цепей. Конечно, есть ограничения и по частотным свойствам кабеля, но изготовить хороший кабель для одной сигнальной цепи гораздо проще, чем для группы цепей. А когда электрический кабель уже «не тянет» требуемые частоту и дальность, можно перейти на оптический, у которого есть в этом плане огромные, еще не освоенные «запасы прочности». Устраивать же параллельный оптический интерфейс — слишком дорогое удовольствие.[1]

Вышеприведенные соображения объясняют современную тенденцию перехода на последовательный способ передачи данных.

Взаимодействие программ с периферийными устройствами

Периферийные устройства могут подключаться к интерфейсам системного уровня (ISA, PCI, PCI-X, PCI-Express, AGP, LPC) или к периферийным интерфейсам (порты COM, LPT, Game; шины USB, FireWire, SCSI). Абстрагируясь от конкретной реализации подключения на системном уровне, можно говорить о логической

[1] В 10-гигабитной версии технологии Ethernet есть параллельно-последовательный вариант.

[2] Другие платформы могут отличаться по набору свойств, например не иметь выделенного пространства ввода/вывода.

системной шине PC-совместимого компьютера[2] — интерфейсе со следующими базовыми свойствами:

- интерфейс обеспечивает транзакции обращения к пространствам памяти и ввода/вывода;
- в транзакциях фигурируют физические адреса пространств памяти и ввода/вывода;
- адресные пространства памяти и ввода/вывода являются «плоскими»: адрес выражается одним числом в диапазоне, определенном принятой разрядностью адресации. Любой адрес может принадлежать регистру (ячейке памяти) только одного устройства (или системной памяти, включающей ОЗУ и энергонезависимую память);
- транзакции могут инициироваться как центральным процессором (процессорами), так и активными устройствами (мастерами шины);
- все адресуемые элементы безусловно доступны центральному процессору; на адресуемость элементов со стороны мастеров шин могут накладываться специфические ограничения[1];
- устройства, подключенные к системной шине, могут посылать процессору (процессорам) запросы аппаратных прерываний.

Взаимодействие программ с *устройствами, подключенными к системной шине*, возможно следующими способами:

- через регистры устройств, отображенные на пространство памяти или пространство ввода/вывода;
- через области адресов памяти, принадлежащей устройству (физически расположенной на контроллере или адаптере устройства);
- через регистры конфигурационного пространства PCI (для устройств, подключенных к PCI, PCI-X, PCI-Express, AGP);
- через области системного ОЗУ, доступные активным устройствам-мастерам шины (обмен с использованием DMA);
- через аппаратные прерывания, сигнализируемые устройствами по доступным им линиям `IRQx` (ISA) или `INTx#` (PCI), а также по сообщениям MSI (PCI).

Обращения к регистрам конфигурационного пространства PCI (также «плоского») не относятся к базовым свойствам системной шины, поскольку программно они реализуются операциями обращения к пространству ввода/вывода и (или) памяти.

С устройствами, *подключенными к интерфейсам периферийного уровня*, взаимодействие возможно только через их контроллеры (адаптеры), подключенные к системной шине. На системной шине «видны» и доступны только эти адаптеры и контроллеры. Способы взаимодействия с устройствами определяются интерфейсом контроллера. Особенности взаимодействия с устройствами шин USB и FireWire рассмотрены далее, интерфейсы хост-контроллеров этих шин рассмотрены в главах 15 и 25.

---

[1] Например, мастеру шины ISA доступна системная память лишь в пределах первых 16 Мбайт; мастерам шин PCI безусловно доступна системная память, но могут быть недоступны другие устройства PCI, от которых их отделяет главный мост.

Программное обеспечение компьютера состоит из ряда компонентов: прикладного ПО (исполняемые модули — .EXE-файлы), драйверов устройств, системных драйверов, динамически компонуемых модулей, BIOS. Эти компоненты имеют различные возможности взаимодействия с устройствами, состав используемых компонентов зависит от операционной системы и уровня разделяемости данного устройства. Между прикладным ПО и периферийными устройствами возможны следующие варианты отношений:

- одиночное монопольное подключение: ПУ подключено к системной шине (возможно, через промежуточный периферийный интерфейс). С этим устройством в любой момент времени может взаимодействовать лишь одно приложение. Это самый простой вариант в плане организации взаимодействия ПО и устройства;
- групповое монопольное подключение: группа конечных ПУ подключена через промежуточный интерфейс к одному контроллеру, подключенному к системной шине. С данными устройствами может в любой момент времени взаимодействовать только одно приложение. По сравнению с предыдущим вариантом, здесь появляется небольшое усложнение, связанное с выбором конкретного конечного устройства для текущей операции обмена. Промежуточный интерфейс должен обеспечивать адресацию, управляемую приложением;
- одиночное разделяемое подключение: единственное конечное ПУ, подключенное к системной шине, может использоваться несколькими приложениями и/или процессами. Каждое из приложений (процессов) взаимодействует с устройством так, как будто оно — единственный «клиент» данного устройства. В структуре ПО, обеспечивающего взаимодействие с данным устройством, должны присутствовать средства *виртуализации* данного *устройства*. Эти средства и создают приложениям иллюзию монопольного взаимодействия с устройством. Пример — дисплей, подключенный к графическому адаптеру, с поддержкой оконного интерфейса. Здесь каждое приложение в отведенном ему логическом окне выполняет вывод изображения, не заботясь о текущем положении окна. Другой пример — клавиатура, обеспечивающая ввод символов в активное приложение;
- множественное разделяемое подключение: множество конечных ПУ подключено через периферийный интерфейс к общему контроллеру, связанному с системной шиной. Возможно одновременное взаимодействие нескольких приложений (процессов) с различными конечными ПУ. Сложность заключается в разделяемости: все взаимодействия осуществляются через общий контроллер периферийного интерфейса. Дополнительно возможно и коллективное использование конечных ПУ, для чего требуется виртуализация этих устройств. Примеры подключения: шины SCSI, USB, FireWire.

Вполне очевидно, что с точки зрения логических связей прикладное ПО может взаимодействовать с устройством непосредственно лишь в монопольном варианте использования. Все остальные варианты требуют выделения специальных модулей — *драйверов*, решающих задачи организации разделяемого использования контроллера и/или виртуализации устройства. Однако и для монопольного ва-

рианта отделение драйвера от прикладного ПО полезно с точки зрения модульности: при должной организации интерфейса между драйвером и прикладным ПО переход на использование новой модели устройства и/или на иной интерфейс его подключения потребует только смены драйвера, но не переработки прикладного ПО.

Взаимодействие программ, выполняемых центральным процессором (хост-программ), с периферийными устройствами возможно тремя основными способами:

- программно-управляемый обмен;
- прямой доступ к памяти;
- прерывания.

Программно-управляемый обмен — PIO (Programmed Input-Output). В исполняемой программе (или драйверах, которыми она пользуется) присутствуют инструкции ввода/вывода для портов устройства или инструкции обращений к областям памяти, находящейся в устройстве. Реальное физическое взаимодействие с устройством (и вызываемые этим изменения состояния устройств) происходит в момент выполнения этих инструкций. Такая жесткая синхронизация программы и устройства из рассматриваемых шин имеется в PCI/PCI-X, когда устройство является ведомым (target). Из интерфейсов, не рассмотренных в данной книге, такой способ обмена используется в LPT и COM портах (в режимах без FIFO и DMA), а также в шине ATA (при доступе к регистрам устройства и обмене данными в режиме PIO). Данный способ взаимодействия позволяет предельно упростить интерфейсную часть периферийного устройства. Расплатой за это упрощение является нагрузка на центральный процессор. Отметим, что применительно к шине PCI (и всем ее «родственникам») программно-управляемый обмен не позволяет приблизиться к декларированной высокой пропускной способности шины. Причиной тому является неспособность процессора породить длинные пакетные транзакции на шине PCI (см. главу 2), поэтому следует избегать данного способа взаимодействия при интенсивном обмене данными;

Прямой доступ к памяти — DMA (Direct Memory Access). Обмен между системной памятью (ОЗУ) и устройством выполняется без непосредственного участия процессора. Обмен осуществляет контроллер прямого доступа, для устройств шины PCI (и всех ее «родственников») контроллер является частью устройства — мастера шины (Bus Master). Штатного централизованного контроллера DMA, как это было в архитектуре ISA, для шины PCI нет[1]. В зависимости от того, кто является инициатором обмена, различают два варианта прямого доступа:

- DMA по инициативе хоста (Host Initiated DMA). Задание на пересылку каждого блока формирует программа, исполняемая на ЦП; она же сообщает контроллеру DMA параметры сеанса (начальный адрес, длину блока и направление передачи) записью в его регистры. Физические операции обмена синхронизируются с устройством — оно своими внутренними сигналами запускает обмен

[1] Эмуляция централизованного контроллера, введенная на время миграции с ISA на PCI (см. главу 2), в расчет не берется.

и, если требуется, управляет потоком (вводит сигнал готовности). Этот вариант требует довольно простых аппаратных средств устройства, расплата за упрощение — необходимость привлечения ЦП к организации каждого сеанса (обычно по прерываниям). Это не очень эффективно при передаче больших объемов данных, которые могут располагаться в разных не смежных страницах физической памяти (см. далее);

- DMA по инициативе устройства (Target Initiated DMA). Здесь хост-программа формирует в памяти программу ввода/вывода для устройства, обычно представляющую собой связанный список дескрипторов передач, и указывает устройству на начало списка. Контроллер устройства считывает эти дескрипторы из ОЗУ и по ним организует сеансы передачи данных между устройством и буферами в ОЗУ, описанными дескрипторами передач. Формирование программы может быть статическим или динамическим. В первом случае хост-программа передает устройству указатель на готовый список дескрипторов и не имеет права его модифицировать до тех пор, пока устройство не отработает список до конца. Так, например, работает PCI-контроллер шины ATA. При динамическом формировании хост может добавлять новые дескрипторы (в конец списка), постоянно «подбрасывая» контроллеру новые задания. Так работают контроллеры шины USB и FireWire, PCI-контроллеры локальных сетей и ряд других. Работа устройства по программе требует усложнения его контроллера, но эти затраты окупаются повышением производительности и эффективности ввода/вывода. При этом стараются минимизировать число прерываний центрального процессора, инициируемых устройством.

Прерывания (*Interrupts*) — сигнализация от устройства (его контроллера) центральному процессору (процессорам в мультипроцессорных системах) о некоторых событиях, требующих программных действий хоста. Эти события асинхронны по отношению к программному коду, исполняемому процессором. Прерывания требуют приостановки выполнения текущего потока инструкций (с сохранением состояния) и запуска исполнения процедуры-обработчика прерывания *ISR* (Interrupt Service Routine). Эта процедура первым делом должна идентифицировать источник прерывания (а их может быть и несколько), затем выполнить действия, связанные с реакцией на событие. Если события должны вызывать некоторые действия прикладной программы, то обработчику прерывания следует только подать сигнал (через ОС), запускающий (или пробуждающий) поток инструкций, выполняющий эти действия. Собственно процедура ISR должна быть оптимизирована по затраченному времени. Обслуживание прерываний, особенно в защищенном режиме, в PC-совместимых компьютерах на процессорах x86 связано со значительными накладными расходами. По этой причине их число стараются сократить. Значительные хлопоты доставляет идентификация источника прерывания — в архитектуре PC-совместимых компьютеров для этого используются традиционные, но неэффективные механизмы. В ряде случаев прерывания от устройств заменяют *поллингом* — программно-управляемым опросом состояния устройств. При этом состояния множества устройств опрашивают по прерыванию от таймера.

В компьютерных системах с «интеллектуальной» системой ввода/вывода (I_2O — Intelligent Input-Output) кроме центрального процессора имеется *процессор ввода/вывода* (IOP — Input-Output Processor). Этот процессор обычно имеет сокращенную систему команд, ориентированную на задачи управления вводом/выводом. В круг этих задач входит пересылка блоков данных, подсчет четности (для дисковых массивов RAID 3 и 5), преобразование данных между форматами *Big Endian* (популярный в телекоммуникациях) и *Little Endian* (принятый в процессорах Intel). Процессор ввода/вывода может работать как в общем адресном пространстве, так и иметь свое обособленное адресное пространство для управляемой подсистемы ввода/вывода. Взаимодействие процессора ввода/вывода со своими устройствами ведется теми же тремя основными способами, что были описаны ранее.

В рядовых компьютерах обычно ограничиваются *прямым управлением шиной* (bus mastering), которое позволяет контроллерам ПУ (или их интерфейсам) самим обращаться к системным ресурсам, выполняя необходимые обмены данными и управляющей информацией. Для этого контроллер ПУ должен взять на себя (на время) роль инициатора транзакций на интерфейсе, связывающем его с центром (главным образом, с памятью). Поскольку традиционно этот интерфейс является шинным, такой активный контроллер называют *мастером шины* (bus master), даже если он подключается к выделенному двухточечному интерфейсу (порту AGP). Чаще всего прямое управление шиной используется для прямого доступа к оперативной памяти. Прямое управление шиной может использоваться и для сигнализации прерываний (MSI на шине PCI, см. главу 3). В новых версиях шины PCI-X и в PCI Express появилась возможность *равноранговое взаимодействия устройств* (без участия процессора) — обмена сообщениями. При этом в адресации сообщений не фигурируют адреса пространства памяти или ввода/вывода — обращения адресуются по *идентификатору устройства* (DIM — Device Identified Messages).

Особенности взаимодействия в PC-совместимых компьютерах

Архитектурный облик PC-совместимого компьютера определяется свойствами используемых в них процессоров семейства x86. Современные процессоры x86, работающие в защищенном режиме, имеют довольно сложные механизмы виртуализации памяти, ввода/вывода и прерываний, из-за которых приходится различать физические и логические пространства (адреса памяти и ввода/вывода) и события (операции ввода/вывода, прерывания).

Физический адрес ячейки памяти или порта ввода/вывода — это адрес, формируемый на системной шине для обращения к данной ячейке. *Логический адрес* — это тот адрес, который формируется исполняемой программой (по замыслу программиста) для доступа к требуемой ячейке. Логический адрес в процессорах x86 со-

стоит из двух компонентов: селектора сегмента и смещения внутри сегмента; из этих компонентов формируется *линейный адрес* — целое беззнаковое число. В большинстве современных ОС используется *плоская модель памяти*, в которой все доступные сегменты отображены на одно и то же адресное пространство. При этом программа не оперирует селекторами; программист адресует структуры данных в памяти по *линейным адресам* (для современных процессоров и приложений — 32-разрядным). *Физический адрес* формируется из логического с помощью блока страничной переадресации; трансляция адресов выполняется на страничном базисе, популярный размер страницы — 4 Кбайт. Страничная переадресация выполняется для реализации виртуальной памяти с подкачкой страниц. Переадресация выполняется на основе таблиц, формируемых в памяти операционной системой.

Физическая операция ввода/вывода или обращения к памяти — это процесс (шинный цикл), во время которого генерируются электрические сигналы, обеспечивающие доступ к данной ячейке (порту). *Логическая операция* — это исполнение программной инструкции (команды) обращения к интересующей ячейке. Логическая операция не всегда порождает ожидаемую физическую операцию: при определенных условиях она может блокироваться средствами защиты процессора, вызывая даже принудительное завершение программы, или же эмулироваться, создавая иллюзию физического исполнения.

Взаимодействие через пространство памяти

В реальном режиме (при отключенной страничной переадресации) физический адрес, фигурирующий на системной шине, совпадает с линейным адресом, формируемым прикладной программой. Тут все просто, правда, в стандартном (а не большом) реальном режиме доступен только первый мегабайт адресного пространства (то есть из устройств доступны только отображенные на область UMA).

В защищенном режиме, в принципе, доступно все физическое адресное пространство, но появляются проблемы, связанные с отображением линейных адресов на физические. Страничной переадресацией (поддержкой таблиц) ведает ОС, и у разных программных компонентов (приложений, драйверов, динамических модулей) имеются различающиеся возможности взаимодействия с системой управления памятью. Напомним, что у каждой задачи может быть своя карта адресов, в которой не обязательно будут присутствовать физические адреса всех устройств.

Для *обращения к регистрам устройства*, расположенным в пространстве памяти (или к области памяти устройства), программа должна узнать физический адрес данной области. Далее она должна запросить у ОС линейный адрес, на который отображается этот физический адрес, и обращаться к нему по этому линейному адресу. Иного пути добраться до физического устройства у программы нет, и если ОС откажет в данном запросе, устройство окажется для этой программы недоступным. Для обращения к устройствам через пространство памяти у процессоров x86 предусмотрено большое число разнообразных инструкций, выполняющих как про-

сто пересылку, так и работающих с операндами в памяти (то есть в устройстве). Инструкции, модифицирующие ячейки памяти, порождают на системной шине блокированные транзакции «чтение-модификация-запись». Этот тип транзакций не приветствуется с точки зрения эффективности использования времени шины, так что предпочтительно избегать таких инструкций при взаимодействии с устройствами. Заметим, что инструкции процессора обычно не порождают эффективных пакетных транзакций на шине PCI, они вызывают лишь одиночные транзакции[1]. Некоторые программные ухищрения, позволяющие повысить эффективность программно-управляемого обмена, описаны в главе 2.

При *организации прямого доступа к памяти*, как по стандартным каналам DMA, так и при использовании ведущих устройств шин ISA и PCI, возникает ряд проблем, связанных со страничным преобразованием адресов. Программе требуется организовать обмен данными между устройством и некоторым буфером данных в ОЗУ, с которым программа общается по линейным адресам, а устройство — по физическим. Отметим ряд существенных моментов:

- программа должна запросить у ОС физический адрес, которому соответствует линейный адрес предполагаемого буфера обмена. Именно этот физический адрес должен задаваться устройству, осуществляющему DMA (или централизованному контроллеру DMA), при инициализации сеанса обмена (при указании начального адреса, длины блока и запуске канала);
- физические страницы, к которым обращаются по DMA, должны быть зафиксированы: механизм замещения страниц не должен их затрагивать, по крайней мере, пока не завершится обмен по DMA;
- если буфер данных не умещается в одной логической странице, возникает проблема пересечения границ. Обычный контроллер DMA работает по последовательно изменяемым (инкрементируемым или декрементируемым) адресам. При пересечении границы логической страницы, возможно, потребуется скачок физического адреса, поскольку следующая логическая страница может иметь физическое отображение в произвольном (относительно предыдущей страницы) месте ОЗУ. Чаще всего ОС оперирует страницами по 4 Кбайт, при этом пересылка больших блоков данных ведется «короткими перебежками», между которыми должна выполняться повторная инициализация контроллера DMA.

Проблема пересечения границ решается усложнением контроллеров DMA — применением «разбросанной записи» в память (scatter write) и «собирающего чтения» памяти (gather read). В этом случае контроллеру DMA задается список описателей блоков (начальный адрес и длина), каждый из которых не пересекает границ логической страницы. Отработав очередной блок памяти, контроллер переходит к следующему, и так до конца списка. Такие возможности имеет, например, стан-

[1] Передача 32-разрядного слова по невыровненному адресу породит пакетный цикл из двух передач, но эффективным (с точки зрения пропускной способности) его не назовешь.

дартный контроллер PCI IDE. Для передачи логически непрерывного буфера данных описатели его блоков могут быть сокращены. Так, например, можно задать полный физический адрес начала буфера, его длину и только список базовых адресов занимаемых им страниц. На каждой странице, кроме начальной, данные будут начинаться с нулевого адреса; на каждой странице, кроме последней, данные будут доходить до последнего адреса. Вместо длины буфера можно задавать и физический адрес его конца. Такие варианты описаний используются, например, в хост-контроллерах шин USB и FireWire.

Проблема пересечения границ может решаться и иначе, без усложнения контроллера DMA. Для этого в памяти резервируется буфер значительного размера, отображенный на непрерывную область физической памяти, и обмен данными физическое устройство выполняет только с этим буфером. Однако рядовое приложение не может создать такой буфер, он может быть организован лишь драйвером устройства. Приложения могут лишь получать указатели на этот буфер и обмениваться с ним данными. Таким образом, по пути от приложения к устройству появляется дополнительная «перевалочная база» (буфер драйвера) и, соответственно, дополнительная пересылка данных, что приводит к дополнительным затратам времени.

Взаимодействие через пространство ввода/вывода

Для обращения программы к пространству ввода/вывода предназначены всего четыре инструкции процессора: `IN` (ввод из порта в регистр процессора), `OUT` (вывод в порт из регистра процессора), `INS` (ввод из порта в элемент строки памяти) и `OUTS` (вывод элемента из строки памяти в порт). Последние две инструкции, появившиеся с процессором 80286, могут использоваться с префиксом повтора `REP`, что обеспечивает быструю пересылку блоков данных между портом и памятью. Обмен данными с портами, при котором применяют строковые инструкции ввода/вывода, получил название *PIO* (Programmed Input/Output — программированный ввод/вывод).

Разрядность слова, передаваемого за одну инструкцию ввода/вывода, может составлять 8, 16 или 32 бита. В зависимости от выровненности адреса по границе слова и разрядности данных используемой шины это слово может передаваться за один или несколько циклов шины с указанием соответствующего нарастающего адреса в каждом цикле обращения к памяти. Инструкции ввода/вывода порождают шинные циклы обмена, в которых вырабатываются сигналы чтения порта/записи в порт. Во избежание недоразумений и для экономии шинных циклов рекомендуется выравнивать адреса 16-битных портов по границе слова, а 32-битных — по границе двойного слова. Обращения по выровненным адресам выполняется за один цикл системной шины. Обращение по невыровненным адресам выполняется за несколько циклов, причем однозначная последовательность адресов обращений (которая зависит от модели процессора) не гарантируется. Так, например, одна инструкция вывода слова по нечетному адресу приведет к генерации двух смежных шинных циклов записи. При программировании обращений следует учиты-

вать специфику устройств ввода/вывода. Если, например, устройство допускает только 16-разрядные обращения, то старший байт его регистров будет доступен лишь при вводе-выводе слова по четному адресу.

В реальном режиме процессора программе доступно все пространство адресов ввода/вывода. В защищенном режиме инструкции ввода/вывода являются привилегированными: возможность их исполнения зависит от текущего уровня привилегий. В защищенном режиме 32-разрядных процессоров (частным случаем которого является и виртуальный режим V86) имеется возможность программного ограничения доступного пространства ввода/вывода, определяя его максимальный размер (начиная с нулевого адреса и в пределах 64 Кбайт), а внутри разрешенной области доступ может быть разрешен или запрещен для каждого конкретного адреса. Размер области и *карта разрешенных портов* (IO Permission Bitmap) задается операционной системой в дескрипторе сегмента состояния задачи (TSS). Карта разрешений влияет на исполнение инструкций ввода/вывода в зависимости от соотношения текущего уровня привилегий и требуемого уровня привилегий ввода/вывода. При недостаточных привилегиях обращение по неразрешенному адресу вызывает исключение процессора, а поведение его обработчика определяется операционной системой. Возможно снятие задачи-нарушителя (знаменитое сообщение «приложение... выполнило недопустимую операцию и будет закрыто»). Возможен и другой вариант, когда по обращению к порту монитор операционной системы выполняет некоторые действия, создавая для программы иллюзию реальной операции ввода/вывода. Таким образом виртуальная машина по операциям ввода/вывода может общаться с виртуальными устройствами. Программа, выполняемая на нулевом уровне привилегий, безусловно может обращаться ко всем портам непосредственно.

Наиболее корректный (с точки зрения организации ОС) способ общения приложения с портами устройства требует помещения инструкций ввода/вывода в драйвер устройства, работающий на уровне привилегий ОС (на нулевом уровне). Обращение к портам непосредственно из приложения возможно, если в карте разрешения портов бит для данного порта сброшен. Если бит установлен, то обращение к порту вызывает *исключение защиты*, которое обрабатывает VMM (диспетчер виртуальной машины). В этом случае VMM вызывает процедуру, назначенную для данного порта операционной системой. Это может быть либо специальная процедура виртуального драйвера, установленного для данного порта, либо процедура по умолчанию. В первом случае ввод/вывод для данного порта приложению доступен только через виртуальный драйвер, вызов которого каждый раз будет приводить к издержкам переключения задач и смены уровня привилегий (от приложения на третьем уровне к драйверу нулевого уровня). Однако с точки зрения идеологии многозадачности и защиты это естественное решение, обеспечивающее полную виртуализацию ввода/вывода. Процедура по умолчанию (в Windows 9*x*) открывает порт для данного приложения (сбрасывает бит в карте разрешений ввода/вывода) и выполняет собственно инструкцию ввода/вывода (возвращая при-

ложению результат ввода). Таким образом, приложению Windows 9*x* станут доступными любые порты, для которых не установлен виртуальный драйвер. Правда, первое обращение к каждому порту будет происходить медленно (через исключение), но последующие будут выполняться быстро. Если для взаимодействия с устройством задержка первого обращения критична, то при инициализации приложения можно выполнить «безобидные» обращения по адресам всех требуемых портов, что откроет их для дальнейшей непосредственной работы (без издержек).

Заметим, что ОС Windows 9*x* не особо заботится о виртуализации и защите ввода/вывода; здесь, например, из DOS-окна можно обращаться к любым портам, даже к портам устройств, занятых операционной системой. В ОС Windows NT/2000/XP защита портов строже и сложнее.

ГЛАВА 1

Шины PCI и PCI-X

Шины PCI и PCI-X являются основными шинами расширения ввода/вывода в современных компьютерах; для подключения видеоадаптеров их дополняет порт AGP. Шины расширения ввода/вывода (Expansion Bus) являются средствами подключения системного уровня: они позволяют адаптерам и контроллерам периферийных устройств непосредственно использовать системные ресурсы компьютера — пространство адресов памяти и ввода/вывода, прерывания, прямой доступ к памяти. Устройства, подключенные к шинам расширения, могут и сами управлять этими шинами, получая доступ к остальным ресурсам компьютера. Шины расширения механически реализуются в виде слотов (щелевых разъемов) или штырьковых разъемов; для них характерна малая длина проводников, то есть они сугубо локальны, что позволяет достигать высоких скоростей работы. Эти шины могут и не выводиться на разъемы, но использоваться для подключения устройств в интегрированных системных платах.

Поначалу шина PCI вводилась как пристройка (mezzanine bus) к системам с шиной ISA. Она разрабатывалась в расчете на процессоры Pentium, но хорошо сочеталась и с процессорами i486. Позже PCI на некоторое время стала центральной шиной: она соединялась с шиной процессора высокопроизводительным мостом («северным» мостом), входящим в состав чипсета системной платы. Остальные шины расширения ввода/вывода (ISA/EISA или MCA), а также локальная ISA-подобная шина X-BUS и интерфейс LPC, к которым подключаются микросхемы системной платы (ROM BIOS, контроллеры прерываний, клавиатуры, DMA, портов COM и LPT, НГМД и прочие «мелочи»), подключались к шине PCI через «южный» мост. В современных системных платах с «хабовой» архитектурой шину PCI отодвинули на периферию, не ущемляя ее в мощности канала связи с процессором и памятью, но и не нагружая транзитным трафиком устройств других шин.

Шина PCI является *синхронной* — фиксация всех сигналов выполняется по положительному перепаду (фронту) сигнала `CLK`. Номинальной частотой синхронизации считается частота 33,3 МГц, при необходимости она может быть понижена. Начиная с версии PCI 2.1 допускается повышение частоты до 66,6 МГц при «согласии» всех устройств на шине. В PCI-X частота может достигать 133 МГц.

В PCI используется *параллельная мультиплексированная шина* адреса/данных (AD) с типовой разрядностью 32 бит. Спецификация определяет возможность расширения разрядности до 64 бит; в PCI-X версии 2.0 определен также 16-битный вариант шины. При частоте шины 33 МГц теоретическая пропускная способность достигает 132 Мбайт/с для 32-битной шины и 264 Мбайт/с для 64-битной; при частоте синхронизации 66 МГц — 264 Мбайт/с и 528 Мбайт/с соответственно. Однако эти пиковые значения достигаются лишь во время передачи пакета: из-за протокольных накладных расходов реальная средняя пропускная способность шины оказывается ниже.

Сравнительные характеристики шин PCI и PCI-X и других шин расширения PC-совместимых компьютеров приведены в табл. 1.1. Шина ISA из настольных компьютеров уходит, но она сохраняет свои позиции в промышленных и встраиваемых компьютерах, как в традиционном, слотовом, так и в «бутербродном» варианте PC/104. В блокнотных компьютерах широко применяются слоты PCMCIA с шинами PC Card и Card Bus. Шина LPC является современным дешевым средством для подключения нересурсоемких устройств к системной плате.

Таблица 1.1. Сравнительные характеристики шин расширения

Шина	Пиковая пропускная способность, Мбайт/с	Каналы DMA	Bus-Master	ACFG[1]	Разрядность данных	Разрядность адреса	Частота, МГц
ISA-8	4	3	–	–	8	20	8
ISA-16	8	7	+	–	16	24	8
LPC	6,7	7	+	–	8/16/32	32	33
EISA	33,3	7	+	+	32	32	8,33
MCA-16	16	–	+	+	16	24	10
MCA-32	20	–	+	+	32	32	10
VLB	132	–	(+)	–	32/64	32	33–50 (66)
PCI	133–533	–	+	+	32/64	32/64	33/66
PCI-X	533–4256	–	+	+	16/32/64	32/64	66–133
PCI Express	496–15872	–	+	+	1/2/4/8/12/16/32	32/64	2,5 ГГц
AGP 1x/2x/4x/8x	266/533/1066/2132	–	+	+	32	32/64	66
PCMCIA	10/20	+	–	+	8/16	26	10
Card Bus	132	–	+	+	32	32	33

[1] Поддержка автоматического конфигурирования. Для ISA PnP является поздней надстройкой, реализуемой адаптерами и ПО.

Организация шин PCI и PCI-X

Шина PCI позволяет *объединять равноранговые устройства*. Любое устройство шины может выступать как в роли *инициатора транзакций* (задатчика), так и в роли *целевого устройства*. Целевое устройство отвечает на транзакции, адресованные к его ресурсам (областям памяти и портам ввода/вывода). Ядро компьютера (центральный процессор и память) для шины PCI также представляется устройством — *главным мостом* (host bridge). В транзакциях, обращенных к устройствам PCI, инициированных центральным процессором, главный мост является задатчиком. В транзакциях от устройств PCI, обращающихся к ядру (к системной памяти), главный мост является целевым устройством. Право на управление шиной в любой момент времени дается лишь одному устройству данной шины; арбитраж запросов на управление шиной осуществляется централизованным способом. Арбитр, как правило, является частью моста.

Наличие активных устройств (помимо ЦП) позволяет выполнять в компьютере параллельно несколько операций обмена: одновременно с обращениями процессора могут выполняться транзакции от мастеров шины PCI. Эта параллельность — *PCI Concurrency* — возможна лишь для обменов по непересекающимся путям. Одновременный доступ нескольких инициаторов к одному ресурсу (как правило, к системной памяти) требует довольно сложной организации контроллера этого ресурса, но ради повышения суммарной эффективности системы на эти усложнения приходится идти. В системе с несколькими шинами PCI возможна параллельная работа устройств-мастеров на разных шинах — *PCI Peer Concurrency*. Однако если они обращаются к одному ресурсу (системной памяти), то какие-то фазы этих обменов все-таки должны будут выполняться последовательно.

Каждая *физическая шина PCI* позволяет объединять лишь небольшое число устройств: типовое ограничение по электрическим спецификациям — не более шести устройств на шине. Для увеличения числа подключаемых устройств применяют *мосты PCI* (PCI-to-PCI Bridge) — устройства PCI с парой интерфейсов, которыми шины объединяются в древовидную структуру. В корне этой структуры находится *хост* — «хозяин шины», в обязанности которого входит конфигурирование всех устройств, включая и мосты. В роли хоста, как правило, выступает центральный процессор с главным мостом. Мосты позволяют объединять шины PCI и PCI-X с разными характеристиками, а также подключать к PCI/PCI-X иные шины: (E)ISA, MCA, шины блокнотных ПК, PCI Express, Hyper Transport и другие.

Шина PCI/PCI-X имеет несколько *вариантов конструктивного оформления*, часть из которых при наличии специального контроллера допускают горячую замену устройств:

- шина объединения компонентов на печатной плате (системной плате или карте расширения);
- слотовые разъемы для установки карт расширения (в конструктивах PC и MCA);
- разъемы для малогабаритных карт расширения (Card Bus, Small PCI, Mini PCI);
- модульные конструктивы для промышленных и инструментальных компьютеров (CompactPCI, PXI).

Важной частью шины PCI является *система автоматического конфигурирования*; конфигурирование выполняется каждый раз при включении питания и инициализации системы. Специальное конфигурационное ПО позволяет обнаружить и идентифицировать все установленные устройства, а также выяснить их потребности в ресурсах (областях памяти, адресах ввода/вывода, прерываниях). Спецификация PCI требует от устройств способности перемещать все занимаемые ресурсы (области в пространстве памяти и ввода/вывода) в пределах доступного адресного пространства. Это позволяет обеспечить бесконфликтное распределение ресурсов для множества устройств. Одно и то же функциональное устройство может быть сконфигурировано по-разному, отображая свои операционные регистры либо на пространство памяти, либо на пространство адресов ввода/вывода. Драйвер может определить текущую настройку, прочитав содержимое регистра базового адреса устройства. Драйвер также может определить номер запроса на прерывание, который используется устройством. Для поддержки конфигурирования устройств существует специальный набор функций PCI BIOS.

Взаимодействие устройств

С программной точки зрения устройство PCI может иметь следующие компоненты:

- конфигурационные регистры, используемые для идентификации и начального конфигурирования устройства при инициализации системы. Для всех устройств предусмотрен обязательный набор конфигурационных регистров, остальные регистры могут использоваться для текущего управления;
- операционные регистры (необязательные), отображенные на пространство памяти и/или ввода/вывода; эти регистры используются для текущего управления и взаимодействия с устройством;
- локальная память (необязательная), отображенная на выделенные области физических адресов системной памяти;
- источники запросов на прерывания;
- мастер шины, обеспечивающий прямой доступ к системной памяти (DMA) и взаимодействие с другими устройствами.

С устройством PCI, когда оно является целевым, можно взаимодействовать несколькими способами:

- командами *обращения к памяти и портам ввода/вывода*; эти команды адресуются к областям, выделенным устройству при конфигурировании;
- командами *обращения к конфигурационным регистрам*; эти команды адресуются по *идентификатору* — номеру *шины*, *устройства* и *функции* (компонента многофункционального устройства PCI);
- специальными *широковещательными сообщениями*, передаваемыми для всех устройств выбранной шины;
- командами *пересылки сообщений*; команды адресуются по идентификатору устройства (эта возможность появилась в PCI-X 2.0);

Для обращений к пространству памяти используется 32-битная или 64-битная адресация, причем разрядность адресации не зависит от разрядности шины. Таким образом, шина позволяет адресовать до 2^{32} (4 Гбайт) или 2^{64} (более 16 тыс. Пбайт) байт памяти. На шине PCI фигурирует физический адрес памяти. Для адресации портов ввода/вывода используется 32-битная адресация; в компьютерах на базе процессоров x86 из них используются только 16 младших бит. В системе адресации ввода/вывода имеется поддержка особенностей, связанных с адресацией портов в PC-совместимых компьютерах с шиной ISA. Для устройств PCI и PCI-X рекомендуется по возможности избегать использования портов ввода/вывода; операционные регистры устройств рекомендуется отображать на пространство памяти (Memory-Mapped I/O).

Конфигурационные регистры устройств PCI расположены в обособленном пространстве адресов (отдельном от пространства адресов памяти и ввода/вывода). Каждому устройству (точнее, каждой функции сложного устройства) выделяется 256-байтный блок конфигурационных регистров; в спецификации PCI-X 2.0 размер блока увеличен до 4096 байт. Частью этого блока является обязательный набор конфигурационных регистров, с помощью которых осуществляются идентификация устройств, их конфигурирование и управление их свойствами. В конфигурационных регистрах, в частности, указываются адреса, отведенные устройству (как целевому); через них разрешается работа в роли инициатора и целевого устройства, конфигурируются прерывания. Конфигурационные регистры обеспечивают возможность автоматической настройки всех устройств шины PCI. К этим регистрам система обращается на этапе конфигурирования — переучета обнаруженных устройств, выделения им неперекрывающихся ресурсов (областей памяти и пространства ввода/вывода) и назначения номеров аппаратных прерываний. При дальнейшей регулярной работе взаимодействие прикладного ПО с устройствами осуществляется преимущественно путем обращений по назначенным в процессе конфигурирования адресам памяти и ввода/вывода. Конфигурационные же регистры в регулярной работе используются для системных целей: настройки параметров, описывающих поведение устройства на шине, обработки ошибок, идентификации источника прерываний.

Обращения к регистрам и памяти устройств PCI выполняются командами шины PCI. Команды может подавать любой инициатор: как хост (главный мост) по командам центрального процессора, так и рядовое устройство PCI. Возможность распространения ряда команд зависит от взаимного расположения инициатора и целевого устройства на «ветвях» дерева шин PCI. Однако хост может безусловно подавать любую команду любому устройству PCI. Только хост всегда (и по начальному включению) имеет доступ к конфигурационным регистрам всех устройств (и мостов), поэтому он и должен заниматься конфигурированием. После конфигурирования любое устройство PCI может безусловно обратиться к системной памяти, то есть реализовать *прямой доступ к памяти* (DMA).

Устройства PCI могут вырабатывать *запросы аппаратных прерываний*:

- обычных маскируемых — для сигнализации событий в устройстве; эти прерывания могут сигнализироваться как традиционным способом — по специальным сигнальным линиям, так и передачей сообщений (MSI);

- немаскируемых — для сигнализации о серьезных ошибках;
- прерываний системного управления (SMI) — для сигнализации о событиях в системе управления энергопотреблением и некоторых системных целей (например, эмуляции работы стандартного контроллера клавиатуры с помощью устройств USB).

Наиболее эффективно возможности шины PCI используются при применении *активных устройств — мастеров шины* (*PCI Bus Master*). Только эти устройства могут обеспечить скорость передачи данных, приближающуюся к декларированной пиковой пропускной способности. Максимальная производительность обменов по шине PCI достигается только в пакетных транзакциях значительной длины. Транзакции по инициативе программы, исполняемой на ЦП, проводимые главным мостом, как правило, являются одиночными (или очень короткими пакетными). По этой причине программно-управляемый обмен данными с устройствами PCI по производительности значительно уступает обмену, выполняемому устройством-мастером. Таким образом, применение активных устройств дает двойной эффект: разгружает центральный процессор и обеспечивает лучшее использование пропускной способности шины.

Шины, устройства, функции и хост

Каждое устройство PCI при установке в конкретную систему получает *идентификатор*, однозначно определяющий его положение на «дереве» шин PCI данного компьютера. Идентификатор имеет иерархическую структуру и состоит из *номера шины* (bus), *номера устройства* (device) и *номера функции* (function). Идентификатор задает положение блока конфигурационных регистров заданной функции выбранного устройства в общем конфигурационном пространстве данной системы. Идентификаторы фигурируют при обращениях к регистрам конфигурационного пространства (см. главу 5), а также при обмене сообщениями между устройствами (DIM в PCI-X, см. главу 2).

Шина PCI представляет собой набор сигнальных линий (см. главу 6), непосредственно соединяющих интерфейсные выводы группы устройств (слотов, микросхем на плате). В системе может присутствовать несколько шин PCI, соединенных *мостами PCI* (см. главу 4). Мосты электрически отделяют интерфейсные сигналы одной шины от другой, соединяя шины логически; главный мост соединяет главную шину PCI с хостом (процессором и памятью). Каждая шина имеет свой *номер шины* (PCI bus number). Шины нумеруются последовательно, начиная от хоста; шина PCI, подключенная к главному мосту, имеет нулевой номер.

Устройством PCI называется микросхема или карта расширения, подключенная к одной из шин PCI и использующая для доступа к конфигурационным регистрам выделенную ей линию `IDSEL`, принадлежащую этой шине. Устройство может быть многофункциональным, то есть состоять из множества (от 1 до 8) так называемых *функций*. Каждой функции отводится конфигурационное пространство в 256 байт, в PCI-X оно расширено до 4096 байт. Многофункциональные устройства должны отзываться только на конфигурационные циклы с номерами функций, для кото-

рых имеется конфигурационное пространство. При этом функция с номером 0 должна присутствовать обязательно (по результатам обращения к ней определяется присутствие устройства), номера остальных функций назначаются разработчиком устройства произвольно (в диапазоне 1–7). Простые (однофункциональные) устройства, в зависимости от реализации, могут отзываться либо на любой номер функций, либо только на номер функции 0.

Нумерацией и конфигурированием всех устройств PCI занимается *хост* — «хозяин» шины PCI. Роль хоста, как правило, выполняет центральный процессор, связанный с шиной PCI главным мостом, от которого и начинается нумерация шин. Конфигурирование всех устройств шины возможно только со стороны хоста — в этом заключается его особая роль. Ни с одной из шин PCI ни один задатчик не имеет доступа к конфигурационным регистрам всех устройств PCI, без чего полное конфигурирование недоступно. Даже с нулевой шины PCI задатчику недоступны конфигурационные регистры главного моста, а без доступа к ним невозможно запрограммировать распределение адресов между хостом и устройствами PCI. С других шин PCI возможности доступа к конфигурационным регистрам еще скромнее (см. главу 4).

Конфигурирование выполняется для каждой *функции*; полный идентификатор функции состоит из трех частей: номера шины, номера устройства и номера функции. Короткая форма идентификатора вида PCI0:1:2 (например, в сообщениях ОС Unix) означает функцию 2 устройства 1, подключенного к главной (0) шине PCI. Диспетчер устройств (конфигурационное ПО) должен оперировать списком всех функций всех устройств, обнаруженных на всех шинах PCI данной системы (компьютера).

В шине PCI принята *географическая нумерация* — номер устройства определяется местом его подключения. *Номер устройства* (`device number` или `dev`) определяется той линией шины `AD`, к которой подключена его линия сигнала `IDSEL`. В соседних слотах PCI, как правило, задействуются соседние номера устройств; их нумерация определяется разработчиком системной платы (или пассивной кросс-платы в промышленных компьютерах). Часто для слотов используются убывающие номера устройств, начиная с 20 или 15. Группы соседних слотов могут подключаться к разным шинам; на каждой шине PCI нумерация устройств независимая (могут быть и устройства с совпадающими номерами `dev`, но разными номерами шин). Устройства PCI, интегрированные в системную плату, используют ту же систему нумерации. Их номера «запаяны намертво», в то время как номера устройств на картах расширения можно изменять, переставляя их в разные слоты.

Одна *карта PCI* может содержать только одно устройство шины, к которой она подключается, поскольку ей в слоте выделяется только одна линия `IDSEL`. Если на карте размещают несколько устройств (например, 4-портовая карта Ethernet), то на ней приходится устанавливать мост — устройство PCI, к которому и обращаются по линии `IDSEL`, выделенной данной карте. Этот мост организует на карте дополнительную шину PCI, к которой можно подключить множество устройств. Каждое из этих устройств получит свою линию `IDSEL`, но относящуюся уже к дополнительной шине PCI данной карты.

С точки зрения обращения к пространствам памяти и ввода/вывода географический адрес (номер шины и номер устройства) в пределах одной шины безразличен. Однако номер устройства определяет номер линии запроса прерывания, которой может пользоваться устройство. Подробнее об этом рассказывается в главе 3, здесь же отметим, что на одной шине устройства с номерами, отличающимися друг от друга на 4, будут использовать одну и ту же линию прерывания. В системах с несколькими шинами PCI перестановка устройства в слоты разных шин может влиять на производительность, что связано с характеристиками данной шины и ее удаленностью от главного моста.

Разобраться с нумерацией устройств и полученных ими линий прерываний на конкретной плате можно, если устанавливать карту PCI поочередно в каждый из слотов (отключая питание) и смотреть на сообщения об обнаруженных устройствах PCI, выводимых на дисплей в конце теста *POST*. В этих сообщениях будут фигурировать и устройства PCI, установленные непосредственно на системной плате (не отключенные параметрами *CMOS Setup*). Но чтобы не возникло иллюзии простоты, отметим, что «особо умные» операционные системы (например, Windows) не довольствуются полученными назначениями номеров прерываний и изменяют их по своему усмотрению.

Спецификации PCI и PCI-X

PCI (Peripheral Component Interconnect) local bus — шина соединения периферийных компонентов — является основной шиной расширения современных компьютеров. Как уже говорилось ранее, она разрабатывалась в расчете на процессоры Pentium, но хорошо сочеталась и с процессорами 486. Сейчас PCI является жестко стандартизованной высокопроизводительной и надежной шиной расширения, поддерживаемой рядом компьютерных платформ, включая PC-совместимые компьютеры, PowerPC и другие. Спецификации шины PCI периодически обновляются. Настоящее описание охватывает все стандарты шины PCI вплоть до версии 2.3 и PCI-X до версии 2.0. Рассмотрим основные этапы развития этой технологии:

- PCI 1.0 (1992 год) — определена общая концепция, описаны сигналы и протокол 32-разрядной параллельной синхронной шины с тактовой частотой до 33,3... МГц и пиковой пропускной способностью 132 Мбайт/с;
- в PCI 2.0 (1993 год) — введена спецификация коннекторов и карт расширения с возможным расширением разрядности до 64 бит (пропускная способность до 264 Мбайт/с), предусмотрены варианты питания интерфейсных схем напряжением 5 В и 3,3В;
- в версии 2.1 (1995 год) — введена частота 66 МГц (только для устройств с напряжением питания 3,3 В), что позволило обеспечить пиковую пропускную способность до 264 Мбайт/с в 32-битном варианте и 528 Мбайт/с в 64-битном;
- спецификация PCI 2.2 («PCI Local Bus Specification. Revision 2.2» от 18.12.1998) уточняет и разъясняет некоторые положения предшествующей версии 2.1; здесь появился новый механизм сигнализации прерываний MSI;

- в версии PCI 2.3 (2002 год) определены биты для прерываний, облегчающие идентификацию источника; отменены карты расширения с питанием 5 В (остались только универсальные и 3,3 В); введен низкопрофильный (low profile) конструктив карт расширения; добавлены сигналы дополнительной шины SMBus. Эта версия, описанная в документе PCI Local Bus Specification, Revision 2.3, является базой для современных расширений;
- в версии PCI 3.0 (2004 год) отменены системные платы на 5 В (остались только универсальные и 3,3 В).

На базе PCI 2.3 в 1999 году появилось *расширение PCI-X*, призванное существенно повысить пиковую пропускную способность шины за счет увеличения частоты передачи, а также повысить эффективность работы за счет оптимизации протокола. В протокол введены расщепленные транзакции и атрибуты, позволяющие участникам транзакции планировать свои действия. Расширение обеспечивают совместимость (механическую, электрическую и программную) устройств и системных плат PCI-X и обычной PCI, но, естественно, все устройства шины подстраиваются под самого слабого участника.

- В версии PCI-X 1.0 повышена тактовая частота до 133 МГц (для 3,3 В интерфейса), что дает варианты, называемые PCI-X66, PCI-X100, PCI-X133. Пиковая пропускная способность достигает 528 Мбайт/с в 32-битном варианте и более 1 Гбайт/с в 64-битном. Версия описана в документе PCI-X Addendum to the PCI Local Bus Specification, Revision 1.0b (2002 г.).
- В версии PCI-X 2.0 введены новые режимы синхронизации с удвоенной (PCI-X266) и учетверенной (PCI-X533) частотами передачи данных относительно тактовой частоты 133 МГц. Столь высокая частота требует низковольтного интерфейса (1,5 В), режима коррекции ошибок (ECC). Кроме 32- и 64-битных вариантов появился и 16-битный (для встроенных компьютеров). Добавлен новый тип транзакций — сообщения, адресуемые устройству по его идентификатору (DIM). Конфигурационное пространство функции расширено до 4096 байт. Версия PCI-X 2.0 описывается парой документов: PCI-X Protocol Addendum to the PCI Local Bus Specification, Revision 2.0 (PCI-X PT 2.0) — дополнения к протоколу и PCI-X Electrical and Mechanical Addendum to the PCI Local Bus Specification, Revision 2.0 (PCI-X EM 2.0) — дополнения к электрическим и механическим спецификациям.

В дополнение к спецификациям шины имеется ряд дополнительных спецификаций:

- спецификация на мосты, связывающие шины PCI друг с другом (и иными шинами), — PCI to PCI Bridge Architecture Specification, Revision 1.1 (PCI Bridge 1.1);
- спецификация PCI BIOS — конфигурирование устройств PCI и контроллера прерываний;
- обеспечение «горячего» подключения/отключения устройств — PCI Hot-Plug Specification, Revision 1.1 (PCI HP 1.1);

- управление энергопотреблением — PCI Power Management Interface Specification, Revision 1.1 (PCI PM 1.1).

На базе шины PCI 2.0 фирмой Intel был разработан выделенный интерфейс для подключения графического акселератора AGP (см. главу 7).

Спецификации PCI публикуются и поддерживаются организацией PCI SIG (Special Interest Group, http://www.pcisig.org). Шина PCI существует в разных конструктивных исполнениях: Mini-PCI, Small PCI, Card Bus, Compact PCI (CPCI), PXI.

ГЛАВА 2

Протокол, команды и транзакции шин PCI и PCI-X

Обмен информацией по шине PCI и PCI-X организован в виде *транзакций* — логически завершенных операций обмена. В типовой транзакции участвуют два устройства — *инициатор обмена* (initiator), он же *ведущее* устройство (master), и *целевое устройство* (ЦУ, target)), оно же *ведомое* (slave). Правила взаимодействия этих устройств определяются *протоколом шины PCI*. Устройство может следить за транзакциями на шине и не являясь их участником (не вводя никаких сигналов); режиму слежения соответствует термин *Snooping*. Есть особый тип транзакции (Special Cycle) — широковещательный, в котором инициатор протокольно не взаимодействует ни с одним из устройств. В каждой транзакции выполняется одна *команда* — как правило, чтение или запись данных по указанному адресу. Транзакция начинается с *фазы адреса*, в которой инициатор задает команду и целевой адрес. Далее могут следовать *фазы данных*, в которых одно устройство (источник данных) помещает данные на шину, а другое (приемник) их считывает. Транзакции, в которых присутствует множество фаз данных, называются *пакетными*. Есть и одиночные транзакции (с одной фазой данных). Транзакция может завершиться и без фаз данных, если целевое устройство (или инициатор) не готово к обмену. В шине PCI-X добавлена *фаза атрибутов*, в которой передается дополнительная информация о транзакции.

Сигнальный протокол шин PCI и PCI-X

Состав и назначение интерфейсных сигналов шины раскрывает табл. 2.1. Состояния всех сигнальных линий воспринимаются по положительному перепаду CLK, и именно эти моменты в дальнейшем описании подразумеваются под тактами шины (на рисунках отмечены вертикальными пунктирными линиями). В разные моменты времени одними и теми же сигнальными линиями управляют разные устройства шины, и для корректной (бесконфликтной) «передачи полномочий» требуется, чтобы существовал промежуток времени, в течение которого линией не управляет ни одно устройство. На временных диаграммах это событие — так называемый «пируэт» (turnaround) — обозначается парой полукруглых стрелок.

Таблица 2.1. Сигналы шины PCI

Сигнал	Назначение
AD[31:0]	Address/Data — мультиплексированная шина адреса/данных. В начале транзакции передается адрес, в последующих тактах — данные
C/BE[3:0]#	Command/Byte Enable — команда/разрешение обращения к байтам. Команда, определяющая тип очередного цикла шины, задается четырехбитным кодом в фазе адреса
FRAME#	Кадр. Введением сигнала отмечается начало транзакции (фаза адреса), снятие сигнала указывает на то, что последующий цикл передачи данных является последним в транзакции
DEVSEL#	Device Select — устройство выбрано (ответ ЦУ на адресованную к нему транзакцию)
IRDY#	Initiator Ready — готовность ведущего устройства к обмену данными
TRDY#	Target Ready — готовность ЦУ к обмену данными
STOP#	Запрос ЦУ к ведущему устройству на остановку текущей транзакции
LOCK#	Сигнал блокировки (захвата) шины для обеспечения целостного выполнения операции. Используется мостом, которому для выполнения одной операции требуется выполнить несколько транзакций PCI
REQ#	Request — запрос от ведущего устройства на захват шины
GNT#	Grant — предоставление ведущему устройству управления шиной
PAR	Parity — общий бит четности для линий AD[31:0] и C/BE[3:0]#
PERR#	Parity Error — сигнал об ошибке четности (для всех циклов, кроме специальных). Вырабатывается любым устройством, обнаружившим ошибку
PME#	Power Management Event — сигнал о событиях, вызывающих изменение режима потребления (дополнительный сигнал, введенный в PCI 2.2)
CLKRUN#	Clock running — шина работает на номинальной частоте синхронизации. Снятие сигнала означает замедление или остановку синхронизации с целью снижения потребления (для мобильных применений)
PRSNT[1,2]#	Present — индикаторы присутствия платы, кодирующие запрос потребляемой мощности. На карте расширения одна или две линии индикаторов соединяются с шиной GND, что воспринимается системной платой
RST#	Reset — сброс всех регистров в начальное состояние (по кнопке «Reset» и при перезагрузке)
IDSEL	Initialization Device Select — выбор устройства в циклах конфигурационного считывания и записи; на эти циклы отвечает устройство, обнаружившее на данной линии высокий уровень сигнала
SERR#	System Error — системная ошибка. Ошибка четности адреса или данных в специальном цикле или иная катастрофическая ошибка, обнаруженная устройством. Активизируется любым устройством PCI и вызывает NMI
REQ64#	Request 64 bit — запрос на 64-битный обмен. Сигнал вводится 64-битным инициатором, по времени он совпадает с сигналом FRAME#. Во время окончания сброса (сигналом RST#) сигнализирует 64-битному устройству о том, что оно подключено к 64-битной шине. Если 64-битное устройство не обнаружит этого сигнала, оно должно переконфигурироваться на 32-битный режим, отключив буферные схемы старших байтов

продолжение

Таблица 2.1 (*продолжение*)

Сигнал	Назначение
ACK64#	Подтверждение 64-битного обмена. Сигнал вводится 64-битным ЦУ, опознавшим свой адрес, одновременно с DEVSEL#. Отсутствие этого подтверждения заставит инициатор выполнять обмен с 32-битной разрядностью
INTA#, INTB#, INTC#, INTD#	Interrupt A, B, C, D — линии запросов прерывания, чувствительность к уровню, активный уровень — низкий, что допускает разделяемость (совместное использование) линий
CLK	Clock — тактовая частота шины. Должна лежать в пределах 20—33 МГц, начиная с PCI 2.1 может быть до 66 МГц, в PCI-X до 100 и 133 МГц
M66EN	66MHz Enable — разрешение частоты синхронизации до 66 МГц (на картах 33 МГц заземлен, на 66 МГц — свободен)
PCIXCAP (38B)	Возможности PCI-X: на платах PCI — заземлен, на PCI-X133 соединен с землей через конденсатор 0,01 мкФ, на PCI-X66 — параллельной RC-цепочкой 10 кОм, 0,01 мкФ.
SDONE	Snoop Done — сигнал завершенности цикла слежения для текущей транзакции. Низкий уровень указывает на незавершенность цикла слежения за когерентностью памяти и кэша. Необязательный сигнал, используется только устройствами шины с кэшируемой памятью. Исключен начиная с PCI 2.2
SBO#	Snoop Backoff — попадание текущего обращения к памяти абонента шины в модифицированную строку кэша. Необязательный сигнал, используется только абонентами шины с кэшируемой памятью при алгоритме обратной записи. Исключен начиная с PCI 2.2
SMBCLK	SMBus Clock — тактовый сигнал шины SMBus (интерфейс I^2C). Введен начиная с PCI 2.3
SMBDAT	SMBus Data — последовательные данные шины SMBus (интерфейс I^2C) . Введен начиная с PCI 2.3
TCK	Test Clock — синхронизация тестового интерфейса JTAG
TDI	Test Data Input — входные данные тестового интерфейса JTAG
TDO	Test Data Output — выходные данные тестового интерфейса JTAG
TMS	Test Mode Select — выбор режима для тестового интерфейса JTAG
TRST	Test Logic Reset — сброс тестовой логики

В каждый момент времени шиной может управлять только одно ведущее устройство, получившее на это право от арбитра. Каждое ведущее устройство имеет пару сигналов — REQ# для запроса на управление шиной и GNT# для подтверждения предоставления управления шиной. Устройство может начинать транзакцию (устанавливать сигнал FRAME#) только при полученном активном сигнале GNT# и дождавшись отсутствия активности шины. Заметим, что за время ожидания покоя арбитр может «передумать» и отдать управление шиной другому устройству с более высоким приоритетом. Снятие сигнала GNT# не позволяет устройству начать следующую транзакцию, а при определенных условиях (см. далее) может заста-

вить прекратить начатую транзакцию. Арбитражем запросов на использование шины занимается специальный узел — арбитр, входящий в мост, соединяющий данную шину с центром. Схема приоритетов (фиксированный, циклический, комбинированный) определяется программированием арбитра.

Для адреса и данных используются общие мультиплексированные линии AD. Четыре мультиплексированные линии C/BE[3:0] обеспечивают кодирование команд в фазе адреса и разрешение байтов в фазе данных. В транзакциях записи линии C/BE[3:0] разрешают использование байтов данных одновременно с их присутствием на шине AD, в транзакциях чтения эти сигналы относятся к байтам следующей за ними фазы данных. В фазе адреса (начало транзакции) ведущее устройство активирует сигнал FRAME#, передает целевой адрес по шине AD, а по линиям C/BE# — информацию о типе транзакции (команду). Адресованное целевое устройство отзывается сигналом DEVSEL#. Ведущее устройство указывает на свою готовность к обмену данными сигналом IRDY#, эта готовность может быть выставлена и до получения DEVSEL#. Когда и целевое устройство будет готово к обмену данными, оно установит сигнал TRDY#. Данные по шине AD передаются только при одновременном наличии сигналов IRDY# и TRDY#. С помощью этих сигналов ведущее и целевое устройства согласовывают свои скорости, вводя *такты ожидания* (wait states). На рис. 2.1 приведена временная диаграмма обмена, в которой и ведущее и целевое устройства вводят такты ожидания. Если бы они оба ввели сигналы готовности в конце фазы адреса и не снимали бы их до конца обмена, то в каждом такте после фазы адреса передавались бы по 32 бита данных, что обеспечило бы выход на предельную производительность обмена. В транзакциях чтения после фазы адреса необходим дополнительный такт для пируэта, во время которого инициатор прекращает управление линией AD; целевое устройство сможет взять на себя управление шиной AD только в следующем такте. В транзакции записи пируэт не нужен, поскольку данные передает инициатор.

На шине PCI все транзакции трактуются как *пакетные*: каждая транзакция начинается фазой адреса, за которой может следовать одна или несколько фаз данных. Количество фаз данных в пакете явно не указывается, но в такте последней фазы данных ведущее устройство при введенном сигнале IRDY# снимает сигнал FRAME#. В одиночных транзакциях сигнал FRAME# активен лишь в течение одного такта. Если устройство не поддерживает пакетные транзакции в ведомом режиме, то оно должно потребовать прекращения пакетной транзакции в течение первой фазы данных (выставив сигнал STOP# одновременно с TRDY#). В ответ на это ведущее устройство завершит данную транзакцию и продолжит обмен последующей транзакцией со следующим значением адреса. После завершающей фазы данных ведущее устройство снимает сигнал IRDY#, и шина переходит в *состояние покоя* (Idle) — оба сигнала: — FRAME# и IRDY# — находятся в пассивном состоянии.

Инициатор может начать следующую транзакцию и без такта покоя, установив FRAME# одновременно со снятием IRDY#. Такие *быстрые смежные транзакции* (Fast Back-to-Back) могут быть обращены как к одному, так и к разным целевым устройствам. Первый тип быстрых смежных транзакций поддерживается всеми устройствами PCI, выступающими в роли целевого устройства. На поддержку второ-

го типа смежных транзакций (такая поддержка необязательна) указывает бит 7 регистра состояния (см. главу 5). Инициатору разрешают (если он умеет) использовать быстрые смежные транзакции с различными устройствами (разрешение определяется битом 9 регистра команд), только если все агенты шины допускают быстрые обращения. При обмене данных в режиме PCI-X быстрые смежные транзакции недопустимы.

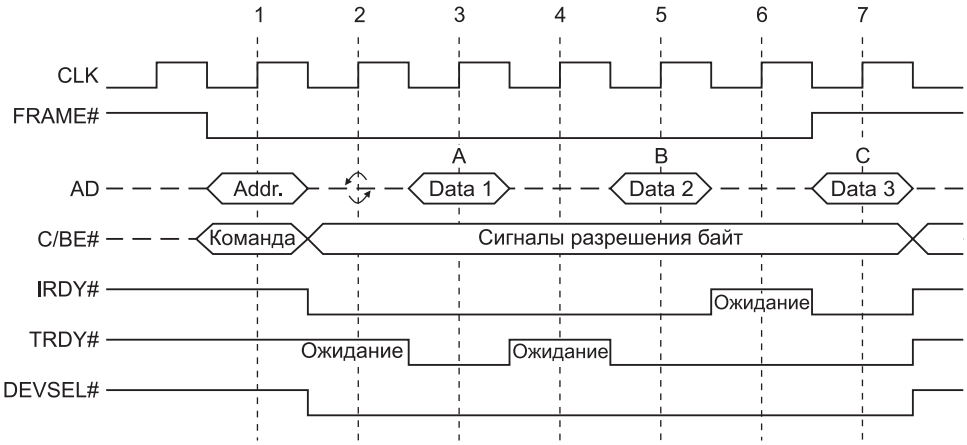


Рис. 2.1. Цикл обмена на шине PCI

Протокол шины обеспечивает *надежность обмена* — ведущее устройство всегда получает информацию об отработке транзакции целевым устройством. Средством *повышения достоверности обмена* является применение контроля четности: линии AD[31:0] и C/BE[3:0]# и в фазе адреса, и в фазе данных защищены битом четности PAR (количество установленных битов этих линий, включая PAR, должно быть четным). Действительное значение PAR появляется на шине с задержкой в один такт относительно линий AD и C/BE#. При обнаружении ошибки устройство вырабатывает сигнал PERR# (со сдвигом на такт после появления на шине действительного бита четности). В подсчете четности при передаче данных учитываются все байты, включая и недействительные (отмеченные высоким уровнем сигнала C/BEx#). Состояние бит, даже и в недействительных байтах данных, во время фазы данных должно оставаться стабильным.

Каждая транзакция на шине должна быть завершена планово или прекращена, при этом шина должна перейти в состояние покоя (сигналы FRAME# и IRDY# пассивны). Завершение транзакции выполняется либо по инициативе ведущего устройства, либо по инициативе целевого устройства.

Ведущее устройство может завершить транзакцию одним из следующих способов:

- *comletion — нормальное завершение* по окончании обмена данными;
- *time-out — завершение по тайм-ауту*. Происходит, когда во время транзакции у ведущего устройства отбирают право на управление шиной (снятием сигнала

GNT#), и истекает время, указанное в его таймере Latency Timer. Это может произойти, если адресованное целевое устройство оказалось непредвиденно медленным или запланирована слишком длинная транзакция. Короткие транзакции (с одной-двумя фазами данных) даже в случае снятия сигнала GNT# и срабатывания таймера завершаются нормально;

- *master-Abort — прекращение транзакции*, когда в течение заданного времени ведущее устройство не получает ответа от целевого устройства (сигнала DEVSEL#).

Транзакция может быть прекращена *по инициативе целевого устройства*; для этого оно может ввести сигнал STOP#. Возможны три типа прекращения транзакции:

- *retry — повтор*, введение сигнала STOP# при пассивном сигнале TRDY# до первой фазы данных. Эта ситуация возникает, когда целевое устройство из-за внутренней занятости не успевает выдать первые данные в положенный срок (16 тактов). Прекращение типа *retry* является указанием ведущему устройству на необходимость повторного запуска той же транзакции;
- *disconnect — отключение*, введение сигнала STOP# в течение или после первой фазы данных. Если сигнал STOP# введен при активном сигнале TRDY# очередной фазы данных, то эти данные передаются, на чем транзакция и завершается. Если сигнал STOP# выставлен при пассивном сигнале TRDY#, то транзакция завершается без передачи данных очередной фазы. Отключение производится, когда целевое устройство не способно своевременно выдать или принять очередную порцию данных пакета. Отключение является указанием ведущему устройству на необходимость повторного запуска этой транзакции, но с модифицированным стартовым адресом;
- *target-abort — отказ*, введение сигнала STOP# одновременно со снятием сигнала DEVSEL# (в предыдущих случаях во время появления сигнала STOP# сигнал DEVSEL# был активен). После этого данные уже не передаются. Отказ производится, когда целевое устрйство обнаруживает фатальную ошибку или иные условия, по которым оно уже никак не сможет обслужить данный запрос (в том числе и неподдерживаемую команду).

Использование трех типов прекращения транзакции вовсе не обязательно для всех целевых устройств, однако любое ведущее устройство должно быть готово к завершению транзакций по любой из этих причин.

Прекращение типа *retry* используется для организации *отложенных транзакций* (delayed transactions). Отложенные транзакции используются только медленными целевыми устройствами, а также мостами PCI при трансляции транзакций на другую шину. Прекращая (для инициатора) транзакцию условием *retry*, целевое устройство внутренне выполняет данную транзакцию. Когда инициатор повторит эту транзакцию (выдаст ту же команду с тем же адресом и тем же набором сигналов C/BE# в фазе данных), у целевого устройства (или моста) уже будет готов результат (данные чтения или состояние выполнения записи), который оно быстро вернет инициатору. Результат отложенной транзакции, выполненной данным устройством, устройство или мост должны хранить до тех пор, пока результаты не будут запрошены инициатором. Однако он может и «забыть» повторить транзак-

цию (из-за каких-либо нештатных ситуаций). Чтобы избежать переполнения буфера хранения результатов, устройству приходится отбрасывать (discard) эти результаты. Отбрасывание может быть выполнено без побочных эффектов, если откладывалась транзакция к памяти, допускающей предвыборку (с атрибутом *prefetchable*, см. далее). Остальные типы транзакций в общем случае безнаказанно отбрасывать нельзя (может нарушиться целостность данных), для них отбрасывание разрешается только после безрезультатного ожидания повтора в течение 2^{15} тактов шины (по срабатыванию `discard timer`). Об этой особой ситуации устройство может сообщить своему драйверу (или всей системе).

Инициатор транзакции может потребовать монопольного использования шины PCI на все время выполнения операции обмена, требующей нескольких шинных транзакций. Так, например, если центральный процессор выполняет инструкцию модификации данных в ячейке памяти, принадлежащей устройству PCI, ему нужно прочитать данные из устройства, модифицировать их в своем АЛУ и вернуть результат в устройство. Чтобы в эту операцию не вклинивались транзакции от других инициаторов (что чревато нарушением целостности данных), главный мост выполняет ее как *блокированную* — на все время исполнения операции подается шинный сигнал `LOCK#`. Этот сигнал никак не используется (и не вырабатывается) обычными устройствами PCI (не мостами); он используется только мостами для управления арбитражем.

Команды шины PCI

Команды PCI определяют направление и тип транзакций, а также адресное пространство, к которому они относятся. Набор команд шины PCI включает следующие:

- *I/O Read*, *I/O Write* — *команды чтения и записи ввода/вывода*, служат для обращения к пространству портов;
- *Memory Read*, *Memory Write* — *команды чтения и записи памяти*, служат для выполнения коротких (как правило, непакетных) транзакций. Их прямое назначение — обращение к отображенным на память устройствам ввода/вывода. Для «настоящей» памяти, допускающей предвыборки, предназначены команды чтения строк, множественного чтения и записи с инвалидацией;
- *Memory-Read Line* — *чтение строки памяти*, применяется, когда в транзакции планируется чтение до конца строки кэша. Выделение данного типа чтения позволяет повысить производительность обмена с памятью;
- *Multiple Memory Read* — *множественное чтение памяти*, используется для транзакций, затрагивающих более одной строки кэш-памяти. Использование данного типа транзакций позволяет контроллеру памяти выполнять упреждающую выборку строк, что дает дополнительный выигрыш производительности;
- *Memory Write and Invalidate* (MVI) — *запись с инвалидацией*, применяется к целым строкам кэша, причем все байты во всех фазах должны быть разрешены.

Эта операция заставляет контроллер кэш-памяти очищать «грязные» строки кэша, соответствующие записываемой области, без их выгрузки в ОЗУ, что экономит время. Инициатор, вводящий эту команду, должен знать размер строки кэша в данной системе (для этого у него есть специальный регистр в конфигурационном пространстве);

- *Dual Address Cycle* (DAC) — *двухадресный цикл*, позволяет по 32-битной шине обращаться к устройствам с 64-битной адресацией. В этом случае младшие 32 бита адреса передаются одновременно с данной командой, а далее следует обычный цикл, определяющий команду обмена и несущий старшие 32 бита адреса. Шина PCI допускает 64-битную адресацию как памяти, так и портов ввода/вывода (последнее для систем на x86 бесполезно, но PCI существует и на других платформах);
- *Configuration Read*, *Configuration Write* — *команды конфигурационного чтения и записи*, адресуются к конфигурационному пространству устройств. Обращение производится только выровненными двойными словами, биты AD[1:0] используются для идентификации типа цикла (см. ниже). Для генерации данных команд требуется специальный аппаратно-программный механизм (см. главу 5);
- *Special Cycle* — *специальный цикл*, отличается от всех других тем, что является широковещательным. Однако ни один агент на него не отвечает, а главный мост или иное устройство, вводящее этот цикл, всегда завершает его способом *Master Abort* (на него требуется 6 тактов шины). Специальный цикл предназначен для генерации широковещательных сообщений, которые могут читать любые «заинтересованные» агенты шины. Тип сообщения декодируется содержимым линий AD[15:0]; на линиях AD[31:16] могут помещаться данные, передаваемые в сообщении. Фаза адреса в этом цикле обычными устройствами игнорируется, но мосты используют ее информацию для управления распространением сообщения. Сообщения с кодами 0000h, 0001h и 0002h требуются для указания на отключение (сообщение *Shutdown*), остановку (сообщение *Halt*) процессора или специфические функции процессора x86, связанные с кэшем и трассировкой. Коды 0003h–FFFFh зарезервированы. Специальный цикл может генерироваться тем же аппаратно-программным механизмом, что и конфигурационные циклы, но со специфическим значением адреса;
- *Interrupt Acknowledge* (INTA) — *команда подтверждения прерывания*, предназначена для чтения вектора прерываний. По протоколу она выглядит как команда чтения, неявно адресованная к системному контроллеру прерываний. Здесь в фазе адреса по шине AD полезная информация не передается (BE[3:0]# задают размер вектора), но ее инициатор (главный мост) должен обеспечить стабильность сигналов и корректность бита четности. В x86-архитектуре 8-битный вектор передается в байте 0 по готовности контроллера прерываний (по сигналу TRDY#). Подтверждение прерываний выполняется за один цикл (первый холостой цикл, который процессоры x86 делают в дань совместимости со стариной, мостом подавляется).

Команды кодируются значениями бит C/BE# в фазе адреса (табл. 2.2), специфические команды PCI-X рассмотрены в последующих разделах.

Таблица 2.2. Декодирование команд шин PCI и PCI-X

Код	PCI	PCI-X		
C/BE[3:0]	Команда	Команда	Длина	Возможность расщепления
0000	*Interrupt Acknowledge*	*Interrupt Acknowledge*, подтверждение прерывания	DWORD	+
0001	*Special Cycle*	*Special Cycle*, специальный широковещательный цикл	DWORD	–
0010	*I/O Read*	*I/O Read*, чтение ввода/вывода	DWORD	+
0011	*I/O Write*	*I/O Write*, запись ввода/вывода	DWORD	+
0100	Резерв	Резерв	–	–
0101	Резерв	*Device ID Message* (*DIM*), посылка собщения устройству (PCI-X 2.0)	Пакет	–
0110	*Memory Read*	*Memory Read DWORD*, одиночное чтение памяти	DWORD	+
0111	*Memory Write*	*Memory Write*, запись памяти	Пакет	–
1000	Резерв	*Alias to Memory Read Block*, псевдоним чтения блока памяти	Пакет	+
1001	Резерв	*Alias to Memory Write Block*, псевдоним записи блока памяти	Пакет	–
1010	*Configuration Read*	*Configuration Read*, конфигурационное чтение	DWORD	+
1011	*Configuration Write*	*Configuration Write*, конфигурационная запись	DWORD	+
1100	*Memory Read Multiple*	*Split Completion*, завершение расщепленной транзакции	Пакет	–
1101	*Dual Address Cycle*	*Dual Address Cycle* (*DAC*), цикл передачи расширенного адреса памяти	–	–
1110	*Memory Read Line*	*Memory Read Block*, чтение блока памяти	Пакет	+
1111	*Memory Write and Invalidate*	*Memory Write Block*, запись блока памяти	Пакет	–

В каждой команде шины указывается адрес, относящийся к первой фазе данных пакета. Для каждой последующей фазы данных пакета адрес увеличивается на 4 (следующее двойное слово) или 8 (для 64-битных передач), но в командах обращения к памяти предусматривался и иной порядок (см. далее).

В шине PCI байты шины AD, несущие реальную информацию, определяются сигналами C/BE[3:0]# в фазах данных. Разрешенные байты могут быть разрозненными; возможны фазы данных, в которых не разрешено ни одного байта. В PCI-X правила разрешения байтов изменились (см. ниже). Сигналами C/BE[3:0]# управляет инициатор, он указывает требуемые байты для каждой фазы данных и не меняет состояние этих сигналов в течение всей этой фазы. В транзакциях чтения байты «заказывает» опять же инициатор; если поведение целевого устройства (источника данных для чтения) зависит от того, какие байты заказаны, то целевое устройство вынуждено растягивать каждую фазу данных. При этом в первом такте каждой фазы данных целевое устройство принимает C/BE[3:0]# и только в последующем такте (а может, и с дополнительным ожиданием) выдает данные чтения.

В отличие от шины ISA на PCI нет динамического изменения разрядности — все устройства должны подключаться к шине 32- или 64-разрядным способом. Если в устройстве PCI применяются функциональные схемы иной разрядности (к примеру, нужно подключить микросхему 8255, имеющую 8-битную шину данных и четыре регистра), то приходится применять схемотехнические методы преобразования, отображающие все регистры на 32-разрядную шину AD. Возможность 16-битных подключений появилась только во второй версии PCI-X.

Для каждого из трех пространств — памяти, портов ввода/вывода и конфигурационных регистров — адресация различна; в специальных циклах адрес игнорируется.

Адресация памяти

На шине PCI передается *физический* адрес памяти; в процессорах x86 (и других) он получается из логического посредством постраничной табличной трансляции блоком MMU.

В *командах обращения к памяти* на шине PCI адрес, выровненный по границе двойного слова, передается по линиям AD[31:2]; линии AD[1:0] задают порядок адресов в пакете:

- 00 — *линейное инкрементирование*; адрес последующей фазы отличается от предыдущего на число байтов шины (4 для 32-битной и 8 для 64-битной шины);
- 10 — *сворачивание адресов* с учетом длины строки кэш-памяти (Cacheline Wrap mode). В транзакции адрес для очередной фазы увеличивается до достижения границы строки кэша, после чего переходит на начало этой строки и увеличивается до адреса, предшествующего начальному. Если транзакция длиннее строки кэша, то она продолжится в следующей строке с того же смещения, что и началась. Так, при длине строки 16 байт и 32-битной шине транзакция, начавшаяся

с адреса xxxxxx08h, будет иметь последующие фазы данных, относящиеся к адресам xxxxxx0Ch, xxxxxx00h, xxxxxx04h; и далее к xxxxxx18h, xxxxxx1Ch, xxxxxx10h, xxxxxx14h. Длина строки кэша прописывается в конфигурационном пространстве устройства (см. главу 5). Если устройство не имеет регистра `Cache Line Size`, то оно должно прекратить транзакцию после первой фазы данных, поскольку порядок чередования адресов оказывается неопределенным;

- 01 и 11 — зарезервированы, могут использоваться как указание на *отключение* (Disconnect) после первой фазы данных.

Если требуется доступ к адресам свыше 4 Гбайт, то используется двухадресный цикл, передающий младшие 32 бита полного 64-битного адреса для последующей команды, вместе с которой передаются старшие биты адреса. В обычных командах подразумевается нулевое значение бит [63:32] полного адреса.

В *PCI-X* передается полный адрес памяти — используются все линии AD[31:0]. В пакетных транзакциях адрес определяет точное положение начального байта пакета, и всегда подразумевается линейный нарастающий порядок адресов. В пакетной транзакции участвуют все байты, начиная с данного и до последнего (по счетчику байтов). Запреты отдельных байтов в пакете (как в PCI) для транзакций PCI-X недопустимы. В одиночных транзакциях (DWORD) биты адреса AD[1:0] определяют байты, которые могут быть разрешены сигналами C/BE[3:0]#[1]: При AD[1:0] = 00 допустимо C/BE[3:0]# = xxxx (могут быть разрешены любые байты), при AD[1:0] = = 01 — C/BE[3:0]# = xxx1, при AD[1:0] = 10 — C/BE[3:0]# = xx11, при AD[1:0] = 11 — — C/BE[3:0]# = x111 (передается лишь байт 3 или ни один байт не разрешен).

Адресация ввода/вывода

В *командах обращения к портам ввода/вывода* для адресации любого байта используются (декодируются) все линии AD[31:0]. При этом биты адреса AD[31:2] указывают на адрес двойного слова, к которому принадлежат передаваемые данные, а младшие биты адреса AD[1:0] определяют байты, которые могут быть разрешены сигналами C/BE[3:0]#. Для *транзакций ввода/вывода на PCI* правила несколько иные, чем для обращений к памяти: если передается хотя бы один байт, то всегда должен быть разрешен и тот байт, на который указывает адрес. При AD[1:0] = 00 допустимо C/BE[3:0]# = xxx0 или 1111, при AD[1:0] = 01 — C/BE[3:0]# = xx01 или 1111, при AD[1:0] = 10 — C/BE[3:0]# = x011 или 1111, при AD[1:0] = 11 — C/BE[3:0]# = = 0111 (передается лишь байт 3) или 1111 (ни один байт не разрешен). Эти циклы формально тоже могут быть пакетными, хотя реально эта возможность практически не используется. Для адресации портов на шине PCI доступны все 32 бита адреса, но процессоры x86 могут использовать только младшие 16 бит.

Для *транзакций ввода/вывода на PCI-X* распространяются те же взаимосвязи сигналов C/BE[3:0]# и те же адреса, что и для одиночных (DWORD) транзакций с памятью, приведенные в предыдущем разделе. Эти транзакции всегда одиночные (DWORD).

---

[1] Сигналы C/BE[3:0]# инверсные: 0 — байт разрешен, 1 — запрещен.

Адресация конфигурационных регистров и специальный цикл

В *командах конфигурационной записи/считывания* применяется специфическая трактовка адреса, здесь формат адреса может быть одним из двух типов. Для доступа к регистрам устройства, расположенного на данной шине, используются *конфигурационные транзакции типа 0* (Type 0, рис. 2.2, *а*). При этом *устройство* (карта расширения) выбирается индивидуальным сигналом IDSEL, который формируется мостом этой шины из номера устройства. Выбранное устройство на шине в битах AD[10:8] «видит» *номер функции* Fun, а в битах AD[7:2] — *номер конфигурационного регистра* Reg, при этом AD[1:0] = 00 является признаком типа 0. Линии AD[31:11] используются в качестве источника сигналов IDSEL для устройств данной шины. В спецификации на саму шину по линии AD11 передается IDSEL для устройства 0, по AD12 — для устройства 1, ... по AD31 — для устройства 20. В спецификации на мосты приводится таблица, в которой используются только линии с AD16 (устройство 0) по AD31 (устройство 15). Устройства PCI, скомбинированные с мостом (расположенные с ним в одной микросхеме), могут использовать и большие номера, для которых линий AD уже не хватает. В PCI-X по линиям AD[15:11] передается недекодированный *номер устройства* Dev — он нужен устройству для передачи в качестве части своего идентификатора в атрибутах транзакции. Для устройств, работающих в Mode 1, для IDSEL используются линии AD[31:16] (рис. 2.2, *б*), а для Mode 2 — только AD[23:16], так что максимальный номер устройства — 7. Это позволяет расширить конфигурационное пространство функции до 4 Кбайт: в качестве *старших бит номера конфигурационного регистра* UReg используются линии AD[27:24] (рис. 2.2, *в*).

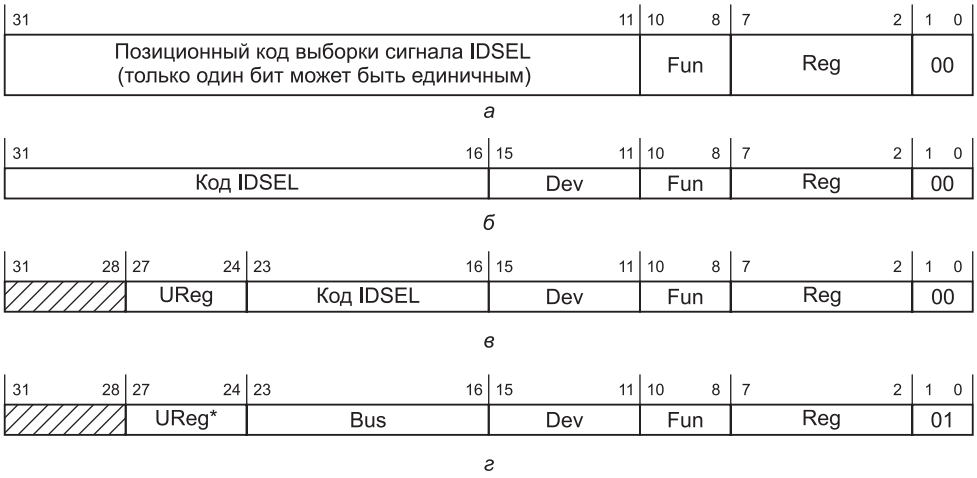


Рис. 2.2. Формат адреса в конфигурационных циклах: *а* — цикл типа 0 для шины PCI; *б* — цикл типа 0 для шины PCI-X Mode 1; *в* — цикл типа 0 для шины PCI-X Mode 2; *г* — цикл типа 1 (* в PCI — резерв)

Для доступа к устройствам других шин используются *конфигурационные транзакции типа 1* (Type 1, рис. 2.2, *г*). Здесь *номер шины* Bus, на котором расположено искомое устройство, определяется битами AD[23:16]; *номер устройства* Dev — битами AD[15:11], *номер функции* Fun — битами AD[10:8]; *номер регистра* Reg — битами AD[7:2]; при этом AD[1:0] = 01 является признаком типа 1. В PCI-X для Mode 2 по AD[27:24] передаются *старшие биты номера регистра* (UReg).

Поскольку биты AD[1:0] используются для идентификации типа транзакции, конфигурационные регистры адресуются только двойными словами. Различение двух типов конфигурационных транзакций используется для построения иерархической системы конфигурирования PCI. Мост, приведший конфигурационную транзакцию к шине, на которой расположено целевое устройство, преобразует транзакцию типа 1 в транзакцию типа 2. Заметим, что в отличие от транзакций по адресам памяти и ввода/вывода, которые от инициатора до адресованного целевого устройства доберутся при любом их взаимном расположении, конфигурационные транзакции распространяются по иерархии шин только «вниз» — от хоста (центрального процессора) через главную шину к подчиненным. Таким образом, только хост может выполнить конфигурирование всех устройств PCI (включая и мосты), и это его «почетная обязанность».

В широковещательной команде PCI, называемой *специальным циклом*, в фазе адреса информация по шине AD не передается. Любой агент шины PCI может вызвать специальный цикл на любой конкретно заданной шине, используя транзакцию конфигурационной записи типа 1, указав номер шины в битах AD[23:16]. При этом в полях номеров устройства и функции (в битах AD[15:8]) должны быть все единицы, в поле номера регистра — нули. Эта транзакция проходит между шинами независимо от взаимного расположения источника и целевой шины, и только самым последним мостом, приведшим ее к целевой шине, преобразуется в собственно специальный цикл.

Модификация протокола в PCI-X

Протокол шины PCI-X во многом совпадает с вышеописанным: то же тактирование по перепаду CLK, то же назначение управляющих сигналов. Изменение протокола нацелено на повышение эффективности использования тактов шины. Для этого в протокол ввели дополнения, позволяющие устройствам «предвидеть» грядущие события и выбирать адекватное поведение.

В обычной PCI все транзакции начинаются одинаково (с фазы адреса) как пакетные с заранее не известной длиной. При этом реально транзакции ввода/вывода всегда имеют лишь одну фазу данных; длинные пакеты эффективны (и используются) только для обращений к памяти. В PCI-X транзакции по длине разделены на два типа:

- *пакетные* (*Burst*) — все команды, обращенные к памяти, кроме *Memory Read DWORD*;
- *одиночные* размером в двойное слово (*DWORD*) — остальные команды.

В каждой транзакции после фазы адреса присутствует новая *фаза передачи атрибутов* транзакции, в которой инициатор сообщает свой *идентификатор* (RBN — *номер шины*, RDN — *номер устройства* и RFN — *номер функции*), 5-битный *тег*, 12-битный *счетчик байтов* (только для пакетных транзакций, UBC — старшие биты, LBC — младшие биты) и дополнительные характеристики (биты RO и NS) области памяти, к которой относится транзакция. Атрибуты передаются по линиям шины AD[31:0] и BE[3:0]# (рис. 2.3). Идентификатор инициатора вместе с тегом определяют *последовательность* (Sequence) — одну или несколько транзакций, обеспечивающих одну *логическую передачу* данных, запланированную инициатором. Благодаря 5-битному тегу каждый инициатор может одновременно выполнять до 32 логических передач (повторное назначение тега другой логической передаче возможно только после завершения предшествующей, использовавшей то же значение тега). Логическая передача (последовательность) может иметь длину до 4096 байт (значение счетчика байтов 00…01 соответствует числу 1, 11…11 — 4095, 00…00 — 4096); в атрибутах каждой транзакции указывается число байт, которые должны быть переданы до конца данной последовательности. Количество байт, которые будут переданы в каждой транзакции, заранее не определено (транзакцию может остановить как инициатор, так и целевое устройство). Однако для повышения эффективности работы к пакетным транзакциям предъявляются жесткие требования. Если в транзакции оказывается более одной фазы данных, то она может завершаться либо по передаче всех заявленных байтов (по счетчику в атрибутах), либо только на границах строк кэша (по 128-байтным границам адресов памяти). Если участники транзакции не готовы принять такие условия, кто-то из них должен остановить транзакцию после первой фазы данных. Только у целевого устройства есть еще право аварийного завершения транзакции в любой момент; инициатор жестко обязан отвечать за свои начинания.

Байты шины AD, участвующие в транзакциях, определяются сигналами BEx#, но иначе, чем в PCI. Для одиночных транзакций эти сигналы действуют в фазе атрибутов (на рис. 2.3 обозначены полем BE). Для пакетных транзакций эти сигналы действуют только в команде *Memory Write* (в каждой фазе данных), для остальных пакетных обращений предполагается, что все байты, от начального адреса до конечного, разрешены.

Характеристики памяти, к которой относится транзакция, позволяют выбирать оптимальный способ обращения к ней при отработке транзакции. Характеристики устанавливает устройство, запрашивающее данную последовательность. Каким образом оно узнает о свойствах памяти — забота его драйвера. Атрибуты характеристики памяти относятся только к транзакциям пакетных обращений к памяти (но не к сообщениям MSI):

- флаг RO (*Relaxed Ordering*) означает, что возможно изменение порядка выполнения отдельных операций записи и чтения;
- флаг NS (*No Snoop*) означает, что область памяти, к которой относится данная транзакция, нигде не кэшируется.

В PCI-X отложенные транзакции (Delayed Transaction) заменены на *расщепленные транзакции* (Split Transaction). Любую транзакцию, кроме всех транзакций записи в память, целевое устройство может завершать либо немедленно (обыч-

ным для PCI способом), либо с использованием протокола расщепленных транзакций. В последнем случае целевое устройство подает сигнал *Split Response* (расщепление), внутренне исполняет команду, а потом инициирует собственную транзакцию (команда *Split Completion*) для пересылки данных или сообщения о завершении инициатору исходной (расщепленной) транзакции. Целевое устройство обязано расщеплять транзакцию, если не может ответить на нее до истечения начальной задержки (initial latency). Устройство, вызвавшее расщепляемую транзакцию, называется *запросчиком* (Requester). Устройство, *завершающее* расщепленную транзакцию (Completer), будем называть *исполнителем*. Для завершения транзакции исполнитель должен будет запросить управление шиной у арбитра; запросчик на этапе завершения будет выступать в роли целевого устройства. Завершать транзакцию расщепленным способом может устройство, даже и не являющееся формально мастером шины (по признакам в его конфигурационных регистрах). *Транзакция завершения Split Completion* во многом напоминает пакетную транзакцию записи, но отличается в фазе адресации: вместо полного адреса пространства памяти или ввода/вывода по шине AD *передается идентификатор последовательности* (с номером шины, устройства и функции запросчика), к которой относится это завершение, и только младшие 6 бит адреса (рис. 2.4). Исполнитель берет этот идентификатор из атрибутов расщепленной им транзакции. По этому идентификатору (номеру шины запросчика) мосты доводят транзакцию завершения до устройства-запросчика. В фазе атрибутов передается идентификатор исполнителя (CBN — номер шины, CDN — номер устройства и CFN — номер функции, рис. 2.3, *в*). Запросчик должен распознать свой идентификатор последовательности и ответить на транзакцию обычным способом (немедленно). Последовательность может отрабатываться и не одной транзакцией завершения, а их серией, до исчерпания счетчика байтов (или прекращаться по ошибке). К какому стартовому адресу относится каждая из транзакций завершения, запросчик вычисляет сам (он знает, что запрашивал и сколько байтов уже пришло). Транзакция завершения может нести либо запрошенные данные чтения, либо сообщение о результатах транзакции — *Split Complete Message*.

Запросчик должен быть всегда готов к получению данных начатых им последовательностей, причем данные разных последовательностей могут приходить в произвольном порядке. Исполнитель может выдавать транзакции завершения на несколько последовательностей также в произвольном порядке. В пределах каждой последовательности завершения, естественно, должны быть упорядочены по адресам (которые не передаются). Атрибуты в транзакции завершения содержат номер шины, устройства и функции исполнителя и счетчик байтов. Кроме того, здесь присутствуют три специфических флага:

- BCM (Byte Count Modified) — признак того, что будет передано меньше байтов данных, чем просил запросчик (передается с данными завершения);
- SCE (Split Completion Error) — признак ошибки завершения, устанавливается при передаче сообщения завершения как ранний признак ошибки (до декодирования самого сообщения);
- SCM (Split Completion Message) — признак сообщения (отличает сообщение от данных).

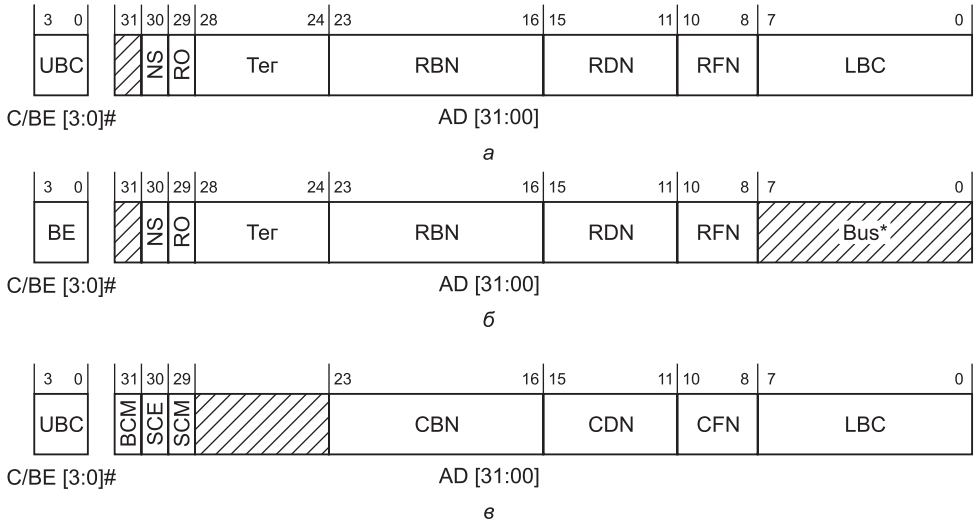


Рис. 2.3. Форматы атрибутов транзакции PCI-X: *а* — пакетной, *б* — одиночной (* биты [7:0] используются только в цикле конфигурационной записи), *в* — завершения расщепленной

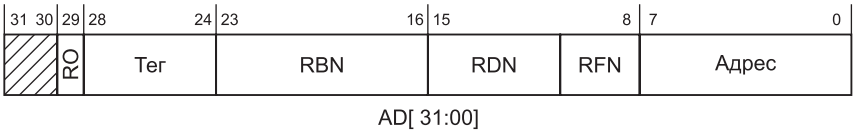


Рис. 2.4. Формат адреса завершения расщепленной транзакции PCI-X

Особенности передачи данных в PCI-X 2.0

В PCI-X 2.0 вдобавок к вышеописанным изменениям протокола появился новый режим *Mode 2*, отличающийся ускорением блочной записи в память и применением ECC-контроля. Этот режим возможен только при низком (1,5 В) напряжении питания интерфейсных схем. Режим *Mode 2* имеет следующие особенности:

- во всех транзакциях на 1 такт увеличено время декодирования адреса целевым устройством — задержки его ответа сигналом DEVSEL# на обращенную к нему команду. Этого лишнего такта требует ECC-контроль (устройство проверяет достоверность адреса и команды). По той же причине минимальное время покоя шины между транзакциями увеличено с 1 до 2 тактов;
- в *транзакциях пакетной записи в память* (команда *Memory Write Block*) используется удвоенная или учетверенная скорость передачи данных по отношению к тактовой частоте. В этих транзакциях сигналы BEx# используются для *синхронизации от источника данных* (по прямому назначению они не используются, поскольку подразумевается обязательное разрешение всех байтов). Каждая передача данных (32, 64 или 16 бит) сопровождается стробами, в качестве которых используются сигналы BEx#. Пары линий BE[1:0]# и BE[3:2]# передают диф-

ференциальные стробирующие сигналы для линий данных AD[15:0] и AD[31:16] соответственно. В одном такте шины может быть две или четыре *подфазы данных* (data subphase), этим и обеспечиваются режимы PCI-X266 и PCI-X533 при частоте шины 133 МГц. Поскольку все управляющие сигналы синхронизируются по сигналу общей синхронизации (CLK), гранулярность передач становится равной двум или четырем подфазам данных. Для 32-разрядной шины это означает, что в транзакциях можно передавать (а также останавливать и приостанавливать передачи) данные порциями, кратными 8 или 16 байтам.

В 64-битном варианте шины линии AD[63:32] используются только в фазах данных; для адреса (даже 64-битного) и атрибутов используется только 32-битная шина.

Для устройств, работающих в *Mode 2*, вводится возможность использования 16-битной шины. При этом фазы адреса и атрибутов занимают по два такта, а фазы данных идут всегда парами (обеспечивая обычную гранулярность). В шине AD используются линии AD[16:31], по которым в первой фазе пары передаются биты [0:15], а во второй — [16:31]. По линиям C/BE[2:3]# в первой фазе передаются биты C/BE[0:1]#, а во второй — C/BE[2:3]#. Для ECC-контроля используются линии ECC[2:5], по которым в первой фазе передаются биты ECC[0, 1, 6] и специальный бит контроля E16, а во второй — ECC[2, 3, 4, 5]. 16-битная шина предназначена только для встроенных применений (слоты и карты расширения не предусматриваются).

Обмен сообщениями между устройствами (команда DIM)

В PCI-X 2.0 введена возможность передачи информации (сообщений) устройству, адресуясь с помощью *идентификатора* (номера шины, устройства и функции). Для адресации и маршрутизации этих сообщений, которыми могут обмениваться любые устройства шины (включая и главный мост), не используется адресное пространство памяти или ввода/вывода. Сообщения передаются последовательностями, в которых используются команды *DIM* (Device ID Message), отличающиеся специфичностью адреса и атрибутов. В *фазе адреса* (рис. 2.5, *а*) передается *идентификатор получателя* сообщений (Completer ID) — номер его шины (CBN), устройства (CDN) и функции (CFN). Бит RT (Route Type) указывает тип маршрутизации сообщения: 0 — явная адресация с использованием вышеуказанного идентификатора, 1 — неявная адресация к главному мосту (при этом идентификатор не используется). Бит SD (Silent Drop) задает способ отработки ошибок при выполнении данной транзакции: 0 — обычный (как для записи в память), 1 — игнорирование некоторых типов ошибок (но не контроля четности или ECC). Поле Message Class задает класс сообщения, в соответствии с которым трактуется младший байт адреса. Транзакция может использовать и двухадресный цикл, при этом в первой фазе адреса по линиям C/BE[3:0]# передается код команды *DAC*, содержимое бит AD[31:00] соответствует рис. 2.5, *а*. Во второй фазе адреса по линиям C/BE[3:0]# передается код команды *DIM*, а все биты AD[31:00] трактуются в зависимости от класса

сообщения. Устройство, поддерживающее обмен сообщениями, декодировав команду *DIM*, проверяет поля идентификаторов получателя на соответствие своему собственному.

В *фазе атрибутов* (рис. 2.5, *б*) передается *идентификатор источника сообщения* (RBN, RDN и RFN), *тег сообщения* (Tag), 12-битный *счетчик байтов* (UBC и LBC) и дополнительные биты-признаки. Бит IR (Initial Request) является признаком начала сообщения, которое может быть разорвано на несколько частей инициатором, получателем или промежуточными мостами (во всех последующих частях бит обнулен). Бит RO (Relaxed Ordering) указывает на возможность неупорядоченной доставки данного сообщения по отношению к другим сообщениям и записям в память, распространяемым в том же направлении (порядок доставки фрагментов данного сообщения сохраняется всегда).

Тело сообщения, передаваемое в фазах данных, может иметь длину до 4096 байт (предел обусловлен 12-битным счетчиком длины). Содержимое тела определяется классом сообщения; класс 0 отдается на использование по воле производителя.

Сообщения с явной маршрутизацией маршрутизируются мостами на основе номера шины получателя. Проблемы передачи могут возникать только на главных мостах: если в системе имеется несколько главных мостов, то архитектурная связь между ними может быть очень сложной (например, через магистрали контроллера памяти). Передача сообщений с шины на шину через главные мосты желательна (это проще, чем передача транзакций всех типов), но не строго обязательна. Поддержка этой передачи дает больше свободы пользователю (не приходится при расстановке устройств принимать во внимание всю топологию шин). *Сообщения с неявной маршрутизацией* передаются только по направлению к хосту.

Поддержка *DIM* для устройств PCI-X необязательна; мосты PCI-X Mode 2 обязаны поддерживать *DIM*. Если сообщение DIM адресуется к устройству, находящемуся на шине, работающей в стандартном режиме PCI (или путь к нему ведет через PCI), мост либо просто аннулирует это сообщение (если SD = 1), либо отвергает транзакцию (*Target Abort*, если SD = 0).

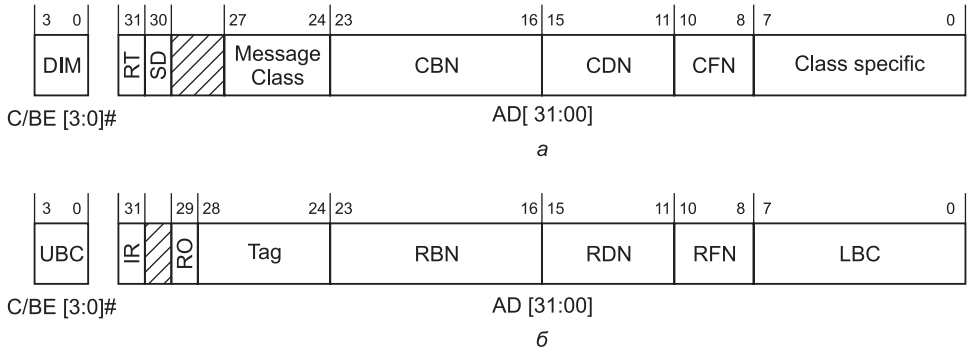


Рис. 2.5. Форматы в транзакции DIM: *а* — адрес, *б* — атрибуты

Границы диапазонов адресов и транзакций

Области пространств памяти и ввода/вывода, занимаемые устройством (точнее, функцией), описываются регистрами BAR (Base Address Register) в заголовке конфигурационного пространства. При этом подразумевается, что длина области выражается числом 2^n ($n = 0, 1, 2...$) и область выровнена естественным образом. В PCI области памяти выделяются по 2^n параграфов (16 байт), то есть минимальный размер области — 16 байт. Области ввода/вывода выделяются по 2^n двойных слов. Мосты PCI-PCI имеют карты адресов памяти с гранулярностью 1 Мбайт и карты ввода/вывода с гранулярностью 4 кбайт.

В PCI пакетная транзакция может быть прервана на границе любого двойного слова (в 64-битных операциях — учетверенного слова). В PCI-X ради оптимизации обращений к памяти пакетные транзакции разрешается прерывать только в разрешенных точках, называемых *ADB* (Allowable Disconnect Boundary — разрешенные границы отключения). Точки *ADB* располагаются с интервалом 128 байт — это целое число (1, 2, 4 или 8) строк кэша современных процессоров. Конечно, это ограничение относится только к границам транзакций внутри последовательности. Если последовательность должна по плану заканчиваться не на границе *ADB*, то и ее последняя транзакция будет завершена не на границе. Однако этой ситуации стараются избегать, разрабатывая такие структуры данных, которые могут быть выровнены подходящим образом (иногда даже ценой избыточности).

С границами адресов связан термин *ADQ* (ADB Delimited Quantum) — часть транзакции или буферной памяти (в мостах и устройствах), лежащая между границами соседних *ADB*. Например, транзакция, пересекающая одну границу *ADB*, состоит из двух *ADQ* (квантов) данных и занимает в мосте два буфера *ADQ*.

В соответствии с разрешенными границами транзакций области памяти, занимаемые устройствами PCI-X, также должны начинаться и заканчиваться на *ADB* — память выделяется квантами *ADQ*. Таким образом, минимальная область памяти, выделяемая устройству PCI-X, не может быть меньше 128 байт, а с учетом правил описания области ее размер может составлять 128×2^n байт.

Время выполнения транзакций, таймеры и буферы

Протокол PCI регламентирует время (число тактов), допустимое для различных фаз транзакций. Работа шины контролируется несколькими таймерами, не позволяющими попусту расходовать такты шины и помогающими планировать распределение полосы пропускания.

Каждое целевое устройство должно достаточно быстро отвечать на адресованную ему транзакцию. Ответ адресованного целевого устройства (сигнал DEVSEL#) должен появиться в 1–3 такте после фазы адреса, в зависимости от «проворности» устройства: 1 такт — быстрое (Fast), 2 — среднее (Medium), 3 — медленное (Slow)

декодирование. Следующий такт при отсутствии ответа отводится на перехват транзакции мостом с субтрактивным декодированием адреса.

Задержка первой фазы данных (target initial latency), то есть задержка появления сигнала TRDY# относительно FRAME#, не должна превышать 16 тактов шины. Если устройство по своей природе иногда может не успевать уложиться в этот интервал, оно должно формировать сигнал STOP#, прекращая транзакцию. Это заставит ведущее устройство повторить транзакцию, и с большой вероятностью эта попытка окажется успешной. Если устройство медленное и часто не укладывается в 16 тактов, то оно должно выполнять *отложенную транзакцию* (Delayed Transaction, см. выше). Целевое устройство имеет *инкрементный механизм слежения* за длительностью циклов (Incremental Latency Mechanism), который не позволяет интервалу между соседними фазами данных в пакете (target subsequent latency) превышать 8 тактов шины. Если целевое устройство не успевает работать в таком темпе, оно обязано остановить транзакцию. Желательно, чтобы устройство сообщало о своем «неуспевании» как можно раньше, не выжидая предельных 16 или 8 тактов, — это экономит полосу пропускания шины.

Инициатор также не должен задерживать поток — допустимая задержка от начала FRAME# до сигнала IRDY# (master data latency) и между фазами данных не должна превышать 8 тактов. Целевое устройство время от времени может отвергать операцию записи в память с запросом повтора (это, к примеру, может происходить при записи в видеопамять). У инициатора есть «предел терпения» для завершения операции — таймер *максимального времени исполнения* (maximum complete time). Таймер имеет порог 10 мкс — 334 такта при 33 МГц или 668 тактов на 66 МГц. За это время инициатор должен иметь возможность «протолкнуть» хоть одну фазу данных. Таймер начинает отсчет с момента запроса повтора операции записи в память и сбрасывается при последующем завершении транзакции записи в память, отличном от запроса повтора. Устройства, не способные выдерживать ограничение на максимальное время исполнения записи в память, должны предоставлять драйверу возможность определять их состояние, в котором достаточно быстрая запись в память невозможна. Драйвер, естественно, должен учитывать это состояние и не «напрягать» шину и устройство бесплодными попытками записи.

Право на управление шиной (сигнал GNT#) может быть отобрано у инициатора в любой момент времени. В зависимости от исполняемой команды и состояния сигналов ведущее устройство должно либо прервать транзакцию, либо продолжать ее до запланированного завершения. Каждое ведущее устройство, способное сформировать пакет с более чем двумя фазами данных, должно иметь собственный программируемый *таймер задержки* (`master latency timer`, или просто `latency timer`). Этот таймер фактически задает ограничение на длину пакетной транзакции и, следовательно, на пропускную способность шины, предоставляемую этому устройству. Таймер запускается каждый раз по выставлении этим устройством сигнала FRAME# и отсчитывает такты шины до достижения значения, указанного в одноименном конфигурационном регистре. Поведение ведущего устройства после срабатывания таймера зависит от типа команды и состояния сигналов FRAME# и GNT# на момент срабатывания таймера:

- если ведущее устройство снимает сигнал FRAME# до срабатывания таймера, транзакция завершается нормально;
- Если сигнал GNT# снят и исполняемая команда не является записью памяти с инвалидацией, то инициатор обязан сократить транзакцию, сняв сигнал FRAME#. При этом ему позволяется завершить текущую и выполнить еще одну фазу данных;
- если сигнал GNT# снят и исполняется запись в память с инвалидацией, то инициатор должен завершить транзакцию по концу текущей (если передается не последнее двойное слово строки) или следующей (если двойное слово — последнее) строки кэша.

Задержка арбитража (arbitration latency) определяется как число тактов от подачи инициатором запроса REQ# до получения права управления шиной GNT#. Эта задержка зависит от активности других инициаторов, быстродействия устройств (чем меньше они вводят тактов ожидания, тем лучше) и «проворности» собственно арбитра.

При конфигурировании ведущие устройства сообщают свои потребности, указывая максимально допустимую *задержку предоставления доступа к шине* (Max_Lat) и *минимальное время*, на которое им должно предоставляться *управление шиной* (Min_GNT). Эти потребности определяются присущим устройству темпом передачи данных и его организацией. Однако будут ли эти потребности реально удовлетворены (по ним должна определяться стратегия арбитража) — неясно[1]. Задержка предоставления доступа определяется как время от подачи запроса REQ# до получения GNT# и перехода шины в состояние покоя (только с этого момента данное устройство может начать транзакцию). Она зависит от числа ведущих устройств на шине, их активности и назначенных им значений таймеров задержки (в их регистрах Latency Timer, где время задается в тактах шины). Чем больше значения у этих таймеров, тем большее время придется другим устройствам ожидать предоставления управления шиной при ее значительной загрузке. Одно из слагаемых задержки предоставления доступа — задержка арбитража.

Шина позволяет уменьшить мощность (ток), потребляемую устройствами, ценой увеличения числа тактов в транзакциях, применяя *пошаговое переключение линий* AD[31:0] и PAR, — так называемый *степпинг* (address/data stepping). Здесь возможны два варианта:

- *плавный шаг* (continuous stepping) — начало формирования сигналов слаботочными формирователями за несколько тактов до выставления сигнала-квалификатора действительной информации (FRAME# в фазе адреса, IRDY# или TRDY# в фазе данных). За эти несколько тактов сигналы «доползут» до требуемого значения при меньшем токе потребления;
- *дискретный шаг* (diskrete stepping) — нормальные формирователи срабатывают не все сразу, а группами (например, побайтно), в каждом такте по группе. При этом снижаются броски тока, поскольку одновременно переключается меньше формирователей.

[1] Автору пока не удалось найти следы управления арбитражем в ОС Windows и UNIX.

Устройство само может и не пользоваться этими возможностями (см. описание бита 7 регистра команд в главе 5), но должно «понимать» такие циклы. Задерживая сигнал FRAME#, устройство рискует потерять право доступа к шине, если арбитр получит запрос от более приоритетного устройства. По этой причине PCI 2.3 степпинг отменен для всех транзакций, кроме обращений к конфигурационному пространству устройств (конфигурационные циклы типа 0). В этих циклах устройство может и не успеть в первом же такте транзакции распознать сигнал выборки IDSEL, который приходит с соответствующей линии ADx через резистор.

В *PCI-X* требования к количеству тактов ужесточились:

- инициатор не имеет права вводить такты ожидания. В транзакциях записи инициатор выставляет на шину начальные данные (*Data0*) через 2 такта после фазы атрибутов; если транзакция пакетная, то следующие (*Data1*) — через 2 такта после ответа устройства сигналом DEVSEL#. Если целевое устройство не дает готовности (сигнала TRDY#), то инициатор должен в каждом такте чередовать данные *Data0*–*Data1*, пока целевое устройство не даст готовность (ему позволительно вводить только четное число тактов ожидания);
- целевое устройство имеет право вводить такты ожидания только для начальной фазы данных транзакции; для последующих фаз данных ожидание недопустимо.

Для максимального использования возможностей шины устройства должны иметь буферы, чтобы накапливать в них данные для пакетных транзакций. Рекомендуется для устройств со скоростью передачи данных до 5 Мбайт/с иметь буфер, по крайней мере на 4 двойных слова. Для более высоких скоростей рекомендуется буфер на 32 двойных слова. Для обмена с системной памятью наиболее эффективны транзакции, работающие с целыми строками кэша, что тоже учитывают при определении размера буфера. Однако увеличение размера буфера может вызвать трудности при обработке ошибок, а также вести к увеличению задержек доставки данных (пока устройство не заполнит определенный объем буфера, оно не начнет передачу этих данных по шине, и их потребители будут ожидать).

В спецификации приводится пример организации карты Fast Ethernet (скорость передачи — 10 Мбайт/с), у которой для каждого направления передачи имеется 64-байтный буфер, разделенный на две половины (ping-pong buffer). Когда адаптер заполняет одну половину буфера приходящим кадром, он выводит в память накопленное содержимое другой половины, после чего половины меняются ролями. Каждая половина выводится в память за 8 фаз данных (около 0,25 мкс на частоте 33 МГц), что соответствует установке MIN_GNT = 1. При скорости прихода данных 10 Мбайт/с каждая половина заполняется за 3,2 мкс, что соответствует установке MAX_LAT = 12 (в регистрах MIN_GNT и MAX_LAT время задается в интервалах по 0,25 мкс).

Контроль достоверности передачи и обработка ошибок

Для контроля достоверности передачи информации на шине PCI применяется проверка четности адреса и данных; в PCI-X используется и ECC-контроль с исправлением однобитных ошибок. ECC-контроль обязателен при работе в *Mode 2*, он может использоваться и при работе в *Mode 1*. Метод контроля достоверности сообщается мостом в шаблоне инициализации по окончании аппаратного сброса шины. Мост выбирает тот метод контроля, который поддерживают все абоненты его вторичной шины (и он сам). Для сообщения об ошибках служат сигналы PERR# (протокольная сигнализация между устройствами) и SERR# (сигнал фатальной ошибки, вызывающий, как правило, немаскируемое прерывание системы).

При *контроле четности* используются сигналы PAR и PAR64, обеспечивающие четность числа «единиц» на наборах линий AD[31:0], C/BE[3:0]#, PAR и AD[63:32], C/BE[7:4]#, PAR64. Сигналы четности PAR и PAR64 вырабатываются тем устройством, которое в данный момент управляет шиной AD (выводит команду и адрес, атрибуты или данные). Сигналы четности в режиме PCI вырабатываются с задержкой на один такт относительно контролируемых ими линий AD и C/BE#. В PCI-X при операциях чтения правила немного иные: биты четности в такте N относятся к битам данных такта N – 1 и сигналам C/BE# такта N – 2. Сигналы PERR# и SERR# вырабатываются приемником информации в такте, следующем за тактом, в котором нарушено условие четности.

При *ECC-контроле* в 32-разрядном режиме для контроля линий AD[31:0] и C/BE[3:0]# применяется 7-битный код ECC с сигналами ECC[6:0]; в 64-разрядном режиме применяется 8-битный код с сигналами ECC[7:0]; в 16-разрядном режиме применяется несколько измененная схема *ECC7 + 1*. В любом из режимов ECC-контроль позволяет исправлять только одиночные ошибки и обнаруживать большинство ошибок с большей кратностью. Исправление ошибок может быть запрещено программно (через регистр управления ECC-контролем), при этом обнаруживаются все ошибки кратности 1, 2 и 3. В любом случае в регистрах ECC-контроля сохраняется диагностическая информация. Биты ECC выводятся на шину по тем же правилам и с теми же задержками, как и биты четности. Однако сигналы PERR# и SERR# вырабатываются приемником информации через такт после действительных бит ECC — «лишний» такт отдается на анализ синдрома ECC и попытку исправления ошибки.

Обнаруженная ошибка четности, как и ошибка более чем в одном бите, обнаруженная при ECC-контроле, считается *неисправимой* (unrecoverable). Достоверность информации в фазе адреса, а в PCI-X и в фазе атрибутов, проверяется целевым устройством. В случае обнаружения неисправимой ошибки в этих фазах целевое устройство подает сигнал SERR# (в течение одного такта) и устанавливает в своем регистре состояния бит 14 — `Signaled System Error`. В фазе данных достоверность проверяет устройство-приемник данных; в случае обнаружения неисправимой ошибки оно подает сигнал PERR# и устанавливает в своем регистре состояния бит 15 — `Detected Parity Error`.

В регистре состояния устройства имеется бит 8 (`Master Data Parity Error`), который отражает неудачу выполнения транзакции (последовательности) из-за обнаруженной ошибки данных. В PCI и PCI-X его правила установки формально различны:

- в PCI этот бит устанавливается только *инициатором* транзакции, когда он сам ввел (при чтении) или обнаружил (при записи) сигнал PERR#;
- в PCI-X этот бит устанавливается *запросчиком* транзакции или мостом: будучи инициатором команд чтения, мост обнаруживает ошибку в данных; будучи инициатором команд записи, мост обнаруживает сигнал PERR#; будучи целевым устройством, мост получает данные завершения с ошибкой или сообщение завершения с ошибкой транзакции записи от одного из устройств.

В случае обнаружения ошибки данных у устройства PCI-X и его драйвера есть два варианта поведения:

- не пытаясь выполнить какие-то действия по восстановлению и продолжению работы, подать сигнал SERR# — это сигнал катастрофической ошибки, который может трактоваться ОС как повод к перезагрузке. Для устройств PCI это единственно возможный вариант поведения;
- не подавать сигнал SERR#, а пытаться обработать ошибку самостоятельно. Это можно делать только программно с учетом всех возможных побочных эффектов от лишних операций (простой повтор чтения может, например, привести к потере данных).

Выбор варианта поведения обеспечивается установкой бита 0 (`Uncorrectable Data Error Recovery Enable`) в регистре `PCI-X Command`. По умолчанию (после сброса) он обнулен — по ошибке данных устанавливается сигнал SERR#; иной вариант должен выбирать драйвер, «умеющий» самостоятельно обрабатывать ошибки.

Обнаружение ошибки в фазе адреса или атрибутов всегда является фатальной ошибкой.

Инициатор (запросчик) транзакции должен иметь возможность уведомить драйвер об отвержении транзакции по условию *Master Abort* (нет ответа от целевого устройства) или *Target Abort* (отказ целевого устройства), используя прерывания или другие подходящие средства. Если такое уведомление невозможно, устройство должно подавать сигнал SERR#.

Прямой доступ к памяти, эмуляция ISA DMA (PC/PCI, DDMA)

Как было сказано выше, шина PCI не предоставляет возможности прямого доступа к памяти с использованием централизованного контроллера в стиле 8237A (как для шины ISA). Для разгрузки центрального процессора от рутинных перекачек данных предлагается прямое управление шиной со стороны устройств, называемых *ведущими устройствами шины* (PCI Bus Master). Степень интеллектуальнос-

ти ведущего устройства может быть различной. В простейшем варианте ведущее устройство обеспечивает пересылку блоков данных между устройством и системной памятью (или памятью других устройств) по указанию от CPU. Здесь CPU командами обращения к определенным регистрам ведущего устройства задает начальный адрес, длину блока, направление пересылки и разрешает запуск передачи. После этого пересылка выполняется по готовности (или инициативе) устройства, без отвлечения CPU. Таким образом выполняется *прямой доступ к памяти* (DMA). Более сложный контроллер DMA может организовывать сцепку буферов при чтении, разбросанную запись и т. п. — возможности, знакомые еще по «продвинутым» контроллерам DMA для ISA/EISA. Более интеллектуальное ведущее устройство, как правило, обладающее собственным микроконтроллером, не ограничивается такой простой работой по указке CPU — оно выполняет обмены уже по программе своего контроллера. Таким интеллектом обладают, например, хост-контроллеры последовательных шин USB и IEEE 1394, рассмотренные в данной книге.

Для совместимости устройств PCI со старым PC-ориентированным ПО и упрощения устройств PCI фирма Intel разработала специальный *протокол PC/PCI DMA*, позволяющий централизованно эмулировать стандартную (для PC) связку контроллеров DMA 8237. Альтернативное решение — *механизм DDMA* (Distributed DMA — распределенный DMA) позволяет «расчленить» стандартный контроллер и отдельные его каналы эмулировать средствами карт PCI. Оба этих механизма реализуемы только как часть моста между первичной шиной PCI и шиной ISA, поэтому их поддержка может обеспечиваться (или не обеспечиваться) только на системной плате и разрешаться в CMOS Setup.

Для поддержки *протокола PC/PCI* главный мост PCI комбинируется с контроллером DMA, программно-совместимым с парой 8237. Так, например, в хабе-контроллере ввода/вывода *ICH3* (микросхема 82801CA фирмы Intel) имеется 7-канальный контроллер DMA, у которого любой из каналов может быть подключен к протоколу PC/PCI или шине LPC. Подключением каналов к тому или иному протоколу управляет специальный 16-битный регистр (смещение 90h в конфигурационном пространстве нулевой функции устройства ICH), в котором каждому каналу отводится пара бит. Хаб ICH является и главным мостом PCI, обеспечивая, естественно, и функции арбитража для абонентов шины, включая и свои интегрированные устройства. В протоколе PC/PCI меняется назначение пары сигналов REQi# и GNTi# для заранее выбранного агента шины PCI, являющегося «проводником» DMA. Этот агент имеет внешние (по отношению к шине PCI) пары сигналов DRQx# и DACKx# с логикой, аналогичной одноименным сигналам ISA DMA, а линии REQi# и GNTi# в процессе запроса управления шиной использует особым образом (рис. 2.6). Когда агент получает запрос DRQx (один или несколько), он по линии REQi# передает в последовательном коде состояние активности линий запросов DRQx. В первом такте CLK передается старт-бит — низкий уровень REQi#, во втором — активность запроса DRQ0, затем DRQ1 и так далее до DRQ7, после чего сохраняется низкий уровень REQ#. На это сообщение хаб[1] ответит по линии GNTi#

[1] Или иной вариант реализации моста, но дальше для краткости будем ссылаться на хаб.

посылкой с указанием номера канала, которому дается подтверждение DACKx# для передачи данных в последующих транзакциях.

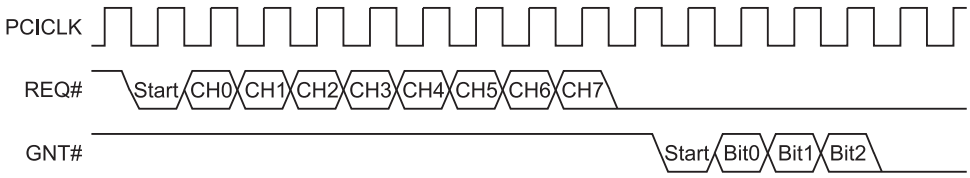


Рис. 2.6. Протокол PC/PCI

Далее хаб (как инициатор на PCI) организует передачу данных между агентом PCI DMA и памятью, при этом направлением передачи и текущим адресом в памяти управляет контроллер 8237, находящийся в том же хабе. Во время этой передачи сигнал GNTi# остается активным. Передача каждого байта или слова данных ведется не за один шинный цикл, как в ISA DMA, а за пару шинных циклов PCI: в одном цикле происходит обращение к памяти, в другом — к агенту PCI DMA. Порядок циклов определяется направлением передачи (понятно, что в первом цикле данные должны быть считаны, а во втором — записаны). В этих циклах данные передаются только по линиям AD[7:0] при работе с 8-битными каналами и по AD[15:0] при работе с 16-битными с соответствующими сигналами BE[3:0]#. В цикле обращения к памяти (команда *Memory Read* или *Memory Write*) фигурирует адрес памяти, сформированный соответствующим каналом 8237. В цикле ввода/вывода (обращения к агенту) команды *IO Read* или *IO Write* адресуются к специальным адресам:

- 00h — передача данных;
- 04h — передача данных с признаком конца цикла (сигнал TC контроллера 8237).

Кроме передачи данных протокол PC/PCI позволяет эмулировать и режим верификации DMA (без передачи данных), здесь используется команда *IO Read* с другими адресами:

- C0h — верификация данных;
- C4h — верификация данных с признаком конца цикла.

Агент должен сообщать хабу обо всех изменениях состояний линий запросов DRQx, в том числе и о снятии сигналов запроса. Если агент сигнализирует об установке более одной линии запроса, то после того, как какой-то из них будет обслужен, агент должен повторить посылку запроса для необслуженного канала. Для посылки новой информации о запросах агент на один такт снимает сигнал REQi# и снова вводит посылку запроса, начинающуюся со старт-бита. О снятии DRQx, соответствующего обслуживаемому в данный момент каналу, агент сигнализирует снятием сигнала REQi# на два такта PCI; это он должен сделать за 7 тактов до подачи им сигнала TRDY# в цикле ввода/вывода, иначе хаб начнет следующий цикл передачи.

Механизм PC/PCI DMA реализуют только в чипсете системной платы. В частности, вышеупомянутый хаб ICH3 позволяет запрограммировать на поддержку PC/PCI не более двух пар сигнальных линий REQi# и GNTi#. При этом данные линии не смогут использоваться для обычного арбитража устройств PCI. Сам агент PCI DMA тоже должен находиться на системной плате, он обеспечивает каналами DMA устройства шины ISA. Поддержку PC/PCI можно разрешать и запрещать через CMOS Setup. Через слоты PCI протокол PC/PCI, очевидно, не используется: упоминаний о механизме «объяснения» устройствам PCI, как должны использоваться их линии GNT#/REQ# (штатно или по протоколу PC/PCI), автору найти не удалось.

Пропускная способность шин PCI и PCI-X

Шина PCI является самой высокоскоростной шиной расширения современных ПК, однако и ее реальная пропускная способность, увы, не так уж и высока. Рассмотрим наиболее распространенный вариант: разрядность 32 бита, частота 33 МГц. Как указывалось выше, пиковая скорость передачи данных внутри пакетного цикла составляет 132 Мбайт/с, то есть за каждый такт шины передаются 4 байт данных ($33 \times 4 = 132$). Однако пакетные циклы выполняются далеко не всегда. Процессор общается с устройствами PCI инструкциями обращения к памяти или вводу-выводу через главный мост, который шинные транзакции процессора транслирует в транзакции шины PCI. Поскольку у процессоров x86 основные регистры 32-разрядные, то одна инструкция порождает транзакцию с устройством PCI, в которой передается не более 4 байт данных, что соответствует одиночной передаче. Если же адрес передаваемого (двойного) слова не выровнен по соответствующей границе, то будут порождены два одиночных цикла или один пакетный с двумя фазами данных, но в любом случае это обращение будет выполняться дольше, чем при выровненном адресе.

При записи массива данных в устройство PCI (передача с последовательно нарастающим адресом) мост может пытаться организовать пакетные циклы. У современных процессоров (начиная с Pentium) шина данных 64-битная и применяется буферизация записи, так что два последовательных 32-битных запроса записи объединятся в один 64-битный. Этот запрос, если он адресован к 32-битному устройству, мост попытается передать пакетом с двумя фазами данных. «Продвинутый» мост может пытаться собирать в пакет и последовательные запросы, что может породить пакет существенной длины. Пакетные циклы записи можно наблюдать, например, передавая массив данных из ОЗУ в устройство PCI строковой инструкцией MOVSD, используя префикс повтора REP. Тот же эффект даст и цикл последовательных операций LODSW, STOSW (и иных инструкций обращения к памяти). Поскольку у современных процессоров ядро исполняет инструкции гораздо быстрее, чем шина способна вывести их результаты, между инструкциями, порождающими объединяемые записи, процессор может успеть выполнить еще несколько операций. Однако если пересылка данных организуется директивой языка высокого уровня, которая ради универсальности работает гораздо сложнее вышеприведен-

ных ассемблерных примитивов, транзакции, скорее всего, будут уже одиночными (у буферов записи процессора не хватит «терпения» придержать один 32-битный запрос до появления следующего, или же произойдет принудительная выгрузка буферов записи процессора или моста по запросу чтения).

Что касается чтения из устройства PCI, то здесь пакетный режим организовать сложнее. Буферизации чтения у процессора, естественно, нет (операцию чтения можно считать выполненной лишь по получении реальных данных), и даже строковые инструкции будут порождать одиночные циклы. Однако у современных процессоров имеются возможности генерации запросов чтения более 4 байт. Для этого можно использовать инструкции загрузки данных в регистры MMX (8 байт) или XMM (16 байт), а из них уже выгружать данные в ОЗУ (которое работает много быстрее устройств PCI).

Строковые инструкции ввода/вывода (INSW, OUTSW с префиксом повторения REP), используемые для программированного ввода/вывода блоков данных (PIO), порождают серии одиночных транзакций, поскольку все данные блока относятся к одному адресу PCI.

Посмотреть, каким образом происходит обращение к устройству, несложно при наличии осциллографа: в одиночных транзакциях сигнал FRAME# активен в течение всего одного такта, в пакетных он длиннее. Число фаз данных в пакете соответствует числу тактов, во время которых активны оба сигнала IRDY# и TRDY#.

Стремиться к пакетизации транзакций записи стоит только в том случае, если устройство PCI поддерживает пакетные передачи в ведомом (target) режиме. Если это не так, то попытка пакетизации приведет даже к небольшой потере производительности, поскольку транзакция будет завершаться по инициативе ведомого устройства (сигналом STOP#), а не инициатора обмена, на чем теряется один такт шины. Так, к примеру, можно наблюдать, как при записи массива в память PCI, выполняемой директивой языка высокого уровня, устройство среднего быстродействия (вводящее лишь 3 такта ожидания готовности) принимает данные каждые 7 тактов, что при частоте 33 МГц и разрядности 32 бита дает скорость $33 \times 4/7 = 18{,}8$ Мбайт/с. Здесь 4 такта занимает активная часть транзакции (от сигнала FRAME# до снятия сигнала IRDY#) и 3 такта паузы. То же устройство по инструкции MOVSD принимает данные каждые 8 тактов шины ($33 \times 4/8 = 16{,}5$ Мбайт/с). Эти данные — результат наблюдения работы PCI-ядра, выполненного на основе микросхемы FPGA фирмы Altera, не поддерживающего пакетные транзакции в ведомом режиме. То же самое устройство при чтении памяти PCI работает существенно медленнее — инструкцией REP MOWSW с него удалось получать данные каждые 19–21 тактов шины (скорость $33 \times 4/20 = 6{,}6$ Мбайт/с). Здесь сказывается и большая задержка устройства (оно выдает данные лишь в 8 такте после появления сигнала FRAME#), и то, что процессор начинает следующую пересылку, лишь дождавшись данных от предыдущей. Трюк с использованием регистра XMM здесь дает положительный эффект, несмотря на потерю такта (на прекращение транзакции непакетным устройством), поскольку каждый 64-битный запрос процессора выполняется парой смежных транзакций PCI, между которыми пауза всего в пару тактов.

Для определения теоретического предела пропускной способности вернемся к рис. 2.1, чтобы определить минимальное время (число тактов) транзакций чтения и записи. В транзакции чтения после подачи команды и адреса инициатором (такт 1) меняется текущий «владелец» шины AD. На этот так называемый *пируэт* (turnaround) уходит такт 2, что обусловливается задержкой сигнала TRDY# целевым устройством. Далее может следовать фаза данных (такт 3), если целевое устройство достаточно расторопно. После последней фазы данных требуется еще 1 такт на обратный пируэт шины AD (в нашем случае это такт 4). Таким образом, одиночное чтение двойного слова (4 байта) занимает минимум 4 такта по 30 нс (33 МГц). Если эти транзакции следуют непосредственно друг за другом (если на такое способен инициатор и у него не отбирают право на управление шиной), то можно говорить о максимальной скорости чтения в 33 Мбайт/с при одиночных транзакциях. В транзакциях записи шиной AD все время управляет инициатор, так что здесь нет потери тактов на пируэт. При расторопном целевом устройстве, не вносящем дополнительных тактов ожидания, скорость записи может достигать 66 Мбайт/с.

Скорость, соизмеримую с максимальной пиковой, можно получить только при пакетных передачах, когда дополнительные 3 такта при чтении и 1 такт при записи добавляются не к одной фазе данных, а к их последовательности. Так, для чтения пакета с числом фаз данных 4 требуется 7 тактов ($V = 16/(7 \times 30)$ байт/нс = = 76 Мбайт/с), а для записи — 5 ($V = 16/(5 \times 30)$ байт/нс = 106,6 Мбайт/с). При 16 фазах данных скорость чтения может достигать 112 Мбайт/с, а записи — 125 Мбайт/с.

В этих выкладках не учитывались потери времени, связанные со сменой инициатора. Инициатор может начинать транзакцию по получении сигнала GNT#, только убедившись в том, что шина находится в покое (сигналы FRAME# и IRDY# пассивны); на фиксацию состояния покоя уходит 1 такт. Как видно, захватывать для одного инициатора большую часть пропускной способности шины можно, увеличивая длину пакета. Однако при этом возрастет задержка получения управления шиной для других устройств, что не всегда допустимо. Отметим также, что далеко не все устройства способны отвечать на транзакции без тактов ожидания, так что реальные цифры будут скромнее.

Итак, для выхода на максимальную производительность обмена устройства PCI сами должны быть ведущими устройствами шины, причем способными генерировать пакетные циклы. Поддержку пакетного режима имеют далеко не все устройства PCI, а у имеющих, как правило, есть существенные ограничения на максимальную длину пакета. Радикально повысить пропускную способность позволяет переход на частоту 66 МГц и разрядность 64 бита, что обходится недешево. Для того чтобы на шине могли нормально работать устройства, критичные к времени доставки данных (сетевые адаптеры, устройства, участвующие в записи и воспроизведении аудио-видеоданных и др.), не следует пытаться выжать из шины ее декларированную полосу пропускания полностью. Перегрузка шины может привести, например, к потере пакетов из-за несвоевременности доставки данных. Заметим, что адаптер Fast Ethernet (100 Мбит/с) в полудуплексном режиме занимает полосу около 13 Мбайт/с (10% декларируемой полосы обычной шины), а в полно-

дуплексном — уже 26 Мбайт/с. Адаптер Gigabit Ethernet даже в полудуплексном режиме вписывается в полосу шины уже с натяжкой (он «выживает» лишь за счет больших внутренних буферов), для него больше подходит 64 бит / 66 МГц. Существенное повышение пиковой скорости и эффективной пропускной способности дает переход на PCI-X с более высокими тактовыми частотами (PCI-X66, PCI-X100, PCI-X133) и быстрой записью в память (PCI-X266 и PCI-X533).

Говоря о пропускной способности шины и эффективной скорости обмена с устройствами PCI, следует помнить об издержках, вносимых дополнительными мостами PCI/PCI. Устройство, находящееся на дальней шине, получит меньшую пропускную способность, чем устройство, находящееся сразу за главным мостом и для которого справедливы вышеприведенные рассуждения. Это обусловлено механизмом работы моста — транзакции через мост выполняются в несколько этапов (см. главу 4).

ГЛАВА 3

Прерывания PCI: INTx#, PME#, MSI и SERR#

Устройства PCI имеют возможность сигнализации об асинхронных событиях с помощью прерываний. На шине PCI возможны четыре типа сигнализации прерываний:

- традиционная проводная сигнализация по линиям INTx;
- проводная сигнализация событий управления энергопотреблением по линии PME#;
- сигнализация с помощью сообщений — *MSI*;
- сигнализация фатальной ошибки по линии SERR#.

В данной главе рассматриваются все эти типы сигнализации, а также общая картина поддержки аппаратных прерываний в PC-совместимых компьютерах.

Аппаратные прерывания в PC-совместимых компьютерах

Аппаратные прерывания обеспечивают реакцию процессора на события, происходящие асинхронно по отношению к исполняемому программному коду. Прерывания в процессорах x86, используемых в PC-совместимых компьютерах, подробно рассмотрены в литературе [2]. Напомним, что аппаратные прерывания делятся на маскируемые и немаскируемые. Процессор x86 по сигналу прерывания приостанавливает выполнение текущего потока инструкций, сохраняя в стеке состояние (флаги и адрес возврата), и выполняет процедуру обработки прерывания. Конкретная процедура обработки выбирается из таблицы прерываний по *вектору прерывания* — однобайтному номеру элемента в данной таблице. Вектор прерывания доводится до процессора разными способами: для немаскируемого прерывания он фиксирован, для маскируемых прерываний его сообщает специальный *контроллер прерываний*. Кроме аппаратных прерываний у процессоров x86 имеются также *внутренние прерывания — исключения* (exceptions), связанные с особыми случаями выполнения инструкций, и *программные прерывания*. Для исключений вектор

определяется самим особым условием, и под исключения фирмой Intel зарезервированы первые 32 вектора (0–31 или 00–1Fh). В программных прерываниях номер вектора содержится в самой инструкции (программные прерывания — это лишь специфический способ вызова процедур по номеру, с предварительным сохранением в стеке регистра флагов). Все эти прерывания используют один и тот же набор из 256 возможных векторов. Исторически сложилось так, что векторы, используемые для аппаратных прерываний, пересекаются с векторами исключений и векторами для программных прерываний, используемых для вызовов сервисов BIOS и DOS. Таким образом, для ряда номеров векторов процедура, на которую ссылается таблица прерываний, должна в начале содержать программный код, определяющий, по какому поводу она вызвана: из-за исключения, аппаратного прерывания или же для вызова какого-то системного сервиса. Таким образом, процедура, собственно и обеспечивающая реакцию процессора на то самое асинхронное событие, будет вызвана только после ряда действий по идентификации источника прерываний. Здесь еще заметим, что один и тот же вектор прерывания может использоваться и несколькими периферийными устройствами — это так называемое *разделяемое использование прерываний*, которое подробно обсуждается ниже.

Вызов процедуры обслуживания прерываний в реальном и защищенном режимах процессора существенно различается:

- в реальном режиме таблица прерываний содержит 4-байтные *дальние указатели* (сегмент и смещение) на соответствующие процедуры, которые вызываются дальним вызовом (`Call Far` с предварительным сохранением флагов). Размер (256 × 4 байт) и положение таблицы (начинается с адреса 0) фиксированы;
- в защищенном режиме (и в его частном случае — режиме V86) таблица содержит 8-байтные *дескрипторы прерываний*, которые могут быть шлюзами прерываний (`Interrupt Gate`), ловушек (`Trap Gate`) или задач (`Task Gate`). Размер таблицы может быть уменьшен (максимальный — 256 × 8 байт), положение таблицы может меняться (определяется содержимым регистра `IDT` процессора). Код обработчика прерываний должен быть не менее привилегированным, чем код прерываемой задачи (иначе сработает исключение защиты). По этой причине обработчики прерываний должны работать на уровне ядра ОС (на нулевом уровне привилегий). Смена уровня привилегии при вызове обработчика приводит к дополнительным затратам времени на переопределение стека. Прерывания, вызывающие переключение задач (через `Task Gate`), расходуют значительное время на переключение контекста — выгрузку регистров процессора в сегмент состояния старой задачи и их загрузку из сегмента состояния новой.

Номера векторов, используемых для аппаратных прерываний в операционных системах защищенного режима, отличаются от номеров, используемых в ОС реального режима, чтобы исключить их конфликты с векторами, используемыми для исключений процессора.

На *немаскируемое прерывание* (NMI — Non-Maskable Interrrupt) процессор реагирует всегда (если обслуживание предыдущего NMI завершено); этому прерыва-

нию соответствует фиксированный вектор 2. Немаскируемые прерывания в PC используются для сигнализации о фатальных аппаратных ошибках. Сигнал на линию NMI приходит от схем контроля памяти (четности или ECC), от линий контроля шины ISA (IOCHK) и шины PCI (SERR#). Сигнал NMI блокируется до входа процессора установкой в 1 бита 7 порта 070h, отдельные источники разрешаются и идентифицируются битами порта 061h:

- бит 2 R/W — ERP — разрешение контроля ОЗУ и сигнала SERR# шины PCI;
- бит 3 R/W — EIC — разрешение контроля шины ISA;
- бит 6 R — IOCHK — ошибка контроля на шине ISA (сигнал IOCHK#);
- бит 7 R — PCK — ошибка четности ОЗУ или сигнал SERR# на шине PCI.

Реакция процессора на *маскируемые прерывания* может быть задержана сбросом его внутреннего флага IF (инструкция CLI запрещает прерывания, STI — разрешает). Маскируемые прерывания используются для сигнализации о событиях в устройствах. По возникновении события, требующего реакции, адаптер (контроллер) устройства формирует *запрос прерывания*, который поступает на вход *контроллера прерываний*. Задача контроллера прерываний — довести до процессора запрос прерывания и сообщить вектор, по которому выбирается программная процедура обработки прерываний.

Процедура обработки прерывания от устройства должна выполнить действия по обслуживанию данного устройства, включая сброс его запроса для обеспечения возможности реакции на следующие события, и послать команды завершения в контроллер прерываний. Вызывая процедуру обработки, процессор автоматически сохраняет в стеке значение всех флагов и сбрасывает флаг IF, что запрещает маскируемые прерывания. При возврате из этой процедуры (по инструкции IRET) процессор восстанавливает сохраненные флаги, в том числе и установленный (до прерывания) IF, что снова разрешает прерывания. Если во время работы обработчика прерываний требуется реакция на иные прерывания (более приоритетные), то в обработчике должна присутствовать инструкция STI. Особенно это касается длинных обработчиков; здесь инструкция STI должна вводиться как можно раньше, сразу после критической (не допускающей прерываний) секции. Следующие прерывания того же или более низкого уровня приоритета контроллер прерываний будет обслуживать только после получения команды завершения прерывания EOI (End Of Interrupt).

В IBM PC-совместимых компьютерах применяется два основных типа контроллеров прерываний:

- *PIC* (Peripheral Interrupt Controller) — периферийный контроллер прерываний, программно совместимый с «историческим» контроллером 8259A, применявшимся еще в первых моделях IBM PC. Со времен IBM PC/AT применяется связка из пары каскадно соединенных PIC, позволяющая обслуживать до 15 линий запросов прерываний;
- *APIC* (Advanced Peripheral Interrupt Controller) — усовершенствованный периферийный контроллер прерываний, введенный для поддержки мультипроцес-

сорных систем в компьютеры на базе процессоров 4–5 поколений (486 и Pentium) и используемый поныне для более поздних моделей процессоров. Кроме поддержки мультипроцессорных конфигураций современный APIC позволяет увеличивать число доступных линий прерываний и обрабатывать запросы прерываний от устройств PCI, посылаемые через механизм сообщений (MSI). Компьютер, оснащенный контроллером APIC, обязательно имеет возможность функционировать и в режиме, совместимом со стандартной связкой пары PIC. Этот режим включается по аппаратному сбросу (и включению питания), что позволяет использовать старые ОС и приложения MS DOS, «не знающие» APIC и мультипроцессирования.

Традиционная схема формирования запросов прерываний с использованием пары PIC изображена на рис. 3.1.

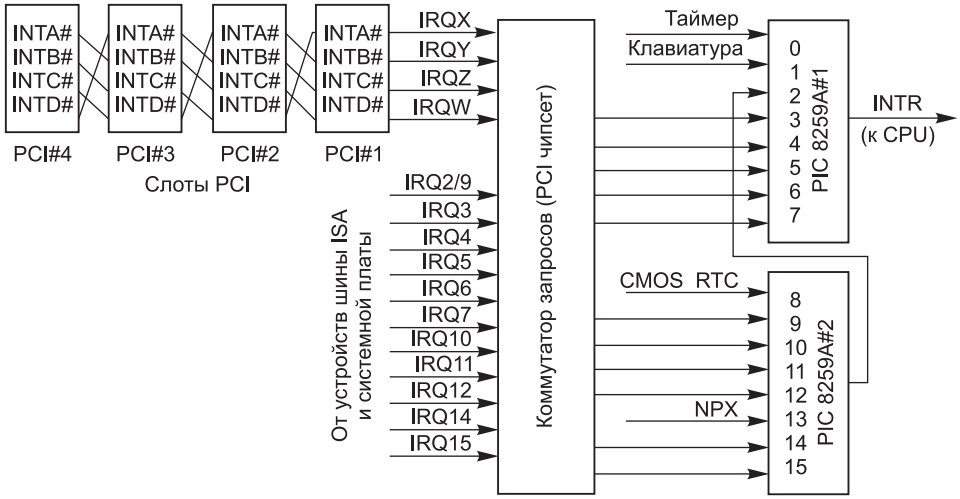


Рис. 3.1. Коммутация запросов прерываний

На входы контроллеров прерываний поступают запросы от системных устройств (клавиатура, системный таймер, CMOS-таймер, сопроцессор), периферийных контроллеров системной платы и карт расширения. Традиционно все линии запросов, не занятые перечисленными устройствами, присутствуют на всех слотах шины ISA/EISA. Эти линии обозначаются как `IRQx` и имеют общепринятое назначение (табл. 3.1). Часть этих линий отдается в распоряжение шины PCI. В таблице отражены и приоритеты прерываний — запросы расположены в порядке их убывания. Номера векторов, соответствующих линиям запросов контроллеров, система приоритетов и некоторые другие параметры задаются программно при инициализации контроллеров. Эти основные настройки остаются традиционными для обеспечения совместимости с программным обеспечением, но различаются для ОС реального и защищенного режимов. Так, например, в ОС Windows базовые векторы для ведущего и ведомого контроллеров — 50h и 58h соответственно.

Таблица 3.1. Аппаратные прерывания (в порядке убывания приоритета)

Имя (номер[1])	Вектор[2]	Вектор[3]	Контроллер/маска	Описание
NMI	02h			Контроль канала, четность памяти (в XT — сопроцессор)
IRQ0	08h	50h	#1/1h	Таймер (канал 0 8253/8254)
IRQ1	09h	51h	#1/2h	Клавиатура
IRQ2	0Ah	52h	#1/4h	XT — резерв, AT — недоступно (подключается каскад IRQ8–IRQ15)
IRQ8	70h	58h	#2/1h	CMOS RTC — часы реального времени
IRQ9	71h	59h	#2/2h	Резерв
IRQ10	72h	5Ah	#2/4h	Резерв
IRQ11	73h	5Bh	#2/8h	Резерв
IRQ12	74h	5Ch	#2/10h	PS/2-Mouse (резерв)
IRQ13	75h	5Dh	#2/20h	Математический сопроцессор
IRQ14	76h	5Eh	#2/40h	HDC — контроллер НЖМД
IRQ15	77h	5Fh	#2/80h	Резерв
IRQ3	0Bh	52h	#1/4h	COM2, COM4
IRQ4	0Ch	53h	#1/10h	COM1, COM3
IRQ5	0Dh	54h	#1/20h	XT — HDC, AT — LPT2, Sound (резерв)
IRQ6	0Eh	55h	#1/40h	FDC — контроллер НГМД
IRQ7	0Fh	56h	#1/80h	LPT1 — принтер

[1] Запросы прерываний 0, 1, 8 и 13 на шины расширения не выводятся.

[2] Указаны номера векторов при работе в реальном режиме процессора.

[3] Указаны номера векторов при работе в ОС Windows.

Каждому устройству, для поддержки работы которого требуются прерывания, должен быть назначен свой номер прерывания. Назначения номеров прерываний выполняются с двух сторон: во-первых, адаптер, нуждающийся в прерываниях, должен быть сконфигурирован на использование конкретной линии шины (джамперами или программно). Во-вторых, программное обеспечение, поддерживающее данный адаптер, должно быть проинформировано о номере используемого вектора. В процессе назначения прерываний может участвовать система PnP для шин ISA и PCI, для распределения линий запросов между шинами служат специальные параметры CMOS Setup. Современные ОС имеют возможность изменить назначение запросов относительно распределения, сделанного через CMOS Setup.

После того как произведено конфигурирование системы прерываний (проинициализирован контроллер прерываний, устройствам назначены линии запросов

и установлены указатели на процедуры обработки), отработка маскируемых аппаратных прерываний происходит следующим образом:

- устройство по событию прерывания возбуждает назначенную ему линию запроса прерывания;
- контроллер принимает *сигналы запросов от источников прерываний* (сигналы IRQx) и при наличии незамаскированного запроса подает сигнал *общего запроса прерывания* (сигнал INTR) процессору x86;
- процессор, реагируя на запрос (когда прерывания флагом IF разрешены), сохраняет в стеке содержимое регистра флагов и адрес возврата, после чего формирует *шинный цикл INTA* (Interrupt Acknowledge, подтверждение прерывания), который доводится до контроллера прерываний;
- в момент получения сигнала INTA контроллер прерываний фиксирует состояние своих входов запросов — к этому моменту их состояние могло измениться: могли появиться новые запросы или пропасть запрос от «нетерпеливого» устройства. Контроллер анализирует поступившие запросы в соответствии с запрограммированной схемой приоритетов и посылает процессору *вектор прерывания*, соответствующий самому приоритетному незамаскированному запросу, присутствующему на входе контроллера в момент подачи шинной команды *INTA*. При этом контроллер выполняет и некоторые действия в соответствии с установленной приоритетной политикой, учитывающие, какой именно вектор был послан (какой из запросов пошел на обслуживание);
- получив вектор прерывания, процессор по его номеру вызывает соответствующую процедуру обработки прерывания. Если данный вектор прерывания используется не только для аппаратных прерываний, но и для исключений и/или программных прерываний, то процедура в первую очередь должна определить, к какому из этих типов относится данное событие. Для этого процедура может обратиться к контроллеру PIC (прочитать регистр ISR) и проанализировать состояние регистров процессора. Дальнейшие шаги рассматриваются для случая, когда обнаружено аппаратное прерывание;
- процедура обработки прерывания должна идентифицировать источник прерывания — определить устройство, его вызвавшее. В случае разделяемого использования несколькими устройствами данного номера запроса (следовательно, и вектора) идентифицировать источник прерывания можно только последовательными обращениями к регистрам каждого из этих устройств. При этом следует учитывать возможность поступления запросов от нескольких устройств одновременно или в процессе обработки прерывания от одного из них;
- процедура должна обслужить устройство-источник прерывания — выполнить «полезные» действия, связанные с событием, о котором и сигнализировало устройство. Это обслуживание должно обеспечить и *снятие сигнала запроса прерывания* от данного устройства. В случае разделяемых прерываний источников может быть и несколько, и все они требуют обслуживания;
- если обработка прерывания занимает значительное время, в течение которого требуется реакция системы на более приоритетные запросы, то после критиче-

ской секции в обработчик включают инструкцию `STI`, устанавливающую флаг разрешения прерываний (`IF`) в процессоре. С этого момента возможны *вложенные прерывания*, прерывающие работу данного обработчика другой, более приоритетной процедурой;

- процедура обработки прерывания должна послать контроллеру *команду завершения обработки прерывания EOI* (End Of Interrupt), по которой контроллер разрешит последующий прием сигнала с обслуженного входа и менее приоритетных. Это должно быть сделано после снятия сигнала прерывания от обслуженных устройств, иначе контроллер после *EOI* пошлет повторный запрос. Обработчик прерывания, для которого запрос поступил от ведомого контроллера, должен послать *EOI* как ведомому, так и ведущему контроллеру. Участок обработчика, начинающийся от подачи команды *EOI* до завершения (инструкции `IRET`), должен быть *непрерываемым*, то есть он является *критической секцией*. Если обработчик разрешал вложенные прерывания, то перед подачей команды *EOI* должна присутствовать инструкция `CLI`, запрещающая прерывания;
- завершается обработка прерывания инструкцией `IRET`, по которой процессор возвращается к выполнению прерванного потока инструкций, предварительно извлекая из стека содержимое регистра флагов. При этом аппаратные прерывания снова окажутся разрешенными.

Эта последовательность описана применительно к обычному контроллеру прерываний (PIC), в системах с APIC меняется способ доставки вектора прерывания от контроллера к процессору, а в прерываниях MSI меняется способ доставки сигнала от устройства к контроллеру APIC. Эти нюансы описаны в последующих разделах.

Традиционный контроллер прерываний — PIC

Контроллер прерываний (PIC) 8259A является периферийным устройством, которое связано с процессором через системную шину. По этой шине процессор может обращаться к регистрам контроллера, программируя его режимы и управляя им, а также получать от контроллера 8-битный *вектор прерывания*, для чего в интерфейсе системной шины процессора имеется специальная команда подтверждения прерывания (*INTA*). Контроллер 8259A имеет 8 входов запросов от источников и один выход общего запроса. Каждому из входов соответствует свой вектор; программированием регистров контроллера задается номер вектора для входа 0, остальным входам соответствуют последующие номера векторов. Каждый вход может быть программно замаскирован — тогда он не будет вызывать сигнал общего запроса. Контроллер занимает два адреса в пространстве ввода/вывода, программное обращение по этим адресам обеспечивает выполнение следующих функций:

- управление режимами работы контроллера;
- управление приоритетами и масками запросов;
- идентификацию запросов прерывания без вырабатывания общего запроса — обслуживание прерываний по опросу (*полинг*).

В контроллере имеется три регистра: IRR, IMR и ISR. С каждым входом запроса в контроллере связано по одному биту каждого из этих регистров; бит 0 каждого из этих регистров относится к входу 0, бит 1 — к входу 1, бит 7 — ко входу 7. Регистры имеют следующие назначения:

- IRR (Interrupt Request Register) — регистр запросов прерываний, в котором бит устанавливается при обнаружении сигнала прерывания на соответствующем входе, независимо от маски;
- IMR (Interrupt Mask Register) — регистр масок прерываний. Единичное значение бита означает замаскированность данного входа — по запросу с замаскированного входа общий запрос прерывания не генерируется;
- ISR (Interrupt Service Register) — регистр обслуживаемого прерывания. Во время цикла подтверждения (*INTA*) в регистре устанавливается бит, соответствующий наиболее приоритетному запросу и, соответственно, формируемому вектору прерывания. В этот же момент обнуляется соответствующий бит запроса в регистре IRR. Бит в ISR обнуляется по получении контроллером команды *EOI*, а в режиме автоматического подтверждения (по команде *AEOI*) он обнуляется сразу после передачи вектора прерывания.

Контроллер прерываний позволяет программировать свои входы на чувствительность к уровню или перепаду сигнала:

- *чувствительность к уровню* (*level sensitive*) означает, что контроллер прерываний вырабатывает запрос прерывания процессора по факту обнаружения определенного уровня на входе IRQx. Если к моменту завершения обработки этого запроса (после записи команды *EOI* в регистр контроллера прерываний) контроллер снова обнаруживает активный уровень на том же входе DRQx, то он снова сформирует запрос на прерывание процессора;
- *чувствительность к перепаду* (*edge sensitive*) означает, что контроллер прерываний вырабатывает запрос прерывания процессора только по факту обнаружения перепада (на ISA — положительного) на входе IRQx. Повторно запрос по этому входу возможен только по следующему такому же перепаду, то есть сигнал предварительно должен вернуться в исходное состояние.

В любом случае сигнал запроса аппаратного прерывания IRQx должен удерживаться генерирующей его схемой, по крайней мере, до цикла подтверждения прерывания процессором — именно в этот момент PIC определяет самый приоритетный незамаскированный запрос и по нему формирует вектор. Если к этому моменту запрос будет снят, источник прерывания корректно идентифицирован не будет и контроллер сообщит *ложный вектор прерывания* (spurious interrupt), соответствующий его входу с максимальным номером (IRQ7 для первого контроллера и IRQ15 для второго). Обычно периферийные устройства строят так, что сигнал запроса сбрасывается при обращении программы обслуживания прерывания к соответствующим регистрам адаптера, так что ложных прерываний возникать не должно.

Стандартный PIC позволяет управлять чувствительностью только для всех входов одновременно. В шине ISA и системной периферии (таймеры, контроллер клавиатуры) прерывание сигнализируется *положительным перепадом сигнала* на

линии запроса, так что традиционно контроллеры PIC программируют на чувствительность к перепаду. Более поздние модификации PIC, применяемые в системах с шиной EISA и ISA с поддержкой PnP, а также новые контроллеры APIC при работе в PIC-совместимом режиме позволяют управлять чувствительностью для каждого входа индивидуально.

Один PIC 8259A позволяет обслуживать 8 запросов прерываний; в PC/AT применяется *каскадное соединение двух контроллеров*, один из которых является ведущим, другой — ведомым. *Ведущий контроллер 8259A#1* обслуживает запросы 0, 1, 3–7; его выход подключается к входу запроса прерываний процессора. К его входу 2 подключен *ведомый контроллер 8259A#2*, который обслуживает запросы 8–15. При этом поддерживается вложенность приоритетов — запросы 8–15 со своим рядом убывающих приоритетов вклиниваются между запросами 1 и 3 ведущего контроллера, приоритеты запросов которого также убывают с ростом номера. В XT каскадирование не применялось и один контроллер 8259A обслуживал все 8 линий запросов.

Контроллер 8259A позволяет работать с запросами в различных режимах:

- *Fully Nested Mode* — режим полной вложенности приоритетов; каждому входу (уровню) запросов назначается свой приоритет (самый приоритетный — вход 0). В момент подтверждения прерывания контроллер устанавливает в регистре `ISR` бит, соответствующий самому приоритетному запросу на данный момент (и переданному вектору прерывания), и до его сброса игнорирует последующие запросы с данного входа и менее приоритетные запросы;
- *Special Fully Nested Mode* — специальный режим полной вложенности, используемый в ведущем контроллере при каскадном соединении. В этом режиме ведущий контроллер не блокирует запрос от входа, к которому подключен ведомый контроллер. Это позволяет ведомому контроллеру сигнализировать о запросе, более приоритетном, чем предыдущий. В конце процедуры обработки ведомому контроллеру посылается *неспецифический EOI*, после чего считывается его `ISR`. Если в `ISR` ни один бит не установлен, то неспецифический *EOI* посылается и ведущему контроллеру, что позволит ему обслуживать и менее приоритетные входы. Если же в `ISR` ведомого контроллера есть ненулевые биты, то ведущий контроллер снова подаст общий запрос прерывания, и на его подтверждение ведомый пошлет соответствующий вектор. Упоминаний о проверке на 0 значения `ISR` вторичного контроллера до подачи *EOI* первичному применительно к PC автору не встречалось. Очевидно, что в ней нет необходимости, если не используется ротация приоритетов: если у вторичного контроллера есть еще не обслуженные запросы, то они будут обслужены в соответствии со своим приоритетом (до `IRQ3`...`IRQ7`);
- *Automatic Rotation Mode* — режим автоматической ротации приоритетов позволяет организовать равноприоритетное обслуживание всех запросов. В этом режиме уровень, запрос от которого пошел на обслуживание, получает низший приоритет. Ротация приоритетов организуется подачей команд *OCW2* с кодом операции 101 или 100 (см. далее);

- *Specific Rotation Mode* — режим специфицированной ротации: командой *OCW2* с кодом операции 111 или 110 указанному уровню устанавливается низший приоритет;
- *Poll Mode* — режим опроса, в котором общий запрос INTR не вырабатывается. По команде *Poll*, посылаемой через *OCW3*, контроллер фиксирует самый проритетный запрос. Последующее считывание из регистра контроллера даст байт, в котором бит 7 указывает на наличие запроса, а в битах [2:0] содержится номер самого приоритетного запроса;
- *Normal EOI* — нормальный режим завершения, в котором бит ISR сбрасывается явной командой *EOI*, посылаемой контроллеру в конце исполнения обработчика прерывания. Обычно используется команда *неспецифического EOI* (код 20h), по которой сбрасывается бит ISR, соответствующий самому приоритетному из обслуживаемых запросов. Возможна команда и *специфического EOI*, которая сбрасывает в ISR бит, указанный в данной команде. Бит для замаскированного запроса таким способом сброшен быть не может;
- *Auto EOI* — автоматическая генерация *неспецифического EOI* контроллером в конце цикла подтверждения прерывания. Этот режим применим лишь в случаях, когда не требуется поддержка вложенности прерываний, и только для ведущего контроллера.

Контроллер 8259A своими 8-битными регистрами приписывается к пространству ввода/вывода и занимает 2 смежных адреса. Обмен с регистрами контроллеров должен производиться только однобайтными операциями ввода/вывода. В современных PC-совместимых компьютерах контроллеры имеют и дополнительные регистры (ELCR). Положение регистров в пространстве ввода/вывода приведено в табл. 3.2.

Таблица 3.2. Регистры контроллеров прерываний

Адрес 8259A#1	8259A#2	Назначение	Тип
020h	0A0h	Подача команд *ICW1*, *OCW2*, *OCW3*	WO
		Чтение регистров IRR, ISR или данных полинга (в зависимости от *OCW3*)	RO
021h	0A1h	Подача команд *ICW2*, *ICW3*, *ICW4*, *OCW1*	WO
		Обращение к регистру IMR	RW
4D0h	4D1h	Обращение к регистру ELCR	RW

Контроллер имеет два режима работы: режим инициализации и операционный. После сброса контроллер инициализируется последовательностью команд *ICW1–ICW4* (Initialization Command Words) длиной до 4 байт, после чего переходит в операционный режим. В *операционном режиме* (*Operation Mode*)контроллер воспринимает *команды управления OCW1–OCW3* (Operation Control Words). В операционном режиме он может быть и реинициализирован, признаком начала ини-

циализации является единичное значение бита 4 в байте, записываемом по адресу 020h (0A0h). Назначение команд и регистров контроллера прерываний приведено ниже

Команда инициализации ICW1 (запись по адресу 020h или 0A0h) служит для конфигурирования контроллера:

- биты [7:5]: 0 (в PC не используются);
- бит 4:1 — признак команды инициализации (в командах управления он нулевой);
- бит 3 — чувствительность линий запроса: 0 — прерывание по перепаду, 1 — прерывание по уровню. В современных контроллерах игнорируется, в них используется отдельный регистр ELCR, управляющий чувствительностью каждого из входов;
- бит 2 = 0 (в PC не используется);
- бит 1:0 — каскадное включение пары контроллеров, 1 — одиночный контроллер;
- бит 0:1 — признак использования *ICW4*.

Команда инициализации ICW2 (запись по адресу 021h или 0A1h) задает номер вектора, генерируемого данным контроллером для входа 0. Биты [2:0] должны иметь нулевые значения.

Команда инициализации ICW3 (запись по адресу 021h или 0A1h) используется только для каскадного включения:

- для *ведущего контроллера* (*Master*, 8259A#1) биты [0:7] указывают на наличие ведомых контроллеров на линиях IRQ0–IRQ7 соответственно. В AT *ICW3* = 04h (ведомый на IRQ2);
- для *ведомого контроллера* (*Slave*, 8259A#2) биты [2:0] содержат номер входа ведущего контроллера, к которому он подключен, биты [7:3] сброшены. В AT *ICW3* = 02h (подключен к IRQ2 ведущего).

Команда инициализации ICW4 (запись по адресу 021h или 0A1h) задает режим работы контроллера, его применение обязательно при начальной инициализации:

- биты [7:5]: 0 — не используются;
- бит 4 — SFNM (Special Fully Nested Mode): 1 — специальный режим полной вложенности разрешен, 0 — запрещен;
- бит 3 — BUF (Buffered): 1 — признак буферизованности шины;
- бит 2 — M/S (Master/Slave), положение в каскаде: 1 — ведущий, 0 — ведомый. В современных контроллерах не используется (сброшен);
- бит 1 — AEOI (Auto End Of Interrupt): 1 — разрешение автоматического завершения прерывания, 0 — нормальный режим (требует подачи *EOI*);
- бит 0 — тип используемого процессора: 1 — 8086/8088 и далее, 0 — 8080.

Команда управления OCW1 (запись по адресу 021h или 0A1h) задает маски запросов, единичное значение бита означает маскирование запроса. Биты [0:7]определяют маски запросов для IRQ0–IRQ7 (8259#1) или IRQ8–IRQ15 (8259#2).

Команда управления OCW2 (запись по адресу 020h или 0A0h) — завершение обслуживания прерывания (команда *EOI*), управление приоритетом:

- биты [7:5] задают код операции (в операциях, помеченных звездочкой, используется поле LLL в битах 2–0):
  - 001 — *неспецифический EOI*;
  - 011* — *специфический EOI* для запроса LLL;
  - 101 — *неспецифический EOI* с ротацией приоритета;
  - 100 — установка ротации приоритета в режиме *AEOI*;
  - 000 — сброс ротации приоритета в режиме *AEOI*;
  - 111* — *специфический EOI* с ротацией приоритета (установкой низшего приоритета для заданного уровня);
  - 110* — установка низшего приоритета для заданного уровня;
  - 010 — нет операции.
- биты [4:3]: 00 — признак *OCW2*;
- биты [2:0] — поле LLL — номер уровня, к которому относится команда (только для команд, помеченных звездочкой).

Команда управления OCW3 (запись по адресу 020h или 0A0h) — оперативное управление контроллером:

- бит 7:0 — не используется;
- биты [6:5] — режим специального маскирования (в PC не используется): 11 — установить, 10 — сбросить, 00, 01 — не изменять;
- биты [4:3]: 01 — признак *OCW3*;
- бит 2 — признак команды опроса (полинга). После команды полинга на последующую команду чтения порта 020h или 0A0h контроллер ответит байтом, кодирующим запрос прерывания с максимальным приоритетом. Для PC полинг обычно не используется (бит 2 — нулевой), а контроллер передает вектор прерывания по команде *INTA*;
- биты [1:0] — управление чтением регистров при операциях ввода по адресу 020h или 0A0h:
  - 10 — чтение IRR — регистра запросов;
  - 11 — чтение ISR — регистра обслуживаемого прерывания;
  - 00, 01 — не изменять выбор регистра.

Регистры ELCR, имеющиеся в современных компьютерах, позволяют селективно управлять чувствительностью входов. В этих регистрах каждый бит отвечает за режим своего входа запроса: 0 — чувствительность к положительному перепаду, 1 — чувствительность к высокому уровню. Для входов IRQ0, 1, 2, 8 и 13 (таймер, клавиатура, вторичный контроллер прерываний, часы и исключение сопроцессора), допускается чувствительность только к перепаду (соответствующие биты должны быть нулевыми, но чипсет может их и игнорировать). Линии запросов прерывания от PCI по пути ко входам инвертируются, так как на них запрос сигнализируется низким уровнем.

В IBM PC/XT/AT используется специальный режим вложенных прерываний с фиксированным приоритетом и автоматическим неспецифическим завершением; типовые байты инициализации и управления приведены в табл. 3.3. После инициализации (процедурой POST и при загрузке ОС) все неиспользуемые входы контроллеров замаскированы (на запросы прерываний реагировать не будут), а их векторы прерываний указывают на «заглушку» — процедуру с единственной инструкцией `IRET`. Для подключения обработчика прерывания от устройства первым делом следует загрузить обработчик в память и установить указатель на него в таблице прерываний. Далее следует размаскировать соответствующий ему вход в контроллере прерываний, для чего выполняется чтение регистра маски (адрес 21h для 8259A#1, A1h для 8259A#2), обнуление соответствующего бита (см. табл. 3.1) и запись в регистр нового значения маски. Если обработчик прерывания удаляется из памяти, предварительно должен быть замаскирован соответствующий ему вход контроллера. Все изменения в таблице прерываний должны выполняться при замаскированных прерываниях, чтобы избежать попытки использования вектора в процессе его модификации (это приведет к «вылету» программы — вызову по некорректному адресу).

Каждая процедура обработки аппаратного прерывания должна завершаться командой *неспецифического EOI* — посылкой *OCW2* = 20h контроллеру:

- для запросов от ведущего контроллера — посылка байта 20h по адресу 20h;
- для запросов от ведомого контроллера — посылка байта 20h по адресу A0h (*EOI* для ведомого контроллера); затем посылка байта 20h по адресу 20h (*EOI* для ведущего контроллера).

Некорректно завершенная процедура не позволит повторно использовать данный или другие запросы прерываний. Заметим, что команды *EOI*, как и инструкция `IRET`, в современных ОС включены в общий обработчик прерывания. В прикладных обработчиках (в драйверах устройств) этих команд быть не должно, иначе ОС не сможет собрать цепочку обработчиков разделяемого прерывания. В среде MS-DOS команды *EOI* и инструкцию `IRET` должен подавать прикладной обработчик.

Таблица 3.3. Байты инициализации контроллеров прерываний

Байт	8259A#1	8259A#2
ICW1	10h	10h
ICW2	08h	70h
ICW3	04h	02h
ICW4	1Fh	1Bh
OCW3	0Ah	0Ah

На современных системных платах функции контроллеров прерываний возлагаются на чипсет, который может иметь и более гибкие возможности управления, чем пара контроллеров 8259A. Процедура инициализации контроллеров может и отличаться от традиционной, но ею занимается тест POST, который «знает» осо-

бенности системной платы. Однако в операционном режиме всегда сохраняется программная совместимость с 8259A.

«Продвинутый» контроллер прерываний — APIC

Контроллер APIC в первую очередь предназначен для симметричных мультипроцесорных систем (SMP), описанных в документе Intel «MultiProcessor Specification» (MPS), в Сети доступна версия 1.4, 1997 год. Здесь симметрия рассматривается в двух аспектах:

- симметрия памяти — все процессоры пользуются общей памятью, работают с одной копией ОС;
- симметрия ввода/вывода — все процессоры разделяют общие устройства ввода/вывода и общие контроллеры прерываний.

Система может быть симметричной по памяти, но асимметричной по прерываниям от ввода/вывода, если для них используется выделенный процессор. В x86 симметрию по прерываниям обеспечивает APIC. Система с APIC состоит из локальных контроллеров, установленных в процессорах, и контроллеров прерываний от ввода/вывода (одного или нескольких). Все контроллеры APIC соединены между собой локальной шиной, по которой они обмениваются друг с другом сообщениями. Задача каждого *локального контроллера* (Local APIC) — трансляция сообщений, принятых по локальной шине, в сигналы, вызывающие все аппаратные прерывания своего процессора — маскируемые (`INTR`), немаскируемые (`NMI`) и прерывания системного обслуживания (`SMI`). Кроме того, локальные APIC позволяют каждому процессору генерировать прерывания для других процессоров. Локальный контроллер имеет внутренний интервальный таймер, позволяющий вырабатывать прерывания через программируемый интервал времени. *Контроллер прерываний от ввода/вывода* (I/O APIC) преобразует запросы аппаратных прерываний от устройств в сообщения протокола локальной шины APIC. В мультипроцессорном режиме он отвечает за распределение прерываний по процессорам, для чего может использоваться статическое или динамическое распределение. В случае статического распределения для каждого номера прерывания указывается номер процессора, который его обслуживает. В случае динамического распределения каждое прерывание направляется наименее приоритетному в данный момент процессору. Этот же контроллер отвечает за распространение сигналов о системных событиях (`NMI`, `INIT`, `SMI`) и межпроцессорных прерываний. Прерывания в мультипроцессорных системах подробно рассмотрены в документе «Intel Architecture Software Developer's Manual Volume 3: System Programming Guide», доступном на сайте `http://www.intel.com`. Здесь же ограничимся описаниями возможностей, предоставляемыми для сигнализации прерываний ввода/вывода контроллерами APIC.

Контроллер I/O APIC является частью чипсета системной платы, например, он входит в хабы ICH2 и ICH3 чипсетов Intel. В спецификации MPS определено три режима обработки прерываний:

- *режим PIC* (PIC Mode) — эмуляция пары PIC 8259A с традиционной передачей сигналов прерывания одному процессору (загрузочному, BSP Bootstrap Processor) по линиям `INTR` и `NMI`;

- *режим «виртуальных проводов»* (Virtual Wire Mode) — то же, но с подачей сигналов прерывания по локальной шине APIC. При этом I/O APIC может работать совместно с PIC 8259A, обеспечивая дополнительные возможности (в частности, дополнительные входы запросов прерываний);
- *симметричный режим* (Symmetric I/O Mode) — сообщения о прерываниях от устройств генерирует APIC; прерывания могут доставляться любому процессору; каждый вход запроса индивидуально программируется с помощью *таблицы перенаправления прерываний* (I/O Redirection Table).

Первые два режима обеспечивают полную совместимость с системой прерываний PC/AT, с программной точки зрения они эквивалентны, различия лежат в области схемотехники. По аппаратному сбросу (и включении питания) система начинает работать в одном из этих режимов. Когда система подготовится к переходу в MP-режим, APIC переводится в симметричный режим и активизирует таблицу перенаправлений прерываний (предварительно программно инициализированную).

В MP-системе присутствует таблица описаний ее компонентов; к прерываниям в этой таблице относятся описатели всех I/O APIC, а также описатели назначений всех используемых источников прерываний, связанных с I/O APIC и локальными APIC. В описателе назначения для каждого источника прерываний указывается:

- тип прерывания: векторное с передачей вектора через APIC, векторное с внешней передачей вектора (от PIC 8259A), `NMI` или `SMI`;
- полярность сигнала и его тип (уровень или перепад);
- идентификатор шины, на которой расположен источник;
- идентификатор запроса на этой шине;
- идентификатор и номер входа APIC, к которому подключен данный запрос.

Согласно MPS, для симметричных систем допустимы векторы в диапазоне 10h–FEh. Уровень приоритета прерывания определяется номером его вектора, деленным на 16. Самый приоритетный уровень — нулевой.

Выделение для сообщений APIC отдельной локальной шины позволяет освободить системную шину процессора от трафика, связанного с обслуживанием прерываний (подачи подтверждений прерываний для получения вектора). В современных процессорах используется локальная шина, состоящая из трех сигнальных линий: `PICD[1:0]` — двунаправленная шина данных и `PICCLK` — сигнал синхронизации (тактовая частота). Протокол шины обеспечивает распределенный механизм арбитража: в любой момент времени каждый APIC (локальный и I/O APIC) имеет уникальное значение *приоритета арбитража* (0–15), которое динамически меняется после успешной передачи сообщения. При попытке одновременного начала передачи сообщения несколькими APIC после фазы арбитража остается единственный победитель. Получатель сообщения подтверждает успешный прием; в случае неудачи сообщение передается повторно (обеспечивается надежная доставка). Сообщения, передаваемые по локальной шине APIC, программно-невидимы; реализация и протокол шины могут быть изменены производителями процессоров и чипсетов системных плат, но это не отразится на ПО.

Контроллер I/O APIC позволяет вырабатывать значительное число запросов прерываний; каждому запросу соответствует свой элемент в таблице перенаправле-

ний, находящейся в APIC. Каждый элемент определяет способ реакции на свой запрос, вектор прерывания и процессор (процессоры) назначения, которые должны его обработать. С запросами связаны индивидуальные входы INTINn; определенный уровень или перепад сигнала на этих входах вызывает соответствующие запросы. Чувствительность и вектор (следовательно, и приоритет) для каждого запроса программируется индивидуально. Более совершенные модели I/O APIC позволяют вызывать прерывание и записью номера входа в регистр контроллера, что, например, используются для поддержки прерываний MSI на шине PCI. При этом возможна и экономия сигнальных входов: APIC может иметь входы INTINn не для всех номеров запросов, посылаемых через запись в этот регистр. Однако число запросов всегда ограничивается размером таблицы перенаправлений.

Регистры контроллеров APIC отображаются на пространство памяти. Все *локальные контроллеры APIC* используют один и тот же диапазон адресов (по умолчанию базовый адрес FEE0 0000h) — к их регистрам обращаются только программы, исполняемые на их же процессорах, и эти обращения не выводятся на системную шину. *Контроллеры I/O APIC* доступны всем процессорам, по умолчанию базовый адрес первого I/O APIC — FEC0 0000h, базовые адреса остальных контроллеров (если таковые имеются) назначаются последовательно с шагом 1000h. Часть регистров адресуется непосредственно (табл. 3.4), большая часть регистров, включая и таблицу перенаправлений, адресуется косвенно (табл. 3.5).

Таблица 3.4. Непосредственно адресуемые регистры APIC

Адрес	Размер, бит	Тип	Назначение
FEC0_0000h	8	R/W	Index Register, индекс для доступа к косвенно адресуемым регистрам
FEC0_0010h	32	R/W	Data Register, данные для обращений к косвенно адресуемым регистрам
FECO_0020h	8	WO	IRQ Pin Assertion Register, регистр программной установки запросов прерываний (запись числа 0–23 эквивалентна подаче сигнала на соответствующий вход INTINn)
FECO_0040h	8	WO	EOI Register — регистр завершения прерываний для входов, чувствительных к уровню. Запись байта — вектора прерывания — вызывает сброс бита Remote_IRR для всех входов, которым назначен данный вектор (аналогичное действие IOAPIC выполняет по сообщению *EOI*, полученному по локальной шине)

Таблица 3.5. Косвенно-адресуемые регистры APIC

Адрес	Размер, бит	Тип	Назначение
00h	32	R/W	ID (биты 24:27) — идентификатор (физический номер), программно назначаемый данному APIC. Остальные биты — резерв

продолжение ⇩

Таблица 3.5 (*продолжение*)

Адрес	Размер, бит	Тип	Назначение
01h	32	RO	`Version` — версия (возможности IOAPIC): биты 23:16 — максимальный номер элемента в таблице перенаправления; бит 15 (PRQ) — признак наличия регистра программной установки запросов прерываний; биты 7:0 — номер версии; остальные биты — резерв
02h	32	RO	`Arbitration ID` (биты 24:27) — текущее значение приоритета арбитража. Остальные биты — резерв
03h	32	R/W	`Boot Configuration` — конфигурация: бит 0 — `DT` (Delivery Type), управление механизмом доставки сообщений: 0 — через локальную шину APIC, 1 — через сообщения по системной шине (режим I/O(x)APIC)
03– 0Fh		RO	Резерв
10– 11h	64	R/W	`Redirection Table 0` — первый элемент таблицы перенаправления (см. табл. 3.6)
...	...	...	
3E– 3Fh	64	R/W	`Redirection Table 23` — последний элемент таблицы перенаправления
40– FFh		RO	Резерв

Таблица 3.6. Формат элемента таблицы перенаправлений

Биты	Назначение
63:56	`Destination` (R/W), идентификатор назначения. Если используется физическая адресация (бит 11 = 0), то биты [59:56] задают идентификатор локального APIC (биты 63:59 должны быть нулевыми). При логической адресации (бит 11 = 1) биты [63:56] задают логический адрес набора процессоров
55:17	Резерв (нули)
16	`Mask` (R/W) — маска запроса: 1 — прерывание замаскировано (но запрос не сбрасывается)
15	`Trigger Mode` (R/W) — чувствительность входа: 0 — к перепаду, 1 — к уровню
14	`Remote IRR` (R/W), удаленное подтверждение запроса прерывания (только для линий, чувствительных к уровню). Устанавливается, когда локальный APIC принимает этот запрос от IOAPIC; сбрасывается, когда IOAPIC получает команду EOI с соответствующим номером вектора
13	`Interrupt Input Pin Polarity` (R/W), полярность сигнала запроса: 0 — активный уровень высокий, 1 — низкий
12	`Delivery Status` (RO), состояние доставки: 0 — нет активности, 1 — запрос пришел, но по шине APIC еще не доставлен адресату
11	`Destination Mode` (R/W), адресация сообщения: 0 — физическая (по APIC ID), 1 — логическая (по идентификатору набора процессоров)
10:8	`Delivery Mode` (R/W) — режим доставки (см. табл. 3.7)
7:0	Vector (R/W) — вектор прерывания

Таблица 3.7. Режимы доставки сообщений

Режим	Описание
000	Доставка сигнала на входы INTR процессора (или группы) в соответствии с адресатом назначения
001	Доставка сигнала на вход INTR одного процессора из адресованной группы, выполняющего самую низкоприоритетную задачу. Если несколько процессоров выполняют задачи с одинаково низким приоритетом, сообщение получит тот, чей APIC выиграет в этот момент арбитраж на локальной шине
010	SMI/PMI (System Management Interrupt, Power Management Interrupt), прерывание системного управления и системы управления энергопотреблением, только для входов, чувствительных к перепаду. Вектор игнорируется (но должен быть нулевым)
011	Резерв
100	NMI, немаскируемое прерывание, доставляется на входы NMI всех адресованных процессоров. Должно использоваться для входов, чувствительных к перепаду
101	INIT, «мягкая» инициализация всех адресованных процессоров. Должна использоваться для входов, чувствительных к перепаду
110	Резерв
111	ExtINT, внешнее прерывание, вектор которого доставляется от внешнего контроллера PIC по команде INTA. Доставляется на входы INTR всех адресованных процессоров. Должно использоваться для входов, чувствительных к перепаду

Кроме использования последовательной локальной шины есть и иной вариант доставки сообщений к локальным APIC, использущий обращения к пространству памяти. Для этого локальные APIC настраиваются на отслеживание операций записи по определенным адресам. Источник сообщений выполняет операцию записи в пространство памяти, в которой и адрес и данные несут информацию о событии прерывания (табл. 3.8). В качестве источника сообщений может выступать расширенный контроллер, называемый *I/O(x)APIC*. Вышеупомянутый хаб ICH3 имеет возможность работы в режиме I/O(x)APIC.

Таблица 3.8. Формат сообщения о прерывании, передаваемого по системной шине

Бит	Назначение
Назначение бит адреса	
31:20	Всегда FEEh
19:12	`Destination ID`, идентификатор получателя, аналогично битам 63:56 элемента таблицы перенаправлений
11:4	Резерв (0)
3	`Redirection Hint`, признак перенаправления: 0 — сообщение доставляется агенту (процессору), идентификатор которого указан в битах 19:12; 1 — сообщение доставляется агенту с минимальным приоритетом прерываний
2	Destination Mode, режим назначения, используется только при единичном признаке перенаправления. Если биты 2 и 3 имеют единичное значение, то сообщение направляется по логическому идентификатору группы процессоров
1:0	Всегда 00

продолжение ⇩

Таблица 3.8 (*продолжение*)

Бит	Назначение
Назначение бит данных	
31:16	Всегда 0000
15	`Trigger Mode`, режим чувствительности: 1 — уровень, 0 — перепад
14	`Delivery Status`, признак для прерываний по уровню: 1 — установка активного уровня, 0 — снятие (для прерываний по перепаду всегда 1)
13:12	Всегда 00
11	`Destination Mode`, режим назначения: 1 — логический, 0 — физический
10:8	`Delivery Mode`, режим доставки (см. табл. 3.7)
7:0	Вектор прерывания

Проблема разделяемых прерываний

Линии запросов прерываний в компьютере, насыщенном периферийными устройствами, являются самым дефицитным ресурсом, поэтому приходится использовать эти линии совместно, то есть применять *разделяемые прерывания* между несколькими устройствами (*shared interrupts*). Для шины PCI с аппаратной точки зрения проблема разделения прерываний решена — здесь активным уровнем запроса является низкий, и контроллер прерываний чувствителен к уровню, а не перепаду. Для шины ISA с ее запросами прерываний по положительному перепаду разделяемость прерываний невозможна. Исключения составляют системные платы и устройства с поддержкой ISA PnP, которые можно заставить работать и по низкому уровню.

После успешного решения аппаратной задачи обеспечения разделяемости линий запроса возникает задача *идентификации источника* каждого прерывания, чтобы запустить выполнение соответствующей процедуры обработки. Желательно, чтобы эта задача решалась средствами ОС и с минимальными потерями времени.

В первых версиях PCI (до PCI 2.2 включительно) не было общепринятого способа программной индикации и запрета прерываний. К сожалению, в конфигурационных регистрах не нашлось стандартного места для бита, индицирующего введение запроса прерывания данным устройством, — тогда бы в прерываниях для PCI не было бы проблем с унификацией поддержки разделяемых прерываний. В каждом устройстве для работы с прерываниями используются свои специфические биты операционных регистров, относящихся к пространству памяти или ввода/вывода (иногда и к конфигурационному). При этом определить, является ли данное устройство в текущий момент источником прерывания, может только его обработчик прерывания (*ISR*, Interrupt Service Routine), входящий в драйвер данного устройства. Таким образом, у ОС нет иной возможности диспетчеризации разделяемых прерываний, кроме как выстроить их ISR-ы в цепочку. За расторопность и корректность ISR отвечает его разработчик. В PCI 2.3 наконец-то появился фиксирован-

ный бит (`Interrupt Status`) в регистре состояния конфигурационного пространства устройства (функции), по которому ОС может определить источник разделяемого прерывания и вызвать только его ISR. Однако упоминание о поддержке PCI 2.3 в описаниях устройств и операционных систем встречается не часто.

Обработчики прерываний устройств должны вести себя корректно, учитывая возможность попадания в цепочку обработчиков разделяемого прерывания. В процессе обработки прерывания очередной обработчик в цепочке чтением известного ему регистра своего устройства должен определить, не это ли устройство вызвало прерывание. Если это, то обработчик должен выполнить необходимые действия и сбросить сигнал запроса прерывания от своего устройства, после чего передать управление следующему обработчику в цепочке; в противном случае он просто передает управление следующему обработчику. Встречается типичная ошибка обработчика прерываний: прочитав регистр состояния устройства и не обнаружив признака запроса, драйвер «на всякий случай» выполняет сброс всех источников запроса (а то и сброс всего устройства). Эту ошибку порождает незадачливый разработчик драйвера, не учитывающий возможности разделяемости прерываний и не доверяющий разработчикам аппаратных средств. Увидев в процессе отладки эту неожиданную ситуацию (прерывание вызвано, а источник не виден), он ее «учитывает» введением вредного фрагмента программного кода. Вредность заключается в том, что с момента чтения регистра устройства (не давшего признака запроса) и до выполнения этого ненужного сброса в устройстве может возникнуть запрос прерывания, который будет «вслепую» сброшен и, следовательно, потерян.

Однако и при корректности обработчиков, выстроенных в цепочку, разделяемые прерывания для разнотипных устройств в общем случае работоспособными считать нельзя — возможны потери прерываний от устройств, требующих быстрой реакции. Это может происходить, если обработчик такого устройства окажется в конце цепочки, а предшествующие ему обработчики окажутся «нерасторопными» (не самым быстрым способом обнаружат, что прерывание — чужое). Поведение системы в такой ситуации может меняться в зависимости от порядка загрузки драйверов. Для нескольких однотипных устройств (например, сетевых адаптеров на однотипных микросхемах контроллеров), пользующихся одним драйвером, разделяемые прерывания работают вполне успешно.

Проявления конфликтов по прерываниям могут быть разнообразными. Сетевая карта не сможет принимать кадры из сети или будет их иногда терять (при этом она может их успешно посылать). У устройств хранения доступ к данным будет поразительно медленным (иногда можно минутами ожидать, например, появления информации о файлах и каталогах) или вообще невозможным. Звуковые карты будут молчать или «заикаться», на видеопроигрывателях изображение будет дергаться и т. д. Конфликты могут приводить и к внезапным перезагрузкам компьютера, например по приходу кадра из сети или сигналу от модема. Спасением от бед разделяемости может быть перестановка карт PCI в подходящий слот, в котором конфликты не наблюдаются (это может и не означать, что их нет). Однако попадаются «подарки разработчиков» интегрированных плат, у которых из нескольких слотов PCI неразделяемая линия прерывания есть только у одного (а то

и нет вообще). Такие недуги без скальпеля и паяльника, как правило, не лечатся. Более радикальный способ — переход на сигнализацию прерываний через сообщения — MSI (см. ниже).

Традиционные прерывания PCI — INTx#

Для устройств PCI выделяется четыре проводных линии запросов (IRQX, IRQY, IRQZ, IRQW), соединяемых с контактами INTA#, INTB#, INTC# и INTD# всех слотов PCI с циклическим смещением цепей (см. рис. 3.1). Соответствие линий INTx# и входов IRQ для устройства любой шины PCI приведено в табл. 3.9. Мосты PCI просто электрически соединяют одноименные линии INTx своих первичных и вторичных шин. В системах с APIC, в которых число входов запросов увеличено до 24, дополнительные 8 входов могут использоваться периферийными устройствами, установленными на системной плате. На слотах PCI остаются доступными лишь четыре обычные линии запросов.

Таблица 3.9. Коммутация запросов прерываний для устройств PCI

Контакт слота	Вход коммутатора запроса для устройства с номером:			
	0, 4, 8, … 28	1, 5, 9, … 29	2, 6, 10, … 30	3, 7, 11, … 31
INTA#	IRQW	IRQX	IRQY	IRQZ
INTB#	IRQX	IRQY	IRQZ	IRQW
INTC#	IRQY	IRQZ	IRQW	IRQX
INTD#	IRQZ	IRQW	IRQX	IRQY

Устройство PCI вводит сигнал прерывания *низким* уровнем (выходом с открытым коллектором или стоком) на выбранную линию INTx#. Этот сигнал должен удерживаться до тех пор, пока программный драйвер, вызванный по прерыванию, не сбросит запрос прерывания, обратившись по шине к данному устройству. Если после этого контроллер прерываний снова обнаруживает низкий уровень на линии запроса, это означает, что запрос на ту же линию ввело другое устройство, разделяющее данную линию с первым, и оно тоже требует обслуживания.

Заметим, что распространение сигнала прерывания не синхронизируется с передачей данных. Возможна ситуация, когда активное устройство, выполнив пересылку данных в память, посылает сигнал прерывания, оповещающий об этом событии. Однако записи, отосланные устройством, могут задержаться в мостах (если шина слишком загружена), и процессор начнет обрабатывать прерывание, еще не получив всех этих данных. Чтобы гарантировать целостность данных, программа ISR первым делом должна выполнить чтение какого-либо регистра своего устройства — чтение «из-за моста» принудит все мосты к выгрузке всех буферов отправленных записей (см. главу 4).

Линии запросов от слотов PCI и PCI-устройств системной платы коммутируются на входы контроллеров прерываний относительно произвольно. Конфигурационное ПО может определить и указать занятые линии запросов и номер входа кон-

троллера прерываний обращением к конфигурационному пространству устройства (см. главу 5). Программный драйвер, прочитав конфигурационные регистры, тоже может определить эти параметры для того, чтобы установить обработчик прерываний на нужный вектор и при обслуживании сбрасывать запрос с требуемой линии.

Каждая функция устройства PCI может задействовать свою линию запроса прерывания, но его обработчик прерывания должен быть готовым к ее разделению (совместному использованию) с другими устройствами. Если устройству требуется только одна линия запроса, то оно должно занимать линию INTA#, если две — INTA# и INTB#, и т. д. С учетом циклического сдвига линий запроса это правило позволяет установить в 4 соседних слота 4 простых устройства, и каждое из них будет занимать отдельную линию запроса прерывания. Если какой-то карте требуется 2 линии, то для монопольного использования прерываний нужно оставить соседний слот свободным. Однако не следует забывать, что PCI-устройства системной платы тоже задействуют прерывания с той же закономерностью (кроме контроллера IDE, который, к счастью, держится особняком). Порт AGP в плане прерываний следует рассматривать наравне со слотом PCI. Таким образом, может оказаться, что монопольные линии прерывания присутствуют далеко не на всех слотах.

Назначение прерываний устройствам (функциям) выполняет процедура POST, и этот процесс управляем лишь частично. Параметрами CMOS Setup (PCI/PNP Configuration) пользователь определяет номера запросов прерываний, доступных шине PCI. В зависимости от версии BIOS это может выглядеть по-разному: либо каждой линии INTA#...INTD# явно назначается свой номер, либо ряд номеров отдается «на откуп» устройствам PCI вместе с устройствами ISA PnP (в противоположность устройствам «Legacy ISA»). В итоге POST определяет соответствие линий INTA#...INTD# номерам запросов контроллера и соответствующим образом программирует коммутатор запросов. По воле пользователя может оказаться так, что не каждой линии запроса шины PCI достается отдельный вход контроллера прерываний. Тогда коммутатор организует объединение нескольких линий запросов PCI на один вход контроллера, то есть разделяемыми станут даже разные линии запросов прерываний для PCI. В самом худшем случае устройствам PCI не достанется ни одного входа контроллера прерываний. Заметим, что BIOS вряд ли отдаст шине PCI прерывания 14 и 15 (их забирает контроллер IDE, если он не отключен), а также 3 и 4 (COM-порты). Новые версии ОС настолько сильно вникают в аппаратную платформу, что позволяют себе (зная чипсет системной платы или пользуясь функциями PCI BIOS) управлять коммутатором запросов прерываний. Эту возможность можно запретить или разрешить, например, в ОС Windows снятием или установкой флажка Использовать управление IRQ (PCI Interrupt Steering) в свойствах шины PCI (Панель управления ▸ Системные устройства ▸ Шина PCI).

Драйвер (или иное ПО), работающий с устройством PCI, определяет *номер входа контроллера прерывания*, доставшийся устройству (точнее, функции), чтением конфигурационного регистра Interrupt Line. По этому номеру определяется вектор (см. табл. 3.1), значение 255 означает, что номер не назначен. Номер входа каж-

дому устройству заносит тест POST. Для этого он считывает регистр `Interrupt Pin` каждой обнаруженной функции и по номеру устройства (читай: географическому адресу!) определяет, какая из линий INTA#...INTD# (на входе коммутатора запросов) используется. Заметим, что правила, по которым на системной плате определяется соответствие между `Interrupt Pin` и входными линиями коммутатора запросов в зависимости от номера устройства, строго не регламентированы (деление номера устройства на 4 — это всего лишь рекомендация), но их твердо знает версия BIOS данной системной платы. К этому моменту тест POST уже определил таблицу соответствия этих линий номерам входов; пользуясь этой таблицей, он записывает нужное значение в конфигурационный регистр `Interrupt Line`. Определить, есть ли еще претенденты на тот же номер прерывания, можно, лишь просмотрев конфигурационные регистры функций всех устройств, обнаруженных на шине (это не так уж сложно сделать, пользуясь функциями PCI BIOS).

В PCI BIOS (см. главу 5) начиная с версии 2.1 имеются функции определения возможностей и конфигурирования прерываний. Одна из функций возвращает структуру данных, в которой для каждого устройства (на каждой шине) сообщается, с какими входами контроллера прерываний (IRQx) могут быть связаны его линии INTx и с каким именно связаны в данный момент. Также указывается и физический номер слота, в который установлено данное устройство. Кроме того, возвращается и битовая карта, показывающая, какие входы IRQx отводятся исключительно шине PCI (и не используются абонентами других шин). Функция установки для заданного устройства устанавливает связь выбранного сигнала (INTx) с выбранным входом контроллера прерываний (IRQx), то есть программирует коммутатор. Эта функция предназначена для использования только конфигурационным ПО (BIOS, ОС), но никак не драйвером устройства. Тот, кто ею пользуется, сам отвечает за возможные конфликты, за правильное программирование контроллера прерываний (выбранный вход должен быть чувствительным к низкому уровню, а не положительному перепаду), за корректировку информации в конфигурационном пространстве всех затронутых устройств (у которых линии запроса связаны с выбранной линией INTx).

Сигнализация событий управления энергопотреблением — PME#

Линия PME#, введенная в PCI 2.0, служит для сигнализации в системе управления энергопотреблением PM (Power Management): смены состояния устройств, генерации пробуждения системы по событию. Эта линия электрически доступна всем устройствам PCI; как и линии INTx#, PME# никак не обрабатывается мостами, а лишь доводится до всех абонентов. Логика сигнализации аналогична INTx#: устройство сигнализирует о событии, замыкая линию PME# на «землю», таким образом, сигналы о событиях логически собираются по функции ИЛИ. Обработчик этого прерывания может выявить устройство, подавшее сигнал, путем программных обращений к конфигурационным регистрам всех устройств, способных к генерации этого

сигнала. Устройства (функции), имеющие отношение к управлению энергопотреблением, имеют в конфигурационном пространстве структуру с идентификатором Capability ID = 01 и набор регистров:

- PMC (Power Management Capabilities) — регистр возможностей PM: версия спецификации, какие состояния поддерживаются, в состояниях возможна генерация PME#, нужен ли сигнал CLK для генерации PME#, каково потребление по линии 3,3VAux;
- PMCSR (Power Management Control/Status Register) — регистр управления и состояния PM: признак введения PME#, его сброс и разрешение; состояние PM, управление данными, выводимыми через регистр Data;
- Data — регистр (необязательный), через который может выводиться, например, информация о потребляемой мощности;
- PMCSR_BSE (Bridge Support Extensions) — регистр расширенного управления мостом: признак поддержки мостом управления вторичной шиной в зависимости от состояния PM; состояние вторичной шины при переходе в состояние потребления *D3* (останов синхронизации или еще и снятие питания).

Подробности управления энергопотреблением в PCI и форматы соответствующих конфигурационных регистров приведены в PCI PM 1.1.

Прерывания сообщениями — MSI

На шине PCI имеется прогрессивный механизм оповещения об асинхронных событиях, основанный на *передаче сообщений MSI* (Message Signaled Interrupts). Здесь для сигнализации запроса прерывания устройство запрашивает управление шиной и, получив его, посылает сообщение. Сообщение выглядит как обычная запись двойного слова в ячейку памяти, *адрес* (32-битный или 64-битный) и *шаблон сообщения* на этапе конфигурирования устройств записываются в конфигурационные регистры устройства (точнее, функции). В сообщении старшие 16 бит всегда нулевые, а младшие 16 бит несут информацию об источнике прерывания. Устройство (функция) могут нуждаться в сигнализации нескольких типов запросов; в соответствии с его потребностями и своими возможностями система указывает устройству (функции), сколько различных типов запросов оно может вырабатывать.

Возможность использования MSI описывается в конфигурационном пространстве структурой MSI Capability (CAP_ID = 05h), которая должна присутствовать в пространстве каждой функции, поддерживающей MSI. В структуре имеется 3 или 4 регистра (рис. 3.2):

- Message Address — 32-разрядный адрес памяти, по которому передается сообщение (биты [1:0] = 00). Если используется 64-битный адрес (установлен бит 7 в регистре Message Control), то его старшая часть располагается в регистре Message Upper Address. Значения в регистры адреса заносит системное ПО на этапе конфигурирования;

- `Message Data` — 16-битный шаблон данных, передаваемых в сообщении по линиям AD[15:0]. Значение шаблона записывается системным ПО на этапе конфигурирования. В сообщении, передаваемом функцией, для различения разных условий прерывания могут модифицироваться только несколько младших бит, количество которых определяется значением поля `Multiple Message Enable`. Остальные биты сообщения должны соответствовать шаблону; биты [31:16] всегда нулевые;
- `Message Control` — управление сообщениями (16 бит). Битом 7 функция сообщает о способности генерировать 64-битный адрес. В поле `Multiple Message Capable` функции задается ее способность генерировать различимые условия прерывания, в поле `Multiple Message Enable` система указывает функции допустимое число условий. Здесь значения 000–101 двоично кодируют число младших бит шаблона, которые устройство может модифицировать для идентификации источника прерывания: 000 — ни одного (устройству доступен лишь один идентификатор), 101 — пять, значения 110 и 111 зарезервированы. Бит `MSI_Enable` разрешает использование MSI.

По аппаратному сбросу MSI запрещен; его можно разрешить программно установкой бита `MSI Enable` (после программирования адреса и шаблона сообщения), и тогда генерация прерываний по линиям INTx# запрещается. Самая «богатая» прерываниями функция (для которой `Multiple Message Enable` = 101) при записи сообщения идентифицирует конкретное условие прерывания (одно из 32 возможных) по линиям AD[4:0].

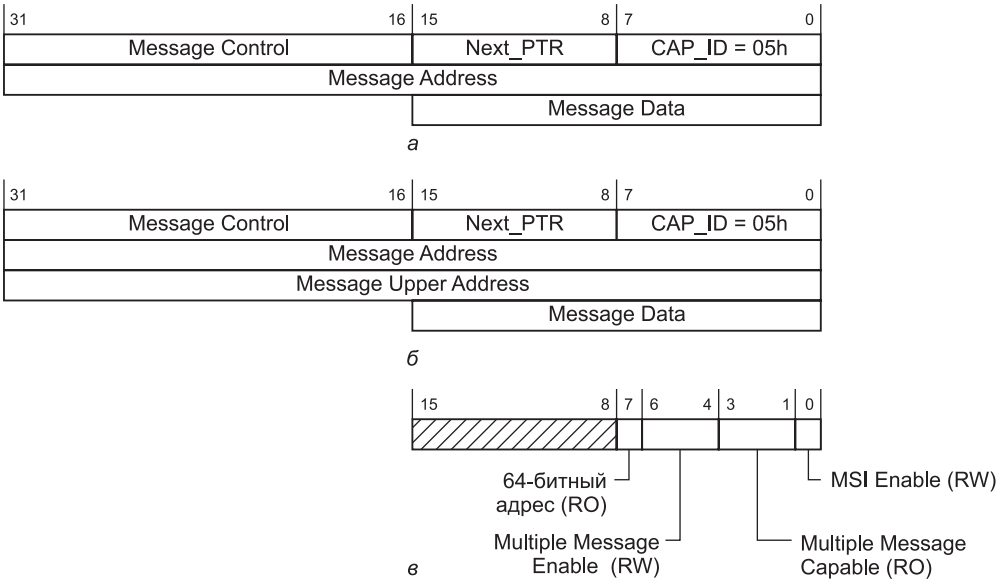


Рис.3.2. Формат регистров MSI: *а* — расположение регистров в структуре MSI Capability с 32-битным адресом; *б* — с 64-битным адресом; *в* — регистр Message_Control

Отметим, что быстрая посылка подряд двух и более одинаковых сообщений может быть воспринята системой как одно прерывание (из-за низкой реактивности). Если требуется обработка каждого из них, обработчик должен обеспечивать уведомление устройства, а оно не должно посылать следующее сообщение до получения уведомления, чтобы прерывания не терялись. Различающиеся сообщения друг другу не мешают.

Прерывания через MSI позволяют избежать бед разделяемости, обусловленной дефицитом линий запросов прерывания в PC. Кроме того, они решают проблему целостности данных: все данные, записываемые устройством до посылки MSI, дойдут до получателя гарантированно раньше начала обработки MSI. Прерывания через MSI от одних устройств в одной системе могут использоваться наряду с обычными INTx# от других устройств. Но каждое устройство (функция), использующее MSI, не должно использовать прерывания через линии INTx#.

Механизм MSI может использоваться на системных платах, имеющих «продвинутый» контроллер прерываний APIC. Правда, конкретная реализация поддержки MSI может потенциальные возможности облегчения идентификации большого числа запросов прерывания свести лишь к увеличению числа доступных запросов прерываний (и используемых ими векторов). Так, например, для системных плат на чипсетах с хабом ICH2 и ICH3 фирмы Intel поддержка MSI сводится к организации альтернативных путей подачи запросов IRQ[1:23] на входы APIC (запросы IRQ с номерами 0, 2, 8 и 13 через MSI не передаются). Всем устройствам PCI назначается один и тот же адрес сообщений (Message Address = FEC00020h), по которому в APIC находится регистр IRQ Pin assertion. В сообщении используются лишь младшие 5 бит, в которых указывается номер взводимого запроса прерывания в диапазоне 1–23 (исключая 2, 8 и 13). Линии запросов, используемые в MSI, программируются на чувствительность к перепаду (сообщения имитируют только фронт сигнала). По этой причине прерывания с номерами, используемыми в MSI, не могут использоваться совместно (разделяемо) с прерываниями, полученными другими способами (по линиям запросов от устройств PCI и от других устройств системной платы). Возможно, что в других платформах прерывания через MSI используются более эффективно.

ГЛАВА 4

Мосты PCI и PCI-X

Мосты PCI (*PCI Bridge*) — специальные аппаратные средства соединения шин PCI (и PCI-X) между собой и с другими шинами. *Главный мост* (*Host Bridge*) используется для подключения PCI к центру компьютера (системной памяти и процессору). «Почетной обязанностью» главного моста является генерация обращений к конфигурационному пространству под управлением центрального процессора, что позволяет хосту (центральному процессору) выполнять конфигурирование всей подсистемы шин PCI. В системе может быть и несколько главных мостов, что позволяет предоставить высокопроизводительную связь с центром большему числу устройств (число устройств на одной шине ограниченно). Из этих шин одна назначается условно главной (bus 0).

Равноранговые мосты PCI (*Peer-to-Peer Bridge*) используются для подключения дополнительных шин PCI. Эти мосты всегда вносят дополнительные накладные расходы на передачу данных, так что эффективная производительность при обмене устройства с центром снижается с каждым встающим на пути мостом.

Для подключения шин PCMCIA, CardBus, MCA, ISA/EISA, X-Bus и LPC используются специальные мосты, входящие в чипсеты системных плат или же являющиеся отдельными устройствами PCI (микросхемами). Эти мосты выполняют преобразование интерфейсов соединяемых ими шин, синхронизацию и буферизацию обменов данных.

Каждый мост программируется — ему указываются диапазоны адресов в пространствах памяти и ввода-вывода, отведенные устройствам его шин. Если адрес ЦУ текущей транзакции на одной шине (стороне) моста относится к шине противоположной стороны, мост транслирует транзакцию на соответствующую шину и обеспечивает согласование протоколов шин. Таким образом, совокупность мостов PCI выполняет *маршрутизацию* (routing) обращений по связанным шинам. Если в системе имеется несколько главных мостов, то сквозная маршрутизация между устройствами разных шин может оказаться невозможной: главные мосты друг с другом могут оказаться связанными лишь через магистральные пути контроллера памяти. Поддержка трансляции всех типов транзакций PCI через главные мосты в этом случае оказывается чересчур сложной, а потому спецификацией PCI строго и не требуется. Таким образом, все активные устройства всех шин PCI могут обращаться к системной памяти, но возможность равнорангового общения может оказаться в зависимости от принадлежности этих устройств той или иной шине PCI.

Применение мостов PCI предоставляет такие возможности, как:

- увеличение возможного числа подключенных устройств, преодолевая ограничения электрических спецификаций шины;
- разделение устройств PCI на сегменты — шины PCI — с различными характеристиками разрядности (32/64 бит), тактовой частоты (33/66/100/133 МГц), протокола (PCI, PCI-X Mode 1, PCI-X Mode 2, PCI Express). На каждой шине все абоненты равняются на самого слабого участника; правильная расстановка устройств по шинам позволяет с максимальной эффективностью использовать возможности устройств и системной платы;
- организация сегментов с «горячим» подключением/отключением устройств;
- организация одновременного параллельного выполнения транзакций от инициаторов, расположенных на разных шинах.

Каждый мост PCI соединяет только две шины: *первичную* (primary bus), находящуюся ближе к вершине иерархии, с *вторичной* (secondary bus); интерфейсы моста, которыми он связан с этими шинами, называются соответственно первичным и вторичным. Допускается только чисто древовидная конфигурация, то есть две шины соединяются друг с другом лишь одним мостом и нет «петель» из мостов. Шины, подсоединяемые ко вторичному интерфейсу данного моста другими мостами, называются *подчиненными* (subordinated bus). Мосты PCI образуют иерархию шин PCI, на вершине которой находится *главная шина* с нулевым номером, подключенная к главному мосту. Если главных мостов несколько, то из их шин (равных друг другу по рангу) условно главной будет шина, которой назначен нулевой номер.

Мост должен выполнять ряд обязательных функций:

- обслуживать шину, подключенную к его вторичному интерфейсу:
  - выполнять арбитраж — прием сигналов запроса `REQx#` от ведущих устройств шины и предоставление им права на управление шиной сигналами `GNTx#`;
  - парковать шину — подавать сигнал `GNTx#` какому-то устройству, когда управление шиной не требуется ни одному из задатчиков;
  - генерировать конфигурационные циклы типа 0 с формированием индивидуальных сигналов `IDSEL` к адресуемому устройству PCI;
  - «подтягивать» управляющие сигналы к высокому уровню;
  - определять возможности подключенных устройств и выбирать удовлетворяющий их режим работы шины (частота, разрядность, протокол);
  - формировать аппаратный сброс (`RST#`) по сбросу от первичного интерфейса и по команде, сообщая о выбранном режиме специальной сигнализацией (см. главу 6).
- поддерживать карты ресурсов, находящихся по разные стороны моста;
- отвечать под видом целевого устройства на транзакции, инициированные мастером на одном интерфейсе и адресованные к ресурсу, находящемся со стороны другого интерфейса; транслировать эти транзакции на другой интерфейс,

выступая в роли ведущего устройства (мастера), и передавать их результаты истинному инициатору.

Мосты, выполняющие данные функции, называются *прозрачными* (transparrent bridge); для работы с устройствами, находящимися за такими мостами, не требуется дополнительных драйверов моста. Именно такие мосты описаны в спецификации PCI Bridge 1.1, и для них, как устройств PCI, есть специальный класс (06). В данном случае подразумевается «плоская» модель адресации ресурсов (памяти и ввода-вывода): каждое устройство имеет свои адреса, уникальные (не пересекающиеся с другими) в пределах данной системы (компьютера).

Существуют и *непрозрачные мосты* (non-transparrent bridge), которые позволяют организовывать обособленные сегменты со своими локальными адресными пространствами. Непрозрачный мост выполняет трансляцию (преобразование) адресов для транзакций, у которых инициатор и целевое устройство находятся по разные стороны моста. Досягаемыми через такой мост могут быть и не все ресурсы (диапазоны адресов) противоположной стороны. Непрозрачные мосты используются, например, когда в компьютере выделяется подсистема «интеллигентного ввода-вывода» (I_20) со своим процессором ввода-вывода и локальным адресным пространством.

Маршрутизирующие функции прозрачного моста

Задача маршрутизации — определение, где по отношению к мосту находится ресурс, адресованный каждой транзакции, — является первоочередной при обработке каждой транзакции, «увиденной» мостом на любом из своих интерфейсов. Эта задача решается двояко, поскольку в фазе адреса может передаваться как иерархический адрес PCI (шина ▸ устройство ▸ функция), так и «плоский» адрес памяти или порта ввода-вывода.

Маршрутизация по иерархическому адресу

Через номера шины и устройства адресуются транзакции конфигурационной записи и чтения, генерации специального цикла, а в PCI-X еще и завершение расщепленной транзакции, а также сообщения DIM. Для этих транзакций маршрутизация основана на системе нумерации шин. Номера назначаются шинам PCI при конфигурировании системы строго последовательно, номера мостов соответствуют номерам их вторичных шин. Так, главный мост имеет номер 0. Номера подчиненных шин моста начинаются с номера, следующего за номером его вторичной шины. Таким образом, для каждого моста необходимые ему знания топологии шин системы описываются *списком номеров шин* — тремя числовыми параметрами в его конфигурационном пространстве:

- `Primary Bus Number` — номер первичной шины;
- `Secondary Bus Number` — номер вторичной шины (это и номер моста);

- `Subordinate Bus Number` — максимальный номер подчиненной шины.

Все шины с номерами в диапазоне от `Secondary Bus Number` до `Subordinate Bus Number` включительно будут лежать со стороны вторичного интерфейса, все остальные — на стороне первичного.

Знание номеров шины позволяет мостам распространять обращения к конфигурационным регистрам устройств в сторону от хоста к подчиненным шинам и распространять специальные циклы во всех направлениях. Ответы на расщепленные транзакции (*Split Complete*) мост транслирует с одного интерфейса на другой, если они адресованы к шине противоположного интерфейса.

Конфигурационные циклы типа 0 и специальные циклы мостами не транслируются. Конфигурационные транзакции типа 1, обнаруженные на первичном интерфейсе, мост обрабатывает следующим образом:

- Преобразует их в конфигурационные циклы типа 0 или специальные циклы, если номер шины (на линиях `AD[23:16]`) соответствует номеру вторичной шины. При преобразовании в цикл типа 0 номер устройства с первичной шины, полученный в фазе адреса, декодируется в позиционный код на вторичной шине (см. главу 2), номер функции и регистра передается без изменений, биты `AD[1:0]` на вторичной шине обнуляются. В PCI-X кроме позиционного кода на вторичную шину передается и номер устройства. Преобразование в специальный цикл (изменение кода команды) производится, если в полях номера устройства и функции все биты единичные, а в поле номера регистра — нулевые.
- Пропускает их с первичного интерфейса на вторичный без изменения, если номер шины соответствует диапазону номеров подчиненных шин.
- Игнорирует, если номер шины лежит вне диапазона номеров шин стороны вторичного интерфейса.

Со стороны вторичного интерфейса мост передает на первичный только конфигурационные циклы типа 1, относящиеся к специальным циклам (в полях номера устройства и функции все биты единичные, а в поле номера регистра — нулевые). Если номер шины соответствует номеру первичной шины, мост преобразует эту транзакцию в специальный цикл.

Если конфигурационный цикл не воспринимается ни одним из устройств, мосты могут эту ситуацию отрабатывать двояко: фиксировать отсутствие устройства (сработает *Master Abort*) или же выполнять операции вхолостую. Однако в любом случае чтение конфигурационного регистра несуществующего устройства (функции) должно возвращать значение FFFFFFFFh (это будет безопасной информацией, поскольку даст недопустимое значение идентификатора устройства).

Маршрутизация по «плоскому» адресу

Для манипулирования с транзакциями обращения к памяти и портам ввода-вывода мосту нужны *карты адресов*, на которых отмечены области, принадлежащие устройствам вторичной и подчиненных шин. В системе с плоской уникальной адресацией этого достаточно. Для отмеченных областей мост должен отвечать в ка-

честве целевого устройства на транзакции, «увиденные» им на первичном интерфейсе, и инициировать их в роли мастера на вторичном интерфейсе; остальные транзакции на первичном интерфейсе он игнорирует. Для адресов вне этих областей мост должен вести себя «зеркально»: отвечать в качестве целевого устройства на транзакции, «увиденные» им на вторичном интерфейсе, и инициировать их на первичном интерфейсе; остальные транзакции на вторичном интерфейсе он игнорирует. Каким образом мост транслирует транзакции, описано далее.

Каждый мост PCI-PCI имеет по одному описателю на каждый из трех типов ресурсов: ввода-вывода, «настоящей» памяти (допускающей предвыборку) и памяти, на которую отображены регистры ввода-вывода. В описателе указывается базовый адрес и размер области. Ресурсы одного типа для всех устройств, находящихся за мостом (на вторичной и всех подчиненных шинах), должны быть собраны в одну, по возможности компактную, область.

Область адресов ввода-вывода задается 8-битными регистрами `I/O Base` и `I/O Limit` с гранулярностью 4 Кбайт. Эти регистры своими старшими битами определяют только 4 старших бит 16-разрядного адреса начала и конца транслируемой области. Младшие 12 бит для `I/O Base` подразумеваются 000h, для `I/O Limit` — FFFh. Если на вторичной стороне моста нет портов ввода-вывода, то в `I/O Limit` записывается число меньшее, чем в `I/O Base`. Если мост не поддерживает карту адресов ввода-вывода, то оба регистра при чтении всегда возвращают нули; такой мост транзакции ввода-вывода с первичной на вторичную сторону не транслирует. Если мост поддерживает только 16-битную адресацию ввода-вывода, то при чтении в младших 4 бит обоих регистров всегда возвращает нули. При этом подразумевается, что старшие биты адресов `AD[31:16]` = 0, но они также подлежат декодированию. Если мост поддерживает 32-битную адресацию ввода-вывода, то при чтении в младших четырех битах обоих регистров возвращается 0001. При этом старшие 16 бит нижней и верхней границ находятся в регистрах `I/O Base Upper 16 Bits` и `I/O Limit Upper 16 Bits`.

Мост транслирует транзакции ввода-вывода указанной области с первичного интерфейса на вторичный только при установленном бите `I/O Space Enable` в регистре команд. Транзакции ввода-вывода со вторичного интерфейса на первичный транслируются только при установленном бите `Bus Master Enable`.

Ввод-вывод, отображенный на память, может использовать адреса в пределах первых 4 Гбайт (предел 32-битной адресации) с гранулярностью 1 Мбайт. Транслируемая область задается регистрами `Memory Base` (начальный адрес) и `Memory Limit` (конечный адрес), в которых задаются только старшие 12 бит адреса `AD[31:20]`, младшие биты `AD[19:0]` подразумеваются равными 0 и FFFFFh соответственно. Кроме того, транслироваться может и область памяти VGA (см. далеее).

«Настоящая» память устройств PCI, допускающая предвыборку, может располагаться как в пределах 32-битной адресации (4 Гбайт), так и 64-битной, с гранулярностью 1 Мбайт. Транслируемая область задается регистрами `Prefetchable Memory Base` (начальный адрес) и `Prefetchable Memory Limit` (конечный адрес). Если в младших битах [3:0] этих регистров чтение возвращает 0001, то это признак

поддержки 64-битной адресации. В этом случае старшая часть адресов находится в регистрах `Prefetchable Base Upper 32 Bits` и `Prefetchable Limit Upper 32 Bits`. Мост может и не иметь специальной поддержки предвыбираемой памяти, тогда вышеуказанные регистры будут при чтении возвращать нули.

Мост транслирует транзакции памяти указанных областей с первичного интерфейса на вторичный только при установленном бите `Memory Space Enable` в регистре команд. Транзакции памяти со вторичного интерфейса на первичный транслируются только при установленном бите `Bus Master Enable`.

С мостами связаны понятия позитивного и субтрактивного декодирования адресов. Рядовые агенты PCI (устройства и мосты) отзываются только на обращения по адресам, принадлежащим областям, описанным в их конфигурационном пространстве (через базовые адреса и диапазоны памяти или ввода-вывода). Такой способ декодирования называется *позитивным*. Мост с *позитивным декодированием* (*positive decoding*) пропускает через себя только обращения, принадлежащие определенному списку адресов, заданному в его конфигурационных регистрах. Мост с *субтрактивным декодированием* (*subtractive decoding*) пропускает через себя обращения, не относящиеся к другим устройствам. Его области прозрачности формируются как бы вычитанием (откуда и название) из общего пространства областей, описанных в конфигурационных пространствах других устройств. Физически субтрактивное декодирование устройством (мостом) выполняется проще: устройство отслеживает на шине все транзакции интересующего его типа (обычно обращения к портам или памяти), и если не видит на них ответа (сигнала `DEVSEL#` в тактах 1–3 после `FRAME#`) ни от одного из обычных устройств, считает эту транзакцию «своей» и само вводит `DEVSEL#`. Возможность субтрактивного декодирования имеется только у мостов определенного типа, и она является дополнением к позитивному декодированию. Субтрактивное декодирование приходится применять для старых устройств (ISA, EISA), чьи адреса разбросаны по пространству так, что их не собрать в область позитивного декодирования приемлемого размера. Субтрактивное декодирование применяется для мостов, подключающих старые шины расширения (ISA, EISA). Позитивное и субтрактивное декодирование относится только к обращениям, направленным в пространства памяти и ввода-вывода. Конфигурационные обращения маршрутизируются с помощью номера шины, передаваемого в циклах типа 1 (см. главу 2): каждый мост «знает» номера всех шин, его окружающих. На поддержку субтрактивного декодирования может указывать только специфический код класса 060401h, обнаруженный в заголовке конфигурационных регистров данного моста.

Поддержка адресации ввода-вывода шины ISA

В адресации портов ввода-вывода есть особенности, связанные с «наследием», доставшимся от шины ISA. 10-битное декодирование адреса, применявшееся в шине ISA, приводит к тому, что каждый из адресов диапазона 0–3FFh (предел охвата 10-битным адресом) имеет еще по 63 псевдонима (aliase), по которым можно обращаться к тому же устройству ISA. Так, например, для адреса 0378h псевдонимами являются x778h, xB78h и xF78h (x — любая шестнадцатеричная цифра). Псевдо-

нимы адресов ISA используются в разных целях, в частности, и в системе ISA PnP. Область адресов 0–FFh зарезервирована за системными (не пользовательскими) устройствами ISA, для которых псевдонимы не используют. Таким образом, в каждом килобайте адресного пространства ввода-вывода последние 768 байт (адреса 100–1FF) могут являться псевдонимами, а первые 256 байт (0–0FFh) — нет. В регистре управления мостом присутствует бит `ISA Enable`, установка которого приведет к вычеркиванию областей-псевдонимов из общей области адресов, описанной регистрами моста `I/O Base` и `I/O Limit`. Это вычеркивание действует только для первых 64 Кбайт адресного пространства (16-битного адреса). Мост не будет транслировать с первичного интерфейса на вторичный транзакции, принадлежащие этим вычеркнутым областям. И наоборот, с вторичного интерфейса транзакции, относящиеся к данным областям, будут транслироваться на первичный. Эта возможность нужна для совместного использования малого (64 Кбайт) пространства адресов устройствами PCI и ISA, примиряя «изрезанность» карты адресов ISA с возможностью задания лишь одной области адресов ввода-вывода для каждого моста. Данный бит имеет смысл устанавливать для мостов, за которыми нет устройств ISA. Эти мосты будут транслировать «вниз» все транзакции ввода-вывода, адресованные к первым 256 байтам каждого килобайта области адресов, описанной регистрами моста `I/O Base` и `I/O Limit`. Эти адреса конфигурационное ПО может выделять устройствам PCI, находящимся «ниже» данного моста (кроме адресов 0000h–00FFh, относящихся к устройствам системной платы).

Специальная поддержка VGA

В мостах может *присутствовать специальная поддержка графического адаптера VGA*, который может находиться на стороне вторичного интерфейса моста. Эта поддержка индицируется и разрешается битом `VGA Enable` конфигурационного регистра моста. При включенной поддержке мост осуществляет трансляцию обращений к памяти VGA в диапазоне адресов 0A0000h–0BFFFFh, а также регистрам ввода-вывода в диапазонах 3B0h–3BBh и 3C0h–3DFh и всех их 64 псевдонимов (линии адреса `AD[15:10]` не декодируются). Такой особый подход объясняется данью обеспечения совместимости с самым распространенным графическим адаптером и невозможностью описания всех необходимых областей в таблицах диапазонов адресов для позитивного декодирования. Кроме того, для поддержки VGA требуется особый подход к обращениям в регистры палитр, которые расположены по адресам 3C6h, 3C8h и 3C9h, и их псевдонимам (здесь опять же линии адреса `AD[15:10]` не декодируются).

Слежение за записью в палитры VGA (*VGA Palette Snooping*) является исключением из правила однозначной маршрутизации обращений к памяти и вводу-выводу. Графическая карта в компьютере с шиной PCI обычно устанавливается в эту шину или в порт AGP, что логически эквивалентно установке в шину PCI. На VGA-карте имеются регистры палитр (`Palette Registers`), традиционно приписанные к пространству ввода-вывода. Если графическая система содержит еще и карту смешения сигналов графического адаптера с сигналом «живого видео», перехватывая двоичную информацию о цвете текущего пиксела по шине VESA Feature

Connector (снимаемую до регистра палитр), цветовая гамма будет определяться регистрами палитр, размещенными на этой дополнительной карте. Возникает ситуация, когда операция записи в регистр палитр должна отрабатываться одновременно и в графическом адаптере (на шине PCI или AGP), и в карте видеорасширения, которая может размещаться даже на другой шине (в том числе и ISA). В CMOS Setup может присутствовать параметр `PCI VGA Palette Snoop`, управляющий битом `VGA Snoop Enable` в конфигурационном регистре моста PCI-ISA. При его включении запись в порты ввода-вывода по адресу регистров палитр будет вызывать транзакцию не только на той шине, на которой установлен графический адаптер, но и на других шинах. Чтение же по этим адресам будет выполняться только с самого графического адаптера. Заметим, что если установлен бит `VGA Enable`, то через мост пойдут и транзакции чтения, поскольку адреса регистров палитр входят в диапазон общих адресов портов VGA. Реализация слежения может возлагаться и на графическую карту PCI. Для этого она во время записи в регистр палитр фиксирует данные, но сигналы квитирования `DEVSEL#` и `TRDY#` не вырабатывает, в результате мост распространяет этот неопознанный запрос на шину ISA.

Транслирование транзакций и буферизация

Транслирование транзакций — довольно сложная задача моста, и от способа ее решения зависит производительность системы в целом. Какие именно транзакции следует транслировать с одного интерфейса на другой, решает вышеописанная часть моста, занимающаяся маршрутизацией. При транслировании транзакции мост, как целевое устройство PCI, сразу отвечает ее инициатору, независимо от того, что происходит на другой стороне. Это позволяет мосту, как любому устройству PCI, соблюдать ограничения на время отклика и выполнения транзакций. Далее, мост запрашивает управление шиной на противоположной стороне и, получив управление, проводит эту транзакцию от своего имени. Если транслируется транзакция чтения, то мост должен принять ее результаты, чтобы далее переслать их истинному инициатору транзакции. Этот общий сценарий для различных команд реализуется по-разному, но «при всем богатстве выбора» у моста PCI есть всего два варианта ответа инициатору:

- отложить транзакцию, ответив условием *Retry*. Этот вариант называется *отложенной транзакцией* (delayed transaction); он заставит инициатора через некоторое время повторить попытку данной транзакции. За это время мост должен «провернуть» заказанную транзакцию на другой стороне интерфейса;
- сделать вид, что транзакция успешно завершена. Такой вариант, называемый *отправленной записью* (*posted write*), возможен только для операций записи в память. Реальная запись произойдет позже, когда мост сумеет получить управление шиной на противоположном интерфейсе.

Мост PCI-X для транзакций, транслируемых с шины, работающей в режиме PCI-X, должен вместо откладывания транзакции выполнять ее расщепление.

Чтобы ускорить выполнение транзакций, пришедших с первичной шины, мосту выгодно парковать вторичную шину на самого себя: если вторичная шина свободна, при трансляции транзакции мост не будет терять время на захват шины.

Отложенные транзакции

Отложенные транзакции выполняются мостом PCI для всех команд обращения к вводу-выводу и конфигурационным регистрам, а также всех разновидностей чтения памяти. Отложенные транзакции выполняются в три фазы:

- запрос транзакции инициатором (обмена данными с целевым устройством еще нет);
- завершение целевым устройством;
- завершение инициатором.

Для того чтобы отложить транзакцию, мост должен поставить в очередь *отложенный запрос* (*delayed request*) и сигналом STOP# ввести условие *Retry*. На этом первая фаза отложенной транзакции завершается. В запросе содержатся зафиксированные значения адреса, команды, разрешенных байт, линий четности (и линии REQ64# для 64-битных шин); для отложенных транзакций записи нужно сохранить еще и данные. Этой информации мосту достаточно, чтобы инициировать транзакцию на противоположном интерфейсе, — вторую фазу отложенной транзакции. Ее результатом будет преобразование в очереди отложенного запроса в *отложенное завершение* (*delayed completion*) — запрос вместе с ответом в виде состояния (и запрошенных данных чтения). Изначальный инициатор транзакции, получив условие *Retry*, должен через некоторое время повторить запрос, причем в точности совпадающий с первоначальным (иначе он мостом будет воспринят как новый). Если к этому моменту мост успел отработать данную транзакцию, то этот повторный запрос будет завершен нормальным образом (или отвергнут, если так ответило целевое устройство), а мост удалит отложенное завершение из своей очереди. Если транзакция еще не отработана, то мост снова запросит повтор, и инициатор должен будет повторять свой запрос до тех пор, пока мост не обеспечит нормального завершения. Это будет третьей, заключительной фазой отложенной транзакции.

Инициатор, получивший условие *Retry*, обязан в точности повторить тот же запрос транзакции, иначе мост будет накапливать невостребованные ответы. Конечно, и мост должен отслеживать невостребованные транзакции и через некоторое время (2^{10} или 2^{15} тактов шины, в зависимости от значений в регистре `Bridge Control`) удалять их из своей очереди, чтобы не переполнить ее по «забывчивости» инициатора.

Откладывание транзакций мостом заметно увеличивает время исполнения каждой из них (с точки зрения инициатора), однако оно позволяет обслуживать множество транзакций, находящихся в очередях мостов. В результате достигается увеличение суммарного объема произведенных транзакций на всех шинах PCI за единицу времени, то есть повышается производительность системы в целом. Мост, как держатель очереди транзакций, в принципе, может одновременно выполнять

две транзакции, каждую на своем интерфейсе. Если бы транзакции не откладывались, а выполнялись непосредственно, то инициатору пришлось бы держать свою шину до тех пор, пока не освободится шина назначения (и все промежуточные шины, если транзакция проходит через несколько мостов). В результате число бесполезных тактов ожидания на всех шинах было бы недопустимо велико.

Транслируя транзакции чтения памяти (как отложенные запросы), мост в некоторых случаях может использовать *предвыборку* (*prefetch*, чтение про запас) с целью ускорения работы с памятью. Выполняя предвыборку, мост рискует считать из источника данных больше, чем инициатор заберет от него в данной транзакции. В фазе завершения транзакции инициатором лишние данные в буфере моста проще всего аннулировать, поскольку до возможного последующего востребования в их реальном источнике они могут быть уже модифицированы. Более сложный мост может отслеживать и эти изменения, аннулируя лишь модифицированные данные. Обращения командами обычного чтения памяти разрешают мосту считать только точно затребованное количество данных. При этом возможности ускорения передач меньше, но не возникнет побочных эффектов от лишних чтений. Лишние чтения совершенно недопустимы для регистров ввода-вывода, отображенных на память. Например, чтение управляющих регистров может изменять их состояние; лишнее (с неиспользуемым результатом) чтение регистров данных может привести к потере данных. Мост может смело выполнять предвыборку, когда отрабатывает запросы с командами чтения строк и множественного чтения, транслируемые в любом направлении. Применивший эти команды мастер отвечает за то, что для адресованных областей предвыборка допустима. Если у моста есть регистры, описывающие предвыбираемую память, то транзакции с командами простого чтения с первичного интерфейса, обращенные к предвыбираемой памяти на вторичном интерфейсе, при трансляции могут преобразовываться в команды чтения строк или множественного чтения. Мост может предположить также, что все транзакции к памяти с вторичного интерфейса имеют устройством назначения оперативную память и, следовательно, допускают предвыборку. Однако мост должен иметь специальный бит, запрещающий преобразование команд и предвыборку по этому предположению («слепая» предвыборка может порождать проблемы взаимодействия ПО и аппаратуры).

Отправленные записи

Для транзакций записи в память, находящуюся по другую сторону моста, мост должен производить *отправленную запись* (*posted write*). При этом адрес и данные записи принимаются в буферы моста, и для инициатора транзакция завершится раньше, чем данные дойдут до реального получателя. Мост выполнит их доставку в удобное для другой стороны время, причем эта доставка может выполняться и не за одну транзакцию, инициированную уже мостом. Конечно, если мост не успевает освободить свои буферы отправленной записи (их размер ограничен), то ему придется на некоторые транзакции записи в память отвечать условием «повтор» (*Retry*). Однако это не будет отложенной транзакцией — запросы на запись в па-

мять в очередь отложенных транзакций мост не ставит. Для отправленных записей у моста имеются отдельные буферы. Отправленная запись в общем случае применима только к памяти. Записи в порты ввода-вывода отправлять имеет право только главный мост, и то только для транзакций, инициированных центральным процессором. Запись в конфигурационное пространство отправлена быть не может.

Мосты могут преобразовывать транслируемые ими отправленные записи с целью оптимизации пропускной способности шины и эффективности всей системы. Мост может, например, одну длинную пакетную транзакцию обычной записи в память *MW* (*Memory Write*) блока, не выровненного по границам строк кэша, разбить на три: *MW* от начала до ближайшей границы строки, *MWI* (*Memory Write Invalidate*, запись с инвалидацией) с одной или несколькими целыми строками кэша и *MW* от последней границы строки до конца блока. Кроме того, несколько последовательных транзакций записи могут объединяться в одну пакетную, в которой лишние записи могут блокироваться с помощью сигналов разрешения байтов. Например, последовательность одиночных записей двойных слов по адресам 0, 4, Ch может быть *скомбинирована* (*write combining*) в один пакет с начальным адресом 0, а во время третьей фазы данных (когда предполагается нетребуемый адрес 8) все сигналы `C/BE[3:0]#` будут пассивны. Записи отдельных байтов в определенных случаях могут быть *объединены* (*byte merging*) в одну транзакцию, это допустимо для предвыбираемой памяти. Так, например, последовательность записей байтов по адресам 3, 1, 0 и 2 может быть объединена в одну запись двойного слова, поскольку эти байты принадлежат одному адресуемому двойному слову. Комбинирование и объединение могут работать независимо (объединенные транзакции могут комбинироваться), однако эти преобразования не изменяют порядок следования физических записей в устройства. Наличие этих возможностей не обязательно — оно зависит от «ловкости» мостов. Цель преобразований — сократить число отдельных транзакций (каждая имеет по крайней мере одну «лишнюю» фазу адреса) и, по возможности, фаз данных. Однако мост не имеет права коллапсировать записи (*write collapsing*): если к нему поступает две и более отправленных записи с одинаковым стартовым адресом, он должен все их отработать.

Устройства PCI должны нормально отрабатывать комбинированные записи — если устройство не допускает комбинирования, оно неправильно спроектировано. Если устройство не допускает объединения байтов, то оно в описании своей памяти должно иметь обнуленным бит `Prefetchable`.

Особенности мостов PCI-X

Протокол шины PCI-X обеспечивает мостам возможность более эффективной работы. Детерминированность длины транзакции позволяет мосту более эффективно планировать трансляцию транзакций. К размеру буферов мостов предъявляются особые требования: для каждого типа очередей буферы должны вмещать не менее двух строк кэша. По отношению к мостам PCI мост PCI-X имеет ряд особенностей, отмеченных далее.

Интерфейсы моста PCI-X могут работать как в режиме PCI, так и PCI-X (Mode 1 или Mode 2). Мост должен определить возможности самого слабого устройства на своем вторичном интерфейсе и перевести эту шину (все устройства) в соответствующий режим (по протоколу и частоте синхронизации).

В случае соединения шин PCI и PCI-X мосту приходится преобразовывать некоторые команды, а также преобразовывать протокол. При трансляции транзакции с PCI на PCI-X мосту приходится формировать атрибуты транзакции. Для них номер шины берется из регистров моста, номера устройства и функции устанавливаются нулевыми. Значение счетчика байтов для команд обращения к памяти мост может «придумать» исходя из команды (для чтения и записи строк кэша длину можно вычислить из длины строки) или адреса (определить возможность предвыборки).

Все одиночные (*DWORD*) транзакции, а также все пакетные чтения с шины PCI-X, адресуемые за мост, завершаются мостом как расщепленные транзакции (а не отложенные, как в PCI). Это более выгодное использование шины, поскольку инициатору (запросчику) транзакции не нужно периодически повторять запрос — ответ придет к нему сам, по мере своей готовности. Все пакетные записи в память отрабатываются как отправленные записи. Конечно, если у моста заполнены буферы запросов, то ему придется отложить транзакцию (условием *Retry*).

Порядок выполнения операций и синхронизация

Механизмы отправленных записей и отложенных транзакций нацелены на по возможности одновременное выполнение множества операций обмена в системе шин PCI. Каждый мост имеет буферы и очереди отправленных записей и отложенных транзакций для команд, транслируемых в обоих направлениях. При этом мост одновременно может выполнять обмены данными на обоих своих интерфейсах, будучи как инициатором, так и играя роль целевого устройства. Возникает вопрос о порядке выполнения транзакций, причем речь идет именно о порядке завершений (фаз, в которых происходит взаимодействие с конечным целевым устройством). Мосты подчиняются следующим основным правилам:

- *отправленные записи*, проходящие через мост в одном направлении, завершаются в устройстве назначения в том же порядке, что и на шине инициатора;
- *транзакции записей*, идущие через мост во встречных направлениях, по порядку друг с другом не увязываются;
- *транзакция чтения* выталкивает из моста все записи, отправленные с той же стороны до ее прихода. Перед тем как эта транзакция завершится на стороне ее инициатора (перед третьей фазой отложенной транзакции), она выталкивает из моста и все записи, отправленные с противоположной стороны до завершения данного чтения конечным целевым устройством. Таким образом, сохраняется очередность операций записи и чтения;
- *мост* (как целевое устройство) не будет принимать для посылки транзакцию записи в память до тех пор, пока он не завершит неблокированную транзакцию как ведущее устройство на той же шине.

Мосты сами по себе не предпринимают никаких действий для синхронизации транзакций и запросов прерываний. В то время как транзакции буферируются (могут на некоторое время «застрять» в очередях мостов), сигналы запросов прерывания (INTx#) транслируются мостом совершенно прозрачно (мост просто электрически соединяет эти линии первичного и вторичного интерфейса). Для корректной работы ПО с устройствами в общем случае требуется, чтобы все данные, посланные до выдачи сигнала прерывания, дошли до своих получателей. Для этого нужно разгрузить все буферы всех мостов, находящихся между устройством, выдавшим запрос прерывания, и его конечными партнерами по транзакциям. Программно этого легко достичь чтением любого регистра устройства — чтение через мост выгружает буферы. Возможен и аппаратный вариант: до посылки сигнала прерывания устройство выполняет чтение последних записанных им данных. С прерываниями MSI дело обстоит проще: сообщение MSI не может обогнать данные, ранее посланные этим устройством.

Одной из особенностей применения шины PCI с ее мостовыми соединениями является возможность действительно одновременного выполнения более одного обмена данными по непересекающимся путям — *Concurrent PCI Transferring* или *PCI Concurrency*. Например, во время взаимодействия процессора с памятью ведущее устройство шины PCI может обмениваться данными с другим устройством PCI. Этот пример одновременности обмена скорее теоретический, поскольку ведущее устройство шины PCI, как правило, обменивается данными с системной памятью. Более интересный случай — обмен графического адаптера, подключенного к порту AGP («родственнику» PCI, см. главу 7), с памятью одновременно с обменом процессора с устройством PCI или, наоборот, загрузка данных процессором в графический адаптер одновременно с обменом между ведущим устройством шины PCI и системной памятью. Одновременность требует довольно сложной логики централизованного арбитража запросов всех агентов системы и различных ухищрений в буферизации данных. Одновременность реализуется не всеми чипсетами (в описаниях она всегда специально подчеркивается) и может быть запрещена настройками CMOS Setup.

ГЛАВА 5

Конфигурирование и BIOS устройств PCI и PCI-X

В шину PCI изначально заложены возможности автоматического конфигурирования системных ресурсов (пространств памяти и ввода-вывода и линий запроса прерываний). Автоматическое конфигурирование устройств (выбор адресов и прерываний) поддерживается средствами BIOS и ОС; оно ориентировано на технологию PnP. Стандарт PCI определяет для каждой функции конфигурационное пространство размером до 256 регистров (8-битных), не приписанных ни к пространству памяти, ни к пространству ввода-вывода. Доступ к ним осуществляется по специальным командам шины *Configuration Read* и *Configuration Write*, вырабатываемым с помощью одного из аппаратно-программных механизмов, описанных далее. В этом пространстве есть области, обязательные для всех устройств, и специфические. Конкретное устройство может иметь регистры не во всех адресах, но должно поддерживать нормальное завершение для адресуемых к ним операций. При этом чтение несуществующих регистров должно возвращать нули, а запись выполняться как холостая операция.

Конфигурационное пространство функции начинается со *стандартного заголовка*, в котором содержатся идентификаторы производителя, устройства и его класса, а также описание требуемых и занимаемых системных ресурсов. Структура заголовка стандартизована для обычных устройств (тип 0), мостов PCI-PCI (тип 1), мостов PCI-CardBus (тип 2). Тип заголовка определяет расположение общеизвестных регистров и назначение их бит. После заголовка могут располагаться регистры, специфичные для устройства. Для *стандартизованных свойств* (capability) устройств (например, управления энергопотреблением) в конфигурационном пространстве имеются блоки регистров известного назначения. Эти блоки организуются в цепочки, на первый такой блок есть ссылка в стандартном заголовке (`CAP_PTR`); в первом же регистре блока есть ссылка на следующий блок (или 0, если данный блок — последний). Таким образом, просмотрев цепочку, конфигурационное ПО получает список всех доступных свойств устройства и их позиций в конфигурационном пространстве функции. В PCI 2.3 определены следующие идентификаторы `CAP_ID`, часть из которых рассматривается в данной книге:

- 01 — управление энергопотреблением;
- 02 — порт AGP;

- 03 — *VPD* (Vital Product Data), данные, дающие исчерпывающее описание аппаратных (возможно, и программных) свойств устройств;
- 04 — нумерация слотов и шасси;
- 05 — прерывания MSI;
- 06 — *Hot Swap*, горячее подключение для Compact PCI;
- 07 — протокольные расширения PCI-X;
- 08 — зарезервировано для AMD;
- 09 — на усмотрение производителя (Vendor Specific);
- 0Ah — отладочный порт (Debug Port);
- 0Bh — *PCI Hot Plug*, стандартное обеспечение «горячего подключения».

В PCI-X для устройств Mode 2 конфигурационное пространство расширено до 4096 байт; в расширенном пространстве могут присутствовать расширенные описания свойств (см. ниже).

После аппаратного сброса (или при включении питания) устройства PCI не отвечают на обращения к пространству памяти и ввода-вывода, они доступны только для операций конфигурационного считывания и записи. В этих операциях устройства выбираются по индивидуальным сигналам `IDSEL`, чтением регистров конфигурационное ПО узнает о потребностях в ресурсах и возможных вариантах конфигурирования устройств. После распределения ресурсов, выполняемого программой конфигурирования (во время теста POST или при загрузке ОС), в конфигурационные регистры устройства записываются параметры конфигурирования (базовые адреса). Только после этого устройствам (точнее, функциям) устанавливаются биты, разрешающие им отвечать на команды обращения к памяти и портам ввода-вывода, а также самим управлять шиной. Для того чтобы всегда можно было найти работоспособную конфигурацию, все ресурсы, занимаемые картами, должны быть перемещаемыми в своих пространствах. Для многофункциональных устройств каждая функция должна иметь собственное конфигурационное пространство. Устройство может одни и те же регистры отображать и на память, и на пространство ввода-вывода. При этом в их конфигурационных регистрах должны присутствовать оба описателя, но драйвер должен использовать только один способ обращения (предпочтительно через память).

В заголовке конфигурационного пространства описываются потребности в адресах трех типов:

- *регистры в пространстве ввода-вывода* (*I/O Space*));
- *регистры ввода-вывода, отображенные на память* (*Memory Mapped I/O*). Это область памяти, обращения к которой должны производиться в строгом соответствии с тем, что запрашивает инициатор обмена. Обращение к этим регистрам может изменять внутреннее состояние периферийных устройств;
- *память, допускающая предвыборку* (*Prefetchable Memory*). Это область памяти, «лишнее» чтение которой (с неиспользуемыми результатами) не приводит к

побочным эффектам, все байты считываются независимо от сигналов BE[3:0]#, и записи отдельных байтов мостом могут быть объединены (то есть это память в чистом виде).

Потребности в адресах указываются в *регистрах базовых адресов* — BAR (Base Address Register). Конфигурирующая программа может определить и размеры требуемых областей. Для этого после аппаратного сброса она должна считать и сохранить значения базовых адресов (это будут адреса по умолчанию), записать в каждый регистр FFFFFFFFh и снова считать их значение. В полученных словах нужно обнулить биты декодирования типа (биты [3:0] для памяти и биты [1:0] для ввода-вывода), инвертировать и инкрементировать полученное 32-битное слово — результатом будет длина области (для портов биты [31:16] игнорировать). Метод подразумевает, что длина области выражается числом 2^n и область выровнена естественным образом. Стандартный заголовок вмещает до 6 регистров базового адреса, но при использовании 64-битной адресации число описываемых блоков сокращается. Неиспользуемые регистры BAR при чтении всегда должны возвращать нули.

В PCI имеется *поддержка старых* (*legacy*) *устройств* (VGA, IDE), которые сами себя таковыми объявляют по коду класса в заголовке. Их традиционные (фиксированные) адреса портов не заявляются в конфигурационном пространстве, но как только устанавливается бит разрешения обращения к портам, устройствам разрешается ответ и по этим адресам.

Конфигурационное пространство обычных устройств (тип 0)

Формат заголовка конфигурационного пространства приведен на рис. 5.1, серым цветом здесь выделены поля, обязательные для всех устройств; регистры, специфичные для устройства, могут занимать адреса конфигурационного пространства в пределах 40–FFh.

Перечисленные ниже *поля идентификации* допускают только чтение:

- Device ID — *идентификатор устройства*, назначаемый производителем;
- Vendor ID — *идентификатор производителя* микросхемы PCI, назначенный PCI SIG. Идентификатор FFFFh является недопустимым; это значение должно возвращаться при чтении конфигурационного пространства несуществующего устройства;
- Revision ID — *версия продукта*, назначенная производителем. Используется как расширение поля Device ID;
- Header Type — *тип заголовка* (биты [6:0]), определяющий формат ячеек в диапазоне 10–3Fh и несущий признак многофункционального устройства (если бит 7 установлен). На рисунке приведен формат заголовка типа 0, относящийся

именно к устройствам PCI. Тип 01 относится к мостам PCI-PCI; тип 02 относится к мостам для CardBus;

- `Class Code` — *код класса*, определяющий основную функцию устройства, а иногда и его программный интерфейс (см. далее). Старший байт (адрес 0Bh) определяет *базовый класс*, средний — *подкласс*, младший — *программный интерфейс* (если он стандартизован).

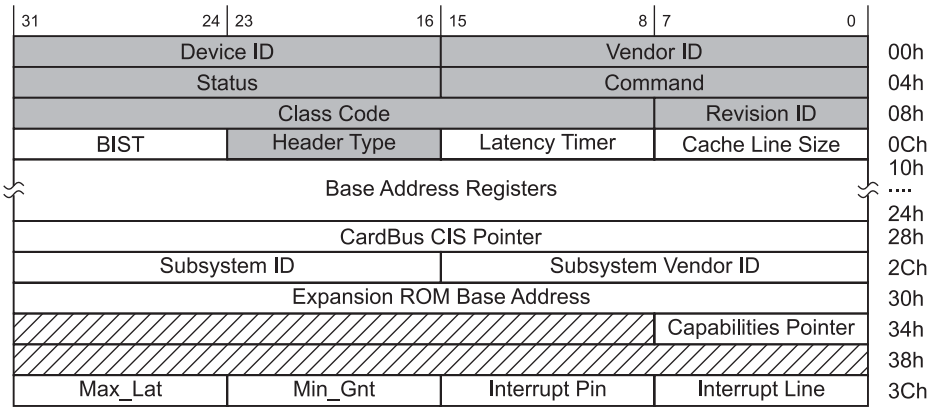


Рис. 5.1. Формат заголовка типа 0 конфигурационного пространства устройства PCI

Остальные поля заголовка являются *регистрами устройств*, допускающими как запись, так и чтение.

Регистр команд `Command` (RW) служит для управления поведением устройства на шине PCI. Регистр допускает как запись, так и чтение. После аппаратного сброса все биты регистра (кроме специально оговоренных исключений) обнулены. Назначение бит регистра команд:

- бит 0 — `IO Space` — разрешение ответа на обращения к пространству ввода-вывода;
- бит 1 — `Memory Space` — разрешение ответа на обращения к пространству памяти;
- бит 2 — `Bus Master` — разрешение работы инициатором (в режиме прямого управления шиной); игнорируется в PCI-X при завершениях расщепленных транзакций;
- бит 3 — `Special Cycles` — разрешение реакции на специальные циклы;
- бит 4 — `Memory Write& and Invalidate enable` — разрешение использовать команды «запись с инвалидацией» при работе инициатором (если бит обнулен, то вместо этих команд должна использоваться обычная запись в память); игнорируется в PCI-X;
- бит 5 — `VGA palette snoop` — разрешение слежения за записью в регистр палитр;
- бит 6 — `Parity Error Response` — разрешение нормальной реакции (вырабатывать сигнал PERR#) на обнаруженную ошибку четности или ECC. Если бит

обнулен, то устройство должно только фиксировать ошибку в регистре состояния и продолжать выполнение транзакции; при ECC-контроле данные об ошибке записываются в регистры ECC;

- бит 7 — `Stepping Control` — возможность пошагового переключения (address/data stepping) линий (если устройство никогда этого не делает, бит регистра «запаян» в «0», если делает всегда — в «1», устройство с такой возможностью по сбросу устанавливает этот бит в «1»). В версии 2.3 и PCI-X бит освобожден (в связи с отменой степинга);
- бит 8 — `SERR# Enable` — разрешение генерации сигнала ошибки SERR# (ошибка адреса сообщается, когда этот бит и бит 6 установлены);
- бит 9 — `Fast Back-to-Back Enable` (необязательный, игнорируется в PCI-X) — разрешение ведущему устройству использовать быстрые смежные обращения к разным устройствам (если бит обнулен, быстрые обращения допустимы лишь для транзакций с одним агентом);
- бит 10 `Interrupt Disable` —, запрет генерации сигнала прерываний по линиям INTx (по аппаратному сбросу и включению питания бит обнулен — прерывания разрешены). Бит определен начиная с PCI 2.3. Ранее был резервным;
- биты [11:15] — резерв.

Регистр `Status` служит для определения *состояния* и *свойств* устройства. Биты, помеченные как RO, допускают только считывание. Другие биты регистра могут быть программно модифицированы операцией записи, с помощью которой можно только обнулять биты, но не устанавливать. При записи в позиции обнуляемых бит устанавливаются единичные значения. Назначение бит регистра состояния:

- биты [0:2] — резерв;
- бит 3— `Interrupt Status` (RO), наличие запроса прерывания. Устанавливается в единицу перед подачей сигнала по линии INTx, независимо от значения бита `Interrupt Disable`. С прерываниями MSI не связан. Бит определен начиная с PCI 2.3, ранее был резервным. В PCI-X 2.0 бит обязателен;
- бит 4 — `Capability List` (RO, необязательный) — признак наличия указателя новых возможностей в регистре со смещением 34h;
- бит 5 — `66 MHz Capable` (RO, необязательный) — признак поддержки частоты 66 МГц;
- бит 6 — резерв;
- бит 7 — `Fast Back-to-Back Capable` (RO, необязательный) — признак поддержки быстрых смежных транзакций (fast back-to-back) с разными устройствами;
- бит 8 — `Master Data Parity Error` (только для устройств с прямым управлением) — инициатор (запросчик) транзакции обнаружил неисправимую ошибку данных;
- биты [10:9] — `DEVSEL Timing` — скорость выборки: 00 — быстрая, 01 — средняя, 10 — низкая (определяет самую медленную реакцию DEVSEL# на все команды, кроме `Configuration Read` и `Configuration Write`);

- бит 11 — `Signaled Target Abort` — устанавливается целевым устройством, когда оно отвергает транзакцию;
- бит 12 — `Received Target Abort` — устанавливается инициатором, когда он обнаруживает отвергнутую транзакцию;
- бит 13 — `Received Master Abort` — устанавливается ведущим устройством, когда оно отвергает транзакцию (кроме специального цикла);
- бит 14 — `Signaled System Error` — устанавливается устройством, подавшим сигнал SERR#;
- бит 15 — `Detected Parity Error` — устанавливается устройством, обнаружившим ошибку данных.

Регистр `Cache Line Size` (RW) служит для задания размера строки кэша (0–128, допустимые значения 2^n, иные трактуются как 0). По этому параметру инициатор определяет, какой командой чтения воспользоваться (обычное чтение, чтение строки или множественное чтение). Ведомое устройство использует этот параметр для поддержки пересечения границ строк при пакетных обращениях к памяти. По сбросу регистр обнуляется.

Регистр `Latency Timer` (RW) задает значение таймера, ограничивающего длину транзакции при снятии сигнала GNT# (см. главу 2). Значение указывается в виде числа тактов шины, часть битов может не допускать изменения (обычно младшие три бита неизменны, так что таймер программируется с дискретностью в 8 тактов).

Регистр `BIST` (RW) служит для управления встроенным самотестированием (Built-In Self Test). Назначение бит регистра:

- бит 7 — возможность BIST;
- бит 6 — запуск теста: запись единицы инициирует тест, по окончании устройство сбрасывает бит (тест должен быть завершен не более чем за 2 с);
- биты [5:4] — резерв (0);
- биты [3:0] — код завершения теста: 0 — тест прошел успешно.

Регистр `Card Bus CIS Pointer` (необязательный) содержит указатель на структуру описателя Card Bus для комбинированного устройства PCI + Card Bus.

Регистр `Interrupt Line` (RW) хранит номер входа контроллера прерывания для используемой линии запроса (0–15 — IRQ0–IRQ15, в системах с APIC может иметь и большее значение; 255 — неизвестный вход или не используется).

Регистр `Interrupt Pin` (RO) задает линию, используемую для запроса прерывания: 0 — не используется, 1 — INTA#, 2 — INTB#, 3 — INTC#, 4 — INTD#, 5–FFh — резерв.

Регистр `Min_GNT` (RO) задает минимальное время, на которое ведущему устройству должно предоставляться управление шиной из расчета на частоту 33 МГц, в интервалах по 0,25 мкс.

Регистр `Max_Lat` (RO) задает максимально допустимую задержку предоставления ведущему устройству доступа к шине, в интервалах по 0,25 мкс (0 — нет специальных требований).

Регистры `Subsystem ID` (RO, задается производителем) и `Subsystem Vendor ID` (RO, производитель получает в PCI SIG) хранят идентификаторы, позволяющие точно идентифицировать карты и устройства (в системе могут быть установлены несколько карт с совпадающими идентификаторами устройства и производителя `Device ID` и `Vendor ID`). В поле 2Ch ставится идентификатор производителя карты PCI (может совпадать со значением в поле 0, если фирма выпускает и микросхемы, и карты).

Регистр `Capability Pointer` (`CAP_PTR`) содержит указатель на цепочку блоков регистров свойств функции, представленных в конфигурационных регистрах. Каждый блок представляет собой набор регистров, начинающийся с границы двойного слова (в указателе биты [1:0] сброшены). Каждый блок начинается с байта идентификатора типа свойства (`CAP_ID`, определенный PCI SIG), за которым следует указатель на следующий блок (нулевой указатель является признаком конца списка блоков), после чего расположены байты описаний самих свойств. Через `CAP_PTR`, например, отыскиваются регистры управления энергопотреблением (если есть), регистры AGP, некоторые регистры хост-контроллера USB 2.0 и ряд других.

Регистры `Base Address Registers` (BAR) описывают области памяти и портов ввода-вывода. Программными манипуляциями с регистрами можно определить размеры областей. Для областей памяти и портов описания различаются:

- *область памяти* (размером не более 2 Гбайт) описывается следующим образом:
  - бит 0 = 0 — признак памяти;
  - биты [2:1] — тип адресации: 00 — 32-битная адресация, 10 — 64-битная (в этом случае регистр расширяется следующим за ним 4-байтным словом, 64-битная адресация обязательна для PCI-X), 01 и 11 — резерв (01 в прежних версиях предназначались для памяти в пределах первого мегабайта);
  - бит 3 (`Prefetchable`) — признак «настоящей» памяти, то есть допускающей предвыборку;
  - биты [31:4] — базовый адрес памяти;
- *область портов* (размером до 256 байт) описывается следующим образом:
  - бит 0 = 1 — признак области портов;
  - бит 1 = 0 (резерв);
  - биты [31:2] — базовый адрес блока портов.

Регистр `Expansion ROM Base Address` управляет адресацией ПЗУ программной поддержки устройства. Размер ПЗУ определяется так же, как и в регистрах базовых адресов (см. выше). Обращение к ПЗУ возможно лишь при разрешенном использовании памяти (бит 1 в регистре команд). Назначение бит регистра:

- бит 0 — разрешение использования ПЗУ;
- биты [1:10] — резерв;
- биты [11:31] — базовый адрес.

Специальные регистры устройств PCI-X

Устройства PCI-X имеют дополнительные регистры (рис. 5.2), положение которых определяется через список свойств (`Capability ID` = 07). Регистры для ECC-контроля появились только в версии PCI-X 2.0.

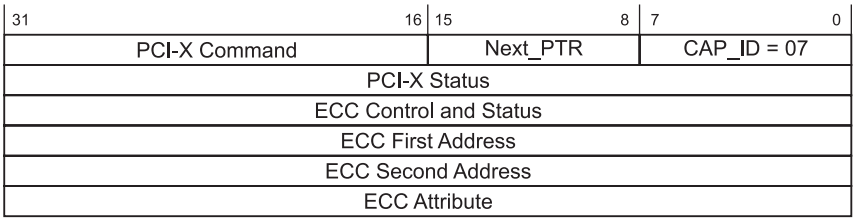


Рис. 5.2. Дополнительные регистры PCI-X

Регистр `PCI-X Command` служит для управления новыми свойствами протокола PCI-X:

- бит 0 (RW) — `Uncorrectable Data Error Recovery Enable`, разрешение попытки восстановления после обнаружения неисправимой ошибки. Если бит не установлен, по обнаружении ошибки четности будет формироваться сигнал SERR#;
- бит 1 (RW) — `Enable Relaxed Ordering`, разрешение установки признака `RO` в атрибутах транзакции;
- биты [3:2] (RW) — `Maximum Memory Read Byte Count`, предел для счетчика байтов в транзакциях чтения памяти: `0` — 512 байт, `1` — 1024, `2` — 2048, `3` — 4096;
- биты [6:4] (RW) — `Maximum Outstanding Split Transactions`, предельное число незавершенных расщепленных транзакций: 0...7 — 1, 2, 3, 4, 8, 12, 16, 32 транзакции соответственно;
- биты [11:7] — резерв;
- биты [13:12] (RO) — версия возможностей PCI-X (поддержка ECC):
  - 00 — ECC не поддерживается;
  - 01 — ECC только в Mode 2;
  - 10 — ECC в Mode 1 и Mode 2.
- биты [15:14] — резерв.

Регистр `PCI-X Status` содержит идентификатор функции — ее адрес в иерархии конфигурационного пространства, который устройство «подсматривает» на шине, при выполнении операции конфигурационной записи. Этот идентификатор требуется устройству для передачи его в фазе атрибутов. Кроме того, в регистре имеются признаки возможностей устройства, а также индикаторы ошибок, связанных с расщепленными транзакциями. Назначение битов регистра `PCI-X Status`:

- биты [2:0] (RO) — `Function Number`, номер функции;

- биты [7:3] (RO) — Device Number, номер устройства, который оно узнает по значению AD[15:11] в фазе адреса конфигурационной записи, обращенной к данному устройству, выбранному линией IDSEL (после сброса устанавливается 1F);
- биты [15:8] (RO) — Bus Number, номер шины, который оно узнает по значению AD[7:0] в фазе атрибутов конфигурационной записи, обращенной к данному устройству (после сброса устанавливается FF);
- бит 16 (RO) — 64-bit Device, признак 64-битной шины AD;
- бит 17 (RO) — 133 MHz Capable, признак поддержки частоты 133 МГц (иначе 66 МГц);
- бит 18 (RWC) — Split Completion Discarded, признак отброшенного завершения расщепленной транзакции (запросчик его отверг);
- бит 19 (RWC) — Unexpected Split Completion, признак неожиданного получения завершения расщепленной транзакции;
- бит 20 (RO) — Device Complexity, признак сложного устройства (моста);
- биты [22:21] (RO) — Designed Maximum Memory Read Byte Count, максимальное число байт в последовательности, инициируемой устройством (его возможности): 0 — 512 байт, 1 — 1024, 2 — 2048, 3 — 4096;
- биты [25:23] (RO) — Designed Maximum Outstanding Split Transactions, предельное число незавершенных расщепленных транзакций: 0…7 — 1, 2, 3, 4, 8, 12, 16, 32 транзакции;
- биты [28:26] (RO) — Designed Maximum Cumulative Read Size, максимальный суммарный объем данных чтения памяти, ожидаемых устройством (запросы отправлены, ответы еще не получены): 0...7 — 8, 16, 32...1024 квантов *ADQ*;
- бит 29 (RWC) — Received Split Completion Error Message, признак получения сообщения об ошибке завершения расщепленной транзакции;
- бит 30 (RO) — PCI-X 266 Capable, поддержка режима PCI-X 266 (Mode 2);
- бит 31 (RO) — PCI-X 533 Capable, поддержка режима PCI-X 533 (Mode 2).

Регистры ECC-контроля служат для управления контролем и диагностики. Регистр ECC Control and Status Register служит для управления ECC-контролем: разрешает ECC в Mode 1 (в Mode 2 ECC обязателен) и разрешает исправление однократных ошибок. В этом же регистре сообщаются признаки обнаружения ошибки, команды и фазы шины, в которой обнаружена ошибка, а также значения синдрома ошибки и атрибутов транзакции. Регистры ECC First Address, ECC Second Address и ECC Attribute Register содержат адрес, при обращении по которому обнаружена ошибка ECC, и атрибуты транзакции.

Расширенное конфигурационное пространство PCI-X

В спецификации PCI-X 2.0 введено расширение конфигурационного пространства одной функции до 4096 байт. При этом стандартный 256-байтный набор регистров

и формат заголовка сохраняется, а дополнительное пространство используется для нужд устройства, включая и размещение описаний дополнительных возможностей. Для доступа к расширенному конфигурационному пространству может использоваться как расширенный вариант механизма 1 (см. далее) с передачей дополнительных 4 бит номера регистра по линиям AD[27:24], так и отображение конфигурационных регистров на адрес памяти. В случае отображения на адрес памяти иерархический адрес конфигурационных регистров всех устройств PCI отображается на биты A[27:0], базовый адрес (A[63:28]) зависит от реализации системы и сообщается операционной системе. Все конфигурационные регистры всех устройств всех шин PCI требуют для отображения в памяти область размером 256 Мбайт. Схема отображения простая и логичная:

- A[27:20] — Bus[7:0], номер шины;
- A[19:15] — Device[4:0], номер устройства;
- A[14:12] — Function[2:0], номер функции;
- A[11:8] — Extended Register [3:0], расширение номера регистра;
- A[7:0] — Register[7:0], номер регистра.

Устройство должно воспринимать и отрабатывать конфигурационные обращения, выполненные любым способом. При этом разработчик устройства должен помнить, что при попадании устройства в систему с обычной шиной PCI программно доступными окажутся лишь первые 256 байт конфигурационного пространства функции, так что в расширенное пространство следует помещать только те регистры, которые не используются в стандартном режиме работы PCI.

Для расширенного пространства введен и новый формат описания свойств с учетом «длинного» (10-битного) адреса регистра. Расширенный список свойств должен начинаться с адреса 100h (или же там должна быть структура, не позволяющая трактовать этот фрагмент как начало цепочки). Каждое свойство начинается с 32-битного идентификатора, за которым располагаются регистры, описывающие данное свойство. 32-разрядный идентификатор расширенных возможностей — PCI Extended Capability ID имеет следующую структуру:

- биты [15:0] — Capability ID, идентификатор свойства;
- биты [19:16] — Capability Version Number, номер версии свойства;
- биты [31:20] — Next Capability Offset, смещение следующего идентификатора (относительно нулевого регистра).

Конфигурационное пространство мостов PCI

Заголовок конфигурационного пространства мостов PCI-PCI приведен на рис. 5.3. Регистры в диапазоне адресов 00–17h полностью совпадают с регистрами обычного устройства PCI и описывают поведение и состояние моста на первичной шине. Заметим, что бит 2 регистра команд (Bus Master Enable) управляет возможно-

стью трансляции транзакций с вторичной шины на первичную. Если этот бит обнулен, то мост не должен на вторичной стороне отзываться как целевое устройство в транзакциях записи/чтения памяти и ввода-вывода, поскольку он не сможет транслировать эти транзакции на первичную шину. Регистры BAR описывают только область специфических (зависящих от реализации) регистров моста, к маршрутизации эти регистры отношения не имеют.

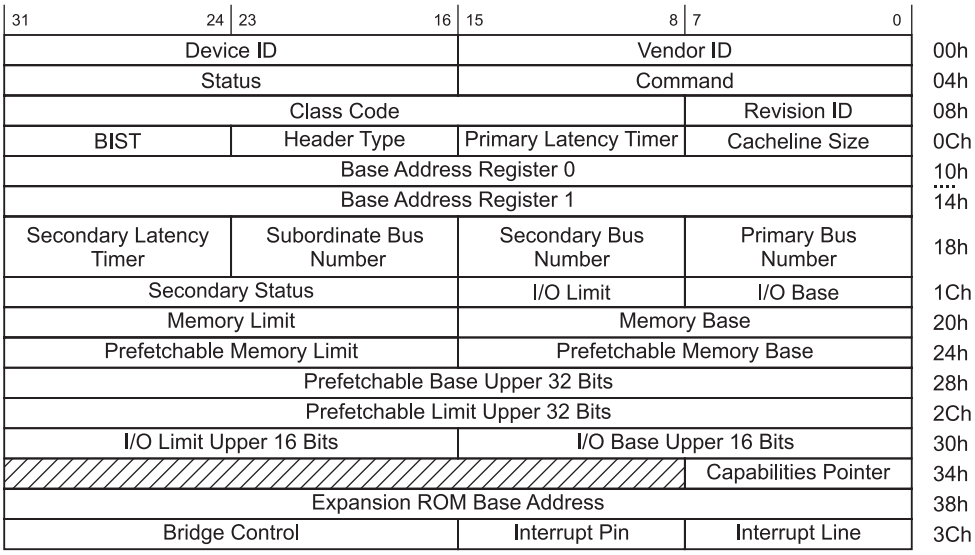


Рис. 5.3. Формат заголовка конфигурационного пространства моста PCI/PCI (заголовок типа 1)

Маршрутизирующие свойства моста определяются следующими регистрами (подробности см. в главе 4):

- `Primary Bus Number` — номер первичной шины;
- `Secondary Bus Number` — номер вторичной шины (это и номер моста);
- `Subordinate Bus Number` — максимальный номер подчиненной шины;
- `I/O Base` и `I/O Limit` — регистры, задающие начальный и конечный адрес пространства ввода-вывода устройств, расположенных за мостом. Эти регистры задают только старшие 4 бита 16-битного адреса ввода-вывода, так что гранулярность выделения адресов составляет 4 Кбайт;
- `I/O Limit Upper 16 Bits` и `I/O Base Upper 16 Bits` — регистры старшей части адреса ввода-вывода, если используется 32-битная адресация ввода-вывода (на это указывают установленные биты 0 регистров `I/O Base` и `I/O Limit`);
- `Memory Base` и `Memory Limit` — регистры, задающие начальный и конечный адрес пространства памяти, на которую отображены регистры ввода-вывода устройств, расположенных за мостом. Эти регистры задают только старшие 12 бит 32-битного адреса памяти, так что гранулярность выделения адресов составляет 1 Мбайт;

- `Prefetchable Memory Base` и `Prefetchable Memory Limit` — регистры, задающие начальный и конечный адрес «настоящей» (допускающей предвыборку) памяти устройств, расположенных за мостом. Эти регистры задают только старшие 12 бит 32-битного адреса памяти, так что гранулярность выделения адресов составляет 1 Мбайт;
- `Prefetchable Base Upper 32 Bits` и `Prefetchable Limit Upper 32 Bits` — регистры старшей части адреса «настоящей» памяти, если используется 64-битная адресация (на это указывают установленные биты 0 регистров `Prefetchable Memory Base` и `Prefetchable Memory Limit`).

Регистр `Secondary Status` аналогичен обычному регистру состояния (`Status`), но его признаки относятся ко вторичной шине. Единственное отличие — бит 14 (`Received System Error`) в `Secondary Status` несет признак обнаружения сигнала SERR# на вторичном интерфейсе, а не его введения данным устройством.

Регистр `Expansion ROM Base Address`, как и для обычного устройства, задает положение ПЗУ расширения BIOS (если это ПЗУ присутствует в мосте).

Регистры `Interrupt Line` и `Interrupt Pin` относятся к прерываниям, вырабатываемым мостом (если таковые имеются). К линиям прерывания, транслируемым мостом, эти регистры отношения не имеют.

Регистр `Bridge Control` служит для управления работой моста и индикации невостребованных завершений отложенных транзакций:

- бит 0 — `Parity Error Response Enable`, разрешение мосту сигнализировать на вторичный интерфейс об обнаружении ошибки четности адреса и данных;
- бит 1 — `SERR# Enable`, разрешение трансляции сигнала SERR# со вторичного интерфейса на первичный (для трансляции должен быть установлен и одноименный бит в регистре команд);
- бит 2 — `ISA Enable`, разрешение поддержки адресации ввода-вывода для шины ISA (вычеркивания последних 768 байт из каждого килобайта диапазона адресов, заданного регистрами `I/O Base` и `I/O Limit`);
- бит 3 — `VGA Enable`, разрешение специальной поддержки VGA;
- бит 4 — резерв;
- бит 5 — `Master-Abort Mode`, поведение моста в случае, когда, транслируя транзакцию, он не получает ответа от целевого устройства: 0 — игнорировать эту ситуацию, возвращая при чтении FF...FFh и отбрасывая данные записи; 1 — сообщать инициатору транзакции условием *Target-Abort*, а если это невозможно (в случае отправленной записи), подавать сигнал SERR#;
- бит 6 — `Secondary Bus Reset`, подача сигнала RST# на вторичный интерфейс (когда бит сброшен, RST# на вторичном интерфейсе вырабатывается по RST# на первичном);
- бит 7 — `Fast Back-to-Back Enable`, разрешение генерации быстрых смежных транзакций на вторичном интерфейсе;
- бит 8 — `Primary Discard Timer`, таймер отбрасывания результатов отложенных транзакций, инициированных мастером с первичного интерфейса: 0 — ожи-

дание 2^{15} тактов шины, 1 — 2^{10}. Отсчет начинается, когда результат отложенной транзакции подходит к голове очереди. Если результат не будет забран мастером (повтором транзакции) за указанное время, результат отбрасывается;

- бит 9 — `Secondary Discard Timer`, таймер отбрасывания результатов отложенных транзакций, инициированных мастером с вторичного интерфейса (аналогично предыдущему);
- бит 10 — `Discard Timer Status`, признак отбрасывания отложенных транзакций на любом интерфейсе;
- бит 11 — `Discard Timer SERR# Enable`, разрешение генерации SERR# (на первичном интерфейсе) по срабатыванию таймера отбрасывания;
- биты [12:15] — резерв.

Регистр `Secondary Latency Timer` управляет поведением моста как мастера на вторичной шине, когда у него отбирают управление шиной (правда, отбирает он сам у себя, поскольку арбитр — часть моста).

Для больших систем с шасси расширения мост может иметь *возможность нумерации шасси и слотов*, для чего он должен иметь свойство с `Capabilities ID` = 04 (рис. 5.4).

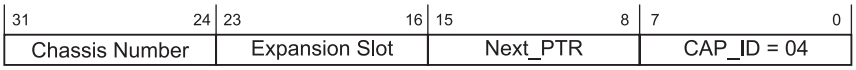


Рис. 5.4. Структура нумерации шасси и слотов

Регистр `Expansion Slot` характеризует положение и вторичную шину моста:

- биты [4:0] — `Expansion Slots Provided`, число слотов на вторичной шине моста;
- бит 5 — `First in Chassis`, признак первого моста в шасси расширения. Указывает и на наличие шасси и, следовательно, использование регистра номера шасси. Если в шасси имеется несколько мостов, то первым является либо мост с минимальным номером первичной шины (остальные будут для него подчиненными), либо с минимальным номером устройства (остальные будут того же ранга, но их вторичные шины будут иметь большие номера);
- биты [7:6] — резерв.

Регистр `Chassis Number` задает номер шасси, в котором находится данный мост (0 — шасси, на котором находится процессор, выполняющий конфигурирование).

Программная генерация конфигурационных и специальных циклов

Поскольку конфигурационное пространство PCI обособлено, в главный мост приходится вводить специальный механизм доступа к нему командами процессора, инструкции которого «умеют» обращаться только к памяти или вводу-выводу. Этот же механизм используется и для генерации специальных циклов. Для PC-совме-

стимых компьютеров предусмотрено два механизма, из которых в спецификации PCI 2.2 оставлен только первый (*Configuration Mechanism #1*) как более прозрачный. Номер механизма, которым пользуется конкретная системная плата, можно узнать путем вызова PCI BIOS. Для доступа к расширенному конфигурационному пространству устройств PCI-X эти механизмы непригодны (доступ к нему возможен только через прямое отображение на память, см. ранее).

Конфигурационные циклы адресуются к конкретному устройству (микросхеме PCI), расположенному на шине с известным номером. Декодированием номера шины и устройства, для которого должен быть сформирован сигнал выборки IDSEL (единичное значение), занимаются мосты. Номер функции и адрес регистра декодируется самим устройством.

Для работы механизма № 1 в пространстве ввода-вывода зарезервированы 32-битные порты с адресами 0CF8h и 0CFCh, входящие в главный мост. Для обращения к конфигурационному пространству в порт CONFIG_ADDRESS (RW, адрес CF8h) заносят 32-разрядный адрес, декодируемый в соответствии с рис. 5.5. После занесения адреса обращением к порту CONFIG_DATA (RW, адрес CFCh) можно прочитать или записать содержимое требуемого конфигурационного регистра. В регистре CONFIG_ADDRESS бит 31 является разрешением формирования конфигурационных и специальных циклов. В зависимости от номера шины, указанного в этом регистре, главный мост генерирует конфигурационные циклы одного из двух типов:

- для обращения к устройству, находящемуся на нулевой шине (подключенной к главному мосту), используется *цикл типа 0* (см. рис. 2.2, *а–в* на с. 51). В этом цикле в фазе адреса на линии AD[31:11] мост помещает позиционный код *выбора устройства*, на AD[10:8] — *номер функции*, на AD[7:2] — *адрес регистра*, а биты 1:0 = 00 являются признаком цикла типа 0. В PCI-X в фазе адреса на линии AD[15:11] помещается номер устройства; расширенное конфигурационное пространство через данный механизм не доступно;
- для обращения к устройству, находящемуся на ненулевой шине, используется *цикл типа 1*. Здесь главный мост передает всю адресную информацию из CONGIG_ADDRESS (номер шины, устройства, функции и регистра) на нулевую шину PCI, обнуляя старшие биты (31:24) и устанавливая в битах 1:0 признак типа «01» (рис. 2.2, *г*).

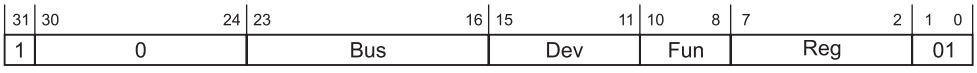


Рис. 5.5. Формат адреса в регистре CONFIG_ADDRESS

Специальный цикл генерируется при записи в CONFIG_DATA, когда в регистре CONGIG_ADDRESS все биты [15:8], единичные, а биты [7:0] — нулевые; номер шины, на которой формируется цикл, задается битами [23:16]. В специальном цикле адресная информация не передается (он широковещательный), но путем задания номера шины можно управлять его распространением. Если хост генерирует специальный цикл, указав нулевой адрес шины, то этот цикл будет выполнен только на

главной шине и всеми остальными мостами распространяться не будет. Если указан ненулевой адрес шины, то главный мост сформирует цикл конфигурационной записи типа 1, который в специальный цикл будет преобразован только мостом на шине назначения. Специальный цикл, генерируемый ведущим устройством шины, действует только на шине этого устройства и не распространяется через мосты. Если требуется сгенерировать этот цикл на другой шине, то ведущее устройство может это сделать посредством записей в регистры CONGIG_ADDRESS и CONFIG_DATA[1].

Для работы устаревшего и неудобного *механизма № 2* в пространстве ввода-вывода зарезервированы два 8-битных порта с адресами 0CF8h и 0CFAh, входящие в главный мост. Этот механизм использует отображение конфигурационного пространства устройств PCI на область C000–CFFF пространства ввода-вывода. Поскольку этой области (4096 портов) недостаточно для отображения конфигурационного пространства всех устройств всех шин PCI, формирование адреса выполняется весьма замысловатым образом. В регистре CSE (Configuration Space Enable) с адресом 0CF8h биты 7:4 являются ключом разрешения отображения, а биты [3:1] несут номер функции, к пространству которой адресуются обращения. Бит 0 (SCE — Special Cycle Enable) при единичном значении вместо конфигурационных циклов вызывает формирование специального цикла. При нулевом ключе область портов C000–CFFFh остается нормальной частью пространства ввода-вывода, а при ненулевом на нее отображаются конфигурационные пространства указанных функций 16 возможных устройств[2]. При обращении к конфигурационному пространству устройств нулевой шины чтение или запись двойного слова в порт по адресу C000–CFFCh генерирует конфигурационный цикл, в котором из адреса порта биты [2:7] поступают на шину AD[2:7] как индекс регистра конфигурационного пространства, а биты [11:8] декодируются в позиционный код выбора устройства (линии IDSEL) на линиях AD[31:16]. Номер функции на линии AD[10:8] поступает из регистра CSE, линии AD[1:0] нулевые. Для обращения к устройствам ненулевой шины служит *регистр перенаправления* (Forward Register) с адресом 0CFAh, в который помещают номер шины (по сбросу этот регистр обнуляется). Если номер шины ненулевой, то генерируется цикл типа 1, в котором номер функции поступает из регистра CSE, младшие 4 бита номера устройства поступают с битов адреса (AD15 = 0), а номер шины — из регистра перенаправления (биты AD[1:0] = = 01 и AD[31:24] = 0 формируются аппаратно).

Для генерации специального цикла по этому механизму в регистре CSE устанавливается ненулевой ключ, номер функции 111 и SCE = 1, после чего выполняется запись по адресу порта CF00h. В зависимости от содержимого регистра перенаправления будет сгенерирован специальный цикл или конфигурационный цикл типа 1, который на целевой шине будет преобразован в специальный.

---

[1] Это наводит на крамольные мысли о возможности конфигурирования устройств PCI не только хостом, пользуясь данным конфигурационным механизмом. Однако такие действия возможны только после конфигурирования мостов, так что начальное конфигурирование остается все же прерогативой хоста.

[2] Логичнее было бы отобразить все функции одного устройства, для чего потребовалась бы даже вдвое меньшая область.

Классификация устройств PCI

Важной частью спецификации PCI является классификация устройств и указание кода класса в его конфигурационном пространстве (3 байта `Class Code`). Старший байт определяет *базовый класс*, средний — *подкласс*, младший — *программный интерфейс* (если он стандартизован). Код класса позволяет идентифицировать наличие определенных устройств в системе, это может быть сделано с помощью PCI BIOS. Для стандартизованных классов и интерфейсов (например, 01:01:80 — контроллер IDE или 07:00:01 — последовательный порт 16450) «заинтересованная» программа может найти требуемое устройство и выбрать подходящий вариант драйвера. Классификатор определяет организация PCI SIG, он регулярно обновляется на сайте http://www.pcisig.com. Нулевые значения полей, как правило, дают самые неопределенные описания. Значение подкласса 80h относится к «иным устройствам». Некоторые классы устройств приведены в табл. 5.1.

Таблица 5.1. Некоторые классы устройств PCI

Подкласс	Интерфейс	Назначение
Базовый класс 00 — устройства, разработанные до принятия классификации		
00	00	Все, кроме VGA-совместимых
01	00	VGA-совместимый графический адаптер
Базовый класс 01 — контроллеры устройств хранения		
00	00	Контроллер шины SCSI
01	xx	Контроллер IDE
02	00	Контроллер НГМД
03	00	Контроллер шины IPI
04	00	Контроллер RAID
Базовый класс 02 — сетевые контроллеры		
00	00	Ethernet
01	00	Token Ring
02	00	FDDI
03	00	ATM
04	00	ISDN
Базовый класс 03 — дисплейные контроллеры		
00	00	Совместимый с VGA (память 0A0000–0BFFFFh, порты 3B0–3BBh и 3C0–3DFh)
00	01	Совместимый с IBM-8514 (порты 2E8h, 2EAh-2EFh)
01	00	Контроллер XGA
02	00	Контроллер 3D

Подкласс	Интерфейс	Назначение
Базовый класс 04 — мультимедийные устройства		
00	00	Видео
01	00	Аудио
02	00	Компьютерная телефония
Базовый класс 05 — контроллеры памяти		
00	00	Контроллер памяти с произвольным доступом (RAM)
01	00	Контроллер флэш-памяти
Базовый класс 06 — мосты		
00	00	Главный мост (Host bridge)
01	00	Мост PCI-ISA
02	00	Мост PCI-EISA
03	00	Мост PCI-MCA
04	00	Мост PCI-PCI
04	01	Мост PCI-PCI с субтрактивным декодированием
05	00	Мост PCI-PCMCIA
06	00	Мост PCI-NuBus
07	00	Мост PCI-CardBus
08	xx	Мост PCI-RACEway
Базовый класс 07 — коммуникационные контроллеры		
00	00	UART, совместимый с 8250
	01	UART, совместимый с 16450
	02	UART, совместимый с 16550
	...	...
	06	UART, совместимый с 16950
01	00	Стандартный LPT-порт
	01	Двунаправленный LPT-порт
	02	Параллельный порт ECP 1.X
	03	Контроллер IEEE 1284
	FEh	Целевое устройство IEEE 1284
02	00	Мультипортовый последовательный контроллер
03	00	Модем
	01	Hayes-модем с интерфейсом 16450
	02	Hayes-модем с интерфейсом 16550
	03	Hayes-модем с интерфейсом 16650
	04	Hayes-модем с интерфейсом 16750

продолжение ⇩

Таблица 5.1 (*продолжение*)

Подкласс	Интерфейс	Назначение
Базовый класс 08 — системная периферия		
00	00	Контроллер прерываний 8259 (PIC)
	01	Контроллер прерываний ISA
	02	Контроллер прерываний EISA
	10	Контроллер прерываний I/O APIC
	20	Контроллер прерываний I/O(x) APIC
01	00	Контроллер DMA 8237
	01	Контроллер DMA ISA
	02	Контроллер DMA EISA
02	00	Системный таймер 8254
	01	Системный таймер ISA
	02	Системный таймер EISA
03	00	Часы (RTC)
	01	Часы (RTC) ISA
04	00	Контроллер горячего подключения PCI
Базовый класс 09 — контроллеры устройств ввода		
00	00	Контроллер клавиатуры
01	00	Дигитайзер (перо)
02	00	Контроллер мыши
03	00	Контроллер сканера
04	00	Игровой порт с фиксированным адресом
	01	Игровой порт с перемещаемым адресом
Базовый класс 0Ah — док-станции		
Базовый класс 0Bh — процессоры		
Базовый класс 0Ch — контроллеры последовательных шин		
00	00	Контроллер IEEE 1394 (FireWire)
	10	Контроллер IEEE 1394 по спецификации OpenHCI
01	00	Контроллер ACCESS.bus
02	00	Контроллер SSA
03	00	Контроллер USB по UHCI
	10h	Контроллер USB по OHCI
	20h	Контроллер USB по EHCI
	FEh	Устройство USB
Базовый класс 0Dh — контроллеры беспроводных интерфейсов		

Подкласс	Интерфейс	Назначение
Базовый класс 0Eh — контроллеры интеллигентного ввода-вывода (I_2O)		
Базовый класс 0Fh — контроллеры спутниковых коммуникаций		
Базовый класс 10h — контроллеры шифрования		

PCI BIOS

Для облегчения взаимодействия с устройствами PCI имеются дополнительные функции BIOS, доступные как из реального, так и защищенного режима работы процессора. Функции PCI BIOS используются только для поиска и конфигурирования устройств PCI — процедур, требующих доступа к их конфигурационному пространству. Функции приходится поддерживать и использовать потому, что циклы конфигурационных обращений, как и специальный цикл, выполняются специфическим образом. Кроме того, PCI BIOS позволяет управлять коммутатором запроса прерываний (PCI Interrupt Steering), скрывая специфический программный интерфейс чипсета системной платы. Остальное взаимодействие с устройствами через их пространства памяти и ввода-вывода, а также обработка прерываний в поддержке со стороны BIOS не нуждаются, поскольку выполняются непосредственно командами процессора и не зависят от платформы (чипсета системной платы). Регулярная работа с этими устройствами выполняется через обращения к регистрам устройств по адресам, полученным при конфигурировании, и обработку известных номеров прерываний от этих устройств. Функция проверки наличия PCI BIOS позволяет определить доступные конфигурационные механизмы, и, зная их работу, программа в дальнейшем может обходиться и без вызовов PCI BIOS.

Программы с помощью функций PCI BIOS могут искать интересующие их устройства по идентификаторам или кодам класса. Если стоит задача полного «переучета» установленных устройств, то она решается чтением конфигурационной информации по всем функциям всех устройств всех шин — это быстрее, чем перебирать все возможные сочетания идентификаторов или классов кодов. Для найденных устройств программы должны определять реальные настройки чтением регистров конфигурационного пространства, учитывая возможность перемещения ресурсов по всему пространству и даже между пространствами памяти и ввода-вывода.

Для *16-битного интерфейса* реального режима, V86 и 16-битного защищенного режима, функции PCI BIOS вызываются через прерывание `Int 1Ah`; номер функции задается при вызове в регистре `AX`. Возможна и программная имитация прерывания дальним вызовом по физическому адресу 000FFE6Eh (стандартная точка входа в обработчик `Int 1Ah`) с предварительным занесением в стек регистра флагов.

Для 32-разрядных вызовов защищенного режима все эти же функции вызываются через точку входа, найденную через каталог 32-разрядных сервисов (см. ниже), при этом назначение входных и выходных регистров и флага CF сохраняется. До использования 32-разрядного интерфейса следует сначала найти его каталог и убедиться в наличии сервисов PCI BIOS по идентификатору «$PCI» (049435024h).

Вызовы требуют глубокого стека (до 1024 байт). Признаком нормального выполнения является CF = 0 и AH = 0; при CF = 1 AH содержит *код ошибки*:

- 81h — неподдерживаемая функция;
- 83h — неправильный идентификатор производителя;
- 86h — устройство не найдено;
- 87h — неправильный номер регистра PCI;
- 88h — установка не удалась;
- 89h — слишком маленький буфер для данных.

Функции PCI BIOS перечислены ниже:

AX = *B101h — проверка присутствия PCI BIOS*. При наличии PCI BIOS возвращает CF = 0, AH = 0 и EDX = 20494350h (строка символов «PCI»); проверяться должны все три признака. При этом в AL находится описатель аппаратного механизма доступа к конфигурационному пространству и генерации специальных циклов PCI:

- бит 0 — поддержка механизма № 1 для доступа к конфигурационному пространству;
- бит 1 — поддержка механизма № 2 для доступа к конфигурационному пространству;
- биты [2:3] = 00 (резерв);
- бит 4 — поддержка генерации специального цикла с использованием механизма № 1;
- бит 5 — поддержка генерации специального цикла с использованием механизма № 2;
- биты [6:7] = 00 (резерв).

В регистрах BH и BL возвращается старший и младший номер версии (BCD-цифры), в CL — максимальный номер шины PCI, присутствующий в системе (число шин – 1, поскольку они нумеруются с нуля последовательно). В регистре EDI может возвращаться линейный адрес точки входа 32-разрядных сервисов BIOS. Этот адрес возвращается не всеми версиями BIOS (некоторые не изменяют EDI); для проверки можно при вызове обнулять EDI и проверять на нуль возвращенное значение.

- AX = *B102h — поиск устройства по идентификатору*. При вызове в CX указывается идентификатор устройства, в DX — идентификатор производителя, в SI — индекс (порядковый номер) устройства. При успешном возврате в BH — номер шины, в BL[7:3] — номер устройства, BL[2:0] — номер функции. Для нахождения всех устройств с указанными идентификаторами вызовы выполняют, последовательно инкрементируя SI от 0 до получения кода возврата 86h.

- AX = *B103h — поиск устройства по коду класса*. При вызове в ECX[23:16] указывается код класса, в ECX[15:8] — подкласса, в ECX[7:0] — интерфейс, в SI — индекс устройства (аналогично предыдущему). При успешном возврате в BH — номер шины, в BL[7:3] — номер устройства, BL[2:0] — номер функции.
- AX = *B106h — генерация специального цикла PCI*. При вызове в BL указывается номер шины, в EDX — данные специального цикла.
- AX = *B108h — чтение байта из конфигурационного пространства устройства PCI*. При вызове в BH — номер шины, в BL[7:3] — номер устройства, BL[2:0] — номер функции, в DI — номер регистра (0–FFh). При успешном возврате в CL — считанный байт.
- AX = *B109h — чтение слова из конфигурационного пространства устройства PCI*. При вызове в BH — номер шины, в BL[7:3] — номер устройства, BL[2:0] — номер функции, в DI — номер регистра (0–FFh, четный). При успешном возврате в CX — считанное слово.
- AX = *B10Ah — чтение двойного слова из конфигурационного пространства устройства PCI*. При вызове в BH — номер шины, в BL[7:3] — номер устройства, BL[2:0] — номер функции, в DI — номер регистра (0–FFh, кратный 4). При успешном возврате в ECX — считанное двойное слово.
- AX = *B10Bh — запись байта в конфигурационное пространство устройства PCI*. При вызове в BH — номер шины, в BL[7:3] — номер устройства, BL[2:0] — номер функции, в DI — номер регистра (0–FFh), в CL — записываемый байт.
- AX = *B10Ch — запись слова в конфигурационное пространство устройства PCI*. При вызове в BH — номер шины, в BL[7:3] — номер устройства, BL[2:0] — номер функции, в DI — номер регистра (0–FFh, четный), в CX — записываемое слово.
- AX = *B10Dh — запись двойного слова в конфигурационное пространство устройства PCI*. При вызове в BH — номер шины, в BL[7:3] — номер устройства, BL[2:0] — номер функции, в DI — номер регистра (0–FFh, кратный 4), в ECX — записываемое двойное слово.
- AX = *B10Eh — определение возможностей назначения прерываний* (GET_IRQ_ROUTING_OPTIONS). При вызове BX=0, ES:EDI указывает на структуру параметров буфера для результата, состоящую из слова с длиной буфера, за которым располагается дальний указатель на его начало. DS в 16-разрядом режиме указывает на сегмент с физическим адресом F0000, в 32-разрядном определяется правилами из следующего раздела. При успешном возврате в BX находится битовая карта запросов IRQx, в которой единичное значение бита означает, что данный вход контроллера прерываний используется исключительно шиной PCI. В буфер помещается последовательный набор структур, описывающих возможности и назначение прерываний для каждого устройства PCI (табл. 5.2). При возврате в структуре параметров буфера возвращается его реальная длина; если при вызове указан буфер, не вмещающий весь результат, устанавливается код ошибки 89h.
- AX = *B10Fh — назначение линий запроса прерываний* (SET_PCI_IRQ). При вызове в BH задается номер шины, в BL — номер устройства (биты [7:3]) и функции (биты

[2:0]), для которой назначается запрос; в CL указывается вывод (0Ah — INTA#,... 0Dh — INTD#), в CH — желаемый номер IRQx (0...0Fh, причем 0 соответствует отключению INTx# от входов контроллера). Значение DS аналогично предыдущей функции. Если заказанное назначение невозможно, при возврате устанавливается код ошибки 88h. При использовании данной функции следует выполнять и сопутствующие изменения в конфигурационных регистров всех затрагиваемых устройств и их функций (см. главу 3).

Таблица 5.2. Описание опций прерываний для одного устройства PCI

Смещение	Размер	Назначение
0	byte	PCI Bus number — номер шины PCI
1	byte	PCI Device number — номер устройства PCI
2	byte	Назначенная связь для линии INTA# (0 — нет, 1 — IRQ1, ...0Fh — IRQ15)
3	word	Битовая карта возможных назначений для INTA# (бит 0 — IRQ0, ... бит 15 — IRQ15)
5	byte	Назначенная связь для линии INTB# (аналогично)
6	word	Битовая карта возможных назначений для INTB# (аналогично)
8	byte	Назначенная связь для линии INTC# (аналогично)
9	word	Битовая карта возможных назначений для INTC# (аналогично)
11	byte	Назначенная связь для линии INTD# (аналогично)
12	word	Битовая карта возможных назначений для INTD# (аналогично)
14	byte	Номер слота (для физической идентификации карты)
15	byte	Резерв

Поиск 32-разрядных сервисов BIOS

32-разрядные сервисы BIOS32 ищутся через *каталог 32-разрядных сервисов*[1]. Адрес точки входа в каталог сервисов заранее не известен, но известен способ его нахождения: в диапазоне адресов памяти 0E0000–0FFFFFh в началах параграфов (младшие 4 бита адреса нулевые) ищется строка-сигнатура "_32_" заголовка (число 325F5F33h), за которой следует 32-разрядный физический адрес точки входа в каталог. Точки входа в сами сервисы ищутся через каталог сервисов. Номер, параметры вызываемых функций и результаты передаются на регистрах процессора.

Для поиска сервиса в каталоге 4-байтная строка-идентификатор сервиса заносятся в регистр EAX, в EBX заносится 0 (код функции поиска в каталоге) и выполняется дальний вызов (CALL FAR) по адресу точки входа в каталог. Результат поиска передается на регистрах: AL = 00h — сервис найден, при этом в EBX — базовый

[1] Смотри документ «Standard BIOS 32-bit Service Directory».

адрес сервиса, в ECX — его длина (определяет длину сегмента), EDX — смещение точки входа от начала сервиса (от EBX). Если AL = 81h — сервис не найден.

До попытки использования каталога сервисов следует убедиться в корректности заголовка, проверив его контрольную сумму: накопленная сумма всех байтов заголовка должна быть нулевой. Длина заголовка (в параграфах) указана в байте со смещением 9, в байте 8 — номер ревизии заголовка. Проверка контрольной суммы обязательна, поскольку 4-байтная сигнатура может совпасть с фрагментом программного кода BIOS (строка _32_ дизассемблируется как POP DI; XOR SI,[BP + SI]).

32-разрядные сервисы вызываются дальними вызовами (CALL FAR), при этом сегмент кода CS должен иметь базу в начале 4-килобайтной страницы, в которую попадает точка входа, а лимит должен позволять охватывать эту и следующую страницу[1]. Сегмент DS должен иметь такую же базу и не меньший лимит. Напомним, что здесь идет речь о физических адресах (после страничного преобразования линейных).

Expansion ROM карт PCI

В микросхеме ROM BIOS, установленной на системной плате, поддерживаются только стандартные (по назначению и реализации) устройства. При необходимости дополнительные устройства, устанавливаемые в слоты шин расширения (ISA, PCI, PCMCIA), могут иметь микросхемы ПЗУ своей программной поддержки — *Additional ROM BIOS* (дополнительные модули ROM BIOS), они же *Expansion ROM*. Эта необходимость возникает, когда программная поддержка устройств требуется до загрузки ОС и прикладного ПО. Роль Expansion ROM может и не ограничиваться поддержкой данного устройства — в таком модуле может содержаться и вся программа функционирования специализированного бездискового контроллера на базе PC. Расширения ROM BIOS используют графические адаптеры EGA/VGA/SVGA, некоторые контроллеры жестких дисков, контроллеры SCSI, сетевые адаптеры с удаленной загрузкой и другие периферийные устройства. Для модулей расширения устройств с шиной ISA в пространстве памяти зарезервирована область C8000h–F4000h. POST сканирует эту область с шагом 2 Кбайт в поисках дополнительных модулей BIOS на завершающем этапе выполнения (после загрузки векторов прерываний указателями на собственные обработчики). Дополнительный модуль BIOS графического адаптера (EGA, VGA, SVGA) имеет фиксированный адрес C0000 и инициализируется раньше (на шаге инициализации видеоадаптера). Устройства с шиной PCI в своем конфигурационном пространстве содержат лишь признак использования модуля расширения, а его приписку к адресам памяти назначает POST.

Дополнительный модуль ROM BIOS должен иметь заголовок, выровненный по границе 2-килобайтной страницы памяти, формат заголовка ПЗУ иллюстрирует табл. 5.3.

[1] Сегмент может начинаться и на несколько страниц раньше.

Таблица 5.3. Заголовок модуля дополнительного ПЗУ

Смещение	Длина	Назначение
0	2	Сигнатура (признак начала модуля): байт 0=55h, байт 1=AAh
2	1	Длина, указанная в блоках по 512 байт
3	3	Точка входа процедуры инициализации, заканчивающейся дальним возвратом Ret Far (вызывается инструкцией Far Call во время POST). Обычно здесь располагается 3-байтная инструкция JMP, указывающая на начало процедуры
6–17h		Резерв
18h	2	Указатель на структуру данных PCI (только для карт PCI)
1Ah	2	Указатель на структуру расширенного заголовка карт ISA PnP

В традиционном заголовке присутствовали только первые три поля, указатели на структуры PCI и ISA PnP ввели позже. Корректным считается модуль, начинающийся с признака AA55h (значения 16-битного слова с учетом порядка байтов) и нулевой суммой (по модулю 256) всех байтов в объявленной области (реальная длина модуля может превышать объявленную, но байт контрольной суммы, естественно, должен входить в объявленную область).

В случае обнаружения корректного модуля POST дальним вызовом (`Call Far`) вызывает процедуру инициализации модуля, начинающуюся с 3-го адреса заголовка модуля. Ответственность за ее корректность полностью ложится на разработчика. Процедура может переопределять векторы прерываний, обслуживаемых BIOS. Переопределив на себя вектор *Bootstrap* (`Int 19h`), можно получить управление при загрузке, что и используется, например, для удаленной загрузки компьютеров через локальную сеть (Remote Boot Reset). Если стандартное продолжение процедуры загрузки не требуется, а дополнительный модуль представляет собой, например, управляющую программу для какого-либо оборудования, вместо процедуры инициализации в ПЗУ может находиться и основная программа, не возвращающая управление системной последовательности POST, которая бы выполнила обычную загрузку.

Процедура инициализации и программная поддержка устройства в ПЗУ должны быть написаны таким образом, чтобы им были безразличны абсолютные адреса, по которым они размещаются в пространстве памяти. На картах расширения, как правило, имеются средства изменения базового адреса, а иногда и размера ПЗУ (джамперы или программно-управляемые переключатели). Это позволяет бесконфликтно разместить модули ПЗУ нескольких установленных карт.

Для содержимого ПЗУ расширения BIOS, установленных на картах PCI, принят стандарт, несколько отличающийся от традиционных дополнительных модулей ROM BIOS. Заголовок ПЗУ соответствует традиционному, но дополнительно имеет указатель на *структуру данных PCI* (табл. 5.4). Идентификаторы производителя и устройства, а также код класса совпадают с описанными в конфигурационном пространстве устройства PCI. Поскольку шина PCI используется не толь-

ко в PC, в ПЗУ карты может храниться несколько программных модулей. Каждый модуль начинается со структуры данных, сам модуль следует сразу за структурой. За ним начинается структура для следующего модуля (если у предыдущего не установлен признак последнего модуля) и т. д. Тип платформы (процессора) указывается в заголовке модуля и при инициализации BIOS активизируется только нужный. Такой механизм позволяет, например, один и тот же графический адаптер устанавливать и в IBM PC, и в Power PC.

Таблица 5.4. Структура данных PCI

Смещение	Длина, байт	Назначение
0	4	Сигнатура, строка символов «PCIR»
4	2	Идентификатор производителя
6	2	Идентификатор устройства
8	2	Резерв[1]
Ah	2	Длина структуры (в байтах), начиная с сигнатуры
Ch	1	Версия структуры (0 для данной версии)
Dh	3	Код класса
10h	2	Длина рабочего образа
12h	2	Версия кода/данных
14h	1	Тип кода: 0 — x86 для PC-AT, 2 — HP PA-RISC
15h	1	Индикатор: 1 — последний образ, 0 — не последний
16h	2	Резерв

Применительно к дополнительному ПЗУ карты PCI имеется три параметра, относящихся к размерам областей памяти. Размер ПЗУ определяется чтением конфигурационного пространства. Размер, указанный в байте 2 заголовка, указывает на длину модуля на этапе инициализации. Этот модуль POST загружает в ОЗУ перед тем, как вызвать процедуру инициализации (точка входа со смещением 3). Контрольная сумма, расположенная обычно в конце модуля, обеспечивает нулевую сумму всех байтов. Длина рабочего образа, указанная в структуре данных PCI (слово со смещением 10h), описывает размер области, которая должна постоянно оставаться в памяти в режиме нормального функционирования (она может быть меньше указанной в байте 2 заголовка, поскольку код процедуры инициализации уже не требуется). Эта область также защищается контрольной суммой.

Работа с модулями ПЗУ для карт PCI выполняется в соответствии с моделью *DDIM* (Device Driver Initialization Model — модель инициализации драйвера устройств). POST определяет наличие ПЗУ по полю `Expansion ROM Base Address` в конфигурационном пространстве устройства, обнаруженного на карте, и назначает ему

[1] До спецификации PCI 2.2 здесь помещался указатель на строку Vital Product Data (важные сведения о продукте).

адрес в свободном пространстве памяти. После этого программированием регистра команд данного устройства разрешается считывание ПЗУ и в нем ищется сигнатура заголовка AA55h. Когда сигнатура найдена, POST ищет подходящий образ (по типу программного кода и совпадающий по идентификаторам с обнаруженными устройствами PCI) и копирует его в ОЗУ (в область C0000–DFFFFh), оставляя разрешенной запись в эту область. Далее чтение ПЗУ запрещается (записью в поле `Expansion ROM Base Address`) и вызывается процедура инициализации модуля (по адресу 3). При вызове процедуры POST сообщает номер шины (в регистре `AH`), номер устройства (`AL[7:3]`) и номер функции (`AL[2:0]`), благодаря чему процедура узнает точные координаты (идентификатор на шине PCI) аппаратных средств, которые ей предстоит инициализировать. После отработки инициализации определяется размер области, которую следует оставить в памяти (по байту 2, который может быть модифицирован процедурой инициализации), и для этой области запрещается запись. Если процедура инициализации «урезает» занимаемую память, она должна позаботиться о достоверности контрольной суммы области, описанной байтом 2. Если память освобождается полностью (процедура обнуляет байт 2), то контрольная сумма, естественно, не нужна. Расширение для VGA (определяется по коду класса) обрабатывается особым образом — загружается по адресу C0000h. Процедура инициализации может определить наличие PnP BIOS в системе, проверив значение контрольной структуры PnP по адресу, указанному ей программой POST в регистрах `ES:DI`, и исполняться в зависимости от обнаруженного системного окружения.

Для более эффективной работы во время инициализации драйвера устройства желательно использовать не только стандартную, но и расширенную память (за пределами первого мегабайта), в то время как POST работает в реальном режиме процессора. Решить эту проблему помогает режим «*Big Real Mode*», который поддерживают все 32-разрядные процессоры. Специально для предоставления доступа ко всей памяти процедурами инициализации фирмы Phoenics и Intel разработали спецификацию *PMM* (POST Memory Manager Specification), версия 1.01 была опубликована в конце 1997 года. Эта спецификация определяет несколько дополнительных сервисов BIOS, позволяющих выделять, находить и освобождать блоки в любой, в том числе и расширенной, памяти. Клиенты этого сервиса запрашивают блок памяти требуемого размера, а BIOS возвращает физический 32-разрядный адрес начала выделяемого блока (если она способна его выделить). Клиент помечает свой блок 32-битным *индексом* (handle), по которому его в дальнейшем можно найти функцией поиска. Анонимный блок (индекс FFFFFFFFh) поиску не поддается. Этими сервисами можно пользоваться только до начала процедуры начальной загрузки (`Int 19h`), работу с вентилем *Gate A20* они берут на себя. Перед начальной загрузкой BIOS освобождает и обнуляет все блоки расширенной памяти, занятые с помощью этих сервисов. Наличие сервисов PMM определяется по контрольной структуре, начинающейся со строки-сигнатуры `$PMM` и расположенной на границе раздела в области E0000–FFFF0h. Программный интерфейс можно найти в вышеуказанном документе, который доступен на сайте http://www.phoenix.com/techs.

ГЛАВА 6

Электрический интерфейс и конструктивы для шин PCI и PCI-X

Электрический интерфейс

Для работы на шине PCI используются микросхемы КМОП (CMOS), причем имеются две спецификации: с напряжениями питания интерфейсных схем 5 и 3,3 В. Для них применимы параметры сигналов по постоянному току, приведенные в табл. 6.1. Однако мощность интерфейсных элементов (транзисторов для вентилей) выбрана меньшей, чем требовалось бы для переключения сигналов на высокой частоте (33 или 66 МГц). Здесь используется эффект отражения сигналов, формируемых микросхемами на проводниках шины, от несогласованных концов этих проводников, являющихся для таких высоких частот длинными линиями. На концах проводников шины нет терминаторов, поэтому от них приходящая волна сигнала отражается с тем же знаком и с той же амплитудой. Складываясь с прямым сигналом, обратная волна и обеспечивает нужный приемнику уровень сигнала. Таким образом, передатчик генерирует сигнал, уровень которого до прихода отраженного сигнала находится между уровнями переключения, и достигает требуемого уровня только после прихода отраженной волны. Это накладывает ограничение на физическую протяженность шины: сигнал должен успеть обернуться (дойти до конца и вернуться отраженным) за время, составляющее менее трети периода синхронизации (то есть 10 нс при 33 МГц, 5 нс при 66 МГц).

Линии управляющих сигналов FRAME#, TRDY#, IRDY#, DEVSEL#, STOP#, SERR#, PERR#, LOCK#, INTA#, INTB#, INTC#, INTD#, REQ64# и ACK64# на системной плате подтягиваются к шине питания резисторами (типичные номиналы: 2,7 кОм для версии 5 В и 8,2 кОм для 3,3 В), чтобы не было ложных срабатываний при пассивности всех агентов шины.

Электрические спецификации рассчитаны на два типовых варианта нагрузки одной шины: 2 устройства PCI на системной плате плюс 4 слота или 6 устройств плюс 2 слота. При этом подразумевается, что одно устройство на каждую линию шины PCI дает только единичную КМОП-нагрузку. В слоты могут устанавливаться

карты, также дающие только единичную нагрузку. При использовании компонентов и трассировки плат с характеристиками, превосходящими требования спецификации, возможны и иные сочетания числа слотов и устройств. Так, например, часто встречаются системные платы и с пятью слотами на одной физической шине. На длину проводников, а также на топологию расположения элементов и проводников для карт расширения накладываются жесткие ограничения. Длина сигнальных проводников не должна превышать 1,5 дюйма (3,8 см). Из вышесказанного понятно, что изготовление самодельных карт расширения на логических микросхемах средней степени интеграции, как это можно было делать для шин ISA, для PCI невозможно.

Таблица 6.1. Параметры интерфейсных сигналов на постоянном токе

Параметр	5 В интерфейс	3,3 В интерфейс
Входное напряжение низкого уровня, В	$-0{,}5 \le U_{IL} \le 0{,}8$	$-0{,}5 \le U_{IL} \le 0{,}3 \times V_{CC}$
Входное напряжение высокого уровня, В	$2 \le U_{IH} \le V_{CC} + 0{,}5$	$V_{CC}/2 \le U_{IH} \le V_{CC} + 0{,}5$
Выходное напряжение низкого уровня, В	$U_{OL} \le 0{,}55$	$U_{OL} \le 0{,}1 \times V_{CC}$
Выходное напряжение высокого уровня, В	$U_{OH} \ge 0{,}8$	$U_{OH} \ge 0{,}9 \times V_{CC}$
Напряжение питания V_{CC}, В	$4{,}75 \le U_{CC} \le 5{,}25$	$3{,}0 \le U_{CC} \le 3{,}6$

Тактовая частота шины определяется по возможностям всех абонентов шины, включая и мосты (и главный мост, входящий в чипсет системной платы). Высокая частота шины PCI 66 МГц может устанавливаться тактовым генератором только при высоком уровне на линии `M66EN`. Таким образом, установка любой карты, не поддерживающей 66 МГц (с заземленным контактом этой линии), приведет к понижению частоты шины до 33 МГц. Серверные системные платы, на которых имеется несколько шин PCI, позволяют использовать на разных шинах разные частоты (66 и 33 МГц). Так, например, можно на 64-битных слотах использовать частоту 66 МГц, а на 32-битных — 33. Разгон нормальной частоты 33 МГц до 40–50 МГц аппаратно не контролируется, но может приводить к ошибкам работы карт расширения.

Согласно спецификации PCI, устройства должны нормально работать при снижении частоты от номинальной (33 МГц) до нуля. Изменение частоты во время работы устройств допустимо при условии, чтобы все время соблюдались ограничения по минимальной длительности высокого и низкого уровней сигнала `CLK`. Останавливаться сигнал `CLK` должен только на низком уровне. После возобновления подачи импульсов `CLK` устройства должны продолжить работу, как будто остановки синхронизации и не было.

При работе с частотой 66 МГц и выше для снижения электромагнитных помех (EMI) от сигнала фиксированной частоты может применяться *расширение спектра сигнала* `CLK` (*spread spectrum*): неглубокая частотная модуляция с частотой модуляции 30–33 кГц. Если в устройствах для синхронизации используются схемы с ФАПЧ (PLL), то их быстродействие должно быть достаточным для отработки этой модуляции. В спецификации PCI-X диапазоны допустимых изменений тактовой частоты зависят от режима шины (см. табл. 6.5).

Стандартные слоты и карты PCI

Стандартные слоты PCI и PCI-X представляют собой щелевые разъемы, имеющие контакты с шагом 0,05 дюйма. Слоты расположены несколько дальше от задней панели, чем ISA/EISA или MCA. Компоненты карт PCI расположены на левой поверхности плат. По этой причине крайний PCI-слот обычно совместно использует посадочное место адаптера (прорезь на задней стенке корпуса) с соседним ISA-слотом. Такой слот называют *разделяемым* (shared slot), в него может устанавливаться либо карта ISA, либо PCI.

Карты PCI могут предназначаться для интерфейсных сигналов уровня 5 В и 3,3 В, а также быть универсальными. Слоты PCI имеют уровни сигналов, соответствующие питанию микросхем PCI-устройств системной платы (включая главный мост): либо 5 В, либо 3,3 В. Во избежание ошибочного подключения слоты имеют ключи, определяющие номинал напряжения. Ключами являются пропущенные ряды контактов 12, 13 и/или 50, 51:

- для *слота на 5 В* ключ (перегородка) расположен на месте контактов 50, 51 (ближе к передней стенке корпуса); такие слоты отменены в PCI 3.0;
- для *слота на 3,3 В* перегородка находится на месте контактов 12, 13 (ближе к задней стенке корпуса);
- на *универсальных слотах* перегородок нет;
- на краевых разъемах *карт 5 В* имеются ответные прорези только на месте контактов 50, 51; такие карты отменены в PCI 2.3;
- на *картах 3,3 В* прорези только на месте контактов 12, 13;
- на *универсальных картах* имеется оба ключа (две прорези).

Ключи не позволяют установить карту в слот с неподходящим напряжением питания. Карты и слоты различаются лишь питанием буферных схем, которое поступает с линий `+V I/O`:

- на слоте «5 В» на линии `+V I/O` подается + 5 В;
- на слоте «3,3 В» на линии `+V I/O` подается + (3,3–3,6) В;
- на карте «5 В» буферные микросхемы рассчитаны только на питание + 5 В;
- на карте «3,3 В» буферные микросхемы рассчитаны только на питание + (3,3–3,6) В;
- на универсальной карте буферные микросхемы допускают оба варианта питания и будут нормально формировать и воспринимать сигналы по спецификациям 5 или 3,3 В, в зависимости от типа слота, в который установлена карта (то есть от напряжения на контактах `+V I/O`).

На слотах обоих типов присутствуют *питающие напряжения* + 3,3, + 5, + 12 и –12 В на одноименных линиях. В PCI 2.2 определена дополнительная линия `3.3Vaux` — «дежурное» питание + 3,3 В для устройств, формирующих сигнал `PME#` при отключенном основном питании.

ПРИМЕЧАНИЕ

Выше приведены положения из официальных спецификаций PCI. На современных системных платах пока чаще всего встречаются слоты, по ключу являющиеся 5-вольтовыми. Однако при этом напряжение на линиях +V I/O и уровни сигналов интерфейса являются 3,3-вольтовыми. В этих слотах нормально работают все современные карты с 5-вольтовыми ключами — их интерфейсные схемы работают при питании как 3,3, так и 5 В. Интерфейс с 5-вольтовым питанием может работать только на частоте до 33 МГц. «Настоящие» 5-вольтовые системные платы были только для процессоров 486 и первых моделей Pentium.

Наибольшее распространение получили 32-битные слоты, заканчивающиеся контактами A62/B62. 64-битные слоты встречаются реже, они длиннее и заканчиваются контактами A94/B94. Конструкция разъемов и протокол позволяют устанавливать 64-битные карты как в 64-битные, так и в 32-битные разъемы, и наоборот, 34-битные карты как в 32-битные, так и в 64-битные разъемы. При этом разрядность обмена будет соответствовать слабейшему компоненту.

Для сигнализации об установке карты и потребляемой ею мощности на разъемах PCI предусмотрено два контакта — PRSNT1# и PRSNT2#, из которых хотя бы один соединяется на карте с шиной GND. С их помощью система может определить присутствие карты в слоте и ее энергопотребление. Кодирование потребляемой мощности приведено в табл. 6.2; здесь приведены значения и для малогабаритных карт Small PCI (см. ниже).

Таблица 6.2. Сигнализация присутствия карты и потребляемой мощности PCI

Соединение контактов		Потребляемая мощность	
PRSNT1#	**PRSNT2#**	**PCI**	**Small PCI**
–	–	Нет карты	Нет карты
GND	–	25 Вт макс	10 Вт макс
–	GND	15 Вт макс	5 Вт макс
GND	GND	7,5 Вт макс	2 Вт макс

Карты и слоты PCI-X по механическим ключам соответствуют 3,3-вольтовым картам и слотам; напряжение питания + V I/O для PCI-X Mode 2 устанавливается 1,5 В.

На рис. 6.1 изображены карты PCI в конструктиве PC/AT-совместимых компьютеров. Полноразмерные карты (Long Card, 107×312 мм) используются редко, чаще применяются укороченные платы (Short Card, 107×175 мм), но многие карты имеют и меньшие размеры. Карта имеет обрамление (скобку), стандартное для конструктива ISA (раньше встречались карты и с обрамлением в стиле MCA IBM PS/2). У низкопрофильных карт (Low Profile) высота не превышает 64,4 мм; их скобки также имеют меньшую высоту. Такие карты могут устанавливаться вертикально в 19-дюймовые корпуса высотой 2U (около 9 см).

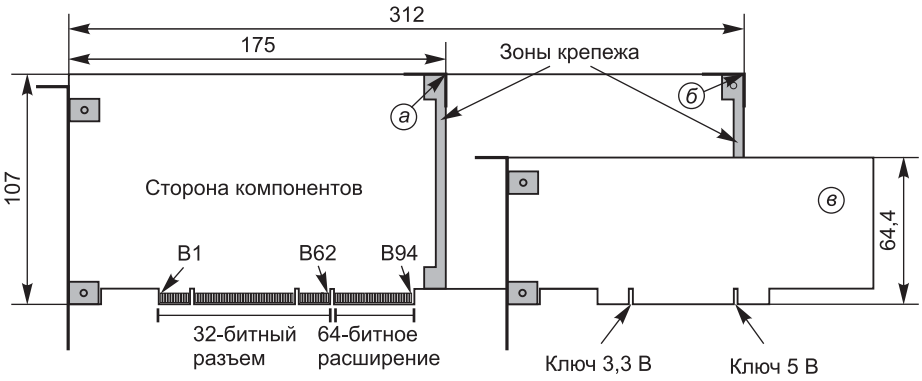


Рис. 6.1. Карты расширения для шины PCI: *а* — укороченная (обычная); *б* — полноразмерная, *в* — низкопрофильная

Назначение выводов разъема карт PCI/PCI-X приведено в табл. 6.3.

Таблица 6.3. Разъемы шины PCI

Ряд B	№	Ряд A	Ряд B	№	Ряд A
−12 B	1	TRST#	GND/M66EN [1]	49	AD9
TCK	2	+12 B	GND/Ключ 5 B/MODE2	50	GND/Ключ 5 B
GND	3	TMS	GND/Ключ 5 B	51	GND/Ключ 5 B
TDO	4	TDI	AD8	52	C/BE0#
+5 B	5	+5 B	AD7	53	+3,3 B
+5 B	6	INTA#	+3,3 B	54	AD6
INTB#	7	INTC#	AD5	55	AD4
INTD#	8	+5 B	AD3	56	GND
PRSNT1#	9	ECC5 [2]	GND	57	AD2
ECC4 [2]	10	+V I/O	AD1	58	AD0
PRSNT2#	11	ECC3 [2]	+V I/O	59	+V I/O
GND/Ключ 3,3 B	12	GND/Ключ 3,3 B	ACK64#/ ECC1	60	REQ64#/ ECC6
GND/Ключ 3,3 B	13	GND/Ключ 3,3 B	+5 B	61	+5 B
ECC2 [2]	14	3.3Vaux [3]	+5 B	62	+5 B
GND	15	RST#	Конец 32-битного разъема		
CLK	16	+V I/O	Резерв	63	GND
GND	17	GNT#	GND	64	C/BE7#
REQ#	18	GND	C/BE6#	65	C/BE5#
+V I/O	19	PME# [3]	C/BE4#	66	+V I/O
AD31	20	AD30	GND	67	PAR64/ECC7 [2]

продолжение

Таблица 6.3 (*продолжение*)

Ряд B	№	Ряд A	Ряд B	№	Ряд A
AD29	21	+3,3 B	AD63	68	AD62
GND	22	AD28	AD61	69	GND
AD27	23	AD26	+V I/O	70	AD60
AD25	24	GND	AD59	71	AD58
+3,3 B	25	AD24	AD57	72	GND
C/BE3#	26	IDSEL	GND	73	AD56
AD23	27	+3,3 B	AD55	74	AD54
GND	28	AD22	AD53	75	+V I/O
AD21	29	AD20	GND	76	AD52
AD19	30	GND	AD51	77	AD50
+3.3 B	31	AD18	AD49	78	GND
AD17	32	AD16	+V I/O	79	AD48
C/BE2#	33	+3,3 B	AD47	80	AD46
GND	34	FRAME#	AD45	81	GND
IRDY#	35	GND	GND	82	AD44
+3,3 B	36	TRDY#	AD43	83	AD42
DEVSEL#	37	GND	AD41	84	+V I/O
PCIXCAP [4]	38	STOP#	GND	85	AD40
LOCK#	39	+3,3 B	AD39	86	AD38
PERR#	40	SMBCLK [5]	AD37	87	GND
+3,3 B	41	SMBDAT [5]	+V I/O	88	AD36
SERR#	42	GND	AD35	89	AD34
+3,3 B	43	PAR/ECC0	AD33	90	GND
C/BE1#	44	AD15	GND	91	AD32
AD14	45	+3,3 B	Резерв	92	Резерв
GND	46	AD13	Резерв	93	GND
AD12	47	AD11	GND	94	Резерв
AD10	48	GND	Конец 64-битного разъема		

[1] Сигнал M66EN определен в PCI 2.1 только для слотов на 3,3 В.

[2] Сигнал введен в PCI-X 2.0 (прежде был резерв).

[3] Сигнал введен в PCI 2.2 (прежде был резерв).

[4] Сигнал введен в PCI-X (в PCI — GND).

[5] Сигналы введены в PCI 2.3. В PCI 2.0 и 2.1 контакты A40 (SDONE#) и A41 (SBOFF#) использовались для слежения за кэшем; в PCI 2.2 они были освобождены (для совместимости на системной плате эти цепи подтягивались к высокому уровню резисторами 5 кОм).

На слотах PCI имеются контакты для тестирования адаптеров по интерфейсу JTAG (сигналы TCK, TDI, TDO, TMS и TRST#). На системной плате эти сигналы задействованы не всегда, но они могут и организовывать логическую цепочку тестируемых адаптеров, к которой можно подключить внешнее тестовое оборудование. Для непрерывности цепочки на карте, не использующей JTAG, должна быть связь TDI–TDO.

На некоторых старых системных платах позади одного из слотов PCI встречается разъем Media Bus, на который выводятся сигналы ISA. Он предназначен для размещения на карте PCI звукового чипсета, предназначенного для шины ISA.

Большинство сигналов PCI соединяются по чистой шинной топологии, то есть одноименные контакты слотов одной шины PCI электрически соединяются друг с другом. Из этого правила есть несколько исключений:

- сигналы REQ# и GNT# индивидуальны для каждого слота, они соединяют слот с арбитром (обычно — мостом, подключающим эту шину к вышестоящей);
- сигнал IDSEL для каждого слота соединяется (возможно, через резистор) с одной из линий AD[31:11], задавая номер устройства на шине;
- сигналы INTA#, INTB#, INTC#, INTD# циклически сдвигаются по контактам (см. рис. 3.1), обеспечивая распределение запросов прерываний;
- сигнал CLK заводится на каждый слот индивидуально от своего выхода буфера синхронизации; длина подводящих проводников выравнивается, обеспечивая синхронность сигнала на всех слотах (для 33 МГц допуск ± 2 нс, для 66 МГц — ± 1 нс).

Когда обычная системная плата используется в низкопрофильных корпусах, для подключения карт расширения можно использовать *пассивный переходник* (*Riser Card*), устанавливаемый в один из слотов PCI. Если в переходник устанавливается более одной карты, то для реализации вышеупомянутых исключений используют выносные разъемы PCI (маленькие печатные платы), с помощью которых вышеперечисленные сигналы берутся от других, свободных слотов PCI на системной плате. Переставляя эти разъемы, можно менять номера устройств на слотах переходника, а главное — их раскладку по линиям запросов прерывания. Беда такого подключения — длинные (10–15 см) шлейфы, соединяющие переходник со слотами. Все сигналы в этом шлейфе передаются по параллельным неперевитым проводам, что очень плохо для сигнала CLK: его форма искажается и вносится значительная задержка. Результатом могут быть внезапные «зависания» компьютера без всяких диагностических сообщений. В такой ситуации может помочь отделение провода CLK от общего шлейфа и встречное скручивание его свободного конца (это уменьшает индуктивность проводника). Остальные сигналы в шлейфе не так критичны к качеству разводки. Лучшим решением будет использование низкопрофильных карт PCI, устанавливаемых в системную плату без переходников. Проблема не возникала бы, если бы на переходнике была установлена микросхема источника синхронизации, раздающего синхросигнал на все слоты переходника. Однако это требует применения микросхем с ФАПЧ (PLL), привязывающих свой выходной сигнал к сигналу синхронизации от системной платы, что несколько удорожает переходник.

Инициализация и определение режима работы шины PCI-X

Каждый сегмент PCI-X (физическая шина) должен работать в самом прогрессивном режиме, доступном всем его абонентам, включая и главный для этой шины мост. В стандартной шине PCI «прогрессивность» определяется только допустимой тактовой частотой (33 или 66 МГц), и свои способности карта сообщает по контакту B49 (M66EN, см. выше). В шине PCI-X появляются новые возможности: поддержка собственно протокола PCI-X (*Mode 1* в терминах PCI-X 2.0) и ускоренных передач (*Mode 2*). Эти возможности карта сообщает через контакт B38 (PCIXCAP), который может быть подключен к шине GND через резистор определенного номинала или оставаться неподключенным (NC), как указано в табл. 6.4). Номиналы резисторов выбраны так, что мост может определить возможности карт в многослотовых шинах, когда цепи PCIXCAP всех карт соединяются параллельно (кроме резисторов на картах имеются и конденсаторы). Мост, которому подчиняется данная шина, проверяет состояние линий M66EN и PCIXCAP по началу сигнала сброса. В соответствии с увиденными возможностями (они будут соответствовать самому слабому абоненту) мост выбирает режим работы шины. Этот режим доводится до всех абонентов с помощью *шаблона инициализации* (PCI-X Initialization Pattern) — уровней сигналов PERR#, DEVSEL#, STOP# и TRDY# в момент окончания сигнала RST# (по его нарастающему фронту). К этому моменту на слоты уже подается соответствующее напряжение +VI/O. Возможные режимы шины и их шаблоны инициализации приведены в табл. 6.5.

Таблица 6.4. Сообщение свойств карт PCI/PCI-X

Соединение на карте для контактов		Способности карты расширения
B49 (M66EN)	**B38 (PCIXCAP)**	
GND	GND	PCI 33 МГц
NC	GND	PCI 66 МГц
GND или NC	GND через R1	PCI-X 66
GND или NC	NC	PCI-X 133
GND или NC	GND через R2	PCI-X 266
GND или NC	GND через R3	PCI-X 533

Таблица 6.5. Режимы и шаблоны инициализации шины PCI/PCI-X

Сигнал				Режим шины	Частота,	Контроль
PERR#	**DEVSEL#**	**STOP#**	**TRDY#**	**(протокол)**	**МГц**	**достоверности**
H	H	H	H	PCI	0–33	Четность
H	H	H	H	PCI	33–66	Четность
H	H	H	L	PCI-X Mode1	50–66	Четность
H	H	L	H	PCI-X Mode1	66–100	Четность

Сигнал PERR#	DEVSEL#	STOP#	TRDY#	Режим шины (протокол)	Частота, МГц	Контроль достоверности
H	H	L	L	PCI-X Mode1	100–133	Четность
H	L	H	H	PCI-X Mode1	Резерв	ECC
H	L	H	L	PCI-X Mode1	50–66	ECC
H	L	L	H	PCI-X Mode1	66–100	ECC
H	L	L	L	PCI-X Mode1	100–133	ECC
L	H	H	H	PCI-X266 Mode2	Резерв	ECC
L	H	H	L	PCI-X266 Mode2	50–66	ECC
L	H	L	H	PCI-X266 Mode2	66–100	ECC
L	H	L	L	PCI-X266 Mode2	100–133	ECC
L	L	H	H	PCI-X533 Mode2	Резерв	ECC
L	L	H	L	PCI-X533 Mode2	50–66	ECC
L	L	L	H	PCI-X533 Mode2	66–100	ECC
L	L	L	L	PCI-X533 Mode2	100–133	ECC

«Горячее» подключение устройств — Hot Plug

«Горячее» подключение-отключение устройств PCI (*Hot Plug*) требует наличия в системе специального контроллера (*Hot-Plug Controller*), управляющего слотами «горячего» подключения, и соответствующей программной поддержки — ОС, драйверов устройств и контроллера.

Слот с «горячим» подключением должен быть подключен к шине PCI через коммутирующие цепи, обеспечивающие:

- управляемую коммутацию (электронными ключами) всех сигнальных цепей PCI;
- управляемую подачу напряжения питания.

Контроллер «горячего» подключения должен обеспечивать для каждого своего слота:

- индивидуальное управление коммутацией сигналов и подачей питания;
- индивидуальное управление сигналом RST#;
- индивидуальную идентификацию состояния линий PRSNT[1:2]#, независимо от состояния слота (подключен или изолирован);
- индивидуальную идентификацию состояния линии M66EN, независимо от состояния слота (подключен или изолирован);
- индивидуальный индикатор «*Внимание*», сигнализирующий о состоянии питания слота (можно ли извлекать и устанавливать модуль). Индикатор управляется программно, сигнализируя пользователю и о проблемах, обнаруженных системой для устройства в данном слоте.

В «горячем» подключении участвует пользователь, который должен устанавливать (и извлекать) модули (карты расширения) только в слот с отключенным пи-

танием (при этом сигналы слота отключены от шины). После установки модуля на него подается питание, затем через некоторое время на него подается сигнал RST#, и устройство приходит в исходное состояние. Только после этого коммутатор соединяет сигналы слота с шиной. Далее программная поддержка должна выполнить идентификацию и конфигурирование подключенного устройства. Дополнительные сложности возникают, если к шине, работающей на частоте 66 МГц, подключается модуль на 33 МГц. Поскольку тактовую частоту на шине можно менять только во время действия сигнала RST#, а подключаемое устройство работать на высокой частоте не может, подключение потребует выполнения сброса на шине PCI (с последующей инициализацией всех устройств). Перед отключением питания слота на него подается сигнал RST# и его сигналы отключаются от шины.

Малогабаритные конструктивы с шиной PCI

Стандартный конструктив PCI для настольных PC/AT-совместимых компьютеров для ряда применений является слишком громоздким. Существуют более компактные варианты:

- *Low-Profile PCI* — низкопрофильный вариант карты PCI для системных плат AT/ATX. Эти карты имеют такой же краевой печатный разъем, как и обычные карты, но высота карты уменьшена до 64 мм и уменьшен размер крепежной скобки. Карты можно устанавливать вертикально (без переходника riser card) в низкопрофильные корпуса (например, 19-дюймового формата высотой 2U). Для установки этих карт в полноразмерные (настольные) корпуса на карте следует установить обычную крепежную скобку (в комплект поставки карты может входить дополнительная скобка). Для этих карт в спецификации предусматривается напряжение питания интерфейсных схем только 3,3 В, хотя часто встречаются формально (по ключам) 5-вольтовые низкопрофильные карты;
- малогабаритные конструктивы для блокнотных компьютеров (их параметры приведены в табл. 6.6):
  - для карт расширения, устанавливаемых пользователем без вскрытия компьютера (с возможностью «горячего» подключения), применяется конструктив PCMCIA, впоследствии переименованный в PC Card. В этом конструктиве возможны четыре различных варианта интерфейса (см. далее), одним из которых является CardBus — шина PCI 32 бит/33 МГц;
  - для комплектования компьютера изготовителем внутри компьютера (недоступно пользователю) применяются конструктивы Small PCI и Mini PCI.

Таблица 6.6. Конструктивы и интерфейсы периферии портативных ПК

	PC Card	Small PC Card	Express Card	Small PCI	Mini PCI Type I и II	Mini PCI Type III
Длина, мм	85,6	42,8	75	85,6	70/78	51/44,6
Ширина (по стороне с разъемом), мм	54,0	45,0	34/54	54,0	46	60

	PC Card	Small PC Card	Express Card	Small PCI	Mini PCI Type I и II	Mini PCI Type III
Толщина, мм	3,3/5,0/10,5	3,3/5,0/10,5	5	3,3/5,0/10,5	7,5/5,5/17,5	5
Коннектор	Штырьковый, 1,27×1,27 мм	Штырьковый, 1,27×1,27 мм	[1]	Штырьковый, 0,8×1 мм	Штырьковый, 0,8×1 мм	Печатный, 0,8 мм
Число контактов	68	68	26	108	100	124
Интерфейсы	Память, ввод-вывод, ATA, CardBus (PCI)	Память, ввод-вывод, ATA, CardBus (PCI)	PCI Express, USB 2.0	PCI	PCI, PCI-X	PCI, PCI-X

[1] О конструкции разъема сведений нет, автору известно только название «beam on blade».

Конструктивы Small PCI и Mini PCI

Small PCI (*SPCI*) — спецификация PCI в миниатюрном исполнении, прежде называвшаяся *SFF PCI* (Small Form-Factor). Эта спецификация, предназначенная в основном для портативных компьютеров, логически совпадает с обычной шиной PCI, разрядность 32 бита, частота 33/66 МГц. В дополнение к обычному набору сигналов появился новый CLKRUN#, с помощью которого хост и устройства могут управлять частотой синхронизации в интересах энергосбережения. Благодаря уменьшению размеров (длины проводников) понижены требования к мощности формирователей сигналов PCI. По размерам карты SPCI *Style A* и *Style B* совпадают с *PC Card* и *Card Bus Type II* и *III* соответственно, но специальные ключи предотвращают ошибки подключения. Карты SPCI могут быть трех видов: с питанием 5 В, 3,3 В и универсальные 5/3,3 В; ошибочная установка предотвращается механическими ключами — прорезями в направляющих на карте и выступами в направляющих сокета (рис. 6.2). Для подключения карт SPCI на системной плате устанавливается двухрядный 108-контактный штырьковый разъем (табл. 6.7) с шагом контактов 0,8 мм по горизонтали и 1 мм между рядами. На карте, соответственно, имеется ответная часть (гнезда с тем же шагом). На противоположной стороне карты для периферийных интерфейсов предусматривается два варианта разъемов (на рисунке они не показаны): аналогичный (гнезда с шагом 0,8×1 мм) или двусторонний печатный с шагом 2 мм. Периферийные цепи подключаются через ленточный кабель с соответствующим разъемом. В стандарте приводятся рекомендации назначения контактов периферийных разъемов для сетевых адаптеров (Ethernet, Token Ring) и графических адаптеров (интерфейс VGA). Шина SPCI является внутренней (карты расширения находятся под крышкой корпуса и устанавливаются изготовителем при выключенном питании) и поэтому не нацелена на замену Card Bus (шина для внешних подключений с возможностью «горячей» замены). Карты SPCI позволяют использовать преимущества модульных решений, обеспечивая высокую производительность обмена.

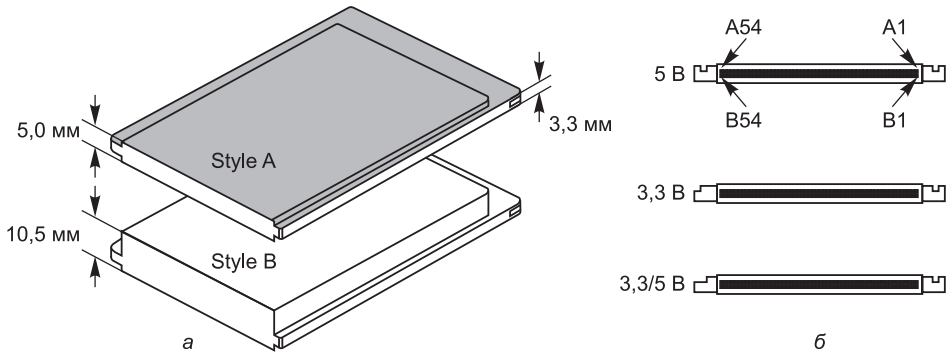


Рис. 6.2. Карты Small PCI: *а* — карты Style A и Style B; *б* — разъемы и ключи

Таблица 6.7. Назначение контактов разъема Small PCI

№	Ряд B	Ряд A	№	Ряд B	Ряд A
1	GND	GND	28	AD17	AD16
2	INTB#	+12V	29	C/BE2#	+3.3V
3	+5V	INTA#	30	GND	FRAME#
4	INTD#	INTC#	31	IRDY#	GND
5	-12V	+5V	32	+3.3V	TRDY#
6	PRSNT1#	Резерв	33	DEVSEL#	GND
7	Резерв	Резерв	34	GND	STOP#
8	PRSNT2#	+5V	35	LOCK#	+3.3V
9	CLK	RST#	36	PERR#	SDONE
10	GND	GNT#	37	GND	SBO#
11	REQ#	GND	38	SERR#	GND
12	+5V	CLKRUN#	39	+3.3V	PAR
13	AD31	AD30	40	C/BE1#	AD15
14	AD29	+5V	41	AD14	+3.3V
15	GND	AD28	42	GND	AD13
16	AD27	AD26	43	AD12	AD11
17	AD25	GND	44	AD10	M66EN
18	+V(I/O)	AD24	45	+V(I/O)	AD09
19	Резерв	+V(I/O)	46	AD08	C/BE0#
20	GND	Резерв	47	AD07	+V(I/O)
21	C/BE3#	GND	48	+5V	AD06
22	+3.3V	IDSEL	49	AD05	AD04
23	AD23	+3.3V	50	AD03	+5V
24	GND	AD22	51	+5V	AD02

№	Ряд B	Ряд A	№	Ряд B	Ряд A
25	AD21	AD20	52	AD01	AD00
26	AD19	GND	53	ACK64#	GND
27	+3.3V	AD18	54	GND	REQ64#

Mini PCI Specification — малогабаритный вариант карт расширения с шиной PCI, устанавливаемых внутрь блокнотного ПК изготовителем. Логически и электрически соответствует PCI 32 бит, частота 33/66 МГц. Основное питание и уровень сигналов — 3,3 В; питание +5 В для карты доступно, но с максимальным потребляемым током 100 мА; питание ±12 В не предусмотрено. Суммарная потребляемая мощность карты не должна превышать 2 Вт. На разъеме MiniPCI имеются дополнительные сигналы для подключения аудиокодека AC'97, аналоговых аудиосигналов, телефонной линии, интерфейса локальной сети, а также отдельная линия +5 В для питания аналоговых цепей. Назначение дополнительных сигналов Mini PCI приведено в табл. 6.8. В отличие от карт Small PCI и PC Card (PCMCIA) карты Mini PCI не имеют защитного корпуса — их элементы открыты, как и на «больших» картах PCI. В спецификации имеется несколько вариантов конструктивных исполнений (Form Factor):

- *Type IA* и *Type IB* (рис. 6.3, *а*) — для карт, устанавливаемых не у края корпуса компьютера. Системный разъем — штырьковый двухрядный стековый (штырьки контактов перпендикулярны плоскости платы). Кроме системного разъема на них могут устанавливаться внутренние малогабаритные разъемы для подключения к телефонной линии и локальной сети (в любых сочетаниях, но на штатных местах), а также альтернативный разъем ввода-вывода для иных применений. Карта имеет размер 46×70 мм, она крепится на трех крепежных площадках с отверстиями 3 мм. Толщина печатной платы не должна превышать 1 мм. *Type IB* отличается меньшей допустимой высотой компонентов;
- *Type IIA* и *Type IIB* (рис. 6.3, *б*) — для карт, устанавливаемых у края корпуса компьютера. Здесь состав разъемов тот же, но плата увеличена до размера 46×78 мм (крепежные отверстия остались на прежних местах относительно системного разъема), а для подключения телефона и локальной сети предусмотрена установка гнезд RJ-11 и RJ-45 в экранированных корпусах. На карте *Type IIA* компоненты могут иметь большую высоту (до 13,5 мм) на большей части платы; на картах *Type IIB* такая высота допустима только для самих гнезд;
- *Type IIIA* и *Type IIIB* (рис. 6.3, *в*) — для карт, устанавливаемых не у края корпуса компьютера. Здесь применен иной системный разъем — двусторонний печатный с шагом контактов 0,8 мм, толщина платы — 1 мм. На плате определено место для внутренних малогабаритных разъемов подключения к телефонной линии и локальной сети. Карта крепится направляющими разъема и фиксируется защелками в прорезях. Карты *Type IIIA* и *Type IIIB* различаются размером (60×51 мм и 60×44,6 мм соответственно).

Назначение контактов системных разъемов для карт Mini PCI приведено в табл. 6.9, системный разъем для карт *Type III* по сравнению с *Type I* и *Type II* имеет дополнительные цепи, сгруппированные по его краям.

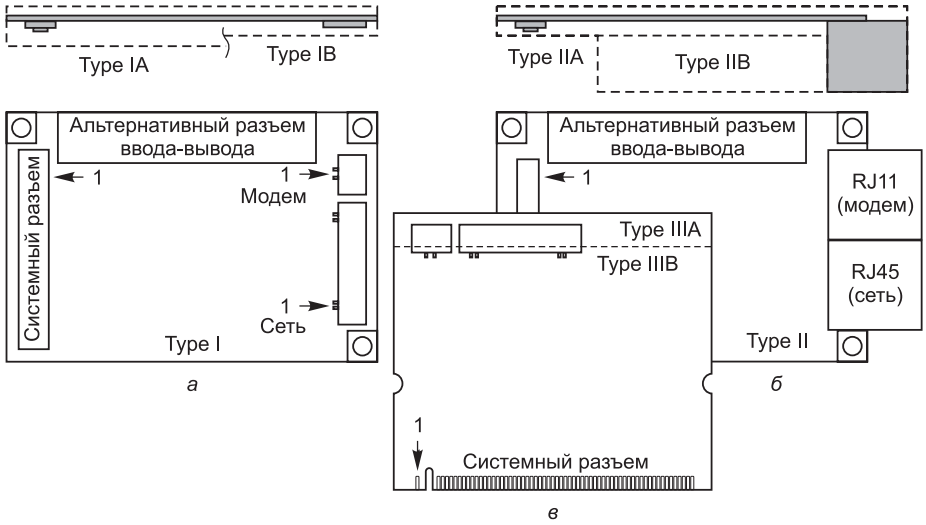


Рис. 6.3. Карты Mini PCI: *а* — Type I; *б* — Type II; *в* — Type III

Таблица 6.8. Назначение дополнительных цепей Mini PCI

Сигнал	Назначение
TIP, RING	Цепи телефонной линии
8PMJ-1… 8PMJ-8	Контакты разъема RJ-45 для подключения к локальной сети
LED1_GRNP, LED1_GRNN, LED2_YELP, LED2_YELN	Выводы светодиодных индикаторов разъема локальной сети (желтого и зеленого)
CHSGND	Заземление (корпус)
AC_SYNC, AC_SDATA_IN, AC_SDATA_OUT, AC_BIT_CLK, AC_CODEC_IDO#, AC_CODEC_ID1#, AC_RESET#	Сигналы связи с аудиокодеком AC'97 (AC-link)
MOD_AUDIO_MON	Аудиомонитор модема
SYS_AUDIO_OUT, SYS_AUDIO_IN	Аудиосигнал для телефона
SYS_AUDIO_OUT_GND, SYS_AUDIO_IN_GND, AUDIO_GND	«Земля» для аудиосигналов
MPCIACT#	Дополнительный сигнал управления энергопотреблением: карта активна и требует для обслуживания максимальной производительности системы

Таблица 6.9. Назначение контактов разъема Mini PCI

Контакт для карт типов		Цепь	Контакт для карт типов		Цепь
I, II	III		I, II	III	
	1	TIP		2	RING
	Ключ			Ключ	
	3	8PMJ-3		4	8PMJ-1

Контакт для карт типов I, II	Контакт для карт типов III	Цепь	Контакт для карт типов I, II	Контакт для карт типов III	Цепь
	5	8PMJ-6		6	8PMJ-2
	7	8PMJ-7		8	8PMJ-4
	9	8PMJ-8		10	8PMJ-5
	11	LED1_GRNP		12	LED2_YELP
	13	LED1_GRNN		14	LED2_YELN
	15	CHSGND		16	Резерв
1	17	INTB#	2	18	5V
3	19	3.3V	4	20	INTA#
5	21	Резерв	6	22	Резерв
7	23	GND	8	24	3.3VAUX
9	25	CLK	10	26	RST#
11	27	GND	12	28	3.3V
13	29	REQ#	14	30	GNT#
15	31	3.3V	16	32	GND
17	33	AD31	18	34	PME#
19	35	AD29	20	36	Резерв
21	37	GND	22	38	AD30
23	39	AD27	24	40	3.3V
25	41	AD25	26	42	AD28
27	43	Резерв	28	44	AD26
29	45	C/BE3#	30	46	AD24
31	47	AD23	32	48	IDSEL
33	49	GND	34	50	GND
35	51	AD21	36	52	AD22
37	53	AD19	38	54	AD20
39	55	GND	40	56	PAR
41	57	AD17	42	58	AD18
43	59	C/BE2#	44	60	AD16
45	61	IRDY#	46	62	GND
47	63	3.3V	48	64	FRAME#
49	65	CLKRUN#	50	66	TRDY#
51	67	SERR#	52	68	STOP#
53	69	GND	54	70	3.3V
55	71	PERR#	56	72	DEVSEL#
57	73	C/BE1#	58	74	GND
59	75	AD14	60	76	AD15
61	77	GND	62	78	AD13

продолжение ⇩

Таблица 6.9 (*продолжение*)

Контакт для карт типов I, II	Контакт для карт типов III	Цепь	Контакт для карт типов I, II	Контакт для карт типов III	Цепь
63	79	AD12	64	80	AD11
65	81	AD10	66	82	GND
67	83	GND	68	84	AD09
69	85	AD08	70	86	C/BE0#
71	87	AD07	72	88	3.3V
73	89	3.3V	74	90	AD06
75	91	AD05	76	92	AD04
77	93	Резерв	78	94	AD02
79	95	AD03	80	96	AD00
81	97	5V	82	98	Резерв
83	99	AD01	84	100	Резерв
85	101	GND	86	102	GND
87	103	AC_SYNC	88	104	M66EN
89	105	AC_SDATA_IN	90	106	AC_SDATA_OUT
91	107	AC_BIT_CLK	92	108	AC_CODEC_IDO#
93	109	AC_CODEC_ID1#	94	110	AC_RESET#
95	111	MOD_AUDIO_MON	96	112	Резерв
97	113	AUDIO_GND	98	114	GND
99	115	SYS_AUDIO_OUT	100	116	SYS_AUDIO_IN
	117	SYS_AUDIO_OUT GND		118	SYS_AUDIO_IN GND
	119	AUDIO_GND		120	AUDIO_GND
	121	Резерв		122	MPCIACT#
	123	VCC5VA		124	3.3VAUX

Карты PCMCIA: интерфейсы PC Card, CardBus

В начале 90-х годов организация *PCMCIA* (Personal Computer Memory Card International Association — Международная ассоциация производителей карт памяти для персональных компьютеров) начала работы по стандартизации шин расширения блокнотных компьютеров, в первую очередь предназначенных для расширения памяти. Первым появился стандарт *PCMCIA Standard Release 1.0/JEIDA 4.0* (июнь 1990 года), в котором был описан 68-контактный интерфейсный разъем и два типоразмера карт: *Type I* и *Type II PC Card*. Поначалу стандарт касался электрических и физических требований только для карт памяти. Был введен метаформат информационной структуры карты *CIS* (Card Information Structure), в которой описываются характеристики и возможности карты, — ключевой элемент взаимозаменяемости карт и обеспечения механизма PnP.

Следующая версия *PCMCIA 2.0* (1991 год) для того же разъема определила интерфейс операций ввода-вывода, двойное питание для карт памяти, а также методики тестирования. В версии 2.01 были добавлены спецификация *PC CardATA*, новый типоразмер *Type III*, спецификация автоиндексируемой массовой памяти AIMS (Auto-Indexing Mass Storage) и начальный вариант сервисной спецификации (Card Services Specification). В версии 2.1 (1994 год) расширили спецификации сервисов карт и сокетов (Card and Socket Services Specificaiton) и развили структуру CIS.

Стандарт *PC Card* (1995 год) явился продолжением предыдущих; в нем введены дополнительные требования для улучшения совместимости и новые возможности: питание 3,3 В, поддержка DMA, а также 32-битной шины PCI — *CardBus*. В дальнейшем в стандарт были введены и другие дополнительные возможности.

Все карты PCMCIA и PC Card имеют 68-контактный разъем, назначение контактов у которого варьирует в зависимости от типа интерфейса карты. Тип интерфейса «заказывается» картой при установке ее в слот, который, естественно, должен поддерживать требуемый интерфейс. *Интерфейс памяти* обеспечивает 8- и 16-битные обращения с минимальным временем цикла 100 нс, что дает максимальную производительность 10 и 20 Мбайт/с соответственно. *Интерфейс ввода-вывода* имеет минимальную длительность цикла 255 нс, что соответствует 3,92/7,84 Мбайт/с для 8-/16-битных обращений. *Интерфейс CardBus* поддерживает протокол обмена PCI; тактовая частота 33 МГц, разрядность 32 бита. Здесь используется та же система автоматического конфигурирования, что и в PCI (через регистры конфигурационного пространства). В интерфейс заложены дополнительные возможности для цифровой передачи аудиосигнала, причем как в традиционной форме ИКМ (PCM), так и в новой (забытой старой) форме ШИМ (PWM). Для дисковых *устройств ATA* в формате PC Card имеется специальная спецификация интерфейса.

Существует несколько конструктивных типов PC Card (рис. 6.4): у них у всех размер в плане 54×85,5 мм, но разная толщина (меньшие адаптеры встают в большие гнезда):

- *PC Card Type I* — 3,3 мм — карты памяти;
- *PC Card Type II* — 5 мм — карты устройств ввода-вывода, модемы, адаптеры локальных сетей;
- *PC Card Type III* — 10,5 мм — дисковые устройства хранения;
- *PC Card Type IV* — 16 мм (упоминания об этом типе на сайте `http://www.pc-card.com` найти не удалось).

Есть еще и маленькие карты *Small PC Card* размером 45×42,8 мм с тем же коннектором и теми же типами по толщине.

Большинство карт PC Card выпускается с поддержкой технологии PnP и предусматривает «горячее» подключение — интерфейсные карты могут вставляться и выниматься без выключения компьютера. Для этого контакты шин питания имеют большую длину, чем сигнальные, обеспечивая их упреждающее подключение и запаздывающее отключение. Два контакта обнаружения карты `CD1#` и `CD2#` (Card

Detect) короче остальных — их замыкание для хоста означает, что карта полностью вставлена в слот. Несмотря на возможность динамического конфигурирования, в некоторых случаях при изменении конфигурации требуется перезагрузка системы.

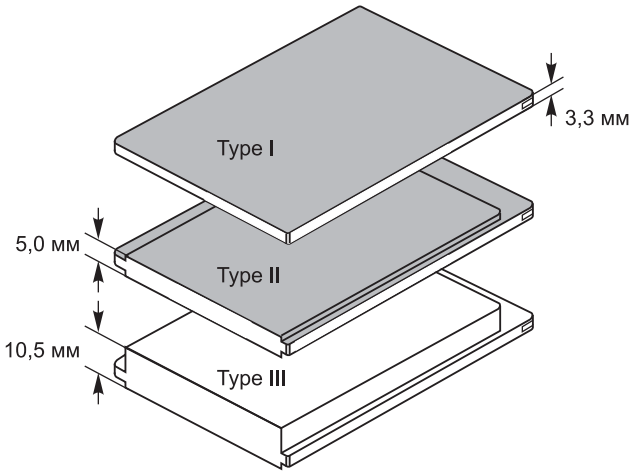


Рис. 6.4. Карты PC Card (PCMCIA)

Первоначально карты и хост-системы использовали напряжение питания логики +5 В. Для перехода на низковольтное питание (3,3 В) был введен механический ключ, не допускающий установки карты на 3,3 В в слот, дающий только 5 В. Кроме того, были определены контакты 43 (VS1#) и 57 (VS2#) для выбора питающего напряжения. На картах с питанием 5 В они оба свободны; на картах 3,3 В контакт VS1# заземлен, а VS2# свободен. По этим линиям хост, допускающий оба варианта напряжения питания, определяет потребности установленной карты и подает соответствующее напряжение. Если хост не способен обеспечить требуемый номинал, он должен не подавать питание, а выдать сообщение об ошибке подключения. Карты обычно поддерживают управление энергопотреблением (APM), что особо актуально при автономном питании компьютера.

В стандарте PC Card выпускают самые разнообразные устройства — память, устройства хранения, коммуникационные средства, интерфейсные порты, игровые адаптеры, мультимедийные устройства и т. п., правда, все они существенно дороже своих крупногабаритных аналогов. Через слот PC Card портативные компьютеры могут подключаться к док-станциям, в которые может быть установлена обычная периферия. Недостаточно строгое следование производителей стандарту иногда приводит к проблемам совместимости.

Слоты PC Card подключаются к системной шине блокнотного ПК через мост; для компьютеров с внутренней шиной PCI это будет мост PCI–PC Card. В блокнотных ПК могут быть и слоты *Small PCI* (*SPCI*, см. выше), но они недоступны без вскрытия корпуса и не допускают «горячей» замены устройств.

Настольный ПК можно снабдить слотами PC Card с помощью специальной карты адаптера-моста, устанавливаемой в слот PCI или ISA. Сами слоты PC Card(1–2 штуки) оформляются в корпус трехдюймового устройства и выводятся на лицевую панель ПК; этот корпус соединяется с картой-мостом ленточным кабелем-шлейфом.

Назначение контактов разъемов для разных типов интерфейса приведено в табл. 6.10, назначение сигналов для интерфейсов карт памяти и ввода-вывода — в табл. 6.11. Для карт CardBus обозначение сигналов начинается с префикса «C», за которым следует имя сигнала, принятое для шины PCI.

Таблица 6.10. Разъем PC Card (PCMCIA)

№	Тип интерфейса				№	Тип интерфейса			
	16 бит			32 бита		16 бит			32 бита
	Mem	I/O+Mem	ATA	CardBus		Mem	I/O+Mem	ATA	CardBus
1	GND	GND	GND	GND	35	GND	GND	GND	GND
2	D3	D3	DD3	CAD0	36	CD1#	CD1#	CD1#	CCD1#
3	D4	D4	DD4	CAD1	37	D11	D11	DD11	CAD2
4	D5	D5	DD5	CAD3	38	D12	D12	DD12	CAD4
5	D6	D6	DD6	CAD5	39	D13	D13	DD13	CAD6
6	D7	D7	DD7	CAD7	40	D14	D14	DD14	Резерв
7	CE1#	CE1#	CS0#	CCBE0#	41	D15	D15	DD15	CAD8
8	A10	A10		CAD9	42	CE2#	CE2#	CS1#	CAD10
9	OE#	OE#	SELATA#	CAD11	43	VS1#	VS1#		CVS1
10	A11	A11		CAD12	44	Резерв	IORD#	DIOR#	CAD13
11	A9	A9	CS1#	CAD14	45	Резерв	IOWR#	DIOW#	CAD15
12	A8	A8		CCBE1#	46	A17	A17		CAD16
13	A13	A13		CPAR	47	A18	A18		Резерв
14	A14	A14		CPERR#	48	A19	A19		CBLOCK#
15	WE#	WE#		CGNT#	49	A20	A20		CSTOP#
16	READY	IREQ#	INTRQ	CINT#	50	A21	A21		CDEVSEL#
17	Vcc	Vcc	+5 B	Vcc	51	Vcc	Vcc	+5 B	Vcc
18	Vpp1	Vpp1		Vpp1	52	Vpp2	Vpp2		Vpp2
19	A16	A16		CCLK	53	A22	A22		CTRDY#
20	A15	A15		CIRDY#	54	A23	A23		CFRAME#
21	A12	A12		CCBE2#	55	A24	A24	M/S#	CAD17
22	A7	A7		CAD18	56	A25	A25	CSEL	CAD19
23	A6	A6		CAD20	57	VS2#	VS2#		CVS2
24	A5	A5		CAD21	58	RESET	RESET	RESET#	CRST#
25	A4	A4		CAD22	59	WAIT#	WAIT#	IORDY#	CSERR#

продолжение

Таблица 6.10 (*продолжение*)

№	Тип интерфейса				№	Тип интерфейса			
	16 бит			32 бита		16 бит			32 бита
	Mem	I/O+Mem	ATA	CardBus		Mem	I/O+Mem	ATA	CardBus
26	A3	A3		CAD23	60	Резерв	INPACK#	DMARQ	CREQ#
27	A2	A2	DA2	CAD24	61	REG#	REG#	DMACK#	CCBE3#
28	A1	A1	DA1	CAD25	62	BVD2	SPKR#	DASP#	CAUDIO
29	A0	A0	DA0	CAD26	63	BVD1	STSCHG#	PDIAG#	CSTSCHG
30	D0	D0	DD0	CAD27	64	D8	D8	DD8	CAD28
31	D1	D1	DD1	CAD29	65	D9	D9	DD9	CAD30
32	D2	D2	DD2	Резерв	66	D10	D10	DD10	CAD31
33	WP	IOIS16#		CCLKRUN#	67	CD2#	CD2#	CD2#	CCD2#
34	GND	GND	GND	GND	68	GND	GND	GND	GND

Таблица 6.11. Назначение сигналов карт памяти и ввода-вывода

Сигнал	I/O	Назначение
A[10:0]	I	Линии шины адреса
BVD1, BVD2	I/O	Battery Volt Detection — идентификаторы батарейного питания
STSCHG#	I/O	(IO) Сигнализация хосту о смене состояния RDY/BSY# и Write Protect. Использование этого сигнала контролируется регистром управления и состояния карты Card Config and Status Register
SPKR#	O	(IO) Дискретный аудиовыход (на динамик)
CD1#, CD2#	O	Card Detect — сигналы обнаружения (заземлены на карте), по которым хост определяет, что карта полностью вставлена в слот
CE1#, CE2#	I	(IO, Mem) Card Enable — выбор (разрешение) карты и определение разрядности передачи. Сигнал CE2# всегда относится к нечетному байту, CE1# — к четному или нечетному, в зависимости от A0 и CE2#. С помощью этих сигналов 8-битный хост может обмениваться с 16-битными картами по линиям D[7:0]
D[15:0]	I/O	Шина данных (у 8-битных сигналы D[15:8] отсутствуют)
INPACK#	O	(IO) Input Acknowledge — подтверждение ввода, ответ карты на сигнал IORD# (по этому сигналу хост открывает свои буферы данных)
IORD#	I	Строб команды чтения портов
IOWR#	I	Строб команды записи портов (данные должны фиксироваться по положительному перепаду)
OE#	I	Чтение данных из памяти, конфигурационных регистров и CIS
RDY/BSY#	I	Готовность карты к обмену данными (при высоком уровне)
IREQ#	O	Запрос прерывания (низким уровнем)
INTRQ	O	Запрос прерывания (высоким уровнем)

Сигнал	I/O	Назначение
REG#	I	Выбор памяти атрибутов (Mem). Для карт IO сигнал должен быть активен в циклах команд ввода-вывода. В режиме ATA пассивен (соединен с Vcc на стороне хоста)
RESET	I	Сброс (высоким уровнем)
VS1#, VS2#	O	Voltage Sense — сигналы определения номинала питания. Заземленный сигнал VS1# означает способность чтения карты при питании 3,3 В
WAIT#	O	Запрос (низким уровнем) на продление цикла обращения
WE#	I	Строб записи в память и конфигурационные регистры (в ATA не используется, соединяется хостом с Vcc)
WP	O	Write Protect — защита от записи (для карт памяти), запись в память возможна при низком уровне
IOCS16#	O	Разрешение 16-битного обмена

Интерфейс карт памяти и ввода-вывода прост — он практически совпадает с интерфейсом статической асинхронной памяти. Карта выбирается сигналами CE#, действующими одновременно с установленным адресом. Чтение памяти и конфигурационных регистров выполняется по сигналу OE#, запись — по сигналу WE#. Признаком, разделяющим в этих обращениях основную память и конфигурационные регистры, принадлежащие области памяти атрибутов карты, является сигнал REG#, действующий одновременно с CE# и адресом. Для обращения к портам ввода-вывода служат отдельные сигналы IORD# и IOWR#; во время их действия должен быть активен и сигнал REG#. В процессе обращения к портам карта может выдать признак возможности 16-битных обращений сигналом IOSC16# (как на шине ISA). Чтение порта устройство должно подтверждать сигналом INPACK#, устанавливаемым и снимаемым картой по сигналу CE#. Благодаря этому сигналу хост может убедиться в том, что он читает не пустой слот.

Для мультимедийных карт имеется возможность переключения интерфейса в специальный режим *ZV Port* (Zoomed Video), в котором организуется отдельный двухточечный интерфейс передачи данных между картой и хост-системой. По смыслу интерфейс напоминает коннектор VFC графических карт — выделенная шина для передачи видеоданных, не связанная с остальными шинами (и не загружающая их), но имеет иной протокол. В режиме ZV Port адресные линии A[25:4], а также линии BVD2/SPKR#, INPACK# и IOIS16# получают иное назначение — по ним передаются видеоданные и 4 цифровых аудиоканала. Для обычного интерфейса остаются лишь 4 адресные линии, позволяющие адресоваться к 16 байтам общей памяти и атрибутов карты.

Интерфейс порта ZV соответствует временным диаграммам CCIR601, что позволяет декодеру NTSC в реальном времени доставлять видеоданные с карты в экранный буфер VGA. Видеоданные могут поступать на карту как с внешнего видеовхода, так и с декодера MPEG.

Карты имеют специальное выделенное пространство памяти атрибутов, в котором находятся конфигурационные и управляющие регистры карты, предназначенные

для автоконфигурирования. Стандартом описан формат *информационной структуры карты* (*Card Information Structure*, *CIS*). Карты могут быть многофункциональными (например, комбинация модема и сетевого адаптера). В спецификации MFPC (*Multiple Function PC Cards*) для каждой функции предусматриваются отдельные конфигурационные регистры и определяются правила разделения (совместного использования) линии запроса прерывания.

Для устройств внешней памяти стандарт описывает форматы хранения данных, совместимые с FAT MS-DOS, а также ориентированные на флэш-память как основной носитель информации. Для непосредственного исполнения модулей ПО, хранящихся в ПЗУ карты, имеется спецификация *XIP* (*eXecute In Place*), описывающая программный интерфейс вызова этих модулей (вместо загрузки ПО в ОЗУ).

Стандарт описывает программный *интерфейс сервисов карт* (*Card Services*), обеспечивающий унификацию взаимодействия его *клиентов* (драйверов, прикладного ПО и утилит) с устройствами. Имеется также и *интерфейс сервисов сокета* (*Socket Services*), с помощью которого выполняются операции, связанные с обнаружением фактов подключения-отключения карт, их идентификации, конфигурирования питания и аппаратного интерфейса.

В стандарте имеются описания специфических особенностей, свойственных двум организациям, ведущим стандарт PC Card:

- PCMCIA описывает автоиндексируемую массовую память (*AIMS*) для хранения больших массивов данных (изображений, мультимедийных данных) на блочно-ориентированных устройствах. Имеется также спецификация 15-контактного экранированного разъема для подключения модемов и адаптеров локальной сети (15-pin Shielded Modem I/O connector) и 7-контактного для подключения модемов (7-pin Modem I/O connector);
- JEDIA для карт памяти предлагает формат файлов Small Block Flash Format, упрощающий файловую систему. Формат *SISRIF* (Still Image, Sound and Related Information Format) предназначен для записи изображений и звука на карты памяти. Имеется и спецификация для карт динамической памяти.

PCI в инструментальных системах: cPCI и PXI

Для устройств промышленного назначения в начале 1995 года был принят стандарт *Compact PCI*. Шина Compact PCI (cPCI) разрабатывалась на основе спецификации PCI 2.1. Этот стандарт принят Организацией производителей промышленных компьютеров PCIMG (PCI Industrial Computer Manufacturers Group). Шина отличается поддержкой большого количества слотов: 8 против 4 в обычной PCI. Как и PCI, шина поддерживает 32-битный и 64-битный обмен. Шина обладает всеми возможностями автоконфигурирования, присущими PCI. Кроме того, шина дает возможность программного прочтения «географического адреса» модуля. Географическая адресация дает дополнительную возможность идентификации физического местоположения модуля (хотя его можно определить и по номерам

шины и устройства, пользуясь стандартными конфигурационными функциями PCI). Конструктивно платы Compact PCI представляют собой еврокарты высотой 3U (100×160 мм) с двумя коннекторами (J1 и J2) или 6U (233,35×160 мм) с 4–5 коннекторами (J1...J5). На шасси разъемы обозначаются P1...P5; платы (модули) устанавливаются с шагом 20,32 мм (0,8 дюйма). Одно посадочное место (слот), как правило крайнее левое, отводится под системный модуль, остальные — под периферийные. В системный слот устанавливается *контроллер шины*, на который возлагаются функции арбитража и синхронизации. На его коннекторе шиной используется большее число контактов, чем на остальных. Вид шасси и модулей Compact PCI приведен на рис. 6.5, на рисунке видно символическое обозначение места контроллера (номер системного слота — в треугольнике, периферийных — в круге). На шасси может быть несколько *сегментов cPCI* — независимых шин; если требуется, связь между ними организуется через модули-мосты, устанавливаемые в слоты. В каждом сегменте должен быть собственный контроллер шины.

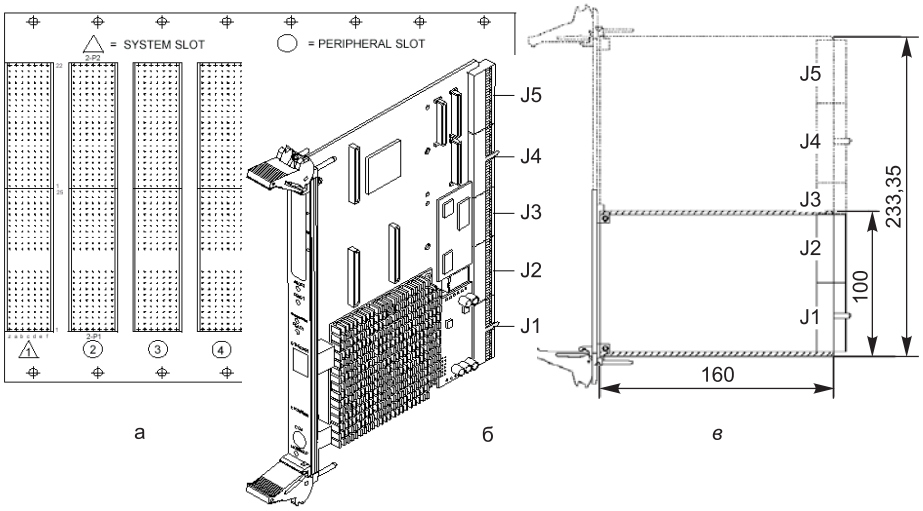


Рис. 6.5. Конструктив Compact PCI: *а* — шасси; *б* — вид полноразмерного модуля; *в* — варианты модулей

Коннекторы — 7-рядные экранированные штырьковые разъемы с шагом 2 мм между контактами, на кросс-плате — вилка, на модулях — розетки. Поскольку модули (и разъемы) обычно устанавливаются вертикально, ряды удобнее назвать колонками, что и подразумевается ниже. Из 7 колонок контактов (z, a, b, c, d, e, f) только 5 (a–e) используются для сигнальных цепей, а колонки z и f — только для экрана (соединяются с шиной `GND`)[1]. Контакты коннекторов имеют разную длину: более длинные контакты цепей питания при установке модуля соединяются раньше, а при вынимании разъединяются позже, чем сигнальные. Такое решение позволяет производить «горячую» замену модулей. Собственно шина использует только кон-

[1] В спецификации разъемы называют и 5-рядными.

некторы J1 и J2, плотно примыкающие друг к другу (они могут выглядеть как единый 47-позиционный разъем). Контакты J1 используются для сигналов 32-битной шины PCI; периферийная 32-битная плата может и не иметь разъема J2. Ряды 12–14 используются как ключи для вариантов с уровнями сигналов 5В/3,3В. Здесь возможны универсальные платы, но не допускаются универсальные шасси. Разъем J2 используется по-разному: системный контроллер использует его для сигналов арбитража и синхронизации, разводящихся по периферийным слотам радиально. На периферийных платах он может и отсутствовать. В 64-разрядных системах J2 используется для расширения шины; в 32-разрядных он может использоваться для сигналов ввода-вывода, разводящихся через шасси. На этот же разъем выведены сигналы географической адресации (которые можно и не использовать). Назначение контактов разъемов J1 и J2 Compact PCI версии 2.1 приведено в табл. 6.12. Разъемы J3...J5 отводятся для прикладного использования. Конструкция коннекторов позволяет применять для них специфические модификации (например, с разделяющим экраном и механическими ключами). В шине предусматривается наличие независимых источников питания + 5 В, + 3,3 В и ± 12 В.

Таблица 6.12. Назначение контактов разъема cPCI

Разъем, контакт	№	Ряд				
		a	b	C	d	e
J2	22	GA4	GA3	GA2	GA1	GA0
	21	CLK6	GND	RSV	RSV	RSV
	20	CLK5	GND	RSV	GND	RSV
	19	GND	GND	RSV	RSV	RSV
	18	BRSVP2A18	BRSVP2B18	BRSVP2C18	GND	BRSVP2E18
	17	BRSVP2A17	GND	PRST#	REQ6#	GNT6#
	16	BRSVP2A16	BRSVP2B16	DEG#	GND	BRSVP2E16
	15	BRSVP2A15	GND	FAL#	REQ5#	GNT5#
	14	AD35	AD34	AD33	GND	AD32
	13	AD38	GND	V(I/O)	AD37	AD36
	12	AD42	AD41	AD40	GND	AD39
	11	AD45	GND	V(I/O)	AD44	AD43
	10	AD49	AD48	AD47	GND	AD46
	9	AD52	GND	V(I/O)	AD51	AD50
	8	AD56	AD55	AD54	GND	AD53
	7	AD59	GND	V(I/O)	AD58	AD57
	6	AD63	AD62	AD61	GND	AD60
	5	C/BE5#	GND	V(I/O)	C/BE4#	PAR64
	4	V(I/O)	BRSVP2B4	C/BE7#	GND	C/BE6#

Разъем, контакт	№	Ряд				
		a	b	C	d	e
	3	CLK4	GND	GNT3#	REQ4#	GNT4#
	2	CLK2	CLK3	SYSEN#	GNT2#	REQ3#
	1	CLK1	GND	REQ1#	GNT1#	REQ2#
J1	25	5V	REQ64#	ENUM#	3.3V	5V
	24	AD1	5V	V(I/O)	AD0	ACK64#
	23	3.3V	AD4	AD3	5V	AD2
	22	AD7	GND	3.3V	AD6	AD5
	21	3.3V	AD9	AD8	M66EN	C/BE0#
	20	AD12	GND	V(I/O)	AD11	AD10
	19	3.3V	AD15	AD14	GND	AD13
	18	SERR#	GND	3.3V	PAR	C/BE1#
	17	3.3V	IPMB_SCL (SDONE)[1]	IPMB_SDA (SBO#)[1]	GND	PERR#
	16	DEVSEL#	GND	V(I/O)	STOP#	LOCK#
	15	3.3V	FRAME#	IRDY#	BD_SEL# (GND)[2]	TRDY#
	12–14	Зона ключа				
	11	AD18	AD17	AD16	GND	C/BE2#
	10	AD21	GND	3.3V	AD20	AD19
	9	C/BE3#	IDSEL	AD23	GND	AD22
	8	AD26	GND	V(I/O)	AD25	AD24
	7	AD30	AD29	AD28	GND	AD27
	6	REQ#	GND	3.3V	CLK	AD31
	5	BRSVP1A5	BRSVP1B5	RST#	GND	GNT#
	4	IPMB_PWR (BRSVP1A4)[1]	HEALTHY# (GND)[1]	V(I/O)	INTP	INTS
	3	INTA#	INTB#	INTC#	5V	INTD#
	2	TCK	5V	TMS	TDO	TDI
	1	5V	-12V	TRST#	+12V	5V

[1] Назначение в скобках — для старых версий.

[2] На системном слоте — GND.

В основном сигналы Compact PCI совпадают с сигналами обычной шины PCI (см. табл. 2.1 на стр. 41), назначение специфических сигналов приведено в табл. 6.13.

Таблица 6.13. Специфические сигналы шины Compact PCI

Сигнал	Назначение
BD_SEL#	Сигнал от модуля о том, что он установлен в слот и питание подано (подается через один из укороченных контактов, который соединяется после всех основных)
BRSVxxxx	Зарезервированные на будущее сигналы, шинно разведенные по слотам; xxxx обозначает позиционный номер контакта (BRSVP1A4 — на контакте A4 разъема P1)
CLK[0:6], GNT#[0:6], REQ#[0:6]	Сигналы, радиально разводящиеся от разъема J2 системного слота к периферийным (сигналы CLK0, GNT0# и REQ0# расположены на местах CLK, GNT# и REQ# разъема J1)
DEG#	Предупреждение о деградации питания
ENUM#	Все аппаратные модули установлены, можно производить нумерацию и конфигурирование устройств
FAL#	Отказ питания
GA0-GA4	Географический адрес. Коммутацией на «землю» для каждого слота задается его двоичный географический адрес на шасси
SMB_SDA, SMB_SCL, SMB_ALERT#	Сигналы шины SMBus (только на системном слоте) Сигнал прерывания по шине SMBus
HEALTHY#	Сигнал от модуля, что он получает нормальное питание (PwrGood) и сигнал его сброса снят
INTP, INTS	Прерывания от первичного и вторичного контроллеров IDE
IPMB_PWR,	Батарейное питание шины IPMB (Independent Platform Management Bus, независимая последовательная шина управления платформой)
IPMB_SCL, IPMB_SDA	Синхронизация и данные шины IPMB
PRST#	Push Button Reset, сигнал от кнопки «Сброс»
RSV	Резерв на будущее
SYSEN#	Идентификация системного слота (на системном слоте контакт заземлен, что позволяет модулю опознать установку в это место)
UNC	Не подключен

На базе шины Compact PCI фирмой National Instruments разработана *спецификация PXI* (PCI eXtensions for Instrumentation — расширение PCI для инструментальных систем) в тех же конструктивах. По сравнению с cPCI в PXI более жестко определяется местоположение модулей. На шасси левый слот отводится для *контроллера шины*, следующий за ним — для *контроллера синхронизации* (его номер пишут в ромбе), остальные — для *периферийных модулей*. При необходимости контроллер может расширяться влево, занимая дополнительные слоты, разъемы которых не связаны с общей шиной.

В шине PXI часть контактов J2/P2, определенных в Compact PCI как резервные, предназначаются для организации дополнительных локальных шин и синхронизации. Резервными остались только PXI_BRSVA15 и PXI_BRSVB4, разведенные по всем

слотам шасси. Топологию соединений на шасси PXI иллюстрирует рис. 6.6. Назначение контактов J2 для периферийных модулей, системного контроллера и контроллера синхронизации приведено в табл. 6.14–6.16. Специфические сигналы шины PXI имеют обозначения, начинающиеся с префикса PXI_. Назначение контактов J1 см. в табл. 6.12.

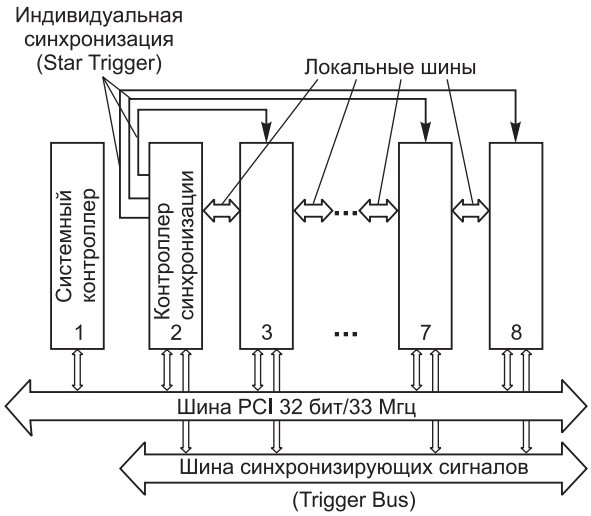


Рис. 6.6. Топология соединений в PXI

В инструментальных системах зачастую требуется синхронизация разных модулей, для этого в PXI имеются специальные сигнальные линии. *Шина синхронизирующих сигналов Trigger Bus* (8 линий) PXI_TRIG[0:7] объединяет все слоты одного сегмента PXI, за исключением системного. Кроме того, имеются 13 линий *индивидуальной синхронизации* (*Star Trigger*), звездообразно соединяющих слот контроллера синхронизации с остальными периферийными слотами всего шасси (может распространяться и на два сегмента PXI). Каждая из линий PXI_STAR[0:12] слота контроллера синхронизации соединяется с линией PXI_STAR своего слота. Разводка линий обеспечивает идентичность задержек, вносимых ими в распространение сигнала между слотом контроллера синхронизации и периферийными слотами. Линии PXI_TRIG[0:7] и PXI_STAR[0:12] могут использоваться двояко: как для подачи команд запуска модулям от контроллера синхронизации, так и для сообщения модулями своего состояния (зависит от приложения шасси). Для прецизионной синхронизации имеется сигнал опорной частоты 10 МГц PXI_CLK10, который шасси синхронно (со сдвигом не более 1 нс) доставляет ко всем слотам. Для каждого слота предоставляется отдельный выход буфера; источником сигнала может быть как шасси, так и контроллер синхронизации (через сигнал PXI_CLK10_IN).

Локальные шины в PXI предназначены для связи соседних пар слотов. Локальные шины объединяют смежные слоты попарно (исключая слот системного контроллера), образуя *цепочку устройств* (daisy chain). Каждая локальная шина имеет

13 линий, соединяющих цепи PXI_LBR[0:12] левого слота с цепями PXI_LBL[0:12] правого слота пары. Линии могут использоваться как для цифровых, так и аналоговых (до 48 В, 200 мА) сигналов. Цепи PXI_LBR[0:12] последнего (самого правого) слота могут выводиться на внешний разъем шасси. Для слота контроллера синхронизации линии PXI_LBL[0:12] недоступны — их контакты заняты звездообразными сигналами синхронизации.

Кроме механических и электрических характеристик PXI определяет ПО модулей: основной ОС считается Windows NT/2000/9x, и модули должны поставляться с соответствующими драйверами. Это экономит время, необходимое для системной интеграции. В качестве средств разработки ПО предлагается использовать пакеты LabVIEW, LabWindows/CVI фирмы National Instruments; Visual Basic, Visual C/C++ от Microsoft и Turbo C/C++ от Borland. Модули PXI совместимы с шиной Compact PCI, а модули Compact PCI — с шиной PXI. Однако все преимущества спецификации реализуются только при установке модулей PXI в шину PXI.

Таблица 6.14. Разъем J2/P2 PXI для периферийного слота

Контакт	a	b	c	d	e
22	GA4	GA3	GA2	GA1	GA0
21	PXI_LBR0	GND	PXI_LBR1	PXI_LBR2	PXI_LBR3
20	PXI_LBR4	PXI_LBR5	PXI_LBL0	GND	PXI_LBL1
19	PXI_LBL2	GND	PXI_LBL3	PXI_LBL4	PXI_LBL5
18	PXI_TRIG3	PXI_TRIG4	PXI_TRIG5	GND	PXI_TRIG6
17	PXI_TRIG2	GND	RSV	PXI_STAR	PXI_CLK10
16	PXI_TRIG1	PXI_TRIG0	RSV	GND	PXI_TRIG7
15	PXI_BRSVA15	GND	RSV	PXI_LBL6	PXI_LBR6
Ряды 5–14 как у Compact PCI					
4	V(I/O)	PXI_BRSVB4	C/BE7#	GND	C/BE6#
3	PXI_LBR7	GND	PXI_LBR8	PXI_LBR9	PXI_LBR10
2	PXI_LBR11	PXI_LBR12	UNC	PXI_LBL7	PXI_LBL8
1	PXI_LBL9	GND	PXI_LBL10	PXI_LBL11	PXI_LBL12

Таблица 6.15. Разъем J2/P2 PXI для системного слота

Контакт	a	b	c	d	e
22	GA4	GA3	GA2	GA1	GA0
21	CLK6	GND	RSV	RSV	RSV
20	CLK5	GND	RSV	GND	RSV
19	GND	GND	SMB_SDA	SMB_SCL	SMB_ALERT#
18	PXI_TRIG3	PXI_TRIG4	PXI_TRIG5	GND	PXI_TRIG6
17	PXI_TRIG2	GND	PRST#	REQ6#	GNT6#

Контакт	a	b	c	d	e
16	PXI_TRIG1	PXI_TRIG0	DEG#	GND	PXI_TRIG7
15	PXI_BRSVA15	GND	FAL#	REQ5#	GNT5#
Ряды 5–14 как у Compact PCI					
4	V(I/O)	PXI_BRSVB4	C/BE7#	GND	C/BE6#
3	CLK4	GND	GNT3#	REQ4#	GNT4#
2	CLK2	CLK3	SYSEN#	GNT2#	REQ3#
1	CLK1	GND	REQ1#	GNT1#	REQ2#

Таблица 6.16. Разъем J2/P2 PXI для слота контроллера синхронизации

Контакт	a	b	c	d	e
22	GA4	GA3	GA2	GA1	GA0
21	PXI_LBR0	GND	PXI_LBR1	PXI_LBR2	PXI_LBR3
20	PXI_LBR4	PXI_LBR5	PXI_STAR0	GND	PXI_STAR1
19	PXI_STAR2	GND	PXI_STAR3	PXI_STAR4	PXI_STAR5
18	PXI_TRIG3	PXI_TRIG4	PXI_TRIG5	GND	PXI_TRIG6
17	PXI_TRIG2	GND	RSV	PXI_CLK10_IN	PXI_CLK10
16	PXI_TRIG1	PXI_TRIG0	RSV	GND	PXI_TRIG7
15	PXI_BRSVA15	GND	RSV	PXI_STAR6	PXI_LBR6
Ряды 5–14 как у Compact PCI					
4	V(I/O)	PXI_BRSVB4	C/BE7#	GND	C/BE6#
3	GND	PXI_LBR8	PXI_LBR9	PXI_LBR10	
2	PXI_LBR12	UNC	PXI_STAR7	PXI_STAR8	
1	GND	PXI_STAR10	PXI_STAR11	PXI_STAR12	

ГЛАВА 7

Порт графического акселератора — AGP

Порт AGP (Accelerated Graphic Port — порт ускоренной графики) был введен для подключения графических адаптеров с 3D-акселераторами. Такой адаптер содержит *акселератор* — специализированный графический процессор; *локальную память*, используемую как видеопамять и как локальное ОЗУ графического процессора; *управляющие и конфигурационные регистры*, доступные как локальному, так и центральному процессорам. Акселератор может обращаться и к локальной памяти, и к системному ОЗУ, в котором для него могут храниться наборы данных, не умещающиеся в локальной памяти (как правило, текстуры большого объема). Основная идея AGP заключается в предоставлении акселератору максимально быстрого доступа к системной памяти (локальная ему и так близка), более приоритетного, чем доступ к ОЗУ со стороны других устройств.

Порт AGP представляет собой 32-разрядный параллельный синхронный интерфейс с тактовой частотой 66 МГц; большая часть сигналов позаимствована из шины PCI. Однако в отличие от PCI порт AGP представляет собой двухточечный интерфейс, соединяющий графический акселератор с памятью и системной шиной процессора каналами данных чипсета системной платы, не пересекаясь с «узким местом» — шиной PCI. Обмен через порт может происходить как по протоколу PCI, так и по протоколу AGP. Отличительные особенности порта AGP:

- конвейеризация обращений к памяти;
- умноженная (2x/4x/8x) частота передачи данных (относительно тактовой частоты порта);
- «внеполосная» подача команд (SBA), обеспеченная демультиплексированием шин адреса и данных.

Идею конвейеризации обращений к памяти иллюстрирует рис. 7.1, где сравниваются обращения к памяти по шине PCI и через порт AGP. В PCI во время реакции памяти на запрос шина простаивает (но не свободна). Конвейерный доступ AGP позволяет в это время передавать следующие запросы, а потом получить поток ответов.

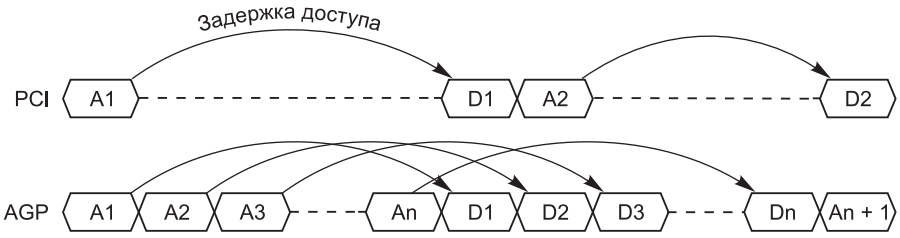


Рис. 7.1. Циклы обращения к памяти PCI и AGP

Умножение частоты передачи данных обеспечивает при частоте 66 МГц пиковую пропускную способность до 533 (2x), 1066 (4x) и 2132 Мбайт/с в режиме 8x. Выше 66 МГц тактовую частоту официально не поднимают.

Демультиплексирование (разделение) шины адреса и данных сделано несколько необычным образом. С целью экономии числа интерфейсных линий шину адреса и команды в демультиплексированном режиме AGP представляют всего 8 линий *SBA* (SideBand Address), по которым команда, адрес и значение длины передачи передаются последовательно за несколько тактов. Поддержка демультиплексированной адресации не являлась обязательной для устройства AGP 1.0, поскольку имеется альтернативный способ подачи адреса по шине `AD`. В версии AGP 2.0 она стала обязательной, а в 3.0 это уже единственный способ подачи адреса.

Отметим, что порт AGP дает только преимущества, которые могут быть реализованы лишь при поддержке аппаратными средствами графического адаптера и специального ПО. Графический адаптер с интерфейсом AGP может реально вести себя по-разному:

- не задействовать конвейеризацию, а использовать только *быструю запись PCI* (Fast Write);
- не работать с текстурами, расположенными в системной памяти, но использовать более быстрый обмен данными между памятью и локальным буфером;
- использовать все возможности порта, когда акселератор имеет быстрый доступ к системной памяти, а центральный процессор может быстро закачивать данные в локальную память адаптера.

Порт AGP содержит практически полный набор сигналов шины PCI и дополнительные сигналы AGP. Устройство, подключаемое к порту AGP, может предназначаться как исключительно для операций AGP, так и быть комбинацией AGP + PCI. Акселератор адаптера является *мастером* (ведущим устройством) *порта AGP*, свои запросы он может выполнять как в режиме AGP, так и в режиме PCI (см. главу 2). В режиме AGP обмены выполняются с поддержкой (или без поддержки) таких свойств, как внеполосная адресация (SBA) и скорость 2x/4x/8x. Для транзакций в режиме AGP ему доступно только системное ОЗУ (но не локальная память устройств PCI). Кроме того, адаптер является *целевым устройством PCI*, для которого, кроме обычных команд PCI, может поддерживаться (или не поддерживаться) быстрая запись (Fast write) со стороны процессора (сокоростью 2x/4x/8x).

В качестве целевого устройства адаптер выступает при обращениях ЦП к его локальной памяти, регистрам ввода-вывода и конфигурационного пространства.

Устройство, подключаемое к AGP, обязательно должно выполнять функции ведущего устройства AGP (иначе порт AGP для него теряет смысл) и функции ведомого устройства PCI со всеми его атрибутами (конфигурационными регистрами и т. п.); дополнительно оно может быть и ведущим устройством PCI.

Порт AGP позволяет акселератору работать в двух режимах — DMA и DIME (DIrect Memory Execute). *В режиме DMA* акселератор при вычислениях рассматривает локальную память как первичную, а когда ее недостаточно, подкачивает в нее данные из основной памяти. *В режиме DIME* (он же режим исполнения, Executive Mode) локальная память и основная память для акселератора логически равнозначны и располагаются в едином адресном пространстве. В режиме DMA для трафика порта характерны длительные блочные передачи, в режиме DIME трафик порта будет насыщен короткими произвольными запросами.

Спецификации AGP разрабатывались фирмой Intel на базе шины PCI 2.1 с частотой 66 МГц; пока имеется три основные версии спецификаций:

- AGP 1.0 (1996 год) — определен порт с конвейерным обращением к памяти, двумя альтернативными способами подачи команд: внеполосной (по шине SBA) и внутриполосной (по сигналу PIPE#). Режимы передачи 1x/2x, питание интерфейса — 3,3 В;
- AGP 2.0 (1998 год) — добавлена возможность быстрой записи в режиме PCI (Fast Writes), а также режим 4x с питанием 1,5 В;
- AGP 3.0 (2002 год, проект назывался AGP8X) — добавлен режим 8x с питанием 0,8 В и динамическим инвертированием байтов, отменены скорости 1x и 2x; оставлен один способ подачи команд — внеполосный (SBA); исключены некоторые команды AGP; введены команды изохронного обмена; введена возможность выбора размера страниц, описанных в GART; введена селективная поддержка когерентности при обращениях к разным страницам в пределах GART.

Порт AGP предназначен только для подключения интеллектуального графического адаптера, имеющего 3D-акселератор, причем только одного. Системная логика порта AGP отличается сложным контроллером памяти, который выполняет глубокую буферизацию и высокопроизводительное обслуживание запросов AGP (от адаптера) и других своих клиентов — центрального процессора (одного или нескольких) и шины PCI. Единственный вариант подключения нескольких адаптеров с AGP — организация на системной плате нескольких портов AGP, что вряд ли будет применяться.

AGP может реализовать всю пропускную способность 64-битной системы памяти современного компьютера. При этом возможны конкурирующие обращения к памяти как со стороны процессора, так и со стороны мостов шин PCI. Фирма Intel впервые ввела поддержку AGP в чипсеты для процессоров P6, конкуренты используют AGP и в системных платах для процессоров с интерфейсом Pentium (сокет *Super 7*). В настоящее время порт AGP имеется практически во всех системных платах для PC-совместимых компьютеров и других платформ (даже Macintosh).

Протоколы транзакций

Транзакции в режиме PCI, инициируемые акселератором, начинаются с подачи сигнала FRAME# и выполняются обычным для PCI способом (см. главу 2). Заметим, что при этом на все время транзакции шина AD занята, причем транзакции чтения памяти занимают шину на большее число тактов, чем транзакции записи, — после подачи адреса неизбежны такты ожидания на время доступа к памяти. Запись на шине происходит быстрее — данные записи задатчик посылает сразу за адресом, а на время доступа к памяти они «оседают» в буфере контроллера памяти. Контроллер памяти позволяет завершить транзакцию и освободить шину до физической записи в память.

Конвейерные транзакции AGP (команды AGP) инициируются только акселератором; логикой AGP они ставятся в очереди на обслуживание и исполняются в зависимости от приоритета, порядка поступления запросов и готовности данных. Эти транзакции могут быть адресованы акселератором только к системному ОЗУ. Если устройству на AGP требуется обратиться к локальной памяти каких-либо устройств PCI, то оно должно выполнять эти транзакции в режиме PCI.

Обращения со стороны процессора (или задатчиков шины PCI), адресованные к устройству на AGP, отрабатываются им как ведомым устройством PCI, однако имеется возможность *быстрой записи* в локальную память — *FW* (Fast Write), в которой данные передаются на скорости AGP (2x/4x/8x), и управление потоком их передач ближе к протоколу AGP, нежели PCI. Транзакции *FW* инициируются процессором и предназначены для принудительного «заталкивания» данных в локальную память акселератора.

Концепцию *конвейера AGP* иллюстрирует рис. 7.2. Порт AGP может находиться в одном из четырех состояний:

- *IDLE* — покой;
- *DATA* — передача данных конвейеризированных транзакций;
- *AGP* — постановка в очередь команды AGP;
- *PCI* — выполнение транзакции в режиме PCI.

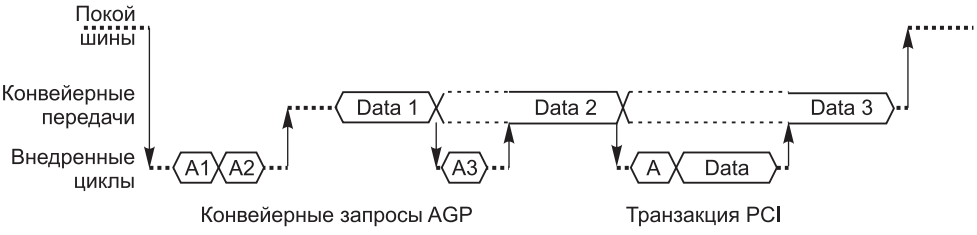


Рис. 7.2. Конвейер AGP

Из состояния покоя *IDLE* порт может вывести запрос транзакции PCI (как от акселератора, так и с системной стороны) или запрос AGP (только от акселератора). В состоянии *PCI* транзакция PCI выполняется целиком, от подачи адреса и коман-

ды до завершения передачи данных. В состоянии *AGP* ведущее устройство передает только команду и адрес для транзакции (по сигналу PIPE# или через шину SBA), ставящейся в *очередь*; несколько запросов могут следовать сразу друг за другом. В состояние *DATA* порт переходит, когда у него в очереди имеется необслуженная команда, готовая к исполнению. В этом состоянии происходит передача данных для команд, стоящих в очереди. Это состояние может прерываться вторжением запросов *PCI* (для выполнения целой транзакции) или *AGP* (для постановки в очередь новой команды), но прерывание[1] возможно только на границах данных транзакций AGP. Когда порт AGP обслужит все команды, он снова переходит в состояние покоя. Все переходы происходят под управлением арбитра порта AGP, реагирующего на поступающие запросы (сигнал REQ# от акселератора и внешние обращения от процессора или других устройств PCI) и ответы контроллера памяти.

Транзакции AGP отличаются от транзакций PCI некоторыми деталями реализации:

- фаза данных отделена от фазы адреса, чем и обеспечивается конвейеризация;
- используется собственный набор команд;
- транзакции адресуются только к системной памяти, используя пространство физических адресов (как и в PCI). Транзакции могут иметь длину, кратную 8 байтам, и начинаться только по 8-байтной границе. Транзакции чтения иного размера должны выполняться только в режиме PCI; транзакции записи могут использовать сигналы C/BE[3:0]# для маскирования лишних байтов;
- длина транзакции явно указывается в запросе;
- конвейерные запросы не гарантируют когерентности памяти и кэша. Для операций, требующих когерентности, должны использоваться транзакции PCI. В AGP 3.0 возможно указать области памяти, для которых когерентность обеспечивается и для конвейерных транзакций.

Возможны *два способа подачи команд AGP* (постановок и запросов в очередь), из которых в текущей конфигурации выбирается один, причем изменение способа «на ходу» не допускается:

- запросы вводятся по шине AD[31:0] и C/BE[3:0] с помощью сигнала PIPE#, по каждому фронту CLK ведущее устройство передает очередное двойное слово запроса вместе с кодом команды;
- команды подаются через *внеполосные* (sideband) линии адреса SBA[7:0]. «Внеполосность» означает, что эти сигналы используются независимо от занятости шины AD. Синхронизация подачи запросов зависит от режима (1x/2x/4x/8x).

При подаче команд по шине AD во время активности сигнала PIPE# код команды AGP (CCCC) кодируется сигналами C/BE[3:0], при этом на шине AD помещаются начальный адрес (на AD[31:3]) и длина *n* (на AD[2:0]) запрашиваемого блока данных. Определены следующие команды (в скобках указан код CCCC):

---

[1] Здесь прерыванием называем вклинивание команд AGP и транзакций PCI в поток передач данных; к прерываниям процессора это не имеет отношения.

- *Read* (0000) — чтение из памяти (n + 1) учетверенных слов (по 8 байт) данных, начиная с указанного адреса;
- *HP Read* (0001) — чтение с высоким приоритетом (упразднено в AGP 3.0);
- *Write* (0100) — запись в память;
- *HP Write* (0101) — запись с высоким приоритетом (упразднено в AGP 3.0);
- *Long Read* (1000) — «длинное» чтение (n + 1)×4 учетверенных слов (до 256 байт данных, упразднено в AGP 3.0);
- *HP Long Read* (1001) — «длинное» чтение с высоким приоритетом (упразднено в AGP 3.0);
- *Flush* (1010) — очистка, выгрузка данных всех предыдущих команд записи по адресам назначения (на порте AGP выглядит как чтение, возвращающее произвольное учетверенное слово в качестве подтверждения исполнения; адрес и длина, указанные в запросе, значения не имеют);
- *Fence* (1100) — установка «ограждений», позволяющих низкоприоритетному потоку записей не пропускать чтения;
- *Dual Address Cycle, DAC* (1101) — двухадресный цикл для 64-битной адресации: в первом такте по AD передаются младшая часть адреса и длина запроса, а во втором — старшая часть адреса (по AD) и код исполняемой команды (по C/BE[3:0]).

Для изохронных передач в AGP 3.0 выделены специальные команды (см. далее).

При внеполосной подаче команд по шине SBA[7:0] передаются 16-битные посылки четырех типов. Тип посылки кодируется старшими битами:

- тип 1: 0AAA AAAA AAAA ALLL — поле длины (LLL) и младшие биты адреса (A[14:03]);
- тип 2: 10CC CCRA AAAA AAAA — код команды (CCCC) и средние биты адреса (A[23:15]);
- тип 3: 110R AAAA AAAA AAAA — старшие биты адреса (A[35:24]);
- тип 4: 1110 AAAA AAAA AAAA — дополнительные старшие биты адреса, если требуется 64-битная адресация.

Посылка из всех единиц является пустой командой (NOP); такие посылки означают покой шины SBA. Биты «R» зарезервированы. Посылки типов 2, 3 и 4 являются «*липкими*» (sticky) — значения, ими определяемые, сохраняются до введения новой посылки того же типа. Постановку команды в очередь инициирует посылка типа 1, задающая длину транзакции и ее младшие адреса, — код команды и остальная часть адреса должны быть определены ранее введенными посылками типов 2–4. Такой способ очень экономно использует такты шины для подачи команд при пересылках массивов. Каждая 2-байтная посылка передается по 8-битной шине SBA за два приема (сначала старший, потом младший байт). Синхронизация байтов зависит от режима порта:

- в режиме 1x каждый байт передается по фронту CLK; начало посылки (старший байт) определяется по получению байта, отличного от 11111111b, по последующему фронту передается младший байт. Очередная команда (посылкой типа 1)

может вводиться *за каждую пару тактов* CLK (при условии, что код команды и старший адрес уже введены предыдущими посылками). Полный цикл ввода команды занимает 10 тактов;

- в режиме 2x для SBA используется отдельный строб SB_STB, по его спаду передается старший байт, а по последующему фронту — младший. Частота этого строба (но не фаза) совпадает с CLK, так что очередная команда может вводиться *в каждом такте* CLK;
- в режиме 4x используется еще и дополнительный (инверсный) строб SB_STB#. Старший байт фиксируется по спаду SB_STB, а младший — по последующему спаду SB_STB#. Частота стробов в два раза выше, чем CLK, так что в каждом такте CLK может вводиться *пара посылок*. Однако мастер AGP может вводить в каждом такте не более одной посылки типа 1, то есть ставить в очередь не более одного запроса;
- В режиме 8x стробы называются иначе (SB_STBF и SB_STBS), их частота в четыре раза выше CLK, так что в каждом такте CLK умещаются уже 4 посылки. Однако темп постановки команд в очередь все равно ограничен одной командой за такт CLK.

В ответ на полученные команды порт AGP выполняет *передачи данных*, причем фаза данных AGP явно не привязана к фазе команды/адреса. Фаза данных будет вводиться портом AGP по готовности системной памяти к запрашиваемому обмену.

Передачи данных AGP выполняются, когда шина находится в состоянии *DATA*. Фазы данных вводит порт AGP (системная логика), исходя из порядка ранее пришедших к нему команд от акселератора. Акселератор узнает о назначения шины AD в последующей транзакции по сигналам ST[2:0] (действительны только во время сигнала GNT#, коды 100–110 зарезервированы):

- 000 — устройству будут передаваться данные низкоприоритетного запроса чтения (в AGP 3.0 — просто асинхронного чтения), ранее поставленного в очередь, или выполняется очистка;
- 001 — устройству будут передаваться данные высокоприоритетного запроса чтения (резерв в AGP 3.0);
- 010 — устройство должно будет предоставлять данные низкоприоритетного запроса записи (в AGP 3.0 — просто асинхронной записи);
- 011 — устройство должно будет предоставлять данные высокоприоритетного запроса записи (резерв в AGP 3.0);
- 111 — устройству разрешается поставить в очередь команду AGP (сигналом PIPE#) или начать транзакцию PCI (сигналом FRAME#);
- 110 — цикл калибровки приемопередатчиков (в AGP 3.0 для скорости 8x).

Акселератор узнает лишь тип и приоритет команды, результаты которой последуют в данной транзакции. Какую именно команду из очереди отрабатывает порт, акселератор определяет сам, так как именно он ставил их в очередь (ему известен порядок). Никаких тегов транзакций (как, например, в системной шине процессоров P6 или в PCI-X) в интерфейсе AGP нет. Имеются только независимые очереди для каждого типа команд (*чтение низкоприоритетное*, *чтение высокоприори-*

тетное, *запись низкоприоритетная*, *запись высокоприоритетная*). Фазы исполнения команд разных очередей могут чередоваться произвольным образом; порт имеет право исполнять их в порядке, оптимальном с точки зрения производительности. Реальный порядок исполнения команд (чтения и записи памяти) тоже может изменяться. Однако для каждой очереди порядок выполнения всегда совпадает с порядком подачи команд (об этом знают и акселератор, и порт). В AGP 3.0 приоритеты очередей отменили, но ввели возможность изохронных транзакций.

Запросы AGP с высоким приоритетом для арбитра системной логики являются более приоритетными, чем запросы от центрального процессора и ведущих устройств шины PCI. Запросы AGP с низким приоритетом для арбитра имеют приоритет ниже, чем от процессора, но выше, чем от остальных ведущих устройств. Хотя принятый протокол никак явно не ограничивает глубину очередей, спецификация AGP формально ее ограничивает до 256 запросов. На этапе конфигурирования устройства система PnP устанавливает реальное ограничение (в конфигурационном регистре акселератора) в соответствии с его возможностями и возможностями системной платы. Программы, работающие с акселератором (исполняемые и локальным, и центральным процессорами), не должны допускать превышения числа необслуженных команд в очереди (у них для этого имеется вся необходимая информация).

При передаче данных AGP управляющие сигналы, заимствованные от PCI, имеют почти такое же назначение, что и в PCI. Передача данных AGP в режиме 1x очень похожа на циклы PCI, но немного упрощена процедура квитирования (поскольку это выделенный порт и обмен выполняется только с быстрым контроллером системной памяти). В режимах 2x/4x/8x имеется специфика стробирования:

- *в режиме 1x* данные (4 байта на `AD[31:0]`) фиксируются получателем по положительному перепаду каждого такта `CLK`, что обеспечивает пиковую скорость 66,6×4 = 266 Мбайт/с;
- *в режиме 2x* используются стробы данных `AD_STB0` и `AD_STB1` для линий `AD[0:15]` и `AD[16:31]` соответственно. Стробы формируются источником данных, приемник фиксирует данные и по спаду, и по фронту строба. Частота стробов совпадает с частотой `CLK`, что и обеспечивает пиковую скорость 66,6×2×4 = 533 Мбайт/с;
- *в режиме 4x* используются еще и дополнительные (инверсные) стробы `AD_STB0#` и `AD_STB1#`. Данные фиксируются по спадам и прямых и инверсных стробов (пары стробов могут использоваться и как два отдельных сигнала, и как один дифференциальный). Частота стробов в два раза выше, чем `CLK`, что и обеспечивает пиковую скорость 66,6×2×2×4 = 1066 Мбайт/с;
- *в режиме 8x* пары стробов получили новые названия `AD_STBF[1:0]` (First — первый) и `AD_STBS[1:0]` (Second — второй), четные порции данных защелкиваются по положительному перепаду первого, нечетные — по перепаду второго. Частота переключения каждого из этих стробов в четыре раза выше `CLK`, стробы сдвинуты относительно друг друга на половину своего периода, чем и обеспечивается восьмикратная частота стробирования информации на линиях `AD`. Отсюда пиковая скорость 66,6×4×2×4 = 2132 Мбайт/с.

Порт AGP должен отслеживать состояние готовности буферов акселератора к посылке или получению данных транзакций, поставленных в очередь. Сигналом RBF# (Read Buffer Full) акселератор может информировать порт о неготовности к приему данных низкоприоритетных транзакций чтения (к приему высокоприоритетных он должен быть всегда готов). Сигналом WBF# (Write Buffer Full) он информирует о неспособности принять первую порцию данных быстрой записи *FW*.

Трансляция адресов — GART и апертура AGP

Порт AGP обеспечивает *трансляцию логических адресов*, фигурирующих в запросах акселератора к системной памяти, в *физические адреса*, согласуя видение ОЗУ программой, выполняемой акселератором, и программой, выполняемой на центральном процессоре. Трансляция осуществляется в постраничном базисе (по умолчанию размер страницы — 4 Кбайт), принятом в системе виртуальной памяти с подкачкой страниц по запросу, используемой в процессорах x86 (и других современных процессорах). Трансляции подлежат обращения, попадающие в *апертуру AGP*, — область физических адресов памяти, лежащую выше границы ОЗУ и, как правило, примыкающую к области локальной памяти адаптера (рис. 7.3). Таким образом, при работе в режиме DIME акселератору доступна непрерывная область памяти, часть которой составляет локальная память адаптера. Остальная часть адресуемой им памяти отображается на системное ОЗУ через апертуру с помощью *таблицы* GART (Graphics Address Remapping Table — таблица переопределения графических адресов). Каждый элемент этой таблицы описывает свою страницу в области апертуры. В каждом элементе GART есть признак его действительности; в действительных элементах указывается адрес страницы физической памяти, на которую отображается соответствующая область апертуры. Таблица GART физически находится в системном ОЗУ, она выровнена по границе 4-килобайтовой страницы, на ее начало указывают конфигурационные регистры порта AGP.

Размер апертуры AGP (определяющий и размер таблицы GART) задается программированием регистров чипсета. Путем настройки параметров CMOS Setup или внешних утилит его можно задать размером 8, 16, 32...256 и более Мбайт. Оптимальное значение апертуры зависит от объема памяти и используемых программ, но можно ориентироваться на половину объема системного ОЗУ. Заметим, что назначение размера апертуры в большинстве случаев еще не означает отчуждения указанного объема ОЗУ от системного ОЗУ — это лишь предельный размер памяти, которую ОС будет выделять акселератору по его запросам. Пока акселератору хватает своей локальной памяти, он не запрашивает память в системном ОЗУ. Только при нехватке локальной памяти он будет динамически запрашивать дополнительную, и эти запросы будут удовлетворяться в пределах установленной апертуры. По мере уменьшения потребности в дополнительной памяти она будет динамически высвобождаться для обычных нужд операционной системы. Правда, если у графического акселератора локальной памяти нет вообще (в дешевых ин-

тегрированных адаптерах), то от ОЗУ статически (на все время работы) отчуждается часть памяти (хотя бы под экранный буфер). Это видно по уменьшенному размеру памяти, который POST показывает в начале процесса загрузки.

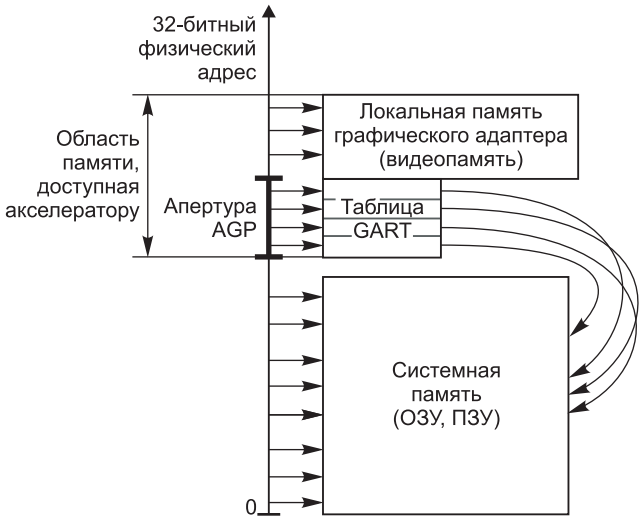


Рис. 7.3. Адресация памяти в системе с AGP

Логика порта обеспечивает *когерентность* всех кэш-памятей системы для обращений мастера AGP *вне* диапазона адресов апертуры. В AGP 3.0 введена возможность выборочного обеспечения когерентности для обращений внутри апертуры. В предыдущих версиях предполагалось, что область памяти *внутри* апертуры должна быть просто некэшируемой. Поскольку эту память предполагалось использовать для хранения текстур (они достаточно статичны), такое упрощение вполне приемлемо.

Изохронные транзакции в AGP 3.0

Для поддержки изохронных транзакций в AGP 3.0 введены новые коды команд и состояний, а также конфигурационные регистры, управляющие изохронным соединением. Изохронные транзакции может выполнять *мастер AGP* только через область апертуры AGP, причем с областью памяти, для которой не обеспечивается когерентность (чтобы избежать непрогнозируемых задержек, связанных с выгрузками «грязных» строк). Изохронные транзакции возможны только на скорости 8x. Соглашение на изохронный обмен описывается набором параметров:

- N — число транзакций чтения или записи за период времени T;
- V — размер блока данных в изохронной транзакции;
- L — максимальная задержка (латентность) доставки данных от подачи команды (в периодах T).

При этом пропускная способность $BW = N \times Y/T$, интервал T принят 1 мкс. Размер блока V может принимать значение 32, 64, 128 или 256 байт; для транзакций чтения, длина которых может принимать те же значения, изохронный блок передается за одну транзакцию. Длина транзакций записи может быть 32 или 64 байт, так что один блок будет передаваться за 1, 2 или 4 транзакции. В зависимости от мощности подсистемы памяти порт AGP может выдерживать изохронный трафик, достаточный для различных применений:

- видеозахват (в настольных ПК): 128 Мбайт/с, $N = 2$, $L = 2$, $Y = 64$;
- видеоредактирование: 320 Мбайт/с, $N = 5$, $L = 2$, $Y = 64$;
- поток 1 канала HDTV: 384 Мбайт/с, $N = 3$, $L = 10$, $Y = 128$;
- поток 2 каналов HDTV (в мощных рабочих станциях): 640 Мбайт/с, $N = 5$, $L = 10$, $Y = 128$.

Новые команды AGP для изохронных транзакций включают:

- *ISOCH Read* (0011) — изохронное чтение, в поле `LLL` — код длины: 000 — 32 байт, 001 — 64, 010 — 128, 011 — 256 байт;
- *ISOCH Write/Unfenced* (0110) — изохронная запись с беспорядочным завершением, в поле `LLL` — код длины: 000 — 32 байта, 001 — 64 байт;
- *ISOCH Write/Fenced* (0111) — изохронная запись с упорядочным завершением, в поле `LLL` — код длины: 000 — 32 байта, 001 — 64 байт;
- *ISOCH Align* (1110) — чтение временного сдвига относительно изохронного периода.

Новые коды состояния порта AGP включают:

- 100 — чтение изохронных данных;
- 101 — запись изохронных данных.

Конфигурационные регистры AGP

Конфигурирование устройств с интерфейсом AGP выполняется так же, как и обычных устройств PCI, — через обращения к регистрам конфигурационного пространства (см. главу 5). При этом карты AGP не требуют внешней линии `IDSEL` — у них внутренний сигнал разрешения доступа к конфигурационным регистрам соединен с линией `AD16`, так что обращение к конфигурационным регистрам AGP обеспечивается при `AD16` = 1.

В процессе инициализации процедура POST только распределяет системные ресурсы, но операции AGP оставляет запрещенными. Работу AGP разрешает загруженная ОС, предварительно установив требуемые параметры AGP: режим обмена, поддержку быстрой записи, адресации свыше 4 Гбайт, способ подачи и допустимое число запросов. Для этого параметры устройств считываются из регистра состояния AGP, а согласованные параметры записываются в регистр команд AGP, расположенный в конфигурационном пространстве. Параметры настройки порта задаются через конфигурационные регистры чипсета системной платы (главного моста).

В конфигурировании системы с AGP фигурируют две функции со своими конфигурационными пространствами:

- собственно *порт AGP* (Core Logic), являющийся целевым устройством в транзакциях AGP;
- *графический адаптер*, являющийся инициатором транзакций AGP.

Их специфические конфигурационные регистры (рис. 7.4) частично совпадают по назначению; серым цветом на рисунке помечены регистры, актуальные лишь для порта; звездочкой отмечены необязательные регистры.

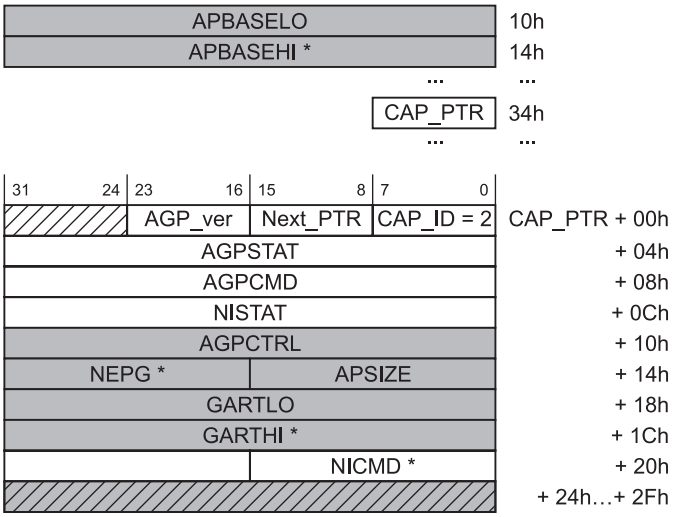


Рис. 7.4. Специфические конфигурационные регистры порта и карты AGP

Регистр `APBASELO` (только в порте) задает местоположение апертуры AGP:

- биты [31:22] задают адрес;
- биты [21:4] всегда нулевые (размер апертуры не может быть меньше 4 Мбайт);
- бит 3 = 1 — признак возможности предвыборки;
- биты [2:1] задают разрядность адреса: 00 — 32 бита, 10 — 64 бита (используется и `APBASEHI`).

Положение остальных регистров определяется значением `CAP_PTR`, который указывает адрес регистра `NCAPID`.

Регистр `NCAPID` (в порте и адаптере) содержит номер версии спецификации AGP:

- биты [31:24] — резерв;
- биты [23:20] — старшая цифра версии;
- биты [19:16] — младшая цифра;
- биты [15:8] — `NEXT_PTR`, указатель на следующий блок свойств (или ноль);
- биты [7:0] — `CAP_ID` = 02, идентификатор AGP;

Регистр состояния AGP — AGPSTAT (в порте и адаптере) сообщает основные возможности AGP: допустимое число запросов в очередях, поддержку внеполосной адресации, быстрой записи, адресации свыше 4 Гбайт, режимы 1x, 2x, 4x, 8x:

- биты [31:24] — RQ (только для порта), допустимое суммарное число запросов, находящихся в очередях: 0 — 1 команда, 255 — 256 команд;
- биты [23:18] — резерв (0);
- бит 17 — ISOCH SUPPORT — поддержка изохронных обменов (AGP 3.0);
- бит 16 — резерв;
- биты [15:13] — ARQSZ (только для порта), указание оптимального размера (*Opt_Rq_Size*) запроса к графическому адаптеру, *Opt_Rq_Size* = 2^(ARQSZ + 4). Введены в AGP 3.0;
- биты [12:10] — Cal_Cycle (только AGP 3.0), период калибровки: 000 — 4 мс, 001 — 16 мс, 010 — 64 мс, 011 — 256 мс, 111 — калибровка не требуется, остальные значения зарезервированы;
- бит 9 — SBA, поддержка внеполосной подачи команд (в AGP 3.0 — резерв);
- бит 8 — ITA_COH, обеспечение когерентности при обращении акселератора через апертуру (при единичном бите когерентности в соответствующем вхождении GART);
- бит 7 — GART64B, поддержка 64-битных элементов GART;
- бит 6 — htrans# (только AGP 3.0), трансляция запросов хоста через апертуру: 0 — при обращениях хоста в диапазоне апертуры адрес транслируется через GART, 1 — хост не посылает запросов в диапазоне апертуры;
- бит 5 — Over4G, поддержка адресации памяти свыше 4 Гбайт;
- бит 4 — FW, поддержка быстрой записи;
- бит 3 — AGP3.0_MODE, режим работы: 0 — AGP 1.0/2.0, 1 — AGP 3.0;
- биты [2:0] — RATE, поддерживаемые скорости обмена по AD и SBA. В режиме AGP 2.0: бит 0 — 1x, бит 1 — 2x, бит 2 — 4x. В режиме AGP 3.0: бит 0 — 4x, бит 1 — 8x.

Регистр команд AGP — AGPCMD (в порте и адаптере) служит для разрешения этих свойств и содержит следующие поля:

- биты [31:24] — RQ_DEPTH, задание глубины очереди команд;
- биты [23:16] — резерв (0);
- биты [15:13] — PARQSZ (только AGP 3.0), задание оптимального размера (*Opt_Rq_Size*) запроса, к которому должен стремиться мастер AGP, *Opt_Rq_Size* = = 2 ^ (PARQSZ + 4);
- биты [12:10] — PCAL_Cycle (только AGP 3.0), задание периода цикла калибровки;
- бит 9 — SBA_ENABLE, установка внеполосной подачи команд;
- бит 8 — AGP_ENABLE, разрешение операций AGP;
- бит 7 — GART64B, разрешение использования 64-битных элементов GART;
- бит 6 — резерв;

- бит 5 — 4G, разрешение адресации памяти свыше 4 Гбайт (двухадресных циклов и посылок 4-го типа по SBA);
- бит 4 — FW_Enable, разрешение быстрой записи;
- бит 3 — резерв (0);
- биты [2:0] — DATA_RATE, установка режима обмена (должен быть установлен лишь один бит). В AGP 2.0: бит 0 — 1x, бит 1 — 2x, бит 2 — 4x. В AGP 3.0: бит 0 — 4x, бит 1 — 8x, бит 3 — резерв.

Регистр NISTAT (в порте и адаптере) определяет возможности изохронных передач (только в AGP 3.0):

- биты [31:24] — резерв (0);
- биты [23:16] — MAXBW, максимальная пропускная способность устройства (суммарная для асинхронных и изохронных передач) в единицах по 32 байта за 1 мкс;
- биты [15:8] — ISOCH_N, максимальное число изохронных транзакций за период 1 мкс;
- биты [7:6] — ISOCH_Y, поддерживаемые размеры изохронных передач: 00 — 32, 64, 128, 256 байт; 01 — 64, 128, 256 байт; 10 —128, 256 и более байт, 11 — 256 байт;
- биты [5:3] — ISOCH_L, максимальная задержка изохронных передач в единицах мкс (1–5);
- бит 2 — резерв;
- биты [1:0] — Isoch-ErrorCode, код ошибки изохронного обмена (00 — нет ошибок): для порта 01 — переполнение очереди изохронных запросов; для карты 01 — «переопустошение» буфера чтения, 10 — переполнение буфера записи.

Регистр NICMD (в порте и адаптере) управляет изохронными передачами (только в AGP 3.0):

- биты [15:8] — PISOCH_N, максимальное число изохронных транзакций за период 1 мкс (для карты);
- биты [7:6] — PISOCH_Y, размер изохронных передач: 00 — 32, 64, 128, 256 байт; 01 — 64, 128, 256 байт; 10 —128, 256 и более байт, 11 — 256 байт;
- биты [5:0] — резерв.

Регистр AGPCTRL (в порте) управляет собственно портом AGP:

- биты [31:10] — резерв;
- бит 9 — CAL_CYCLE_DIS, запрет циклов калибровки;
- бит 8 — APERENB, разрешение работы через апертуру;
- бит 7 — GTLBEN, разрешение работы буферов TLB, ускоряющих трансляцию адресов через GART (если они имеются в порте);
- биты [6:0] — резерв.

Регистр APSIZE (в порте) задает размер апертуры:

- биты [15:12] — резерв;

- биты [11:8, 5:0] — APSIZE, 111...111 — 4 Мбайт, 111...110 — 8 Мбайт, 11110...0 — 256 Мбайт, 000...000 — 4096 Мбайт;
- биты [7:6] = 00.

Регистр NEPG (в порте AGP 3.0) задает размер страницы, описываемой в GART, из списка поддерживаемых:

- биты [15:12] — SEL, выбранный размер страниц (2 ^ (SEL + 12));
- биты [11:0] — битовая карта поддержки страниц разного размера: единица в бите *N* означает поддержку страниц размером (2 ^ (*N* + 12)). Бит 6 всегда единичный — поддержка страниц размером 4 кбайт обязательна для всех портов AGP.

Регистры GARTLO[31:12] и *GARTHI* (в порте) задают начальный адрес таблицы GART.

Слоты и карты AGP

Графический адаптер с интерфейсом AGP может быть встроен в системную плату, а может располагаться и на карте расширения, установленной в *слот AGP*. Внешне карты с портом AGP похожи на PCI (рис. 7.5), но у них используется разъем повышенной плотности с «двухэтажным» (как у EISA) расположением ламелей. Сам разъем находится дальше от задней кромки платы, чем разъем PCI.

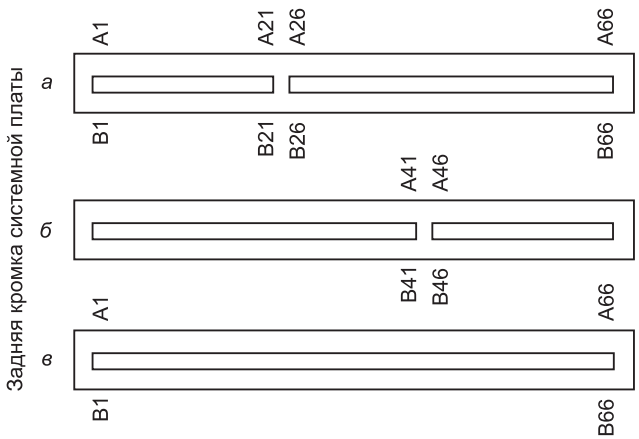


Рис. 7.5. Слоты AGP: *а* — 3,3 В; *б* — 1,5 В; *в* — универсальные

Порт AGP может использовать три возможных номинала питания интерфейсных схем (Vddq): 3,3 В (для 1x и 2x), 1,5 В (для 2x и 4x) и 0,8 В (для 8x). Сигналы RST# и CLK всегда 3-вольтовые. На слотах и картах имеются механические ключи, предотвращающие ошибочные подключения:

- слот и карта AGP 1.0 используют напряжение 3,3 В; они имеют ключи на месте контактов 22–25 (перегородка в слоте, рис. 7.5, *а*, вырез на разъеме карты);
- слот и карта AGP 2.0 используют напряжение 1,5 В, они имеют ключи на месте контактов 42–45;

- универсальный слот AGP 2.0 (3,3 В/1,5 В) не имеет перегородок, а универсальная карта имеет оба выреза. Универсальная системная плата узнает о номинале питания буферов установленной карты по сигналу TYPEDET# — на картах 3,3 В контакт свободен, на картах 1,5 В и универсальных — заземлен. Универсальная карта узнает о номинале питания буферов по уровню напряжения на контактах Vddq (3,3 или 1,5 В). Таким образом и обеспечивается согласование режима карты и системной платы;
- слот и карта AGP 3.0 используют напряжение 0,8 В, но по ключам они аналогичны 1,5-вольтовым слотам и картам (ключи на месте контактов 42–45). Карта узнает порт AGP 3.0 по заземленной линии MB_DET# (в порте AGP 2.0 он свободен);
- универсальный слот AGP 3.0 может работать с картой 8x (напряжение 0,8В) и AGP 2.0 (4x, 1,5 В). Здесь напряжение 0,8 В и режим 8x выбираются логикой порта и карты.

Для работы в режимах 2x/4x/8x приемникам требуется опорное напряжение Vref. Его номинал для 3,3 В составляет 0,4×Vddq, для 1,5 В — 0,5×Vddq, для 0,8 В — 0,233×Vddq. Опорное напряжение для приемников генерируется на стороне передатчиков. На контакт A66 (Vrefgc) графическое устройство подает сигнал для порта, на контакт B66 (Vrefcg) порт (чипсет) подает напряжение для устройства AGP.

При передаче в режиме 8x применяется *динамическое инвертирование данных* на шине AD. Сигнал DBI_LO указывает на инверсию линий AD[15:0], DBI_HI — на инверсию AD[31:0]. Решение об изменении состояния инверсии принимается сравнением выводимой информации с информацией предыдущего такта: если число переключаемых линий в соответствующей половине AD более 8, то соответствующий сигнал DBI_xx меняет состояние на противоположное. Таким образом, на каждой половине шины AD одновременно будет переключаться не более 8 сигнальных линий, что позволяет уменьшить броски тока. Для режима 8x применяется автоматическая *калибровка приемопередатчиков*, позволяющая согласовать их параметры с линией и партнером. Калибровка производится как статически (при начальном запуске), так и динамически в процессе работы, чтобы компенсировать уход параметров из-за изменения температуры.

В табл. 7.1 приведено назначение контактов слота AGP применительно к версии 3.0, в скобках приведены назначения контактов для AGP 1.0 и 2.0. Из-за двух ключей на универсальной карте AGP 2.0 теряется пара контактов для подачи питания VCC3.3, и их остается только 4, что ограничивает потребляемый ток (допустимый ток для каждого контакта — 1 А). На универсальной карте AGP 2.0 также нет дополнительного питания 3,3Vaux, используемого для питания цепей формирования сигнала PME# в режиме «сна».

Таблица 7.1. Назначение контактов порта AGP

Ряд B	№	Ряд A	Ряд B	№	Ряд A
OVRCNT#	1	12V	Vddq	34	Vddq
5.0V	2	TYPEDET#	AD21	35	AD22

продолжение ⇩

Таблица 7.1 (*продолжение*)

Ряд B	№	Ряд A	Ряд B	№	Ряд A
5.0V	3	Резерв	AD19	36	AD20
USB+	4	USB–	GND	37	GND
GND	5	GND	AD17	38	AD18
INTB#	6	INTA#	C/BE2#	39	AD16
CLK	7	RST#	Vddq	40	Vddq
REQ#	8	GNT#	IRDY#	41	FRAME#
VCC3.3	9	VCC3.3	Ключ 1,5 В (3,3Vaux)	42	Ключ 1,5 В (Резерв)
ST0	10	ST1	Ключ 1,5 В (GND)	43	Ключ 1,5 В (GND)
ST2	11	MB_DET#[3]	Ключ 1,5 В (Резерв)	44	Ключ 1,5 В (Резерв)
RBF#	12	DBI_HI (PIPE#)	Ключ 1,5 В (VCC3.3)	45	Ключ 1,5 В (VCC3.3)
GND	13	GND	DEVSEL#	46	TRDY#
DBI_LO [3]	14	WBF#	Vddq	47	STOP#
SBA0	15	SBA1	PERR#	48	PME#
VCC3.3	16	VCC3.3	GND	49	GND
SBA2	17	SBA3	SERR#	50	PAR
SB_STBF (SB_STB)	18	SB_STBS (SB_STB#[1])	C/BE1#	51	AD15
GND	19	GND	Vddq	52	Vddq
SBA4	20	SBA5	AD14	53	AD13
SBA6	21	SBA7	AD12	54	AD11
Резерв (ключ 3,3 В)	22	Резерв (ключ 3,3 В)	GND	55	GND
GND (ключ 3,3 В)	23	GND (ключ 3,3 В)	AD10	56	AD9
3,3Vaux (ключ 3,3 В)	24	Резерв (ключ 3,3 В)	AD8	57	C/BE0#
VCC3.3 (ключ 3,3 В)	25	VCC3.3 (ключ 3,3 В)	Vddq	58	Vddq
AD31	26	AD30	AD_STBF0 (AD_STB0)	59	AD_STBS0 (AD_STB0#[1])
AD29	27	AD28	AD7	60	AD6
VCC3.3	28	VCC3.3	GND	61	GND
AD27	29	AD26	AD5	62	AD4
AD25	30	AD24	AD3	63	AD2
GND	31	GND	Vddq	64	Vddq
AD_STBF1 (AD_STB1)	32	AD_STBS1 (AD_STB1#[1])	AD1	65	AD0
AD23	33	C/BE3#	Vrefcg[2]	66	Vrefgc[2]

[1] Инверсные стробы отсутствуют на картах и слотах 3,3 В (там нет режима 4x/8x).

[2] Опорное напряжение не требуется для слотов и карт 1x.

[3] Только в AGP 3.0.

Кроме собственно AGP в порте AGP заложены *сигналы шины USB*, которую предполагается заводить в монитор (линии USB+, USB– и сигнал OVRCNT#, которым сообщается о перегрузке по току линии питания + 5 В, выводимой в монитор).

Сигнал PME# относится к интерфейсу управления энергопотреблением (Power Management Interface). При наличии дополнительного питания 3,3Vaux этим сигналом карта может инициировать «пробуждение».

Спецификация AGP Pro описывает более мощный коннектор, позволяющий в 4 раза повысить мощность, подводимую к графической карте. При этом сохраняется односторонняя совместимость: карты AGP могут устанавливаться в слот AGP Pro, но не наоборот. В настоящее время от коннектора AGP Pro отказались, а для подачи питания на графическую карту используется дополнительный кабель с разъемом.

Коннектор AGP Pro имеет дополнительные контакты с обеих сторон обычного коннектора AGP (рис. 7.6) для линий GND и питания 3,3 и 12 В, назначение этих контактов приведено в табл. 7.2. Для правильной установки обычной карты со стороны задней кромки системной платы дополнительная часть слота AGP Pro закрывается съемной пластмассовой заглушкой. Карта AGP Pro может также использовать 1–2 соседних слота PCI: чисто механически (как точки опоры и место), как дополнительные коннекторы для подачи питания, как функциональные коннекторы PCI. Потребности в дополнительном питании и креплении взаимосвязаны: высокопроизводительные карты потребляют большую мощность, для отвода которой требуются мощные (и тяжелые) радиаторы и вентиляторы. К счастью, прогресс в технологии изготовления микросхем приводит к улучшению соотношения «мощность/производительность», так что задача питания и крепления графического адаптера несколько упростилась.

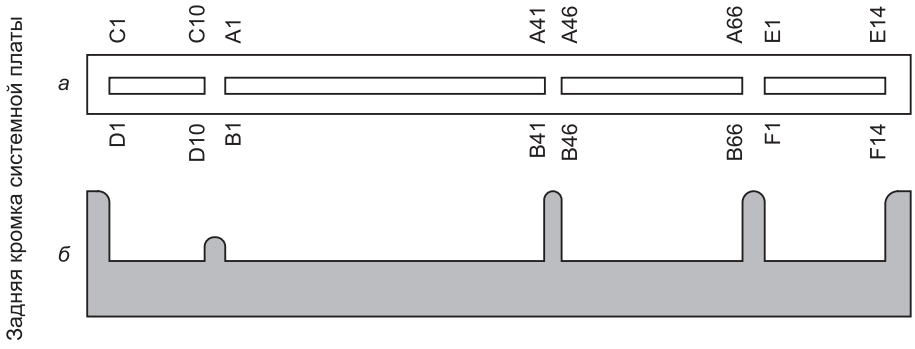


Рис. 7.6. Коннектор карты AGP Pro (показан ключ питания карты 1,5 В): *а* — вид сверху; *б* — профиль ключей

Таблица 7.2. Дополнительные контакты коннектора AGP Pro

Цепь	Контакты
VCC3.3	C1, C3, D1…D8
GND	C2, C4…C8, E3…E14

продолжение

Таблица 7.2 (*продолжение*)

Цепь	Контакты
VCC12	F3...F14
PRSNT1#	D10
PRSNT2#	D9
Резерв	C9, C10, E1, E2, F1, F2

В совокупности карта AGP Pro может потреблять до 110 Вт мощности, забирая ее по шинам питания 3,3 В (до 7,6 А) и 12 В (до 9,2 А) с основного разъема AGP, дополнительного разъема питания AGP Pro и одного-двух разъемов PCI. Карты AGP Pro большой мощности (*High Power*, 50–110 Вт) занимают 2 слота PCI, малой (*Low Power*, 25–50 Вт) — 1 слот. Соответственно скобка крепления к задней панели ПК у них имеет утроенную или удвоенную ширину. Кроме того, карты имеют крепеж к передней стенке ПК. На дополнительном разъеме цепь PRSNT1# служит признаком наличия карты (контакт заземлен), а PRSNT2# — признаком потребляемой мощности (до 50 Вт — контакт свободен, до 110 Вт — заземлен).

ГЛАВА 8

PCI Express

PCI Express — новая архитектура соединения компонентов, введенная под эгидой PCI SIG, известная и под названием *3GIO* (3-Generation Input-Output, ввод/вывод 3-го поколения). Здесь шинное соединение устройств с параллельным интерфейсом заменено на двухточечные последовательные соединения с использованием коммутаторов. В этой архитектуре сохраняются многие программные черты шины PCI, что обеспечивает плавность миграции от PCI к PCI Express. В архитектуре появились новые возможности: управление качеством обслуживания (QoS), потреблением и бюджетом связей. Протокол PCI Express отличается малыми накладными расходами и малыми задержками выполнения транзакций.

PCI Express позиционируется как универсальная архитектура ввода/вывода для компьютеров разных классов, телекоммуникационных устройств и встроенных систем. Высокая пропускная способность достигается при цене, соизмеримой с PCI и ниже. Сфера применения — от соединений между микросхемами на плате до межплатных разъемных и кабельных соединений. Высокая пропускная способность на контакт соединения позволяет минимизировать число соединительных контактов. Малое число сигнальных линий позволяет применять малогабаритные конструктивы. Универсальность дает возможность использования единой программной модели для всех форм-факторов. Спецификация PCI Express Base specification Revision 1.0a опубликована в апреле 2003 года.

Элементы и топология соединений PCI Express

Соединение PCI Express (PCI Express Link) — это пара встречных симплексных каналов, соединяющих два компонента. По этим каналам передаются *пакеты*, несущие команды и данные транзакций, сообщения и управляющие посылки. Канал может быть образован одной или несколькими линиями передачи сигналов (Lane); применение нескольких линий позволяет масштабировать пропускную способность канала. В PCI Express с помощью пакетного протокола реализуются все транзакции чтения и записи, используемые в PCI, причем в расщепленном варианте. Таким образом, здесь фигурируют *запросчик* транзакции (Requester) и *исполнитель* транзакции (Completer). В PCI Express рассматриваются *четыре простран-*

ства: памяти, ввода/вывода, конфигурационное и сообщений. Новое (по сравнению с PCI) *пространство сообщений* (Message Space) используется для передачи в виде пакетов «внеполосных» сигналов PCI: прерываний по линиям INTx, управления потреблением и т. п. Таким образом реализуются «виртуальные провода». *Порт PCI Express* содержит передатчик, приемник и узлы, необходимые для сборки-разборки пакетов.

Пример топологии средств ввода/вывода, иллюстрирующий архитектуру PCI Express, приведен на рис. 8.1. Центральным элементом архитектуры является *корневой комплекс* (Root Comlex), соединяющий иерархию ввода/вывода с центром — процессором (одним или несколькими) и памятью. Корневой комплекс может иметь один и более портов PCI Express, каждый из них определяет свой *домен иерархии* (hierarchy domain). Каждый домен состоит из одной *конечной точки* (Endpoint) или *суб-иерархии* — нескольких конечных точек, связанных коммутаторами. Возможность непосредственных равноранговых коммуникаций между элементами разных доменов обязательной не является, но может присутствовать в конкретных реализациях. Для обеспечения прозрачных равноранговых коммуникаций в корневом комплексе должны присутствовать коммутаторы. Возможность взаимодействия центрального процессора с любым устройством любого домена безусловна, как и возможность обращения любого устройства к памяти. Корневой комплекс должен генерировать запросы к конфигурационному пространству — его роль аналогична главному мосту PCI. Корневой комплекс может генерировать запросы ввода/вывода как запросчик; он может генерировать и блокированные (Locked) запросы, требующие непрерываемого исполнения. Корневой комплекс не должен поддерживать блокированные запросы как *исполнитель* (Completer) — это предотвращает «заклинивание» ввода/вывода.

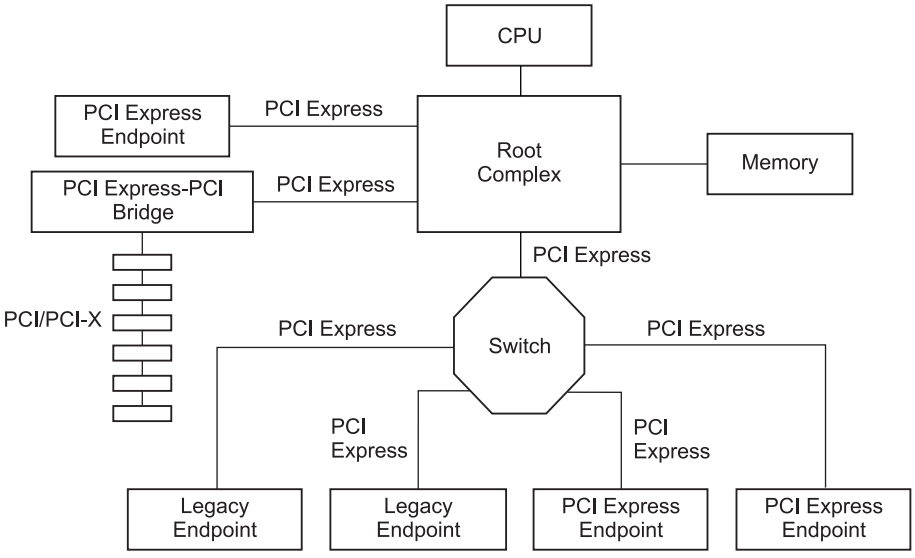


Рис. 8.1. Топология «фабрики» PCI Express

Конечная точка (Endpoint) — это устройство, способное инициировать и/или исполнять транзакции PCI Express от своего имени или от имени устройства не-PCI Express (например, хост контролера USB). Конечная точка должна быть видима в одном из доменов иерархии. Конечная точка должна иметь заголовок конфигурационного пространства типа 0 (см. главу 5) и отвечать как исполнитель на конфигурационные запросы. В качестве механизма сигнализации прерываний все конечные точки используют MSI. В PCI Express рассматриваются два типа конечных точек: «наследники» (Legacy) и новые точки, построенные по идеологии PCI Express. К «наследным» точкам имеется ряд послаблений:

- в плане адресации памяти они могут и не поддерживать более 4 Гбайт;
- ввод/вывод может не быть абсолютно перемещаемым (из пространства ввода/вывода в пространство памяти) с помощью регистров базового адреса (`BAR`), так что могут потребоваться транзакции обращения к пространству ввода/вывода (транзакции к памяти предпочтительнее);
- диапазон занимаемых адресов может быть менее 128 байт (требования к границам были жестко сформированы в PCI-X);
- конфигурационное пространство может не быть расширенным (оставаться в пределах 256 байт);
- программная модель может требовать использования блокированных запросов к устройству (но не от него).

Коммутатор (Switch) имеет несколько портов PCI Express. Логически он представляет собой несколько виртуальных мостов PCI-PCI, соединяющих порты коммутатора со своей внутренней локальной шиной. Виртуальный мост PCI описывается конфигурационными регистрами с заголовком типа 1 (см. главу 5). Порт, ведущий к вершине иерархии, называется *восходящим* (upstream port) — через него коммутатор конфигурируется как набор мостов PCI. Коммутатор транслирует между портами пакеты всех типов, основываясь на адресной информации, актуальной для пакета данного типа. Коммутатор не распространяет блокированные запросы со своих нисходящих портов. Арбитраж между портами коммутатора может учитывать виртуальные каналы и, соответственно, взвешенно распределять пропускную способность. Коммутатор не имеет права разбивать пакеты на более мелкие (аналог этого права имеется в мостах PCI).

Мост PCI-Express–PCI соединяет иерархию шин PCI/PCI-X с «фабрикой» ввода/вывода — корневым комплексом или коммутаторами PCI Express.

Конфигурирование «фабрики» осуществляется либо со 100% совместимостью с конфигурационным механизмом PCI 2.3, либо с использованием расширенного конфигурационного пространства PCI-X. Каждое *соединение PCI Express* с помощью виртуальных мостов отображается в виде *логической шины PCI* со своим номером. Логические устройства отображаются в конфигурационном пространстве как устройства PCI, каждое из которых может иметь 1–8 функций со своим набором конфигурационных регистров.

Архитектурная модель PCI Express

Архитектура PCI Express разделена на три уровня:

- *уровень транзакций* (Transaction Layer) — верний уровень архитектуры, отвечающий за сборку и разборку транзакционных *пакетов TLP* (Transaction Layer Packets). Эти пакеты используются для транзакций чтения и записи, а также сообщения о событиях некоторых типов. Каждый пакет TLP имеет уникальный идентификатор, который позволяет направить ответный пакет его отправителю. В TLP используются различные форматы адресации, зависящие от типов транзакций. Пакет может иметь атрибуты отмены слежения за когерентностью `NS` (No Snoop) и «расслабленной» упорядоченности `RO` (Relaxed Ordering). Каждая транзакция, требующая ответа, выполняется в виде расщепленной (см. PCI-X в главе 2). Уровень транзакций отвечает и за управление потоком, реализованное на основе механизма кредитов;
- *канальный уровень* (Data Link Layer), промежуточный в стеке, первым делом отвечает за управление связью, обнаружение ошибок и организации повторных передач, до успеха или признания отказа соединения. К пакетам, полученным от уровня транзакций, канальный уровень добавляет номера пакетов и контрольные коды. Канальный уровень и сам является генератором и получателем *пакетов DLLP* (Data Link Layer Packet), используемых для управления соединением;
- *физический уровень* изолирует канальный от всех подробностей передачи сигналов. Он состоит из двух субблоков. *Логический субблок* при передаче выполняет распределение данных по линиям, скремблирование, кодирование по схеме 8B/10B, кадрирование и преобразование в последовательный код. При приеме выполняются обратные действия. Дополнительные символы, обеспечиваемые кодированием 8B/10B, используются для служебной сигнализации. Логический субблок отвечает и за согласование соединения, инициализацию и т. п. *Электрический субблок* отвечает за электрическое согласование, синхронизацию, обнаружение приемника. Уровневая модель, принятая в PCI Express, позволяет, не затрагивая остальных уровней, заменить физический уровень или его субблоки, когда появятся более эффективные схемы кодирования и сигнализации. Интерфейс между физическим и канальным уровнем зависит от реализации этих компонентов и выбирается их разработчиком. Интерфейс физического уровня четко специфицирован, что обеспечивает возможность соединения устройств разного происхождения. Для тестирования на соответствие электрическим параметрам достаточно подключить устройство PCI Express к специальному тестеру.

Программная совместимость с PCI/PCI-X

Программная модель PCI Express совместима с PCI в следующих аспектах:

- обнаружение, нумерация и конфигурирование устройств PCI Express выполняются тем же конфигурационным ПО, что используется в PCI (PCI-X 2.0);

- существующие ОС загружаются без каких-либо модификаций;
- драйверы существующих устройств поддерживаются без каких-либо модификаций;
- конфигурирование и разрешение новых функциональных возможностей PCI Express выполняется по общей идее конфигурирования устройств PCI.

Качество обслуживания и виртуальные каналы

В PCI Express имеется поддержка дифференцированных классов по *качеству обслуживания* (QoS), обеспечивающая следующие возможности:

- выделять ресурсы соединения для потока каждого класса (виртуальные каналы);
- конфигурировать политику по QoS для каждого компонента;
- указывать QoS для каждого пакета;
- создавать изохронные соединения.

Для поддержки QoS применяется маркировка трафика: каждый пакет TLP имеет трехбитное поле *метки класса трафика* `TC` (Traffic Class). Это позволяет различать передаваемые данные по типам, создавать дифференцированные условия передачи трафика для разных классов. Порядок исполнения транзакций соблюдается в пределах одного класса, но не между разными классами. Для дифференцирования условий передачи трафика разных классов в коммутирующих элементах PCI Express могут создаваться виртуальные каналы. *Виртуальный канал VC* (Virtual Channel) представляет собой физически обособленные наборы буферов и средств маршрутизации пакетов, которые загружаются только обработкой трафика своего виртуального канала. На основе номеров виртуальных каналов и их приоритетов производится арбитраж при маршрутизации входящих пакетов. Каждый порт, поддерживающий виртуальные каналы, выполняет отображение пакетов определенных классов на соответствующие виртуальные каналы. При этом на один канал может отображаться произвольное число классов. По умолчанию весь трафик маркируется нулевым классом (TC0) и передается дежурным каналом (VC0). Виртуальные каналы вводятся по мере необходимости.

Сигнализация прерываний и управление энергопотреблением

Основной метод *сигнализации прерываний* в PCI Express — с помощью передачи сообщений (MSI), причем с 64-битной адресацией (32-битная разрешена только для «наследных» устройств). Однако ради обеспечения программной совместимости устройство может использовать и эмуляцию прерываний через `INTx#`, передавая эти запросы с помощью специальных пакетов. Получателем пакетов сигнализации прерываний, как MSI, так и эмуляции `INTx#`, как правило, является контроллер прерываний, расположенный в корневом комплексе. Сигнализация

INTx# производится пакетами класса TC0. Прерывания MSI при использовании виртуальных каналов должны использовать класс трафика, соответствующий классу трафика данных, к которым относятся данные прерывания. Иначе возможно нарушение синхронизации из-за относительной неупорядоченности трафика разных классов. Синхронизации можно добиваться и теми же средствами, что и в PCI/PCI-X — чтением (пусть даже нулевой длины) через коммутатор (мост). Такой прием неизбежен, если прерывания относятся к данным нескольких разных классов (виртуальных каналов).

Сигнализация событий управления энергопотреблением возможна в двух вариантах: пакетная эмуляция сигнала PME# (аналогично эмуляции INTx#) и естественная сигнализация PCI Express с помощью соответствующих сообщений. При эмуляции PME# идентификация источника сигнала выполняется последовательным чтением конфигурационных регистров устройств, способных генерировать этот сигнал. Естественная сигнализация гораздо удобнее: идентификатор устройства-источника присутствует в сообщении.

Расширенное управление потреблением и бюджетом мощности (PM — power management) означает:

- возможность идентификации способностей к PM каждой функции;
- возможность перевода функции в указанное состояние потребления;
- возможность получения информации о текущем состоянии потребления функции;
- возможность генерации запроса пробуждения при выключенном основном питании;
- возможность последовательного включения устройств.

«Горячее» подключение

«Горячее» подключение и замена устройств могут выполняться с использованием как существующих механизмов (PCI Hot-Plug и Hot-Swap, см. главу 6), так и естественных для PCI Express, не требующих дополнительных сигналов. Стандартная модель горячего подключения оперирует следующими элементами:

- индикатор питания слота, запрещающий извлечение/установку карты (мигание указывает на процесс перехода в обесточенное состояние);
- индикатор внимания, указывающий на проблемы, связанные с устройством в данном слоте (мигание индикатора используется для поиска нужного слота);
- ручной фиксатор карты;
- датчик состояния ручного фиксатора, позволяющий системному ПО обнаружить открытый замок;
- электромеханическая блокировка, не позволяющая извлекать карту при включенном питании. Специального сигнала для управления блокировкой не предусмотрено; если блокировка имеется, то она должна срабатывать прямо от питания порта;

- кнопка Внимание (Attention) для запроса операции «горячего» подключения;
- программный интерфейс пользователя, позволяющий запросить «горячее» подключение;
- система нумерации слотов, позволяющая визуально определить требуемый слот.

Надежность передачи и целостность данных

Для обеспечения *надежности транзакций* и *целостности данных* применяется CRC-контроль всех транзакций и управляющих пакетов. Запросчик считает транзакцию выполненной по получении подтверждающего сообщения от исполнителя (подтверждение отсутствует только для записей, отправленных в основную память). Обработка ошибок в минимальном варианте аналогична PCI, причем обнаруженные ошибки отображаются в конфигурационных регистрах функций (в регистре состояния). Расширенные возможности сообщения об ошибках дают исходную информацию для развитых процедур изоляции отказов и восстановления, а также мониторинга и регистрации (logging) ошибок. Ошибки делятся на три группы, что позволяет использовать адекватные процедуры восстановления:

- *исправимые* (correctable) ошибки — автоматически вызывающие аппаратную процедуру восстановления (повтора) и не требующие программного вмешательства для нормального исполнения транзакции;
- *неисправимые фатальные* (Fatal) ошибки — требующие для надежного возобновления работы выполнения сброса, в результате которого могут пострадать транзакции, не имеющие прямого отношения к ошибке;
- *неисправимые не фатальные* (Non-fatal) ошибки — не требующие сброса для возобновления работы. В результате этих ошибок могут быть потеряны лишь несколько транзакций, затронутых ошибкой.

Верхние уровни архитектуры PCI Express

Два верхних уровня — канальный и уровень транзакций — определяют основные функциональные возможности взаимодействия компонентов в системе.

Транзакции и форматы пакетов

Весь трафик в PCI Express оформлен в виде пакетов, из которых прикладной интерес представляют *пакеты уровня транзакций — TLP*. Каждый пакет TLP начинается с заголовка, за которым может следовать поле данных и, дополнительно, поле «дайджеста» (Digest) — 32-битного CRC-кода. Длина всего пакета перечисленных полей кратна двойному слову (DW, 32 бит). Заголовок пакета содержит следующие обязательные поля:

- поле формата `Fmt[1:0]`, задающее вид пакета: бит 0 — длина заголовка (0 —3 DW, 1 — 4 DW), бит 1 — присутствие поля данных (0 — нет данных);

- поле типа `Type[4:0]`, определяющее назначение пакета (тип транзакции, табл. 8.1);
- поле класса трафика `TC[2:0]`;
- признак «дайджеста» `TD`: единичное значение указывает на применение 32-битного CRC-кода в конце пакета, защищающего все поля пакета, не изменяемые в процессе его путешествия через коммутаторы PCI Express. Этот дополнительный контроль применяют для особо важных случаев; для обычных транзакций ограничиваются CRC-контролем канального уровня;
- признак ошибки данных `EP`, которым сообщается, что при чтении передаваемых данных произошла ошибка и данные могут быть недействительными (poisoned data);
- длина поля данных `Length[9:0]` (количество двойных слов, 000...01 — 1, 111...111 —1023 DW, 000...000 — 1024 DW.

Таблица 8.1. Пакеты транзакций PCI Express

Мнемоника	Fmt [1:0]	Type [4:0]	Назначение
MRd	00 01	0 0000	Memory Read Request, запрос чтения памяти
MRdLk	00 01	0 0001	Memory Read Request-Locked, запрос блокированного чтения памяти
MWr	10 11	0 0000	Memory Write Request, запрос записи в память (с данными)
IORd	00	0 0010	I/O Read Request, запрос чтения ввода/вывода
IOWr	10	0 0010	I/O Write Request, запрос записи ввода/вывода (с данными)
CfgRd0	00	0 0100	Configuration Read Type 0, конфигурационное чтение типа 0
CfgWr0	10	0 0100	Configuration Write Type 0, конфигурационная запись типа 0
CfgRd1	00	0 0101	Configuration Read Type 1, конфигурационное чтение типа 1
CfgWr1	10	0 0101	Configuration Write Type 1, конфигурационная запись типа 1
Msg	01	1 0 rrr	Message Request, запрос сообщения без данных, rrr определяет механизм маршрутизации
MsgD	11	1 0 rrr	Message Request with data payload, запрос сообщения с данными, rrr определяет механизм маршрутизации
Cpl	00	0 1010	Completion without Data, завершение без данных, используется в ответ на IOWr и CfgWr, а также при завершении любого блокированного чтения с ошибкой
CplD	10	0 1010	Completion with Data, завершение с данными, используется в ответ на запросы чтения
CplLk	00	0 1011	Completion for Locked Memory Read without Data, ошибочное завершение блокированного чтения памяти
CplDLk	10	0 1011	Completion for Locked Memory Read, завершение блокированного чтения памяти

Идентификатором транзакции является идентификатор устройства-запросчика в совокупности с 8-битным тегом; *дескриптор транзакции* содержит еще и атрибуты транзакции (RO и NS), а также класс трафика TC. Тег используется только для транзакций, требующих пакета завершения. По умолчанию запросчик может держать незавершенными до 32 транзакций, так что из 8-битного тега используются лишь 5 бит. Однако можно разрешить и расширенное использование тега, когда будут использоваться все 8 бит (до 256 незавершенных запросов).

В зависимости от типа транзакций в пакетах применяются различные форматы адреса и маршрутизации. Адрес задается с точностью до выровненного двойного слова (биты [1:0] = 00). Для всех транзакций, несущих данные (кроме сообщений), один из байтов заголовка несет битовые признаки действительности байтов в первом и последнем двойном слове поля данных (в середине подразумевается действительность всех байтов). Таким образом пакет может нести произвольное число смежных байтов, начиная с произвольного адреса.

Транзакции с памятью могут быть как с коротким (32-битным), так и с длинным (64-битным) адресом. Сочетания адреса и длины транзакции не должны вызывать пересечение границы 4-килобайтных страниц. Транзакции записи в память выполняются как *отправленные* (posted write), на них не требуется ответа.

Транзакции ввода/вывода сохранены в PCI Express для совместимости с PCI/PCI-X и старым ПО; от этого типа транзакций планируется избавиться. В этих транзакциях используется 32-битный адрес и передается лишь одно двойное слово данных.

Конфигурационные транзакции адресуются и маршрутизируются по идентификатору устройства; в этих транзакциях используется 32-битный адрес и передается лишь одно двойное слово данных. *Идентификатор устройства* имеет формат, используемый для PCI: 8-битное поле *номера шины*, 5-битное поле *номера устройства* и 3-битное поле *номера функции*.

Транзакции сообщений маршрутизируются в зависимости от поля rrr: 000 — к корневому комплексу, 001 — по адресу, 010 — по идентификатору, 011 — широковещание от корневого комплекса, 100 — локальное сообщение (не идет дальше приемника), 101 — собираются и маршрутизируются к корневому комплексу. В сообщении один байт отводится под код сообщения, и ряд сообщений обходится без поля данных. Сообщения с rrr = 100 только изменяют состояние приемника (так, например, реализуются нижеописанные виртуальные провода INTx#). Вариант маршрутизации с rrr = 101 используются для одного из типов сообщений управления потреблением: коммутатор пошлет такое сообщение на восходящий порт, только получив их со всех нисходящих портов. Сообщения используются для эмуляции проводных прерываний, оповещения о событиях управления энергопотреблением и ошибках, а также для коммуникаций между устройствами.

Для *эмуляции прерываний по* INTx# (четырех виртуальных проводов) используются 8 кодов сообщений (по четыре на установку и по четыре на снятие каждого сигнала). Коммутаторы (и корневой комплекс) должны отслеживать состояние виртуальных проводов на каждом из нисходящих портов, учитывая приходы сообщений от соответствующих устройств (циклическое чередование четырех

линий INTx#, как в PCI, сохраняется). В соответствии с этими состояниями по функции ИЛИ формируется состояние виртуальных проводов восходящего порта, и по изменению состояний генерируются соответствующие сообщения. Для эмуляции разделяемых прерываний сообщение на установку запроса передается на восходящий порт по приходу первого (для данного провода) сообщения установки от нисходящего порта. Если до прихода сообщения о снятии запроса для того же провода придет сообщение об установке запроса от другого порта, то это сообщение на восходящий порт не транслируется. Сообщение о снятии запроса будет послано на восходящий порт только по получении сообщений о снятии запросов для данного провода со всех нисходящих портов, ранее сообщивших об установке запроса. Корневой комплекс выполняет аналогичные задачи и, наконец, доводит виртуальные сигналы до реального контроллера прерываний. Таким образом, пакеты сообщений позволяют «объединить» виртуальные провода INTx# устройств всех логических шин.

Передача пакетов и пропускная способность соединения

Пакеты TLP, с помощью которых выполняются транзакции, поступают на канальный уровень. Основная *задача канального уровня* — обеспечить надежную доставку пакетов TLP. Для этого канальный уровень при передаче обрамляет пакет TLP своим *заголовком*, содержащим 12-битный последовательный номер TLP, и 32-битным полем LCRC (CRC канального уровня). Таким образом, канальный уровень к каждому пакету TLP добавляет 6 байт накладных расходов. На каждый пакет TPL передатчик должен получить *положительное подтверждение Ack* — пакет канального уровня *DLLP* (Data Link Layer Packet). Если подтверждение не приходит, то механизм тайм-аута заставляет передатчик повторить посылку пакета. Предусмотрен и пакет *отрицательного подтверждения Nak*, вызывающий повторную передачу без ожидания.

Пакеты DLLP имеют длину 6 байт: 4-байтная информационная часть и 16-битный CRC-код. Кроме подтверждений TLP пакеты DLLP используются для управления потоком, а также для управления энергопотреблением связи.

Физический уровень вводит свое обрамление передаваемых пакетов: перед началом пакета передается специальный символ *STP* (для TLP-пакета) или *SDP* (для DLLP-пакета), после пакета — символ *END*. Эти специальные символы отличаются от символов, представляющих данные после кодирования 8B/10B.

Рассмотрев структуры пакетов, можно оценить «скорострельность» базового соединения PCI Express (разрядность 1 бит, скорость 2,5 Гбит/с). Возьмем самую короткую транзакцию — запись двойного слова в пространство ввода/вывода. Соответствующий пакет TLP имеет длину 4 DW (3DW заголовок и 1 DW данных) — 16 байт; на канальном уровне к нему добавится 6 байт, так что на кодирование 8B/10B поступит 22 восьмибитных символа. К ним добавятся еще 2 символа обрамления физического уровня — итого 24. В линию будет передано 24×10 = 240 бит, что при скорости 2,5 Гбит/с займет 96 нс канала прямого направления. Вспомним, что

транзакция записи в порт требует подтверждения — встречного пакета TLP длиной 3DW, который после канального и физического обрамлений вырастет до 20 символов — 200 бит, занимающие 80 нс встречного канала. Теперь учтем канальные пакеты подтверждений приема каждого TPL (6-байтные Ack) — с физическим обрамлением по 8 символов (80 бит). Они в каждом канале займут по 32 нс. Итак, в прямом канале транзакций записи в порт занимает 96 + 32 = 128 нс (0,128 мкс), в обратном — 80 + 32 = 112 нс. Если подсчитать максимальную скорость передачи данных при непрерывных записях в порт, получаем V = 4/0,128 = = 31,25 Мбайт/с. При этом будет занят и встречный канал с коэффициентом загрузки 112/128 = 0,875. Результат по скорости близок к возможностям стандартной шины PCI (32 бит/33 МГц), в которой такая транзакция потребует четырех тактов шины. Чтение ввода/вывода на PCI Express даст те же результаты (на PCI результат будет хуже из-за «лишнего» такта на «пируэт»).

Теперь возьмем самый выгодный (в состязаниях по производительности) вариант транзакции: запись в память пакета 1024 двойных слов (с короткой 32-битной адресацией). Для этого требуется лишь один пакет TPL (транзакция завершения не требуется) длиной (3 + 1024) двойных слова = 4108 байт. К нему на канальном и физическом уровнях добавится 6 + 2 = 8 символов, что составит 4116×10 = = 41160 бит ≈ 16,5 мкс. При этом скорость передачи данных составляет 4096/16,5 ≈ ≈ 248 Мбайт/с — это уже уровень производительности PCI 32/66 МГц при длинных пакетных передачах. Загрузка встречного канала подтверждениями канального уровня в этом случае пренебрежимо мала. Скорость чтения из памяти будет немного ниже, поскольку каждая транзакция чтения состоит из двух пакетов TLP — запроса чтения (3 или 4 двойных слова) и пакета завершения с данными длиной (3 + N) двойных слова. При большой длине пакета относительная доля дополнительных трех DW будет невеликой.

Если встречный канал удается загрузить полезным трафиком, то можно говорить об удвоении пропускной способности PCI Express за счет возможности работы в полном дуплексе. Однако в примере с записью в порт ввода/вывода о таком удвоении речи быть не может, поскольку встречный канал загружен довольно плотно. Если пересчитать полезную скорость на один сигнальный контакт разъема, то в самом выгодном полнодуплексном варианте получим 248×2/4 = 124 Мбайт/с на контакт. Для сравнения можно взять вариант PCI-X533, обеспечивающий пиковую скорость записи, приближающуюся к 533×4 = 2132 Мбайт/с. В операциях чтения памяти PCI-X выглядит гораздо скромнее — пиковая скорость всего 533 Мбайт/с. При этом используется около 50 сигнальных контактов (не считая многочисленных земляных), так что на каждый контакт приходится примерно по 10–40 Мбайт/с. В порте AGP при той же пиковой скорости сигналов еще больше, так что заявления о высокой эффективности использования контактов в PCI Express имеют под собой основу. Полный дуплекс ни в PCI/PCI-X, ни в AGP невозможен.

Напомним, что данные подсчеты производились для базового соединения (x1, 1 линия); увеличив число линий до 32 бит (x32), можно получить максимальную скорость записи в память 248×32 = 7936 Мбайт/с. А если брать полную загрузку полнодуплексного соединения, то PCI Express может обеспечить суммарную пропускную

способность 15872 Мбайт/с. Таким образом, в самом мощном варианте PCI Express оставляет далеко позади порт AGP с его пиком 2132 Мбайт/с. Правда, говорить о малом числе контактов уже не приходится — канал PCI Express x32 требует 2×2×32 = = 128 сигнальных контактов (в AGP их меньше).

PCI Express и Advanced Switching

На основе физического и канального уровня PCI Express организацией ASI SIG (www.asi-sig.org) разработана *спецификация Advanced Switching* (AS), версия 1.0 опубликована в конце 2003 года. Ее цель — создание единого интерфейса для соединения между микросхемами, платами, модулями и даже блоками, удаленными друг от друга на небольшие расстояния. В данном параграфе сделаем краткое сравнение AS и PCI Express, различие между которыми находится на уровнях транзакций и вышестоящих уровнях.

Уровень транзакций PCI Express наследует основные черты PCI/PCI-X — шин подключения устройств, использующих для взаимодействие общедоступное пространство системной памяти (ОЗУ). Топология соединений — строго древовидная. При этом все устройства централизованно конфигурируются одним хостом (центральным процессором), расположенным в корне дерева. Все устройства работают под управлением одной операционной системы. Возможность непосредственного взаимодействия устройств друг с другом напрямую зависит от их взаимного расположения и не гарантируется, если путь между ними лежит через главный мост (безусловно доступна только системная память). Связь через прозрачные мосты основана на плоской модели памяти — во всей системе используется единое адресное пространство.

Уровень транзакций Advanced Switching отвечает распределенной модели вычислительной системы, в которой используются различные топологии и протоколы взаимодействия равноранговых устройств. Для обеспечения высокого уровня доступности применяется избыточность; большое внимание уделяется управлению качеством обслуживания (QoS).

Верхние уровни AS обеспечивают следующие особенности:

- поддержка мультихостовых систем и динамического конфигурирования. В систему с AS может объединяться множество компьютеров и устройств со своими локальными пространствами памяти (получается общая модель памяти системы — не плоская). Функции (обязанности) конфигурирования системы могут распределяться между несколькими хостами, каждый из которых работает под управлением своей ОС. Изменение состава функционирующих устройств не влечет за собой общей перезагрузки системы («горячее подключение» в PCI-подобных системах подразумевает заранее известный состав подключаемых устройств);
- безусловная возможность взаимодействия равноранговых устройств, независимо от их взаимного расположения. Прямая передача сообщений (не через память) обеспечивает высокую производительность обмена;

- поддержка различных топологий соединений, включая и варианты с множественными путями (звезда, двойная звезда, сетка). Элементы AS образуют *коммутационную фабрику*. Для определения пути пакетов по фабрике используется *маршрутизация от источника* (source routing) — источник сообщения в заголовке пакета явным образом перечисляет коммутаторы и их порты, через которые данный пакет должен доставляться до адресата назначения. В системе с множеством путей это обеспечивает быструю работу коммутаторов (им не приходится выбирать путь);
- управление качеством обслуживания и обеспечение защиты каналов и коммутаторов от перегрузок;
- масштабирование системы с AS — повышение производительности за счет развития «коммутационной фабрики» и распараллеливания потоков информации по разным путям;
- повышение надежности (уровня готовности) системы в целом за счет избыточности;
- единая основа для множества применений. Уровень транзакций AS позволяет инкапсулировать пакеты данных любых интерфейсов, включая PCI/PCI-X/PCI Express, сетевые и любые периферийные интерфейсы. Это позволяет использовать AS как средство подключения модулей ввода/вывода и сетевых интерфейсов в высокопроизводительных серверах и телекоммуникационных устройствах (коммутаторах, маршрутизаторах, фильтрах);
- программная прозрачность — поддержка общепринятых методов работы с устройствами PCI, включая нумерацию и конфигурирование;
- гибкое управление доставкой пакетов — поддержка группового вещания и широковещания (multicast и broadcast).

Физический уровень и конструктивы PCI Express

Физический уровень интерфейса допускает как электрическую, так и оптическую реализацию. *Базовое соединение электрического интерфейса* (1x) состоит из двух дифференциальных низковольтных сигнальных пар — передающей (сигналы PETp0, PETn0) и принимающей (PERp0, PERn0). В интерфейсе применена развязка передатчиков и приемников по постоянному току, что обеспечивает совместимость компонентов независимо от технологии изготовления компонентов и снимает некоторые проблемы передачи сигналов. Для передачи используется самосинхронизирующееся кодирование, что позволяет достигать высоких скоростей передачи. Базовая скорость — 2,5 Гбит/с «сырых» данных (после кодирования 8B/10B) в каждую сторону, в перспективе планируются и более высокие скорости. Для масштабирования пропускной способности возможно *агрегирование сигнальных линий* (lanes, сигнальных пар в электрическом интерфейсе), по одинаковому числу в обоих направлениях. Спецификация рассматривает варианты соединений из 1,

2, 4, 8, 12, 16 и 32 линий (обозначаются как x1, x2, x4, x8, x12, x16 и x32); передаваемые данные между ними распределяются побайтно. В каждой из линий самосинхронизация выплняется независимо, так что явление переноса (бич параллельных интерфейсов) отсутствует. Таким образом достижима скорость до 32×2,5 = 80 Гбит/с, что примерно соответствует пиковой скорости 8 Гбайт/с. Во время аппаратной инициализации в каждом соединении согласуется число линий и скорость передачи; согласование выполняется на низком уровне без какого-либо программного участия. Согласованные параметры соединения действуют на все время последующей работы.

Обеспечение «горячего» подключение на физическом уровне PCI Express не требует каких-либо дополнительных аппаратных затрат, поскольку двухточечное соединение не затрагивает «лишних» участников. Безопасная коммутация сигналов не требуется, возможности подключаемого устройства никак не влияют на режимы работы остальных устройств (для сравнения см. главу 6.).

Малое число сигнальных контактов интерфейса дает большую свободу в выборе *конструктивных реализаций* PCI Express:

- соединение компонентов в пределах платы;
- слоты и карты расширения в конструктивах PC/AT и ATX;
- внутренние и внешние карты расширения мобильных ПК;
- малогабаритные модули ввода/вывода для серверов и коммуникационной аппаратуры;
- модули для промышленных компьютеров;
- разъемное подключение «дочерних» карт (mezannine interface);
- кабельные соединения блоков.

Для карт расширения в конструктивах PC/AT и ATX предусматриваются разные модификации разъема-слота PCI Express, отличающиеся числом пар сигнальных линий (x1, x4, x8, x16) и, соответственно, размером (рис. 8.2). При этом в слоты большего размера можно устанавливать карты с разъемом того же размера или меньшего (это называется *Up-plugging*). Однако противоположный вариант (Down-plugging) — большую карту в меньший слот — механически невозможен (в PCI/PCI-X это возможно). Как было показано выше, самый маленький вариант PCI Express обеспечивает пропускную способность на уровне стандартной шины PCI. Назначение контактов слотов PCI Express приведено в табл. 8.2.

Набор сигналов интерфейса PCI Express невелик:

- PETp0, PETn0… PETp15, PETn15 — выходы передатчиков сигнальных пар 0...15;
- PERp0, PERn0… PERp15, PERn15 — входы приемников;
- REFCLK+ и REFCLK — сигналы опорной частоты 100 МГц;
- PERST# — сигнал сброса карты;
- WAKE# — сигнал «пробуждения» (от карты);
- PRSNT1#, PRSNT2# — сигналы обнаружения подключения-отключения карты для системы горячего подключения. На карте эти цепи соединяются между собой, причем для PRSNT2# выбирается контакт с самым большим номером. Это по-

зволяет точнее отслеживать моменты подключения-отключения (в случае наклона карты). Для определения числа линий подключенной карты данные линии не используются — разрядность линий определяется автоматически при установлении соединения (в процедуре тренировки).

Дополнительно на слоте имеются необязательные сигналы шины SMBus (SMB_CLK и SMB_DATA) и интерфейса JTAG (TCLK, TDI, TDO, TMS, TRST#).

Питание на карты подается по следующим шинам:

- +3,3V — основное питание +3 В при токе до 9 А;
- +12V — основное питание +12 В при токе до 0,5/2,1/4,4А для слотов x1/x4, x8/x16 соответственно;
- +3,3Vaux — дополнительное питание, ток до 375 мА в системах, способных к пробуждению по сигналу от карты и до 20 мА в непробуждаемых системах.

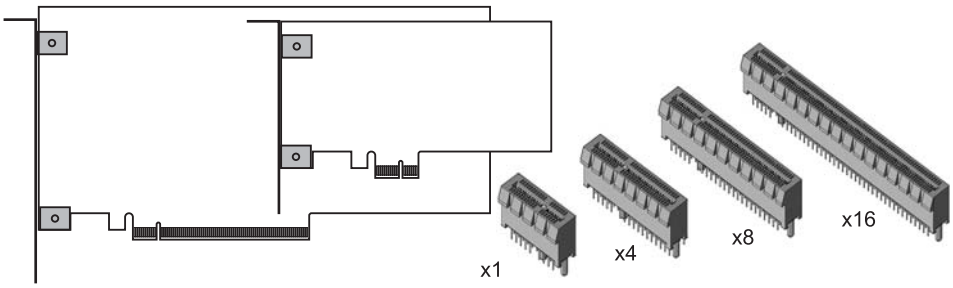


Рис. 8.2. Карты и разъемы PCI Express

Таблица 8.2. Разъемы PCI Express

№	Ряд B	Ряд A	№	Ряд B	Ряд A
1	+12V	PRSNT1#	31	PRSNT2#	GND
2	+12V	+12V	32	GND	Резерв
3	Резерв	+12V	Конец x4-коннектора		
4	GND	GND	33	PETp4	Резерв
5	SMB_CLK	TCK	34	PETn4	GND
6	SMB_DATA	TDI	35	GND	PERp4
7	GND	TDO	36	GND	PERn4
8	+3.3 V	TMS	37	PETp5	GND
9	TRST#	+3.3 V	38	PETn5	GND
10	+3.3 Vaux	+3.3 V	39	GND	PERp5
11	WAKE#	PERST#	40	GND	PERn5
Ключ			41	PETp6	GND
12	Резерв	GND	42	PETn6	GND
13	GND	REFCLK+	43	GND	PERp6
14	PETp0	REFCLK-	44	GND	PERn6

продолжение

Таблица 8.2 (*продолжение*)

№	Ряд B	Ряд A	№	Ряд B	Ряд A
15	PETn0	GND	45	PETp7	GND
16	GND	PERp0	46	PETn7	GND
17	PRSNT2#	PERn0	47	GND	PERp7
18	GND	GND	48	PRSNT2#	PERn7
Конец x1-коннектора			49	GND	GND
19	PETp1	Резерв	Конец x8-коннектора		
20	PETn1	GND	50	PETp8	Резерв
21	GND	PERp1	51	PETn8	GND
22	GND	PERn1	52	GND	PERp8
23	PETp2	GND	53	GND	PERn8
24	PETn2	GND	54	PETp9	GND
25	GND	PERp2		…	…
26	GND	PERn2	79	PETn15	GND
27	PETp2	GND	80	GND	PERp15
28	PETn2	GND	81	PRSNT2#	PERn15
29	GND	PERp3	82	GND	GND
30	Резерв	PERn3	Конец x16-коннектора		

Для мобильных компьютеров PCMCIA ввела конструктив *ExpressCard* (рис. 8.3), для которого на системный разъем выводится два интерфейса: PCI Express (1x) и USB 2.0. Модули ExpressCard компактнее прежних карт PCMCIA (PC Card и CardBus); предлагается две модификации, различающиеся по ширине: ExpressCard/34 (34×75×5 мм) и ExpressCard/54 (54×75×5 мм). Толщина модулей всего 5 мм, но, если требуется, то более длинные модули могут иметь утолщения в части, выходящие за габариты корпуса компьютера (за пределами 75 мм от края разъема). Как и прежние карты PCIMCIA, карты ExpressCard доступны пользователям и поддерживают «горячее» подключение.

Рис. 8.3. Карты ExpressCard

Для внутренних карт расширения блокнотных ПК введен конструктив *Mini PCI Express* (рис. 8.4), формат которого происходит от Mini PCI Type IIIA (см. главу 6). Благодаря уменьшению числа контактов ширина карты уменьшена до 30 мм, так что на месте одной карты Mini PCI можно разместить пару карт Mini PCI Express. На разъем карты (табл. 8.3) кроме PCI Express выведены интерфейсы последовательных шин USB 2.0 (USB_D+ и USB_D-) и SMBus (SMB_CLK и SMB_DATA), питание +3,3 В (750 мА основное и 250 мА дополнительное) и +1,5 В (375 мА). Собственно интерфейс PCI Express (x1) занимает всего 6 контактов (выходы передатчика PETp0 и PETn0, входы приемника PERp0 и PERn0, а также сигналы опорной частоты 100 МГц REFCLK+ и REFCLK-. Сигнал PERST# — сброс карты, сигнал WAKE# — «пробуждение» (от карты). Сигналы LED_Wxxx# служат для управления светодиодными индикаторами состояния.

Таблица 8.3. Разъемы Mini PCI Express

№	Цепь	№	Цепь
1	WAKE#	2	3.3 V
3	Резерв	4	GND
5	Резерв	6	1.5 V
7	Резерв	8	Резерв
9	GND	10	Резерв
11	REFCLK+	12	Резерв
13	REFCLK-	14	Резерв
15	GND	16	Резерв
Ключ			
17	Резерв	18	GND
19	Резерв	20	Резерв
21	GND	22	PERST#
23	PERn0	24	+3.3 V
25	PERp0	26	GND
27	GND	28	+1.5 V
29	GND	30	SMB_CLK
31	PETn0	32	SMB_DATA
33	PETp0	34	GND
35	GND	36	USB_D-
37	Резерв	38	USB_D+
39	Резерв	40	GND
41	Резерв	42	LED_WWAN#
43	Резерв	44	LED_WLAN#
45	Резерв	46	LED_WPAN#

продолжение

Таблица 8.3 (*продолжение*)

№	Цепь	№	Ценпь
47	Резерв	48	+1.5 V
49	Резерв	50	GND
51	Резерв	52	+3.3 V

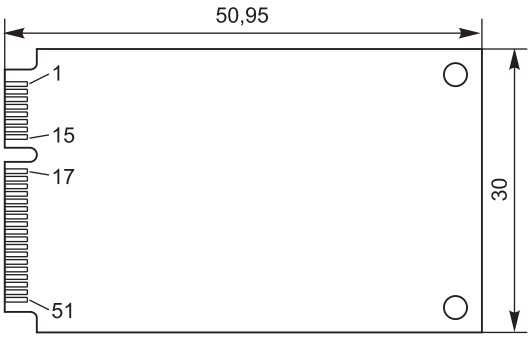


Рис. 8.4. Карты Mini PCI Express

С интерфейсом PCI Express удобно компонуются модули ввода/вывода и сетевых интерфейсов для серверов и коммуникационных устройств стоечного исполнения. Такие модули могут быть достаточно компактными (высота 2U не вызывает проблем размещения разъема), при этом производительности интерфейса достаточно даже для таких критичных модулей, как Fibre Channel, Gigabit Ethernet (GbE), 10GbE.

Интерфейс PCI Express принимается и для промышленных компьютеров, для чего имеются спецификации PICMG 3.4 (малогабаритные конструктивы для x1, x2 и x4), а также конструктивы в формате Compact PCI.

Интерфейс PCI Express существует и в *кабельном исполнении* для кабельных соединений блоков, находящихся на небольшом удалении друг от друга. Так, например, по PCI Express можно подключать док-станции к блокнотным ПК. Возможность вывода интерфейса системного уровня за пределы корпуса компьютера из предшественников PCI Express поддерживала только шина ISA, и то только при низких скоростях обмена (на частотах до 5 МГц). Из новых последовательных интерфейсов системного уровня эта возможность имеется и в InfiniBand. Наличие кабельного варианта высокопроизводительного интерфейса системного уровня может позволить отойти от традиционной компоновки компьютера, при которой в системном блоке концентрируются все компоненты, требующие интенсивного обмена с ядром компьютера.

ГЛАВА 9

Организация шины USB

USB (Universal Serial Bus — универсальная последовательная шина) является промышленным стандартом расширения архитектуры PC, ориентированным на интеграцию с телефонией и устройствами бытовой электроники. Версия стандарта 1.0 была опубликована в начале 1996 года, большинство устройств поддерживают стандарт 1.1, который вышел осенью 1998 года, — в нем были устранены обнаруженные проблемы первой редакции. Весной 2000 года опубликована спецификация USB 2.0, в которой предусмотрено 40-кратное увеличение пропускной способности шины. Первоначально (в версиях 1.0 и 1.1) шина обеспечивала две скорости передачи информации: *полная скорость*, *FS* (full speed) — 12 Мбит/с и *низкая скорость*, *LS* (low speed) — 1,5 Мбит/с. В версии 2.0 определена еще и *высокая скорость*, *HS* (high speed) — 480 Мбит/с, что позволяет существенно расширить круг устройств, подключаемых к шине. В одной и той же системе могут присутствовать и одновременно работать устройства со всеми тремя скоростями. Шина позволяет с использованием промежуточных хабов соединять устройства, удаленные от компьютера на расстояние до 25 м. Подробную и оперативную информацию по USB (на английском языке) можно найти на сайте http://www.usb.org. Разработку устройств и их классификацию и стандартизацию координирует *USB-IF* (USB Implementers Forum, Inc.).

Шина USB обеспечивает обмен данными между хост-компьютером и множеством периферийных устройств (ПУ). USB является единой централизованной аппаратно-программной системой массового обслуживания множества устройств и множества прикладных программных процессов. Связь программных процессов со всеми устройствами обеспечивает хост-контроллер с многоуровневой программной поддержкой. Этим USB существенно отличается от традиционных периферийных интерфейсов (портов LPT, COM, GAME, клавиатуры, мыши и т. п.), сравнение этих типов подключений приводится в табл. 9.1.

Таблица 9.1. Сравнение шины USB с традиционными периферийными интерфейсами

Традиционные интерфейсы (COM, LPT, Game...)	Шина USB
Подключение каждого устройства в общем случае требует присутствия собственного контроллера (адаптера)[1]	Все устройства подключены через один хост-контроллер

продолжение ⇩

Таблица 9.1 (*продолжение*)

Традиционные интерфейсы (COM, LPT, Game…)	Шина USB
Каждый контроллер занимает свои ресурсы (области в пространстве памяти, ввода/вывода, а также запросы прерывания)	Ресурсы занимает только хост-контроллер
Малое количество устройств, которые возможно одновременно подключить к компьютеру	Возможность подключения до 127 устройств
Драйверы устройств могут обращаться непосредственно к контроллерам своих устройств, независимо друг от друга	Драйверы устройств обращаются только к общему драйверу хост-контроллера
Независимость драйверов оборачивается непредсказуемостью результата одновременной работы с множеством устройств, отсутствием гарантий качества обслуживания (возможность задержек и уменьшения скорости передачи) для различных устройств	Централизованный планируемый обмен обеспечивает гарантии качества обслуживания, что позволяет передавать мультимедийные изохронные данные наряду с обычным асинхронным обменом
Разнообразие интерфейсов, разъемов и кабелей, специфичных для каждого типа устройств	Единый удобный и дешевый интерфейс для подключения устройств всех типов. Возможность выбора скорости работы устройства (1,5–15–480 Мбит/с) в зависимости от потребности
Отсутствие встроенных средств обнаружения подключения/отключения и идентификации устройств, сложность поддержки PnP	Возможность «горячего» подключения/отключения устройств, полная поддержка PnP, динамическое конфигурирование
Отсутствие средств контроля ошибок	Встроенные средства обеспечения надежной передачи данных
Отсутствие штатного питания устройств	Возможность питания устройств от шины, а также наличие средств управления энергопотреблением

[1] Возможностью подключения к одному контроллеру множества устройств обладает и шина SCSI, но ее параллельный интерфейс по сравнению с USB слишком дорог, громоздок и более ограничен в топологии.

Основные понятия

Архитектура USB допускает четыре базовых *типа передач* данных между хостом и периферийными устройствами:

- *изохронные передачи* (isochronous transfers) — потоковые передачи в реальном времени, занимающие предварительно согласованную часть пропускной способности шины с гарантированным временем задержки доставки. На полной скорости (FS) можно организовать один канал с полосой до 1,023 Мбайт/с (или два по 0,5 Мбайт/с), заняв 70 % доступной полосы (остаток можно занять и менее емкими каналами). На высокой скорости (HS) можно получить канал до

24 Мбайт/с (192 Мбит/с). Надежность доставки не гарантируется — в случае обнаружения ошибки изохронные данные не повторяются, недействительные пакеты игнорируются. Шина USB позволяет с помощью изохронных передач организовывать *синхронные соединения* между устройствами и прикладными программами. Изохронные передачи нужны для потоковых устройств: видеокамер, цифровых аудиоустройств (колонки USB, микрофон), устройств воспроизведения и записи аудио- и видеоданных (CD и DVD). Видеопоток (без компрессии) шина USB способна передавать только на высокой скорости;

- *прерывания* (interrupts) — передачи спонтанных сообщений, которые должны выполняться с задержкой не более, чем того требует устройство. Предел времени обслуживания устанавливается в диапазоне 10–255 мс для низкой и 1–255 мс для полной скорости. На высокой скорости можно заказать и 125 мкс. Доставка гарантирована, при случайных ошибках обмена выполняется повтор, правда, при этом время обслуживания увеличивается. Прерывания используются, например, при вводе символов с клавиатуры или для передачи сообщений о перемещениях мыши. Прерываниями можно передавать данные и к устройству (как только устройство сигнализирует о потребности в данных, хост своевременно их передает). Размер сообщения может составлять 0–8 байт для низкой скорости, 0–64 байт — для полной и 0–1024 байт — для высокой скорости передачи;
- *передачи массивов данных* (bulk data transfers) — это передачи без каких-либо обязательств по своевременности доставки и по скорости. Передачи массивов могут занимать всю полосу пропускания шины, свободную от передач других типов. Приоритет этих передач самый низкий, они могут приостанавливаться при большой загрузке шины. Доставка гарантированная — при случайной ошибке выполняется повтор. Передачи массивов уместны для обмена данными с принтерами, сканерами, устройствами хранения и т. п.;
- *управляющие передачи* (control transfers) используются для конфигурирования устройств во время их подключения и для управления устройствами в процессе работы. Протокол обеспечивает гарантированную доставку данных и подтверждение устройством успешности выполнения управляющей команды. Управляющая передача позволяет подать устройству команду (запрос, возможно, и с дополнительными данными) и получить на него ответ (подтверждение или отказ от выполнения запроса и, возможно, данные). Только управляющие передачи на USB обеспечивают синхронизацию запросов и ответов; в остальных типах передач явной синхронизации потока ввода с потоком вывода нет.

Аппаратная часть USB включает:

- *периферийные устройства USB*, несущие полезные *функции* (USB-functions);
- *хост-контроллер* (Host Controller), обеспечивающий связь шины с центром компьютера, объединенный с *корневым хабом* (Root Hub), обеспечивающим точки подключения устройств USB. Существует два варианта хост-контроллеров USB 1.x — UHC (Universal Host Controller) и OHC (Open Host Controller), поддерживающие скорости FS/LS; высокую скорость шины USB 2.0 (HS и только) поддерживает EHC (Enhanced Host Controller);

- *хабы USB* (USB Hubs), обеспечивающие дополнительные точки подключения устройств;
- *кабели USB*, соединяющие устройства с хабами.

Программная часть USB включает:

- *клиентское ПО* (CSw, Client Software) — драйверы устройств USB, обеспечивающие доступ к устройствам со стороны прикладного ПО. Эти драйверы взаимодействуют с устройствами только через программный интерфейс с общим драйвером USB (USBD). Непосредственного обращения к каким-либо регистрам аппаратных средств драйверы устройств USB не выполняют;
- *драйвер USB* (USBD, USB Driver), «заведующий» всеми USB-устройствами системы, их нумерацией, конфигурированием, предоставлением служб, распределением пропускной способности шины, мощности питания и т. п.;
- *драйвер хост-контроллера* (HCD, Host Controller Driver), преобразующий запросы ввода/вывода в структуры данных, размещенные в коммуникационной области оперативной памяти, и обращающийся к регистрам хост-контроллера. Хост-контроллер выполняет физические транзакции, руководствуясь этими структурами данных.

Драйверы USBD и HCD составляют хост-часть ПО USB; спецификация USB очерчивает круг их задач, но не описывает интерфейс между ними.

Физическое устройство USB должно иметь интерфейс USB, обеспечивающий полную поддержку протокола USB, выполнение стандартных операций (конфигурирование и сброс) и предоставление информации, описывающей устройство. Физические устройства USB могут быть *комбинированными* (compound devices): включать в себя несколько устройств-функций, подключенных к внутреннему хабу, а также предоставлять своим внутренним хабом дополнительные внешние точки подключения.

Работой всех устройств шины USB управляет *хост-контроллер* (host controller), являющийся программно-аппаратной подсистемой *хост-компьютера*. Хост-контроллер является интеллектуальным устройством шины PCI или составной частью «южного» хаба (моста) системной платы, интенсивно взаимодействующим с оперативной памятью.

Физическая топология шины USB — многоярусная звезда (рис. 9.1, *а*). Ее вершиной является хост-контроллер, объединенный с *корневым хабом* (root hub). Хаб является устройством-разветвителем, он может служить и источником питания для подключенных к нему устройств. К каждому порту хаба может непосредственно подключаться периферийное устройство или промежуточный хаб; шина допускает до пяти уровней (ярусов) каскадирования хабов (не считая корневого). Поскольку комбинированные устройства содержат внутри себя хаб, их подключение к хабу пятого яруса уже недопустимо. Каждый промежуточный хаб имеет несколько *нисходящих* (downstream) портов для подключения периферийных устройств (или нижележащих хабов) и один *восходящий* (upstream) порт для подключения к корневому хабу или нисходящему порту вышестоящего хаба.

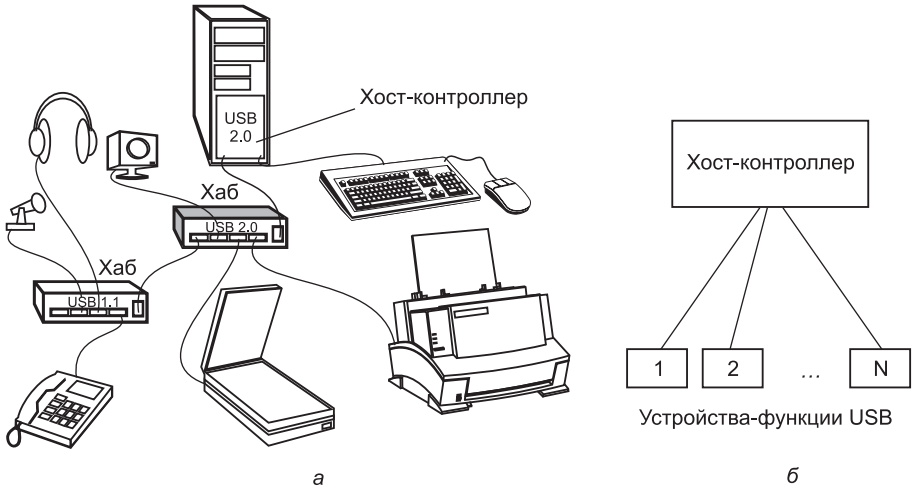
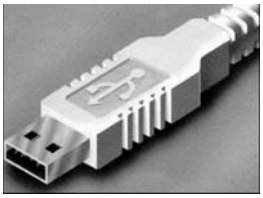
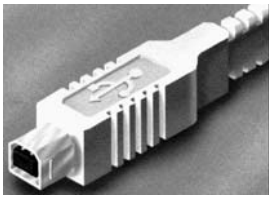


Рис. 9.1. Различные взгляды на отношения в USB: *а* — физическая топология; *б* — логическая топология

Логическая топология USB — звезда. Хабы (включая корневой) создают иллюзию непосредственного подключения каждого логического устройства к хост-контроллеру (рис. 9.1, *б*). В этой звезде устанавливаются сугубо подчиненные отношения по системе опроса-ответа: хост-контроллер по своей инициативе передает данные к выбранному устройству или принимает их. Устройство по своей инициативе передавать данные не может; непосредственные передачи данных между устройствами невозможны. Устройство по своей инициативе может лишь сигнализировать о «*пробуждении*» (wakeup), для чего используется специальная сигнализация, но не передача данных.

Физический интерфейс USB прост и изящен. Конструкция кабелей и коннекторов USB не дает возможности ошибиться при подключении устройств (рис. 9.2, *а* и *б*). Для распознавания разъема USB на корпусе устройства ставится стандартное символическое обозначение (рис. 9.2, *в*). Гнезда типа «A» устанавливаются только на нисходящих портах хабов, вилки типа «A» — на шнурах периферийных устройств или восходящих портов хабов. Гнезда и вилки типа «B» используются только для шнуров, отсоединяемых от периферийных устройств и восходящих портов хабов (от «мелких» устройств — мышей, клавиатур и т. п. кабели, как правило, не отсоединяются). Для малогабаритных устройств имеются разъемы mini-B, а для поддержки OTG (On-the-Go, см. главу 15) имеются и вилки mini-A, и розетки mini-AB. Хабы и устройства обеспечивают возможность «горячего» подключения и отключения с сигнализацией об этих событиях хосту.

При планировании соединений следует учитывать способ питания устройств: устройства, питающиеся от шины, как правило, подключают к хабам, питающимся от сети. К хабам, питающимся от шины, подключают лишь маломощные устройства — так, к клавиатуре USB, содержащей внутри себя хаб, подключают мышь USB и другие устройства-указатели (трекбол, планшет).


а *б* *в*

Рис. 9.2. Коннекторы USB: *а* — вилка типа «A»; *б* —вилка типа «B»; *в* — символическое обозначение

Логическое устройство USB представляет собой набор независимых *конечных точек* (Endpoint, *EP*), с которыми хост-контроллер (и клиентское ПО) обменивается информацией. Каждому логическому устройству USB (как функции, так и хабу) конфигурационная часть ПО хоста назначает 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). Точки *IN4* и *OUT4*, к примеру, представляют собой разные конечные точки, с которыми могут общаться даже модули клиентского ПО. Набор конечных точек зависит от устройства, но всякое устройство USB обязательно имеет *двунаправленную конечную точку 0* (*EP0*), через которую осуществляется его общее управление. Для прикладных целей используются конечные точки с номерами 1–15 (1–2 для низкоскоростных устройств). Адрес устройства, номер и направление конечной точки однозначно идентифицируют приемник или источник информации при обмене хост-контроллера с устройствами USB. Каждая конечная точка имеет набор характеристик, описывающих поддерживаемый тип передачи данных (изохронные данные, массивы, прерывания, управляющие передачи), размер пакета, требования к частоте обслуживания.

Устройство может выполнять несколько различных функциональных задач: например, привод CD-ROM может обеспечивать проигрывание аудиодисков и работать как устройство хранения данных. Для решения каждой задачи в устройстве определяется *интерфейс* — набор конечных точек, предназначенных для выполнения данной задачи, и правила их использования. Таким образом, каждое устройство должно обеспечивать один или несколько интерфейсов. Наличие нескольких интерфейсов позволяет нескольким драйверам, каждый из которых обращается только к своему интерфейсу (представляющему часть устройства USB), работать с одним и тем же устройством USB. Каждый интерфейс может иметь один или несколько *альтернативных вариантов* (альтернативных установок — alternate settings), из которых в данный момент активным может быть только один. Варианты различаются наборами (возможно, и характеристиками) используемых конечных точек.

Набор одновременно поддерживаемых интерфейсов составляет *конфигурацию устройства*. Устройство может иметь одну или несколько возможных конфигураций, из которых на этапе конфигурирования хост выбирает одну, делая ее *активной*. От выбранной конфигурации зависит доступная функциональность, и зачастую —

потребляемая мощность. Пока устройству не назначен номер выбранной конфигурации, оно не может функционировать в прикладном смысле и ток потребления от шины не должен превышать 100 мА. Хост выбирает конфигурацию исходя из доступности всех ресурсов, затребованных данной конфигурацией, включая и ток потребления от шины.

Модель передачи данных

Каждая единица клиентского ПО (обычно представляемая драйвером) связывается с одним интерфейсом своего устройства (функции) монопольно и независимо (рис. 9.3). Связи на этом рисунке обозначают *коммуникационные каналы* (communication pipes), которые устанавливаются между драйверами устройств и их конечными точками. Каналы устанавливаются только с конечными точками устройств, относящимися к выбранным (из альтернативных) вариантам интерфейсов активной конфигурации. Другие конечные точки недоступны.

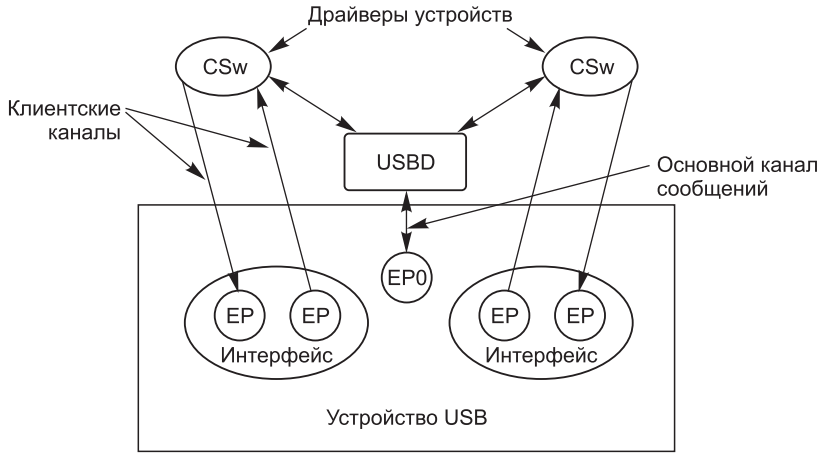


Рис. 9.3. Отношения клиентского ПО (CSw) с интерфейсами устройств USB

Запросы, пакеты и транзакции

Для передачи или приема данных клиентское ПО посылает к каналу *пакет запроса ввода/вывода — IRP* (Input/Output Request Packet) и ждет уведомления о завершении его отработки. Формат IRP определяется реализацией драйвера USBD в конкретной ОС. В IRP имеются только сведения о запросе (местоположение буфера передаваемых данных в оперативной памяти и длина передачи); от свойств конкретного текущего подключения (скорость, допустимый размер пакета) драйвер устройства абстрагируется. Отработкой запроса в виде транзакций на шине USB занимается драйвер USBD; при необходимости он разбивает на части длинные запросы (пакеты), пригодные для передачи за одну транзакцию. *Транзакция*

на шине USB — это последовательность обмена пакетами между хостом и ПУ, в ходе которой может быть передан или принят один *пакет данных* (возможны транзакции, в которых данные не передаются). Отработка запроса считается завершенной, когда успешно выполняются все связанные с ним транзакции. «Временные трудности», встречающиеся при их выполнении (неготовность к обмену данными), до сведения клиентского драйвера не доводятся — ему остается только ждать завершения обменов (или выхода по тайм-ауту). Однако устройство может сигнализировать о серьезных ошибках (ответом *STALL*), что приводит к аварийному завершению запроса, о чем уведомляется клиентский драйвер. В этом случае отбрасываются и все последующие запросы к данному каналу. Возобновление работы с данным каналом возможно лишь после явного уведомления об обработке ошибочной ситуации, которое драйвер устройства делает с помощью специального запроса (тоже вызова USBD).

Длинные запросы разбиваются на транзакции так, чтобы использовать максимальный размер пакета. Последний пакет с остатком может оказаться короче максимального размера. Хост-контроллер имеет средства обнаружения приема от устройства «неполновесного» пакета, размер которого меньше ожидаемого. В запросе IRP указывается, следует ли особым образом реагировать на это событие. Особая реакция может быть двоякой:

- считать короткий пакет разделителем, указывающим на конец блока данных. При этом данный IRP завершается нормально и исполняются следующие запросы к данному каналу;
- считать короткий пакет признаком ошибки, по которому канал останавливается (все его последующие ожидающие запросы сбрасываются).

При передаче массивов использование укороченных пакетов в качестве разделителей наиболее естественно. Таким образом, например, в одном из вариантов протоколов для устройств хранения данных укороченные пакеты известной длины используются в качестве управляющих.

Каналы

Коммуникационные каналы USB разделяются на два типа:

- *потоковый канал* (streaming pipe) доставляет данные от одного конца канала к другому, он всегда *однонаправленный*. Один и тот же номер конечной точки может использоваться для двух *разных* потоковых каналов — ввода и вывода. Передачи данных в *разных* потоковых каналах друг с другом *не синхронизированы*. Это означает, что запросы клиентских драйверов для *разных каналов*, поставленные в определенном порядке друг относительно друга, могут выполняться другом порядке. Запросы для *одного канала* будут исполняться строго *в порядке их поступления*; если во время исполнения какого-либо запроса происходит серьезная ошибка (об этом устройство сообщает ответом *STALL*), поток останавливается. Поток может реализовывать передачи массивов, изохронные и прерывания. Потоки несут данные произвольного формата, определенного разра-

ботчиком устройства (но не спецификацией USB). В потоках типично использование транзакций, в которых длина поля данных соответствует максимальному размеру, допустимому для его конечной точки. Если требуется разделение потока на логические блоки данных, то это можно сделать, применяя в качестве признака конца блока укороченные пакеты. Если оказывается, что блок укладывается в целое число пакетов максимального размера, в качестве разделителя можно использовать пакеты с нулевой длиной поля данных;

- *канал сообщений* (message pipe) является *двунаправленным*. Передачи сообщений во встречных направлениях *синхронизированы* друг с другом и строго *упорядочены*. На каждое сообщение противоположная сторона обязана ответить подтверждением его приема и отработки. Последующее сообщение не может быть послано до обработки предыдущего, но при отработке ошибок возможен сброс необслуженных сообщений. Форматы сообщений определяются спецификацией USB: имеется набор стандартных сообщений (запросов и ответов) и зарезервированных идентификаторов сообщений, формат которых определяется разработчиком устройства или интерфейса.

С каналами связаны характеристики, соответствующие конечной точке (полоса пропускания, тип сервиса, размер буфера и т. п.). Каналы организуются при конфигурировании устройств USB. Полоса пропускания шины делится между всеми установленными каналами. Выделенная полоса закрепляется за каналом, и если установление нового канала требует такой полосы, которая не вписывается в уже существующее распределение, запрос на выделение канала отвергается.

Каналы различаются и по назначению:

- *основной канал сообщений* (Default pipe, он же Control pipe 0), владельцем которого является USBD, используется для доступа к конфигурационной информации всех устройств. Этот канал устанавливается с *нулевой конечной точкой*, *EP0* (endpoint zero), которая у всех устройств всегда поддерживает только управляющие передачи;
- *клиентские каналы* (Client pipes), владельцами которых являются драйверы устройств. По этим каналам могут передаваться как потоки, так и сообщения; они поддерживают любые типы передач USB (изохронные, прерывания, массивы и управление).

Интерфейс устройства, с которым работает клиентский драйвер, представляет собой *связку клиентских каналов* (pipe's bundle). Для этих каналов драйверы устройств являются единственными источниками и потребителями передаваемых данных.

Владельцем основных каналов сообщений всех устройств является драйвер USB (USBD); по этим каналам передается информация конфигурирования, управления и состояния. Основным каналом сообщений может пользоваться и клиентский драйвер для текущего управления и чтения состояния устройства, но опосредованно через USBD. Например, сообщения, передаваемые по основному каналу, используются драйвером принтера USB для опроса текущего состояния (переда-

ются три признака в формате регистра состояния LPT-порта: *ошибка ввода/вывода*, *принтер выбран*, *отсутствие бумаги*).

Кадры и микрокадры

Хост организует обмены с устройствами согласно своему плану распределения ресурсов. Для этого хост-контроллер циклически с периодом 1 мс формирует *кадры* (frames), в которые укладываются все запланированные транзакции (рис. 9.4). Каждый кадр начинается с посылки пакета-маркера *SOF* (Start Of Frame), который является синхронизирующим сигналом для изохронных устройств, а также для хабов. Кадры нумеруются последовательно, в маркере *SOF* передаются 11 младших бит номера кадра. В режиме HS каждый кадр делится на 8 *микрокадров*, и пакеты *SOF* передаются в начале каждого микрокадра (с периодом 125 мкс). При этом во всех восьми микрокадрах *SOF* несет один и тот же номер кадра; новое значение номера кадра передается в нулевом микрокадре. В каждом микрокадре 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 начала интервала времени *EOF* (End of Frame). Период (частота) генерации микрокадров может немного варьироваться с помощью специального регистра хост-контроллера, что позволяет подстраивать частоту для изохронных передач (см. главы 11 и 15).

Рис. 9.4. Кадры шины USB

Кадрирование используется и для обеспечения живучести шины. В конце каждого микрокадра выделяется интервал времени *EOF* (End Of Frame), на время которого хабы запрещают передачу по направлению к контроллеру. Если хаб обнаружит, что с какого-то порта в это время ведется передача данных (к хосту), этот порт отключается, изолируя «болтливое» устройство, о чем информируется USBD.

Счетчик микрокадров в хост-контроллере используется как источник индекса при обращении к таблице дескрипторов кадров. Обычно драйвер USB составляет таблицу дескрипторов для 1024 последовательных кадров[1], к которой он обращается циклически. С помощью этих дескрипторов хост планирует загрузку кадров так, чтобы кроме запланированных изохронных транзакций и прерываний в них всегда находилось место для транзакций управления. Свободное время кадров может заполняться передачами массивов. Спецификация USB позволяет занимать под периодические транзакции (изохронные и прерывания) до 90% пропускной способности шины, то есть времени в каждом микрокадре.

[1] Подробнее планирование для UHC, OHC и EHC описано в главе 15.

ГЛАВА 10

Протокол шины USB

Протокол шины USB обеспечивает обмен данными между хостом и устройством. На протокольном уровне решаются такие задачи, как обеспечение достоверности и надежности передачи, управление потоком. Весь трафик на шине USB передается посредством *транзакций*, в каждой транзакции возможен обмен только между хостом и адресуемым устройством (его конечной точкой).

Транзакции и пакеты

Все транзакции (обмены) с устройствами USB состоят из двух-трех пакетов, типовые последовательности пакетов в транзакциях приведены на рис. 10.1. Каждая транзакция планируется и начинается по инициативе хост-контроллера, который посылает *пакет-маркер транзакции* (token packet). *Маркер транзакции* описывает тип и направление передачи, адрес выбранного устройства USB и номер конечной точки. Адресуемое маркером устройство распознает свой адрес и готовится к обмену. Источник данных, определенный маркером, передает *пакет данных*. На этом этапе транзакции, относящиеся к изохронным передачам, завершаются — здесь нет подтверждения приема пакетов. Для остальных типов передач работает механизм подтверждения, обеспечивающий гарантированную доставку данных. Форматы пакетов приведены на рис. 10.2, типы пакетов — в табл. 10.1. Во всех полях пакетов, кроме поля `CRC`, данные передаются младшим битом вперед (на временных диаграммах младший бит изображается слева). Пакет начинается с синхро-

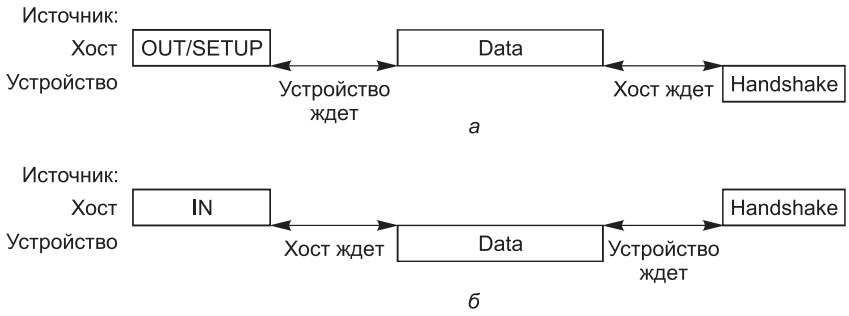


Рис. 10.1. Последовательность пакетов в транзакциях на шине USB:
а — вывод; *б* — ввод данных

последовательности Sync и завершается признаком конца — EOP. Тип пакета определяется полем PID. Назначение остальных полей раскрывается далее. Длина полей Sync и EOP указана для передач на FS/LS, для высокоскоростных передач поле Sync удлинено до 32 битовых интервалов, а EOP до 8 (в пакетах *SOF* поле EOP имеет длину 40 бит, см. главу 12).

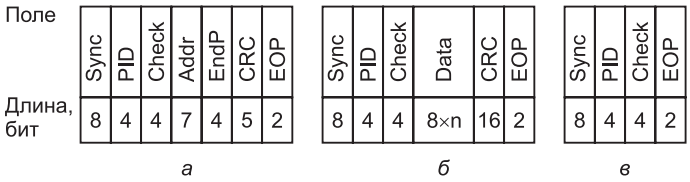


Рис. 10.2. Форматы пакетов USB: *а* — маркер; *б* — пакет данных; *в* — пакет квитирования

Таблица 10.1. Типы пакетов и их идентификаторы PID

Имя	Код PID	Содержимое и назначение
Пакеты-маркеры (Token)		
OUT	0001	Маркер транзакции вывода, несет идентификатор конечной точки (адрес устройства и номер точки; направление точки определяется кодом PID)
IN	1001	Маркер транзакции ввода, несет идентификатор конечной точки (адрес устройства и номер точки; направление точки определяется кодом PID)
SETUP	1101	Маркер транзакции управления, несет идентификатор конечной точки (адрес устройства и номер точки)
SOF	0101	Маркер начала микрокадра, несет 11-битный номер кадра (вместо полей Addr и EndP)
PING	0100	Пробный маркер управления потоком (в USB 2.0)
Пакеты данных		
DATA0 *DATA1*	0011 1011	Пакеты данных; чередование PID позволяет различать четные и нечетные пакеты для контроля правильности подтверждения
DATA2 *MDATA*	0111 1111	Дополнительные типы пакетов данных, используемые в транзакциях с широкополосными изохронными точками (в USB 2.0 для HS)
Пакеты квитирования (Handshake)		
ACK	0010	Подтверждение безошибочного приема пакета
NAK	1010	Индикация занятости (неготовности конечной точки к обмену данными, незавершенности обработки транзакции управления)
STALL	1110	Конечная точка требует вмешательства хоста
NYET	0110	Подтверждение безошибочного приема, но указание на отсутствие места для приема следующего пакета максимального размера (в USB 2.0)
Специальные пакеты (Special)		
PRE	1100	Преамбула (маркер) передачи на низкой скорости (разрешает трансляцию данных на низкоскоростной порт хаба)

Имя	Код PID	Содержимое и назначение
ERR	1100	Сигнализация ошибки в расщепленной транзакции (в USB 2.0)
SPLIT (*SS* и *CS*)	1000	Маркер расщепленной транзакции (в USB 2.0). В зависимости от назначения обозначается как SS (маркер запуска) и CS (маркер завершения), назначение определяется битом SC в теле маркера

Контроль и обработка ошибок передачи

Все принимаемые пакеты проверяются на наличие ошибок, что позволяют принятые форматы пакета и некоторые соглашения:

- пакет начинается с синхронизирующей последовательности, за которой следует его идентификатор `PID` (Packet Identificator). За идентификатором следует его инверсная копия — `Check`. Несовпадение двух копий считается признаком ошибки;
- тело пакета (все поля пакета, исключая `PID` и признак `EOP`) защищается CRC-кодом: 5-битным для пакетов-маркеров, 16-битным — для пакетов данных. Несовпадение CRC с ожидаемым значением считается признаком ошибки;
- пакет завершается специальным сигналом `EOP`; если в пакете оказывается не целое число байт, он считается ошибочным. Ложный `EOP`, даже на границе байта, не позволит принять пакет из-за практически неизбежной в данной ситуации ошибки по CRC-контролю;
- на физический уровень (в шину) данные пакета передаются с использованием вставки бит (bit stuffing, после шести единичных бит вставляется нолик), что предотвращает потерю битовой синхронизации при монотонном сигнале. Прием более шести единичных бит подряд считается ошибкой (на HS — признаком конца кадра).

Обнаружение любой из перечисленных ошибок в пакете заставляет приемник считать его недействительным. На пакеты, принятые с ошибкой, ни устройство, ни хост-контроллер никак не отвечают. При изохронной передаче данные недействительного пакета должны просто игнорироваться (они теряются); для остальных типов передач используются средства обеспечения надежной доставки.

Для обнаружения отсутствия ответа партнера на пакет каждое устройство имеет *счетчик тайм-аута*, который прерывает ожидание ответа по истечении некоторого времени. В USB имеется *ограничение на время оборота* по шине (roundtrip time): время от конца `EOP` сформированного пакета до получения начала ответного пакета. Для конечного устройства (и хост-контроллера) нормируется максимальная *задержка ответа* (response time) от конца увиденного `EOP` до введения им начала пакета. Для хабов нормируется задержка трансляции пакетов, для кабелей — задержка распространения сигналов. Счетчик тайм-аута должен учитывать максимальную задержку, возможную для допустимой конфигурации шины: до 5 промежуточных хабов, до 5 метров каждый кабель. Допустимое значение тайм-аута, выражаемое в битовых интервалах (bt), зависит от скорости:

- для скоростей FS/LS задержка, вводимая одним кабельным сегментом, по сравнению с битовым интервалом (*bt*) невелика. Исходя из этого в USB 1.0 для расчета допустимых задержек принимается следующая модель. На каждый кабельный сегмент отводится допустимая задержка 30 нс, на хаб — 40 нс. Таким образом, пять промежуточных хабов со своими кабелями вносят во время двойного оборота задержку 700 нс, что на FS соответствует примерно 8,5 bt. Для FS-устройства задержка ответа не должна превышать 6,5 bt (а с учетом его кабеля — 7,5 bt). Исходя из этого спецификация предписывает передатчикам на FS использовать счетчик тайм-аута на 16–18 bt;
- на скорости HS задержка в кабельном сегменте много больше битового интервала, и в USB 2.0 модель расчета несколько иная. Здесь на каждый кабельный сегмент отводится по 26 нс, а на хаб — по 4 нс плюс 36 bt. Таким образом, двукратное прохождение 6 кабельных сегментов (2×6×26 = 312 нс ≈ 150 bt) и пять хабов (2×5×4 = 40 нс ≈ 19 bt плюс 2×5×36 = 360 bt) занимает до 529 bt. Задержка ответа устройства допустима до 192 bt, а полная задержка с учетом кабелей и хабов будет до 721 bt. Исходя из этого спецификация предписывает передатчикам на HS использовать счетчик тайм-аута на 736–816 bt.

У хост-контроллера с каждой конечной точкой всех устройств связан свой счетчик ошибок, обнуляемый при планировании каждой транзакции. Этот счетчик считает все протокольные ошибки (включая и ошибки по тайм-ауту), и если число ошибок превышает порог (3), то канал с данной конечной точкой останавливается, о чем уведомляется его владелец (драйвер устройства или USBD). До превышения порога хост отрабатывает ошибки для неизохронных передач попытками повтора транзакций, без уведомления клиентского ПО. Изохронные передачи не повторяются, об обнаружении ошибок хост сообщает сразу.

Подтверждения, управление потоком и сигнализация ошибок устройства

Для подтверждений приема, управления потоком и сигнализации ошибок используются *пакеты квитирования* (handshake packets). Из этих пакетов *хост-контроллер* может посылать устройству только пакет *ACK*, подтверждающий безошибочный прием пакета данных. *Устройство* для ответа хосту использует следующие пакеты квитирования:

- *ACK* — подтверждение (положительное) успешного выполнения транзакции вывода или управления;
- *NAK* — отрицательное подтверждение, является признаком неготовности устройства к выполнению данной транзакции (нет данных для передачи хосту, отсутствует место в буфере для приема, не завершена операция управления). Это является нормальным ответом, о котором не узнает никто, кроме хост-контроллера, вынужденного повторить данную транзакцию позже. В транзакциях ввода ответ *NAK* устройство дает вместо пакета данных, если они не готовы;
- *STALL* — сообщение о серьезной ошибке, которое означает, что без специального программного вмешательства работа с данной конечной точкой становится

невозможной. Этот ответ доводится до сведения и драйвера USBD, отменяющего дальнейшие транзакции с этой точкой, и до клиентского драйвера, от которого и ожидается программное вмешательство, разблокирующее точку. В управляющих транзакциях (*Control*) ответ *STALL* означает невыполнимость данного запроса; разблокирования точки при этом не требуется.

Управление потоком при выводе данных, основанное только на возможности ответа *NAK* в случае неготовности устройства, весьма неэффективно расходует пропускную способность шины: чтобы убедиться в неготовности устройства, по шине впустую передается большой пакет данных. В USB 2.0 этой неприятности в транзакциях *Bulk-OUT* и *Control* избегают, применив *протокол проб* (*Ping Protocol*). Хост может опросить готовность устройства к приему пакета максимального размера, послав ему маркер-пробник *PING*. На этот маркер устройство может ответить подтверждением *ACK* (при готовности) или *NAK* (если не способно принять пакет максимального размера). Отрицательный ответ заставит хост повторить пробу позже, положительный разрешит ему выполнить транзакцию вывода данных. На транзакцию вывода после положительного ответа на пробу ответы устройства более разнообразны:

- *ACK* означает успешный прием и готовность принять следующий полноразмерный пакет;
- *NYET* означает успешный прием, но неготовность к следующему пакету;
- *NAK* — неожиданный ответ (он противоречит успеху пробы), но он возможен, если устройство внезапно стало временно не готово.

Высокоскоростное устройство в дескрипторах конечных точек сообщает о возможной интенсивности посылок *NAK*: поле `bInterval` для конечных точек типа *Bulk* и *Control* указывает число микрокадров, приходящееся на один *NAK* (0 означает, что устройство никогда не ответит *NAK'ом* на транзакцию вывода).

Обеспечение надежной доставки

Передачи массивов, прерываний и управления обеспечивают надежную доставку данных. После успешного приема пакета приемник данных посылает подтверждение — пакет квитирования *ACK*. Если приемник данных обнаружил ошибку, пакет игнорируется и никакого ответа на него не посылается. Источник данных считает, что очередной пакет передан успешно, когда получает от приемника подтверждение *ACK*. Если подтверждение не приходит, то в следующей транзакции источник повторяет посылку того же пакета. Однако пакет подтверждения может быть потерян из-за помехи; чтобы в этом случае повторная посылка пакета приемником не воспринималась как следующая порция данных, пакеты данных нумеруются. Нумерация ведется по модулю 2 (1-битный номер): пакеты делятся на четные (с идентификатором *DATA0*) и нечетные (*DATA1*). Для каждой конечной точки (кроме изохронных) у хоста и в устройстве имеются *биты-переключатели* (`Toggle Bit`), их начальные состояния тем или иным способом согласуются. В транзакциях *IN* и *OUT* передаются и ожидаются пакеты данных с идентификаторами *DATA0*

или *DATA1*, соответствующими текущему состоянию этих бит. Приемник данных переключает свой бит в случае безошибочного приема данных с ожидаемым идентификатором, источник данных — по приему подтверждения. Если приемник получает безошибочный пакет с неожидаемым идентификатором, он посылает подтверждение *ACK*, но данные пакета игнорирует, поскольку этот пакет — повторная посылка уже принятых данных.

Протоколы транзакций для различных типов передач

Транзакции для различных типов передач имеют протокольные различия, обусловленные гарантированием или не гарантированием пропускной способности, времени отклика, надежности доставки и синхронизированности ввода и вывода. В зависимости от этих характеристик в транзакциях используются те или иные из вышеописанных протокольных механизмов. Отметим, что обнаружение ошибок передачи работает во всех транзакциях, так что данные, принятые с ошибкой, всегда игнорируются. Какие именно протокольные механизмы используются в текущей транзакции, «знает» и хост-контроллер (по ранее полученному дескриптору конечной точки), и устройство USB, в котором эта конечная точка реализована.

Транзакции изохронных передач

Изохронные транзакции обеспечивают гарантированную скорость обмена, но не обеспечивают надежности доставки. По этой причине в протоколе отсутствуют подтверждения, поскольку повтор пакета приведет к сбою в планах доставки данных. Управление потоком, основанное на подтверждениях, отсутствует — устройство обязано выдерживать темп обмена, заявленный в дескрипторе изохронной конечной точки. Вопросы синхронизации источников и приемников изохронных данных подробнее рассмотрены в главе 11.

Транзакции изохронного вывода состоят из двух пакетов, посылаемых хост-контроллером, — маркера *OUT* и пакета данных *DATA*. В транзакции ввода хост посылает маркер *IN*, на который устройство отвечает пакетом данных, возможно, и с нулевой длиной поля данных (если нет готовых данных). Любой другой ответ устройства (как и «молчание») хостом расценивается как ошибка, приводящая к остановке данного канала.

При изохронном обмене имеется контроль достоверности (отбрасывание пакетов с ошибками) и целостности данных (обнаружение факта пропажи пакета). Контроль целостности основан на строгой детерминированности темпа обмена — в соответствии со своим дескриптором точка ожидает транзакцию с периодом $2^{bInterval-1}$ микрокадров. Для *обычной изохронной конечной точки* в микрокадре возможна лишь одна транзакция, и ошибка при приеме пакета выражается в отсутствии принятых данных в микрокадре, в котором они ожидаются. Таким образом, нумерация пакетов (переключатель `Toggle Bit`) не требуется. Полноскоростные устройства и хост-

контроллеры должны посылать пакеты только типа *DATA0*[1]. Для *широкополосных изохронных конечных точек* (USB 2.0) в каждом микрокадре возможна передача до трех пакетов данных. Любой из этих пакетов может потеряться, и для обнаружения этой ситуации требуется *нумерация пакетов внутри микрокадра*. Для этой нумерации введено два новых типа пакетов данных: *DATA2* и *MDATA*. Многообразие типов пакетов кроме нумерации позволяет еще и информировать партнера по связи о своих планах на данный микрокадр. В транзакциях *IN* идентификатором пакета устройство указывает, сколько еще пакетов оно собирается выдать в том же микрокадре, что позволяет хосту не делать лишних попыток ввода. Так, если в микрокадре передается один пакет, то это будет *DATA0*; если два — последовательность будет *DATA1*, *DATA0*; три — *DATA2*, *DATA1*, *DATA0*. В транзакциях *OUT* для вывода не последнего пакета в микрокадре используется пакет *MDATA* (More Data), а идентификатор последнего пакета показывает, сколько было до него передано пакетов. Так, при одной транзакции вывода используется пакет *DATA0*, при двух — последовательность *MDATA*, *DATA1*, при трех — *MDATA*, *MDATA*, *DATA2*. Во всех транзакциях, кроме последней в микрокадре, должны использоваться пакеты максимального размера. Отметим, что между широкополосными транзакциями в микрокадре могут вклиниваться другие транзакции.

Транзакции прерываний и передач массивов

Транзакции прерываний и передач массивов на шине выглядят одинаково. Здесь используются все механизмы управления потоком, надежной доставки и сигнализации, описанные выше. Инициализация переключателей (`Toggle Bit`) в устройстве и конечной точке контроля последовательности пакетов выполняется следующим образом:

- для конечных точек передачи массивов любое связанное с ней событие конфигурирования (установка конфигурации, интерфейса, обработка ошибки) устанавливает переключатели в положение *DATA0*. Подача нового запроса передачи (IRP) инициализации переключателей не вызывает;
- в транзакциях прерываний для обычных конечных точек используются только пакеты *DATA0*, поскольку для них в микрокадре возможна передача лишь одного пакета, а период опроса известен;
- в транзакциях прерываний для широкополосных конечных точек в микрокадре первым идет пакет данных *DATA0*, последующие пакеты в том же микрокадре идут с чередованием типов *DATA1-DATA0*.

Отметим, что по спецификациям USB транзакции прерываний могут использоваться как для ввода данных, так и для их вывода по запросу. Правда, случаи вывода по прерываниям рассматриваются не так подробно, как это имеет место для более привычного ввода. Ввод/вывод через транзакции *Interrupt* привлекателен возможностью получения и почти гарантированной скорости (не принимая в расчет ошибок передачи), и гарантированной доставки.

---

[1] Хотя, согласно спецификации, принимать должны как *DATA0*, так и *DATA1*.

Транзакции управляющих передач

Управляющие передачи предназначены для подачи *команды* (*Write Control*) или *запроса* (*Read Control*) устройству с индикацией результата выполнения. Передачи состоят из двух или трех стадий и выполняются с помощью нескольких транзакций:

- *стадия установки*, *Setup Stage*, предназначена для передачи управляющего сообщения от хоста к устройству. Это сообщение описывает команду (запрос), которую должно выполнить устройство. Команда может быть связана с передачей или приемом данных;
- *стадия передачи данных*, *Data Stage*, предназначена для посылки дополнительной управляющей информации (в передаче *Write Control*) или приема информации от устройства (в передаче *Read Control*). Эта стадия может отсутствовать, если не требуется ввод информации, а выводимая информация умещается в сообщении стадии установки;
- *стадия передачи состояния*, *Status Stage*, предназначена для уведомления хоста о факте и результате (успешно или нет) завершения исполнения команды.

Стадия установки выполняется в виде одной транзакции, начинающейся с маркера Setup. Далее хост посылает 8-байтный пакет данных (*DATA0*) с сообщением-запросом, имеющим стандартную структуру (см. главу 13). Устройство подтверждает успешный прием этого пакета ответом *ACK* и приступает к отработке команды-запроса. Отсутствие ответа заставит хост повторить запрос.

Стадия передачи данных (только для трехстадийных передач) выполняется хостом при помощи одной или нескольких транзакций *IN* (передача *Read Control*) или *OUT* (передача *Write Control*), обеспечивая управление потоком и надежную доставку с помощью повторов и чередования номеров. Первая транзакция фазы данных начинается с пакета *DATA1*.

Переход к *стадии передачи состояния* хост сигнализирует транзакцией, в которой направление передачи данных противоположно тому, что было на предыдущей стадии (*Setup* или *DATA*). Если стадии данных не было или выполнялась передача *Write Control*, хост выполняет транзакцию *IN*. Если выполнялась передача *Read Control*, хост выполняет транзакцию *OUT* (выводит пакет *DATA1* с нулевой длиной данных) или посылает маркер *PING* (в USB 2.0). На этой стадии интерес представляет только ответ ПУ:

- если устройство еще не завершило выполнение команды, оно будет отвечать пакетом *NAK*. Хост должен будет повторять эту транзакцию до получения иного ответа;
- если устройство успешно выполнило команду, оно на транзакцию *IN* ответит пакетом *DATA1* нулевой длины, а на транзакцию *OUT* (или маркер *PING*) ответит подтверждением *ACK*;
- если устройство завершило выполнение команды, но с ошибкой, то оно ответит пакетом *STALL*. Для управляющих точек данный ответ является обычным, не вызывает блокировки канала и не требует специальных действий для продолжения работы.

До получения ответа, указывающего на завершение выполнения команды, хост не имеет права обращаться к данной конечной точке с новой командой. Таким образом обеспечивается сериализация выполнения команд: устройство не «захлебнется» потоком команд, которые оно не в состоянии быстро исполнить; вводимые данные будут отвечать состоянию устройства на момент подачи команды-запроса, которой могли предшествовать управляющие записи. Такая упорядоченность (синхронизация) потоков ввода и вывода для устройств USB непосредственно не обеспечивается никакими другими типами передач. Управляющие передачи во всех устройствах USB поддерживают нулевые точки (*EP0*); поддержка управляющих передач для клиентских конечных точек бывает далеко не всегда.

ГЛАВА 11

Пропускная способность USB и изохронные передачи

Шина USB предназначена для подключения устройств с разными требованиями к пропускной способности интерфейса и задержкам доставки данных. Шина позволяет передавать асинхронный трафик одновременно с изохронным, обеспечивая возможность работы аудио- и видеоустройств. В данной главе рассматриваются вопросы пропускной способности шины и устройств для различных скоростей и типов передач, а также вопросы синхронизации при изохронных обменах.

Скорость обмена данными

Скорость последовательной передачи (1,5, 12 и 480 Мбит/с для LS, FS и HS соответственно) является только отправной точкой для определения реальной производительности обмена с конкретным устройством и всеми устройствами на шине в целом. Пропускная способность шины в целом определяется еще и соотношением накладных расходов и передаваемых полезных данных. Ниже рассматриваются источники накладных расходов, доля накладных расходов в общем трафике и загрузка шины транзакциями разных типов с разным размером блока данных. Для оценки возможной скорости обмена данными с конкретным устройством, подключенным к USB, отметим ряд моментов:

- *с обычной конечной точкой периодических передач* (изохронные и прерывания) в каждом *n*-м микрокадре производится лишь одна транзакция (*n* определяется параметром `bInterval` дескриптора конечной точки);
- *с широкополосной конечной точкой* в микрокадре может производиться до трех транзакций. В табл. 11.4 широкополосные точки представлены размером поля данных 1024–3072 байт, и загрузка шины, которую они дают, относится ко всем (от 1 до 3) их транзакциям в микрокадре. Пропускная способность *Vmax* точки с периодической передачей определяется делением размера поля данных паке-

та максимальной длины *Dmax* на длительность периода обслуживания *T*: *Vmax* = = *Dmax*/*T*. Период обслуживания *T* определяется следующим образом:

- для изохронных конечных точек $T = Tk \times 2^{\texttt{bInterval}-1}$, где *Tk* — период посылки маркеров `SOF` (1 мс для полной скорости и 125 мкс для высокой); `bInterval` лежит в диапазоне 1–16. Таким образом, для FS период обслуживания может быть в пределах 1–32768 мс, для HS — 0,125–4096 мс;
- для FS/LS конечных точек прерываний $T = 1 \times \texttt{bInterval}$ (мс), `bInterval` лежит в диапазоне 1–255 (период обслуживания может быть в пределах 1–255 мс);
- для HS-конечных точек прерываний $T = 0{,}125 \times 2^{\texttt{bInterval}-1}$ (мс); `bInterval` лежит в диапазоне 1–16, период обслуживания может быть в пределах 0,125–4096 мс.

- при передаче массивов число транзакций с конечной точкой в одном микрокадре не определено, но его максимум не превосходит указанного в таблицах. Драйвер USB может использовать и простую политику обслуживания очередей, при которой для каждой точки в микрокадре будет выполняться не более одной транзакции. В каждом микрокадре при самом плотном изохронном потоке есть место для 1–2 транзакций передач массивов, но когда на такие передачи претендует множество устройств, средняя скорость передачи для каждого из них, очевидно, будет невысокой.

Накладные расходы и загрузка шины

К накладным расходам при передаче по последовательной шине относятся:

- затраты на служебную информацию (пакеты маркеров и подтверждений, служебные поля пакетов данных);
- затраты на вставку бит: 6 последовательных единиц в любых полях кадра влекут за собой передачу по шине дополнительного вставленного бита. Доля этих накладных расходов может быть в пределах 0–15% от объема полезных данных. Из-за неопределенности этой доли данные затраты в нижеприведенных таблицах не учтены;
- задержки распространения сигналов в кабелях и хабах;
- внутренние задержки устройств при ответах на транзакции;
- затраты на повторы транзакций в случае ошибок приема и неготовности устройства.

Накладные расходы на каждую транзакцию зависят от ее типа; наиболее выгодные — изохронные (нет подтверждения), самые ресурсоемкие — управляющие трехстадийные. Число байт накладных расходов на каждую транзакцию, отнесенное к числу байт полезных данных для полной и высокой скорости, приведено в табл. 11.1 (низкую скорость в «соревнованиях» не рассматриваем). В таблице приведена и эффективность использования пропускной способности шины во время указанных транзакций. Большие накладные расходы на высокой скорости объясня-

ются большим влиянием задержек распространения: на полной скорости время оборота (см. главу 12) «поглощает» до двух байт, а на высокой — до 90 (поскольку битовый интервал около 2 нс много меньше допустимых задержек распространения).

Таблица 11.1. Накладные расходы на одну транзакцию и максимальная эффективность использования шины

Скорость	**FS**	**HS**
Тип	**Накладные расходы/Размер данных — Эффективность**	
Изохронные	9/1023 — 99%	38/1024 — 96%
Прерывания	13/64 — 83%	55/1024 — 95%
Передача массивов	13/64 — 83%	55/512 — 90%
Управление (3 стадии)	45/64 — 59%	173/64 — 27%

Очевидно, что с точки зрения уменьшения доли накладных расходов шины выгодно использовать транзакции с пакетами данных максимальной длины. Однако такие транзакции занимают слишком много времени в микрокадре, оставляя мало места для других. Теоретически за каждый кадр (1 мс) на полной скорости (12 Мбит/с) по шине может быть передано 12 000 бит (вместе со вставленными) — 1,5 Кбайт, хотя реально это число меньше из-за задержек распространения и ответов. На высокой скорости (480 Мбит/с) в микрокадре (125 мкс) передается 60 000 бит — 7,5 кбайт.

В табл. 11.2–11.4 приводятся параметры пропускной способности для разных типов передач в зависимости от размера поля данных. В этих таблицах приняты следующие обозначения: D — размер поля данных, V_{EP} — достижимая скорость для конечной точки, K_F — доля времени микрокадра, занимаемая транзакцией; V_{BUS} — максимальная пропускная способность шины с пакетами указанной длины.

Оценить возможность сочетания различных транзакций в микрокадре можно сложением занимаемых долей кадра (результат не должен превышать 100%). Из табл. 11.2, 11.3 и 11.4 видно, что низкоскоростные устройства при малой пропускной способности расходуют значительную часть времени шины. В USB 1.x с этим мирятся (ради простоты), а в USB 2.0 полоса высокоскоростной шины экономится за счет применения расщепленных транзакций (что требует существенного усложнения хабов).

Таблица 11.2. Пропускная способность транзакций на низкой скорости

D	**V_{EP}, Кбайт/с**	**K_F**	**N**	**V_{BUS}, Кбайт/с**	**V_{EP}, Кбайт/с**	**K_F**	**N**	**V_{BUS}, Кбайт/с**
	Управляющие передачи				**Прерывания**			
1	1	26%	3	3	1	11%	9	9
2	2	27%	3	6	2	11%	8	16

D	V_{EP}, Кбайт/с	K_F	N	V_{BUS}, Кбайт/с	V_{EP}, Кбайт/с	K_F	N	V_{BUS}, Кбайт/с
	Управляющие передачи				**Прерывания**			
4	4	28%	3	12	4	12%	8	32
8	8	30%	3	24	8	14%	6	48

Таблица 11.3. Пропускная способность транзакций на полной скорости

D	V_{EP}, Кбайт/с	K_F	N	V_{BUS}, Кбайт/с	V_{EP}, Кбайт/с	K_F	N	V_{BUS}, Кбайт/с
	Изохронные передачи				**Передача массивов и прерывания**			
1	1	1%	150	150	1	1%	107	107
2	2	1%	136	272	2	1%	100	200
4	4	1%	115	460	4	1%	88	352
8	8	1%	88	704	8	1%	71	568
16	16	2%	60	960	16	2%	51	816
32	32	3%	36	1152	32	3%	33	1056
64	64	5%	20	1280	64	5%	19	1216
128	128	9%	10	1280	Не дост.			
256	256	18%	5	1280				
512	512	35%	2	1024				
1023	1023	69%	1	1023				

Таблица 11.4. Пропускная способность транзакций на высокой скорости

D	V_{EP}, Кбайт/с	K_F	N	V_{BUS}, Кбайт/с	V_{EP}, Кбайт/с	K_F	N	V_{BUS}, Кбайт/с
	Изохронные передачи				**Передача массивов и прерывания**			
1	8	1%	192	1536	8	1%	133	1064
2	16	1%	187	2992	16	1%	131	2096
4	32	1%	178	5696	32	1%	127	4064
8	64	1%	163	10432	64	1%	119	7616
16	128	1%	138	17664	128	1%	105	13440
32	256	1%	107	27392	256	1%	86	22016
64	512	1%	73	37376	512	2%	63	32256
128	1024	2%	45	46080	1024	2%	40	40960
256	2048	4%	25	51200	2048	4%	24	49152
512	4096	7%	13	53248	4096	8%	13	53248

продолжение ⇩

Таблица 11.4 (*продолжение*)

D	V_{EP}, Кбайт/с	K_F	N	V_{BUS}, Кбайт/с	V_{EP}, Кбайт/с	K_F	N	V_{BUS}, Кбайт/с
	Изохронные передачи				**Прерывания**			
1024	8192	14%	7	57344	8192	14%	6	49152
2048[1]	16384	28%	3	49152	16384	28%	3	49152
3072[1]	24576	41%	2	49152	24576	42%	2	49152

[1] Для широкополосной конечной точки строки относятся к двум-трем транзакциям в микрокадре, в каждой из которых длина поля данных не превышает 1024 байт.

Совместная работа устройств с разными скоростями на одной шине

Спецификация USB позволяет к одной шине подключать устройства, работающие на существенно различающихся скоростях передачи. Для их нормального сосуществования в плане распределения времени микрокадров для каждой из скоростей приняты соответствующие ограничения на максимальную длину поля данных пакета:

- *низкая скорость* (LS, 1,5 Мбит/с) — до 8 байт, при этом двухстадийная транзакция управления занимает 30% кадра, а транзакция прерывания — 14%;
- *полная скорость* (FS, 12 Мбит/с) — до 1023 байт для изохронных обменов (69% кадра) и 64 байт для остальных типов (5% кадра);
- *высокая скорость* (HS, 480 Мбит/с) — до 1024 байт для прерываний и изохронных обменов (14% микрокадра), до 512 байт для передач массивов и управления (7–8% микрокадра).

Приемопередатчики (да и соединительные кабели) низкоскоростных устройств не способны работать с сигналами полной скорости, на которой передаются все маркеры *SOF* и пакеты обмена с полноскоростными устройствами. Поэтому 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*). Этот маркер 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 транслировать 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.

Эффективное сосуществование трех скоростей в USB 2.0 реализуется сложнее и обходится дороже. Во-первых, хост-контроллер USB 2.0 содержит фактически два контроллера — EHC, работающий только на высокой скорости, и контроллер-компаньон (возможно, и не один) USB 1.x (UHC или OHC) для полной и низкой скорости. Корневой хаб может иметь равноправные порты, но в процессе автоконфигурирования, в зависимости от свойств подключенного к нему устройства (или хаба), каждый порт соединяется с соответствующим контроллером. Существуют системные платы с фиксированным распределением портов по контроллерам: часть портов отводится под USB 2.0 и подключена к EHC, часть — под USB 1.1 и подключена к UHC или OHC. Ради повышения пропускной способности применяют и индивидуальные контроллеры (UHC или OHC) для каждого порта USB 1.x.

Во-вторых, хабы USB 2.0 имеют более сложную структуру: кроме повторителя он имеет еще и *транслятор транзакций*. Когда восходящий и нисходящие порты хаба работают на одинаковой скорости (FS или HS), хаб работает в режиме повторителя. При этом транзакция с устройством, подключенным к хабу, занимает весь канал от хост-контроллера до устройства на все время своего выполнения. Если же к порту хаба USB 2.0, работающего на HS, подключается устройство или хаб 1.1, то применяются *расщепленные транзакции*. Здесь по части канала от хоста до хаба (его транслятора транзакций) обмен проходит на скорости HS, а между транслятором транзакций и устройством (или хабом) USB 1.x обмен идет уже на его «родной» скорости FS или LS. Эти обмены разнесены во времени, между ними могут вклиниваться любые транзакции на высокой скорости (в том числе и расщепленные). Таким образом, расщепленные транзакции позволяют не расходовать попусту пропускную способность высокоскоростной шины: транзакции с хабом на высокой скорости занимают в 40 (для FS) и даже в 320 (для LS) раз меньше времени шины, чем транзакции с самим целевым устройством[1]. От старых (USB 1.x) устройств и хабов все тонкости расщепленных транзакций скрываются, чем и обеспечивается обратная совместимость.

Порт хаба имеет возможность аппаратно определить, какую скорость поддерживает подключенное устройство. Все HS-устройства по включению работают в ре-

[1] На самом деле отношения немного меньше из-за несколько больших накладных расходов как на HS в целом, так и на расщепление транзакций.

жиме FS, и только после взаимного согласования с портом хаба перейдут в режим HS. Если HS-устройство подключается к хабу USB 1.x, который этого согласования не поддерживает, устройство останется в режиме FS, возможно, с усеченной функциональностью. В системе с USB 2.0 у устройства можно спросить (запросом дескрипторов, см. главу 13), что изменится в его функциональности, если его подключить на другой скорости (изменив топологию соединений).

Вполне понятно, что устройство USB 2.0 сможет реализовать высокую скорость, 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. По возможности ее (конфигурацию) следует исправить, благо переключения кабелей USB можно выполнять на ходу. Устройства и ПО, критичные к полосе пропускания шины, в неправильной конфигурации работать откажутся и категорично потребуют переключений. Если же хост-контроллер старый, то все преимущества USB 2.0 окажутся недоступными пользователю. В этом случае придется менять хост-контроллер (менять системную плату или приобретать PCI-карту контроллера USB 2.0).

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 возможно повышение суммарной пропускной способности шины USB по сравнению с 12 Мбит/с во столько раз, сколько используется портов высокоскоростных хабов. Конечно, при этом суммарная пропускная способность для всех устройств, включая и HS-устройства, не может превышать общую пропускную способность HS-шины (нужно учитывать и накладные расходы). Кроме того, нужно учитывать архитектурные особенности хост-контроллера и хабов. Хост-контроллер может умножать пропускную способность FS/LS на число своих встроенных контроллеров USB 1.x. «Умножительные способности» хаба зависят от реализации его транслятора транзакций (см. главу 14).

Синхронизация при изохронной передаче

Изохронная передача данных связана с синхронизацией устройств, объединяемых в единую систему. 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 этих компонентов может иметь собственные «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 элементом, и его ис-

точник 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 *конверторы частот выборок* (SRC — sample rate converter), которые из *n* входных выборок делают *m* выходных, используя интерполяцию («сочиняя» промежуточные выборки). Эти конверторы позволят микшеру принимать данные от микрофона с его частотой (в нашем случае 8000 выборок/с) и отсылать на колонки с другой (44 100 выборок/с). Естественным решением задачи обеспечения взаимодействия этих компонентов было бы установление между ними *синхронного соединения*, обеспечивающего передачу как потока данных, так и сигнала синхронизации. Универсальная шина USB, обеспечивающая одновременное подключения множества устройств, синхронного интерфейса устройствам не предоставляет. *Синхронное соединение* на USB основано на *изохронных передачах*. При этом приходится иметь дело со следующими частотами:

- *Fs* (sample rate) — частота выборки для источников (source clock) и приемников (sink clock) данных;
- *Fb* (bus clock) — частота шины USB: частота кадров (1 кГц) для полной скорости и микрокадров (8 кГц) для высокой. С этой частотой все устройства USB «видят» маркеры начала микрокадров *SOF*;
- *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* (drift) — отклонения формально одинаковых частот от номиналов (не бывает двух абсолютно одинаковых генераторов);
- *дрожание* (jitter) — колебание частот относительно номинала;
- *фазовый сдвиг*, если сигналы не связаны системой фазовой автоподстройки ФАПЧ (PLL).

В цифровой системе передачи данных эти отклонения выливаются в то, что у источника или приемника может образовываться излишек или недостаток данных, колеблющийся или прогрессирующий 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 *типу синхронизации* источников или получателей данных с системой различают асинхронный, синхронный и адаптивный классы конечных точек, каждому из которых соответствует свой тип канала USB. Тип синхронизации задается битами [3:2] байта атрибутов (см. главу 13) в дескрипторе изохронной конечной точки:

- 00 — *нет синхронизации*;
- 01 — *асинхронная точка устройства*, не имеющего возможности согласования своей частоты выборок с метками *SOF* или иными частотами системы USB.

Частота передачи данных фиксированная или программируемая. Число байт данных, принимаемых за каждый микрокадр USB, не является постоянным. Синхронизация для источников и приемников различается:

- *асинхронный источник* данных *неявно объявляет свою скорость* передачи числом выборок, передаваемых им за один микрокадр, — клиентское ПО будет обрабатывать столько данных, сколько реально поступило. Примерами асинхронного устройства-источника может быть CD-плеер с синхронизацией от кварцевого генератора или приемник спутникового телевещания;
- *асинхронный приемник* данных должен обеспечивать *явную обратную связь* для адаптивного драйвера клиентского ПО, чтобы согласовать темп выдачи потока (см. ниже). Пример приемника — дешевые колонки, работающие от внутреннего источника синхронизации;

◆ 11 — *синхронная точка устройства*, имеющего внутренний генератор, синхронизируемый с маркерами микрокадров *SOF* (1 или 8 кГц). Источники и приемники за каждый микрокадр генерируют (потребляют) одинаковое количество байт данных, которое устанавливается на этапе программирования каналов. Примером синхронного источника может быть цифровой микрофон с частотой выборки, синтезируемой по маркерам *SOF*. Синтезатор частоты должен учитывать возможность пропадания одного-двух 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;

◆ 10 — *адаптивная точка устройства*, имеющего возможность подстройки своей внутренней частоты под требуемый поток данных (в разумных границах):

- *адаптивный источник* позволяет менять скорость под управлением приемника, обеспечивающего *явную обратную связь*. Примером адаптивного источника является CD-плеер со встроенным конвертором частоты;
- *адаптивный приемник* определяет мгновенное значение частоты по количеству данных, принятых за некоторый интервал усреднения. Таким образом осуществляется *неявное прямое объявление частоты*. Пример приемника — высококачественные колонки или наушники USB.

Обратная связь, позволяющая согласовать значения частот устройств с частотой шины, может быть *явной* (explicit feedback) или *неявной* (implicit feedback). Механизм обратной связи рассмотрим на примере асинхронного приемника; для адаптивного источника механизм работает аналогично. Асинхронный приемник должен явным образом сообщать хост-контроллеру требуемую частоту передачи данных относительно частоты микрокадров *Fs/Fb*. Здесь предполагается, что одна выборка представляется одним байтом данных, для иного размера выборки требуется соответствующий пересчет (для устройства и его клиентского ПО), чтобы *Fs/Fb* представляло число байт, передаваемых за один микрокадр. Отношение *Fs/Fb* может оказаться не целым числом, тогда его целая часть определяет постоянный объем передач (размер поля данных) с данной конечной точкой в каждом микрокадре, а дробная часть — это накапливающийся остаток, который будет вызывать

периодическое увеличение объема передач в некоторых микрокадрах. Отношение *Fs/Fb* в данном случае должен вычислять приемник на интервале усреднения не менее 1 с. Это отношение может меняться во времени (хотя бы из-за погрешности округления), так что хост должен периодически запрашивать у устройства отношение *Fs/Fb*, что и будет *данными явной обратной связи* (explicit feedback data).

Частота *Fs* задается с точностью до 1 Гц. Учитывая максимальный размер передачи в кадре (1023 байта) и частоту кадров (1 кГц), на полной скорости и для целой и для дробной части *Fs/Fb* достаточно по 10 бит. На высокой скорости в микрокадре может передаваться до 3072 байт — для целой части нужно 12 бит; частота микрокадров 8 кГц требует уже 13-битной дробной части. Исходя из этого, данные обратной связи представляются:

- на FS — 3-байтным числом, биты [23:14] — целая часть, биты [13:4] — дробная, остальные — резерв (нули);
- на HS — 4-байтным числом, биты [28:17] — целая часть, биты [16:4] — дробная, остальные — резерв (нули).

Заметим, что поток информации обратной связи всегда имеет направление, противоположное управляемому им потоку данных (на то эта связь и обратная). *Данные явной обратной связи* устройства берутся с конечной точки, имеющей такой же номер, что и у точки, используемой для основной передачи данных. Эта точка тоже изохронная, в ее дескрипторе установленные биты [5:4] байта атрибутов указывают на ее использование для обратной связи (у точки для передачи данных эти биты сброшены). В дескрипторе задается и интервал опроса (`bInterval`), с которым хост должен запрашивать данные обратной связи, чтобы своевременно отследить изменения. Это позволит хост-контроллеру постоянно корректировать число передаваемых байт за каждый микрокадр, 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 (микро)кадр генерировать ровно столько данных, сколько требуется хост-контроллеру. Здесь тоже поток данных и поток обратной связи имеют встречные направления, так что для явной обратной связи используется конечная точка с таким же номером, что и у источника данных.

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 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 (см. табл. 13.6) име-

ет значение бит [5:4] = 10. 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 *Synch Frame*, в котором указывает номер кадра (ожидаемого 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 OHC в дескрипторе изохронной передачи указывается номер стартового кадра; для UHC 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 UHC имеется регистр `SOF_Modify`, через который ПО может изменять коэффициент деления частоты 12 МГц для получения частоты кадров 1 кГц в пределах ±0,5%. Естественно, хост-контроллер может подстроиться под частоту внутренней синхронизации только одного устройства.

ГЛАВА 12

Физический интерфейс USB

Физический интерфейс USB определяет механические и электрические спецификации шины: кабели, разъемы, уровни сигналов, методы кодирования, питание от шины. Физический интерфейс обеспечивает возможность «горячего» подключения и отключения устройств с сигнализацией об этих событиях хосту.

Кабели и разъемы

Кабель USB содержит две пары проводов: одну для сигнальных цепей (`D+` и `D-`) и одну пару для схемной земли (`GND`) и подачи питания +5 В (`Vbus`). Допустимая длина сегмента (кабеля от устройства до хаба) — до 5 м. Ограничения на длину сегмента диктуются затуханием сигнала и вносимыми задержками. Задержка распространения сигнала по кабельному сегменту не должна превышать 26 нс, так что при большой погонной задержке допустимая длина кабеля может сократиться. Максимальное удаление устройства от хост-контроллера определяется задержкой, вносимой кабелями, промежуточными хабами и самими устройствами (см. главу 10).

В кабеле USB 1.x для сигнальных цепей используется витая пара проводов калибра 28AWG с импедансом 90 Ом. Характеристики кабеля нормированы в частотном диапазоне до 16 МГц. Для питания используется неперевитая пара проводов калибра 20AWG–28AWG. Требований к экранированию кабелей в USB 1.x не выдвигалось. Для низкой скорости может использоваться кабель с неперевитой парой сигнальных проводов (он дешевле и тоньше), но его длина не должна превышать 3 м.

В кабелях USB 2.0 используются провода тех же калибров, но в спецификации описана конструкция кабеля, в которую входит обязательный экран и связанный с ним дополнительный проводник. Такой кабель пригоден для работы на любых скоростях, включая и HS (480 Мбит/с).

Разъемы USB сконструированы с учетом легкости подключения и отключения устройств. Для обеспечения возможности «горячего» подключения разъемы обеспечивают более раннее соединение и позднее отсоединение питающих цепей по отношению к сигнальным. В USB определено несколько типов разъемов:

- *тип «A»*: гнезда (рис. 12.1, *а*) устанавливаются на нисходящих портах хабов, это стандартные порты подключения устройств. Вилки типа «A» устанавливаются на шнурах периферийных устройств или восходящих портов хабов;

- *тип «B»*: используются для шнуров, отсоединяемых от периферийных устройств и восходящих портов хабов (от «мелких» устройств — мышей, клавиатур и т. п. кабели, как правило, не отсоединяются). На устройстве устанавливается гнездо (рис. 12.1, *б*), на кабеле — вилка;
- *тип «Mini-B»* (рис. 12.1, *в*): используются для отсоединяемых шнуров малогабаритных устройств;
- *тип «Mini-A»*: введен в спецификации OTG (см. главу 15), вилки используются для подключения устройств к портам малогабаритных устройств с гнездом «*mini-AB*».
- *тип «Mini-AB»*: гнезда введены в спецификации OTG для портов двухролевых устройств, которые могут вести себя как хост (если в гнездо вставлена вилка *mini-A*) или как периферийное устройство (если в гнездо вставлена вилка *mini-B*).

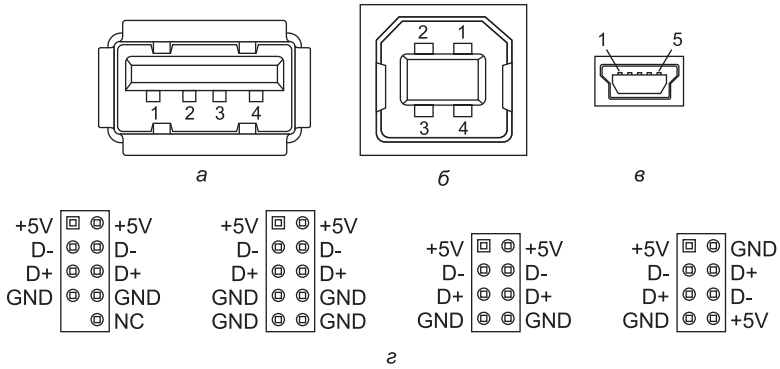


Рис. 12.1. Разъемы USB: *а* — гнездо «A»; *б* — гнездо «B»; *в* — гнездо «mini-B»; *г* — варианты разъема на системной плате

Назначение выводов разъемов USB приведено в табл. 12.1, нумерация контактов показана на рис. 12.1. Штырьковые разъемы, устанавливаемые на системной плате (рис. 12.1, *г*), предназначены для кабелей-«выкидышей», которыми подключаются дополнительные разъемы USB, устанавливаемые на передней или задней стенках корпуса компьютера (иногда и на боковых). На эти разъемы порты выводятся парами, причем у разных производителей подход к универсальности и защите от ошибочных подключений различен. Подключение «выкидыша», не подходящего к разъему, приводит к неработоспособности порта (к счастью, как правило, временной). Ошибка в подключении цепей `GND` и `+5V` может приводить к нагреванию кабелей и разъемов из-за короткого замыкания питающей цепи.

Все кабели USB «прямые» — в них соединяются одноименные цепи разъемов, кроме цепи `ID`, используемой для идентификации роли устройства в OTG. На вилке *mini-A* контакт 4 (`ID`) соединен с контактом 5 (`GND`), что заставляет порт, к которому подсоединена такая вилка, взять на себя роль нисходящего порта хаба. На вилке *mini-B* такого соединения нет.

Ошибка в полярности подводимого питания может повредить подключаемое устройство (и необратимо). По этой причине наиболее безопасными для подключае-

мого устройства являются внешние разъемы USB, запаянные на системной плате или карте контроллера USB.

Таблица 12.1. Назначение выводов разъема USB

Цепь	Контакт стандартного разъема	Контакт мини-разъема
VBus (+5 В)	1	1
D–	2	2
D+	3	3
GND	4	5
ID	–	4

Приемопередатчики

Для передачи сигналов используются два провода D+ и D–. На каждой стороне интерфейса (порте хаба и подключенного устройства, рис. 12.2) имеются:

- дифференциальный приемник, выход которого используется при приеме данных;
- управляемый (отключаемый) дифференциальный FS/LS-передатчик — источник напряжения, позволяющий кроме дифференциального сигнала формировать и «линейный 0» (*SE0*), а также отключаться для обеспечения полудуплексного обмена;
- линейные приемники, сообщающие текущее состояние каждого сигнального провода;
- резисторы, подтягивающие уровни сигналов для обнаружения подключения устройства:
  - Rd1, Rd2 (15 кОм) у хаба;
  - Ruf (у FS/HS-устройства) или Rul (у LS-устройства);
- Дополнительные элементы для работы на высокой скорости (только для устройств HS):
  - коммутатор, отключающий резистор Ruf при выборе высокой скорости;
  - последовательные резисторы Rz1 и Rz2 на выходах дифференциального передатчика, обеспечивающие согласование с линией и нагрузку;
  - управляемый дифференциальный источник тока;
  - детектор амплитуды сигнала;
  - детектор отключения (только на нисходящих портах хабов).

Уровни сигналов передатчиков FS/LS в статическом режиме должны быть ниже 0,3 В (низкий уровень) или выше 2,8 В (высокий уровень). Приемники должны выдерживать входное напряжение в пределах –0,5...+3,8 В. Чувствительность дифференциальных приемников — 200 мВ при синфазном напряжении 0,8–2,5 В. Линейные приемники должны обладать гистерезисом с нижним порогом 0,8 В и верхним порогом 2 В.

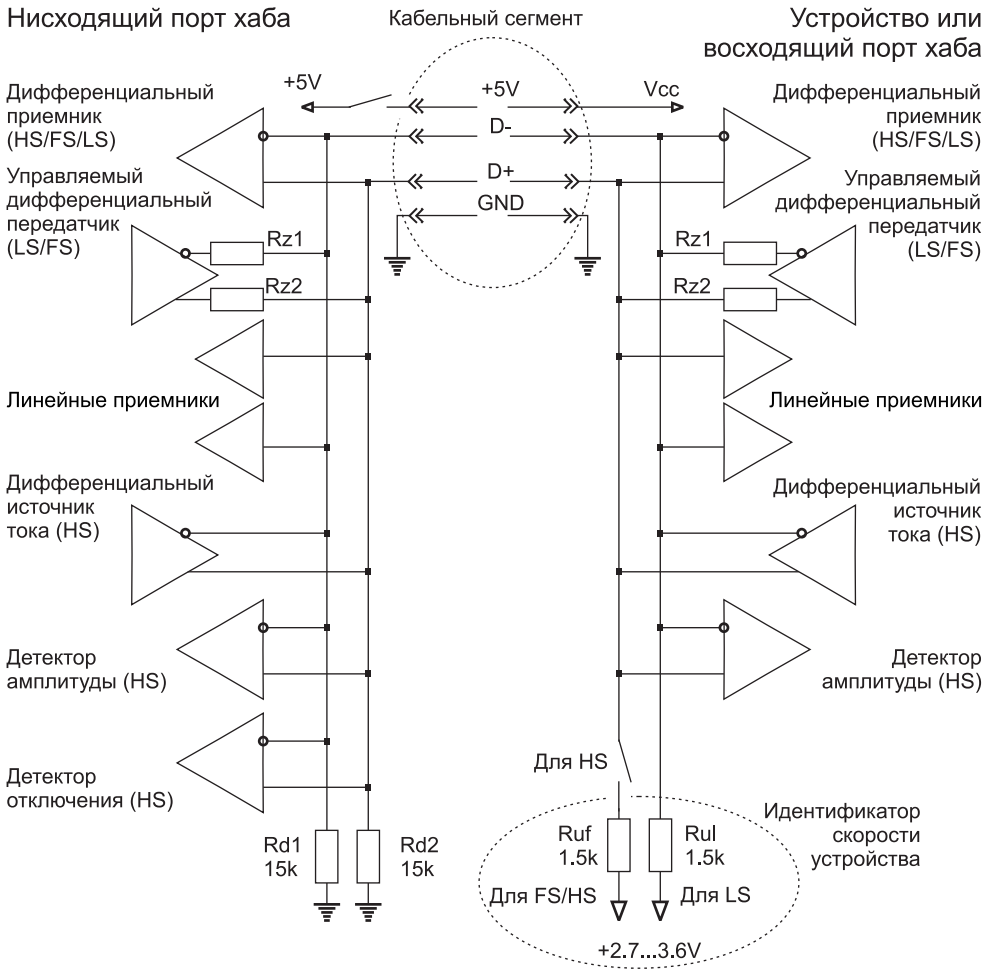


Рис. 12.2. Физический интерфейс USB

Передача данных

Передача по двум проводам USB не ограничивается лишь дифференциальными сигналами. Приемники и передатчики позволяют использовать множество состояний линий и команд, используемых для организации аппаратного интерфейса. При этом учитываются не только уровни электрических сигналов, но и время нахождения их в том или ином состоянии. По уровням напряжения на входах приемников различают сигналы:

- *Diff0*: (D+) – (D-) > 200 мВ при (D+) > 2 В;
- *Diff1*: (D-) – (D+) > 200 мВ при (D-) > 2 В;
- *SE0* (single-ended zero): (D+) < 0,8 В и (D-) < 0,8 В.

Для передачи данных используются сигналы *Diff0* и *Diff1*, они кодируют *состояния J* (*Data J State*) и *K* (*Data K State*). На полной и высокой скорости *состояние J* соответствует сигналу *Diff1*, *состояние K* — сигналу *Diff0*. На низкой скорости назначение обратное: *J* — *Diff0* и *K* — *Diff1*. Последовательная передача информации ведется с использованием кодирования NRZI (рис. 12.3): при передачи нулевого бита в начале битового интервала состояние сигнала (*J* или *K*) меняется на противоположное; при передаче единичного — не меняется. Длительность битового интервала определяется номинальной частотой передачи: 0,666... мкс для низкой скорости (LS, 1,5 Мбит/с); 83,3... нс для полной (FS, 12 Мбит/с) и 2,0833... нс для высокой (HS, 480 Мбит/с).

Состояние покоя (*Bus Idle*) на FS/LS соответствует длительному состоянию *J*, а на HS — состоянию *SE0*.

Признаком начала пакета является переход из состояния покоя в состояние *K*, что является первым битом *синхропоследовательности* (`Sync`), — последовательности нулей, которая в NRZI кодируется переключением состояний (*J* и *K*) в начале каждого битового интервала. Синхропоследовательность позволяет приемнику настроиться на нужную частоту и фазу синхронизации. Синхропоследовательность завершает единичный бит (нет смены состояния), последующие за ним биты относятся к идентификатору и телу пакета. На HS начальная часть синхропоследовательности может быть потеряна хабом (из-за задержки реакции на детектор сигнала). С учетом этого синхросполедовательность для HS удлинена до 32 бит (включая последний единичный бит). Проходя через 5 хабов, каждый из которых может потерять до 4 синхробит, синхропоследовательность может оказаться сокращенной до 12 бит.

Для того чтобы синхронизация не терялась на монотонном сигнале (при передаче длинной последовательности единиц), применяется *техника вставки бит* (bit stuffing): после каждых 6 подряд следующих единиц передатчик вставляет «0», приемник эти вставленные биты удаляет. Если принимается более 6 единиц подряд, это считается ошибкой вставки бит.

Конец пакета (*EOP*) на FS/LS обозначается сигналом *SE0*, длящимся 2 битовых интервала, за которым следует переход в состояние покоя (*Bus Idle*). На HS для признака *EOP* используется нарушение правила вставки бит. Здесь в качестве *EOP* используется передача последовательности 01111111 *без вставки бит*. Прием седьмой единицы вызовет индикацию ошибки вставки бит, которая на HS и является признаком конца пакета. Нормальный пакет при этом от действительно ошибочного будет отличаться целым количеством принятых байт (это условие может и не проверяться) и верным значением CRC. Начальный нолик (вызывающий смену состояния) в *EOP* облегчает точное определение границы тела пакета. В пакетах *SOF* поле *EOP* удлинено до 40 бит для обнаружения отключения устройства (см. ниже).

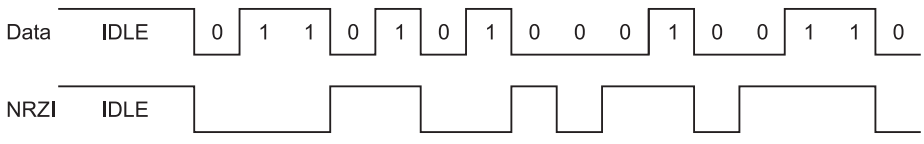


Рис. 12.3. Кодирование данных по методу NRZI

Особенности сигналов в режиме HS

Высокая скорость (480 Мбит/с — всего в 2 раза медленнее, чем Gigabit Ethernet) требует тщательного согласования приемопередатчиков и линий связи. На этой скорости может работать только кабель с экранированной витой парой для сигнальных линий. Для высокой скорости аппаратура USB должна иметь дополнительные специальные приемопередатчики. К разводке проводов на печатной плате устройства от интерфейсной микросхемы USB до разъема (или подключения кабеля) предъявляют жесткие требования (максимальная длина, совпадение длин сигнальных проводников, удаленность от других сигнальных цепей, окружение «землей»).

В отличие от формирователей потенциала для режимов FS и LS передатчики HS являются *источниками тока*, ориентированными на наличие резисторов-терминаторов на обеих сигнальных линиях. Роль терминаторов играют резисторы Rz1 и Rz2 (см. рис. 12.2): при работе на HS дифференциальный передатчик FS/LS формирует *SE0*, то есть оба его выхода заземляются и эти резисторы оказываются нагрузками для линий D+ и D-. Их сопротивление (с учетом выходного импеданса передатчика) составляет 2×45 = 90 Ом, что и обеспечивает согласование с волновым сопротивлением линии (90 Ом). Устройство и хаб включают свои HS-терминаторы (и отключают Ruf) после успешного взаимного подтверждения режима HS, выполняемого в процессе сброса устройства.

Дифференциальные токовые *передатчики* формируют импульсы тока с номинальным значением 17,78 мА, который протекает через нагрузку 22,5 Ом (два нагрузочных резистора на обоих концах каждой сигнальной линии соединяются параллельно). При передаче сигнала *J* ток пропускается в линию D+, при *K* — в D-. Таким образом обеспечивается дифференциальный сигнал передачи около ±400 мВ[1].

На вход дифференциального *приемника* сигнал придет ослабленным; чтобы исключить влияние шумов, в схему устройства введен детектор амплитуды сигнала с порогом 100–150 мВ. Сигнал с дифференциального приемника игнорируется, пока не сработает детектор амплитуды сигнала (в спецификации USB этот прием называется receiver squelch). От срабатывания детектора амплитуды до включения дифференциального приемника может быть задержка до 4 bt, но это приведет лишь к сокращению длины принятой синхропоследовательности в начале пакета (см. далее).

К статическим (уровни) и динамическим (длительности и время нарастания и спада) параметрам сигналов на HS предъявляются жесткие требования, и существуют специальные шаблоны (Eye Pattern), в которые должны укладываться сигналы. Для тестирования могут быть использованы широкополосные (не у II же 1 ГГц)

[1] При таком способе формирования сигналов появляется и линейная (синфазная) составляющая сигнала в двухпроводной линии; амплитуда этого сигнала равна половине амплитуды дифференциального сигнала. Для формирования чисто дифференциального сигнала требуется одновременная подача разнополярных импульсов тока в D+ и D-, что и приходится делать для передачи сигналов на большие расстояния. В USB принятое упрощение дифференциальной сигнализации позволяет в рамках ограничений на дальность (5-метровый кабельный сегмент) применить дешевое решение.

дифференциальные осциллографы и генераторы; выпускаются и специализированные тестеры устройств USB 2.0. Для тестирования HS-устройств (включая и хабы) в USB 2.0 определены специальные управляющие запросы, переводящие выбранный порт в тестовый режим. В стандартных запросах определены следующие тесты:

- *Test_SE0_NAK* — перевод порта в HS-режим для тестирования выходного импеданса, уровня сигнала *SE0* и нагрузочных характеристик;
- *Test_J* — передача состояния *J* (подача тока в D+) для проверки уровня сигнала, выводимого на линию D+;
- *Test_K* — передача состояния *K* (подача тока в D-) для проверки уровня сигнала, выводимого на линию D-;
- *Test_Packet* — передача пакетов фиксированной структуры, позволяющая проверять динамические параметры сигнала и подключенные к нему приемники. Тестовый пакет выглядит как пакет данных *DATA0*, у которого в поле данных последовательно расположены 6 тестовых образцов (test pattern). Далее приводятся последовательности сигналов, передающихся на шину в этих образцах (вставка бит уже произведена):
  - Pattern 1 — 36 пар «JK»;
  - Pattern 2 — 16 повторов «JJKK»;
  - Pattern 3 — 8 повторов «JJJJKKKK» (4J и 4K);
  - Pattern 4 — 8 повторов «JJJJJJJKKKKKKK» (7J и 7K);
  - Pattern 5 — 8 повторов «JJJJJJJK» (7J и 1K);
  - Pattern 6 — 10 повторов «JKKKKKKK» (1J и 7K).
- *Test_Force_Enable* — принудительный перевод нисходящих портов хаба в режим HS, даже при отключенных устройствах (для настройки детектора отключения).

Специальная сигнализация: обнаружение подключения-отключения, сброс устройств, приостановка и пробуждение

Хаб обнаруживает *подключение устройства* по уровням напряжения на линиях D+ и D–:

- при отключенном устройстве на линиях D+ и D- уровни сигнала низкие (состояние *SE0*), что обусловлено резисторами Rd1 и Rd2 хаба;
- при подключении LS-устройства повышается уровень сигнала D- за счет резистора Rul в устройстве (переход в состояние *LS-Idle*);
- при подключении FS/HS-устройства повышается уровень сигнала D+ за счет резистора Rul в устройстве (переход в состояние *FS-Idle*).

Последовательность обнаружения подключения и сброса устройств FS и LS приведена на рис. 12.4, *а* и *б* соответственно. Хаб следит за сигналами нисходящего пор-

та и сигнализирует об их смене. После обнаружения смены состояния системное ПО выжидает около 100 мс (время на успокоение сигналов) и проверяет состояния порта. Обнаружив факт подключения и тип устройства (LS или FS/HS), ПО дает для этого порта команду сброса шины.

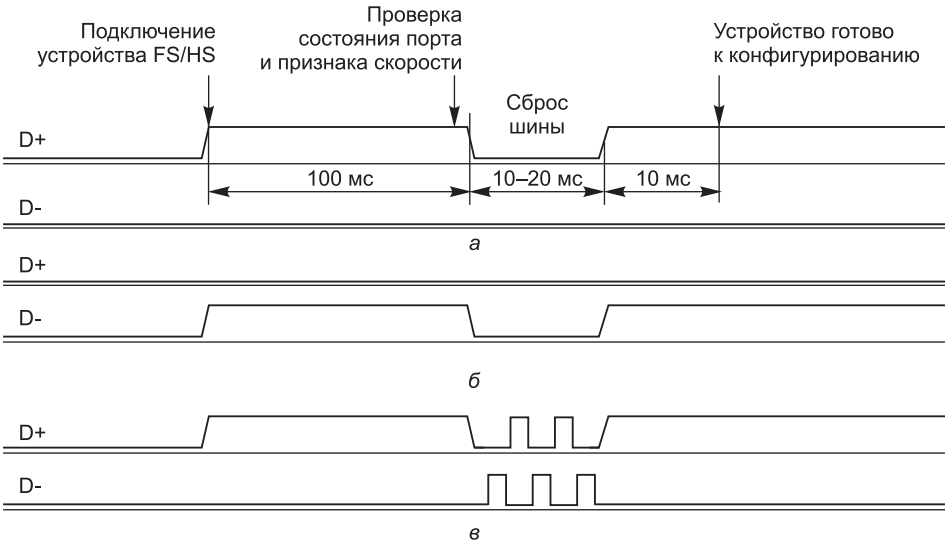


Рис. 12.4. Обнаружение подключения и сброс устройства: *а* — FS, *б* — LS, *в* — HS

Для выполнения *сброса шины* (команда *Bus Reset*) хаб опускает уровень поднятого устройством сигнала (D+ или D-) на 10–20 мс (то есть подает сигнал *SE0* в течение 10–20 мс). Считается, что через 10 мс после этого сброса устройство должно быть готово к конфигурированию (отзываться только на обращения к *EP0* по нулевому адресу устройства).

Сброс шины для устройства HS запускает протокол *согласования скорости*. При подключении, как и по сигналу сброса, HS-устройство устанавливает свои схемы в состояние FS (отключая терминаторы и включая Ruf). Таким образом, поначалу HS-устройство выглядит для хаба как FS-устройство. Для согласования скорости используется так называемое «чириканье» (chirp-sequence): в ответ на состояние *SE0*, введенное хабом для сброса (заземлением линии D+), HS-устройство своим дифференциальным токовым передатчикам вводит состояние «*chirp-K*» (пуская импульс тока в линию D-). На этот импульс HS-хаб ответит импульсом на линии D+, так что получится состояние «*chirp-J*». Такой обмен импульсами повторяется еще дважды; после успеха согласования и устройство и хаб принимают режим работы HS (и резистор Ruf отключается). Все это «чириканье» занимает 10–20 мс, после чего шина переходит в состояние покоя *HS-Idle* (длительный сигнал *SE0*). Теперь хосту надо снова опросить состояние порта хаба, чтобы уточнить режим подключенного устройства (FS или HS). Если HS-устройство подключено к FS-порту, хаб на «чириканье» устройства не ответит.

Отключение устройств FS/LS обнаруживается хабом просто по длительному (более 2 мкс) состоянию *SE0*. Этот факт хаб доводит до сведения системного ПО (USBD), чтобы устройство было вычеркнуто из всех рабочих списков. Отключение устройств HS таким способом обнаружить не удается, поскольку состояние шины (*SE0*) при отключении устройства не изменится. Для обнаружения отключения HS-устройства используют эффект отражения сигнала при потере согласованности линии. Специально для этих целей в схему хаба введен дополнительный *детектор отключения*, а в маркере микрокадра *SOF* признак *EOP* (0111...111) удлинен до 40 битовых интервалов. Транслируя *SOF* на высокоскоростной порт, детектор отключения следит за уровнем сигнала *J*, и если он превышает порог (625 мВ дифференциального сигнала), значит, нагрузки на другой стороне нет, то есть устройство отключено. Удлинение *EOP* необходимо, поскольку устройство может отключиться внутренне, и из-за задержки в кабеле устройства (2×26 нс) отраженный сигнал может задержаться до 25 нс. С целью сокращения накладных расходов это удлинение *EOP* сделали только для пакетов *SOF*, появляющихся всего раз в 125 мкс.

Команду приостановки устройства — Suspend хаб сигнализирует длительным состоянием покоя (*Bus Idle*). При этом он должен переставать транслировать все кадры, включая и маркеры микрокадров на порты, для которых подается эта команда. На порты, работающие в LS-режиме, маркеры кадров не транслируются; чтобы LS-устройство не приостанавливалось при отсутствии полезного трафика, ему вместо маркеров *SOF* хаб с тем же периодом посылает сигнал *LS-EOP* (*SE0* в течении 1,33 мкс). Приостановка делается не менее чем на 20 мс — за это время устройство должно успеть перейти в приостановленное состояние и стать готовым к получению сигнала возобновления.

Команду приостановки HS-порта хаб сигнализирует покоем (*SE0*) в течение 3 мс, после чего переключает свои цепи в режим FS (отключает терминаторы), но помнит, что порт находится в режиме HS. Для HS-устройства команда приостановки поначалу неотличима от сброса. Чтобы их различить, через 3–3,125 мс непрерывного состояния *SE0* HS-устройство переключает свои цепи в режим FS (отключает терминаторы и включает Ruf). Далее, через 100–875 мкс устройство проверяет состояние линий. Если обе лини `D+` и `D-` оказались в низкоуровневом состоянии, значит, хаб подал команду сброса (и устройство должно выполнить chirp-последовательность). Если уровень `D+` высокий, а `D-` низкий (*FS-Idle*), то это сигнал к приостановке. Таким образом, по состоянию сигналов на шине приостановка выглядит как покой LS/FS — то есть состояние *J*.

Сигналом к возобновлению работы (resume) является перевод шины в состояние *K* на длительное время (20 мс), достаточное для «оживления» устройств, после чего хаб посылает сигнал *LS-EOP* (*SE0* в течение 1,33... мкс). После этого шина переходит в состояние покоя соответствующей скорости и начинает передаваться трафик. Сигнал возобновления может подать как хаб, так и приостановленное устройство; последний случай называется *удаленным пробуждением*. По сигналу возобновления устройство, работавшее в HS-режиме, и его порт хаба переключают свои цепи в HS-режим без всякого согласования (они помнят свой режим).

Удаленное пробуждение — Remote Wakeup — это единственный случай на USB, когда сигнальную инициативу проявляет устройство (а не хост). Сигнал пробуждения может подать только приостановленное устройство, для которого шина находится в FS/LS-состоянии *J* (резисторами подтягивается вверх D+ или D-). Для сигнализации пробуждения устройство на некоторое время (1–15 мс) формирует состояние *K*, которое воспримется хабом как сигнал *Resume* и транслируется им на восходящий порт и на все разрешенные нисходящие порты, включая и тот порт, с которого пришел данный сигнал.

Питание от шины

Шина USB обеспечивает устройства питанием по линии Vbus с номинальным напряжением 5 В относительно линии GND. Питание подается на нисходящие порты хабов; устройства-функции могут только потреблять питание (как и хаб по своему восходящему порту). Естественно, устройства могут пользоваться и собственным питанием. Мощность питания от шины выделяют единицами по 100 мА, устройству шина может предоставить максимум 5 единиц (0,5А). При начальном подключении (до конфигурирования) устройство может потреблять не более 1 единицы (100 мА). Порт, обеспечивающий 5 единиц, называют *мощным* (high-power port); *маломощный порт* (low-power port) обеспечивает лишь 1 единицу. По отношению к питанию от шины различают следующие типы устройств:

- корневой хаб, получающий питание вместе с хост-контроллером. При питании от внешнего источника хаб должен иметь мощные порты; при автономном (от батарей) питании порты могут быть как мощными, так и маломощными;
- хаб, питающийся от шины (Bus-powered hub), может иметь только маломощные порты (и не более четырех, поскольку одну единицу мощности потребляет его контроллер). Питание на нисходящие порты этот хаб подает только после того, как будет сконфигурирован (поскольку до того он может потреблять лишь 1 единицу);
- хабы с собственным питанием (Self-powered hub) могут потреблять от шины лишь одну единицу. На свои нисходящие порты он подает питание от другого источника; эти порты могут быть как мощными, так и маломощными (у хаба с батарейным питанием);
- маломощные устройства с питанием от шины (Low-power bus-powered functions) могут потреблять не более одной единицы;
- мощные устройства с питанием от шины (High-power bus-powered functions) могут потреблять до пяти единиц;
- устройства с собственным питанием (Self-powered functions) могут потреблять от шины не более одной единицы, даже если их собственное питание пропадает. Остальную мощность, необходимую для работы, они должны брать от других источников.

Способ питания устройства (и хаба) и максимальный ток, потребляемый от шины (с точностью до 2 мА), описываются в дескрипторе конфигурации устройства

(см. главу 13). У хаба с собственным питанием возможна ситуация, когда это питание в процессе работы теряется. В этом случае хаб должен отключиться от шины и автоматически повторно подключиться, но уже сообщая в своем дескрипторе о питании от шины. При этом временно отсоединяются и все находящиеся за ним устройства, после чего они конфигурируются уже исходя из бюджета новых условий питания.

Питание на порты хабов подается с защитой от перегрузок, из расчета 5 А на порт (это не отменяет нормы потребления). Срабатывание токовой защиты может индицироваться, например, гудком динамика системной платы ПК. Управление подачей питания у хаба может быть как общим (на все порты сразу), так и селективным — эти возможности описаны в дескрипторе нулевой конечной точки хаба.

При питании от шины до устройств доходит напряжение меньшее, чем дает хаб, из-за сопротивления питающих проводов и контактов разъемов. На каждом кабеле (между вилками A и B) в каждой из линий GND и Vbus может падать напряжение до 0,125 В. Худший случай по питанию — когда между источником питания (хабом с собственным питанием) и устройством находится один хаб с питанием от шины, вносящий свое падение напряжения (до 350 мВ). Мощный порт хаба должен под нагрузкой выдавать напряжение в диапазоне 4,75–5,25 В, маломощный — 4,4–5,25 В. Устройство, питающееся от шины, должно быть способно сообщать конфигурационную информацию при напряжении питания на вилке «A» своего кабеля от 4,4 В; маломощное устройство при таких условиях должно и нормально работать. Для нормальной работы мощного устройства требуется как минимум 4,75 В на его вилке.

Управление потреблением: приостановка, возобновление и удаленное пробуждение

USB имеет развитую *систему управления энергопотреблением*. Хост-компьютер может иметь собственную систему управления энергопотреблением (power management system), к которой логически подключается и одноименная система USB. Программное обеспечение USB взаимодействует с этой системой компьютера, поддерживая такие системные события, как *приостановка* (suspend) или *возобновление* (resume). Кроме того, устройства USB могут сами являться источниками событий, отрабатываемых системой управления энергопотреблением хоста.

Все устройства USB должны поддерживать *режим приостановки* (suspended mode), в котором средний потребляемый от шины ток не превышает 500 мкА. Мощным устройствам, которые могут инициировать удаленное пробуждение, позволительно в этом режиме потреблять до 2,5 мА. Устройство должно автоматически приостанавливаться при прекращении активности шины. Приостановка выполняется всегда по инициативе хоста и может быть как глобальной, так и селективной. *Возобновление* (resume) может происходить по разным причинам и сценариям.

Глобальная приостановка выполняется через корневой хаб — специальная управляющая команда запрещает ему транслировать весь нисходящий трафик, что и вы-

зывает сигнализацию приостановки. Это заставит все устройства и хабы шины перейти в состояние приостановки. *Возобновление после глобальной приостановки* возможно несколькими путями:

- по инициативе хоста — командой к корневому хабу, что приведет к сигнализации возобновления для всех подключенных к нему сегментов;
- по инициативе устройства — удаленным пробуждением (remote wakeup). Сигнал возобновления может подать любое устройство, которому управляющим запросом разрешена миссия «будильника». Этот сигнал воспринимается портом хаба, к которому подключено устройство-«будильник», после чего посылается хабом во все разрешенные порты («отражаясь» и на порт-источник сигнала) и на восходящий порт. Распространяясь таким образом, сигнал возобновления дойдет и до корневого хаба, который еще 20 мс транслирует этот сигнал на нисходящие порты, после чего завершает сигнализацию возобновления посылкой сигнала *LS-EOP*;
- по событию порта хаба (подключению или отключению устройства), которому управляющим запросом разрешена генерация удаленного пробуждения;
- сбросом шины, вызывающим реконфигурирование всех устройств.

Селективная приостановка относится к сегменту шины или даже отдельному устройству. Для этого хабу, к которому подключен приостанавливаемый сегмент (устройство), подается управляющий запрос установки *Set_Port_Suspend*, что запретит нисходящую трансляцию для выбранного порта. *Возобновление после селективной приостановки* выполняется несколько иначе:

- по инициативе хоста — посылкой к хабу управляющего запроса отмены *Port_Suspend*. Это приведет к сигнализации возобновления на данный порт в течение 20 мс, завершающейся посылкой сигнала *LS-EOP*. После этого хаб возобновляет трансляцию нисходящего трафика на этот порт, а через 3 мс устанавливает в своем регистре состояния данного порта признак завершения процедуры возобновления. За это время подключенное устройство успеет засинхронизироваться с маркерами кадров;
- по инициативе устройства — удаленным пробуждением. Здесь хаб, который ввел приостановку, ведет себя иначе: он не распространяет сигнал возобновления на другие порты (там могут находиться активно работающие устройства и передаваться трафик). Восприняв сигнал возобновления, хаб сам его посылает в течение 20 мс на тот же порт, затем посылает в него сигнал *LS-EOP* и через 3 мс устанавливает признак завершения возобновления для этого порта (сбрасывает бит `Port_Suspend`);
- по событию порта хаба (подключению или отключению устройства), которому управляющим запросом разрешена генерация удаленного пробуждения;
- сбросом шины, вызывающим реконфигурацию всех устройств.

Если на селективно приостановленном порте хаба происходит событие подключения-отключения, то этот порт из состояния *suspended* перейдет в *connected* или *disconnected* в соответствии с состоянием текущего подключения.

За селективной приостановкой какого-то порта хаба может последовать и общая приостановка данного хаба (глобальная или тоже селективная). Это не помешает распространению сигнала удаленного пробуждения вверх. Когда до хаба сверху дойдет сигнал возобновления, состояние *suspended* его портов, селективно приостановленных, сохранится — хост должен будет снять приостановку соответствующими запросами. Удаленное пробуждение автоматически снимет состояние *suspended* с порта, на который оно пришло.

ГЛАВА 13

Устройства USB

Устройство (функция и хаб) USB должно подчиняться требованиям физического интерфейса и протокола шины, обеспечивать корректную смену своих состояний в терминах спецификации USB, доступ к дескрипторам и выполнение стандартных запросов. В данной главе рассматриваются общие вопросы поведения устройства на шине и механизмы автоматического конфигурирования. Особенности хабов и устройств-функций, связанные с их прикладной частью, рассматриваются в последующих главах.

Структура устройства с интерфейсом USB

Периферийное устройство с интерфейсом USB можно разделить на две части — интерфейсную и функциональную (рис. 13.1). Физически они могут объединяться и на одной микросхеме, но логически их функции четко разделимы.

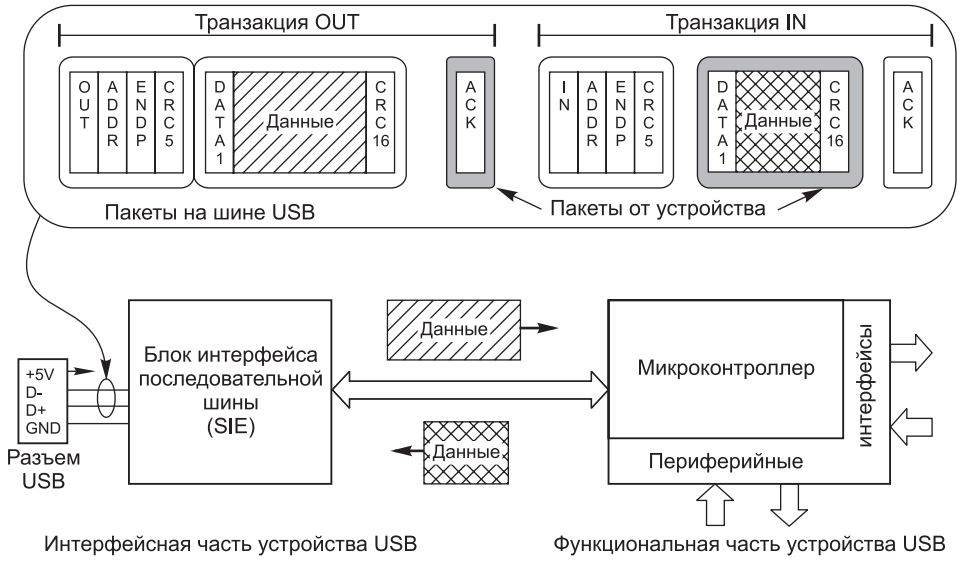


Рис. 13.1. Структура устройства USB

Все протокольные и сигнальные функции USB обеспечивает *блок последовательного интерфейса*, *SIE* (Serial Interface Engine). В сторону USB блок SIE «смотрит» своим портом USB (комплектом приемопередатчиков, рассмотренным в главе 12). Блок SIE занимается последовательным приемом и передачей пакетов, выполняя подсчеты и проверки CRC, вставку битов (bit stuffing) при передаче и их удаление при приеме, кодирование NRZI, проверку форматов, отработку подтверждений и отслеживание корректной последовательности пакетов. С функциональной частью устройства блок SIE обменивается только «чистыми» пользовательскими данными. SIE сигнализирует о приходе очередного пакета к той или иной конечной точке, принимает от функциональной части данные к выдаче (вводу по запросу хоста), сообщает о выполнении этой операции. Количество и тип поддерживаемых конечных точек зависят от реализации SIE. Самые сложные в плане поддержки — точки типа *Control*, по этой причине многие устройства USB поддерживают лишь одну (обязательную) управляющую точку — *EP0*. С каждой поддерживаемой точкой в SIE связана буферная память, объем которой должен соответствовать максимальному размеру пакета, заявленному в дескрипторе точки. Блок SIE ведает и всеми дескрипторами (они размещаются в его локальной памяти) — сообщает их хосту по запросам, устанавливает конфигурацию и альтернативные установки. SIE отрабатывает и все запросы хоста, стандартные и специфические (управляет конечными точками, организует засыпание и пробуждение).

Устройство USB должно поддерживать возможность работы на полной, низкой или высокой скорости, в зависимости от требуемой скорости передачи данных и исходя из технико-экономических соображений. Низкоскоростные устройства (и их кабели) обходятся несколько дешевле, но их широкое использование невыгодно с точки зрения производительности шины в целом (см. главу 11). Высокоскоростной порт USB требуется только при довольно высокой производительности функциональной части устройства, его применение несколько удорожает устройство (правда, на фоне стоимости функциональной части это не так существенно).

Как правило, периферийные устройства с USB имеют встроенный микроконтроллер, который и является источником и приемником информации, посылаемой через конечные точки. Микроконтроллер должен подчиняться указаниям от шины — выполнять сброс и приостановку по сигналам от порта, отрабатывать установки конфигурации и интерфейсов. Запросы управления стандартными свойствами (остановка и разблокирование точек, разрешение посылки удаленного пробуждения) доходят до контроллера опосредованно — в первую очередь их отрабатывает SIE.

Интерфейс между SIE и микроконтроллером обеспечивает передачу данных с необходимыми сигналами управления, а также генерацию прерываний (или иную сигнализацию) для микроконтроллера по таким событиям, как приход пакета, освобождение буфера передающей EP, срабатывание меток времени (для изохронных точек), неисправимые протокольные ошибки, вызывающие блокировку конечных точек.

Состояния устройств

Устройство USB должно поддерживать все *состояния*, определенные спецификацией:

- «*Подключено*» (*Attached State*) — устройство подключено к хабу, но питание от шины не подано, устройство не может никак себя проявить и не управляемо хостом. Если питание от шины не используется (даже для SIE), то это состояние отсутствует;
- «*Запитано*» (*Powered State*) — устройство подключено к порту и ему подано питание, устройство может заявить о себе, подтягивая резистором линию D+ или D- к шине питания. Это промежуточная ступенька к «дежурному» состоянию;
- «*Дежурное*» *состояние* (*Default State*) по включению питания, подключению к порту или по сбросу от порта: устройство имеет *нулевой адрес* (USB Default Address) и отзывается только на обращения к *EP0*, потребляет от шины не более 100 мА;
- «*Адресовано*» (*Addressed State*) — запросом *Set_Address* ему назначен *уникальный адрес на шине* (1–127), но отзывается только на обращения к *EP0*, потребляет от шины не более 100 мА;
- «*Сконфигурировано*» (*Configured State*) — запросом *Set_Configuration* выбрана конфигурация, устройство отзывается на обращения ко всем точкам, описанным в данной конфигурации, и может потреблять от шины заявленный ток. При необходимости можно изменять альтернативные установки интерфесов запросом *Set_Interface*;
- «*Приостановлено*» (*Suspended Mode*) — устройство подключено, запитано (хотя бы по минимуму), но приостановлено (прекращение активности порта, к которому оно подключено). Ему до приостановки мог быть назначен адрес и установлена конфигурация, однако хост не может использовать функции этого устройства, пока не будет выполнено возобновление (*resume*), которое вернет устройство в состояние, бывшее до приостановки. В этом состоянии устройство может подать сигнал удаленного пробуждения, если оно обладает этой возможностью и хост разрешил ее использовать.

Конфигурирование устройств и управление ими

USB поддерживает динамическое конфигурирование, отслеживая подключение и отключение устройств. USB позволяет идентифицировать подключаемые устройства, определять их потребности в ресурсах (полоса пропускания, питание от шины), выбирать нужную конфигурацию и управлять устройствами, что обеспечивает полную поддержку PnP. Для этих целей определены «правила поведения» подключаемых устройств, система дескрипторов и стандартные управляющие запросы к устройствам. Ключевую роль в системе PnP играют хабы, позволяющие

селективно управлять работой подсоединенных к ним сегментов шины, что требуется на этапе конфигурирования. В процессе работы шины постоянно идет процесс *нумерации* (enumeration) устройств, отслеживающий изменения физической топологии.

Автоматическое конфигурирование

Все устройства подключаются через порты хабов. Хабы определяют подключение и отключение устройств к своим портам (см. главу 14) и сообщают состояние портов по запросу от контроллера. Хост своим управляющим запросом *Port_Reset* к хабу выполняет сброс и разрешает работу порта (одного!), на котором обнаружено новое подключение. При начальном подключении или после сброса устройство находится в «*дежурном*» состоянии (*Default State*) — отзывается только на обращения по основному каналу сообщений (*EP0*) и имеет *нулевой адрес* (*USB Default Address*). Таким образом, обращаясь к устройству по нулевому адресу, хост взаимодействует только с одним новоподключенным устройством. Хост стандартными запросами считывает дескрипторы этого устройства и назначает ему *уникальный адрес на шине* (1–127). Таким образом хост заполняет свой перечень подключенных устройств. По назначении уникального адреса устройство переходит в состояние «*адресовано*» (*Addressed State*), но его прикладное функционирование пока еще не разрешено. Полноценная работа устройства (прикладной обмен с хостом, полное потребление питания от шины) возможна только после управляющего запроса от хоста *Set_Configuration,* выбирающего конфигурацию из числа доступных — устройство переходит в состояние «*сконфигурировано*» (*Configured State*).

Если новое устройство является хабом, хост, сконфигурировав его, таким же способом определяет подключенные к нему устройства, идентифицирует их, назначает адреса и конфигурирует. Если новое устройство является функцией, уведомление о подключении передается заинтересованному ПО и, при необходимости, для него загружаются клиентские драйверы.

Когда устройство отключается, хаб автоматически запрещает соответствующий порт и сообщает об отключении хосту, который удаляет сведения о данном устройстве из всех рабочих структур данных (но не из реестра Windows!). Если отключается устройство-функция, уведомление посылается заинтересованному ПО. Если отключается хаб, процесс удаления выполняется для всех подключенных к нему устройств.

Идентификация и классификация устройств

Обнаружив подключение устройства (по сообщению от хаба), система USB считывает его дескрипторы, чтобы определить, какие программные компоненты необходимо загрузить и кого уведомлять о появлении нового устройства. В любом устройстве USB обязательно должны присутствовать дескрипторы устройства, конфигурации, интерфейсов и конечных точек. Подробно структура дескрипторов

описана далее, здесь рассматриваются только фрагменты дескрипторов, участвующих в идентификации устройств.

В *дескрипторе устройства* имеются 2-байтные поля, идентифицирующие устройство в целом:

- `idVendor` — идентификатор производителя (*VID* — Vendor Id), назначаемый USB-IF;
- `idProduct` и `bcdDevice` — идентификатор продукта (*PID* — Product Id) и его версии (*DID* — Device Id), определяются производителем.

Кроме того, здесь могут присутствовать ссылки на строковые дескрипторы, в которых содержатся текстовые названия изготовителя и устройства, а также его серийный номер. Эти текстовые описания имеют произвольную длину и формат (но кодируются в UNICODE), на эти строковые дескрипторы указывают индексы в полях `iManufacturer`, `iProduct` и `iSerialNumber`.

Для определения назначения, возможностей и протоколов, поддерживаемых устройством и его отдельными интерфейсами, в его дескрипторах указываются *коды класса, подкласса и протокола*. Коды класса, подкласса и протокола имеют непосредственное отношение к интерфейсам — по ним могут быть подобраны (операционной системой автоматически) подходящий драйвер и клиентское приложение. Эти коды содержатся в дескрипторах интерфейсов, поддерживаемых устройством. «Штатные» коды в диапазоне 01h–FEh назначает USB-IF, но только для уже стандартизованных устройств. Значение 00h означает отсутствие определения, FFh отдано для специфического назначения разработчикам и производителям устройств. Сообщение штатного кода обязывает устройство соответствовать стандартным требованиям, предъявляемым к интерфейсам с указанным протоколом для заданного класса и подкласса, в том числе и выполнять все специфические запросы и сообщать специфические дескрипторы, если таковые имеются. При этом допускается и расширение возможностей устройства. Для связывания нестандартного устройства со своим драйвером используются идентификаторы *VID* и *PID*.

Коды класса, подкласса и протокола присутствуют не только в дескрипторах интерфейсов, но и в *дескрипторе устройства*. Здесь нулевой код класса означает, что устройство состоит из набора независимых интерфейсов, каждому из которых может быть назначен свой код класса, подкласса и протокола. При этом и подкласс и протокол устройства тоже нулевые (то есть устройство в целом стандартно охарактеризовать нельзя). «Штатный» код класса устройства означает, что его интерфейсы не являются независимыми (агрегированные интерфейсы). При этом код подкласса (тоже от USB-IF) является дополнительным квалификатором. «Штатный» код протокола означает, что устройство поддерживает все протоколы, требуемые для устройства данного класса и подкласса. Нулевой код протокола устройства означает, что протоколы могут быть определены только для отдельных интерфейсов.

Классификация устройств USB относится не к потребительским функциям, выполняемым устройствами, а к способам коммуникаций между хостом и устрой-

ствами. Классификация позволяет обобщать характеристики интерфейсов, при этом, как правило, код протокола задает состав, тип конечных точек и правила их использования, а подкласс определяет форматы данных, передаваемых через те или иные конечные точки. Классификация позволяет сократить многообразие (разнотипность) драйверов, требуемых для различных устройств, — драйвер может абстрагироваться от конкретного устройства-функции, которое он обслуживает. Операционная система связывает имеющиеся в ее распоряжении клиентские драйверы с конкретными интерфейсами устройств, используя коды классов/подклассов и протокола, а также идентификаторы производителя, продукта и его версии.

Примеры классов, подклассов и протоколов приведены в табл. 13.1, полную информацию о состоянии классификации и требования к уже определенным классам, подклассам и протоколам можно найти на http://www.usb.org. Некоторые из этих классов подробнее рассмотрены в главе 16.

Таблица 13.1. Некоторые стандартные классы и протоколы устройств

Подкласс	Протокол (точки, используемые интерфейсом)
Класс 01 — аудиоустройства	
01 — *AUDIOCONTROL*, управляемый модуль аудиообработки (регулятор, фильтр, микшер, ревербератор...) 02 — *AUDIOSTREAMING*, устройство-приемник или источник аудиопотока 03 — *MIDISTREAMING*, устройство-приемник или источник потока MIDI-сообщений	00 — протокол не определен
Класс 03 — человеко-машинный интерфейс (HID-устройства)	
01 — устройства, используемые при загрузке ОС	01 — клавиатура 02 — мышь
Класс 07 — принтеры	
01 — передача к принтеру данных, используя любые языки описания страниц (PCL), применяемые в принтерах с традиционными интерфейсами, и прием информации о состоянии	01 — однонаправленный (*EP0*, *Bulk-OUT*) 02 — двунаправленный (*EP0*, *Bulk-OUT*, *Bulk-IN*) 03 — двунаправленный IEEE 1284.4 (*EP0*, *Bulk-OUT*, *Bulk-IN*)
Класс 08 — устройства хранения данных (mass storage)	
01 — сокращенный набор команд (RBC — Reduced Block Commands), типично для устройств на флэш-памяти, но этот набор команд могут использовать любые устройства хранения 02 — SFF8020i, MMC-2 (ATAPI), типично для CD/DVD-устройств 03 — QIC-157 (ленточные устройства) 04 — UFI, типично для НГМД 05 — SFF8070i, типично для НГМД 06 — прозрачная передача команд SCSI	00 — CBI-транспорт с прерываниями (*EP0*, *Bulk-OUT*, *Bulk-IN*, *Interrupt-IN*) 01 — CBI-транспорт без прерываний (*EP0*, *Bulk-OUT*, *Bulk-IN*, *Interrupt-IN*), только для FS 50h — BO-транспорт (*Bulk-OUT*, *Bulk-IN*, *EP0*)

продолжение ⇩

Таблица 13.1 (*продолжение*)

Подкласс	Протокол (точки, используемые интерфейсом)
Класс 09h — хабы	
00 — деления на подклассы нет	00 — хаб USB 1.x; 01 — хаб USB 2.0 с одним транслятором транзакций; 02 — хаб USB 2.0 с множеством трансляторов транзакций
Класс 0Eh — видеоустройства	
01 —*VIDEOCONTROL*, управляемое устройство видеообработки 02 — *VIDEOSTREAMING*, устройство-приемник или источник видеопотока 03 — *VIDEO_INTERFACE_COLLECTION*, набор связанных интерфейсов видеоустройств	00 — протокол не определен

Дескрипторы

В USB принята иерархия дескрипторов, описывающих все свойства устройств. Стандартные дескрипторы USB начинаются с байта длины дескриптора, за которым следует байт, определяющий тип дескриптора.

- *дескриптор устройства*, *Device Descriptor* (тип 1, табл. 13.2), описывающий устройство в целом (версия USB, класс, производитель, модель, протокол, число возможных конфигураций). Для HS-устройств общее описание дополняется *дескриптором-квалификатором*, *Device Qualfier Descriptor* (тип 6, табл. 13.3), в котором указывается, сколько у устройства будет конфигураций при работе на иной скорости (дескриптор устройства относится к той скорости, на которой устройство работает в данный момент);
- *дескрипторы конфигурации*, *Configuration Descriptor*, (тип 2, табл. 13.4), описывающие число интерфейсов, атрибуты (способ питания и возможность генерации удаленного пробуждения) и мощность, потребляемую от шины в каждой конфигурации для текущей скорости. Для HS-устройств имеются и *дескрипторы конфигураций для иной скорости*, *Other Speed Configuration Descriptor* (тип 7, табл. 13.4), описывающие те же параметры с тем же форматом;
- *дескрипторы интерфейсов*, *Interface Descriptor* (тип 4, табл. 13.5), для каждого из интерфейсов, доступных в указанной конфигурации, сообщает число прикладных конечных точек, класс и протокол интерфейса. В конфигурации может быть несколько *альтернативных вариантов* (alternate settings) данного интерфейса, каждому из которых соответствует свой дескриптор интерфейса. *Первичным интерфейсом* (primary interface) называют нулевой альтернативный вариант (alternate settings = 0);

- *дескрипторы конечных точек*, *Endpoint Descriptor* (тип 5, табл. 13.6) определяют номер и направление точки, атрибуты (тип передач), максимальный размер поля данных и интервал обслуживания (для точек периодических передач);
- *строковые дескрипторы*, *String Descriptor* (тип 3, табл. 13.7) — необязательные текстовые строки информации, которые могут отображаться хостом. Ссылки (однобайтные индексы) на строковые дескрипторы имеются в дескрипторах устройства, конфигураций и интерфейсов. Нулевое значение индекса означает отсутствие строкового дескриптора для данной структуры. Строки состоят из 2-байтных символов UNICODE, явного терминатора строки нет — конец определяется по длине, заявленной в заголовке дескриптора. Строковые дескрипторы могут присутствовать в устройствах на разных языках; для выбора нужного дескриптора в запросе кроме индекса указывается и 16-битный идентификатор языка (`LanguageID`). Строковый дескриптор, вызываемый по нулевому индексу (с любым кодом языка), в своем теле содержит список поддерживаемых идентификаторов языка. Значения `LanguageID` для некоторых языков:
  - русский — 0419h;
  - английский (США) — 0409h;
  - английский (Великобритания) — 0809h;
  - белорусский — 0423h;
  - украинский — 0422h;
- *специфические дескрипторы*, *Class-Specific Descriptor*, могут использоваться в устройствах определенных классов. Так, например, для хабов имеется дескриптор интерфейса питания, *Interface Power* (тип 8). Специфические дескрипторы также должны начинаться с поля длины и типа.

Дескрипторы от устройств хост получает по запросам *Get_Descriptor*, указав тип дескриптора. Таким способом можно явно запросить дескриптор устройства (и квалификатор), дескриптор конфигурации (для текущей и иной скорости) и дескриптор строки (а также дескриптор OTG, см. главу 15). Дескрипторы интерфейсов и конечных точек по отдельности не адресуются, они пристраиваются «хвостом» к дескриптору конфигурации. Возможность чтения всех имеющихся дескрипторов обязательна для всех устройств, дополнительно может поддерживаться и запись дескрипторов (запросом *Set_Descriptor*). Положительный ответ устройства на транзакцию записи дескриптора означает, что он принят и устройство будет подчиниться его свойствам.

По запросу *дескриптора конфигурации* устройство выдает целую цепочку дескрипторов, начинающуюся с собственно дескриптора конфигурации. За ней для каждого доступного интерфейса следует дескриптор первичного интерфейса и его конечных точек, за которым следуют все альтернативные варианты со своими конечными точками. Общая длина всего описания конфигурации заранее не известна, она указывается в поле `wTotalLength` дескриптора конфигурации. Так что для получения конфигурации сначала выполняют запрос, указав длину собственно дескриптора конфигурации (9, хотя достаточно и 4), а потом его повторяют с длиной, прочитанной из данного поля.

В приведенных ниже таблицах и описаниях перед мнемоническими названиями полей префикс b означает байт, w — двухбайтное слово, bm — битовую карту, i — целое число (однобайтный индекс), bcd — неупакованный BCD-формат, id — идентификатор (не число). Отметим, что в пакетах слово передается младшим байтом вперед, так что обращение к слову, размещенному в памяти хоста, будет естественным для процессоров x86 (младший байт по меньшему адресу) и не требует перестановки байтов.

Таблица 13.2. Дескриптор устройства

Смещение	Поле	Длина	Содержание
0	`bLength`	1	Длина дескриптора (18)
1	`bDescriptorType`	1	Тип дескриптора (1)
2	`bcdUSB`	2	Версия спецификации USB в BCD-формате (0200h означает USB 2.0)
4	`bDeviceClass`	1	Код класса устройства
5	`bDeviceSubClass`	1	Код подкласса устройства
6	`bDeviceProtocol`	1	Код протокола устройства
7	`bMaxPacketSize0`	1	Максимальный размер пакета для *EP0* (8, 16, 32 или 64)
8	`idVendor`	2	Идентификатор производителя (Vendor ID)
10	`idProduct`	2	Идентификатор продукта (Product ID)
12	`bcdDevice`	2	Версия устройства в BCD-формате
14	`iManufacturer`	1	Индекс строкового дескриптора производителя
15	`iProduct`	1	Индекс строкового дескриптора продукта
16	`iSerialNumber`	1	Индекс строкового дескриптора серийного номера
17	`bNumConfigurations`	1	Число возможных конфигураций устройства на данной скорости

Таблица 13.3. Дескриптор-квалификатор устройства

Смещение	Поле	Длина	Содержание
0	`bLength`	1	Длина дескриптора (10)
1	`bDescriptorType`	1	Тип дескриптора (6)
2	`bcdUSB`	2	Версия спецификации USB в BCD-формате (0200h означает USB 2.0)
4	`bDeviceClass`	1	Код класса устройства
5	`bDeviceSubClass`	1	Код подкласса устройства
6	`bDeviceProtocol`	1	Код протокола устройства

Смещение	Поле	Длина	Содержание
7	bMaxPacketSize0	1	Максимальный размер пакета для *EP0* на иной скорости
8	bNumConfigurations	1	Число возможных конфигураций устройства на иной скорости
9	bReserved	1	Резерв (0)

Таблица 13.4. Дескрипторы конфигурации

Смещение	Поле	Длина	Содержание
0	bLength	1	Размер данного дескриптора (9)
1	bDescriptorType	1	Тип дескриптора (2 или 7)
2	wTotalLength	2	Общее число байтов полного описания данной конфигурации
4	bNumInterfaces	1	Число поддерживаемых интерфейсов
5	bConfigurationValue	1	Номер конфигурации
6	iConfiguration	1	Индекс строкового дескриптора конфигурации
7	bmAttributes	1	Атрибуты (характеристики конфигурации: D7: резерв (1) D6: Self-powered — возможность автономного питания D5: Remote Wakeup — способность генерации удаленного пробуждения D4...0: резерв (0)
8	bMaxPower	1	Максимальный ток потребления от шины в рабочем режиме в единицах по 2 мА

Таблица 13.5. Дескриптор интерфейса

Смещение	Поле	Длина	Содержание
0	bLength	1	Размер данного дескриптора (9)
1	bDescriptorType	1	Тип дескриптора (4)
2	bInterfaceNumber	1	Номер интерфейса
3	bAlternateSetting	1	Номер альтернативной установки
4	bNumEndpoints	1	Число конечных точек, используемых интерфейсом (исключая *EP0*)
5	bInterfaceClass	1	Класс интерфейса
6	bInterfaceSubClass	1	Подкласс интерфейса
7	bInterfaceProtocol	1	Код протокола
8	iInterface	1	Индекс строкового дескриптора интерфейса

Таблица 13.6. Дескриптор конечной точки

Смещение	Поле	Длина	Содержание
0	`bLength`	1	Размер дескриптора (7)
1	`bDescriptorType`	1	Тип дескриптора (5)
2	`bEndpointAddress`	1	Адрес точки: Биты [3:0] — номер точки; Биты[6:4] — резерв (0); Бит 7 — направление (игнорируется для *EP0*): 0 = *OUT*, 1 = *IN*
3	`bmAttributes`	1	Атрибуты точки: Биты [1:0] — тип точки: 00 = *Control*, 01 = *Isochronous*, 10 = *Bulk*, 11 = *Interrupt*; Биты [5:2] используются только для изохронных точек, для остальных резерв (0): Биты [3:2] — тип синхронизации: 00 = нет синхронизации, 01 = асинхронная, 10 = адаптивная, 11 = синхронная; Биты [5:4] — тип использования: 00 = данные, 01 = обратная связь, 10 = данные с предоставлением неявной обратной связи, 11 = резерв
4	`wMaxPacketSize`	2	Биты [10:0] — максимальный размер пакета; Биты [12:11] — число дополнительных транзакций в микрокадре, только для широкополосных точек, для остальных — резерв (0); Биты [15:13] — резерв (0)
6	`bInterval`	1	Для точек периодических транзакций — интервал обслуживания (см. главу 11), для HS-точек непериодического вывода — допустимая частота посылки NAK (см. главу 10)

Таблица 13.7. Строковый дескриптор

Смещение	Поле	Длина	Содержание
0	`bLength`	1	Размер дескриптора (N+2)
1	`bDescriptorType`	1	Тип дескриптора (3)
2	`bString`	N	Строка байтов

Запросы к устройствам USB (управляющие передачи)

Запросы к устройству выполняются с помощью транзакций управления, адресованных к его конечным точкам типа *Control*. Спецификация определяет формат пакета на стадии установки, позволяющий конструировать разнообразные запросы. Запросы могут адресоваться к устройству в целом, его интерфейсу, конкрет-

ной конечной точке и иному фрагменту устройства. Формат пакета позволяет определить наличие, направление и длину посылки на стадии данных; трактовка этих данных зависит от запроса. Набор запросов включает:

- стандартные запросы для всех устройств. Список запросов и форматы данных определены спецификацией USB;
- запросы для класса. Список запросов и форматы данных определены стандартом для данного класса устройств;
- специфичные запросы, определенные разработчиком конкретного устройства.

Все устройства USB имеют *основной канал управления* через точку *EP0*, к которой выполняются основные стандартизованные запросы. В устройствах могут быть и другие точки с типом *Control*, для них будут свои специфические запросы, но формат стадии установки у них должен быть таким же, как и для *EP0*, чтобы продолжение управляющей транзакции определялось из него тем же стандартным способом. Однако дополнительные точки типа *Control* в устройствах встречаются не часто, что объясняется сложностью их реализации. В большинстве случаев вполне можно обходиться управлением через *EP0*, используя «фирменные» запросы (*Vendor request*).

Устройство должно отвергать запрос, если он не поддерживается или имеет неправильные параметры. Время исполнения запроса (от подачи команды до подтверждения или отвержения) спецификацией ограничивается: общее ограничение — до 5 секунд, но запрос установки адреса должен исполняться не дольше 50 мс (иначе процесс нумерации устройств был бы слишком затяжным).

На *стадии установки* (*Setup Stage*) в 8-байтном поле данных устройству передается собственно запрос (табл. 13.8), в котором указывается и направление, в котором будет передача на стадии передачи данных. Запрос идентифицируется обязательными полями `bmRequestType` и `bRequest`; содержимое полей `wValue`, `wIndex` и `wLength` используется не во всех запросах, неиспользуемые поля должны быть нулевыми. *Стадия данных* (*Data Stage*) используется не во всех запросах (когда ее нет, `wLength` = 0). На *стадии состояния* (*Status Stage*) устройство подтверждает успешное выполнение запроса (пакетом *ACK*) или отвергает его (пакетом *STALL*). До тех пор пока устройство не завершило отработку запроса (или не отвергло его), оно отвечает пакетами *NAK*. Получение пакета *STALL* от управляющей конечной точки является нормальным ответом, не требующим разблокирования данной точки.

Таблица 13.8. Информация, передаваемая на стадии установки

Смещение	Поле	Длина	Содержание
0	`bmRequestType`	1	Характеристики запроса: D7: направление передачи данных: 0 = от хоста к устройству, 1 = от устройства к хосту; D6...5: Тип 0 = стандартный, 1 = для класса, 2 = специфичный, 3 = зарезервирован D4...0: Получатель: 0 = Устройство, 1 = Интерфейс, 2 = Точка, 3 = Другой, 4...31 = зарезервировано

продолжение

Таблица 13.8 (*продолжение*)

Смещение	Поле	Длина	Содержание
1	`bRequest`	1	Запрос
2	`wValue`	2	Параметр запроса
4	`wIndex`	2	Индекс или смещение (использование определяется запросом)
6	`wLength`	2	Число байт, передаваемых в фазе данных

Стандартные запросы к устройствам

Стандартные запросы относятся ко всем устройствам USB, хотя для ряда устройств есть исключения: управление альтернативными установками интерфейсов не требуется, если нет альтернатив; установка меток времени нужна (и возможна) только для устройств с изохронными точками. Стандартные запросы адресуются к *EP0*, признак стандартного запроса — в поле типа `bmRequestType` D[6:5] = 0. Типы и коды стандартных запросов приведены в табл. 13.9.

Таблица 13.9. Стандартные запросы к устройствам

Запрос	bmRequestType	bRequest
Clear_Feature	00000000b 00000001b 00000010b	1
Get_Configuration	10000000b	8
Get_Descriptor	10000000b	6
Get_Interface	10000001b	10
Get_Status	10000000b 10000001b 10000010b	0
Set_Address	00000000b	5
Set_Configuration	00000000b	9
Set_Descriptor	00000000b	7
Set_Feature	00000000b 00000001b 00000010b	3
Set_Interface	00000001b	11
Synch_Frame	10000010b	12

Запрос установки адреса Set_Address адресуется только ко всему устройству, в поле `wValue` передается адрес, назначаемый устройству.

Запросы обращения к дескрипторам Get_Descriptor и *Set_Descriptor* адресуются только ко всему устройству. Здесь поле `wValue` в старшем байте содержит тип дескрип-

тора (1, 2, 3, 6 или 7 для *Get* и только 1, 2 или 3 для *Set*), в младшем — индекс строки (для дескрипторов типа 3) или номер конфигурации (для дескрипторов типа 2 или 7). Поле `wIndex` используется только для строковых дескрипторов для задания языка (`Languge ID`). Поле `wLength` задает длину дескриптора. Если реальная длина считываемого дескриптора больше запрошенной, то считывается только его начало; если меньше, то устройство в транзакции возвращает только реальное количество байтов.

Запросы управления конфигурацией Get_Configuration и *Set_Configuration* также адресуются только к устройству. В запросе установки (*Set*) используется только поле `wValue` — в его младшем байте передается номер устанавливаемой конфигурации. В запросе чтения (*Get*) используется только поле `wLength` (= 1) — ожидается один байт ответа, содержащий номер текущей конфигурации.

Запросы управления альтернативными установками Get_Interface и *Set_Interface* адресуются к интерфейсу, номер которого указывается в поле `wIndex`. В запросе установки (*Set*) в младшем байте поля `wValue` передается номер альтернативной установки. В запросе чтения (*Get*) поле `wLength` (= 1) указывает на ожидание одного байта ответа, содержащего номер текущей альтернативной установки для данного интерфейса.

Запрос установки метки времени Synch_Frame, адресуемый к устройству, в поле `wIndex` содержит номер точки, к которой относится данная метка. Поле `wLength` (= 2) указывает на 2 байта передаваемых данных — номера кадра для данной метки.

Запрос чтения состояния Get_Status может адресоваться к устройству, интерфейсу или конечной точке. Здесь поле `wIndex` определяет номер объекта (интерфейса или точки, для устройства — ноль), поле `wLength` указывает число байтов ожидаемых данных состояния. Трактовка данных состояния зависит от адресата:

- в стандартном запросе состояния устройства (`wLength` = 2) определено значение лишь младших бит слова: D0 (`Self Powered`) — признак автономности питания (0 — питается от шины); D1 (`Remote Wakeup`) — возможность подачи сигнала удаленного пробуждения (0 — нет); D2 (`Port Test`): 1 — порт находится в режиме тестирования;
- чтение состояния интерфейса в стандартном запросе информации не дает (возвращает нули). Однако он может использоваться в запросе для класса; так, например, для принтеров этот запрос (при `wLength` = 1) возвращает байт состояния, аналогичный состоянию LPT-порта (принтер выбран, ошибка, конец бумаги);
- в стандартном запросе состояния конечной точки (`wLength` = 2) определено значение лишь младшего бита слова: D0 (`Halt`) — признак остановки конечной точки (на транзакции с ней устройство отвечает пакетом *STALL*).

Запросы управления свойствами Set_Feature (установить свойство) и *Clear_Feature* (сбросить свойство) также могут адресоваться к устройству, интерфейсу или конечной точке. Здесь поле `wIndex` определяет номер объекта (интерфейса или точки, для устройства — ноль), поле `wValue` задает номер свойства. Набор стандартных управляемых свойств невелик:

- управление возможностью подачи удаленного пробуждения (`Device_Remote_Wakeup`), адресат — устройство, `wValue` = 1;
- управление остановкой (блокировкой) конечной точки (`Endpoint_Halt`), адресат — конечная точка, `wValue` = 0. Остановленная (заблокированная) конечная точка будет на все транзакции отвечать пакетом *STALL*. Сброс признака останова разблокирует и инициализирует точку, включая начальную установку переключателя (`Toggle Bit`);
- управление тестовым режимом приемопередатчиков (`Test_Mode`), адресат — устройство, `wValue` = 2. Здесь используется и поле `wIndex`, определяющее выполняемый тест: 01 — *Test_J*, 02 — *Test_K*, 03 — *Test_SE0_Nack*, 04 — *Test_Packet*, 05 — *Test_Force_Enable*. Значения 06–3Fh зарезервированы для стандартных тестов, C0–FFh отданы разработчикам устройств. Данным запросом можно только включить тест; для выключения теста устройство приходится выключать и включать питание устройства, поскольку управляющие запросы оно уже не воспринимает.

ГЛАВА 14

Хабы USB

Хаб является ключевым элементом системы PnP в архитектуре USB. Хаб выполняет множество функций, в частности:

- обеспечивает физическое подключение устройств, формируя и воспринимая сигналы в соответствии со спецификацией шины на каждом из своих портов и транслируя трафик с восходящего порта на нисходящие и наоборот;
- обеспечивает управляемую информационную связь сегментов шины, включая и связь сегментов, работающих на разных скоростях. Каждому нисходящему порту может быть селективно разрешена или запрещена трансляция трафика;
- отслеживает состояние подключенных к нему устройств, уведомляя хост об изменениях — подключении и отключении устройств;
- обнаруживает ошибки на шине, выполняет процедуры восстановления и изолирует неисправные сегменты шины. Благодаря бдительности хабов неисправное устройство не может заблокировать всю шину;
- управляет энергопотреблением: подает питающее напряжение на нисходящие порты; селективно генерирует сигнал приостановки портов, транслирует эти сигналы в разных направлениях (см. главу 12).

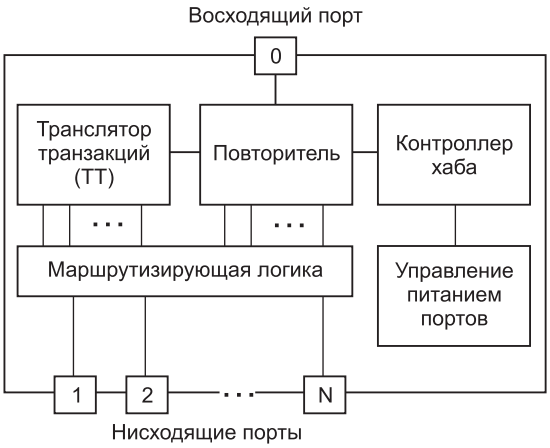


Рис. 14.1. Структура хаба USB 2.0

Структура хаба USB 2.0 приведена на рис. 14.1. Хаб состоит из набора портов, контроллера хаба (устройство-функция USB, подключенная к внутреннему порту), повторителя, транслятора транзакций, маршрутизирующей логики портов и цепей управления подачей питания. Хаб USB 1.x проще: в нем отсутствует транслятор транзакций и логика маршрутизации нисходящих портов — они все подключаются к повторителю.

Порты

Со стороны *восходящего порта* хаб выглядит как и любое устройство USB (см. правую часть рис. 14.2). Этот порт всегда включен и разрешен, для хабов USB 1.x он всегда полноскоростной, для USB 2.0 восходящий порт всегда высокоскоростной, хотя может работать и в полноскоростном режиме.

Нисходящие порты хаба имеют комплект приемопередатчиков, изображенный в левой части того же рисунка. Хост управляет нисходящими портами и определяет их состояние, посылая запросы к контроллеру хаба. Каждый из этих портов может определить, подключено ли к нему устройство и на какой скорости оно работает. Порт может быть селективно *разрешен* (*enabled*) или *запрещен* (*disabled*) по команде от хоста; запрещение может быть и аппаратным. Аппаратно порт запрещается по событию подключения-отключения, а также по ошибке, обнаруженной хабом. Хаб игнорирует сигналы от запрещенных портов и не транслирует на них трафик. На порт может быть подана *команда сброса*, инициирующая соответствующую сигнализацию и уточнение типа устройства (проверяется признак HS-устройства). Также селективно любой порт может быть *приостановлен* (*suspended*), после чего для него может быть подана команда *возобновления* (*resume*) с соответствующей сигнализацией. В плане подачи питания порт может быть *запитан* (*powered*) или нет. Управление подачей питания может быть как селективным, так и общим для всех портов. Порт может оказаться не запитанным по причине срабатывания токовой защиты, причем защита тоже может быть как селективной, так и общей. В последнем случае порт может оказаться не запитанным и из-за перегрузки другого нисходящего порта. Для HS-порта возможна еще и подача команды тестирования.

Каждый нисходящий порт может находиться в одном из нижеперечисленных *состояний*, наблюдаемых и управляемых хостом с помощью запросов к хабу:

- *не запитан* (*Not powered*) — на порт не подается питание по запросу *Clear_Port_Power* от хоста или из-за аварии питания (срабатывание токовой защиты или потеря внешнего питания). Не запитанный порт не пригоден ни к каким интерфейсным взаимодействиям. Только после подачи питания он может распознать подключение устройства и с ним взаимодействовать. Питание включается запросом *Set_Port_Power*;
- *не подключен* (*Disconnected*) — порт способен только к обнаружению подключения устройства. В это состояние порт переходит из любого последующего по обнаружению отключения устройства;

- *запрещен* (*Disabled*) — устройство подключено, но трафик и сигналы возобновления не транслируются. В это состояние порт переходит из любого последующего по запросу *Port_Disable*, по сигналу сброса на восходящем порте, а также при обнаружении хабом серьезной ошибки, требующей изоляции данного порта;
- *разрешен* (*Enabled*) — устройство подключено и с ним возможен полноценный обмен данными и сигналами. В это состояние из любого другого (запитанного) порт переводится сбросом (запросом *Port_Reset*); кроме того, из состояния *запрещен* — запросом *Set_Port_Enable*, из состояния *приостановлен* — запросом *Clear_Port_Suspend*;
- *приостановлен* (*Suspended*) — порт подает сигнал приостановки, трафик не транслируется, от порта воспринимается только сигнал возобновления и отключения устройства. В это состояние порт переходит по запросу *Set_Port_Suspend*; вернуться в состояние *разрешен* можно по сигналу удаленного пробуждения, запросу *Clear_Port_Suspend* или запросу *Port_Reset*.

Переходы из одного состояния в другое инициируются сигналами от устройств (подключение-отключение, удаленное пробуждение), управляющими запросами хоста и хабом (обнаружение серьезных ошибок).

Хабы могут иметь световые *индикаторы состояния нисходящих портов* (пару светодиодов или один двухцветный), управляемые аппаратно (логикой хаба) или программно (хост-контроллером):

- не светится — порт не используется;
- зеленый — нормальная работа;
- желтый — ошибка подключенного устройства или перегрузка порта (порт автоматически отключен);
- зеленый мигающий — программа требует внимания пользователя (*Software attention*);
- желтый мигающий — аппаратура требует внимания пользователя (*Hardware attention*), например, мощное устройство подключено к маломощному порту.

Контроллер хаба

Контроллер хаба является программно-видимым устройством USB, взаимодействуя с которым хост управляет конфигурированием устройств и соединениями на шине. Как и всякое устройство USB, контроллер хаба имеет набор дескрипторов, его описывающих (см. главу 13). Для хабов определен специальный класс устройств (класс 09, подкласс 00). Интерфейс хаба (кроме обязательной нулевой контрольной точки) содержит конечную точку типа *Interrupt-IN* для информирования хоста о смене состояния. Управление хабом в целом и его портами выполняется с помощью специальных запросов к точке *EP0*. Их описание в табл. 14.1–14.4 дает полное представление о возможностях управляемости и наблюдаемости хаба.

Повторитель

Повторитель хаба обеспечивает динамические соединения между портами для трансляции пакетов и сигналов возобновления. В состоянии покоя повторителя все порты работают на прием и ожидают признака начала кадра или сигнала возобновления. По этим событиям устанавливается то или иное соединение портов (рис. 14.2).

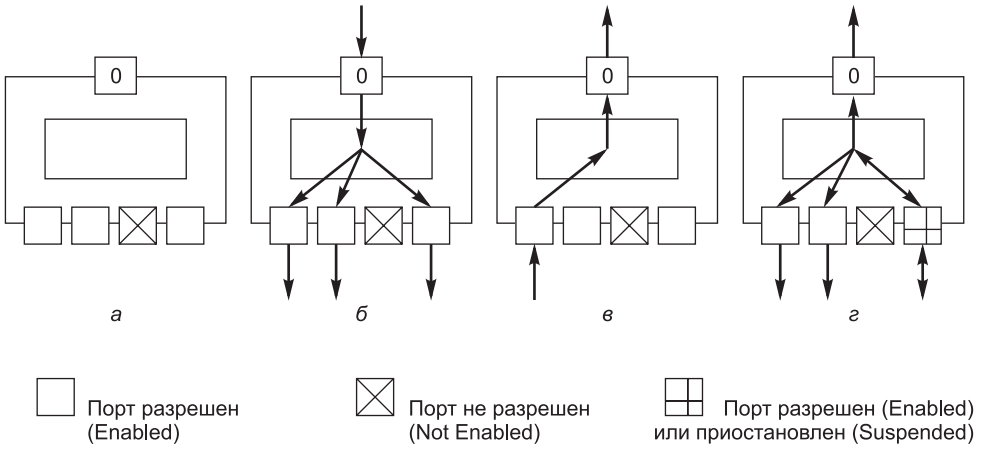


Рис. 14.2. Соединения, обеспечиваемые повторителем хаба: *а* — покой; *б* — трансляция пакета с восходящего порта; *в* — трансляция пакета с нисходящего порта; *г* — трансляция восходящего возобновления от разрешенного порта

В трансляции пакетов участвуют только разрешенные порты (восходящий разрешен всегда). Если какой-либо разрешенный порт обнаружил начало пакета, то устанавливается соединение в соответствии с рис. 14.2 *б* или *в*, и повторитель транслирует пакет, дожидаясь его конца (признака *EOP*). По концу пакета повторитель опять переходит в состояние покоя. Из рисунков видно, что нисходящий трафик транслируется широковещательно. Восходящий трафик транслируется сугубо направленно, так, что его «видят» только хабы, расположенные в цепочке от устройства к хосту, но не другие устройства.

Сигнал возобновления (*resume*) транслируется несколько иначе. С восходящего порта сигнал *resume* транслируется во все нисходящие порты, кроме запрещенных и селективно приостановленных. Сигнал *resume*, обнаруженный на нисходящем разрешенном порте, подхватывается хабом и транслируется в восходящий порт и во все нисходящие порты (включая и порт-источник сигнала), кроме запрещенных и селективно приостановленных. С селективно приостановленного порта сигнал возобновления подхватывается и транслируется обратно только в этот же порт, после чего хаб завершает сигнализацию возобновления (*LS-EOP*) и переводит порт в разрешенное состояние.

Обнаружение и локализация неисправных устройств

Хаб обеспечивает живучесть шины USB, автоматически запрещая работу нисходящих портов, от которых исходит угроза для работоспособности шины. Неисправное устройство, подключенное к порту, 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 микрокадра. Хаб следит за тем, чтобы восходящие пакеты не пересекали границу микрокадра. Для этого в хабе имеется счетчик, который определяет следующие моменты в микрокадрах:

- *EOF1* — после этого момента не имеет права начинаться пакет от устройства;
- *EOF2* — начиная с этой точки, хаб ждет начала пакета только с восходящего порта (ожидается *SOF*).

Во время восходящей трансляции пакета (пока повторитель ожидает *EOP*) возможны особые ситуации:

- обнаружен признак начала пакета на другом разрешенном порте — это *коллизия*; реакция хаба на нее может быть двоякой. Первый вариант — «испортить» трансляцию передачей к хосту вместо тела пакета постоянного состояния *K* или *J*. Хост-контроллер поймет эту ситуацию, правда, ценой потери чужого пакета. Второй вариант — игнорировать этот признак, пока не завершится текущая трансляция пакета. При этом хост о данной проблеме не узнает;
- признак конца пакета в текущей трансляции не обнаружен, а время в микрокадре уже подошло к *EOF2* — эта «болтливость» считается ошибкой для нисходящего порта, с которого идет трансляция. По этой ошибке хаб автоматически запрещает данный порт и фиксирует это изменение состояния в своем регистре. Еще одной причиной автоматического запрещения разрешенного порта является его состояние, отличное от покоя, с момента *EOF2* до конца микрокадра.

Если восходящее соединение устанавливается после *EOF1*, то повторитель вместо трансляции кадра передает признак *FS-EOP*, чтобы вышестоящий хаб не запретил работу порта, к которому подключен данный хаб. Если с нисходящего порта, вызвавшего эту ситуацию, не придет *EOP* до *EOF2*, то этот порт будет автоматически запрещен.

При трансляции пакетов и сигналов *для LS-портов* повторитель выполняет инвертирование сигнала, поскольку представление состояний *J* и *K* для них иное (см. главу 12). Кроме того, нисходящий трафик не транслируется на порты LS, пока не придет специальный пакет *PRE*. После пакета *PRE* «вниз» на LS-порт транслируется только один пакет. Вместо маркеров *SOF* на LS-порт передается признак *LS-EOP*, чтобы устройство не приостановилось.

Для *HS-портов* повторитель устроен функционально сложнее — здесь требуется *ресинхронизация* (re-clocking). Помехи в линии и по питанию влияют на момент

переключения состояния выхода приемника во время нарастания и спада входного сигнала — это приводит к дрожанию фазы (jitter). На невысоких (LS, FS) скоростях дрожание несущественно, поскольку время нарастания/спада сигнала существенно меньше длительности битового интервала. На скорости 480 Мбит/с это соотношение иное, и накопление дрожания в цепочке повторителей (хабов) привело бы к потере битовой синхронизации. Ресинхронизация выполняется с помощью *эластичного буфера* — небольшой FIFO-памяти, хранящей последовательность бит транслируемого сигнала. Информация в буфер — биты, извлеченные из NRZI-сигнала, — поступает от приемника порта на частоте входного сигнала (частота подстраивается по полю `Sync`). Из буфера информация поступает на передатчик(и), но синхронизируясь уже от внутреннего генератора хаба. Поскольку частоты входного сигнала и внутреннего генератора абсолютно точно совпадать не могут, темп заполнения буфера отличается от темпа его опустошения. Для компенсации этих расхождений в начале приема пакета его трансляция задерживается до тех пор, пока буфер не наполнится до половины; за время передачи пакета наполненность буфера может измениться. Эластичный буфер вносит свою лепту в задержку, вносимую хабом. Глубина буфера определяется исходя из допустимого отклонения скоростей: по спецификации частота синхронизации должна выдерживаться с точностью[1] ± 0,05%, так что максимальное расхождение частот приема и передачи может достигать ± 0,1%. При этом на пакете максимальной длины (9644 бит) может «набегать» расхождение до 9644×0,1% ≈ 10 бит; с небольшим запасом половинную длину буфера определили в 12 бит. Аналогичная технология применяется в высокоскоростных локальных сетях (Fast Ethernet, FDDI).

HS-пакет, проходя через повторитель хаба, может не только потерять до 4 бит начала синхропоследовательности, но и «обрасти» дополнительным «хвостом» после признака *EOP*, также до 4 бит длиной (dribble bits). Это «обрастание», как и начальная потеря, обусловлено задержкой, вносимой детектором амплитуды сигнала HS-приемника. При прохождении через 5 хабов число лишних бит может достигать 20, но это не страшно — приемник информации все равно верно определит конец пакета по признаку *EOP*. Вышеуказанная максимальная длина пакета 9644 бит включает в себя эти лишние биты.

Транслятор транзакций

Транслятор транзакций, входящий в хаб USB 2.0, служит для преобразования скоростей обмена по шине: высокой (HS) на стороне восходящего порта в полную или низкую (FS/LS) на стороне нисходящих портов, к которым подключены устройства USB 1.x. Транслятор выполняет расщепленные транзакции ввода/вывода и транслирует маркеры микрокадров в маркеры кадров, передаваемые в порты FS.

---

[1] В зарубежных источниках принято отклонения выражать в ppm (parts per million — миллионные доли); 0,05% соответствует 50 ppm.

Расщепление транзакций организуется хостом, который знает текущую топологию шины (к портам каких хабов USB 2.0 подключены устройства или хабы 1.x). Расщепленные транзакции выполняются в два-три этапа, в зависимости от типа и направления передачи:

- между хостом и транслятором выполняется специальная транзакция *Start Split* (*SS*, расщепленный запуск). Она несет всю информацию, необходимую для запуска транзакции с целевым устройством. На этом этапе транслятор играет роль специфически адресуемого устройства USB;
- между транслятором и целевым устройством (хабом) USB 1.0 выполняется обычная транзакция, в которой транслятор играет роль хост-контроллера;
- между хостом и транслятором выполняется специальная транзакция *Complete Split* (*CS*, расщепленное завершение), которая доносит хосту результаты выполнения транзакции хаба с целевым устройством. Здесь транслятор также играет роль специфически адресуемого устройства USB. Для изохронного вывода этот этап отсутствует, поскольку никакого ответа от целевого устройства не предусмотрено.

Во всех этих транзакциях используется обычная «молчаливая» реакция на прием поврежденных пакетов и механизм тайм-аутов.

Каждый нулевой маркер микрокадра хаб транслирует в виде полноскоростного маркера кадра *SOF*. Расщепленные транзакции на стороне FS/LS выполняются внутри этих кадров. Старт расщепленных транзакций хост планирует на нулевой микрокадр, чтобы к концу последнего (седьмого) микрокадра расщепленная транзакция могла быть завершена и все данные оказались бы переданы (чтобы не накапливать переходящих остатков). Временные соотношения между транзакциями на обеих сторонах транслятора иллюстрирует рис. 14.3, где в качестве примера рассмотрено расщепление транзакции ввода.

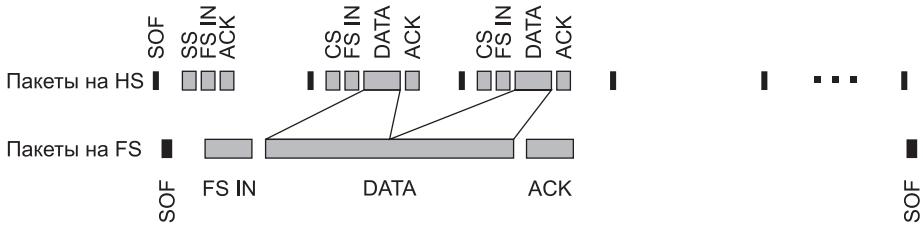


Рис. 14.3. Трансляция транзакций

Общая структура транслятора транзакций приведена на рис. 14.4. *HS-обработчик* (HS Handler) помещает информацию расщепленных запусков в свои буферы, а по запросам завершений выбирает из буферов результаты и передает их в виде пакетов хосту. Этот обработчик отрабатывает все обычные функции протокола USB, включая генерацию и проверку CRC, посылку хосту подтверждений и т. п. *F/LS-обработчик* (F/LS-Handler) по выбранным из буферов запросам запусков формиру-

ет обычные транзакции USB, начинающиеся с маркеров *IN*, *OUT*, *SETUP* (а для LS-портов и с преамбулы *PRE*). Результаты этих транзакций (данные и подтверждения) он помещает в буферы. Транслятор транзакций обладает буферами, в которых размещаются все необходимые данные о текущих выполняемых расщепленных транзакциях. По завершении транзакции (на обеих сторонах) буфер освобождается для обслуживания следующих расщепленных транзакций.

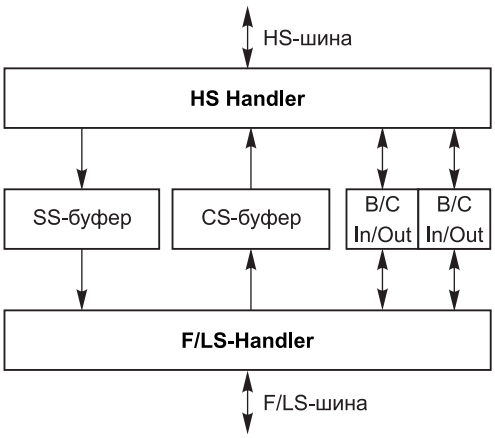


Рис. 14.4. Структура транслятора транзакций

Периодические транзакции (изохронные и прерывания), критичные к времени выполнения, транслятором обрабатываются по конвейерной схеме. Данные запусков периодических транзакций помещаются в *SS-буфер*, из которого они выбираются F/LS-обработчиком (буфер реализует дисциплину FIFO) и запускаются в виде транзакций на вторичную шину. Результаты этих транзакций помещаются в *CS-буфер* (тоже FIFO), откуда их извлекает хост. За порядок наполнения и опустошения этих буферов отвечает хост — он планирует транзакции *SS* и *CS*. Транзакции запусков проходят без подтверждений со стороны транслятора; хост узнает о судьбе транзакции только в фазе завершения.

Непериодические транзакции (управление и передача массивов) обрабатываются иначе: у транслятора имеется два или более буфера *B/C In/Out*, каждый из которых обслуживает по одной транзакции, от начала и до конца. Здесь транзакции запусков подтверждаются транслятором: ответ *NAK* означает невозможность приема транзакции к исполнению (нет свободных буферов), то есть хосту следует повторить попытку запуска. Ответ *ACK* означает, что транзакция принята к исполнению и через некоторое время следует забрать результат, выполнив транзакцию завершения.

Спецификация предусматривает два варианта реализации транслятора; какой именно применяется, можно узнать из кода протокола интерфейса хаба:

- по одному транслятору на каждый нисходящий порт (код протокола — 2); это самый производительный вариант для умножения пропускной способности шины для устройств FS/LS;

- один транслятор на все порты (код протокола — 1); здесь возможности умножения пропускной способности зависят от объема буферов периодических транзакций и количества буферов для непериодических.

В расщепленных транзакциях фаза маркера, определяющая продолжение транзакции, состоит из двух пакетов-маркеров: специального маркера *SPLIT*, формат которого приведен на рис. 14.5, за которым следует маркер обычной транзакции (*IN*, *OUT*, *SETUP*). Пакет-преамбула *PRE* в расщепленных транзакциях не используется — скорость целевого устройства задается в маркере *SPLIT*, что позволяет хабу провести транзакцию на нужной скорости. Транслятор сам генерирует преамбулу, если вторичный порт, через который обращаются к LS-устройству, опознал FS-подключение (это означает, что между хабом, транслирующим транзакцию, и целевым устройством есть хаб USB 1.x). Маркер *SPLIT* адресуется к порту хаба (номера в полях `HubAddr` и `PortAddr`), а следующий за ним обычный маркер адресует конечную точку целевого устройства. Маркеры *SPLIT*, используемые для запуска (*SS*) и завершения (*CS*) расщепленных транзакций, различаются полем `SC`: 0 — для *SS*, 1 — для *CS*. Поле `ET` описывает тип целевой конечной точки, с которой будет производиться транзакция (00 — *Control*, 01 — *Isochronous*, 10 — *Bulk*, 11 — *Interrupt*). Поля `S` и `E` трактуются по-разному. Для управляющих транзакций и прерываний поле `S` определяет скорость (0 — FS, 1 — LS), `E` = 0. Для остальных транзакций, кроме запуска изохронного вывода, поля `S` и `E` содержат нули. В транзакциях запуска изохронного вывода поля `S` и `E` трактуются как признаки начала и конца данных соответственно (см. далее).

Поле	Sync	PID	Check	Hub Addr	SC	Port Addr	S	E	ET	CRC	EOP
Длина, бит	32	4	4	7	1	7	1	1	2	5	8

Рис. 14.5. Формат маркеров SPLIT

Расщепление периодических транзакций

Периодические транзакции (изохронные и прерывания) критичны к времени выполнения, что накладывает отпечаток на их исполнение в расщепленном виде.

Изохронные транзакции на FS синхронизируются с кадрами (1 мс), в то время как на HS — с микрокадрами (125 мкс). Поскольку за время микрокадра на полной скорости может быть передано всего 187,5 байт данных, при расщеплении транзакций на стороне FS можно без ущерба равномерности уменьшить размер передаваемого пакета за один микрокадр (ради облегчения планирования загрузки микрокадров). Исходя из этого, одна транзакция FS, несущая до 1023 байт данных, фрагментируется — разбивается на 1–6 транзакций HS, несущих до 188 байт данных каждая.

Проще всего расщепляются *транзакции изохронного вывода*, поскольку здесь не требуется получения от устройства ответа или данных. Одна FS-транзакция изо-

хронного вывода реализуется в виде цепочки из 1–6 HS-транзакций, в каждой из которых после маркера *SS* посылается маркер *OUT*, адресующийся к конечной точке целевого устройства, и пакет *DATA0* с очередным фрагментом данных. Здесь (и только здесь) в маркере *SS* поля S (Start) и E (End) определяют местоположение пакета данных в полноскоростной транзакции: [S:E] = 10 — стартовый, [S:E] = = 01 — последний, [S:E] = 00 — промежуточный, [S:E] = 11 — в пакете все данные транзакции.

Транслятор транзакций, успешно приняв стартовый пакет (с единичным битом S), может начинать транзакцию вывода, полагая, что последующие данные будут хостом доставлены своевременно. Отработав пакет с признаком последнего фрагмента, транслятор формирует нормальное окончание пакета (CRC код и EOP). Если промежуточные фрагменты приходят с ошибкой, транслятору придется «испортить» выводимый пакет, введя ошибку вставки бит. Таким образом, целевой приемник данные этой транзакции проигнорирует, как некорректные. Транслятор транзакций после приема фрагмента с ошибкой будет игнорировать все расщепленные транзакции, адресованные к данной конечной точке, у которых нет признака начального фрагмента. Последующую транзакцию со стартовым фрагментом транслятор будет отрабатывать как новую.

Транзакции изохронного ввода расщепляются несколько сложнее, поскольку требуют передачи к хосту данных, которые от целевого устройства будут получены с задержкой. Одна FS-транзакция изохронного ввода реализуется в виде цепочки, состоящей из транзакции запуска, содержащей маркер *SS* и маркер *IN*, и нескольких транзакций завершения, в каждой из которых за маркером *CS* следует тот же маркер *IN* и ожидается от хаба пакет с очередным фрагментом данных или пакет квитирования. Здесь также данные одной FS-транзакции разбиваются на 1–6 фрагментов (HS-транзакций). Данные всех фрагментов, кроме последнего, возвращаются пакетами *MDATA*, вынуждающими повторить транзакцию завершения в следующем микрокадре. Последний фрагмент приходит в пакете *DATA0*. Вместо пакета данных контроллер может ответить пакетом *NYET*, если к концу микрокадра он принял от целевого устройства менее трех байт данных. Хост в этом случае повторит транзакцию завершения в следующем микрокадре, если текущий микрокадр не является последним в кадре. Ответ *NYET* в последнем микрокадре кадра трактуется хостом как ошибка, по которой транзакцию следует завершить и сообщить об этом клиентскому драйверу. Если транслятор транзакций принимает от целевого устройства данные с ошибкой, он ответит пакетом *ERR*, что тоже вынудит хост прекратить транзакцию с сообщением об ошибке.

Транзакции прерываний имеют небольшой размер пакета (до 8 байт данных на LS и до 64 на FS), поэтому для них не нужна фрагментация вывода, а их выполнение укладывается в 1–2 микрокадра.

Транзакция запуска *вывода по прерыванию* (*Interrupt OUT*) состоит из последовательности маркеров *SS* и *OUT*, за которыми следует пакет данных *DATA0* или *DATA1*. Транзакция завершения состоит из последовательности маркеров *CS* и *OUT*, на которую транслятор отвечает пакетом подтверждения:

- пакеты *ACK*, *NAK* и *STALL* — это обычные ответы целевой конечной точки;

- *NYET* — признак незавершенности транзакции транслятором; при этом хост должен повторить транзакцию завершения в следующем микрокадре. Если этот ответ получен в седьмом (последнем) микрокадре, данная ситуация считается ошибкой и обрабатывается обычным образом;
- *ERR* — сигнализация об ошибке фазы подтверждения на вторичной шине.

Транзакция запуска *ввода по прерыванию* (*Interrupt-IN*) состоит из последовательности маркеров *SS* и *IN*. Данные от устройства хост получит в пакетах 1–2 транзакций завершения. Транзакция завершения состоит из последовательности маркеров *CS* и *IN*, на которую транслятор отвечает пакетом данных или подтверждения (*NAK*, *NYET*, *STALL* или *ERR* с вышеописанными значениями). Если транслятор принял еще не все данные от целевого устройства, они придут в пакете *MDATA* (если принято менее трех байт, транслятор пошлет *NYET*), на что хост должен повторить транзакцию завершения в следующем микрокадре. Последняя порция данных придет в пакете *DATA0* или *DATA1*.

Расщепление непериодических транзакций

Непериодические транзакции (управление и передача массивов) не критичны к времени выполнения; малый размер блока данных (до 64 байт) позволяет их расщеплять, используя по одной транзакции запуска и завершения.

Транзакции запуска *вывода* состоит из последовательности маркеров *SS* и *OUT* для массивов (*Bulk OUT*) или *SS* и *SETUP* для управления (*Control*), за которыми следует пакет данных *DATA0* или *DATA1*. Транслятор подтверждает запуск ответом *ACK*, после которого хост должен выполнить транзакцию завершения, или отвергает его пакетом *NAK*, после которого хост должен повторить запуск. Транзакция завершения состоит из маркеров *CS* и *OUT*, на что транслятор отвечает подтверждениями *ACK*, *NAK*, *STALL* (это обычные ответы целевой конечной точки) или *NYET* — транзакция еще не завершена, хосту следует позже повторить транзакцию завершения.

Транзакции запуска *ввода* состоит из последовательности маркеров *SS* и *IN*, транслятор своим ответом подтверждает (*ACK*) или отвергает (*NAK*) запуск. Транзакция завершения состоит из маркеров *CS* и *IN*, на что транслятор отвечает пакетом данных (*DATA0* или *DATA1*) или подтверждениями *NAK*, *STALL* или *NYET* с вышеописанным значением.

Специфические дескрипторы и запросы к хабам

Хабы USB относятся к стандартным устройствам с кодами *класса 09 подкласса 00*. Код *протокола* характеризует транслятор транзакций (для хабов 2.0), он зависит от типа, текущей скорости работы и структуры хаба:

- 00 — хаб FS (без транслятора транзакций) или хаб HS, подключенный к на FS-порту;

- 01 — HS-хаб с одним транслятором транзакций;
- 02 — HS-хаб с несколькими трансляторами транзакций.

Эти коды классификации хаб сообщает в своих дескрипторах устройства и интерфейса.

Кроме стандартных дескрипторов хаб имеет специальный *классовый дескриптор* (hub class descriptor), содержание которого раскрывает табл. 14.1.

Таблица 14.1. Классовый дескриптор хаба

Смещение	Поле	Длина	Содержание
0	`bDescLength`	1	Длина дескриптора (зависит от числа портов, 9 для хабов с числом нисходящих портов 1–7)
1	`bDescriptorType`	1	Тип (29h — дескриптор хаба)
2	`bNbrPorts`	1	Число нисходящих портов
3	`wHubCharacteristics`	2	Характеристики хаба: Биты [1:0] — режим управления питанием портов: 00 — для всех портов одновременно (Ganged power switching), 01 — селективное, 1x — нет управления (для хабов 1.0); Бит 2 — признак комбинированного устройства: 1 — к некоторым портам хаба всегда подключены встроенные устройства; Биты [4:3] — Режим защиты от перегрузки по питанию: 00 — глобальная защита, 01 — индивидуальная защита портов, 1x — нет защиты (для хабов, питающихся от шины); Биты [6:5] — характеристика быстродействия транслятора транзакций (TT): 00 — TT делает межкадровый зазор на нисходящих портах до 8 FS bt, 01 — до 16, 10 — до 24, 11 — до 32; Бит 7 — наличие управляемых индикаторов состояния нисходящих портов; Биты [15:8] — резерв (0); в USB 1.x резервными были биты [15:5]
5	`bPwrOn2PwrGood`	1	Время установления нормального питания на нисходящих портах с момента команды включения (в 2 мс-интервалах)
6	`bHubContrCurrent`	1	Максимальный ток, потребляемый контроллером хаба (мА)
7	`DeviceRemovable`	?	Признаки возможности отсоединения устройств от портов. Бит 0 не используются, биты 1…n соответствуют портам 1…n (0 — отсоединяемое устройство, 1 — нет). Длина поля (в байтах) зависит от числа портов, в «лишних» битах — нули
?	`PortPwrCtrlMask`	?	Маска управления питанием портов (в USB 1.x), единичное значение бита означает, что данный порт не подчиняется общей команде управления питания. В USB 2.0 все биты единичные. Связь битов с портами как и в предыдущем поле

Хаб имеет всего один интерфейс, в котором используется (кроме нулевой) только одна конечная точка типа *Interrupt-IN* для опроса признаков изменения состояния портов (*Status Change Endpoint*) с максимально возможным периодом опроса (`bInterval` = FFh). С этой точки хост получает информацию в виде битовой карты, размер которой (в байтах) зависит от числа портов хаба. Самый младший бит карты (бит 0) несет признак смены состояния хаба (1 — есть смена), бит 1 — смены состояния порта-1, бит 2 — порта-2 и т. д. Обычно эти сообщения однобайтные, поскольку более 7 портов в хабе встречается редко. Если изменения состояния портов не произошло, то хаб на опрос отвечает *NAK*'ом (не передает данных). По получении признака изменения состояния хаба хост выполняет запрос чтения состояния хаба (*GetHubStatus*), по получении признака изменения состояния порта — запрос чтения состояния этого порта (*GetPortStatus*).

Хаб поддерживает все стандартные запросы к устройствам, кроме управления интерфейсом (он всего один) и установки метки времени (у хаба нет изохронных точек). Кроме того, он должен поддерживать классовые запросы, определенные для хабов (табл. 14.2).

Таблица 14.2. Классовые запросы хаба

Запрос	bmRequestType	bRequest
ClearHubFeature	00100000B	1
ClearPortFeature	00100011B	1
ClearTTBuffer	00100011B	8
GetHubDescriptor	10100000B	6
GetHubStatus	10100000B	0
GetPortStatus	10100011B	0
ResetTT	00100011B	9
SetHubDescriptor	00100000B	7
SetHubFeature	00100000B	3
SetPortFeature	00100011B	3
GetTTState	10100011B	10
StopTT	00100011B	11

Для *опроса состояния* хаба и каждого его нисходящего порта имеются специальные (классовые) запросы *GetHubStatus* и *GetPortStatus* (`wLength`=4). Ответом на *GetHubStatus* будет *слово состояния хаба* `wHubStatus`, за которым следует *слово изменения состояния* `wHubChange`. В запросе *GetPortStatus* в поле `wIndex` указывается номер порта, ответом на него будет *слово состояния порта* `wPortStatus`, за которым следует *слово изменения состояния порта* `wPortChange`. Форматы этих слов приведены в табл. 14.3.

Таблица 14.3. Форматы слов состояния и изменений состояния хаба и портов

Бит	Имя признака	Назначение
Слово состояния хаба wHubStatus		
0	`Hub_Local_Power`	Признак состояния локального источника питания хаба: 0 — локальное питание в норме, 1 — потеряно локальное питание.
1	`Hub_Over_Current`	Перегрузка по питанию (только для хабов с общей защитой для всех портов): 1 — сработала общая цепь защиты, 0 — нет
[2:15]	Резерв	(0)
Слово изменения состояния wHubChange		
0	`C_Hub_Local_Power`	Признак смены состояния локального питания: 1 — была смена, 0 — нет
1	`C_Hub_Over_Current`	Признак смены состояния общей перегрузки: 1 — была смена, 0 — нет
[2:15]	Резерв	(0)
Слово состояния порта wPortStatus		
0	`Port_Connection`	Текущее состояние подключения к порту: 1 — подключено устройство, 0 — нет
1	`Port_Enable`	Состояние разрешения порта: 0 — *disabled*, 1 — *enabled*. Устанавливается только программно хостом, сбрасывается программно и аппаратно по ошибке устройства
2	`Port_Suspend`	Состояние приостановки порта: 0 — не приостановлен, 1 — приостановлен или находится в процессе возобновления. Устанавливается только программно, сбрасывается программно или сигналом удаленного пробуждения от этого порта
3	`Port_Over_Current`	Перегрузка порта по питанию (только при селективной защите портов): 1 — защита сработала, 0 — нет
4	`Port_Reset`	Сброс порта: 1 — подается сигнал Reset, 0 — нет. Устанавливается программно, сбрасывается аппаратно хабом по окончании сигнализации
[5:7]	Резерв	(0)
8	`Port_Power`	Состояние питания порта (отражает реальную ситуацию только при селективном управлении питанием): 0 — порт не запитан, 1 — порт не обесточен селективно
9	`Port_Low_Speed`	Признак низкой скорости: 1 — обнаружено подключение LS-устройства, 0 — подключенное устройство (если есть) либо FS, либо HS (смотря по биту 10)
10	`Port_High_Speed`	Признак высокой скорости: 1 — с устройством согласована высокая скорость, 0 — нет
11	`Port_Test`	Состояние тестирования порта: 1 — порт в режиме тестирования, 0 — нет (устанавливается программно)
12	`Port_Indicator`	Управление индикатором порта: 0 — индикатор управляется аппаратно, 1 — программно
[13:15]	Резерв	(0)

Бит	Имя признака	Назначение
Слово изменения состояния порта wPortChange		
0	C_Port_Connection	Изменение состояние подключения устройства: 1 — состояние сменилось, 1 — нет
1	C_Port_Enable	Изменение состояния разрешения порта, устанавливается в 1 при автоматическом запрещении работы порта из-за обнаруженной ошибки
2	C_Port_Suspend	Изменение состояния приостановки устройства, устанавливается в 1 при завершении возобновления работы устройства (Resume complete)
3	C_Port_Over_Current	Изменение состояния индивидуальной защиты порта: 1 — состояние изменилось, 0 — нет (или нет индивидуальной защиты)
4	C_Port_Reset	Изменение состояния сброса порта: 0 — нет изменений, 1 — сброс завершен
[5:15]	Резерв	(0)

Управление свойствами хаба (запросы *SetHubFeature* и *ClearHubFeature*) и каждого из нисходящих портов (запросы *SetPortFeature* и *ClearPortFeature*) обеспечивают функции, приведенные в табл. 14.4.

Таблица 14.4. Управление свойствами хаба

Имя свойства	Номер (wValue)	Запрос установки	Запрос сброса
Свойства хаба		**SetHubFeature**	**ClearHubFeature**
C_Hub_Local_Power	0	Установка (для диагностических целей) одноименного признака в слове изменения состояния хаба	Сброс одноименного признака в слове изменения состояния хаба (подтверждение, что хост его воспринял)
C_Hub_Over_Current	1	То же	То же
Свойства порта		**SetPortFeature**	**ClearPortFeature**
Port_Connection	0	Запрос не используется, хаб не выполняет никаких операций	
Port_Enable	1	Перевод запитанного порта в состояние *Enabled*	Перевод порта в состояние *Disabled*
Port_Suspend	2	Приостановка порта	Генерация возобновления на приостановленном порте
Port_Over_Current	3	Запрос не используется, хаб не выполняет никаких операций	
Port_Reset	4	Сброс и перевод запитанного порта в состояние Enabled	Запрос не используется, хаб не выполняет никаких операций
Port_Power	8	Подача питания на порт	Снятие питания с порта

продолжение

Таблица 14.4 (*продолжение*)

Имя свойства	Номер (wValue)	Запрос установки	Запрос сброса
`Port_Low_Speed`	9	Запрос не используется, хаб не выполняет никаких операций	
`C_Port_Connection`	16		Сброс одноименного признака в слове изменения состояния порта (подтверждение, что хост его воспринял)
`C_Port_Enable`	17		То же
`C_Port_Suspend`	18		То же
`C_Port_Over_Current`	19		То же
`C_Port_Reset`	20		То же
`Port_Test`	21	Перевод порта в тестовый режим (только для портов в состоянии *Disabled*, *Disconnected* или *Suspended*)	Нет, выход из теста только по сбросу хаба
`Port_Indicator`	22	Программное управление индикатором порта	Отключение программного управления индикатором порта

В *запросе установки тестового режима* в младшем байте поля `wIndex` задается номер порта, в старшем — селектор, определяющий выполняемый тест (см. главу 12).

В *запросе установки управления индикатором* в младшем байте поля `wIndex` задается номер порта, в старшем — селектор, определяющий режим индикации: 0 — автоматическая работа индикатора, 1 — принудительно янтарный, 2 — принудительно зеленый, 3 — принудительно отключен, 4–FFh — резерв. Выбор нулевого селектора эквивалентен сбросу признака управления индикатором.

Для хабов с протоколами типа 1 и 2 имеются *запросы управления транслятором транзакций* (TT), используемые в основном в отладочных целях:

- *GetTTState* — опрос состояния транслятора (для диагностики);
- *StopTT* — останов транслятора (если трансляторов несколько, то для конкретного порта);
- *ResetTT* — сброс транслятора в исходное состояние (если трансляторов несколько, то для конкретного порта);
- *ClearTTBuffer* — очистка буфера непериодических транзакций. При запросе указывается адрес устройства и конечной точки, а также порта хаба (если транслятор один на все порты, то указывается порт 1).

ГЛАВА 15

Хост USB

Хост является главным действующим лицом в организации конфигурирования и выполнения транзакций USB. У каждой шины USB должен быть один (и только один!) хост — компьютер с контроллером USB. Однако понятие *компьютер* отнюдь не означает лишь привычные варианты настольных, напольных, портативных компьютеров. Компьютер — это сочетание процессора, памяти и периферийных устройств; в таком понимании в большинстве современных устройств присутствуют встроенные компьютеры. Если «интеллекта» этого компьютера и его возможностей диалога с пользователем оказывается достаточно, то он может взять на себя роль хоста USB. Такой вариант хоста рассматривается в последнем параграфе данной главы.

«Классический» хост USB делится на три основных уровня:

- *интерфейс шины USB* обеспечивает физический интерфейс и протокол шины. Интерфейс шины реализуется хост-контроллером, имеющим встроенный корневой хаб, обеспечивающий точки физического подключения к шине (гнезда USB типа «A»). Хост-контроллер отвечает за генерацию микрокадров. На аппаратном уровне хост-контроллер обменивается информацией с основной памятью компьютера, используя прямое управление шиной (bus-mastering) с целью минимизации нагрузки на центральный процессор;
- *система USB*, используя хост-контроллер(ы), транслирует клиентское «видение» обмена данными с устройствами — запросы IRP (I/O Request Packet — пакет запроса ввода/вывода) — в транзакции, выполняемые с реальными устройствами шины. Система отвечает и за распределение ресурсов USB — полосы пропускания и мощности источников питания (для устройств, питающихся от шины). Система состоит из трех основных частей:
  - *драйвер хост-контроллер*а — *HCD* (Host Controller Driver) — модуль, привязанный к конкретной модели контроллера, обеспечивающий абстрагирование драйвера USB и позволяющий в одну систему включать несколько разнотипных контроллеров;
  - *драйвер USB* — *USBD* (USB Driver) — обеспечивает основной интерфейс (USBDI) между клиентами и устройствами USB. Интерфейс HCDI (Host Controller Driver Interface) между USBD и HCD спецификацией USB не регламентируется. Он определяется разработчиками ОС и должен поддерживаться разработчиками хост-контроллеров, желающих иметь поддержку

своих изделий конкретными ОС. Клиенты не могут пользоваться интерфейсом HCDI; для них предназначен интерфейс USBDI. USBD обеспечивает механизм обмена в виде пакетов *IRP*, запрашивающих транспортировку данных по заданному каналу. Кроме того, USBD отвечает за некоторое абстрактное представление устройства USB клиенту, которое позволяет выполнять конфигурирование и управление состоянием устройств (включая и стандартное управление через конечную точку «0»). Реализация интерфейса USBDI определяется операционной системой; в спецификации USB излагаются только общие идеи;

- *программное обеспечение хоста* реализует функции, необходимые для функционирования системы USB в целом: обнаружение подключения и отключения устройств и выполнение соответствующих действий по этим событиям (загрузки требуемых драйверов), нумерацию устройств, распределение полосы пропускания и потребляемой мощности, управление состоянием энергопотребления и т. п.

- *клиенты USB* — программные элементы (приложения или системные компоненты), взаимодействующие с устройствами USB. Клиенты могут взаимодействовать с любыми устройствами (наборами их доступных конечных точек, входящих в выбранные интерфейсы), подключенными к системе USB. Однако система USB изолирует клиентов от непосредственного обмена с какими-либо портами (в пространстве ввода/вывода) или ячейками памяти, представляющими интерфейсную часть контроллера USB.

В совокупности уровни хоста предоставляют следующие возможности:

- обнаружение подключения и отсоединения устройств USB;
- манипулирование потоками управления между устройствами и хостом;
- манипулирование потоками данных;
- сбор статистики активности и состояний устройств;
- управление электрическим интерфейсом между хост-контроллером и устройствами USB, включая управление электропитанием.

Программная часть хоста в полном объеме реализуется операционной системой. До загрузки ОС может функционировать лишь усеченная часть ПО USB, поддерживающая только устройства, требующиеся для загрузки. Так, в BIOS современных системных плат имеется поддержка клавиатуры USB, реализующая функции сервиса `Int 9h`. После загрузки системы USB эта «дозагрузочная» поддержка игнорируется — система начинает работу с контроллером «с чистого листа», то есть со сброса и определения всех подключенных устройств. В спецификации PC'2001 выдвигается ряд требований к BIOS, в частности требование поддержки загрузки ОС с устройств USB.

Хост-контроллер

Хост-контроллер является аппаратным посредником между устройствами USB и хостом. В настоящее время имеется три спецификации хост-контроллеров, каждой из которых соответствует свой комплект драйверов хост-части:

- *UHC* (Universal Host Controller) — универсальный хост-контроллер для шины USB 1.x, разработанный Intel;
- *OHC* (Open Host Controller) — «открытый» хост-контроллер для шины USB 1.x, разработанный Compaq, Microsoft и National Semiconductor;
- *EHC* (Enhanced Host Controller) — расширенный хост-контроллер для поддержки высокой скорости шины USB 2.0.

Все эти варианты контроллеров выполняют одни и те же задачи: организуют физические транзакции с устройствами по шине USB в соответствии с описаниями (дескрипторами) этих транзакций, помещенными в системное ОЗУ драйвером хост-контроллера. При этом транзакции разных типов обрабатываются по-разному. В плане обработки ошибок проще всего устроены изохронные транзакции, где ошибки не требуют повторов. Транзакции передач с гарантированной доставкой в случае ошибок требуют повторов до победного конца или признания неудачи (исчерпания допустимого числа повторов). С точки зрения планирования следует выделить периодические транзакции, которые должны выполняться строго по графику, остальные — как получится, и их ставят в очереди. Из-за особенностей планирования и возможных повторов порядок завершения обработки дескрипторов транзакций (успешных или нет) будет отличаться от порядка их помещения в память[1], что прибавляет забот хост-контроллеру и его драйверу. Три варианта хост-контроллеров решают эти задачи по-разному и используют разные стратегии планирования транзакций, что иллюстрирует рис. 15.1.

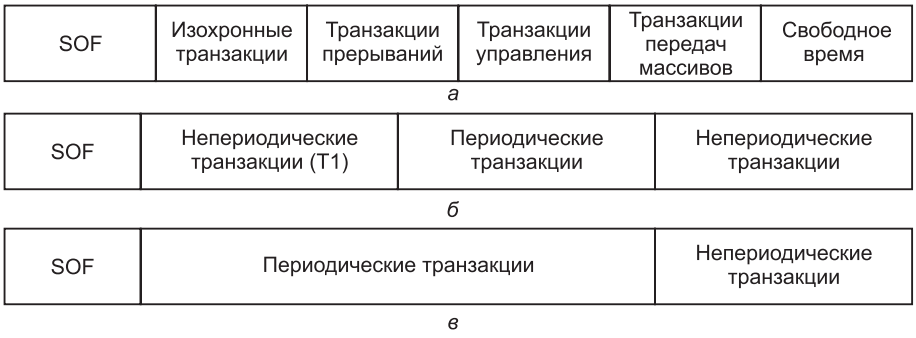


Рис. 15.1. План распределения времени в кадре: *а* — UHC; *б* — OHC; *в* — EHC (в микрокадре)

«Универсальный» хост-контроллер — UHC

Хост-контроллер UHC от Intel появился в микросхеме PIIX3 (мост PCI-ISA) чипсетов системных плат для процессоров Pentium и используется во многих последующих изделиях Intel. Это FS/LS хост-контроллер, который большую часть забот по планированию транзакций перекладывает на ПО, — драйвер контроллера UHC (UHCD). Интерфейс контроллера UHC описан в документе Universal Host Controller Interface (UHCI) Design Guide, версия 1.1 вышла в 1996 году.

[1] Имеется в виду порядок выполнения запросов к разным конечным точкам. Для каждой конечной точки (канала) порядок завершений всегда соответствует порядку помещения запросов.

Драйвер UHC формирует для хост-контроллера дескрипторы, называемые в UHCI «дескрипторами передач» (TD — Transfer Descriptor), на самом деле описывающие каждую *шинную транзакцию*. Напомним, что в терминах спецификации USB одна *передача* (transfer) может состоять из нескольких *транзакций*, а в управляющих передачах используется еще и свой тип транзакции для каждой фазы (см. главу 10). Для транзакций передач с гарантированной доставкой дескрипторы TD приходится организовывать в *очереди*. Очереди нужны для таких передач, поскольку заранее не известно, сколько раз придется пытаться их исполнить. Продвижение очереди возможно только по успешному выполнению транзакции, находящейся в голове очереди, — это правило обеспечивает гарантированный порядок (в пределах своей очереди) доставки пакетов. Каждая очередь имеет свой *заголовок* (QH). Изохронные передачи исполняются всегда однократно (здесь нет гарантированной доставки), что упрощает их планирование. Драйвер размещает дескрипторы TD и QH в памяти и связывает их между собой в соответствии с планом выполнения транзакций в каждом кадре. Драйверу UHC приходится составлять детальное «расписание» для каждого будущего кадра, для чего используется список Frame List на 1024 кадра. Хост-контроллер обходит списки дескрипторов, начиная с точки, на которую указывает Frame List для текущего кадра, и выполняет соответствующие транзакции. Результат исполнения транзакции помечается в ее дескрипторе, отработанная транзакция помечается как «неактивная», и контроллер, встретив ее при очередном обходе, просто переходит к следующей. Драйвер должен периодически просматривать дескрипторы, извлекая уже отработанные и передавая результаты выполнения клиентскому драйверу. Логика работы контроллера подразумевает, что одному *запросу ввода/вывода* (*IRP*) от клиентского драйвера может соответствовать несколько «передач» — элементов очереди. Драйвер UHC разбивает запрос на транзакции и помещает дескрипторы этих транзакций в соответствующую очередь, а очередь включает в ближайшие планы. Драйвер отвечает за балансировку загрузки шины в каждом кадре, в частности, за гарантию предоставления не менее 10% полосы для транзакций управляющих передач. Планированием кадров также обеспечивается требуемая частота обращений к точкам периодических передач.

Контроллер UHC является активным устройством PCI (Bus-Master). Основное взаимодействие драйвера с хост-контроллером происходит с помощью дескрипторов, расположенных в памяти. Контроллер имеет регистры (в пространстве ввода/вывода), с помощью которых можно управлять его поведением: выполнять сброс, глобальную приостановку и пробуждение, подстраивать частоту кадров, управлять запросами прерываний, управлять портами встроенного корневого хаба. Контроллер позволяет работать в отладочном режиме, останавливаясь после выполнения каждой транзакции.

В процессе отработки плана контроллер считывает из памяти дескрипторы и данные, необходимые для начала транзакции. Как только в FIFO-буфер контроллера из памяти поступает информация, достаточная для начала транзакции, контроллер начинает транзакцию на шине USB. В процессе ее исполнения производится передача данных, после завершения контроллер модифицирует дескрипторы в памяти в соответствии с условиями завершения транзакции. В процессе отработки транзакции могут возникать ошибки переполнения или переопустошения

FIFO-буфера, связанные с перегрузкой контроллера системной памяти или шины PCI. Эти серьезные ошибки инициируют аппаратные прерывания. В состав хост-контроллера входит и корневой хаб на 2 или более порта.

Прерывания от UHC могут инициироваться различными событиями, такими как выполнение транзакций (избранных), обнаружение приема короткого пакета, прием сигнала возобновления, или в результате ошибки. Прерываний по подключению-отключению устройств контроллер не вырабатывает.

В контроллере UHC имеется специальная *поддержка традиционного интерфейса клавиатуры и мыши* через контроллер 8042 — перехват обращений к портам 60h и 64h пространства ввода/вывода. При разрешенной эмуляции по обращениям ПО к этим портам UHC вызывает системное прерывание *SMI* (System Management Interrupt), обрабатывающееся в ПК на процессорах x86 в режиме *SMM* (System Management Mode), невидимо для обычных программ. Обработчик SMI, перехватывающий эти обращения, формирует последовательности действий, необходимые для их исполнения с помощью клавиатуры и (или) мыши USB. Единственное исключение делается при перехвате команд, управляющих вентилем `GateA20`, — вместо генерации SMI манипуляции этим вентилем выполняются аппаратно (как это давно делается и в 8042). Эта аппаратная поддержка включается установкой соответствующих параметров CMOS Setup.

Большое неудобство работы с UHC возникает из-за необходимости программного просмотра всех дескрипторов передач на предмет выявления завершенных. Дескрипторы завершенных передач необходимо программно извлекать из цепочек, сохраняя связанность элементов. Планирование транзакций (составление списков дескрипторов и заголовков) — тоже достаточно трудоемкая задача для драйвера. Очевидно, преследовалась цель упрощения аппаратных средств хост-контроллера. Однако это может обернуться зависимостью эффективной производительности шины USB от мощности и загрузки центрального процессора. Такой подход к организации ввода/вывода трудно назвать эффективным.

Структуры данных и регистры контроллера UHC

Драйвер в системной памяти создает *список кадров* `Frame List`, состоящий из 1024 элементов. Каждый элемент этого списка содержит 32-битный указатель на связанный список структур данных, по которым контроллер выполняет транзакции в данном кадре. Хост-контроллер имеет регистр базового адреса списка кадров, указывающий на начало списка. Текущий номер отрабатываемого элемента определяется десятью младшими битами счетчика кадров, находящегося в контроллере и инкрементируемого каждую миллисекунду. Период счета кадров можно немного варьировать, изменяя константу, занесенную в регистр модификации длительности кадра (`SOF Modify Register`), что обеспечивает возможность подстройки частоты кадров для синхронизации изохронных обменов.

Элемент списка кадров может указывать либо на дескриптор изохронной передачи `TD` (Transfer Descriptor), либо (если в данном кадре изохронный обмен не планируется) на заголовок очереди `QH` (Queue Head). Если в данном кадре вообще не планируются передачи, то в элементе устанавливается признак-«заглушка» `T`

(Terminate, конец связанного списка, в данном случае — пустого). Еще раз напомним, что здесь слово «передача» (Transfer, согласно спецификации UHCI) употребляется в узком смысле — она соответствует одной транзакции (передаче не более одного пакета данных). Элемент (32-битное слово) имеет формат, приведенный на рис. 15.2. Поле FLLP (Frame List Link Pointer) — указатель на элемент; бит T — признак последнего элемента (при T = 1 указатель FLLP недействителен). Бит Q задает класс связанного элемента, на который указывает FLLP (0 — TD, 1 — QH).

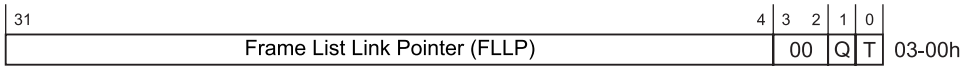


Рис. 15.2. Формат элемента списка кадров для UHC

Для каждого кадра из списка устанавливается своя цепочка дескрипторов изохронных передач (возможно и пустая), последний из этой цепочки должен ссылаться на цепочку заголовков очередей. Цепочки заголовков QH могут быть общими для группы кадров или даже для всех кадров списка. Общая идея построения очередей состоит в том, чтобы создавать свою очередь для каждого установленного канала (для всех сконфигурированных точек, кроме изохронных). «Дежурный» метод обслуживания — по горизонтали, тогда после выполнения транзакции с одной точкой контроллер перейдет к другой точке (другой очереди). Связывание TD и QH через указатели позволяет формировать произвольные конфигурации переходов от одной очереди к другой и даже делать петли — в последнем случае возможно, что с одной точкой в кадре успеют пройти несколько транзакций. Однако это нетипичный способ планирования. Если очередей много (установлено много каналов), то они распределяются по кадрам (из 1024-элементного списка) так, чтобы цепочка каждого кадра обязательно прошла по горизонтали до конца. Это можно спланировать, поскольку максимальное время для отработки одного элемента каждой очереди (как и изохронных транзакций) заранее известно (оно определяется типом передачи, максимальным размером пакета и скоростью устройства, что известно системе USB). При необходимости «горизонтальную справедливость» можно нарушить, задав вертикальный порядок обслуживания, — контроллер, успешно обработав из очереди передачу с признаком V = 1, перейдет к следующему дескриптору из этой же очереди, а не к следующей очереди.

Дескрипторы передач и заголовки очередей размещаются драйвером в ОЗУ по адресам, выровненным по границе параграфа, поскольку в качестве указателей используются лишь старшие 28 бит (биты [3:0] используются для служебных признаков).

Дескриптор передачи (TD) состоит из 32 байтов, из которых хост-контроллер использует только первые четыре 32-битных слова DW0–DW3. Слова DW4–DW7 зарезервированы для использования драйвером UHC (для организации «сборки мусора» — повторного использования отработанных областей). Формат дескриптора передачи приведен на рис. 15.3. Серым цветом выделены поля, модифицируемые хост-контроллером.

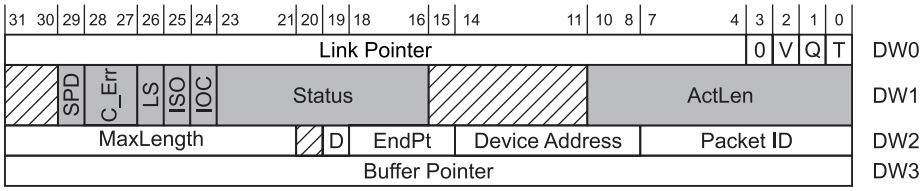


Рис. 15.3. Формат дескриптора передачи для UHC

В *слове DW0* поле `Link Pointer` аналогично полю `FLLP`, а биты `T` и `Q` аналогичны одноименным битам элемента списка кадров. Бит `V` — метод обслуживания `TD` (1 — в глубину, 0 — в ширину).

Слово DW1 используется для управления и определения состояния выполнения передачи, модифицируется хост-контроллером. Поле `ActLen` — действительная длина переданных данных; поле `Status` — *состояние* выполнения передачи:

- бит 23: `Active` — «надо исполнять», устанавливается драйвером, сбрасывается контроллером по успешному исполнению или исчерпанию лимита повторов;
- бит 22: `Stalled` — точка ответила пакетом `STALL`;
- бит 21: `Data Buffer Error` — ошибка буфера данных (переполнение или переопустошение FIFO при выполнении транзакции), транзакция остается активной (при переопустрошении контроллер генерирует пакет с ошибочным CRC, при переполнении не отвечает подтверждением);
- бит 20: `Babble` — при выполнении данной транзакции обнаружена «болтливость» устройства (оно отключается и устанавливается бит `Stalled`);
- бит 19: `NAK` — получение соответствующего ответа (в транзакции *SETUP* получение `NAK` устанавливает и признак ошибки тайм-аута);
- бит 18: `CRC/Time Out Error` — обнаружена ошибка передачи (CRC или тайм-аут);
- бит 17: `Bitstuff Error` — обнаружена ошибка вставки бит.

Биты [24:31] используются для управления передачей. Бит `IOC` заказывает прерывание по исполнению (прерывание генерируется в конце кадра, даже если транзакция уже неактивна, выборка ее дескриптора вызовет прерывание). Бит `ISO` — признак изохронной передачи (указание не делать повторных попыток). Бит `LS` — признак LS-устройства, использовать преамбулу перед передачей. Поле `C_ERR` — счетчик повторных попыток, декрементируемый по каждой ошибке. Переход в 1 или 0 вызывает перевод дескриптора в неактивное состояние. Если драйвер устанавливает нулевое значение, то число повторов неограниченно. Бит `SPD` — детектор короткого пакета: если в транзакции *IN*, стоящей в очереди, успешно принято меньше данных, чем ожидалось, то в конце кадра вырабатывается условие прерывания.

В *слове DW2* содержится информация для выполнения транзакции: `Packet ID` — тип используемого маркера *IN* (69h), *OUT* (E1h) или *SETUP* (2Dh); `Device Address` — адрес устройства USB; `EndPt` — номер и направление конечной точки. Бит `D` (Data

Toggle) — состояние переключателя для передаваемого или посылаемого пакета. Поле MaxLength — длина передаваемых данных (максимальная длина принимаемых), 000 — 1 байт, 001 — 2, 3FF — 1024; 7FFh — 0 (пустой пакет). Допустимые значения до 4FFh — 1280 байт, теоретический предел емкости кадра. Значения 500–7FEh недопустимы, вызывают фатальную ошибку контроллера.

В *слове DW3* содержится Buffer Pointer — указатель на буфер в ОЗУ, используемый для данных этой передачи.

Заголовок очереди (QH) связывает очереди друг с другом (по горизонтали) и ссылается на первый элемент (TD) данной очереди. Хост-контроллер использует два 32-битных слова (рис. 15.4). В поле QHLP (Queue Head Link Pointer) содержится указатель на следующий заголовок очереди (горизонтальная связка). В поле QELP (Queue Element Link Pointer) содержится указатель на элемент очереди (вертикальная связка). Признаки последнего элемента (T) и класс связанного элемента (Q) аналогичны одноименным признакам и классам в вышеприведенных структурах.

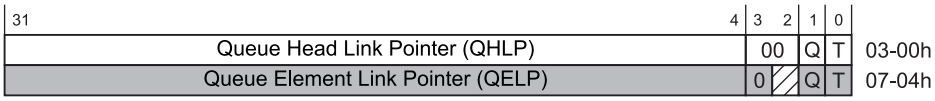


Рис. 15.4. Формат заголовка очереди для UHC

Дескриптор заголовка очереди создается драйвером; хост-контроллер модифицирует в памяти указатель QELP: успешно отработав транзакцию, контроллер берет из DW0 ее дескриптора указатель на следующий элемент и помещает его на место QELP в заголовке очереди. Таким образом, успешно отработанный TD удаляется из очереди. Когда удаляется последний TD, в QELP устанавливается признак пустой очереди (T). В случае неисправимой ошибки при отработке какого-то дескриптора в QELP также устанавливается «заглушка» T — поток с гарантированной доставкой не позволяет пропустить какую-либо транзакцию. Поле QELP может ссылаться как на TD (тривиальный вариант планирования), так и на QH — очередь сама может содержать очереди.

Регистровая модель UHC поясняется в табл. 15.1, где представлены регистры, отображенные на пространство ввода/вывода. Кроме того, как всякое устройство PCI, контроллер UHC имеет регистры в конфигурационном пространстве, в которых, в частности, задаются *коды класса* (0Ch — контроллер последовательной шины), *подкласса* (03 — USB) и *программного интерфейса* (00) в классификации PCI SIG.

Таблица 15.1. Регистры контроллера UHC

Адрес	Назначение
Base + (00–01h)	USBCMD — регистр команд USB Биты 15:8 — резерв Бит 7: MAXP (Max Packet) — допустимый размер пакета (для FS), с которым возможна транзакция при подходе к концу кадра: 1 = 64 байт, 0 = 32 байта

Адрес	Назначение
	Бит 6: CF (Configure Flag) — флаг, которым драйвер отмечает окончание процесса конфигурирования контроллера (программный семафор для ПО) Бит 5: SWDBG (Software Debug) — управление отладкой: 1=режим отладки (останов после каждой транзакции), 0 — нормальный Бит 4: FGR (Force Global Resume) — подача сигнала глобального пробуждения.Устанавливается программно, сбрасывается аппаратно по окончании пробуждения Бит 3: EGSM (Enter Global Suspend Mode) — перевод в режим глобальной приостановки Бит 2: GRESET (Global Reset) — общий сброс контроллера и шины USB Бит 1: HCRESET (Host Controller Reset) — сброс хост-контроллера Бит 0 RS (Run/Stop) управление работой контроллера: 1=Run — выполнение транзакций по плану, 0=Stop — останов
Base + (02–03h)	USBSTS — регистр состояния USB Биты [15:6] — резерв Бит 5: HCHalted — контроллер остановлен, программно или аппаратно (по ошибке или при отладке) Бит 4: Host Controller Process Error — фатальная ошибка исполнения (может возникать и из-за некорректного задания PID в дескрипторе транзакций), вызывает прерывание Бит 3: Host System Error — системная ошибка (неполадки в интерфейсе PCI), вызывает прерывание Бит 2: Resume Detect — получение сигнала возобновления (при глобальной приостановке) Бит 1: USB Error Interrupt — признак прерывания по ошибке выполнения транзакции (переполнение или переопустошение FIFO буфера шины PCI) Бит 0: USBINT (USB Interrupt) — прерывание по выполнению транзакции с установленным битом IOC или приему короткого пакета (при включенном обнаружении короткого пакета)
Base + (04–05h)	USBINTR — регистр разрешения прерываний Биты [15:4] — резерв Бит 3: Short Packet Interrupt Enable — разрешение прерываний по приему короткого пакета Бит 2: IOC (Interrupt On Complete Enable) — разрешение прерываний по завершении транзакции Бит 1: Resume Interrupt Enable — разрешение прерываний по приему сигнала возобновления Бит 0: Timeout/CRC Interrupt Enable — разрешение прерываний по ошибке тайм-аута и CRC-контролю
Base + (06–07h)	FRNUM — регистр номера кадра
Base + (08–0Bh)	FRBASEADD — регистр базового адреса списка кадров
Base + 0Ch	SOFMOD — регистр управления частотой кадров Биты [6:0] — управление длительностью кадра: 0 — 11936 бит, 1 — 11937 бит, … 63 — 11999 бит, 64 — 12000 бит (номинал), 65 — 12001 бит, 127 — 12063 бит
Base + 10–11h)	PORTSC1 — регистр управления и состояния порта 1 Биты [15:13] — резерв (0) Бит 12: (R/W) Suspend — приостановка порта Биты [11:10] — резерв (0) Бит 9: (R/W) Port Reset — сброс порта

продолжение ⇩

Таблица 15.1 (*продолжение*)

Адрес	Назначение
	Бит 8: (RO) `Low Speed Device Attached` — признак подключения LS-устройства Бит 7 — резерв (1) Бит 6: (RW) `Resume Detect` — обнаружение сигнала возобновления. Запись «1» вызывает генерацию сигнала возобновления на порте, последующая запись «0» — завершение сигнала возобновления и посылка LS-EOP Биты [5:4]: (RO) — текущее состояние линий D- и D+ Бит 3: (R/WC) `Port Enable/Disable Change` — признак автоматического запрета порта по ошибке, сбрасывается записью «1» Бит 2: (R/W) `Port Enabled/Disabled` — разрешение работы порта Бит 1: (R/WC) `Connect Status Change` — признак события подключения/отключения устройства Бит 0: (RO) `Current Connect Status` — признак подключенного устройства
Base + (12–13h)	`PORTSC2` — регистр управления и состояния порта 2 (аналогично предыдущему)

«Открытый» хост-контроллер — OHC

Спецификация интерфейса «открытого» хост-контроллера OpenHCI (OHCI) разработана компаниями Compaq, Microsoft и National Semiconductor и описана в документе «Open Host Controller Interface Specification for USB». Версия 1.0a этого документа опубликована в 1999 году. Контроллер OHC, как и UHC, предназначен для поддержки скоростей FS/LS. Однако аппаратные средства OHC берут на себя большую часть забот планирования, разгружая ЦП от рутины постоянной обработки дескрипторов. Контроллер OHC оперирует дескрипторами конечных точек и дескрипторами передач.

Дескрипторы конечных точек `ED` (Endpoint Descriptor) создаются для всех сконфигурированных конечных точек всех подключенных устройств. Эти дескрипторы размещаются в памяти и связываются между собой; конфигурация связей задает порядок их обслуживания хост-контроллером. Дескриптор конечной точки описывает ее полный адрес и направление, тип, допустимый размер пакета, скорость, состояние точки и дескриптора, указатели на очереди передач, связанных с данной точкой, указатель на дескриптор следующей точки. Для всех точек управления (*Control*) и всех точек передач массивов (*Bulk*) создаются отдельные цепочки `ED`, на начала этих цепочек указывают специальные регистры OHC. Дескрипторы точек периодических передач организуются в «поваленное» двоичное дерево (рис. 15.5), в «ветвях» которого размещаются дескрипторы точек прерываний, а в «стволе» — дескрипторы точек прерываний с минимальным интервалом обслуживания и все дескрипторы точек изохронных передач. У дерева имеются 32 конечных ветви, проход по дереву осуществляется от конечных ветвей к стволу. В каждом из 32 смежных кадров вход осуществляется со своей ветви. Для этого в OHC имеется *регистр базового адреса HCCA* (Host Controller Communication Area, область коммуникаций хост-контроллера), указывающий на ветвь с номером 0, и счетчик кадров, 5 младших бит которого задают номер ветви входа для очередного кадра. Таким образом, через каждую ветвь пятого уровня (конечного) обра-

ботчик дескрипторов проходит 1 раз за 32 кадра (T = 32 мс), четвертого — 1 раз за 16 кадров (T = 16 мс), для третьего уровня — T = 8 мс, для второго — T = 4 мс, для первого — T = 2 мс, для нулевого (ствола) — T = 1 мс.

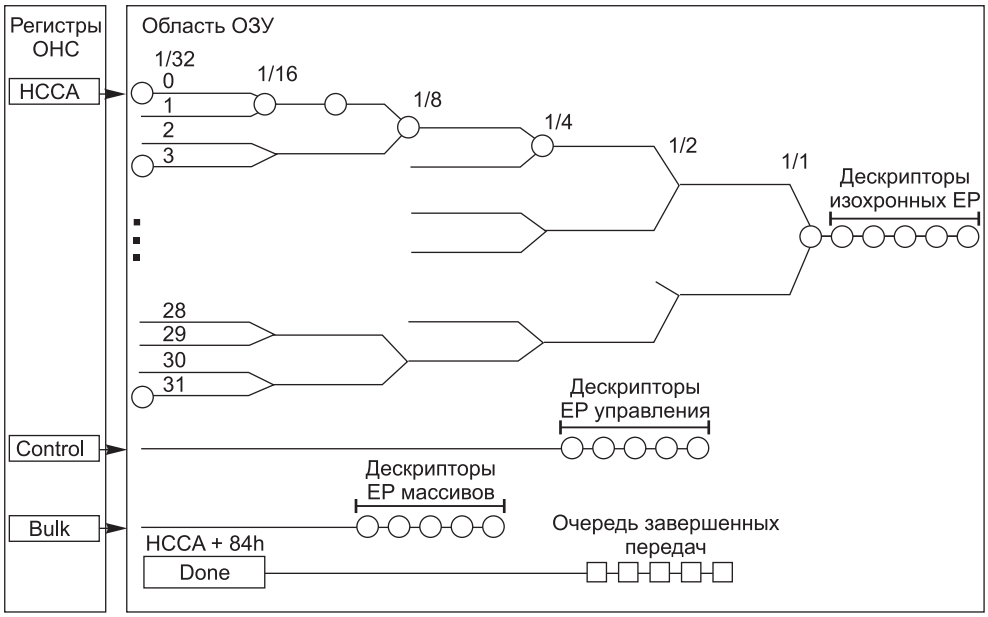


Рис. 15.5. Конфигурация связей дескрипторов конечных точек для ОНС

Дескрипторы передач `TD` (Transfer Descriptor), в отличие от TD UHC, для ОНС действительно описывают *передачи USB*. Каждая передача может разбиваться на несколько транзакций, и это разбиение выполняет хост-контроллер исходя из размера пакета, установленного в дескрипторе конечной точки. Буфер данных для передачи может располагаться в одной или двух физических страницах памяти, возможно, разрозненных. В виртуальном пространстве логических адресов буфер должен быть непрерывной областью. Размер передачи может достигать 8 Кбайт, но если буфер начинается не с начала страницы, то допустимый размер передачи сократится (в худшем случае до 4097 байт). Дескрипторы передач собираются в *очереди*, которые прикрепляются к дескрипторам конечных точек.

Хост-контроллер ОНС имеет *таймеры*, с помощью которых он осуществляет планирование транзакций в кадре в соответствии с рис. 15.1. После `SOF` контроллер начинает обход цепочки `ED` для управляющих передач и выполняет столько из них, сколько успеет за время *T1*. Далее он начинает обход дерева периодических передач, от *n*-й конечной ветви до ствола, пока не пройдет по всем встретившимся `ED`. Если у него еще остается время в кадре, он снова берется за непериодические передачи (*Bulk* и *Control*). Отработанные (успешно или снятые по превышению порога ошибок) дескрипторы контроллер собирает в специальную *очередь обработанных дескрипторов* `Done Queue`, откуда их без труда извлекает драйвер. Контроллер мо-

жет вырабатывать прерывания по завершению обработки TD, причем с заданной (для каждого TD) задержкой (или не вырабатывать запрос).

Контроллер OHC имеет регистр для подстройки частоты кадров. В контроллер входит и корневой хаб на 2 или более порта.

Контроллер OHC, как и UHC, обычно является активным устройством PCI (Bus Master), но по сравнению с UHC наделен большим интеллектом. В контроллере предусмотрена поддержка контроллера клавиатуры и мыши (KBC) с помощью прерываний SMI, но, в отличие от UHC, в OHC имеются и специальные регистры, упрощающие задачу эмуляции.

Структуры данных и регистры контроллера OHC

Все структуры данных OHC, расположенные в системной памяти, выровнены по границе параграфа (четыре младших бита адреса — нулевые). Для работы с OHC в памяти размещают *дескрипторы конечных точек* (ED) выбранных интерфейсов сконфигурированных устройств, *дескрипторы передач* (TD) для них и *коммуникационную область* HCCA. Дескрипторы передач подключаются драйвером к очередям требуемых конечных точек. Дескрипторы исполненных передач (завершенных нормально или снятых аварийно) контроллером подключаются к очереди Done Queue. Буфер данных для каждого дескриптора передачи может размещаться в одной или двух физических страницах, физические адреса этих страниц определяются по физическому адресу начала (начальное значение CBP) и физическому адресу конца буфера. Дескрипторы передач делятся на два типа: общий (для передач *Bulk*, *Control*, *Interrupt*) и изохронный.

Дескриптор конечной точки (ED) имеет размер 4 двойных слова (16 байт), его формат приведен на рис. 15.6, назначение полей приведено ниже:

- FA (FunctionAddress) — адрес устройства;
- EN (EndpointNumber) — номер точки;
- D (Direction) — направление передачи: 01 — *OUT*; 10 — *IN*; 00, 11 — брать из поля PID дескриптора передачи TD;
- S (Speed) — скорость: 0 — FS, 1 — LS;
- K (sKip) — пропустить (не начинать транзакций с данной точкой);
- F (Format) — признак формата TD, связанного с данной точкой: 0 — общий (Bulk, Control, Interrupt), 1 — дескриптор изохронной передачи;
- MPS (MaximumPacketSize) — максимальный размер пакета;
- TailP (TD Queue Tail Pointer) — указатель на конец очереди TD, если совпадает с HeadP — очередь пуста;
- HeadP (TD Queue Head Pointer) — указатель на очередной TD, который должен быть исполнен для данной точки;
- H (Halted) — признак останова точки;
- C (toggleCarry) — текущее значение переключателя Toggle Bit;
- NextED (Next Endpoint Descriptor) — указатель на дескриптор следующей конечной точки (если ноль, то эта точка — последняя в списке).

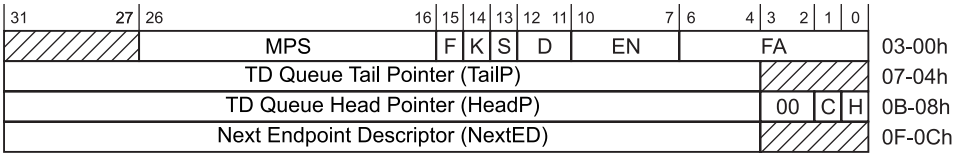


Рис. 15.6. Дескриптор конечной точки OHC

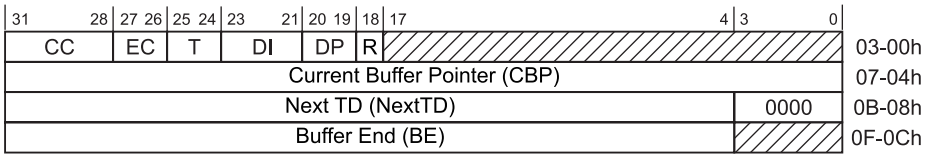


Рис. 15.7. Общий дескриптор передачи OHC

Формат *общего дескриптора передач* приведен на рис. 15.7. Дескриптор позволяет описывать передачу данных указанной длины в пределах 8 кбайт. Количество транзакций для передачи контроллер определяет сам из поля длины пакета в дескрипторе точки (`ED`) и длины буфера; количество транзакций при приеме определяется длиной пакетов, переданных устройством. Назначение полей общего дескриптора передач приведено ниже:

- `R` (bufferRounding) — требование совпадения длины последнего принимаемого пакета с объявленной длиной буфера (обнаружение коротких пакетов): 0 — сообщать об ошибке, если длина последнего пакета меньше длины буфера, 1 — не реагировать на несовпадение длины;
- `DP` (Direction/PID) — тип маркера (направление, должно соответствовать полю `D` в `ED`): 00 — *SETUP*, 01 —*OUT*, 10 — *IN*, 11 — резерв;
- `DI` (DelayInterrupt) — задержка прерывания по отработке передачи: 000–110 — задержка (число пропущенных кадров), 111 — не вызывать прерывание;
- `T` (DataToggle) — поле, используемое для генерации и сравнения переключателя типа пакетов данных. Вместе с битом `C` из `ED` обеспечивает корректное переключение при передачах данных, описанных серией `TD`. При постановке в очередь `TD` для передач типов *Bulk* и *Interupt* драйвер должен установить `T` = 00; для передач типа *Control* в фазе *Setup* драйвер должен установить `T` = 10; в фазе *Status* и в первой фазе данных — `T` = 11. Дальнейшие модификации поля выполняет контроллер;
- `EC` (ErrorCount) — счетчик ошибок передачи для текущей транзакции. Если при `EC` = 2 происходит ошибка, ее код помещается в `CC` и вся передача аварийно завершается. При успешном завершении транзакции счетчик обнуляется;
- `CC` (ConditionCode) — состояние последней попытки транзакции (табл. 15.2);
- `CBP` (Current Buffer Pointer) — текущий физический адрес в буфере, по которому будет записан или прочитан очередной передаваемый пакет. После успешного завершения передачи поле обнуляется, при нештатном завершении — указывает на текущее положение;

- NextTD — указатель на следующий дескриптор передачи для данной точки;
- BE (Buffer End) — физический адрес последнего байта в буфере для данного TD.

Формат *дескриптора изохронной передачи* приведен на рис. 15.8. Дескриптор позволяет описывать передачу данных длиной до 8 кбайт, состоящую из 1–8 транзакций (их количество задано в дескрипторе). Транзакции исполняются в последовательных кадрах, начиная с кадра с указанным номером. Длина пакета в каждой из этих транзакций вычисляется из значений полей OffsetN смежных транзакций (возможны и транзакции с нулевой длиной поля данных). Физический адрес буфера для каждой из этих транзакций определяется следующим образом: младшие 12 бит берутся из младших 12 бит соответствующего поля Offset, старшие биты (номер физической страницы) берутся либо из поля BPO (если в поле Offset бит 12 = 0), или из BE (если в поле Offset бит 12 = 1). Контроллер нормально отрабатывает переход буфера с одной страницы на другую даже в середине пакета. Назначение полей дескриптора изохронной передачи приведено ниже:

- SF (StartingFrame) — младшие 16 бит номера кадра, в котором должен быть передан первый пакет;
- DI (DelayInterrupt) — задержка прерывания по отработке передачи: 000–110 — задержка (число пропущенных кадров), 111 — не вызывать прерывание;
- FC (FrameCount) — число пакетов данных, описанных в данном TD (0 — 1 пакет, 7 — 8 пакетов);
- CC (ConditionCode) — код завершения на момент отправки TD в очередь выполненных заданий;
- BP0 (BufferPage0) — физический номер страницы с первым байтом буфера данных;
- NextTD — указатель на следующий дескриптор передачи для данной точки;
- BE (Buffer End) — физический адрес последнего байта в буфере для данного TD;
- Offset0… Offset7 — смещение, используемое для определения размера и стартового адреса данных каждого пакета;
- PSW0…PSW7 — слово состояния каждого пакета: биты [15:12] — код завершения, биты [10:0] — длина принятого пакета данных (в транзакциях *OUT* — 0), бит 11=0.

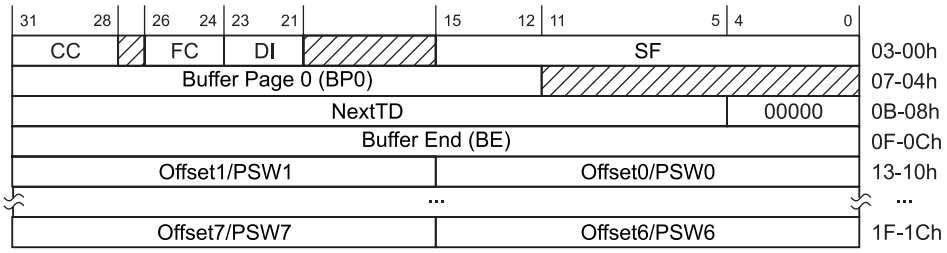


Рис. 15.8. Дескриптор изохронной передачи OHC

Коды завершения для всех типов передач приведены в табл. 15.2.

Таблица 15.2. Коды завершения передач для контроллера OHC

Код CC	Условие
0000	NoError — нормальное завершение без ошибок
0001	CRC — последний принятый пакет содержит ошибку CRC
0010	BitStuffing — последний принятый пакет содержит ошибку вставки бит
0011	DataToggleMismatch — последний принятый пакет данных имеет PID, не отвечающий ожидаемому значению переключателя
0100	Stall — конечная точка остановлена (получен одноименный пакет подтверждения)
0101	DeviceNotResponding — устройство не отвечает
0110	PIDCheckFailure — ошибка контроля достоверности поля PID
0111	UnexpectedPID — прием неожиданного или недопустимого PID
1000	DataOverrun — пакет данных от устройства длиннее разрешенного в поле MPS дескриптора точки
1001	DataUnderrun — точка вернула пакет, меньший чем MPS, и недостаточный для заполнения указанного буфера
1010, 1011	Резерв
1100	BufferOverrun — во время приема данных контроллер не успел записать данные в системную память
1101	BufferUnderrun — во время вывода данных контроллер не успел считать данные из системной памяти
111x	NotAccessed — не было обработки (драйвер записывает этот код в TD перед помещением его в очередь)

Таблица 15.3. Формат коммуникационной области HCCA контроллера OHC

Смещение	Длина	Назначение
0	128	`HccaInterrruptTable` — таблица дескрипторов точек прерываний, 32 двойных слова, указывающих на дескрипторы EP (см. рис. 15.5)
80h	2	`HccaFrameNumber` — текущий номер кадра, модифицируется контроллером перед обходом списка периодических передач
82h	2	`HccaPad1` — это слово контроллер обнуляет, когда модифицирует значение номера кадра
84h	4	`HccaDoneHead` — указатель на очередь исполненных передач. Контроллер модифицирует указатель по мере исполнения передач и сигнализирует об этом установкой бита WDH в регистре HcInterruptStatus. Указатель не будет повторно модифицироваться, пока программа не сбросит бит WDH. Бит 0 указателя используется для сигнализации о наличии иных незамаскированных условий прерываний (если обработчик прерывания, считывая указатель, получает нулевое значение бита, он может не анализировать регистр HcInterruptStatus)
88h	116	Резерв для контроллера

Кроме дескрипторов драйвер OHCI взаимодействует с контроллером через *коммуникационную область* в системной памяти — *HCCA*. На начало этой области указывает регистр HCCA. Структура области приведена в табл. 15.3.

Для оперативного взаимодействия драйвера и хост-контроллера в OHC имеется *блок операционных регистров*, приписанный к пространству памяти (его положение определяется регистрами BAR в конфигурационном пространстве устройства PCI, каковым и является OHC). К этому блоку система должна обеспечивать доступ и в режиме SMM. Размер блока регистров определяется числом нисходящих портов (NDP) корневого хаба, входящего в OHC; при наличии регистров поддержки эмуляции традиционного контроллера клавиатуры и мыши размер достигает 272 байт. Все регистры, кроме специальных регистров эмуляции, требуют 32-разрядных обращений. Операционные регистры обеспечивают управление хост-контроллером в целом, планирование загрузки кадров, связь контроллера с памятью и управление корневым хабом. Назначение регистров поясняется в табл. 15.4; положение полей в наиболее сложных регистрах изображено на рис. 15.9 и 15.10.

Таблица 15.4. Операционные регистры OHC

Смещение	Назначение
Регистры общего управления и состояния OHC	
00h	HcRevision — ревизия контроллера: биты [7:0] — номер версии в BCD-формате, бит 8 — признак поддержки эмуляции контроллера клавиатуры и мыши, остальные биты не используются
04h	HcControl — управление операционными режимами контроллера (рис. 15.9). Назначение полей: CBSR (ControlBulkServiceRatio) — соотношение числа обслуживаемых непустых ED для точек типов Control и Bulk: 0 — 1:1, 1 — 2:1, 2 — 3:1, 3 — 4:1, Определяет приоритет управляющих передач относительно передач массивов PLE (PeriodicListEnable) — разрешение обслуживания списка периодических транзакций (в следующем кадре) IE (IsochronousEnable) — разрешение обслуживания изохронных передач (в следующем кадре) CLE (ControlListEnable) — разрешение обслуживания управляющих передач (в следующем кадре) BLE (BulkListEnable) — разрешение обслуживания передач массивов (в следующем кадре) HCFS (HostControllerFunctionalState) — состояние хост-контроллера в плане USB: 00 — *Reset*, 01 — *Resume*, 10 — нормальный режим, 11 — *Suspend* IR (InterruptRouting) — маршрутизация прерываний (сигнализация событий, указанных в регистре HcInterruptStatus: 0 — нормальные прерывания, 1 — SMI RWC (RemoteWakeupConnected) — признак подключенности контроллера к системе «пробуждения» хоста RWE (RemoteWakeupEnable) — разрешение сигнализации «пробуждения» хоста по установке бита ResumeDetected в регистре HcInterruptStatus
08h	HcCommandStatus — управление и состояние контроллера (рис. 15.9). Назначение полей: HCR (HostControllerReset) — сброс контроллера (обнуляется контроллером по завершении сброса)

Смещение	Назначение
	CLF (ControlListFilled) — признак наличия дескрипторов в очереди управляющих передач (если бит обнулен, контроллер и не начнет обход списка ED точек управления). Устанавливается драйвером по помещении дескриптора в очередь, дальше отслеживается и модифицируется контроллером. BLF (BulkListFilled) — признак наличия дескрипторов в очереди передач массивов (аналогично CLF) OCR (OwnershipChangeRequest) — запрос передачи права управления контроллером SOC (SchedulingOverrunCount) — счетчик ошибок планирования (переполнений)
0Ch	HcInterruptStatus — идентификация событий, вызывающих прерывание (рис. 15.9). Назначение полей: SO (SchedulingOverrun) — признак переполнения при планирования кадра (транзакция не умещается до конца кадра) WDH (WritebackDoneHead) — признак помещения дескриптора в очередь выполненных заданий SF (StartofFrame) — признак начала кадра RD (ResumeDetected) — признак получения сигнала *Resume* UE (UnrecoverableError) — признак неисправимой ошибки, не связанной с USB (требуется сброс контроллера) FNO (FrameNumberOverflow) — переполнение счетчика кадров (перенос из 15-го разряда) RHSC (RootHubStatusChange) — признак смены состояния корневого хаба OC (OwnershipChange) — признак передачи права управления контроллером (вызывает SMI)
10h	HcInterruptEnable — разрешение прерываний по событиям. Назначение полей аналогично предыдущему, дополнительно имеется бит MIE (MasterInterruptEnable) — общее разрешение прерываний. При записи единичное значение каждого бита разрешает соответствующее условие, нулевое — игнорируется. Чтение возвращает текущее состояние (разрешенные условия)
14h	HcInterruptDisable — запрет прерываний по событиям. Назначение полей аналогично предыдущему. При записи единичное значение каждого бита запрещает соответствующее условие, нулевое — игнорируется. Чтение возвращает текущее состояние (разрешенные условия)
Указатели областей памяти	
18h	HcHCCA — физический адрес HCCA (биты [7:0]=0)
1Ch	HcPeriodCurrentED — текущий физический адрес дескриптора ED точки периодических передач (биты [3:0]=0), устанавливается контроллером, драйвер может только читать
20h	HcControlHeadED — физический адрес первого дескриптора точки управления (биты [3:0]=0), устанавливается драйвером
24h	HcControlCurrentED — физический адрес текущего дескриптора точки управления (биты [3:0]=0), устанавливается контроллером
28h	HcBulkHeadED — физический адрес первого дескриптора точки передач массивов (биты [3:0]=0), устанавливается драйвером
2Ch	HcBulkCurrentED — физический адрес текущего дескриптора точки передач массивов (биты [3:0]=0), устанавливается контроллером
30h	HcDoneHead — физический адрес начала очереди завершенных передач (биты [3:0]=0), устанавливается контроллером

продолжение

Таблица 15.4 (*продолжение*)

Смещение	Назначение
Регистры управления кадром	
34h	HcFmInterval — параметры кадра: Биты [13:0] — FI (FrameInterval), длительность кадра (в bt), по умолчанию 2EDFh (11999) Биты [30:16] — FSMPS (FSLargestDataPacket), значение максимального размера пакета данных, загружаемое в счетчик Data Packet Counter в начале каждого кадра (счетчик отражает длину пакета данных, допустимую в текущий момент времени без переполнения планирования) Бит 31 — FIT (FrameIntervalToggle), переключением этого бита драйвер сигнализирует о смене длительности кадра
38h	HcFmRemaining, остаток кадра: Биты [13:0] — FR (FrameRemaining), текущее количество битовых интервалов до конца кадра. Автоматически декрементируется; по обнулению загружается значением из FI и генерируется SOF; Бит 31 — FRT (FrameRemainingToggle), принимает значение FIT по обнулению FI
3Ch	HcFmNumber — номер кадра FN (биты [15:0]), автоматически меняется на границе кадра
40h	HcPeriodicStart — момент начала обхода списка периодических транзакций PS (биты [13:0]), обычно устанавливается на 10% меньше FI. Контроллер приступает к периодическим транзакциям, когда FR достигает значения PS
44h	HcLSThreshold — порог LST (биты [11:0]), до которого контроллер может выполнять LS-транзакции с максимальным (8 байт) размером пакета. Эти транзакции могут начинаться только при FR≥LST, по умолчанию LST=628h (1576)
Регистры управления и состояния корневого хаба	
48h	HcRhDescriptorA — дескриптор-A корневого хаба (рис. 15.10). Назначение полей: NDP (NumberDownstreamPorts) — число нисходящих портов (1–15) PSM (PowerSwitchingMode) — режим управления питанием портов: 1 — селективное управление, 0 — общее NPS (NoPowerSwitching) — признак отсутствия ключа подачи питания (1 — порты запитаны постоянно) DT (DeviceType) — тип устройства: всегда 0, поскольку корневой хаб не может быть компаундным устройством OCPM (OverCurrentProtectionMode) — режим сообщения о срабатывании токовой защиты питания портов: 0 — коллективное, 1 — индивидуальное. Действительно только при NOCP=0 NOCP (NoOverCurrentProtection) — признак невозможности сообщения о перегрузке POTPGT (PowerOnToPowerGoodTime) — время ожидания перед обращением драйвера к контроллеру после подачи питания на порт (в 2-мс интервалах)
4Ch	HcRhDescriptorB — дескриптор-B корневого хаба: Биты [15:0] — DR (DeviceRemovable), признаки отсоединяемости устройств от портов (бит 1 — порт 1,...бит 15 — порт 15) Биты [31:16] — PPCM (PortPowerControlMask), маски управления питанием портов: 0 — глобальное, 1 — селективное
50h	HcRhStatus — регистр состояния корневого хаба (рис. 15.10). Назначение полей:

Смещение	Назначение
	LPS (LocalPowerStatus) — чтение всегда дает 0, запись «1» выключает питание портов с глобальным управлением, запись «0» не имеет значения OCI (OverCurrentIndicator) — индикатор перегрузки для глобальной защиты по току DRWE (DeviceRemoteWakeupEnable) — запись «1» разрешает выход из состояния *Suspend* по смене состояния подключения портов, запись «0» не имеет значения LPSC (LocalPowerStatusChange) — чтение всегда дает 0, запись «1» включает питание портов с глобальным управлением, запись «0» не имеет значения OCIC (OverCurrentIndicatorChange) — запись «1» сбрасывает индикатор перегрузки для глобальной защиты, запись «0» не имеет значения CRWE (ClearRemoteWakeupEnable) — запись «1» запрещает выход из состояния Suspend по смене состояния подключения портов, запись «0» не имеет значения
54h	HcRhPortStatus[1], состояние первого порта корневого хаба (рис. 15.10). Назначение полей: CCS: чтение (CurrentConnectStatus) — текущее состояние подключения к порту: 1 — подключено устройство, 0 — нет; запись «1» запрещает порт, запись «0» не имеет эффекта PES (PortEnableStatus) — состояние разрешения порта: 0 — Disabled, 1 — Enabled; запись «1» разрешает порт, запись «0» не имеет эффекта PSS (PortSuspendStatus) — состояние приостановки порта: 0 — не приостановлен, 1 — приостановлен или находится в процессе возобновления; запись «1» приостанавливает порт, запись «0» не имеет эффекта POCI: чтение (PortOverCurrentIndicator) — перегрузка порта по питанию (только при селективной защите портов): 1 — защита сработала, 0 — нет; запись «1» инициирует возобновление порта (resume), запись «0» не имеет эффекта PRS (PortResetStatus) — чтение состояния сброса порта: 1 — подается сигнал Reset, 0 — нет; запись «1» инициирует подачу сигнала Reset, запись «0» не имеет эффекта. PPS (PortPowerStatus) — состояние питания порта: 0 — порт не запитан, 1 — порт запитан; запись «1» включает питание порта при селективном управлении, запись «0» не имеет эффекта LSDA (LowSpeedDeviceAttached) — признак низкой скорости: 1 — обнаружено подключение LS-устройства, 0 — нет CSC (ConnectStatusChange) — признак смены состояния подключения порта, сбрасывается записью «1» PESC (PortEnableStatusChange) — признак смены состояния разрешения порта, сбрасывается записью «1» PSSC (PortSuspendStatusChange) — признак завершения последовательности возобновления для порта, сбрасывается записью «1» OCIC (PortOverCurrentIndicatorChange) — признак смены состояния индикатора селективной защиты порта, сбрасывается записью «1» PRSC (PortResetStatusChange) — признак завершения формирования сигнала Reset, сбрасывается записью «1»
...	Состояние портов корневого хаба (аналогично предыдущему)
54h+4×NDP	HcRhPortStatus[NDP], состояние последнего порта корневого хаба (аналогично предыдущему)
Регистры поддержки эмуляции контроллера клавиатуры и мыши (KBC)	
100h	HceControl — управление эмуляцией (табл. 15.5)
104h	HceInput — эмуляция входных буферов KBC: программный вывод байта по адресу порта 60h или 64h вызывает запись данных в биты [7:0] этого регистра, биты [31:8] — резерв

продолжение

Таблица 15.4 (*продолжение*)

Смещение	Назначение
108h	`HceOutput` — эмуляция выходного буфера KBC: программный ввод байта из порта 60h вернет данные из бит [7:0] этого регистра (и сбросит флаг `BufferFull` в `HceStatus`), биты [31:8] — резерв
10Ch	`HceStatus` — эмуляция регистра состояния KBC (табл. 15.5)

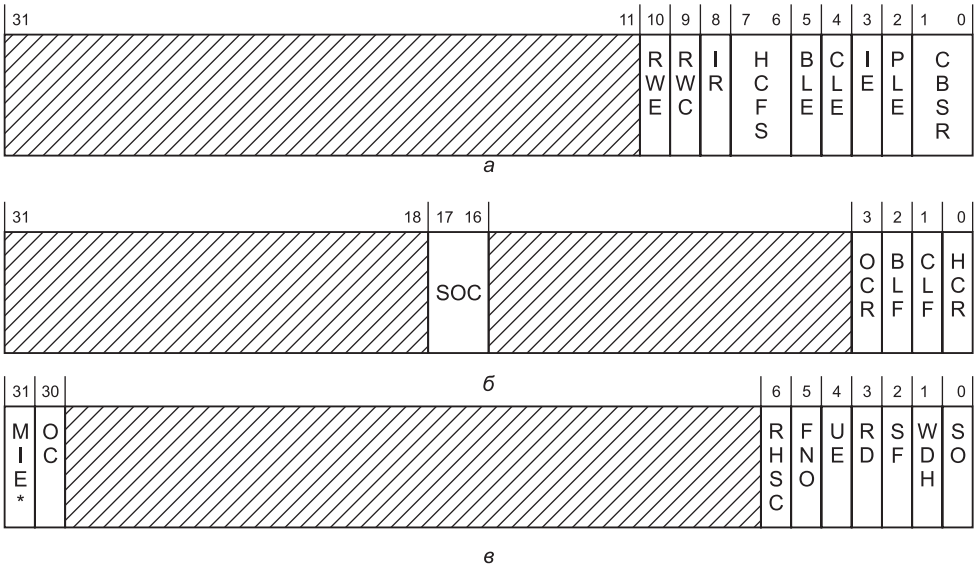


Рис. 15.9. Регистры общего управления и состояния OHC: *а* — HcControl; *б* — HcCommandStatus; *в* — HcInterruptStatus, HcInterruptEnable и HcInterruptDisable (* бит MIE — только в регистре HcInterruptEnable)

При разрешенной эмуляции контроллера клавиатуры и мыши контроллер OHC перехватывает (декодирует) обращения по адресам ввода/вывода 60h и 64h:

- инструкция `IN`, обращенная к порту 60h, вернет байт из регистра `HceOutput`, а также установит в регистре `HceStatus` флаг `OutputFull` = 0;
- инструкция `OUT`, обращенная к порту 60h, запишет байт в регистр `HceInput`, а также установит в регистре `HceStatus` флаги `InputFull` = 1 и `CmdData` = 0;
- инструкция `IN`, обращенная к порту 64h, вернет байт текущего состояния из регистра `HceStatus`;
- инструкция `OUT`, обращенная к порту 64h, запишет байт в регистр `HceInput`, а также установит в регистре `HceStatus` флаги `InputFull` = 0 и `CmdData` = 1.

ПО эмуляции отрабатывает эти обращения, пользуясь клавиатурой и мышью, подключенными к USB. Ряд обращений требует действий, которые эмулятор должен

выполнить невидимо для ПО. Эти действия вызывают условие прерывания для эмулятора, которое может выполняться как системное прерывание *SMI*. При этом оказывается, что с хост-контроллером должны взаимодействовать как нормальный драйвер ОС, так и драйвер эмулятора, работающий в режиме системного управления (*SMM*). Чтобы корректно передавать право на управление контроллером между этими драйверами, в регистрах `HcCommandStatus` и `HcInterruptStatus` предусмотрены соответствующие биты.

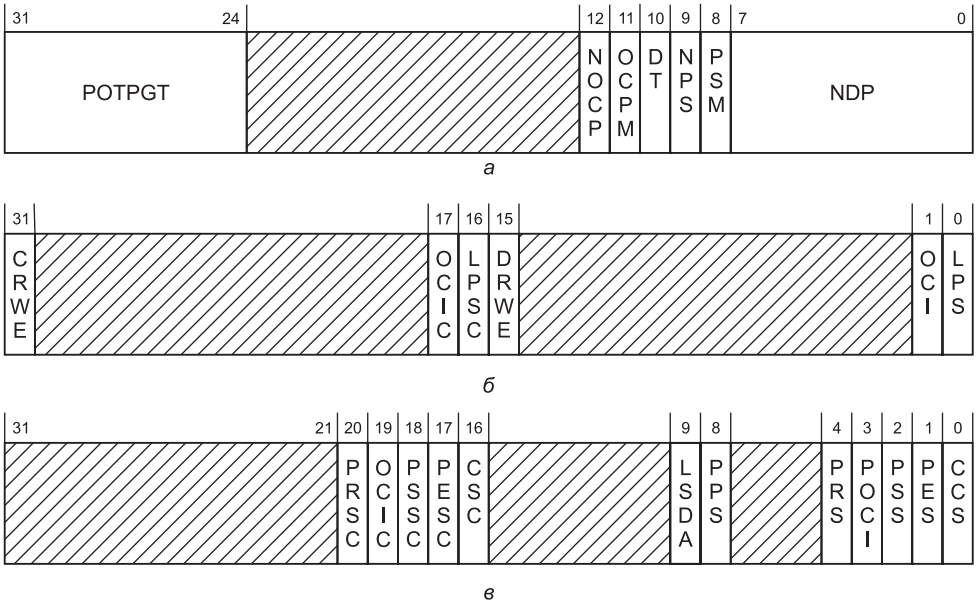


Рис. 15.10. Регистры корневого хаба: *а* — HcRhDescriptorA, *б* — HcRhStatus; *в* — HcRhPortStatus[i]

Таблица 15.5. Назначение бит регистров управления и состояния эмуляции

Бит	Назначение
Регистр HceControl	
0	`EmulationEnable`, разрешение эмуляции
1	`EmulationInterrupt`, признак условия прерывания для эмуляции
2	`CharacterPending`, разрешение прерывания эмуляции по обнулению бита `OutputFull` в регистре `HceStatus` (требуется следующий байт от клавиатуры или мыши)
3	`IRQEn`, разрешение генерации IRQ1 или IRQ12 по установке бита `OutputFull` в регистре `HceStatus`
4	`ExternalIRQEn`, разрешение формирования сигнала прерывания эмуляции по сигналу IRQ1 и IRQ12 от KBC (независимо от состояния EmulationEnable)

продолжение ⇩

Таблица 15.5 (*продолжение*)

Бит	Назначение
5	`GateA20Sequence`, признак команды управления вентилем *GateA20*, устанавливается по записи байта D1h в порт 64h, сбрасывается по записи любого другого значения в этот порт
6	`IRQ1Active`, признак сигнала IRQ1 от KBC (сбрасывается записью «1»)
7	`IRQ12Active`, признак сигнала IRQ12 от KBC (сбрасывается записью «1»)
8	`A20State`, индикатор состояния сигнала GateA20 от KBC
9–31	Резерв (0)
Регистр HceStatus	
0	`OutputFull`, выходной буфер контроллера полон. Сбрасывается при чтении порта 60h. При установке этого бита и при `IRQEn` = 1 вырабатывается `IRQ1` (при AuxOutputFull = 0) или `IRQ12` (при AuxOutputFull = 1)
1	`InputFull`, входной буфер полон. Устанавливается при записи в порт 60h или 64h (кроме команд управления вентилем GateA20)
2	`Flag`, системный флаг для определения «холодной» или «теплой» загрузки
3	`CmdData`, признак команды или данных для контроллера: запись в порт 60h сбрасывает бит, запись в порт 64h — устанавливает
4	`Inhibit Switch`, состояние ключа запрета клавиатуры (1 — клавиатура не запрещена)
5	`AuxOutputFull`, признак прихода данных от мыши
6	`Time-out`, признак тайм-аута при обмене с клавиатурой
7	`Parity`, признак ошибки четности при обмене с клавиатурой

«Расширенный» хост-контроллер — EHC

«Расширенный» хост-контроллер EHC (Enhanced Host Controller) был введен фирмой Intel для поддержки высокой скорости в USB 2.0. Его интерфейс — *EHCI* — описан в документе «Enhanced Host Controller Interface Specification for Universal Serial Bus», версия 1.0 опубликована в марте 2002 года. Контроллер EHC предназначен для работы с устройствами только на высокой скорости подключения к корневому хабу, при этом для устройств FS/LS, которые подключены через промежуточный хаб USB 2.0, контроллер EHC выполняет расщепленные транзакции (см. главу 11). С теми портами корневого хаба, к которым непосредственно подключены хабы и устройства USB 1.x, работает контроллер-компаньон (UHC или OHC). Коммутацию портов и контроллеров осуществляет маршрутизирующая логика, входящая в состав корневого хаба USB 2.0. Обнаружением подключения устройств к корневому хабу занимается драйвер EHC через регистры EHC. Обнаружив подключение FS/LS-устройства, драйвер перекоммутирует данный порт на контроллер-компаньон, и с этого момента порт отдается в ведение компаньону и его драйверу. Компаньон и его драйвер могут и не «знать» о том, что они работают в составе контроллера USB 2.0. Для портов, остающихся в ведении EHC, эмулиру-

ется внешний хаб — ПО манипулирует портами, используя стандартные запросы к хабам USB.

Контроллер EHC имеет конфигурационные регистры PCI, операционные регистры ввода/вывода, отображенные на пространство памяти (memory mapped I/O) и использует область системной памяти для взаимодействия с драйвером.

С точки зрения взаимодействия с драйвером EHC отчасти напоминает UHC, но высокая скорость передачи (480 Мбит/с) потребовала усиления интеллекта контроллера с целью уменьшения числа операций обмена между драйвером, памятью и контроллером. В EHC просматриваются многие идеи, заложенные в OHC. Структуры данных разработаны с учетом минимизации обращений к памяти. Все структуры должны размещаться в памяти так, чтобы они не пересекали границы четырехкилобайтных страниц памяти — это позволяет оптимизировать сосуществование OHC с виртуальной памятью, основанной на страничной переадресации, применяемой в процессорах x86.

В EHCI с точки зрения планирования транзакций передачи делятся на *периодические* (изохронные и прерывания) и *асинхронные* (управляющие передачи и массивы). Каждый из этих двух планов реализуется по-своему и может быть включен в работу и выключен. Контроллер начинает каждый микрокадр с выполнения периодических передач (если они разрешены), оставшееся от них время выделяется для выполнения асинхронных передач (аналогично UHC, см. рис. 15.1, *а*). За то, чтобы в микрокадре оставалось время для асинхронных передач, отвечает драйвер. Хост-контроллер аппаратно следит лишь за тем, чтобы транзакции не пересекали границу микрокадра: если контроллер «видит», что транзакция может не успеть завершиться к моменту *EOF1*, он ее не начнет. При этом возможна перестраховка, поскольку точное время исполнения транзакции заранее не известно (неизвестно, сколько придется вставить бит и каковы задержки в кабелях, хабах и устройстве).

Для всех передач с гарантированной доставкой (прерывания, управление и массивы) используются структуры данных qTD (Queue Element Transfer Descriptor), описывающие очереди буферов, обеспечивающие автоматическое упорядоченное исполнение потоков передач. В EHC под передачей понимается последовательность однотипных транзакций[1]; ограничен лишь суммарный размер передаваемых блоков (20 Кбайт). Транзакции управления хост планирует как последовательность двух-трех «передач» (в терминах EHC). Драйвер может динамически (во время исполнения плана) добавлять новые передачи в очереди. Контроллер аппаратно поддерживает сигнализацию окончания блоков короткими пакетами: приняв короткий пакет, контроллер может пойти по альтернативной последовательности передач для данной очереди (то есть организуется условный переход). Для изохронных передач используются специальные структуры данных (iTD — Isochronous Transaction Descriptors для HS и siTD — Split-transaction IsochronousTransfer Descriptors для расщепления транзакций с FS-устройствами). Для изохронных передач на HS дескриптор может описывать передачу до 24 Кбайт данных, на FS — до 1023 байт.

[1] По сравнению с UHC это уже некоторый прогресс.

Основой *планирования периодических транзакций* является *список кадров* (`Frame List`) на 1024, 512 или 256 вхождений. Базовый адрес и длина списка устанавливается программно, текущий элемент списка выбирается по счетчику *кадров*. Исполнение плана начинается в каждом *микрокадре*, таким образом, каждый текущий элемент списка выбирается 8 раз подряд, после чего контроллер переходит к следующему элементу. Элемент списка может указывать на `iTD`, `siTD` или заголовок очереди (`QH`), относящейся к прерываниям. Кроме того, он может указывать на специальные структуры (`FSTN`), используемые для обеспечения корректности отработки расщепленных транзакций около границы кадра. В элементе списка кроме собственно указателя имеется идентификатор типа структуры (`Typ`), на которую ссылается указатель (`iTD`, `siTD`, `QH` или `FSTN`), а также признак заглушки-терминатора (`T`). Все дескрипторы изохронных передач и заголовки очередей имеют «горизонтальный» указатель на следующую структуру, в котором также задается и тип (`Typ`) этой структуры, и признак заглушки-терминатора (`T`). Цепочка дескрипторов и заголовков очередей, начинающаяся от списка кадров, должна завершаться дескриптором (или заголовком), у которого установлен признак заглушки-терминатора (`T`). Только отработав такой дескриптор (или заголовок), контроллер приступает к исполнению плана асинхронных передач.

Для упрощения планирования расщепленных транзакций (они не должны пересекать границу кадра) контроллер организует фазовый сдвиг между *кадрами шины* (*B-Frame*), которые видны хабам и устройствам по факту смены номера кадра в маркере `SOF`, и *кадрами хоста* (*H-Frame*), которыми оперирует драйвер при построении планов и по которым выбираются периодические транзакции из списка кадров. Кадры шины отстают от кадров хоста на один микрокадр, более подробные пояснения (но не мотивы) приведены в спецификации EHCI. Для обслуживания расщепленных периодических транзакций имеется специальная структура (узел) `FSTN`, содержащая пару указателей: нормальный, обеспечивающий переход к следующей структуре (`iTD`, `siTD`, `QH` или `FSTN`), и обратный, который может указывать только на заголовок очереди. Нюансы планирования расщепленных транзакций здесь приводить не будем, с ними можно ознакомиться в спецификации EHCI.

Основой планирования непериодических транзакций является *асинхронный список* (asynchronous list), представляющий собой кольцо из заголовков очередей. В OHC регистр `AsyncListAddr` указывает на текущий элемент списка; к отработке этого элемента контроллер приступает, завершив отработку периодических передач в данном микрокадре (или сразу, если периодические передачи запрещены или отсутствуют). Далее, по мере отработки очередей, контроллер заносит в этот регистр адреса последующих указателей. Таким образом, обслуживание всех асинхронных очередей выполняется по кругу, без привязки к конкретным кадрам. Контроллер останавливает обход асинхронного списка, когда обнаруживает опустошение всех его очередей, для возобновления обхода требуется вмешательство драйвера. Нормальной дисциплиной обслуживания очередей является отработка одной шинной транзакции из очереди, после чего контроллер переходит к следующей очереди. Возможен и специальный *режим парковки* (Asynchronous Schedule

Park Mode), в котором контроллеру разрешается выполнять подряд по несколько транзакций из одной очереди. Режим парковки распространяется на все очереди высокоскоростных асинхронных передач.

Дескриптор `iTD` описывает изохронную передачу, которая может выполняться за 1–8 *этапов* (микрокадров, в которых происходит обращение к данному дескриптору). Каждому этапу в дескрипторе соответствует своя *запись* (transaction record), управляющая выполнением и отражающая состояние транзакции (активность, ошибки выполнения, необходимость прерывания по выполнению, реальная длина) и содержащая указатель на буфер данных. Каждый этап может выполняться за 1–3 транзакции в микрокадре (точка может быть широкополосной). Дескриптор содержит и описание конечной точки: адрес устройства и точки, направление, размер пакета. Контроллер формирует транзакции исходя из указанного размера пакета. Буферы для данных могут располагаться в разных физических страницах памяти, но логически они должны представлять собой непрерывную область в виртуальной памяти. Для хранения данных (максимум 8 этапов по три транзакции по 1024 байт — 24 576 байт) может потребоваться до 7 страниц по 4 кбайт; для всех этих страниц в дескрипторе имеются соответствующие указатели.

Дескриптор `siTD` описывает одну расщепленную транзакцию. Адресная часть содержит номер и направление конечной точки, адрес устройства, а также адрес и номер порта хаба, выполняющего трансляцию данной транзакции. В дескрипторе имеются поля битовых масок `μFrame_S-mask` и `μFrame_C-mask`, определяющих, в каких микрокадрах данного кадра должны выполняться транзакции `SS` и `CS` соответственно (см. главу 14). Контроллер в дескрипторе отмечает микрокадры, в которых в действительности происходили транзакции `CS`. Дескриптор имеет обычный набор полей, управляющих выполнением и отражающих состояние транзакции (активность, ошибки выполнения, необходимость прерывания по выполнению, реальная длина). Кроме того, в `siTD` имеются специфические поля, управляющие текущей фазой (`SS` или `CS`), а также признак специфической ошибки расщепленной транзакции — пропуска микрокадра, в котором должна выполняться очередная транзакция `CS`. Этот пропуск может случиться, если контроллер не выпустит текущую транзакцию из-за нехватки времени в микрокадре. Блок передаваемых данных (до 1023 байт) может располагаться в одной или двух физических страницах памяти, и в дескрипторе для них имеются необходимые указатели. В `siTD` имеется специфический элемент — обратный указатель (`Back Pointer`) на `siTD` предыдущего кадра, который используется при планировании транзакций *IN*, завершающихся близко к границе кадра.

Элемент очереди-дескриптор передачи `qTD`, описывает одну передачу размером до 20 480 байт. Дескриптор привязан к своему заголовку очереди (`QH`); он содержит пару указателей на следующие элементы данной очереди:

- основной, указывающий на дескриптор следующей передачи, которую требуется выполнить после нормального завершения текущей;
- альтернативный, указывающий на дескриптор передачи, которую требуется выполнить в случае завершения текущей передачи по приему короткого пакета.

Дескриптор имеет обычный набор полей, управляющих выполнением и отражающих состояние транзакции: активность, ошибки выполнения, необходимость прерывания по выполнению, используемый маркер (*IN*, *OUT* или *SETUP*). В дескрипторе указывается общая длина передачи. Буфер данных для передачи должен располагаться в непрерывной области виртуальной памяти; для описания буфера передачи максимальной длины имеется массив из пяти указателей физических страниц.

Заголовок очереди `QH` создается для каждой сконфигурированной неизохронной конечной точки каждого устройства USB. Заголовки очередей непериодических конечных точек связаны между собой по горизонтали в кольцо, для чего в каждом заголовке имеется соответствующий указатель. Заголовок очереди несет исчерпывающее описание конечной точки: номер и направление точки, максимальный размер пакета, число пакетов в микрокадре (для широкополосных точек), адрес устройства, его скорость. Для FS/LS устройств имеется и информация для выполнения расщепленных транзакций: номер хаба и порта, выполняющего расщепление транзакций, маски микрокадров для транзакций `SS` и `CS`. В заголовке очереди имеется оверлейная область, в которую контроллер помещает необходимые ему поля `qTD` текущей транзакции. Продвижение по очереди осуществляет контроллер, помещая в оверлей следующий `qTD` после завершения отработки предыдущего.

Контроллер EHC вырабатывает прерывания для разных категорий событий, и категории могут быть селективно замаскированы:

- по завершении передачи, в дескрипторе которой имеется соответствующий признак, а также по приему короткого пакета. Эти прерывания могут быть задержаны по времени до определенного задаваемого программно порога, что позволяет снизить частоту запросов прерывания от EHC. Без задержки частота запросов может достигать частоты микрокадров (для IBM PC это слишком часто); с задержкой они не смогут появляться быстрее, чем определяет значение порога;
- по событию хост-контроллера: оборот по списку кадров, изменению состояния или перегрузке портов хаба, специальному разрешению программного изменения последовательности заголовков очередей, по ошибке системного подключения (переполнение или переопустошение буфера FIFO из-за занятости шины PCI).

Постановка запросов передач в очереди, как и включение изохронных передач в план, а также добавление/удаление очередей может выполняться драйвером динамически, во время работы хост-контроллера. Однако для сохранения целостности и связанности структур программа должна соблюдать определенные правила взаимодействия, чтобы не пытаться изменять структуры, которые в данный момент отрабатываются контроллером. Для этой синхронизации хост-контроллер использует специальные биты-признаки в своем регистре состояния и в структурах данных. Для «сбора урожая» — поиска отработанных передач — драйверу приходится просматривать во всех дескрипторах передач признаки активности. Такого сервиса, как очередь исполненных передач (как в OHC), контроллер EHC не

предоставляет. Но по сравнению с UHC, конечно же, объем работ драйвера EHC сокращается, поскольку этот контроллер оперирует передачами, а не транзакциями. Однако у драйвера EHC появляется дополнительная довольно сложная задача — планирование расщепленных транзакций.

Структуры данных и регистры EHC

Контроллер EHC имеет несколько наборов регистров:

- *конфигурационные регистры PCI* — стандартный заголовок и специфические регистры, приведенные в табл. 15.6. В заголовке для EHC указывается код класса 0Ch, подкласс 03h, интерфейс 20h;
- *регистры описания структуры контроллера*, отображенные на память (табл. 15.7); на их положение указывает BAR в заголовке конфигурационного пространства;
- *операционные регистры EHC* (32-битные), отображенные на память (табл. 15.8); они расположены вслед за предыдущим набором.

Таблица 15.6. Специфические регистры конфигурационного пространства EHC

Смещение (длина)	Назначение
60h (8 бит)	SBRN (Serial Bus Release Number) — версия шины USB (20h)
61h (8 бит)	FLADJ (Frame Length Adjustment Register) — регистр подстройки длительности кадра, используются биты [5:0]. Длительность кадра в битовых интервалах HS определяется по формуле 59488+16× FLADJ, по умолчанию FLADJ=20h (60 000 bt)
62-63h (16 бит)	PORTWAKECAP, регистр возможности генерации событий пробуждения для портов корневого хаба. Бит 0 — признак наличия данного регистра (0 — нет регистра), биты [15:1] — маски для портов с теми же номерами. На работу EHC регистр не влияет (это только информация для драйвера)
EECP+0h (32 бита)	USBLEGSUP (USB Legacy Support), регистр возможностей поддержки эмуляции старых устройств, адрес определяется в поле EECP регистра HCCPARAMS, отображенного на память. Назначение бит: Биты 31:25 — резерв; Бит 24 — HC OS Owned Semaphore, семафор запроса управления контроллером. ОС устанавливает в «1» как запрос, право считается предоставленным, когда BIOS установит в «0» бит 16; Биты [23:17] — резерв; Бит 16 — HC BIOS Owned Semaphore, семафор права управления контроллером. BIOS устанавливает в «1» как признак владения контроллером; Бит [15:8] — Next EHCI Extended Capability Pointer, указатель на следующий идентификатор расширенных возможностей; [7:0] Capability ID, идентификатор поддержки старых устройств (01h)
EECP+4h (32 бита)	USBLEGCTLSTS (USB Legacy Support Control and Status), регистр управления и состояния эмуляции. BIOS использует этот регистр для разрешения SMI по различным событиям и идентификации событий. Назначение бит: Бит 31 — SMI on BAR, прерывание SMI по смене базового адреса регистров EHC

продолжение ⇩

Таблица 15.6 (*продолжение*)

Смещение (длина)	Назначение
	Бит 30 — SMI on PCI Command, прерывание SMI по записи в регистр команд конфигурационного пространства устройства PCI Бит 29 – SMI on OS Ownership Change, прерывание SMI по смене состояния права управления для ОС Биты [28:22] — резерв Бит 21 — SMI on Async Advance, прерывание SMI по установке бита Async_Advance в регистре USBSTS Бит 20 — SMI on Host System Error, прерывание SMI по системной ошибке EHC Бит 19 — SMI on Frame List Rollover, прерывание SMI по полному обороту списка кадров Бит 18 — SMI on Port Change Detect, прерывание SMI по смене состояния порта Бит 17 — SMI on USB Error, прерывание SMI по ошибке USB Бит 16 — SMI on USB Complete, прерывание SMI по завершению передачи Бит 15 — SMI on BAR Enable, разрешение прерывания SMI по смене базового адреса регистров EHC Бит 14 — SMI on PCI Command Enable, разрешение прерывания SMI по записи в регистр команд конфигурационного пространства устройства PCI Бит 13 — SMI on OS Ownership Change Enable, разрешение прерывания SMI по смене состояния права управления для ОС Биты [12:6] — резерв Бит 5 — SMI on Async Advance Enable, разрешение прерывания SMI по установке бита Async_Advance в регистре USBSTS Бит 4 — SMI on Host System Error Enable, разрешение прерывания SMI по системной ошибке EHC Бит 3 — SMI on Frame List Rollover Enable, разрешение прерывания SMI по полному обороту списка кадров Бит 2 — SMI on Port Change Enable, разрешение прерывания SMI по смене состояния порта Бит 1 — SMI on USB Error Enable, разрешение прерывания SMI по ошибке USB Бит 0 — USB SMI Enable, разрешение прерывания SMI по завершению передачи

Таблица 15.7. Регистры описания структуры контроллера EHC

Смещение (длина)	Назначение
00h (8 бит)	CAPLENGTH, длина набора регистров описания (определяет положение операционных регистров)
01h (8 бит)	Резерв
02h (16 бит)	HCIVERSION, номер версии интерфейса хост-контроллера (0100h)
04h (32 бита)	HCSPARAMS, параметры структуры: Биты [31:24] — резерв Биты [23:20] — Debug Port Number, номер порта, являющегося отладочным (0 — нет такого) Биты 19:17 — резерв

Смещение (длина)	Назначение
	Бит 16 — P_INDICATOR, признак поддержки управления индикаторами портов Биты [15:12] — N_CC, число контроллеров-компаньонов (если 0, то к контроллеру можно подключать только HS-устройства) Биты [11:8] — N_PCC, число портов у каждого контроллера-компаньона Бит 7 — Port Routing Rules, правило сопоставления портов контроллерам-компаньоном: 0 — порты распределяются группами по N_PCC по контроллерам с нарастающими номерами, 1 — распределение в соответствии с таблицей HCSP-PORTROUTE Биты 6:5 — резерв Бит 4 — PPC (Port Power Control), признак наличия управления питанием портов Биты [3:0] — N_PORTS, число нисходящих портов (1–Fh)
08h (32 бита)	HCCPARAMS, параметры свойств: Биты [31:16] — резерв. Биты [15:8] — EECP (EHCI Extended Capabilities Pointer), указатель на положение регистра USBLEGSUP в конфигурационном пространстве Биты [7:4] — Isochronous Scheduling Threshold, порог планирования изохронных передач. Если бит 7=0, то биты [6:4] определяют минимальную дистанцию (число микрокадров) от текущей позиции, на которой драйвер может менять дескрипторы изохронных передач. Единичное значение бита 7 означает, что контроллер способен удержать в своем кэше дескрипторы для целого кадра Бит 3 — резерв Бит 2 — Asynchronous Schedule Park Capability, поддержка режима парковки для заголовков HS-очередей асинхронного плана Бит 1 — Programmable Frame List Flag, признак поддержки программируемого размера списка кадров (0 — размер только 1024 элемента) Бит 0 — 64-bit Addressing Capability, способность использования 64-битной адресации памяти
0Ch (64 бита)	HCSP-PORTROUTE, описание распределения портов по контроллерам-компаньонам, массив 4-байтных номеров контроллера-компаньона для каждого нисходящего порта корневого хаба. Вмещает описания для 15 портов (биты [3:0], формально относящиеся к нулевому порту, не используются)

Таблица 15.8. Операционные регистры контроллера EHC

Смещение (длина)	Назначение
00h	USBCMD, регистр команд USB Биты [31:24] — резерв Биты [23:16] — Interrupt Threshold Control, минимальный интервал (в микрокадрах) генерации прерываний (125 мкс–64 мс). Допустимы значения 2^N, по умолчанию 08h (1 мс) Биты [15:12] — резерв Бит 11 — Asynchronous Schedule Park Mode Enable, разрешение режима парковки для заголовков HS-очередей асинхронного плана (необязательно) Бит 10 — резерв

продолжение ⇩

Таблица 15.8 (*продолжение*)

Смещение (длина)	Назначение
	Биты 9:8 — Asynchronous Schedule Park Mode Count, число успешных транзакций из HS-очереди, которые контроллер может выполнять до продолжения прохода по асинхронному плану Бит 7 — Light Host Controller Reset, сброс хост-контроллера без затрагивания портов, относящихся к компаньонам (необязательно) Бит 6 — Interrupt on Async Advance Doorbell, программный запрос контроллеру на выработку прерывания по очередному продвижению по асинхронному плану Бит 5 — Asynchronous Schedule Enable, разрешение исполнения асинхронного плана Бит 4 — Periodic Schedule Enable, разрешение исполнения периодического плана Биты 3:2 — Frame List Size, размер списка кадров: 00 — 1024 элемента, 01 — 512, 10 — 256, 11 — резерв Бит 1 — HCRESET (Host Controller Reset), программный сброс контроллера (аналогично аппаратному), бит обнуляется контроллером по завершении сброса Бит 0 — RS (Run/Stop), пуск/останов исполнения транзакций. После обнуления контроллер должен через 16 микрокадров остановить все транзакции
04h	USBSTS, регистр состояния USB Биты [31:16] — резерв Бит 15 — Asynchronous Schedule Status, реальное текущее состояние разрешения асинхронного плана (может отставать от команды смены) Бит 14 — Periodic Schedule Status, реальное текущее состояние разрешения периодического плана (может отставать от команды смены) Бит 13 — Reclamation, признак не пустого асинхронного плана. Обнуляется, когда контроллер встречает заголовок очереди с установленным флагом H, устанавливается в «1» при исполнении любой транзакции асинхронного плана. Если при обнуленном бите Reclamation контроллер встречает заголовок очереди с флагом H, он останавливает обход асинхронного плана Бит 12 — HCHalted, состояния останова контроллера Биты [11:6] — резерв Бит 5 — Interrupt on Async Advance, признак прерывания по продвижению асинхронного плана Бит 4 — Host System Error, признак системной ошибки контроллера (как устройства PCI) Бит 3 — Frame List Rollover, признак оборота по списку кадров Бит 2 — Port Change Detect, обнаружена смена состояния порта Бит 1 — USBERRINT (USB Error Interrupt), признак завершения транзакции по ошибке USB Бит 0 —USBINT (USB Interrupt), признак завершения транзакции, для которой предписана генерация прерывания
08h	USBINTR, регистр разрешения прерываний USB: Биты [31:6] — резерв Бит 5 — Interrupt on Async Advance Enable, разрешение прерывания по продвижению асинхронного плана Бит 4 — Host System Error Enable, разрешение прерывания по системной ошибке контроллера Бит 3 — Frame List Rollover Enable, разрешение прерывания по обороту списка кадров

Смещение (длина)	Назначение
	Бит 2 — Port Change Interrupt Enable, разрешение прерывания по смене состояния порта Бит 1 — USB Error Interrupt Enable, разрешение прерывания по ошибке USB Бит 0 — USB Interrupt Enable, разрешение прерывания по выполнению транзакций
0Ch	FRINDEX, индекс кадра. Инкрементируется с каждым микрокадром, биты [N:3] используются как текущий индекс в списке кадров
10h	CTRLDSSEGMENT, старшие биты 64-битного адреса (адрес 4G-сегмента, содержащего все структуры данных и регистры EHC)
14h	PERIODICLISTBASE, базовый адрес списка кадров (биты [11:0] нулевые — список должен быть выровнен по границе станицы 4 Кб)
18h	ASYNCLISTADDR, очередной адрес асинхронного списка, указывает на следующий заголовок очереди в асинхронном плане (биты [4:0] — резерв)
1C-3Fh	Резерв
40h	CONFIGFLAG, флаг конфигурирования: бит 0 устанавливается в 1, когда ПО завершает конфигурирование EHC (по этому флагу включается управление маршрутизацией портов от EHC, при нулевом значении порты безусловно подключены к своим контроллерам-компаньонам). Биты [31:1] не используются
44h... ...40h+4×n	PORTSC(n), управление и состояние n-го порта корневого хаба: Биты [31:23] — резерв Бит 22 — WKOC_E (Wake on Over-current Enable) — разрешение реакции на перегрузку по току как на пробуждающее событие Бит 21 — WKDSCNNT_E (Wake on Disconnect Enable) — разрешение пробуждения по отключению Бит 20 —WKCNNT_E (Wake on Connect Enable) — разрешение пробуждения по подключению устройства Биты [19:16] — Port Test Control, управление режимом тестирования: 0000 — рабочий режим, 0001 — *Test J_STATE*, 0010 — *Test K_STATE*, 0011 — *Test SE0_NAK*, 0100 — *Test Packet*, 0101 — *Test FORCE_ENABLE* Биты 15:14 — Port Indicator Control, управление индикатором порта: 00 — отключен, 01 — янтарный, 10 — зеленый, 11 — неопределено Бит 13 — Port Owner, признак владельца порта: 1 — порт подключен к контроллеру-компаньону (определено подключение устройства не HS или EHC не сконфигурирован) Бит 12 — PP (Port Power), питание порта (если PPC=1): 0 — выключено, 1 — включено Биты [11:10] — Line Status, текущее состояние логических уровней на линиях D+ (бит 11) и D- (бит 10) Бит 9 — резерв Бит 8 — Port Reset, сбос порта, устанавливается программно, сбрасывается контроллером Бит 7 — Suspend, приостановка порта Бит 6 — Force Port Resume, возобновление порта Бит 5 — Over-current Change, признак срабатывания токовой защиты (сбрасывается записью «1») Бит 4 — Over-current Active, состояние защиты (1 — питание отключено по перегрузке)

продолжение ⇩

Таблица 15.8 (*продолжение*)

Смещение (длина)	Назначение
	Бит 3 — `Port Enable/Disable Change`, признак отключения порта по ошибке, зафиксированной хабом (сбрасывается записью «1») Бит 2 — `Port Enabled/Disabled`, разрешение-запрет порта (программно можно только запретить, разрешается по сбросу) Бит 1 — `Connect Status Change`, признак смены состояния подключения Бит 0 — `Current Connect Status`, текущее состояние подключения

Формат *элемента списка кадров* (Frame List Element Pointer) приведен на рис. 15.11. Здесь `Frame List Link Pointer` — указатель на дескриптор изохронной передачи или заголовок очереди (для прерываний); если бит `T` = 1 (Terminate), то указатель не используется. Поле `Typ` описывает тип структуры, на которую ссылается указатель: 00 — `iTD`, 01 — `QH`, 10 — `siTD`, 11 — `FSTN`.

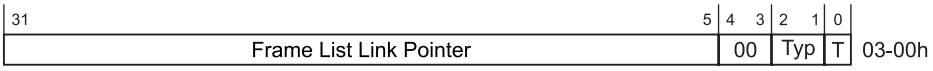


Рис. 15.11. Формат элемента списка кадров для EHC

Дескриптор изохронной передачи `iTD` (Isochronous Transfer Descriptor) относится к точкам HS-устройств, его формат приведен на рис. 15.12. Первое двойное слово по формату совпадает с элементом списка кадров. Далее следует 8-элементный список дескрипторов транзакций, выполняемых в каждом из восьми микрокадров (серым цветом выделены поля, модифицируемые хост-контроллером). В следующих семи двойных словах содержатся адреса физических страниц, в которых может располагаться буфер для транзакций, и описание конечной точки. В этом описании:

- поле `Status` отражает состояние выполнения транзакции:
  - бит 31 — `Active`, активность, устанавливается драйвером как признак необходимости исполнения, сбрасывается контроллером по исполнении транзакции;
  - бит 30 — `Data Buffer Error`, ошибка буфера данных (несвоевременность доставки данных в/из памяти);
  - бит 29 — `Babble Detected`, «болтливость», обнаруженная при исполнении транзакции;
  - бит 28 — `Transaction Error` (`XactErr`), ошибка USB при выполнении транзакции (только для транзакций *IN*).
- поле `Transaction X Length` задает число переданных байтов (0–3072). Для транзакций *IN* драйвер устанавливает ожидаемое число, контроллер его меняет на реальное число принятых байтов;
- бит `IOC` (`Interrupt On Complete`) заказывает прерывание по исполнению;
- поле `PG` (`Page Select`) задает номер страницы буфера (0–6), из которого берутся старшие (31:12) биты адреса для формирования стартового адреса буфера данной транзакции;

- поле `Transaction X Offset` задает смещение начала буфера X-транзакции относительно выбранной страницы;
- поля `Buffer Pointer (Page n)` задают физические адреса страниц 0–7;
- поле `EndPt` задает номер конечной точки;
- поле `I/O` задает направление: 0 — *OUT*, 1 — *IN*;
- поле `Device Address` задает адрес устройства;
- поле `Maximum Packet Size` — максимальный размер пакета для точки (1–1024), в транзакции *IN* используется как порог для определения «болтливости» устройства;
- поле `Mult` задает число транзакций за один микрокадр (00 — резерв).

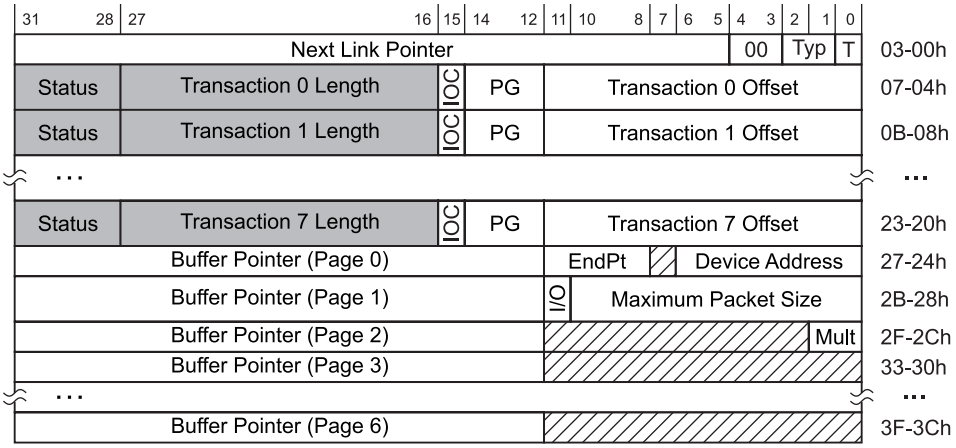


Рис. 15.12. Формат дескриптора изохронной передачи EHC

Дескриптор расщепленной изохронной транзакции `siTD` (Split-transaction Isochronous Transfer Descriptor) относится к точкам FS-устройств, его формат приведен на рис. 15.13. Первое двойное слово по формату совпадает с элементом списка кадров. Поля `I/O`, `EndPt`, `Device Address` описывают координаты и направление конечной точки.

Поля `Hub Addr` и `Port Number` задают адрес расщепляющего транзакцию хаба и номер порта.

Битовые поля `μFrame S-mask` и `μFrame C-mask` задают маски для микрокадров, в которых планируются транзакции *SS* и *CS* (транзакция в микрокадре выполняется при единичном значении маски). В поле `μFrame C-prog-mask` контроллер отмечает кадры, в которых произошли транзакции завершения.

Адрес буфера для начала транзакции задается полями `Buffer Pointer (Page 0)` и `Current offset`, при переходе на другую физическую станицу в качестве ее адреса используется поле `Buffer Pointer (Page 1)`. Бит `P` определяет номер используемой страницы. Ожидаемый размер пакета задается в поле `Total Bytes to Transfer`,

контроллер этот размер заменит реальным. Бит IOC (Interrupt On Complete) заказывает прерывание по исполнении.

Поле TP (Transaction Position) — позиция текущей HS-транзакции: 00 — *All*, HS-транзакция содержит все данные FS-транзакции (не более 188 байт), 01 — *Begin*, первый пакет для FS-транзакции, 10 —*Mid*, промежуточный, 11 — *End*, последний HS-пакет для FS-транзакции.

Поле T-Count (Transaction Count), число HS-транзакций, необходимых для выполнения FS-транзакции (1–6).

Состояние выполнения транзакции определяется полем Status:

- бит 7 — Active, активность, устанавливается драйвером как признак необходимости исполнения, сбрасывается контроллером по исполнении или по ошибке;
- бит 6 — ERR, признак получения одноименного ответа от транслятора транзакций;
- бит 5 — Data Buffer Error, ошибка буфера данных (несвоевременность доставки данных в/из памяти);
- бит 4 — Babble Detected, «болтливость», обнаруженная при исполнении транзакции;
- бит 3 — Transaction Error (XactErr), ошибка USB при выполнении транзакции (только для транзакций *IN*);
- бит 2 — Missed Micro-Frame, пропуск микрокадра (по вине контроллера), в котором должно было быть завершение;
- бит 1 — SplitXstate (Split Transaction State), состояние (фаза) расщепленной транзакции: 0 — *SS*, 1 — *CS*;
- бит 0 — резерв.

Поле si-TD Back Pointer является обратным указателем на декриптор si-TD (если в том же двойном слове бит T = 1, указатель не используется).

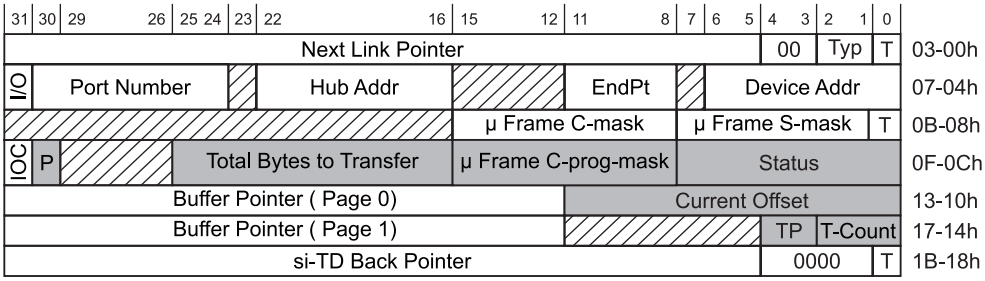


Рис. 15.13. Формат дескриптора расщепленной изохронной передачи EHC

Дескриптор передачи — элемент очереди qTD (Queue Element Transfer Descriptor) имеет формат, приведенный на рис. 15.14. Поле Next qTD Pointer указывает на следующий qTD (если T = 0), к которому следует перейти после нормальной отработки передачи. Поле Alternate Next qTD Pointer позволяет указать на qTD,

к которому следует перейти в случае приема короткого пакета. Ожидаемая длина передачи задается полем Total Bytes to Transfer, по окончании в этом поле окажется реальная длина. Адрес буфера для начала транзакции задается полями Buffer Pointer (Page 0) и Current offset, по мере продвижения контроллер меняет значение поля C_Page, определяющего номер физической страницы. Бит IOC задает прерывание по выполнению. Поле PID задает тип маркера в транзакциях: 00 — *OUT*, 01 — *IN*, 10 — *SETUP*, 11 — резерв. Бит DT — текущее значение Data Toggle для данной передачи, В поле CErr драйвер заносит допустимое число повторов (из-за ошибок) в каждой транзакции данной передачи (0 — число повторов неограниченно). Поле Status отражает состояние текущей транзакции передачи:

- бит 7 — Active, активность, устанавливается драйвером как признак необходимости исполнения, сбрасывается контроллером по исполнении или по достижении лимита повторов;
- бит 6 — Halted — признак получения ответа *STALL*;
- бит 5 — Data Buffer Error, ошибка буфера данных (несвоевременность доставки данных в/из памяти);
- бит 4 — Babble Detected, «болтливость», обнаруженная при исполнении транзакции;
- бит 3 — Transaction Error (XactErr), ошибка USB при выполнении транзакции (только для транзакций *IN*);
- бит 2 — Missed Micro-Frame, пропуск микрокадра (по вине контроллера), в котором должно было быть завершение (только для FS- и LS-точек);
- бит 1 — SplitXstate (Split Transaction State), состояние (фаза) расщепленной транзакции (только для FS- и LS-точек): 0 — *SS*, 1 — *CS*;
- бит 0 — P/ERR, для HS-точек при PID Code = 00 (*OUT*) — состояние протокола Ping (Ping State): 0 — выполнять транзакцию *OUT*, 1 — *PING*. Для не-HS точек — ERR, признак получения подтверждения *ERR* на расщепленную периодическую транзакцию.

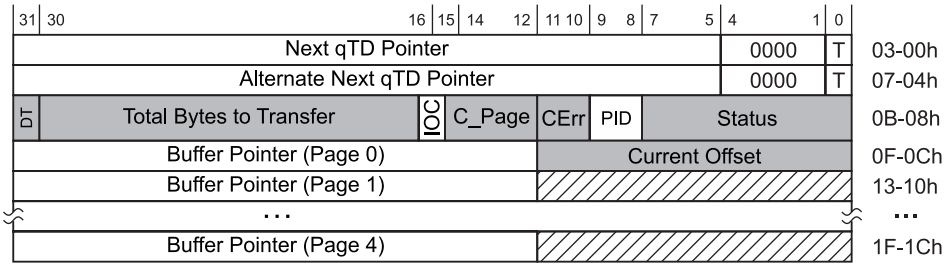


Рис. 15.14. Формат дескриптора передачи элемента очереди (qTD) для EHC

Заголовок очереди QH имеет формат, приведенный на рис. 15.15. Здесь Queue Head Horizontal Link Pointer указывает на следующую структуру по горизонтали, которая может быть заголовком очереди или любым дескриптором изохронной передачи. В последующих двух двойных словах описывается конечная точка, а для

точек LS/FS еще и дополнительные параметры, требуемые для расщепления транзакций. Поля `Device Address`, `EndPt` и `Maximum Packet Length` задают адрес устройства, номер точки и максимальный размер пакета. Поле `EPS` задает скорость: 00 — LS, 01 — FS, 10 — HS, 11 — резерв. Бит `H` (`Head of Reclamation List Flag`) — флаг, которым драйвер помечает один из заголовков очередей из асинхронного плана для определения опустошения всех очередей этого плана (этот флаг вызывает обнуление бита `Reclamation` в регистре состояния контроллера). Бит `DTC` управляет переключателем `Toggle Bit`: 0 — использовать бит `DT` из данного заголовка очереди `QH`, 1 — из `qTD`. Флаг `C` — признак управляющей точки HS-устройства. Флаг `I` (Inactivate on Next Transaction) — программный запрос контроллеру обнулить бит активности при следующей транзакции. Используется только в заголовке очереди периодических транзакций FS/LS-устройств для обеспечения возможности программной коррекции значений полей `S-mask` и `C-mask` в данном заголовке. Поле `RL` (Nak Count Reload) задает значение счетчика ответов *NAK*, загружаемое в поле `Nak Cnt`. Поле `Mult` задает число транзакций в микрокадре для широкополосных точек (0 — резерв). Поля `Port Number`, `Hub Addr`, `μFrame S-mask` и `μFrame C-mask` требуются для точек FS/LS-устройств, по назначению они совпадает с одноименными полями `siTD`.

Поле `Current qTD Pointer` содержит адрес текущего обрабатываемого `qTD`, последующие 8 двойных слов являются *оверлейной областью передачи* (Transfer Overlay), в которую контроллер загружает требуемые параметры для обрабатываемого элемента. Большинство полей по назначению (и положению) совпадает с одноименными полями `qTD`, здесь перечислим только особые. Поле `NakCnt` (Nak Counter) — счетчик ответов *NAK* или *NYET*. В поле `C-prog-mask` контроллер отмечает микрокадры, в которых происходили транзакции завершения расщепленных транзакций. В поле `FrameTag` контроллер записывает тег кадра, в котором производится

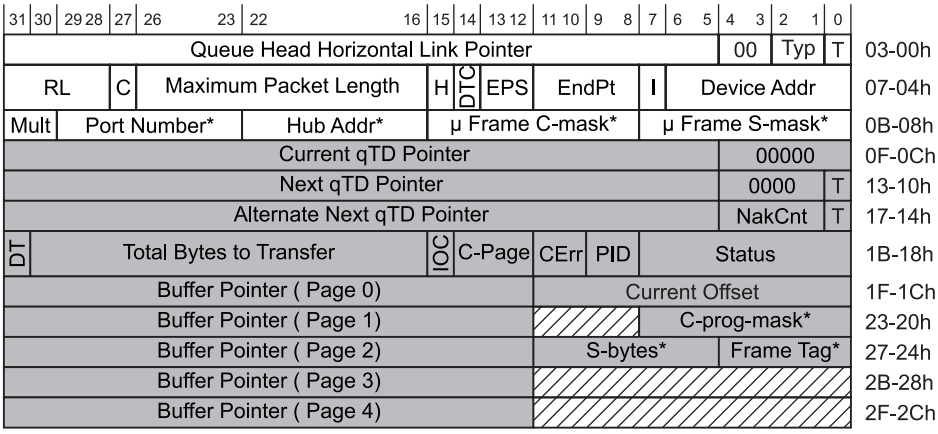


Рис. 15.15. Формат заголовка очереди (QH) для EHC

расщепленная транзакция прерывания. В поле `S-bytes` контроллер указывает число байтов, переданных в расщепленных транзакциях *IN* и *OUT*.

Формат *узла* `FSTN` (Periodic Frame Span Traversal Node) приведен на рис. 15.16. Здесь `Normal Path Link Pointer` может указывать на любую структуру данных, а обратный указатель `Back Path Link Pointer` может указывать только на заголовок очереди.

31 — 5	4 3	2 1	0	
Normal Path Link Pointer	00	Typ	T	03-00h
Back Path Link Pointer	00	01	T	07-04h

Рис. 15.16. Формат узла FSTN для EHC

USB без ПК — расширение On-The-Go

Протокол шины USB ориентирован на сугубо подчиненные отношения: всеми транзакциями со всеми подключенными устройствами управляет хост — как правило, это компьютер (ПК) с контроллером USB. Никакого равноправия в отношениях на шине USB быть не может, однако в ряде случаев хотелось бы обойтись и без компьютера. Так, например, напрашивается непосредственное соединение цифровой фотокамеры и фотопринтера, обеспечивающее печать снимков без участия ПК. Практически все периферийные устройства USB имеют встроенные микроконтроллеры, и функциональные возможности этих микроконтроллеров неуклонно растут. Периферийное устройство, имеющее хотя бы простейшие средства диалога с пользователем (дисплей, отображающий пару строк текста, и несколько кнопок управления), вполне может взять на себя управляющие функции в плане организации транзакций USB. Функции такого мини-хоста можно упростить, если ориентироваться на двухточечное соединение пары устройств без промежуточных хабов. В этом случае мини-хосту остается лишь идентифицировать одно подключенное устройство, и, если ему известно, как это устройство можно использовать, сконфигурировать его. Задача планирования транзакций лишь с одним устройством гораздо проще общей задачи «большого» хоста и хост-контроллера. Именно на создание таких упрощенных связей пары устройств нацелено расширение *OTG* (On-The-Go).

Документ *On-The-Go Supplement to USB 2.0 Specification* (версия 1.0 вышла в июне 2003 года) определяет дополнения к USB 2.0, необходимые для организации упрощенных соединений пары устройств. Большая часть документа посвящена описанию разъемов, и терминология OTG тоже привязана к типам разъемов (собственно, пользователь видит разъемы на устройствах и просто пытается соединить их доступными кабелями). В OTG принято следующее деление устройств:

- *устройство-A* (*A-Device*) — устройство, в гнездо которого вставлена вилка типа *A* (или *mini-A*). Это устройство подает питание (Vbus) 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 (в терминах USB);
- *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 (SRP) и согласования роли хоста (HNP).

Двухролевое устройство может поддерживать и хабы (это усложняет его задачи); однако стандартные хабы USB не позволяют работать протоколам SRP и HNP.

В основной спецификации USB фигурируют три типа разъемов (гнезд и вилок): стандартные 4-контактные *A* и *B* (см. рис. 9.2 на стр. 206), а также 5-контактный *mini-B*. Здесь допустимы кабели с вилкой *A* на одном конце и вилкой (*mini*)*B* на другом, а также неотсоединяемые от устройства кабели с вилкой A. В OTG введены 5-контактные *вилки mini-A* и универсальное 5-контактное *гнездо mini-AB* (рис. 15.17). 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.

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 <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 >10 Ом), то вставлена вилка *mini-B* — значит, подключено *устройство-A*; следовательно, двухролевое устройство должно стать периферийным.

Протокол запроса сеанса, *SRP* (Session Request Protocol), предназначен для дополнительного энергосбережения: когда *устройство-A* не нуждается в обмене по шине, оно может снять питание Vbus. При этом *устройство-B* все-таки может «попро-

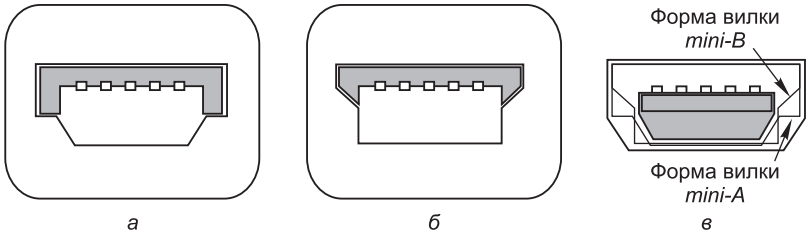


Рис. 15.17. Новые разъемы OTG: *а* — вилка *mini-A*; *б* — вилка *mini-B*; *в* — розетка *mini-AB*

сить внимания» — запросить сеанс связи. Здесь *сеансом* называется интервал времени, в течение которого двухролевое устройство подает достаточное (для работы) напряжение питания. Запрос может выполняться подачей положительных импульсов либо по линии Vbus, либо по сигнальным линиям (D+ или D-). *Устройство-B* должно использовать оба метода подачи запроса, *устройство-A* может распознавать любой из них (как удобнее его разработчику).

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

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

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-A*, еще до выбора конфигурации *устройства-B*, должно сообщить ему свое отношение к HNP. Для этого служат запросы к устройству *Set_Feature* (bmRequestType = 00000000b, bRequest = = 3):

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

ГЛАВА 16

Применение шины USB

Благодаря универсальности и способности эффективно передавать разнородный трафик шина USB применяется для подключения к PC самых разнообразных устройств. Она призвана заменить традиционные порты PC — COM и LPT, а также порты игрового адаптера и интерфейса MIDI. Спецификация USB 2.0 позволяет говорить и о подключении традиционных «клиентов» шин ATA и SCSI, а также захвате части ниши применения шины FireWire. Привлекательность USB придает возможность подключения/отключения устройств на ходу и возможность их использования практически сразу, без перезагрузки ОС. Удобна и возможность подключения большого количества (до 127) устройств к одной шине, правда, при наличии хабов. Хост-контроллер интегрирован во все современные системные платы. Выпускаются и карты расширения с контроллерами USB (обычно для шины PCI). Некоторое время повсеместное применение USB сдерживалось недостаточной активностью разработчиков ПО (производителей оборудования): просматривая перечни устройств, можно было видеть, что для всех указывается поддержка в Windows 98/SE/ME, а вот в графах Linux, MacOS, Unix и даже Windows 2000/NT часто стоят неприятные пометки N/A (Not Allowed — «не дозволено»). В настоящее время ситуация меняется (теперь возникают проблемы поиска драйверов для Windows 9x), но вопросы совместимости ПО иногда доставляют дополнительные хлопоты.

Для того чтобы система USB заработала, необходимо, чтобы были загружены драйверы хост-контроллера (или контроллеров, если их несколько). При подключении устройства к шине USB ОС Windows выдает сообщение «*Обнаружено новое устройство*» и, если устройство подключается впервые, предлагает загрузить для него драйверы. Многие модели устройств уже известны системе, и драйверы входят в дистрибутив ОС. Однако может потребоваться и драйвер изготовителя устройства, который должен входить в комплект поставки устройства, или его придется искать в Сети. К сожалению, не все драйверы работают корректно — «сырой» драйвер начальной версии, возможно, потребуется заменить более «правильным», чтобы устройство нормально опознавалось и хорошо работало. Но это общее горе пользователей любых устройств, а не только устройств для шины USB.

Перечислим основные области применения USB:

- *устройства ввода* — клавиатуры, мыши, трекболы, планшетные указатели и т. п. Здесь USB предоставляет единый интерфейс для различных устройств. Целе-

сообразность использования USB для клавиатуры в настольных ПК пока не очевидна (порты клавиатуры и мыши PS/2 есть на всех системных платах), хотя в паре с мышью USB (подключаемой к порту хаба, встроенного в клавиатуру) сокращается количество кабелей, тянущихся от системного блока к столу пользователя;

- *принтеры*. USB 1.1 обеспечивает почти ту же скорость, что и LPT-порт в режиме ECP, но при использовании USB не возникает проблем с длиной кабеля и подключением нескольких принтеров к одному компьютеру (правда, требуются хабы). USB 2.0 позволит ускорить печать в режиме высокого разрешения за счет сокращения времени на передачу больших массивов данных. Однако существует проблема со старым ПО, которое непосредственно работает с LPT-портом на уровне регистров, — на принтер USB оно печатать не сможет;
- *сканеры*. Применение USB позволяет отказаться от контроллеров SCSI или от занятия LPT-порта. USB 2.0 при этом позволяет еще и повысить скорость передачи данных;
- *аудиоустройства* — колонки, микрофоны, головные телефоны (наушники). USB позволяет передавать потоки аудиоданных, достаточные для обеспечения самого высокого качества. Передача в цифровом виде от самого источника сигнала (микрофона со встроенным преобразователем и адаптером) до приемника и цифровая обработка в хост-компьютере позволяют избавиться от наводок, свойственных аналоговой передаче аудиосигналов. Использование USB-аудиокомпонентов позволяет в ряде случаев избавиться и от звуковой карты компьютера — аудиокодек (АЦП и ЦАП) выводится за пределы компьютера, а все функции обработки сигналов (микшер, эквалайзер и т. п.) реализуются центральным процессором чисто программно. Аудиоустройства могут и не иметь собственных колонок и микрофона, а ограничиться преобразователями и стандартными гнездами («джеками») для подключения обычных аналоговых устройств;
- *музыкальные синтезаторы и MIDI-контроллеры* с интерфейсом USB. Шина USB позволяет компьютеру обрабатывать потоки множества каналов MIDI (пропускная способность традиционного интерфейса MIDI уже гораздо ниже возможностей компьютера);
- *фото- и видеокамеры*. USB 1.1 позволяет передавать статические изображения любого разрешения за приемлемое время, а также передавать поток видеоданных (живое видео) с достаточной (25–30 кадров/с) частотой кадров только с невысоким разрешением или сильным сжатием данных, от которого, естественно, страдает качество изображения. USB 2.0 теоретически позволяет передавать поток видеоданных высокого разрешения без сжатия (и потери качества). С интерфейсом USB выпускают как камеры, так и устройства захвата видеосигнала и TV-тюнеры;
- *коммуникации*. С интерфейсом USB выпускают разнообразные модемы, включая кабельные и xDSL, адаптеры высокоскоростной инфракрасной связи (IrDA FIR) — шина позволяет преодолеть предел скорости COM-порта (115,2 Кбит/с), не увеличивая загрузку центрального процессора. Выпускаются и сетевые адаптеры Ethernet, подключаемые к компьютеру по USB. Непосредственно (без

дополнительных устройств) портами USB соединить между собой даже два компьютера нельзя — на одной шине может присутствовать лишь один хост-контроллер (см. ранее). Специальное устройство для связи пары компьютеров выглядит как «таблетка», врезанная в кабель USB с двумя вилками типа «А» на концах. Объединяющее устройство (USB Link) может располагаться и на системной плате компьютера, с выходом на внешний разъем, — в этом случае компьютеры соединяются просто кабелем, подключаемым к рядовому порту USB одного компьютера и порту USB Link другого. Коммуникационное ПО USB Link позволяет даже строить сеть на основе цепочки соединений по USB. Для соединения нескольких компьютеров в локальную сеть выпускаются специальные устройства, выполняющие коммутацию пакетов между компьютерами. Объединение более двух компьютеров осложняется и топологическими ограничениями USB: длина одного сегмента кабеля не должна превышать 5 м, а использовать хабы для увеличения дальности неэффективно (каждый хаб дает всего 5 м дополнительного расстояния);

- *преобразователи интерфейсов* позволяют через порт USB, имеющийся теперь практически на всех компьютерах, подключать устройства с самыми разнообразными интерфейсами: Centronics и IEEE 1284 (LPT-порты), RS-232C (эмуляция UART 16550A — основы COM-портов) и другие последовательные интерфейсы (RS-422, RS-485, V.35...), эмуляторы портов клавиатуры и даже игрового порта, переходники на шину ATA, ISA, PC Card и любые другие, для которых достаточно производительности. Здесь USB становится палочкой-выручалочкой, когда встает проблема подключения к отсутствующему LPT или COM-порту, например, в блокнотном ПК, да и в других ситуациях. При этом ПО преобразователя может обеспечить эмуляцию классического варианта «железа» стандартных портов IBM PC, но только под управлением ОС защищенного режима. Приложение MS-DOS может обращаться к устройствам по адресам ввода-вывода, памяти, прерываниями, каналами DMA, но только из сеанса MS-DOS, открытого в ОС с поддержкой USB (чаще это Windows). При загрузке «голой» MS-DOS «палочка-выручалочка» не работает. Преобразователи интерфейсов позволяют продлить жизнь устройствам с традиционными интерфейсами, изживаемыми из PC спецификациями PC'99 и PC'2001. Скорость передачи данных через конвертер USB — LPT может оказаться даже выше, чем у реального LPT-порта, работающего в режиме SPP. Однако если подключаемое устройство требует интенсивного *диалогового обмена* (чередований коротких операций ввода и вывода) с ПО, подключение через USB будет работать медленно;
- *устройства хранения* — жесткие диски, устройства чтения и записи CD и DVD, стримеры — при использовании USB 1.1 получают скорость передачи, соизмеримую со скоростью их подключения к LPT, но более удобный интерфейс (как аппаратный, так и программный). При переходе на USB 2.0 скорость передачи данных становится соизмеримой с ATA и SCSI, а ограничений по количеству устройств достичь трудно. Особенно интересно использование USB для электронных устройств энергонезависимого хранения (на флэш-памяти) — такой

накопитель может быть весьма компактным (размером с брелок для ключей) и емким (от десятка мегабайт до гигабайта и более). Логически эти накопители могут представляться различным образом: просто как устройства хранения (USB Mass Storage, рассмотренные далее) или же как устройства считывания флэш-карт (например, Smart Media Card Reader, хотя у них и нет отделяемой SMC-карты). Выпускаются устройства для мобильного подключения накопителей с интерфейсом ATA-ATAPI — по сути, это лишь преобразователи интерфейсов, помещенные в коробку-отсек формата 5" или 3,5", а иногда выполненные прямо в корпусе разъема ATA. Имеются и устройства чтения-записи карт SmartMedia Card, CompactFlash Card и других типов флэш-карт, в которые можно вставлять соответствующие носители;

- *игровые устройства* — джойстики всех видов (от «палочек» до автомобильных рулей), пульты с разнообразными датчиками (непрерывными и дискретными) и исполнительными механизмами (почему бы не сделать кресло автогонщика с вибраторами и качалками?) — подключаются унифицированным способом. При этом исключается ресурсопожирающий интерфейс старого игрового адаптера (упраздненного уже в спецификации PC'99);
- *телефоны* — аналоговые и цифровые (ISDN). Подключение телефонного аппарата позволяет превратить компьютер в секретаря с функциями автодозвона, автоответчика, охраны и т. п.;
- *мониторы* — здесь шина USB используется для управления параметрами монитора. Монитор сообщает системе свой тип и возможности (параметры синхронизации) — это делалось и без USB по шине DDC. Однако USB-мониторы позволяют системе устанавливать параметры монитора. Регулировки яркости, контраста, цветовой температуры и т. п. могут теперь выполняться программно, а не только кнопками на лицевой панели монитора. В мониторы, как правило, встраивают хабы. Это весьма полезно, поскольку настольную периферию не всегда удобно включать в «подстольный» системный блок;
- *электронные ключи* — устройства с любым уровнем интеллектуальности защиты — могут быть выполнены в корпусе вилок USB. Они гораздо компактнее аналогичных устройств для COM- и LPT-портов.

Конечно же, перечисленными классами устройств сфера применения шины USB не ограничивается.

На современных системных платах имеется встроенный хост-контроллер USB; для плат, выпускаемых с 2002–2003 годов, характерно наличие контроллера USB 2.0 (EHC) с несколькими контроллерами-компаньонами (чаще UHC). Типовое число портов — четыре и более (раньше минимумом было два). В случае поддержки USB 2.0 эти порты могут быть как равноправными (поддерживать подключение на любой скорости), так и выделенными (часть — только HS, часть — FS/LS). Часть портов выводится на внешние разъемы системного блока непосредственно, остальные выводятся на штырьковые разъемы, расположенные на системной плате. К этим промежуточным разъемам внешние разъемы подключаются через кабели-«выкидыши». К сожалению, на многих корпусах системного блока отсутствует штатное место для разъемов USB с лицевой стороны, что делает подключение уст-

ройств неудобным. Эта проблема решается установкой разъемов на заглушки для 5” или 3,5”отсеков.

ВНИМАНИЕ

Из-за ошибок при подключении кабеля-«выкидыша» на дополнительных внешних разъемах могут оказаться перепутанными цепи `+5V` и `GND`. Из-за такой ошибки подключаемое устройство, питающееся от шины (например, флэш-память), может выйти из строя (сгореть).

Хабы USB выпускаются как в виде отдельных устройств, так и встраиваются в периферийные устройства (клавиатуры, мониторы). Как правило, хабы питаются от сети переменного тока (они должны питать подключаемые устройства). Выпускают и хабы, устанавливаемые внутрь системного блока компьютера и питающиеся от его блока питания. Такие хабы дешевле внешних и не требуют дополнительной питающей розетки. Один из вариантов исполнения — установка хаба на скобку, монтируемую в окно для дополнительных разъемов. Доступ к их разъемам со «спины» системного блока не очень удобен для пользователей. Другой вариант — хаб, устанавливаемый в 3-дюймовый отсек. Его разъемы легко доступны, индикаторы состояния портов хорошо видны, но не всегда удобны кабели, выходящие с передней панели системного блока. С другой стороны, для подключения электронных ключей (если их приходится часто менять) или миниатюрных накопителей этот вариант — самый удобный.

Выпускаются вспомогательные устройства USB, увеличивающие дальность связи (distance extender). Простейший вариант — это обычный 5-метровый кабель USB, на одном конце которого обычная вилка «A», а на другом — миниатюрный однопортовый хаб с гнездом «B». При необходимости между устройством USB и корневым хабом можно использовать цепочку из 5 таких удлинителей, что дает дальность 25 м плюс длина кабеля устройства (до 5 м). Другой вариант — пара устройств, соединяемых между собой обычным кабелем «витая пара» категории 5 (или даже оптоволокном), включаемая между периферийным устройством и корневым хабом. Дальнее (от хоста) устройство этого удлинителя может быть и хабом на несколько портов. К сожалению, увеличение дистанции упирается в ограничения на время задержки сигнала, свойственные протоколу шины USB, и максимально достижимое расстояние составляет 200 футов (около 60 м). Уже с удлинителем на 50 м дополнительные хабы недопустимы — задержка сигнала в кабеле «съедает» лимиты, отпущенные хабам в спецификации USB. Но даже и эта длина позволяет расширить сферу применения USB, например, для удаленного видеонаблюдения. Утверждения изготовителей о достижимой дальности 100 м (а на оптике и до 500 м!) вызывают некоторое недоверие, поскольку ограничения дальности в спецификации USB жестко ограничено установками тайм-аутов ожидания ответа, а быстрее света сигналы в кабелях разогнать не удается.

Ниже рассматриваются подробности реализации взаимодействия по шине USB с устройствами некоторых классов. Эти примеры позволяют осмыслить прикладные возможности USB, их можно использовать как отправные точки для разработки собственных устройств, в том числе и совершенно оригинальных по функциональному назначению. Последний раздел главы посвящен разрешению проблем,

возникающих при подключении устройств, в том числе и устройств собственной разработки.

Принтеры USB

Класс и протоколы принтеров с интерфейсом USB определены документом «Universal Serial Bus Device Class Definition for Printing Devices», первая версия которого появилась в 1997 году, версия 1.1 — в 2000 г. Протоколы определены с учетом обеспечения легкого перехода от традиционных интерфейсов принтера: последовательного (RS-232, RS-422) и параллельного (однонаправленного Centronics или двунаправленного IEEE 1284). Подключение принтера USB эмулирует его подключение к LPT-порту в режиме SPP или двунаправленном (IEEE 1284). Классовое определение USB принтера не затрагивает данные, передаваемые принтеру: это может быть любой язык описания страниц (PDL — Page Descriptor Language). Принтеры могут работать только на скоростях FS или HS (на низкой скорости в USB нет передач массивов).

Принтер имеет все стандартные дескрипторы устройства USB, специфических классовых дескрипторов нет. В дескрипторе устройства принтер сообщает нулевые коды класса, подкласса и протокола. Принтер имеет по крайней мере одну конфигурацию; в конфигурации имеется один интерфейс. На уровне интерфейса для принтеров определен *класс 07 подкласс 01*. Код *протокола* определяется выбранной альтернатиной установки интерфейса:

- *однонаправленный* (*Unidirectional*) интерфейс: данные на принтер передаются через конечную точку типа *Bulk-OUT*; состояние от принтера в формате, принятом для параллельного порта (3 значимых бита регистра состояния LPT-порта) передается по классово-специфическому запросу *Get_Port_Status* через *EP0*. Этому варианту интерфейса соответствует код протокола 01;
- *двунаправленный* (*Bi-Directional*) интерфейс: данные на принтер передаются через конечную точку типа *Bulk-OUT*; состояние от принтера передается через конечную точку типа *Bulk-IN*. Здесь также доступно получение состояния (3 бита) по запросу *Get_Port_Status* через *EP0*. Этому варианту интерфейса соответствует код протокола 02;
- *двунаправленный интерфейс с доставкой данных по логическим каналам* в соответствии с IEEE 1284.4 (IEEE 1284.4 compatible Bi-directional Interface), введенный в версии 1.1. Модель обмена та же, что и у предыдущего. Этому варианту интерфейса соответствует код протокола 03.

В принтере используется (кроме нулевой) только одна конечная точка для вывода данных (*Bulk-OUT*), для двунаправленного интерфейса — еще и *Bulk-IN* для получения данных обратного канала (состояния).

Принтер поддерживает стандартные запросы к устройствам USB, кроме установки метки времени (у принтера нет изохронных точек). Кроме того, он должен поддерживать классовые запросы (табл. 16.1).

По запросу *Get_Device_Id* принтер возвращает строку данных (Capabilities String), описывающих его в формате и синтаксисе, определенном в IEEE-1284. В запросе

в поле `wValue` указывается номер интересующей конфигурации, старший байт *wIndex* задает номер интерфейса (0), младший — номер альтернативной установки. Возвращаемая строка начинается с 2-байтного поля длины всей строки (старший байт — первый), за которой идет собственно идентификатор.

Таблица 16.1. Классовые запросы к принтерам

Запрос	bmRequestType	bRequest
Get_Device_Id	10100001b	0
Get_Port_Status	10100001b	1
Soft_Reset	00100011b	2

По *запросу Get_Port_Status* принтер возвращает байт состояния, аналогичный байту состояния LPT-порта. Здесь значимы только 3 бита (остальные — нули):

- бит 3 (Not Error), признак ошибки: 1 — нет ошибки, 0 — есть;
- бит 4 (Select), признак выбранности принтера: 1 — выбран (доступен), 0 — нет;
- бит 5 (Paper Empty), признак конца бумаги: 1 — нет бумаги, 0 — есть.

По *запросу Soft_Reset* принтер очищает свой буфер данных, переходит в исходное состояние, переводит в исходное состояние и разблокирует (если они были остановлены) конечные точки. На его состояние в плане интерфейса USB (адресован, сконфигурирован) этот сброс не влияет.

Для работы с USB-принтером следует считать дескриптор устройства и дескрипторы всех возможных конфигураций, выбрать конфигурацию и требуемую альтернативную установку интерфейса. При работе с двунаправленным интерфейсом некоторое неудобство вызывает тип конечной точки обратного канала (*Bulk*): запрос данных о состоянии принтера, который делает драйвер, будет «висеть» до срабатывания тайм-аута драйвера, если принтеру «нечего сказать». Формальное определение результатов запроса *Get_Port_Status* (по крайней мере, согласно спецификации версии 1.1) не позволяет судить о наличии данных в обратном канале. Однако, вспоминая работу LPT-порта, которую эмулирует подключение по USB, можно предположить, что признаком наличия данных обратного канала будет бит 3 (Not Error). В LPT-порту этот бит представляет текущее состояние сигнала Error#, который во всех режимах IEEE 1284 (исключая режим совместимости с SPP) используется для сигнализации наличия данных обратного канала.

Устройства хранения данных

Задача USB для устройств хранения сводится к передаче устройствам команд, определяющих выполняемую операцию, получению от устройства информации о завершении исполнения команды и, наконец, транспортировке хранимых данных. Спецификация USB для устройств хранения (Mass Storage) определяет несколько подклассов и протоколов. *Подкласс* определяет содержимое командного блока, *протокол* — способ транспортировки команд, состояния и данных; подклассы и протоколы независимы (любой формат блока можно доставлять любым транс-

портом). Специальных классовых дескрипторов у устройств хранения нет, но есть два классовых запроса (табл. 16.2).

Таблица 16.2. Классовые запросы к устройствам хранения

Запрос	bmRequestType	bRequest	Применимость для протоколов
Get_Max_Lun	10100001b	FEh	50h
Bulk-Only_Mass_Storage_Reset	00100001b	FFh	50h
ADSC	00100001b	00	00, 01

Протокол Bulk-Only-транспорт (код 50h) применяется в устройствах хранения со скоростями FS и HS, он рекомендован для всех новых разработок. Этот протокол обеспечивает взаимную синхронизацию устройства и хоста, используя никак не синхронизируемые (системой USB) потоки независимых каналов *Bulk-IN* и *Bulk-OUT* через пару соответствующих точек. Кроме того, используются два классовых запроса для определения числа доступных логических устройств и сброса интерфейса.

По *запросу Get_Max_LUN* устройство возвращает байт с максимальным возможным номером логического устройства (`LUN`, нумерация с нуля). В запросе в поле `wIndex` указывается номер интерфейса, `wLength` = 1.

По *запросу Bulk-Only_Mass_Storage_Reset* выполняется сброс интерфейса, указанного в поле `wIndex`: точки *Bulk-IN* и *Bulk-OUT* разблокируются, переключатель (`Toggle Bit`) устанавливается в положение *DATA0*, интерфейс переводится в состояние ожидания команды, все предыдущие запросы сбрасываются.

Блоки команд и состояний распознаются в последовательности пакетов по фиксированной длине пакета (они всегда ложатся точно в один пакет), сигнатурам и соответствию содержимого полей соглашениям (проверяются и нули в резервных полях). Командный блок длиной до 16 байтов позволяет транспортировать любые наборы команд, используемые в устройствах хранения с традиционными интерфейсами (ATA/ATAPI, SCSI и другие). Протокол передачи команд и данных работает следующим образом:

- хост посылает *командный блок* `CBW` (Command Block Wrapper) «в обертке» — пакет фиксированной длины (31 байт), включающий:
  - 4-байтную сигнатуру (`dCBWSignature` = 43425355h);
  - 4-байтный тег (`dCBWTag`), который служит для пометки и идентификации ответного блока состояния;
  - 4-байтное поле длины передаваемых данных (`dCBWDataTransferLength`);
  - байт флагов (`bmCBWFlags`), в котором используется лишь бит 7, указывающий направление передачи данных остальные биты — нули;
  - байт с 4-битным номером логического устройства, к которому обращается данный командный блок (`bCBWLUN` в битах [3:0], остальные биты — нули);
  - байт длины командного блока (`bCBWCBLength` в битах [4:0], допустимо 1–16, остальные биты — нули);

- собственно командный блок (CBWCB) длиной от 1 до 16 байт и заполнитель, доводящий длину этого поля до 16 байт;

- устройство подтверждает успех приема ACK'ом, анализирует пакет и, если «обертка» корректна и командный блок действителен, выполняет прием или передачу заказанного блока данных. Обмен данными инициирует хост в соответствии с посланной командой (данные могут и не предполагаться, тогда указывается их нулевая длина);
- на каждый командный блок (после успешного исполнения или отвергая его) устройство отвечает *блоком состояния* CSW в аналогичной «обертке» — 13-байтным пакетом, содержащим:
  - 4-байтную сигнатуру (dCSWSignature = 53425355h);
  - 4-байтный тег (dCSWTag), привязывающий этот ответ к конкретному командному блоку;
  - 4-байтное поле (dCSWDataResidue), в котором указывается разность между заказанным (и переданным) количеством данных и количеством, реально отработанным устройством;
  - байт состояния (bCSWStatus) выполнения команды: 00 — успешно, 01 — отказ (failed), 02 — фазовая ошибка (нарушение последовательности команд и данных).

Если устройство получает недопустимый командный пакет, оно его отвергает соответствующим CSW (с байтом состояния 01). На каждый выпущенный командный пакет хост должен получить ответ — состояние с тем же тегом (теги назначает хост, устройство ими только метит ответ). Фазовая ошибка (нарушение этой последовательности) отрабатывается с помощью классового запроса «сброс интерфейса», передаваемого через *EP0*, — интерфейс переходит в исходное состояние (готов принять командный блок). Этим же сбросом устраняется возможная блокировка конечных точек.

Протокол Control-Bulk-Interrupt (CBI, коды 00 и 01) предназначен только для FS-устройств, для новых разработок он не рекомендуется, на HS его применение не допускается. Для доставки команд служит классовый запрос *ADSC* (*Access_Device_Specific_Command*), передаваемый через точку *EP0* (Основной канал сообщений). Для доставки данных используются точки *Bulk-IN* и *Bulk-OUT*. Через эти же точки (включая и *EP0*) передается и информация о состоянии завершения команды; в протоколе с кодом 00 для передачи состояний выделяется дополнительная точка *Interrupt-IN* с длиной пакета 2 байта.

Командный блок передается в фазе данных транзакции управления, реализующей запрос *ADSC*. В запросе в поле wIndex указывается номер интерфейса, wLength — длина командного блока. Положительное подтверждение (*ACK*) на стадии состояния означает, что команда успешно принята (этот ответ может быть задержан на неопределенное время посылкой *NACK'ов*).

Состояние выполнения команды может передаваться несколькими способами:

- по прерыванию: когда у устройства есть что сообщит о состоянии выполнения команды, оно в транзакции с точкой *Interrupt-IN* возвращает два байта, трактовка которых зависит от подкласса устройства;

- по основному каналу сообщений — устройство в фазе завершения может ответить пакетом *STALL*, что будет означать отвергнутую команду. Уточнить состояние можно, послав соответствующую команду, предполагающую получение состояния в фазе данных через точку *Bulk-IN*;
- по каналу передачи данных: обнаружив ошибку в процессе выполнения команды, устройство на очередную транзакцию *Bulk-IN/OUT* ответит пакетом *STALL*. Уточнить состояние можно будет, послав следующий командный блок (предварительно разблокировав точки запросом *Clear_EP_Halt*).

Данные передаются пакетами через точки *Bulk-IN* и *Bulk-OUT*, укороченный пакет служит признаком конца блока данных. Хост и устройство отслеживают соответствие объема передаваемых данных запрошенному в команде. В случае обнаружения несоответствия устройство может сигнализировать об этом передачей состояния через точку *Interrupt-IN* или через ответ *STALL* в транзакциях передачи данных.

Сброс устройства можно выполнить, послав в запросе *ADSC* специальный командный блок с содержимым 1D, 04, FF, FF, FF... По этой команде устройство предпримет попытку безопасного прекращения текущей операции, очистит все буферы и очереди. После этого хост должен выполнить запросы *Clear_EP_Halt*, чтобы разблокировать точки и привести в исходное состояние переключатели `Toggle Bit`. Сброс через порт USB (по запросу к хабу) во время исполнения команды чреват потерей данных.

Прерывания (точка *Interrupt-IN*) используются только для протокола 00. На каждый посланный *ADSC* хост ждет прерывание; если устройство ответит на ADSC условием *STALL*, то для этой команды прерывание уже не ожидается. Если у устройства есть непереданный запрос прерывания, а хост посылает уже следующий *ADSC*, то прежний запрос прерывания устройство аннулирует.

Устройства человеко-машинного интерфейса (HID-устройства)

К *классу HID* (Human Interface Device) относятся устройства, обеспечивающие интерфейс между человеком и компьютером:

- клавиатура, мышь, шар, другие устройства-указатели, джойстик;
- органы управления на лицевой панели компьютера — кнопки, переключатели, регуляторы и т. п.;
- органы управления, присущие пультам дистанционного управления аудио- и видеотехникой, телефонам, различным игровым симуляторам (рули, педали, штурвалы и т. п.).

Для этих устройств характерен небольшой объем передаваемых данных, возникающих для компьютера спонтанно (асинхронно), и умеренные требования к задержке обслуживания. К данному классу относятся и иные устройства с похожим характером данных (считыватели штрихкода, термометры, вольтметры и т. п.). Подробно данный класс описан в документе Device Class Definition for Human

Interface Devices (HID), в 2001 году вышла его версия 1.11. Класс HID допускает работу устройств на любой скорости шины USB.

Деление на *подклассы* учитывает только необходимость поддержки данного устройства на этапе загрузки. Специальный *код протокола* выделяет только клавиатуру и мышь — стандартные устройства ввода.

С HID-устройствами хост обменивается сообщениями-*рапортами* (report), которые могут быть трех *типов* (`Report Type`):

- тип 1 — ввод (Input) от устройства;
- тип 2 — вывод (Output) в устройство;
- тип 3 — управление свойствами (Feature).

Если устройство способно обмениваться разнообразными (не по типу, а по назначению) рапортами, то каждый рапорт начинается с байта *идентификатора рапорта* (`Reprt ID`). Например, комбинация клавиатуры с указателем может давать как рапорты нажатий клавиш, так и рапорты устройства-указателя.

Интерфейс HID-устройства обеспечивает двунаправленный обмен рапортами между устройством и драйвером. В нем имеется:

- обязательный двунаправленный канал (через *EP0*) для вывода рапортов и ввода по опросу (полинг);
- обязательный однонаправленный канал асинхронного ввода рапортов по прерываниям (от устройства к его драйверу через точку *Interrupt-IN*);
- необязательный канал вывода по прерываниям; если у устройства имеется точка *Interrupt-OUT*, то выводные рапорты передаются по ней.

HID-устройства имеют специальный *классовый HID-дескриптор*, ссылающийся на дескриптор рапортов и набор физических дескрипторов (указываются тип и длина этих дескрипторов, что позволяет получить их по запросу *Get_Descriptor*).

Дескриптор рапорта представляет собой сложную структуру, описывающую передаваемые данные: назначение (ввод, вывод или управление свойствами), использование, диапазон допустимых значений, размер... Эта информация нужна драйверу HID-устройств, разбирающему (и собирающему) рапорты. Драйвер содержит модуль *Item Parser*, разбирающий, какому приложению следует передать тот или иной рапорт. Без должного дескриптора рапорта приложение рапорт ни принять, ни послать не сможет.

Набор физических дескрипторов описывает, какой частью тела[1] человек воздействует на тот или иной орган управления. Эти дескрипторы необязательны и сообщаются не многими устройствами; они вносят дополнительные сложности в описание, хотя позволяют приложениям точнее использовать те или иные органы.

HID-устройства поддерживают все стандартные запросы к устройствам и *специфические запросы*, приведенные в табл. 16.3.

---

[1] Кроме привычных клавиатур, мышей и джойстиков существуют и более сложные. Например, «кибер перчатка» имеет множество датчиков положения, связанных с разными частями кисти оператора. Физические дескрипторы и позволяют связывать передаваемые параметры с действиями оператора.

Таблица 16.3. Классовые запросы к HID-устройствам

Запрос	bmRequestType	bRequest
Get_Report	10100001	01
Get_Idle	10100001	02
Get_Protocol	10100001	03
Set_Report	00100001	09
Set_Idle	00100001	0A
Set_Protocol	00100001	0B

Запросы Get_Report и *Set_Report* служат для приема и передачи рапортов через *EP0*. Здесь в поле `wValue` старший байт задает тип рапорта, младший — его идентификатор. В поле `wIndex` задается номер интерфейса, поле `wLength` задает длину рапорта, который передается в фазе данных.

Запрос Set_Idle позволяет управлять подачей рапортов по каналу прерываний в случае отсутствия изменений состояния рапортуемых величин. Здесь в поле `wValue` старший байт задает *длительность молчания в покое* (idle duration), младший — идентификатор рапорта. В поле `wIndex` задается номер интерфейса, поле `wLength` = = 0 (фаза данных отсутствует). Если задана нулевая длительность, то устройство будет передавать рапорты только в случае изменений состояния (что требуется, например, для клавиатуры). Ненулевое значение трактуется как длительность интервала (в 4-миллисекундных единицах), в течение которого точка *Interrupt-IN* отвечает на опросы `NAK`'ами, если нет изменений состояния. Так, например, можно уменьшить поток «пустых» данных опроса устройств-указателей (мыши или джойстика), не теряя быстроты реакции на события (задаваемая длительность и `bInterval`, задающий частоту опроса точки *Interrupt-IN*, независимы).

Запрос Get_Idle позволяет считать текущее значение длительности для рапорта, идентификатор которого задан в младшем байте поля `wValue`. В поле `wIndex` задается номер интерфейса, поле `wLength` = 1 (принимается один байт данных).

Запрос Set_Protocol (только для устройств, участвующих в начальной загрузке) позволяет переключать протокол работы устройства. Тип протокола задается в поле `wValue`: 0 — протокол (упрощенный), используемый при загрузке (Boot Protocol); 1 — нормальный протокол рапортов (Report Protocol). В поле `wIndex` задается номер интерфейса, поле `wLength` = 0.

Запрос Get_Protocol позволяет определить текущий протокол. В поле `wIndex` задается номер интерфейса, поле `wLength`=1 (принимается один байт данных).

Аудиоустройства

Аудиоустройства с точки зрения USB представляют собой, как правило, композитные устройства, содержащие набор независимых интерфейсов. Путь любого аудиосигнала в аудиоустройстве начинается с *входного терминала* (Input Terminal) и заканчивается на *выходном терминале* (Output Terminal). Между терминалами

могут находиться различные *модули* (Unit), осуществляющие какие-то преобразования и соединения.

Терминалы могут быть различных типов:

- *USB-терминал* — подключение к шине USB, с помощью которого через конечные точки (как правило, изохронные) осуществляется доставка потоков аудиоданных от хоста к устройству (входной терминал) или от устройства к хосту (выходной терминал)[1]. Для интерфейсов USB-терминалов определен подкласс 02 — *AUDIOSTREAMING* со своими специфическими дескрипторами, описывающими способ доставки (и синхронизации) и формат потока;
- *входные (оконечные) терминалы*: устройства для записи звуков — встроенные в устройства или внешне подключаемые микрофоны различных типов, наборы микрофонов для пространственной записи (каждой разновидности назначен свой код типа);
- *выходные (оконечные) терминалы*: устройства для воспроизведения звуков — колонки разных типов, наушники;
- *двунаправленные (оконечные) терминалы*: телефонные трубки и головные телефонные гарнитуры, без эхоподавления и с ним;
- *телефонные терминалы*: устройства подключения к телефонной сети и мини-АТС;
- *внешние терминалы*: аналоговые и цифровые входы и выходы различных стандартов и форматов, включая интерфейсы S/PDIF и потоки аудиоданных, передаваемые по шине IEEE 1394;
- *встроенные терминалы*: проигрыватели грампластинок, CD, DVD, звуковое сопровождение TV и видеомагнитофонов, приемники (радио, телевизионные, спутниковые), устройства аналоговой и цифровой звукозаписи, радиопередатчики, синтезаторы, измерительные генераторы сигналов.

Всеми этими терминалами требуется управлять по шине USB; для них имеются дескрипторы, описывающие их управляемые свойства.

Модули (units), которые располагаются пути аудиосигнала, могут выполнять самые разнообразные функции: коммутация, регулировка уровня, микширование, панорамирование, фильтрация, реверберация и все возможные эффекты, которые стали легко реализуемыми с применением цифровой формы обработки сигналов. Для этих модулей введен подкласс 01 — *AUDIOCONTROL*, тоже со своими специфическими дескрипторами и запросами.

С аудиоустройствами тесно связаны и *MIDI-устройства*, являющиеся приемниками или источниками потоков MIDI-сообщений[2]. Для них введен подкласс 03 — *MIDISTREAMING* со своими специфическими дескрипторами и запросами. MIDI-устройство USB может выступать как в роли простого конвертора интерфейсов (выполнять доставку сообщений между хостом и MIDI-разъемами на устройстве), так и MIDI-синтезатора, преобразующего MIDI-сообщения в аудиопотоки. При этом обрабатываться могут сообщения как с внешних разъемов, так и с шины USB.

---

[1] Здесь вход и выход рассматриваются с точки зрения аудиоустройства, а не хоста.

[2] Это команды для синтезаторов, но не собственно аудиоданные.

Синтезатор MIDI с точки зрения аудиоустройства является встроенным входным терминалом. Он поставляет аудиопоток, дальнейший путь которого определяется аудиоустройством (через интерфейсы подклассов 01 и 02).

Разрешение проблем при подключении устройств

Все спецификации USB разрабатывались (и разрабатываются) в расчете на полную поддержку Plug and Play и «горячего» подключения-отключения устройств. При этом в идеальном варианте с пользователя снимаются все заботы по конфигурированию подключаемых устройств и установке его программного обеспечения. Однако на практике процесс подключения нового устройства происходит не всегда в таком «безоблачном» варианте. Проблемы, с которыми сталкиваются пользователи готовых устройств, как правило, сводятся к поиску подходящих драйверов и прикладного ПО. Конечно же, не надо забывать и о таких простых неполадках, как случайное отключение (опциями CMOS Setup) контроллера USB, находящегося на системной плате, а также ошибок подключения дополнительных разъемов USB.

В ОС Windows каждое подключенное устройство USB отображается в Диспетчере устройств (Device Manager). Для наглядности удобно в меню Вид выбрать опцию Устройства по подключению, тогда устройства будут отображаться в виде деревьев, «растущих» из шины PCI. Каждое дерево начинается со своего хост-контроллера, к которому подключен корневой концентратор (Root Hub). Заметим, что для каждого дерева (то есть для каждого хост-контроллера) адреса устройств USB назначаются независимо. К сожалению (но для удобства пользователя), диспетчер устройств Windows отображает только подключенные устройства и не отображает свободных портов хабов. Это усложняет понимание организации портов и контроллеров. Более информативна утилита USB View (от Microsoft), которая подробнее отображает дерево устройств (хост-контроллеры, корневые хабы, промежуточные хабы и конечные устройства). Для каждого элемента отображаются дескрипторы устройств, их конечных точек, а также состояние подключения:

- номер выбранной конфигурации (Current Config Value);
- скорость работы (Device Bus Speed);
- адрес устройства (Device Address);
- число открытых каналов (Open Pipes, не считая основного канала сообщений);
- состояние подключения (ConnectionStatus).

Для хабов в дереве отображаются ветки, соответствующие всем имеющимся его портам. Это позволяет определить внутреннюю структуру хост-части системы. Так, например, можно увидеть в компьютере с шестью портами USB и провозглашенной поддержкой USB 2.0 «Расширенный хост-контроллер USB 2.0» (EHC) с шестипортовым корневым хабом и три «Универсальных хост-контроллера USB», у каждого из которых имеется по двухпортовому корневому хабу. В этом случае HS-устройство можно подключать к любому порту и оно маршрутизирующей логикой

(см. главу 15) будет связано с контроллером EHC. Для устройств FS и LS каждая пара портов подключается к своему контроллеру-компаньону (UHC). В такой системе можно говорить о суммарной пропускной способности шины USB, равной 480 + 3×12 = 516 Мбит/с. Правда, из этого следует вычесть накладные расходы (см. главу 11) и не забывать о возможных ограничениях со стороны шины PCI, к которой подключены хост-контроллеры, и со стороны контроллера системной памяти, которой контроллеры пользуются очень активно. Если бы мы в шестипортовой системе увидели расширенный контроллер с четырехпортовым корневым хабом, это означало бы, что поддержка USB 2.0 имеется только на четырех портах из шести.

Еще более развернутую информацию об устройствах USB дают утилиты, поставляемые для разработчиков производителями микросхем USB. Примером может быть утилита USB Monitor от Cypress Semiconductors, позволяющая прочитать все дескрипторы, имеющиеся у устройства USB.

Для облегчения отладки собственных устройств полезно как-нибудь отображать информацию о текущем состоянии его интерфейсной части (блока SIE, см главу 13). Это отображение должно быть автономным (индикация на самом устройстве, а не средствами хост-компьютера). На начальном этапе отладки достаточно индицировать четыре состояния:

- «*Запитано*» (*Powered State*);
- «*Дежурное*» *состояние* (*Default State*);
- «*Адресовано*» (*Addressed State*);
- «*Сконфигурировано*» (*Configured State*).

Устройство, подключаемое к шине USB, должно последовательно пройти от состояния «*запитано*» до «*сконфигурировано*». По тому, в каком состоянии «застревает» подключаемое устройство, можно судить о событиях, происшедших в системе, и найти неполадки.

Если устройство при подключении «зависает» в состоянии «*запитано*», это может быть вызвано рядом причин:

- хост не получает сигнала о подключении устройства. Следует проверить наличие высокого логического уровня на линии D+ (для FS/HS-устройства) или D- (для LS-устройства), который должно обеспечить устройство. Этот сигнал может не дойти до хоста при неплотном соединении разъема USB (в нем сигнальные цепи замыкаются позже питающих);
- хост не реагирует на сигнал подключения. Нормально хост реагирует посылкой сигнала шинного сброса — занулением линии `D+` или `D-` на 10–20 мс, этот импульс легко можно увидеть с помощью осциллографа. «Заснувший» хост может и не реагировать на подключение (не посылать сигнал сброса). Доводилось наблюдать, как хост с ОС UNIX перестает реагировать на подключение LS-устройства (даже стандартной мыши) после многократных (с интервалом в несколько секунд) подключений/отключений его к одному и тому же порту корневого хаба. После этого подключение того же устройства к другому порту воспринимается нормально. Более того, последующее подключение этого устройства к порту, на котором реакция на подключение прекращалась, снова воспринимается нормально;

- устройство не реагирует на сигнал шинного сброса. Нормально устройство должно по сигналу шинного сброса перейти в «*дежурное*» состояние.

Если устройство «зависает» в «*дежурном*» состоянии, это означает его неспособность сообщить дескрипторы и исполнить запрос назначения уникального адреса. Здесь определить неисправность несколько сложнее, поскольку требуется просмотреть пакеты запросов, принимаемые и передаваемые устройством. Причины некорректного обмена пакетами могут быть как в электронике, так и в микропрограммном обеспечении (firmware). В электронной части, например, может быть неправильная частота генератора, синхронизирующего SIE контроллера устройства USB, или неисправности в сигнальных цепях. В микропрограммном обеспечении следует проверить дескрипторы, сообщаемые устройством.

Зависание в состоянии «*адресовано*» может означать, что подключенным устройством никто в системе не заинтересовался и не пытается сконфигурировать. Это происходит, например, когда ОС не может связать с обнаруженным устройством подходящий драйвер. Обнаруженное устройство идентифицируется кодами `VID` (код производителя) и `PID` (код продукта) в своих дескрипторах (см. главу 13). Также это может происходить в случае подключения HS-устройства, критичного к скорости передачи, к FS-порту.

В поддержке устройств USB, по крайней мере в ОС Windows, наблюдается странность (с точки зрения автора) в учете устройств. Каждое вновь подключенное устройство после установки его ПО поддержки оставляет в реестре Windows запись, в которой некоторый идентификатор устройства связывается с именами модулей ПО, его поддерживающих. Странность заключается в том, что вместе с такими бесспорными идентификаторами устройства, как `VID` и `PID`, в реестре фиксируется и номер порта. Если то же устройство переключить в другой порт, то ОС это воспримет как подключение нового(!) устройства и снова начнет установку ПО поддержки, создавая и новую запись в реестре. Отключение устройства не вызывает удаления записи из реестра, так что реестр будет «обрастать» лишними записями. Само по себе это не так уж важно (вопрос о компактности записей, похоже, давно уже решен в пользу лени), но ПО некоторых устройств отказывается загружаться (или работать), когда в реестре прописан его двойник с другим адресом. В этой ситуации приходится либо возвращать устройство в порт первоначального подключения (что не всегда удобно), либо вручную чистить реестр (что не к лицу рядовому пользователю). Вероятная причина этой странности — упрощение идентификации нескольких однотипных устройств, одновременно подключенных к компьютеру. Использование номера порта, который является сравнительно случайным (пользователь подключает устройство в первый попавшийся разъем), для идентификации устройства следовало бы заменить на уникальный идентификатор конкретного экземпляра устройства. Для этого спецификацией USB в дескрипторах устройства определены все необходимые элементы, включая строковый дескриптор с серийным номером. «Глубина залегания» этой странности — типовые драйверы устройств (подвластные их разработчикам) или системный драйвер USBD (за него отвечает разработчик ОС), на момент написания книги автором не выяснена.

ГЛАВА 17

Шина IEEE 1394 — FireWire

Высокопроизводительная последовательная шина (High Performance Serial Bus) IEEE 1394 — FireWire создавалась как более дешевая и удобная альтернатива параллельным шинам (SCSI) для соединения равноранговых устройств. Шина позволяет связать до 63 устройств без применения дополнительной аппаратуры (хабов). Устройства бытовой электроники — цифровые камкордеры (записывающие видеокамеры), камеры для видеоконференций, фотокамеры, приемники кабельного и спутникового телевидения, цифровые видеоплееры (CD и DVD), акустические системы, цифровые музыкальные инструменты, а также периферийные устройства компьютеров (принтеры, сканеры, устройства хранения данных) и сами компьютеры могут объединяться в единую сеть. Шина не требует управления со стороны компьютера. Шина поддерживает *динамическое реконфигурирование* — возможность «горячего» подключения и отключения устройств. События подключения/отключения вызывают сброс и реинициализацию: определение структуры шины (дерева), назначение физических адресов всем узлам и, если требуется, выборы мастера циклов, диспетчера изохронных ресурсов и контроллера шины. Через доли секунды после сброса все ресурсы становятся доступными для последующего использования, и каждое устройство имеет полное представление обо всех подключенных устройствах и их возможностях. Благодаря наличию линий питания, интерфейсная часть устройства может оставаться подключенной к шине даже при отключении питания функциональной части устройства.

По инициативе VESA шина позиционируется как основа «домашней сети», объединяющей всю бытовую и компьютерную технику в единый комплекс. Эта сеть является одноранговой (peer-to-peer), чем существенно отличается от USB.

Основные свойства шины FireWire перечислены далее.

- *Равноранговость*. Шина позволяет любым своим абонентам обмениваться данными друг с другом. Для организации обменов не требуется хост-компьютер (и его ресурсы — память и процессор), который мог бы стать «бутылочным горлом» при интенсивных обменах.
- *Универсальность*. Шина обеспечивает передачу как асинхронного, так и изохронного трафика. Это позволяет объединять в единую сеть компьютеры, их периферийные устройства (принтеры, сканеры, устройства хранения) и цифровую аудио-видеотехнику.

- *Надежность*. Шина обеспечивает контроль достоверности передачи, обработку и исправление ошибок.
- *Легкость установки и использования*. Для начала работы достаточно соединить устройства, соблюдая несложные топологические правила.
- *Большое число соединяемых устройств*. Одна шина может объединять до 63 устройств (узлов). Возможно объединение в единую сеть нескольких шин (формально — до 1024) с помощью мостов или коммутаторов.
- *Свободная топология*. Устройства могут иметь один или несколько портов. Посредством унифицированных кабелей устройства соединяются произвольным образом, исключая лишь петлевые соединения. Ограничение — между любой парой конечных узлов должно быть не более 16 промежуточных узлов.
- *Большая протяженность*. Длина одного кабельного сегмента, соединяющего пару устройств, может достигать 4,5 м. Ограничение — суммарная длина кабеля в одной шине не должна превышать 72 м.
- *Полная поддержка PnP с динамическим конфигурированием*:
  - *автоматическое конфигурирование*. В процессе инициализации устройства шины автоматически организуются в иерархическую структуру (дерево) и самоидентифицируются (определяют свои номера узлов и предоставляют информацию о себе);
  - *поддержка «горячего» подключения-отключения*. Любое событие подключения/отключения вызывает реинициализацию шины, после которой формируются новое дерево и новая нумерация узлов. Во время реинициализации «полезный» обмен данными прерывается, но время реконфигурирования невелико — менее 400 мс в старой шине и менее 200 мкс для 1394a.
- *Сосуществование на одной шине устройств с различными скоростями обмена*. Для шины определен ряд стандартных скоростей: S100, S200, S400; в IEEE 1394b (2002 год) определены новые скорости: S800, S1600 и S3200. При обменах выбирается скорость, доступная узлам, вовлеченным в передачу.
- *Высокая скорость обмена и изохронные передачи*. Даже на начальном уровне S100 (около 100 Мбит/с) по шине можно передавать одновременно два канала видео вещательного качества (30 кадров в секунду) и аудиосигнал с качеством CD (стерео).
- *Питание от шины*. Шина обеспечивает питание устройств постоянным током до 1,5 А с напряжением 8–40 В.
- *Малое число цепей*. В шине используются две экранированные витые пары для передачи сигналов и дополнительно пара проводов для питания от шины.
- *Возможность гальванической развязки*. Компоненты физического уровня, электрически связанные с коннекторами и кабелями, могут быть развязаны по постоянному току от остальных компонентов устройства. В новых вариантах среды передачи IEEE 1394b возможна полная гальваническая развязка узла от кабеля и применение оптоволоконной связи.
- *Низкая цена компонентов и кабеля* (по сравнению со SCSI).

Организация и топология шины

Стандарт IEEE 1394 описывает шину с последовательным интерфейсом, по которой информация передается *пакетами*. Источник пакетов должен получить право передачи пакета, используя механизм арбитража, в котором задействуются все устройства, подключенные к шине. *Арбитраж* предоставляет узлам право доступа в соответствии с запрошенным типом передачи. Для асинхронных транзакций арбитраж обеспечивает справедливое распределение полосы пропускания, для изохронных передач — гарантированную (предварительно согласованную) полосу пропускания для каждого канала. Коллизии (столкновения пакетов от нескольких устройств) в исправной шине отсутствуют.

Все устройства соединяются друг с другом кабелями на основе любой топологии (древовидной, цепочечной, звездообразной). Каждое устройство (узел сети) обычно имеет несколько равноправных соединительных разъемов, представляющих его порты. Некоторые устройства имеют только один разъем, что ограничивает возможные варианты их месторасположения. В современной редакции стандарт допускает до 16 портов (разъемов) на одном устройстве, чаще встречаются 1–4-портовые устройства. Многопортовые узлы позволяют соединять множество узлов IEEE 1394 без использования вспомогательного оборудования (хабов). Внутри многопортового узла имеется повторитель, транслирующий пакеты и управляющие сигналы между портами.

Устройства на шине могут передавать данные на разных скоростях. *Базовой скоростью*, поддерживаемой любым устройством, является S100. На этой общедоступной скорости передаются все служебные пакеты, в том числе и пакеты самоидентификации. Если устройство поддерживает высокую скорость (например, S400), то оно обязано поддерживать и все более низкие скорости вплоть до базовой. Пакеты транзакций могут передаваться на любой скорости, доступной узлам, связанным кабельным сегментом. При этом перед посылкой пакета на скорости, отличной от базовой (S100), передающий узел посылает сигнал выбранной скорости (см. главу 22). Пакет, приходящий на один порт устройства на высокой скорости, узел не будет транслировать на порт, для которого установлена более низкая скорость. Таким образом, скорость, на которой возможно прохождение пакета между произвольной парой узлов, зависит от скоростных возможностей этих узлов и промежуточных узлов, лежащих на пути между ними. Отсутствие ответа на пакет, посланный на высокой скорости, служит поводом для повторной попытки посылки на более низкой скорости. Заранее узнать доступную скорость можно по карте скоростей — двухмерной матрице, в которой для каждой пары узлов шины указывается максимальная возможная скорость передачи. Эта карта доступна только при наличии диспетчера шины (см. главу 21).

Кабельная шина допускает большую свободу выбора *топологии физических соединений* при соблюдении следующих ограничений:

- на шине может быть не более 63 узлов;
- между любой парой узлов может быть не более 16 кабельных сегментов (в 1394a допускается до 24 сегментов);

- длина сегмента стандартного кабеля не должна превышать 4,5 м;
- суммарная длина кабеля не должна превышать 72 м;
- топология не должна содержать петель. В первых редакциях стандарта за это отвечал только пользователь, в IEEE 1394b имеются средства автоматического исключения петель.

В IEEE 1394b введены новые варианты среды передачи, допускающие большую длину сегментов при соединении узлов друг с другом. Здесь используется иная сигнализация, не совместимая с традиционной сигнализацией IEEE 1394 и IEEE 1394a. Однако в 1394b есть и «двуязычные» узлы, способные частью своих портов работать со старыми узлами 1394/1394a, а другой — с узлами 1394b. С помощью таких узлов можно строить смешанные сети и преодолевать вышеуказанные ограничения по расстоянию.

При любой физической топологии *логическая топология для передачи данных* остается *шинной* — пакеты распространяются от источника ко всем узлам шины[1]. *логическая топология для арбитража — древовидная* иерархическая, «верховный арбитр» — корневой узел.

В IEEE 1394 кроме кабельной сети определены и спецификации использования последовательной шины в качестве кросс-шины (*Backplane Environment*, см. главу 22) для объединения узлов в пределах одного устройства. Здесь используется несколько иной физический уровень (PHY) — всегда однопортовый, с некоторыми отличиями в регистрах. Конфигурация шины при этом фиксирована, механизм автоконфигурирования упрощен, физические идентификаторы узлов назначаются программно (записью в регистр PHY). При идентификации скорости формально указывается S100, но реально это соответствует скоростям S50 или S25.

Архитектура сети

Архитектура IEEE 1394 позволяет организовывать сети, состоящие из одной или нескольких (до 1023) шин, причем не только последовательных. К шинам IEEE 1394 подключаются физические устройства, которые должны иметь по крайней мере один порт. Физическое устройство может иметь сложную внутреннюю структуру, что иллюстрирует рис. 17.1. К топологии сети относятся следующие понятия:

- *сеть* — совокупность узлов, подключенных к одной шине или нескольким шинам, соединенным мостами. Все узлы сети имеют возможность взаимодействия друг с другом;
- *шина* — совокупность узлов, связанных друг с другом кабельными сегментами. Все узлы шины используют ее общую (разделяемую) среду передачи, получая право на передачу путем арбитража. Подключение-отключение узлов вызывает реконфигурацию данной шины, в ходе которой узлы получают новые номера (физические адреса);

---

[1] Пакет, передаваемый на высокой скорости, не будет доставлен узлу, не поддерживающему эту скорость. Также доставке могут препятствовать низкоскоростные узлы, находящиеся по пути между источником и получателем пакета.

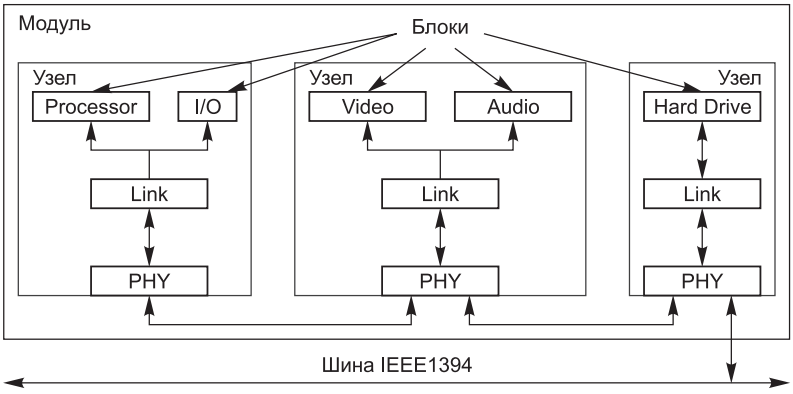


Рис. 17.1. Модуль, узлы и блоки

- *модуль* (module) — физическое устройство («коробка»), подключаемое к шине, и содержащее один или несколько узлов;
- *узел* (node) — логический объект, имеющий уникальный физический адрес на шине. Узел имеет собственное адресное пространство, в котором обязательно присутствуют стандартные регистры управления и состояния (CSR), а также постоянная память (ROM) со стандартным набором структур, описывающих узел;
- *порт* (port) — внешний физический интерфейс узла, обеспечивающий соединение узлов в шину. Узел может иметь несколько портов, что обеспечивает связь множества узлов без использования каких-либо дополнительных устройств-концентраторов;
- *блок* (unit) — часть узла, обеспечивающая его отдельную функциональность: память, ввод/вывод, обработку данных. Блоки имеют свои регистры и/или области памяти, отображенные на общее адресное пространство узла. Блоки функционируют относительно независимо и управляются собственными драйверами.

Если проводить параллели с USB, *блок* (IEEE 1394) можно соотнести с *интерфейсом* (USB), *узел 1394* — с *устройством USB*, а *модуль 1394* — с *композитным устройством USB*, содержащим несколько функций и хаб[1]. Понятие шины в USB и IEEE 1394 совпадает, хотя организация и возможности этих шин значительно различаются. Аналога сети в USB нет.

Шина IEEE 1394 позволяет любому узлу взаимодействовать с другими различными путями:

- выполнять асинхронные транзакции чтения и записи с регистрами и областями памяти другого (удаленного) узла. При этом сами операции в удаленном узле могут выполняться как чисто аппаратно (прямой доступ к памяти удаленного узла), так и программными средствами удаленного узла;
- передавать и принимать потоки данных по изохронным каналам (с гарантированной полосой пропускания, но без гарантий доставки);
- передавать и принимать асинхронные потоки данных (без гарантий);

[1] При этом хаб сам является дополнительным устройством USB.

- выполнять блокированные транзакции (чтение-модификация запись) со специальными регистрами удаленного узла. Эти транзакции служат для упорядочивания взаимодействия с какими-либо ресурсами, к которым может обращаться множество узлов шины;
- вызывать прерывания на удаленном узле (асинхронно транзакцией записи в специальный регистр узла).

Асинхронные транзакции могут быть как направленными (адресованными конкретному узлу), так и широковещательными. Для направленных асинхронных транзакций протокол шины обеспечивает надежную доставку. Потоковые передачи также могут быть широковещательными и направленными. Прикладные возможности использования всех этих базовых примитивов взаимодействия рассмотрены в главах 24–26.

Адресное пространство сети и узла

Каждому узлу выделяется адресное пространство размером 256 Терабайт, которое является частью адресного пространства одной шины. Шин в системе может быть множество; все связанные шины объединяются в общее адресное пространство размером 16 Экзабайт (2^{64} байт). Объединение шин осуществляется *мостами*; объединяться могут любые шины, отвечающие архитектуре CSR (как последовательные, так и параллельные). Формат адреса для IEEE 1394 приведен на рис. 17.2.

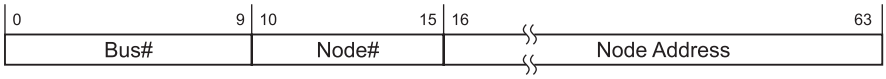


Рис. 17.2. Формат адреса IEEE 1394

ВНИМАНИЕ

в IEEE 1394 принято соглашение об адресации *Big Endian*: старший бит адреса имеет номер 0; адрес указывает на старший байт адресуемой структуры; последующие адреса относятся к байтам по убывающему старшинству. Типовой адресуемой единицей является *квадлет* (quadlet) — 4-байтное (32-битное) число.

На рисунках старший бит квадлета (бит 0) изображается слева, по шине он передается первым.

Распределение *адресного пространства узла* изображено на рис. 17.3.

Начальное пространство памяти (Initial Memory Space), занимающее большую часть пространства узла, используется для основного взаимодействия между устройствами, связанными шиной.

Приватное пространство (Private Space) размером 256 Мбайт используется для локальных нужд узла.

Пространство регистров (Register Space) имеет размер 256 Мбайт. Разрядность всех регистров — 32 бит. В дальнейшем описании в скобках указаны относительные адреса регистров; полный адрес (внутри узла) получается сложением относительного адреса и FFFF F000 0000h. Пространство регистров делится на две части:

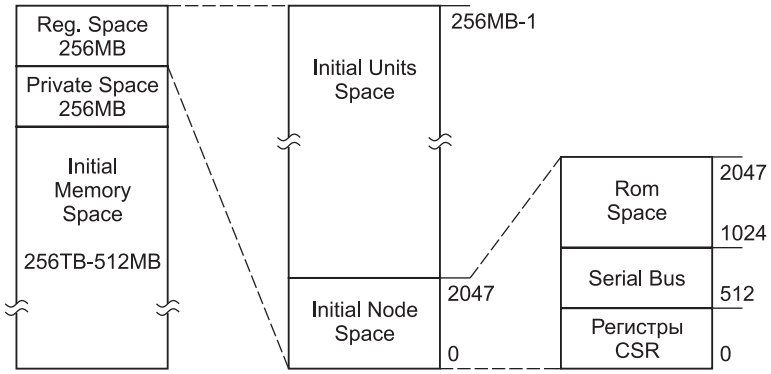


Рис. 17.3. Адресное пространство узла

- начальное пространство регистров узла (Initial Node Space) размером 2 Кбайт, используемое для общего конфигурирования и управления узлом. В него входят:
  - архитектурные регистры CSR (512 байт);
  - регистры последовательной шины (512 байт);
  - пространство памяти конфигурации (ROM Space), в котором отображается первый килобайт этой памяти. Если эта память имеет больший размер, то его продолжение «залезает» в начальное адресное пространство узла.
- Начальное пространство регистров блоков, входящих в данный узел (Initial Units Space), размером почти 256 Мбайт. Из этого пространства область 0800h–FFFCh в IEEE 1394 отводится под нужды последовательной шины. В частности, здесь располагаются:
  - у узла-диспетчера шины — *карта топологии* `Topology_Map` (1000–13FCh) и *карта скоростей* `Speed_Map` (2000–2FFCh), описанные в главе 21;
  - у узлов, причастных к изохронному обмену, — регистры управления «штекерами» изохронных передач PCR (0900–09FFh), описанные в главах 18 и 26;
  - Регистры `FCP command frame` (0B00–0CFC) и `FCP responce frame` (0D00–0EFC), определенные спецификацией *IEC 61883-1/FDIS*.

Архитектура узла

В плане описания работы шины наибольший интерес представляет узел. Узел имеет явно выраженную трехуровневую структуру средств FireWire, к которой обращаются драйверы прикладного и системного ПО (рис. 17.4):

- *физический уровень — PHY* (Physical Layer) выполняет основные функции, связанные с подключением узла к шине:
  - подключение узла к шине (механическое и электрическое);
  - автоматическое конфигурирование шины и узла при инициализации;
  - арбитраж при передаче данных;
  - кодирование и декодирование сигналов состояния шины и потоков данных;

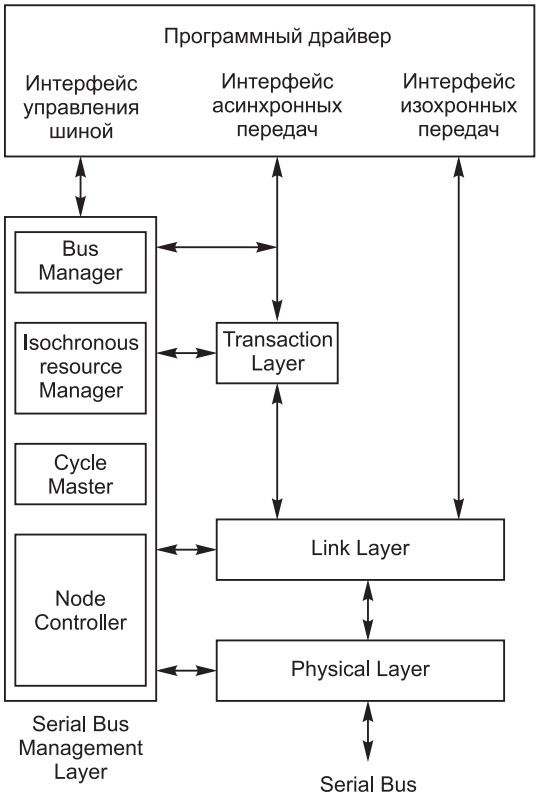


Рис. 17.4. Трехуровневая структура FireWire

- предоставление сервисов канальному уровню;
- связь сегментов сети в единую шину, если он имеет более одного порта IEEE 1394 (трансляцию сигналов между своими портами);
- предоставление питания по кабельной шине.

Физический уровень допускает несколько вариантов физического интерфейса (см. главу 22):

- *кабельная шина 1394/1394a* с DS-кодированием, поддерживающая скорости S100, S200 и S400 на экранированной витой паре;
- *кросс-шина* (backplane serial bus, BPSB) с DS-кодированием, поддерживающая скорости S50 и S100 при связи узлов в пределах шасси;
- *кабельная шина 1394b* с кодированием 8B10B (так называемый бета-режим), поддерживающая скорости от S100 до S1600 и разные варианты кабелей: экранированная витая пара, неэкранированная (UTP-5), пластиковое и стеклянное многомодовое оптоволокно.

◆ *канальный уровень* LINK (Link Layer) из данных физического уровня формирует пакеты и выполняет обратные преобразования. При этом он формирует (при передаче) и проверяет (при приеме) формат пакета и контрольные поля

(CRC-коды). Он обеспечивает асинхронный обмен узлов дейтаграммами (пакетами запросов, ответов и пакетами квитирования), а также передачу и прием изохронных потоков. Канальный уровень отвечает за адресацию — выявление и прием пакетов, предназначенных данному узлу:

- широковещательных;
- по адресу узла (для асинхронных транзакций);
- по номеру канала (для изохронных и асинхронных потоков);

◆ *уровень транзакций* (Transaction Layer) предоставляет приложениям сервисы для асинхронных обменов с регистрам и памятью любых узлов сети, состоящей из множества шин, объединенных мостами. Операции обменов включают *чтение*, *запись*, *блокированные операции* (чтение-модификация-запись). Операцией записи в специальный регистр узла можно вызвать *прерывание* для данного узла, при этом биты данного регистра будут нести информацию о соответствующих условиях прерывания. Уровень транзакций реализует протокол запросов-ответов, соответствующий стандарту ISO/IEC 13213:1994 (ANSI/IEEE 1212, редакция 1994 года) архитектуры регистров управления и состояния CSR (Control and Status Register) для микрокомпьютерных шин. Это облегчает связь шины 1394 со стандартными параллельными шинами. На уровне транзакций выполняется часть действий по обработке ошибок и организации повторов передач (канальный уровень только сообщает об обнаруженных ошибках).

Драйверы прикладного и системного ПО для организации асинхронных транзакций пользуются сервисами уровня транзакций. В плане обработки ошибок уровень транзакций предоставляет только уведомления об успехе или неудаче выполнения транзакции. В последнем случае организация повторов ложится на драйвер. Для изохронных передач (и потоковых асинхронных) драйвер пользуется сервисами канального уровня, который в данном случае обеспечивает лишь передачу пакетов, прием пакетов требуемых каналов с индикацией наличия или отсутствия ошибки в данных.

Управление шиной (Bus management) затрагивает все вышеперечисленные уровни. Шина может иметь различные степени управляемости: полностью управляемая, частично управляемая (с диспетчером изохронных ресурсов, необходимым, если есть узлы с изохронным обменом) и даже неуправляемая шина. Различные аспекты управления рассмотрены в главе 21.

Узел может быть вырожденным до простого кабельного концентратора — иметь только компоненты физического уровня. Его многопортовый PHY будет выполнять функции повторителя, не нуждаясь в вышестоящих уровнях.

Интерфейс IEEE 1394 реализуется аппаратно-программными средствами устройства. Аппаратная часть FireWire обычно состоит из двух специализированных микросхем — трансивера физического уровня (*PHY Transceiver*) и моста связи с микропроцессорной шиной (*LINK Chip*). Интерфейс между ними описан стандартом IEEE 1394. Микросхема LINK выполняет все функции канального уровня и часть функций уровня транзакций; остальная часть уровня транзакций выполняется программно. Микросхема PHY выполняет сигнальное кодирование-декодирование данных, распознавание адресов, функции арбитража, а также трансляцию

сигналов между своими портами. Уровень PHY достаточно автономен, все «общественнополезные» функции узла он может выполнять и при отключенных вышестоящих уровнях. Физический уровень может быть (но не обязательно) гальванически развязан с канальным уровнем. В бета-режиме (1394b) гальваническая развязка (более эффективная) возможна на уровне кабельного интерфейса. Гальваническая развязка необходима для предотвращения возникновения паразитных контуров общего провода, которые могут появиться через провода защитного заземления блоков питания.

Физический и канальный уровни могут различаться в плане поддерживаемых скоростей передачи. Если многопортовый PHY поддерживает более высокие скорости, чем LINK, то он способен транслировать высокоскоростные пакеты между своими портами. Однако скорость, на которой сам узел может общаться с остальными узлами шины, определяется самым слабым звеном в данной паре PHY-LINK. В этом случае она будет ограничиваться возможностями LINK-уровня; эти возможности могут зависеть от организации узла. Для компьютера, подключаемого к 1394, поддерживаемая скорость LINK зависит от производительности шины, которой подключен адаптер, и производительности его контролера памяти. Физический уровень для различных устройств практически одинаков, различия касаются поддерживаемых скоростей передачи, а в 1394b — и используемой среды передачи (разновидностей медных и оптических кабелей). Канальный уровень существенно зависит от прикладной части устройства — микропроцессора, на котором базируется устройство, и интерфейса подключения канального уровня.

Конфигурирование шины

Конфигурирование шины происходит автоматически при включении питания, при подключении/отключении устройств, а также по инициативе какого-либо узла. Процесс конфигурирования выполняется только с участием уровней PHY — вышестоящие компоненты могут быть отключены. Процесс состоит из трех последовательных этапов:

- сброс (Bus Reset) — приведение всех узлов в исходное (несконфигурированное) состояние;
- идентификация дерева — построение иерархической структуры шины;
- самоидентификация узлов — назначение физических адресов и сообщение ими своих свойств, относящихся к шине. На этом этапе выбирается (не обязательно) и узел-диспетчер изохронных ресурсов.

После этого шина переходит в состояние *Arbitration*, в котором она готова к выполнению асинхронных транзакций и изохронных передач. Теперь может быть выбран диспетчер шины, который должен подать команду на включение LINK-уровня всех узлов (с учетом бюджета мощности питания от шины). При отсутствии диспетчера шины эту задачу исполнит диспетчер изохронных ресурсов.

С этого момента начинается регулярная работа в плане асинхронных транзакций. Для начала изохронного вещания узлы должны получить у диспетчера изохронные ресурсы — номер канала и доступную полосу вещания. Сброс на шине крат-

ковременно прерывает выполнение изохронных передач, но сразу по завершении самоидентификации узлов (переходу в состояние *Arbitration*) узлы, имевшие изохронные ресурсы до сброса, могут начинать изохронные передачи.

Весь процесс конфигурирования занимает доли секунды. На время сброса и конфигурирования передача полезного трафика останавливается, все ожидающие транзакции сбрасываются. После конфигурирования могут измениться адреса узлов. Номера изохронных каналов, вещавших до сброса, могут и сохраниться — ранее вещавшие узлы имеют приоритет при выделении изохронных ресурсов, что позволяет минимизировать потери вещания.

Идентификация дерева

Изначально (по включению питания или сбросу) шина (совокупность соединенных узлов) имеет плоскую (неоформленную) структуру (рис. 17.5, *а*). В ходе инициализации первым делом автоматически (взаимодействием только уровней PHY) выполняется *идентификация дерева* (tree identification), в результате чего шина обретает форму перевернутого дерева (рис. 17.5, *б*). В этом дереве имеется:

- *корень* (root), являющийся «верховным арбитром». Корень выбирается автоматически в зависимости от топологии шины; при необходимости стать корнем программно можно заставить любой узел (см. главу 20);
- *листья* (leaf) — конечные узлы, подключенные к шине только одним портом;
- *ветки* (branch) — узлы, подключенные несколькими портами и находящиеся в иерархии между корнем и листьями.

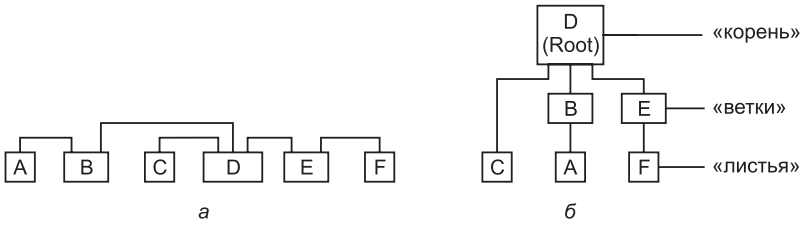


Рис. 17.5. Топология шины: *а* — после сброса; *б* — после идентификации дерева

И корень и ветки являются промежуточными узлами, обеспечивающими трансляцию трафика. В первоначальной (плоской) структуре все порты узлов равноправны. После идентификации дерева порты дифференцируются в зависимости от их направленности в дереве:

- «родительский», или p-порт (parent port), направлен в сторону корня (вверх);
- «дочерний», или c-порт (child port), направлен в сторону листьев (вниз).

Родительским может быть только один из портов узла. Если на шине присутствуют только два узла, то одно из них (определяемое случайным образом) станет корнем с c-портом, другое — листом с p-портом. В процессе идентификации дерева участвуют только те порты, к которым подключены другие узлы (физический уровень способен определить состояние подключения).

Тип порта влияет только на распространение сигналов арбитража (см. главу 19). Данные, принимаемые по одному порту, узел транслирует на все остальные порты, к которым подключены узлы-соседи, но с учетом скорости: высокоскоростные пакеты данных не транслируются на те порты, для которых установлена более низкая скорость работы. Данные, исходящие из узла, транслируются на все его порты, к которым подключены устройства и которые способны работать на скорости передачи данного пакета.

Самоидентификация узлов

После идентификации дерева выполняется *самоидентификация узлов* (Self Identification), в результате которой все узлы шины получают уникальные адреса — *физические идентификаторы* (`phy_ID`). В процессе самоидентификации с помощью арбитража каждый узел последовательно получает право на передачу широковещательного *пакета самоидентификации*. В этом пакете содержится идентификатор узла, число его портов, их состояние и назначение, скоростные возможности устройства и некоторые другие параметры. Устройство назначает себе идентификатор само, исходя из числа ранее «услышанных» им пакетов самоидентификации от других устройств. Первое устройство, которому предоставили право на передачу, получает `phy_ID = 0`; последним самоидентифицируется корень — у него будет максимальное значение `phy_ID` (не превышающее 62). После посылки пакета самоидентификации узел обменивается со своим партнером-«родителем» сигналами, идентифицирующими максимальную скорость работы. Это позволяет непосредственным партнерам попарно согласовать свои скоростные возможности.

По информации, собранной из всех пакетов самоидентификации, любой узел может построить *карту топологии сети* (topology map), а на ее основе и *карту достижимых скоростей* обмена (speed map) между любой парой узлов. Эти карты должен строить и публиковать узел-диспетчер (менеджер) шины, предоставляя доступ к ним любым узлам. Кроме того, информация пакетов самоидентификации используется для управления питанием от шины и оптимизации трафика.

Спецификации IEEE 1394

Стандарт для высокопроизводительной последовательной шины (High Performance Serial Bus), получивший официальное название IEEE 1394, был принят в 1995 году. Стандарт основан на шине FireWire, используемой фирмой Apple Computer еще с 1986 года в качестве дешевой альтернативы SCSI в своих компьютерах. Название FireWire («огненный провод») теперь применяется и к реализациям IEEE 1394, оно сосуществует с кратким обозначением: «*1394*». Другое название того же интерфейса — *iLink*, а иногда и *Digital Link* — используется фирмой Sony применительно к устройствам бытовой электроники. *MultiMedia Connection* — имя, используемое в логотипе 1394 High Performance Serial Bus Trade Association (1394TA).

К устройствам с интерфейсом IEEE 1394 относится большое количество спецификаций, описывающих не только собственно интерфейс последовательной шины,

но и использование этой шины для транспортировки прикладных данных разного назначения, наборы команд и форматы данных. Основной источник информации по технологии, стандартам и продуктам — сайт ассоциации High Performance Serial Bus Trade Association (http://www.1394ta.org). Спецификация IEEE 1394 официально доступна через http://www.ieee.org (платно). Вопросы лицензирования и интеллектуальной собственности — на сайте http://www.1394la.com.

Документ *IEEE 1394-1995 Standard for a High Performance Serial Bus* определяет архитектуру шины, основанную на трехуровневой модели, и протоколы, обеспечивающие автоматическое конфигурирование, арбитраж и передачу изохронного и асинхронного трафика. В стандарте определены три возможные скорости передачи сигналов по кабелям: 98,304, 196,608 и 393,216 Мбит/с, которые округляют до 100, 200 и 400 Мбит/с и обозначаются как S100, S200 и S400 соответственно. Стандартизованы кабель и 6-контактный разъем, позволяющий передавать сигналы и питание.

В *дополнении IEEE 1394a-2000 IEEE Standard for a High Performance Serial bus (Supplement)* введен ряд усовершенствований:

- приняты меры для сокращения потерь времени при подключении устройств;
- введены механизмы ускоренного арбитража;
- разрешена конкатенация (соединение) пакетов, передаваемых на разных скоростях;
- расширены средства управления энергопотреблением и введена возможность приостановки и запрета портов;
- введена возможность общения с регистрами PHY удаленного узла;
- введен миниатюрный 4-контактный разъем (без питающих линий);
- утвержден стандартный интерфейс PHY-LINK и возможная схема гальванической развязки в этом интерфейсе;
- уточнены исходные спецификации ради обеспечения более высокого уровня совместимости реализаций стандарта.

Новых скоростей в этом стандарте не появилось; изменения вводились с учетом обеспечения обратной совместимости с устройствами, отвечающими исходному стандарту.

Дополнения IEEE 1394b (2002 год) в основном касаются повышения скорости и дальности передачи:

- введены скорости S800, S1600 и архитектурная поддержка S3200;
- введен новый (для 1394) метод сигнализации и соответствующий бета-режим работы портов. Для передачи используется пара встречных однонаправленных линий и кодирование 8B/10B, широко используемое в современных коммуникационных технологиях;
- введен новый метод арбитража (BOSS), повышающий эффективность использования пропускной способности шины за счет исключения простоев шины (зазоров арбитража);

- введены новые типы среды передачи для бета-режима:
  - пара пластиковых или стеклянных оптических волокон для расстояний до 50 или до 100 м;
  - кабель UTP-5 (используются 2 пары) с разъемами RJ-45 для расстояний до 100 м с трансформаторной гальванической развязкой.
- Введен миниатюрный 9-контактный разъем для коротких дистанций, позволяющий передавать данные со скоростью до 3,2 Гбит/с и подавать питание по шине.

Совместимость с 1394 и 1394a обеспечивается «двуязычными» уровнями PHY, способными работать с разными методами сигнализации на своих разных портах. При этом возможно построение гибридной шины, состоящей из одного или нескольких «облаков» узлов с бета-сигнализацией, связанных друг с другом фрагментами с традиционной сигнализацией.

Проект стандарта P1394.1 относится к сетям, состоящим из нескольких шин IEEE 1394. Основа построения сети — двухпортовый мост, способный передавать трафик между соединяемыми шинами. При этом мост для процессов сброса и автоконфигурирования, прерывающих передачу полезного трафика на длительное время, изолирует шины друг от друга. Возможны и многопортовые мосты, которые с протокольной точки зрения являются комбинацией двухпортовых.

В основу IEEE 1394 положен протокол запросов-ответов архитектуры регистров управления и состояния для микрокомпьютерных шин, описанной в стандарте ISO/IEC 13213:1994 Control and Status Register Architecture for Microcomputer Busses (CSR-архитектура). Ревизия этого стандарта предполагается в проекте *P1212r*. Ревизия касается возможностей бесконфликтного расширения форматов содержимого памяти конфигурации, а также регистров CSR. Планируется привести спецификацию CSR в соответствие с практикой ее использования.

К управлению энергопотреблением IEEE 1394 относится спецификация 1394 TA Power Spec, состоящая из трех частей:

- Part 1: Cable Power Distribution — описание механизма подачи питания через кабельную шину, форматы структур данных и регистров, описывающих потребности в питании и возможности поставки питания на шину;
- Part 2: Suspend/Resume — описание механизмов приостановки и возобновления работы отдельных портов и частей кабельной шины;
- Part 3: Power State Management — описание четырех возможных уровней потребления узла и его блоков и механизмов управления сменой уровней.

ГЛАВА 18

Передача данных по шине IEEE 1394

Шина IEEE 1394 поддерживает два типа передач данных:

- *асинхронные передачи* без каких-либо требований к скорости и задержке доставки. Целостность данных контролируется CRC-кодом. По адресации различают две разновидности:
  - *направленная асинхронная передача* адресуется конкретному узлу, гарантированную доставку обеспечивает механизм квитирования и повторов;
  - *широковещательная асинхронная передача* адресуется всем узлам и выполняется без гарантии доставки (квитирование и повторы не применяются).
- *Изохронные передачи* с гарантированной пропускной способностью. Целостность данных контролируется CRC-кодом, гарантии доставки нет — квитирование и повторы не применяются.

Направленные асинхронные передачи являются основой для выполнения *асинхронных транзакций* — логически завершенных обменов между парами узлов. Протокол шины позволяет узлам с помощью асинхронных транзакций обращаться к памяти (регистрам) друг друга в режиме прямого доступа (DMA). При этом они не нуждаются в памяти и процессорных ресурсах «третьих лиц»[1].

Изохронные передачи представляют собой *потоки пакетов* данных. Эти передачи ведутся широковещательно и адресуются через *номер канала*, передаваемый в каждом пакете. На шине может быть организовано до 64 изохронных каналов, передачи всех каналов «слышат» все устройства шины, но из всех пакетов принимают только данные интересующих их каналов. По шине могут передаваться и *асинхронные потоки*, для которых, в отличие от изохронных, не предоставляется гарантированная полоса пропускания.

Асинхронные транзакции

Асинхронные транзакции на шине IEEE 1394 реализуют подмножество операций протокола запросов-ответов, соответствующего стандарту архитектуры регистров

[1] Для сравнения: между устройствами USB взаимодействие возможно только через память хост-компьютера и лишь под управлением его процессора.

управления и состояния CSR[1] (Control and Status Register) для микропроцессорных шин. Асинхронные передачи обеспечивают *три типа транзакций*:

- *чтение* (Read);
- *запись* (Write);
- *блокированные операции* «чтение-модификация-запись» (Lock).

В каждой асинхронной транзакции участвуют два устройства: *запросчик* (requester) и *ответчик* (responder). Протокол запросов-ответов для этих типов транзакций иллюстрирует рис. 18.1. Каждая асинхронная транзакция состоит из двух *субакций* (subaction) — шагов исполнения:

- *запрос* (request) — передается тип, адрес транзакции и, возможно, данные;
- *ответ* (response) — передается состояние выполнения (успех-неуспех) транзакции и, возможно, данные.

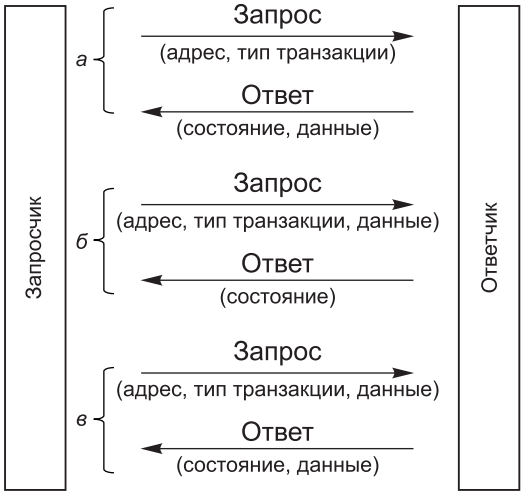


Рис. 18.1. Протокол запросов-ответов: *а* — чтение; *б* — запись; *в* — чтение-модификация-запись

Реализация данного протокола в последовательной шине, где весьма вероятны ошибки при передаче информации, потребовала введения квитирования (и механизма повторов) для каждой субакции[2]. Каждая субакция состоит из основного пакета (запроса или ответа) и пакета-квитанции. В общем случае для того, чтобы начать передачу пакета, узел должен получить на это право — выиграть арбитраж. Передача пакетов квитирования выполняется без арбитража — адресованный узел должен послать квитанцию через короткий интервал (*ack_gap*) после получения пакета.

---

[1] ISO/IEC 13213:1994 (ANSI/IEEE 1212, редакция 1994 года).

[2] В архитектурной модели CSR квитирование не предусмотрено.

Формы выполнения транзакций

Асинхронные транзакции могут выполняться в разных формах, различающихся числом субакций, необходимостью квитирования и способом получения права передачи (количеством операций арбитража). Эти формы иллюстрирует рис. 18.2. На рисунке показаны операции арбитража (arb), короткие зазоры подтверждений (ack gap), длинные зазоры субакций (subaction gap), признаки начала (Data Prefix) и конца (Data End) пакетов. Механизм арбитража и зазоры рассмотрены в гл. 19.

Расщепленные транзакции используются при обращении к медленным устройствам (рис. 18.2, *а*). Здесь между запросом и ответом могут вклиниваться другие транзакции на шине, так что перед ответом требуется арбитраж и зазор. В этой форме могут выполняться запись (Split Write), чтение (Normal Split Read) и блокированные транзакции (Normal Split Lock);

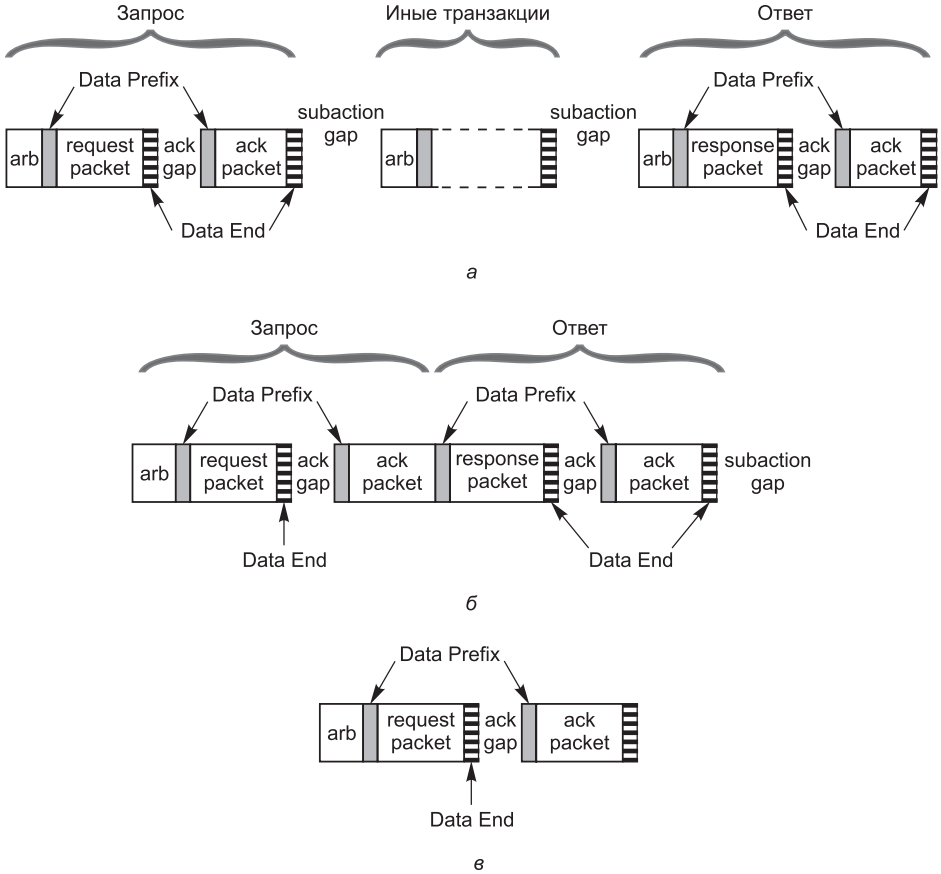


Рис. 18.2. Формы асинхронных транзакций: *а* — расщепленная; *б* — соединенная расщепленная; *в* — объединенная

Соединенные расщепленные транзакции используются при обращении к быстрым устройствам (рис. 18.2, *б*). Здесь пакет ответа передается сразу за пакетом квитирования (отделяясь только префиксом данных), так что дополнительный арбитраж (и зазор) не требуется. В этой форме могут выполняться запись (Concatenated Split Write), чтение (Concatenated Read) и блокированные транзакции (Concatenated Lock);

Объединенные транзакции используются в тех случаях, когда не требуется содержательного ответа (рис. 18.2, *в*). В этой форме возможна только запись (Unified Write) — самый быстрый вариант, предназначенный для тех операций, которые заведомо проходят успешно (или успех не волнует запросчика).

Типы транзакций

Транзакции записи начинаются с пакета запроса, в котором передается полный адрес назначения, идентификатор источника, которому нужно будет послать ответ, и сами данные записи. Пакет ответа записи несет код результата выполнения.

Транзакции чтения в пакете запроса несут полный адрес назначения, длину запрашиваемого блока и идентификатор источника. Запрашиваемая длина определяется кодом типа транзакции (чтение квадлета или блока), а для чтения блока — полем длины. Пакет ответа чтения несет код результата выполнения и собственно данные чтения.

Блокированные транзакции чтения-модификации-записи в пакете запроса кроме адреса назначения несут значения *аргумента* и *данных* транзакции. Эти значения могут быть как 32, так и 64-битными (одинаковой длины), аргумент может и отсутствовать. Расширенный код транзакции задает тип выполняемой операции (табл. 18.1) и, соответственно, наличие или отсутствие аргумента. Поле длины определяет разрядность аргумента и данных и может принимать значение 4 (только 4-байтные данные), 8 (аргумент и данные по 4 байта или только данные размером 8 байт) или 16 (аргумент и данные по 8 байт). Пакет ответа по формату аналогичен ответу на чтение блока, где в поле данных возвращаются старые данные (считанные перед модификацией). Длина блока данных ответа может составлять 4 или 8 байт.

Таблица 18.1. Типы блокированных транзакций

Расширенный код транзакции	Имя	Назначение
0000h	Резерв	Резерв
0001h	*mask_swap*	Биты целевой ячейки, которым соответствуют единичные значения в `arg_value`, заменяются на биты из `data_value`
0002h	*compare_swap*	Содержимое целевой ячейки заменяется на `data_value`, если ее прежнее значение совпадает с `arg_value`
0003h	*fetch_add*	Содержимое целевой ячейки складывается с `data_value`; числа рассматриваются как целые, целевой адрес указывает на самый старший байт числа (`big endian`)
0004h	*litle_add*	То же, но целевой адрес указывает на самый младший байт числа (`litle endian`)

Расширенный код транзакции	Имя	Назначение
0005h	*bounded_add*	Если содержимое целевой ячейки не равно `arg_value`, то она заменяется на сумму прежнего и `data_value`; иначе целевая ячейка не изменяется
0006h	*wrap_add*	Если содержимое целевой ячейки не равно `arg_value`, то она заменяется на сумму прежнего и `data_value`; иначе она заменяется на `data_value`
0007h	*vendor*	Назначение определяется разработчиком
0008-FFFFh	резерв	Резерв

Пакеты асинхронных транзакций

Пакеты запросов и ответов асинхронных передач для различных транзакций имеют форматы, представленные на рис. 18.3–18.5. Пакет состоит из заголовка и необя зательно блока данных. Ниже перечислено назначение полей, используемых в *заголовках пакетов* различных типов:

- 64-битный адрес назначения (см. рис. 17.2, состоящий из полей `destination_ID` (10-битный номер шины и 6-битный номер узла) и `destination_Offset` (48-битный адрес в пространстве целевого узла);
- метка транзакции `tl` (Transaction Label, 6 бит), с помощью которой запросчик определяет, к какому его запросу относится полученный ответ. Если метка используется, то ответчик помещает в пакет ответа метку, полученную запросчиком (аналогично тегам на PCI-X);
- код повтора `rt` (Retry Code, 2 бита), по которому определяется, является ли данный пакет первой попыткой транзакции; для повторной попытки значение поля задается в зависимости от кода квитирования, полученного от целевого устройства: `rt` = 00 — первая попытка, `rt` = 01 — *Retry_x*, `rt` = 10 — *Retry_A*, `rt` = 11 — *Retry_B*;
- поле типа транзакции (субакции) `tcode` (4 бита), определяющий назначение пакета и его формат в соответствии с табл. 18.2;
- поле приоритета `pri` (Priority, 4 бита, используется только для кросс-шины), в кабельных сетях не используется;
- поле `rcode` (4 бита) содержит код результата выполнения операции (табл. 18.3);
- поле `data_length` (16 бит) содержит длину блока (в байтах);
- поле расширенного кода транзакции `extended_tcode` (16 бит) уточняет выполняемую операцию (используется не для всех операций);
- проверочный код `header_CRC` контролирует целостность заголовка.

Блок данных `data_block`, следующий за заголовком, и его проверочный код `data_CRC` присутствуют в пакетах с `tcode` = 1, 7, 9, A. Блок содержит целое

число квадлетов; при необходимости остаток последнего квадлета заполняется нулями. Максимальный размер блока определяется скоростью передачи (табл. 18.4).

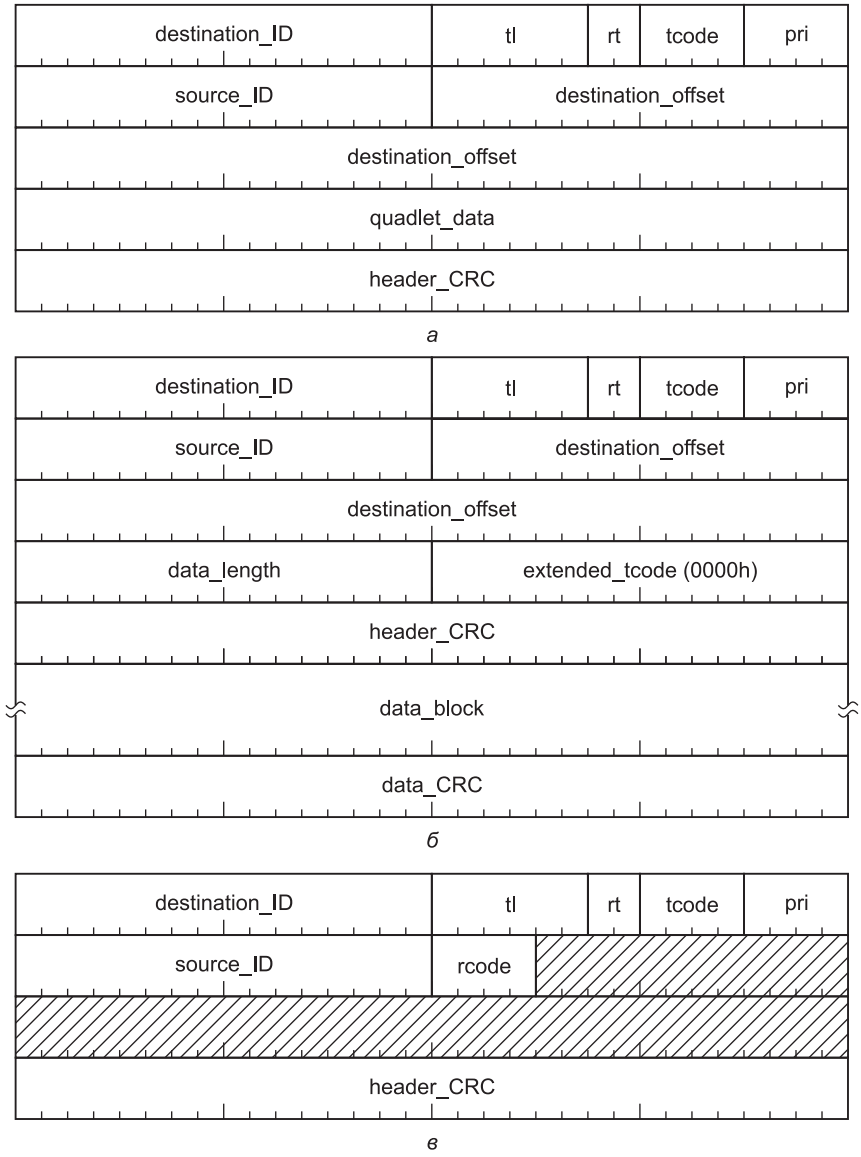


Рис. 18.3. Пакеты транзакций записи: *а* — запрос записи квадлета; *б* — запрос записи блока; *в* — ответ

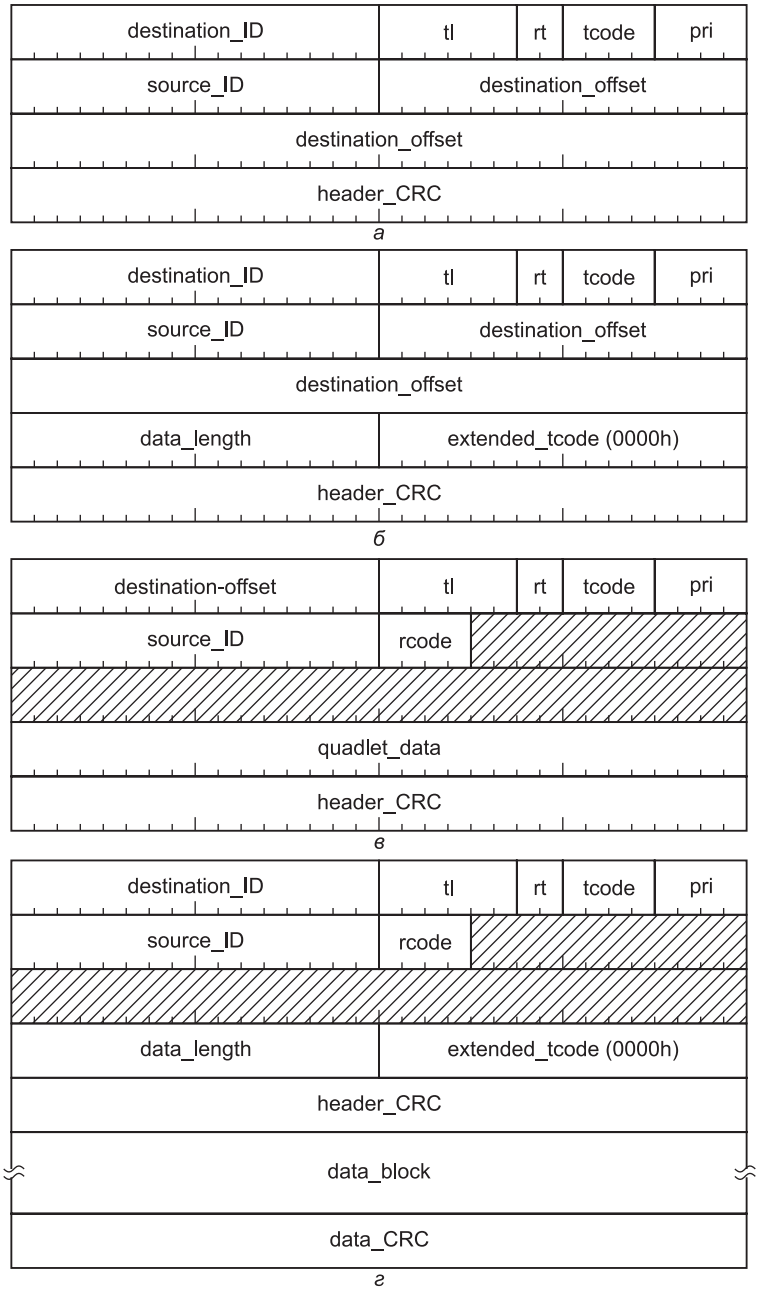


Рис. 18.4. Пакеты транзакций чтения: *а* — запрос чтения квадлета; *б* — запрос чтения блока; *в* — ответ чтения квадлета; *г* — ответ чтения блока

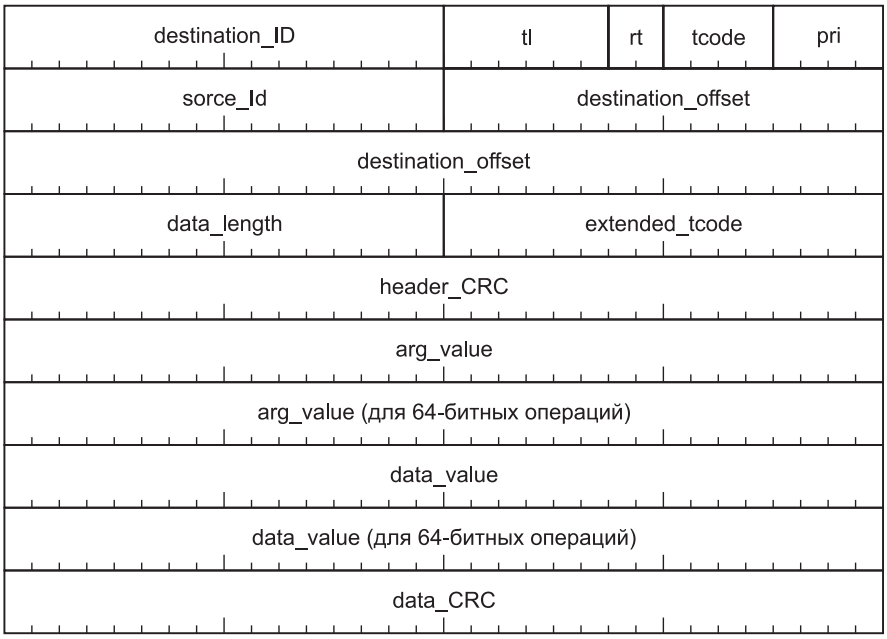


Рис. 18.5. Пакет запроса блокированных транзакций

Таблица 18.2. Типы пакетов асинхронных транзакций

tcode	Транзакция (субакция)
0	Запрос записи квадлета данных (*Write Request*)
1	Запрос записи блока данных (*Write Request*)
2	Ответ записи (*Write Response*)
3, C, D, F	Резерв
4	Запрос чтения квадлета данных (*Read Request*)
5	Запрос чтения блока данных (*Read Request*)
6	Ответ чтения квадлета данных (*Read Response*)
7	Ответ чтения блока данных (*Read Response*)
8	Начало цикла (*Cycle Start*)
9	Запрос блокированной транзакции (*Lock Request*)
A	Пакет асинхронного потока (*Asynchronous Streaming Packet*)
B	Ответ блокированной транзакции (*Lock Response*)
E	Не стандартизован

Допустимый размер блока данных, передаваемых в одном пакете, зависит от скорости (табл. 18.4).

Таблица 18.3. Коды результата выполнения

rcode	Результат
0	*resp_comlete*, успешное выполнение запроса
1–3	резерв
4	*resp_conflict_error*, отвечающий агент обнаружил конфликт ресурсов (запрос можно повторить позже)
5	*resp_data_error*, аппаратная ошибка, данные недоступны
6	*resp_type_error*, недопустимое значение полей в пакете запроса
7	*resp_address_error*, указанные адрес недоступен
8...Fh	резерв

Таблица 18.4. Допустимый размер блока данных пакета

Скорость	Максимальный размер для асинхронных передач, байт	Максимальный размер для изохронных передач, байт
S100	512	1024
S200	1024	2048
S400	2048	4096
S800	4096	8192
S1600	4096	16 384
S3200	4096	32 768

Пакет квитирования имеет длину всего 1 байт (рис. 18.6), в нем содержится 4-битный *код квитирования* `ack_code` (табл. 18.5), достоверность которого проверяется по его двоичному дополнению `ack_parity`.

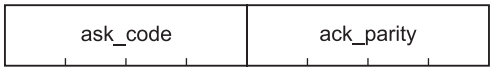


Рис. 18.6. Пакет квитирования

Таблица 18.5. Коды квитирования

ack_code	Имя	Значение
0	Резерв	Резерв
1	*ack_complete*	Пакет успешно принят. Если это квитанция пакета запроса, то пакет ответа не предполагается
2	*ack_pending*	Пакет запроса успешно принят, пакет ответа будет послан (для пакетов ответа не используется)
3	Резерв	Резерв
4	*ack_busy_x*	Пакет не принят из-за занятости узла (возможен успех при повторной попытке)

продолжение

Таблица 18.5 (*продолжение*)

ack_code	Имя	Значение
5	*ack_busy_A*	Пакет не принят из-за занятости узла. Данные будут приняты при следующем повторе (фаза A), когда узел освободится
6	*ack_busy_B*	Пакет не принят из-за занятости узла. Данные будут приняты при следующем повторе (фаза B), когда узел освободится
7–Ah	Резерв	Резерв
Bh	*ack_tardy*	Пакет не принят из-за неготовности узла, находящегося в состоянии *standby*; попытку можно повторить через некоторое время (1394a)
Ch	*ack_conflict_error*	Пакет не принят из-за конфликта ресурсов (1394a)
Dh	*ack_data_error*	Пакет не принят из-за ошибки данных (CRC или несоответствие длины пакета заявленной)
Eh	*ack_type_error*	Пакет не принят из-за недопустимого типа (неверные параметры, неподдерживаемый тип транзакции)
Fh	*ack_address_error*	Пакет не принят из-за недоступности указанного адреса (`Dest_Offset`) в целевом узле(1394a)

Обработка ошибок и механизм повторов

Успешному выполнению асинхронных транзакций может помешать ряд причин:

- занятость целевого узла (заполнение его входных и выходных очередей);
- заблокированность целевого узла (он находится в процессе выполнения блокированной транзакции от другого узла);
- ошибки при передаче пакетов.

Уровень транзакций предоставляет драйверам надежные сервисы, уведомляющие драйвер об успехе или неуспехе транзакции. Однако средства IEEE 1394 (уровень транзакций) обеспечивают автоматические повторы только для случая занятости целевого узла.

Протоколы повторов при занятости

Для организации повторов из-за занятости используется один из двух протоколов, который выбирает целевое (занятое) устройство: однофазный или двухфазный. Первая попытка посылки пакета (запроса или ответа) выполняется всегда одинаково: в заголовке пакета поле `rt` = 00. Если на этот пакет приходит квитанция с одним из кодов, означающим занятость — 4 (*ack_busy_x*), 5 (*ack_busy_A*) или 6 (*ack_busy_B*), то узел организует последующие попытки передачи этого пакета, устанавливая в поле `rt` значение: 01 (*Retry_x*), 10 (*Retry_A*) или 11 (*Retry_B*) соответственно. Попытки повторов выполняются с обычным (справедливым) арбитражем, «предел терпения» определяется в зависимости от используемого протокола повторов.

Протокол однофазных повторов (Single Phase Retry) использует только код квитирования 4 (*ack_busy_x*), в ответ на который повторные попытки передачи делаются с кодом `rt` = 01 (*Retry_x*). Максимальное число повторов одного пакета ограничено 4-битным полем `retry_limit` регистра `BUSY_TIMEOUT` (смещение 210h в пространстве регистров CSR, рис. 18.7). По достижении этого лимита повторные попытки прекращаются, и уровень транзакций сообщает драйверу о невозможности посылки данного пакета из-за занятости адресуемого узла.

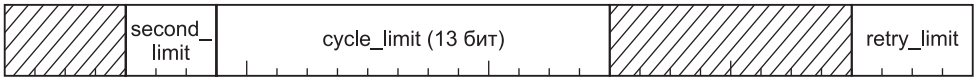


Рис. 18.7. Формат регистра BUSY_TIMEOUT

Протокол двухфазных повторов (Dual Phase Retry) позволяет несколько упорядочить обслуживание занятым (перегруженным) узлом пакетов разных транзакций. Для этого вводится чередование двух фаз — фазы A и фазы B. Если занятый узел получает пакет первой попытки (`rt` = 00), он на него отвечает кодом квитирования 5 (*ack_busy_A*) и ожидает его повторного прихода с кодом `rt` = 10 (*Retry_A*). Теперь этот узел находится в фазе A и будет пытаться обслуживать только пакеты с `rt` = 10. На все первые попытки других транзакций узел будет отвечать кодом квитирования 6 (*ack_busy_B*), что заставит узлы, их посылающие, делать попытки повторов с `rt` = 11 (*Retry_B*), в данной фазе не обслуживаемые. Только после того, как наш узел сумеет обслужить пакет с `rt` = 10 и в течение четырех подряд интервалов справедливости не будет получать другие пакеты с `rt` = 10 (*Retry_A*), он перейдет в фазу B и начнет обслуживать пакеты с `rt` = 11 (*Retry_B*). При этом на новые первичные пакеты он будет отвечать кодом квитирования 5 (*ack_busy_A*). После обслуживания пакетов фазы B узел снова перейдет в фазу A — получается подобие игры в пинг-понг. Предел «терпения» узла, посылающего повторные пакеты по двухфазному протоколу, определяется уже по времени: ограничивается время, отсчитываемое от момента первой попытки передачи. Это время не должно превышать значение, заданное полями `seconds_limit` и `cycle_limit` в том же регистре `BUSY_TIMEOUT`. 13-битный счетчик, соответствующий полю `cycle_limit`, считает по модулю 8000 (полный оборот за 1 секунду).

Обработка ошибок передачи

Для обеспечения надежных транзакций достоверность передачи заголовков и полей данных пакетов контролируются CRC-кодами. Пакеты запросов и ответов, принятые с обнаруженной ошибкой в заголовке, игнорируются (достоверной информации об их источнике и адресате назначения нет). Об остальных ошибках узел, которому адресован пакет запроса или ответа, сообщает его источнику указанием соответствующего кода в пакете квитирования (табл. 18.5). Пакеты-квитанции контролируются проще (по двоичному дополнению кода ответа), неверные квитанции игнорируются. Обработкой ошибок занимается прикладной драйвер, которому уровень транзакций сообщает результат выполнения. Канальный уро-

вень принятые пакеты только проверяет на корректность формата и CRC; он передает уровню транзакций все пакеты, адресованные данному узлу, но с соответствующими признаками корректности или ошибки.

Типовая асинхронная транзакция состоит из двух субакций (запроса и ответа), каждая из которых выполняется с квитированием. Таким образом, транзакция состоит из четырех пакетов, каждый из которых при передаче может оказаться поврежденным. Рассмотрим поэтапно возможные ошибки передачи в типовой транзакции.

Пакет запроса пришел к адресованному узлу с ошибкой CRC данных:

- *действия узла-ответчика*: канальный уровень передает его уровню транзакций поврежденный пакет с признаком ошибки. Уровень транзакций отбрасывает пакет и через канальный уровень посылает квитанцию с кодом ошибки. Прикладной драйвер никаких уведомлений не получает;
- *действия узла-запросчика*: канальный уровень принимает эту квитанцию и передает ее код уровню транзакций, который информирует прикладной драйвер о неудаче. Какая на это последует реакция — зависит от приложения, которое знает, что до ответчика запрос не дошел.

Пакет запроса пришел нормально, но *пакет квитирования* «потерялся» (канальный уровень запросчика не дождался корректной квитанции):

- *действия узла-запросчика*: канальный уровень посылает об этом уведомление своему уровню транзакций, который сообщит прикладному драйверу об отсутствии квитанции. Реакция опять же зависит от приложения, но состояние ответчика (пришел к нему запрос или нет) ему неизвестно.

Пакет ответа пришел к запросчику с ошибкой CRC данных:

- *действия узла-запросчика*: канальный уровень передает пакет уровню транзакций поврежденный пакет с признаком ошибки. Уровень транзакций отбрасывает пакет и через канальный уровень посылает квитанцию с кодом ошибки. Прикладному драйверу об этом не сообщается, для него данный запрос остается в состоянии ожидания ответа (пока позволяет тайм-аут ожидания ответа);
- *действия узла-ответчика*: канальный уровень принимает эту квитанцию и передает ее код уровню транзакций, который информирует прикладной драйвер об ошибке передачи ответа. Прикладной драйвер может повторить ответ — запросчик, возможно, его еще ждет.

Пакет ответа не пришел к запросчику до срабатывания тайм-аута ожидания ответа. Об этом информируется прикладной драйвер, который может решить повторить запрос. Однако этот повтор может иметь побочные эффекты, если, например, запрашивалось чтение регистра (или памяти), не допускающего предвыборки: одно чтение уже было сделано, правда, результаты потерялись.

Пакет ответа пришел нормально, но *пакет квитированиия* «потерялся». Об этом уведомляется драйвер ответчика, который обрабатывает эту ситуацию в зависимости от приложения. Дошел ли при этом ответ до запросчика — неизвестно.

Взаимодействие драйвера с уровнем транзакций

Уровень транзакций взаимодействует с прикладным драйвером четырьмя примитивами сервисов:

- *запрос* (request), используемый запросчиком транзакции для инициирования субакции запроса. В запросе передается:
  - код типа транзакции (Read, Write, Lock с указанием операции);
  - адрес получателя;
  - длина данных;
  - данные (для Write и Lock);
  - скорость передачи.

Передавая запрос канальному уровню, уровень транзакций добавляет еще метку транзакции и код повтора.

- *Индикация* (indication), уведомляющая ответчика о приходе запроса. Драйверу ответчика передается:
  - код типа транзакции (Read, Write, Lock с указанием операции);
  - адрес узла-запросчика (получателя для данной субакции);
  - длина данных;
  - данные (для Write и Lock);
  - метка транзакции;
  - код повтора;
  - скорость передачи.
- *Ответ* (response), используемый ответчиком для получения состояния или данных запроса, запускающий субакцию ответа. В ответе передается:
  - код типа транзакции;
  - адрес запросчика (получателя);
  - длина данных;
  - данные (для Read и Lock);
  - код ответа;
  - метка транзакции;
  - скорость передачи.
- *Подтверждение* (confirmation), уведомляющее запросчика о завершении транзакции. В нем прикладному драйверу передается:
  - состояние запроса: завершение (успешное), тайм-аут (отсутствие своевременного ответа), отсутствие квитанции о приеме запроса, исчерпание предела повторов, ошибка принятых данных;
  - код ответа (завершение или ошибка данных);
  - данные и их длина (для Read и Lock).

Организация потоковых передач и изохронный обмен

Потоковые передачи могут быть изохронными и асинхронными. В обоих случаях используются пакеты одного и того же формата, но есть разница в организации:

- *изохронные передачи* выполняются в специально выделенном периоде цикла, право на их передачу получается с коротким зазором арбитража. Для использования изохронных передач узел должен у диспетчера изохронных ресурсов получить *номер канала* и выделенную *полосу пропускания* шины (максимальное время передачи пакета в каждом цикле). Каждый активный передающий узел в каждом цикле шины должен посылать по одному изохронному пакету для каждого из своих каналов. Если для данного канала в очередном цикле нет передаваемых данных, должен посылаться изохронный пакет с нулевой длиной поля данных. Регулярность посылок пакетов является основой поддержания синхронизации передатчика и приемников;
- *асинхронные потоковые передачи* выполняются в оставшееся время цикла. Для них у диспетчера изохронных ресурсов запрашивается только номер канала. Полоса пропускания асинхронного потокового канала не нормируется.

Пакеты изохронных передач передаются широковещательно и физически адресуются по номеру канала (0–63), указанному в заголовке пакетов.

Получение номера канала и выделение полосы пропускания осуществляется обращениями узла к регистрам диспетчера изохронных ресурсов `CHANNEL_AVAILABLE` и `BANDWIDTH_AVAILABLE` (см. главу 21). Если канал и полосу получить не удалось, узел может периодически повторять запросы. Когда изохронный обмен становится ненужным узлу, он должен освободить свою полосу и номер канала, чтобы этими ресурсами смогли воспользоваться другие устройства. Обмен управляющей информацией с диспетчером производится асинхронными транзакциями.

Для выполнения изохронных обменов прикладной драйвер пользуется *сервисами LINK-уровня*:

- *Запрос изохронной передачи* (Link Isochronous Request), в котором драйвер передает:
  - номер канала;
  - длину данных;
  - данные;
  - скорость;
  - тег, определяющий формат данных;
  - код синхронизации (специфичен для приложений).
- *Индикация изохронной передачи* (Link Isochronous Indication), в которой драйвер получает те же данные. LINK-уровень будет давать индикацию только при приеме пакетов с требуемыми номерами каналов;

- *управление изохронным обменом* (Link Isochronous Control), в котором драйвер указывает номер канала, прием которого следует начать или завершить;
- *синхронизация циклов* (Link Cycle Synch) — индикация приема пакета начала цикла с сообщением текущего значения счетчика циклов и счетчика секунд. По этой информации и своим часам приложение может определить опоздание начала цикла и обеспечить свою синхронизацию с остальными участниками обмена.

Пакеты для потоковых передач

Потоковые передачи выполняются в широковещательной форме, в которой присутствует только пакет запроса и нет ни квитирования, ни ответов. Потоковые данные передаются *пакетами* с `tcode` = Ah. Этот код транзакции указывает на то, что пакет адресуется не по номеру узла, а по *номеру канала* `channel` (рис. 18.8, *а*). Формат данных в пакете определяется 2-битным полем `tag`: 00 — неформатированные данные, 01 — общий формат CIP (Common Isochronous Packet), 1x — резерв. Информацию о синхронизации, специфическую для приложений, может нести поле `sy` (4 бита). Длина блока данных `data_length` (в байтах) ограничивается в соответствии с табл. 18.4. Пакет может иметь и нулевую длину данных, при этом размер пакета сокращается до двух квадлетов заголовка. Если длина данных не кратна квадлету, то последний квадлет данных дополняется до целого байтами-заполнителями.

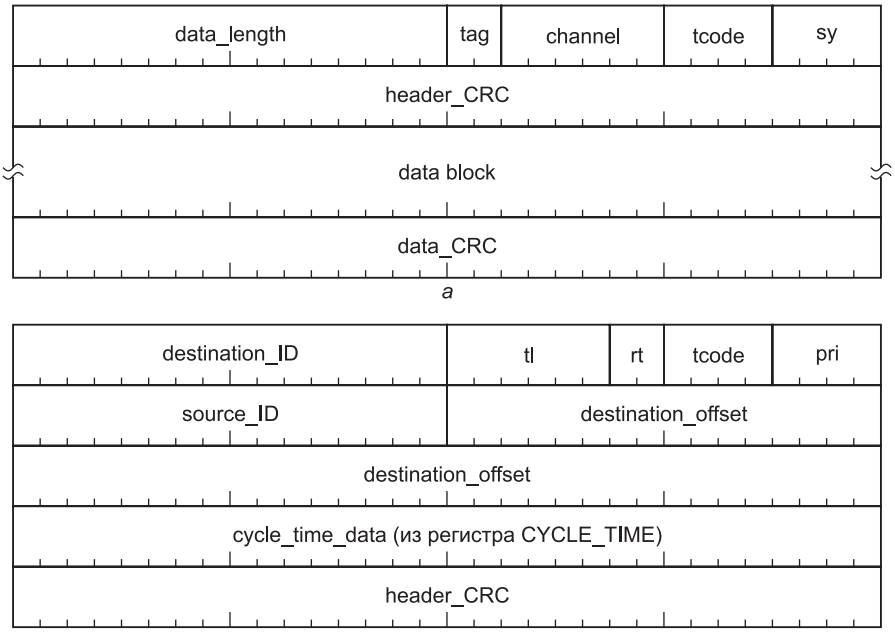


Рис. 18.8. Форматы пакетов для потоковых передач: *а* — пакет данных; *б* — пакет начала цикла

Мастер циклов регулярно (с периодом 125 мкс) передает *пакеты начала цикла* (рис. 18.8, *б*). В каждом таком пакете мастер циклов передает значение 32-битного счетчика времени (CYCLE_TIME), инкрементируемого с частотой 24,576 МГц. Метки времени нужны каждому узлу, поддерживающему изохронный обмен. Если к моменту подачи очередного пакета начала цикла шина оказывается занятой передачей, то пакет начала цикла задерживается до момента освобождения шины. В поле данных cycle_time_data передается значение регистра CYCLE_TIME мастера циклов на момент его фактической передачи, так что точная синхронизация узлов обеспечивается и при опозданиях пакета. Пакет передается во время асинхронной части цикла, но с коротким зазором арбитража, что обеспечивает приоритетность данных пакетов. Пакет передается широковещательно (destination_ID = = 3Fh), его принимают все узлы, интересующиеся изохронными передачами. В поле destination_offset указывается адрес регистра CYCLE_TIME (смещение 200h в пространстве регистров CSR). Поле tl не используется (ответов не предусматривается), rt = = 00 (без квитирования). Поле source_ID содержит номер шины и идентификатор корневого узла, который и должен быть мастером циклов.

Организация изохронных соединений

Для установления логических связей между источниками и приемниками изохронных потоков служат *регистры управления входными и выходными штекерами — PCR* (Plug Control Register), описанные ниже. *Штекер* (Plug) в 1394 — это метафора разъема, по которому передавался бы аудио- или видеосигнал в аналоговой системе. В 1394 штекеры, которыми соединяются между собой источники и приемники изохронных данных, отображаются регистрами PCR. Эти регистры используются для организации двухточечных и широковещательных соединений.

Для изохронных передач могут быть установлены соединения одного из трех основных типов:

- точка-точка (point-to-point) — соединение, явно видимое в PCR-регистрах участников: один из регистров OUTPUT_PLUG передающего узла и один из регистров INPUT_PLUG принимающего узла содержат одинаковый номер канала channel и у обоих ненулевые значения поля point_to_point;
- широковещательная передача (broadcast out), ведущаяся без каких-либо признаков «слушателей». В регистре OUTPUT_PLUG передающего узла устанавливается бит b — признак широковещания и указывается номер канала channel;
- широковещательный прием (broadcast in), ведущийся без индикации присутствия передатчика. В регистре INPUT_PLUG передающего узла устанавливается бит b — признак широковещания и указывается номер канала channel.

Для каждого штекера может быть установлено несколько соединений точка-точка (или ни одного); широковещательное соединение может быть только одно (или ни одного). Эти типы соединений могут независимо сосуществовать на одном штекере. Установление нового двухточечного соединения увеличивает значение поля point_to_point, разрыв соединения — уменьшает (в пределе до нуля).

Номер канала при установлении широковещательной передачи для штекера, не имеющего двухточечного соединения, определяется номером штекера и базовым адресом для данного узла (полем `broadcast_base` регистра `OUTPUT_MASTER_PLUG`). Если `broadcast_base` ≠ 63, то номер канала будет суммой (по модулю 64) базового адреса и номера штекера, иначе устанавливается номер канала 63.

Приемник и передатчик изохронных данных могут соединяться двояко, различия касаются прав на разрыв соединения:

- двухточечным соединением, установленным в каждом из них. Это соединение может разорвать только тот, кто его устанавливал;
- установлением широковещательной передачи в одном узле и широковещательного приема в другом. Это соединение может разорвать любой из участников.

Регистры управления штекерами (PCR)

Регистры управления штекерами (PCR) располагаются в адресном пространстве узлов, причастных к изохронным передачам. Они занимают место в начале пространства, отведенного под блоки узла. Регистры PCR допускают только операции чтения и блокированные операции (чтение с условной модификацией). Операции записи в эти регистры отвергаются. В состав регистров PCR входят следующие:

- *общий регистр выходных штекеров* `OUTPUT_MASTER_PLUG` (0900h, рис. 18.9, *а*) определяет общие характеристики узла как изохронного передатчика. Поле `output_plugs` определяет количество выходных штекеров (0–31). Поле `spd` задает максимальную скорость передачи по любому из штекеров (0 — S100, 1 — S200, 2 — S400, 3 — S800). Поле `broadcast_base` задает базовый номер широковещательного канала (см. главу 26). Поля `persistent_ext`, `nonpersistent_ext` и `r` зарезервированы. Этот регистр должен быть у узлов, поддерживающик как изохронную передачу, так и прием;
- *регистры выходных штекеров* `OUTPUT_PLUG` (0904–097Ch, рис. 18.9, *б*) присутствуют только у узлов, способных вести изохронную передачу. Присутствующие регистры (их число определяется регистра `OUTPUT_MASTER_PLUG`) располагаются по смежным адресам. Каждый из этих регистров описывает широковещательную или двухточечную передачу потока, исходящего от узла через соответствующий штекер. Бит `o` (online) указывает на возможность конфигурирования и использования данного штекера для передачи (`o`=0 — передача невозможна). Поле `channel` определяет номер канала, фигурирующий в качестве адреса исходящих изохронных пакетов. Бит `b` (broadcast) указывает на наличие широковещательного соединения для данного штекера. Поле `point_to_point` определяет число двухточечных соединений, установленных для данного штекера. Поле `spd` задает скорость передачи пакетов (0 — S100, 1 — S200, 2 — S400, 3 — S800). Поле `payload` задает максимальный размер поля данных изохронного пакета в квадлетах (0 соответствует 1024 квадлетам). Поле `overhead` позволяет более точно вычислить запрашиваемое значение полосы пропускания `bw`, за-

прашиваемое для данного канала (значение, вычитаемое из значения регистра BANDWIDTH_AVAILABLE):

bw = overhead × 32 + (payload+3)×2^{4-spd} (при overhead ≠ 0);

Если overhead = 0, то запрашиваемая полоса определяется по формуле

bw = 512 + (payload + 3) × 2^{4-spd}.

- *Общий регистр входных штекеров* INPUT_MASTER_PLUG (0980h, рис. 18.9, *в*) определяет количество входных штекеров (input_plugs в диапазоне 0–31) и максимальную поддерживаемую скорость (по любому из штекеров). Поля persistent_ext, nonpersistent_ext и r зарезервированы. Этот регистр должен быть у узлов, поддерживающих как изохронную передачу, так и прием;
- *регистры входных штекеров* INPUT_PLUG (0984–09FCh, рис. 18.9, *г*) присутствуют только у узлов, способных вести изохронный прием. Присутствующие регистры (их число определяется регистром INPUT_MASTER_PLUG) располагаются по смежным адресам. Каждый из этих регистров описывает широковещательное или двухточечное соединение для соответствующего штекера. Бит o (online) указывает на возможность конфигурирования и использования данного штекера для приема (o = 0 — прием невозможен). Поле channel определяет номер принимаемого канала. Бит b (broadcast) указывает на наличие широковещательного соединения для данного штекера. Поле point_to_point определяет число двухточечных соединений, установленных для данного штекера.

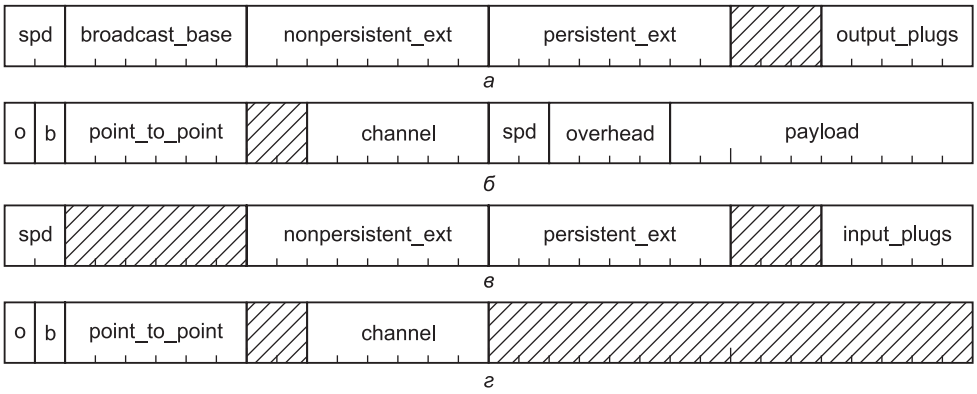


Рис. 18.9. Регистры управления штекерами: *а* — OUTPUT_MASTER_PLUG; *б* — OUTPUT_PLUG; *в* — INPUT_MASTER_PLUG; *г* — INPUT_PLUG

ГЛАВА 19

Арбитраж и распределение времени шины IEEE 1394

Арбитраж определяет, какому из узлов, запрашивающих передачу, предоставляется это право. Арбитраж обеспечивает гарантированную пропускную способность для изохронных передач и справедливое предоставление доступа узлам для асинхронных транзакций. Арбитраж на шине IEEE 1394 выполняется перед посылкой любого пакета запроса (синхронного или изохронного) или ответа. Исключением является соединенная (concatenated) форма выполнения транзакций (см. главу 18). Пакеты квитирования посылаются без арбитража — право на их передачу разыгрывать не надо, поскольку квитанцию посылает только тот единственный узел, к которому адресовался подтверждаемый пакет запроса или ответа.

Арбитражем занимается физический уровень каждого узла шины. Арбитраж выполняется распределенно иерархически: им занимаются все узлы, «верховным» арбитром является корневой узел (root node), автоматически выбираемый на этапе конфигурирования шины.

Физический уровень (PHY) предоставляет канальному уровню (LINK) следующие *сервисы арбитража*, перечисленные в порядке нарастания приоритетности:

- *справедливый арбитраж* (fair arbitration service), используемый для передачи обычных асинхронных пакетов;
- *приоритетный арбитраж* (priority arbitration service), используемый для передачи пакетов начала цикла и приоритетных асинхронных пакетов;
- *немедленный арбитраж* (immediate arbitration service), используемый для передачи пакетов квитирования;
- *изохронный арбитраж* (isochronous arbitration service), используемый для передачи изохронных пакетов.

Приоритет в арбитраже на шине IEEE 1394 определяется длительностью *зазора арбитража* (arbitration gap) — временем, в течение которого узел наблюдает покой шины перед началом передачи запроса арбитража. Чем меньше этот зазор, тем больше шансов у узла получить право на передачу. Исходная схема арбитража 1394 усовершенствовалась дважды: в 1394a были введены механизмы ускоренного арбитража, а в 1394b с его дуплексными соединениями был введен новый механизм —

BOSS-арбитраж. Все усовершенствования направлены на снижение непродуктивных затрат времени.

Если на шине используются изохронные передачи, то все транзакции организуются в последовательность *циклов* — интервалов времени с номинальной длительностью 125 мкс. Начало каждого цикла отмечается широковещательным *пакетом начала цикла* (Cycle Start). Эти пакеты посылает узел, являющийся *мастером циклов*. Право на передачу этого пакета мастер получает через арбитраж, используя высокий приоритет. Организация циклов представлена на рис. 19.1, где изображена работа двух изохронных каналов (Ch#J и Ch#K) и передача асинхронных пакетов A и B. После пакета начала цикла каждый узел, которому выделены изохронные каналы, имеет право передать по одному пакету для каждого канала (до прихода следующего пакета начала цикла). Для изохронных передач используется короткий зазор арбитража, так что асинхронные транзакции, использующие более длинный зазор, в изохронную часть цикла вклиниться не могут. После того как иссякнут изохронные пакеты данного цикла, выполняются асинхронные передачи, у которых для арбитража используются более длинные зазоры. Когда наступает пора посылки следующего пакета начала цикла, мастер цикла, дождавшись освобождения шины, снова получает право доступа (пользуясь своим приоритетом, обусловленным его положением в корне дерева) и посылает следующий пакет начала цикла. Таким образом, длительность цикла может отклоняться от номинального значения 125 мкс. Отклонения длительности цикла от номинального не страшны, поскольку пакет начала цикла несет значение системного времени точно на момент фактической передачи этого пакета.

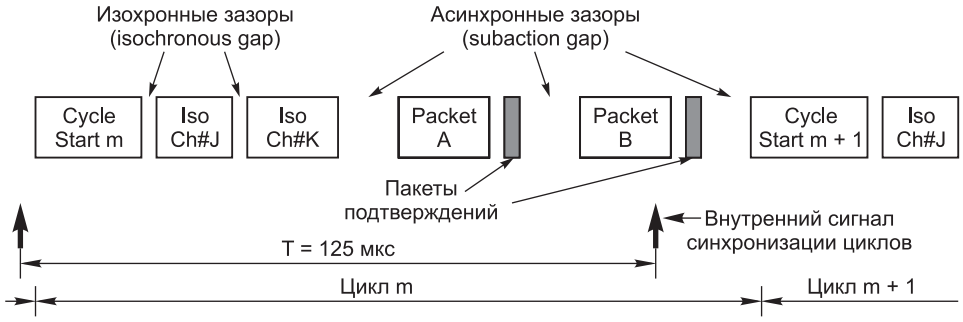


Рис. 19.1. Организация циклов

Если на шине не используются изохронные передачи, то мастер циклов может отсутствовать и пакетов начала цикла на шине не будет. В этом случае все время на шине может заполняться асинхронными передачами с их длинными зазорами арбитража.

Базовый механизм арбитража

Узел, желающий передать пакет, посылает вверх по дереву сигнал запроса арбитража *TX_REQUEST*. Узел имеет право на передачу пакета запроса или ответа, ког-

да он получит «сверху» сигнал предоставления *RX_GRANT*. Если несколько узлов запрашивают шину одновременно, то доступ получит то устройство, которое пошлет сигнал арбитража раньше. Запрос арбитража узел имеет право послать, лишь наблюдая состояние покоя шины (*Bus Idle*) в течение некоторого промежутка времени — *зазора арбитража*. Таким образом, более приоритетным окажется узел, использующий для задержки запроса арбитража меньшее значение зазора и находящийся ближе к корню — его запрос дойдет быстрее. В связи с этим мастер циклов должен находится в корне дерева, чтобы иметь приоритет посылки пакетов начала цикла по отношению к асинхронным пакетам других узлов. Для шины IEEE 1394 различают две длительности зазора:

- *изохронный зазор* (Isochronous gap), длительность которого лежит в пределах 40–50 нс;
- *зазор асинхронных транзакций* (subaction gap) — более длительный (0,3–10,6 мкс), величина которого подбирается так, чтобы не слишком затягивать арбитраж и не конфликтовать с задержками посылок пакетов квитирования. Длительность зазора должна быть не меньше, чем время оборота по шине для самой удаленной друг от друга пары узлов.

Немедленный и изохронный арбитраж используют одинаковую (самую маленькую) длительность зазора — 40–50 нс. Однако это не вызывает проблем с приоритетом изохронных пакетов: немедленный арбитраж выполняется только для пакетов-квитанций, которыми узлы отвечают только на асинхронные пакеты запросов-ответов. Пакеты запросов-ответов требуют приоритетного или справедливого арбитража, так что они могут передаваться только после всех изохронных пакетов данного цикла. Следовательно, немедленный арбитраж никогда не происходит в изохронной части цикла.

Справедливый арбитраж для асинхронных передач предохраняет шину и ее узлы от возможных перегрузок. Для этого на шине устанавливается *интервал справедливости* (fairness interval) — период времени, в течение которого узел имеет право выполнить лишь одну субакцию — послать один асинхронный пакет запроса или ответа. Длительность интервала справедливости зависит от числа активных узлов (у которых включен уровень LINK) и от загруженности шины изохронным трафиком. Справедливость обеспечивает PHY, у которого имеется специальный *бит разрешения арбитража*. Физический уровень будет посылать запрос арбитража только при установленном бите разрешения. Этот бит изначально устанавливается после самоидентификации устройств. Далее, как только физический уровень (по запросу от канального) посылает запрос арбитража, бит сбрасывается. Повторно он установится только после того, как PHY будет видеть шину в покое в течение интервала `arb_reset_gap` — *зазора сброса арбитража* (по умолчанию 21 мкс). Этот интервал гораздо больше изохронного и асинхронного зазора; такое длительное «молчание» всех устройств означает, что у них больше нет асинхронных пакетов, разрешенных к передаче в этом интервале справедливости. Таким образом, интервал справедливости — это время между соседними зазорами сброса арбитража, и его длительность неопределенна (рис. 19.2).

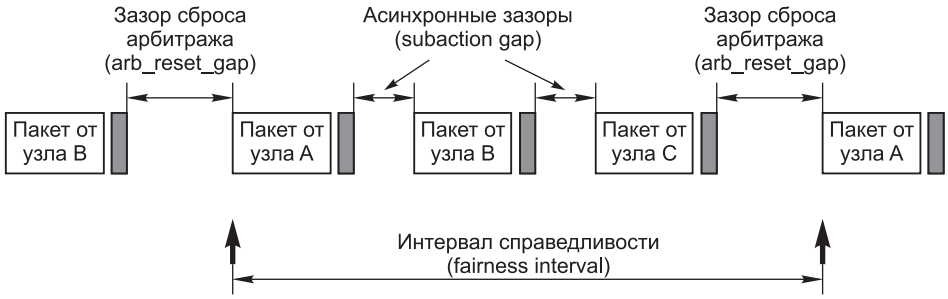


Рис. 19.2. Определение интервала справедливости

Усовершенствование арбитража в IEEE 1394a

Простота механизма арбитража оборачивается бесполезным расходованием времени на зазоры. Эти расходы снижают эффективную пропускную способность шины, особенно для коротких пакетов и пакетов, передаваемых на высокой скорости. Для скорости S100 на пакетах максимального размера эффективность использования шины составляет 80%, на коротких (64 байта данных) эффективность падает до 20%. При скорости S400 на длинных пакетах эффективность снижается до 50%, а на коротких — до 10%. В IEEE 1394a введен приоритетный арбитраж и ускоренные механизмы, сокращающие издержки времени на выжидание интервалов зазоров.

Приоритетный арбитраж (в IEEE 1394a) позволяет узлу за один интервал справедливости передать более одного асинхронного запроса. Число запросов, разрешенных узлу за один интервал, определяется полем `pri_req` его регистра `FAIRNESS_BUDGET` (смещение 218h в пространстве регистров CSR, другое имя того же регистра — `PRIORITY_BUDGET`, формат приведен на рис. 19.3). Значение этого поля заносит диспетчер шины исходя из числа узлов и их потребностей, которые они сообщают в поле `pri_max` своих регистров. Диспетчер назначает узлам значения `pri_req`, так чтобы их сумма не превышала $63–N$, где N — число узлов на шине.

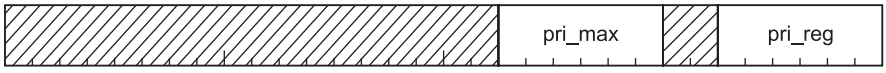


Рис. 19.3. Формат регистра FAIRNESS_BUDGET

Механизмы ускорения позволяют посылать пакеты запросов и ответов «с оказией», без задержек арбитража. *Ускорения* позволяют узлам, находящимся дальше от корня, обогнать в арбитраже более близкие к корню узлы, находящиеся в более выгодном положении. Однако и при использовании механизмов ускорения должен соблюдаться интервал справедливости. Ускорения используются узлами с «умным» PHY, способным анализировать пакеты. Ускорения могут быть реализованы двумя способами:

- используя *ускоренный арбитраж*, основанный на пакетах квитирования (ack-accelerated). Здесь возможны два варианта:
  - узел, отвечающий пакетом квитирования на какой-либо запрос (ответ), может послать свой асинхронный пакет запроса (ответа) вслед за пакетом-квитанцией, отделяя его лишь префиксом данных. Таким образом производится соединение (конкатенация) двух пакетов (см. рис. 18.2, *б*);
  - узел, желающий послать пакет запроса или ответа, дожидается прохождения пакета-квитанции и с коротким зазором (40 нс, как у изохронного арбитража) посылает запрос арбитража. При этом он опередит обычный узел, который в этом случае будет выдерживать нормальный асинхронный зазор.
- *Арбитраж «на лету»* (Fly-by) позволяет узлу присоединять свой пакет запроса к пакету, транслируемому им в сторону корня и не требующему квитирования (то есть к изохронным пакетам и квитанциям). При этом нельзя присоединять пакет, передаваемый на скорости S100, к пакетам, идущим на более высокой скорости (S100 передается без сигнализации о скорости, так что устройства 1394 будут ошибочно пытаться принять этот пакет на скорости пакета-«носителя»).

Во время исполнения ускоренного арбитража корневой узел не имеет возможности захватить шину. Если он работает мастером циклов (необходимым для изохронного обмена), то он может не суметь вовремя передать пакет начала цикла. Чтобы этого не происходило, узлы с ускоренным арбитражем должны иметь регистры `CYCLE_TIME` и запрещать ускоренный арбитраж близко к ожидаемой границе цикла, пока не получат очередной пакет начала цикла.

Новый механизм арбитража в IEEE 1394b

В бета-режиме IEEE 1394b введен новый метод арбитража — BOSS (Bus Owner/Supervisor/Selector), предназначенный как для чистых B-шин, все узлы которых работают в бета-режиме, так и для гибридных шин, часть узлов которых работает с традиционной DS-сигнализацией. Метод BOSS позволяет передавать запросы на право передачи во время передачи пакетов, что обеспечивает конвейеризацию работы шины и сокращает ее простои.

BOSS-арбитраж в чистой B-шине

Метод BOSS использует возможность полнодуплексного обмена каждой пары соединенных узлов, которая имеется только в бета-режиме. В этом режиме запросы на управление шиной посылаются по передающей линии; запросы представляют собой специальные 10-битные символы, отличимые от символов кодов данных. Узел может посылать запросы в любые свои порты, не занятые передачей данных. По логике работы повторителей многопортовых узлов оказывается, что к узлу, ведущему передачу данных, сигнальные пути от всех других узлов шины свободны. Исходя из этого узлу, ведущему передачу последнего пакета субакции, логично стать «*боссом*» (BOSS) — принимать решение о том, какому из узлов предоста-

вить право следующей передачи. Роль «босса» передается между PHY узлов шины по следующим правилам:

- данный PHY станет «боссом», если он является источником пакета. По этому признаку боссом станет узел, посылающий пакет квитирования или пакет ответа физического уровня;
- данный PHY станет «боссом», когда он явно или неявно получает право передачи. Явное получение права — прием символа *GRANT* или *GRANT_ISOCH* в качестве признака завершения пакета или в ответ на запрос арбитража. Неявное получение права происходит, когда принимающий PHY может самостоятельно определить, что субакция завершена (шина находится в изохронном интервале или последний пакет был пакетом квитирования в асинхронном интервале);
- корневой узел берет на себя роль «босса» после длительного отсутствия активности на шине (из-за потери пакета квитирования или иной ошибки);
- PHY перестает быть «боссом» по приему пакета, а также когда он явно или неявно предоставляет право передачи другому узлу.

Вместо интервалов покоя, используемых в традиционном механизме арбитража для обозначения завершения фазы самоидентификации при конфигурировании, завершения изохронного интервала, завершения субакции и границы интервала справедливости, в бета-режиме используются маркеры. *Маркер* (token) — это специальный управляющий символ, передаваемый «боссом» в течение времени, достаточного для его обнаружения по всей шине. Маркер может передаваться, когда шина находится в покое или когда она удерживается в занятом состоянии потоком символов *DATA_NULL* (маркеры встраиваются в этот поток).

Как и в традиционном режиме, передача пакетов ведется в двух интервалах — изохронном и асинхронном. Изохронный интервал начинается, когда мастер циклов передает маркер *CYCLE_START_EVEN/ODD* и пакет начала цикла. Заканчивается изохронный интервал, когда текущий «босс» не имеет относящегося к текущей фазе изохронного запроса от своего LINK-уровня или от какого-либо активного порта, работающего в бета-режиме. В этом случае «босс» передает маркер *ASYNC_EVEN* или *ASYNC_ODD*, означающий начало асинхронного интервала. Переход шины в состояние нормальной работы после самоидентификации происходит по маркеру *ASYNC_EVEN*, который передается корневым узлом после передачи его пакетов самоидентификации.

Асинхронный период (время, не занятое под изохронные интервалы), делится на интервалы справедливости. Граница интервала справедливости отмечается маркером *ASYNC_EVEN* (начало четного интервала справедливости) или *ASYNC_ODD* (начало нечетного интервала справедливости), отличающимся от предыдущего асинхронного маркера. Четные и нечетные интервалы чередуются. Возможно, что маркер *ASYNC_EVEN/ODD* одновременно будет означать и завершение изохронного интервала, и начало нового интервала справедливости. Для повышения устойчивости шины к потере маркера корневой узел, обнаружив покой шины, непрерывно посылает маркеры *ASYNC_EVEN/ODD*.

Передача запросов в виде символов позволяет осуществлять их приоритезацию и конвейеризацию, введя множество типов запросов и фаз шины. Изохронные и асинхронные интервалы делятся на свои четные (even) и нечетные (odd) фазы, чередующиеся независимо друг от друга. Изохронные фазы меняются по концу каждого изохронного интервала цикла (по маркеру *ASYNC_EVEN/ODD*). Асинхронные фазы меняются на границе каждого интервала справедливости, в соответствии с маркером *ASYNC_EVEN/ODD*. После сброса изохронные и асинхронные фазы считаются четными.

Запросы арбитража могут предназначаться как для текущей фазы, так и для последующей (противоположной), что является конвейеризацией запросов. PHY бета-узла посылает «боссу» изохронные и асинхронные запросы с наивысшим приоритетом, полученные от своего LINK-уровня или от других бета-портов. Приоритетность зависит от текущего интервала шины и фазы.

Вслед за передачей последнего пакета субакции текущий «босс» передает право на передачу для запроса, наиболее приоритетного для данного интервала и фазы. Если к этому времени «босс» не принял запросов, относящихся к данной фазе, он передачей одного или более маркеров меняет интервал (с изохронного на асинхронный) и/или фазу асинхронного интервала:

- если шина находится в изохронном интервале и нет изохронного запроса к текущей фазе, то «босс» завершает изохронный интервал маркером *ASYNC_EVEN/ODD*, не меняя текущей асинхронной фазы;
- если шина находится в асинхронном интервале и нет запроса к текущей фазе, то «босс» маркером *ASYNC_EVEN/ODD* меняет текущую асинхронную фазу (начинается новый интервал справедливости);
- если и для новой асинхронной фазы нет запросов, то «босс» передает управление шиной узлу, стоящему выше него по иерархии. В конце концов, управление (роль «босса») получит корневой узел. Асинхронная фаза не будет сменяться другой до тех пор, пока в текущей фазе не будет передан хотя бы один пакет.

В асинхронном интервале узел может послать запрос для текущей фазы, только если он имеет бюджет приоритета для текущего интервала справедливости. Если бюджет исчерпан, то узел может послать запрос для следующей фазы, на который право передачи будет дано в следующем интервале справедливости. Таким образом, сменой фаз осуществляется справедливый арбитраж без потерь времени на зазоры. Асинхронные фазы меняются только тогда, когда все узлы, имеющие пакеты для передачи в данной фазе, выполнят эти передачи.

Арбитраж в гибридной шине

В гибридной шине, состоящей из бета-узлов и традиционных узлов с DS-сигнализацией, новый механизм арбитража обеспечивает совместимость устройств, правда, с применением зазоров. Гибридная шина состоит из B-облаков (одного или нескольких), между которыми находятся фрагменты с DS-сигнализацией. При этом корневой узел может оказаться как в B-облаке, так и в DS-фрагменте.

В каждом B-облаке имеется один *старший пограничный узел* (senior border PHY): это последний узел, передающий в облако пакет самоидентификации без кода скорости. Старший пограничный узел облака, содержащего корневой узел, называется *прокси-корнем*. Этот узел будет работать дежурным арбитром, пока другие узлы не играют роль «босса». Если корнем является традиционный узел, то прокси-корня нет. P-порт старшего пограничного узла (если он не является прокси-корнем) всегда является DS-портом.

Старший пограничный узел в каждом B-облаке наблюдает за традиционными зазорами арбитража, и только он имеет право посылать асинхронные маркеры. Обнаружив зазор асинхронных транзакций, он посылает асинхронные маркеры без смены фазы. Маркер со сменой фазы он посылает, обнаружив зазор сброса арбитража. В состоянии покоя шины он посылает маркеры текущей фазы.

В ответ на запрос от B-порта старший пограничный узел начинает процедуру арбитража, используя традиционный механизм зазоров. Если старший пограничный узел является прокси-корнем, то он предоставляет право передачи B-узлу. Если старший пограничный узел не является прокси-корнем, то он посылает запрос арбитража через свой p-порт. Выиграв традиционный арбитраж, он выполняет B-арбитраж, и начинается передача бета-трафика. Как только в B-облаке начинается бета-передача, в нем включается механизм BOSS-арбитража, и этот механизм будет работать до тех пор, пока не завершится субакция и в облаке не закончатся запросы для текущей фазы.

В период передачи трафика B-облака старший пограничный узел должен обеспечить видимость занятости шины для всех узлов остальной части шины. Пакеты, передаваемые в традиционном формате, пограничный узел транслирует нормальным образом (заменяя пакет префиксом данных, если пакет передается на недопустимо высокой скорости). Пакеты бета-формата пограничный узел транслировать из облака не может, вместо них он передает префикс данных до тех пор, пока к нему не придет пакет традиционного формата. Чтобы эта передача префикса закончилась, «босс» по окончании субакции и при отсутствии запросов текущей фазы должен послать пакет традиционного формата с нулевой длиной данных.

В конце передачи бета-трафика текущий «босс» не посылает никаких маркеров, а передает право управления вышестоящему узлу. Таким образом это право достигнет старшего пограничника, который начнет формирование интервала покоя шины.

В гибридной шине маркер *CYCLE_START* может быть только в облаке, в котором находится мастер цикла. Это облако называется старшим B-облаком (senior B cloud). Такового может и не быть, поскольку мастер циклов может оказаться и в традиционном фрагменте шины. Остальные B-облака называются младшими (junior B cloud). Поскольку маркеры по традиционному фрагменту распространяться не могут, воспроизведение маркеров начала цикла в младших облаках проблематично. PHY узлов младшего облака узнают о начале изохронного периода от своего LINK-уровня. Поскольку информации о номере фазы у узлов младшего

облака нет, изохронная транзакция запрашивается сигналом (символом) *ISOCH_CURRENT*. Этот запрос достигает старшего пограничного узла, который выполняет немедленный арбитраж в традиционном фрагменте шины. Запросы *ISOCH_EVEN/ODD* в младшем облаке не фигурируют.

Пограничный узел принимает традиционные запросы арбитража (сигналом *TX_REQUEST*) и преобразует их в символ запроса *LEGACY_REQUEST*, посылаемый в свой p-порт. Остальные свои порты он блокирует посылкой в них символа *DATA_NULL*. Если арбитраж выигран, то узел получит символ *GRANT* или *GRANT_ISOCH*; если проигран — *DATA_PREFIX* или *DATA_NULL*.

ГЛАВА 20

Конфигурирование шины и узлов IEEE 1394

Конфигурирование шины IEEE 1394 выполняется в различных ситуациях:

- автоматически при изменении конфигурации — при подсоединении и отсоединении устройств, а также включении/выключении их PHY-уровня;
- при обнаружении каким-либо узлом фатальной ошибки — «зависания» шины;
- по инициативе какого-либо узла, желающего, например, изменить топологию (сменить корневой узел).

Конфигурирование состоит из трех последовательных этапов.

1. *Сброс* (Bus Reset), с момента которого прекращается передача полезного трафика.
2. *Идентификация дерева* (Tree Identification), во время которой узлы выстраиваются в иерархическую структуру.
3. *Самоидентификация узлов* (Self Identification), во время которой узлы присваивают себе уникальные физические идентификаторы.

Конфигурирование шины приводит ее в состояние, пригодное для передачи полезного трафика. Конфигурирование шины осуществляется исключительно аппаратными средствами PHY-уровня каждого узла (LINK-уровень конфигурируемых узлов может быть и отключен). Программные средства в этом процессе не участвуют. Благодаря чисто аппаратной реализации автоконфигурирование производится настолько быстро, что возможно сохранение непрерывности изохронных потоков.

В первоначальной версии шины самое большое время во всей процедуре конфигурирования занимал сброс. В физический уровень 1394 вносились усовершенствования, направленные на минимизацию потерь времени при сбросе. Остальные этапы конфигурирования происходят быстрее, но в случае образования петлевого соединения идентификация дерева никогда не закончится. Эта ситуация обнаруживается любым узлом, и сообщение о ней доводится до пользователя. В 1394b приняты меры по автоматическому исключению петлевых соединений.

Свойства любого узла сконфигурированной шины наблюдаемы и управляемы через его архитектурные регистры и память конфигурации. Они доступны со стороны шины через асинхронные транзакции к определенным адресам. Архитектурные регистры определяют поведение узла на шине. Память конфигурации раскрывает «прикладную ценность» узла и обеспечивает его уникальную идентификацию, не зависящую от непостоянного физического идентификатора.

Сброс шины (Bus Reset)

Сброс шины приводит к ряду эффектов:

- все узлы переводятся в исходное состояние;
- передача трафика прерывается, незавершенные транзакции отбрасываются;
- вся информация о топологии разрушается;
- все узлы переходят в состояние инициализации.

Сброс шины может быть вызван любым устройством. Сброс распространяется всеми узлами беспрепятственно, поскольку его сигнал по праву доминирования единиц (см. главу 22) превалирует над всеми другими сигналами. После сигнала сброса, длительность которого Treset = 167 мкс (или 1,4 мкс — короткий сброс, см. ниже), узлы выдерживают паузу Twait = 0,16 мкс, после чего начинается процесс идентификации дерева.

Сигнал сброса вырабатывается по ряду причин:

- сброс при включении питания PHY (состояние питания LINK не важно);
- сброс при подключении узла к порту и его отключении (обнаруживается узлом по изменению напряжения смещения);
- сброс по тайм-ауту арбитража, предотвращающий «зависание» шины. Он вырабатывается узлом при длительном нахождении шины в одном состоянии по истечении `MAX_ARB_STATE_TIME` = 200–400 мкс. Контроль длительности не распространяется на состояния покоя (bus idle), старта идентификации дерева и состояния, контролируемые своими счетчиками тайм-аутов;
- программно-инициируемый сброс, выполняемый по запросу приложения на каком-либо узле.

В случае изменения топологии сброс вырабатывается по крайней мере два раза подряд. Первый раз его сигнализирует узел-новичок, подключаемый к шине. Второй раз сброс подаст узел, к которому он подключился, поскольку он обнаружит изменение состояния порта. Два сброса подряд — это сигнал к установке стандартного зазора арбитража (`gap_cnt`=63); новое оптимальное значение зазора должно определяться по новой топологии.

При подключении/отключении устройств в разъемах возникает дребезг контактов — серия замыканий и размыканий, которая может длиться десятки и сотни миллисекунд. В 1394 это порождает так называемый *шторм сбросов*, что парализует работу шины на длительное время (сотни миллисекунд). В 1394a эту проблему

решили введением защиты от дребезга: узел посылает сброс по обнаружении смены состояний не сразу, а через 330–350 мс (когда дребезг закончится).

Соединяемые узлы воспринимают факт смены состояния не одновременно — разница в моментах определения изменения может составлять десятки миллисекунд (они проверяют замыкание разных цепей). В 1394a приняты меры, чтобы устройство, определившее факт смены позднее, не давало лишнего сброса. Порт, который во время выжидания интервала защиты от дребезга обнаружил сигнал сброса (от более быстрого устройства), должен прекратить выжидание и начать трансляцию сигнала сброса.

Из-за особенностей интерфейса PHY–LINK сигнал сброса должен быть весьма длительным, чтобы уведомление о нем получили LINK-уровни всех узлов. Сигнал сброса приходит в любой момент времени. В узле, ведущем передачу пакета, интерфейсом PHY–LINK владеет LINK. Таким образом, PHY не имеет возможности сигнализировать LINK'у о сбросе, пока передача пакета не будет завершена. Из-за этого длительность сигнала сброса должна быть больше длительности максимального пакета, поэтому принято Treset = 166,7 мкс. Поскольку во время сброса трафик по шине передаваться не может, такой длинный сброс приводит к потерям изохронных данных (может быть потерян целый 125-микросекундный цикл шины). В 1394a эту проблему решили, введя новый вариант сброса — *короткий сброс* (Short Bus Reset) длительностью 1,3–1,4 мкс. Короткий сброс узел может подать, только выиграв арбитраж (Arbitrated Bus Reset), — это гарантия того, что ни один узел в это время не передает пакет. Однако применение короткого сброса имеет свою специфику:

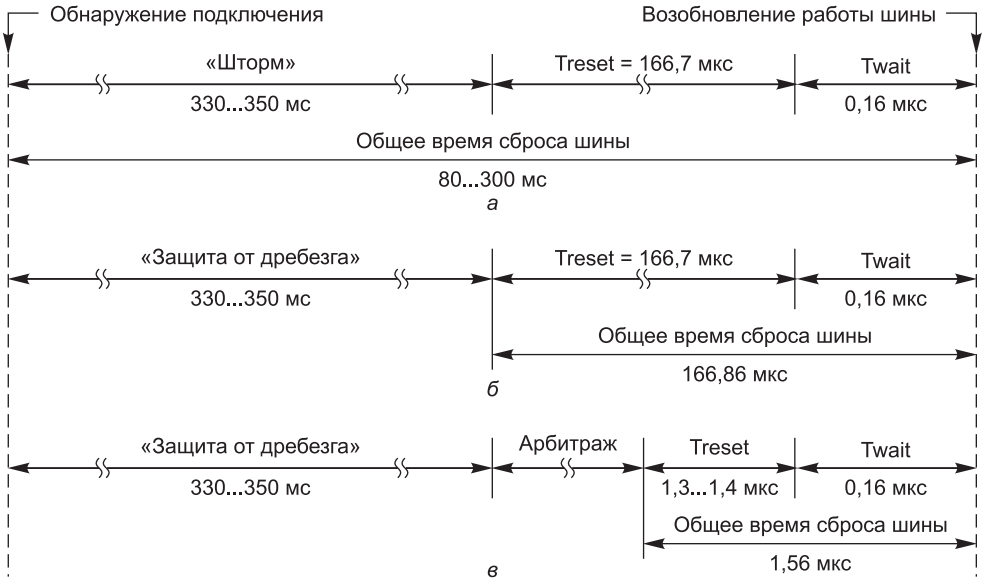


Рис. 20.1. Варианты сброса по подключению узла: *а* — сброс со штормом (1394); *б* — сброс с защитой от дребезга (1394a); *в* — короткий сброс (1394a)

- если обнаруживается отключение на р-порте, то арбитраж попросту невозможен (корень уже недоступен);
- если при попытке арбитража срабатывает тайм-аут, сброс должен быть длинным;
- одиночный узел при подключении должен задержать подачу сброса на 1 с, чтобы дать возможность другому узлу подать короткий сброс (по обнаружении подключения). Если сброс не приходит, то узел генерирует обычный сброс, если приходит — генерировать дополнительный сброс уже не надо.

Три различных варианта сброса шины по факту подключения узла приведены на рис. 20.1. Из диаграммы видно, как усовершенствование механизма сброса позволяет сократить время простоя шины с сотен миллисекунд (1394) до 167 мкс (1394a) и даже до 1,56 мкс (короткий сброс в 1394a).

Идентификация дерева

Во время идентификации дерева узлы выстраиваются в иерархическую структуру и выбирается корень шины. Для узлов идентификация дерева сводится к определению статуса активных портов: присвоение им типов: *с-порт* (к которому подключен дочерний узел) или *р-порт* (которым он подключается к родительскому). Этап идентификации дерева начинается после сброса. Для идентификации используется сигнализация арбитража (см. табл. 22.4 на стр. 412).

В начале идентификации дочерние узлы ищут своих родителей, посылая им сигнал уведомления *Parent_Notify*, а родители признают свои дочерние узлы сигналом *Child_Notify*. Сигнал *Parent_Notify* безусловно посылают узлы-листья (имеющие всего по одному порту). Узлы-ветви могут послать сигнал *Parent_Notify* только на один из своих портов, при условии, что на все остальные активные порты пришли аналогичные сигналы. При этом порты, на которые пришел сигнал *Parent_Notify*, помечаются как с-порты. На эти порты посылается уведомление *Child_Notify*, но не раньше, чем когда сигнал *Parent_Notify* будет обнаружен на всех активных портах, кроме одного. Порт, на который пришел сигнал *Child_Notify*, помечается как р-порт; он перестает посылать сигнал *Parent_Notify*, что является подтверждением приема сигнала *Child_Notify*.

Узел, у которого на все порты приходят сигналы *Parent_Notify*, становится корнем. Все его порты становятся с-портами. Он посылает на эти порты сигнал *Child_Notify*, а получив подтверждение (снятые сигналы *Parent_Notify*), и сам прекращает подавать сигналы *Child_Notify*. На этом идентификация дерева заканчивается, и шина переходит к этапу самоидентификации узлов.

Возможна ситуация, когда два соединенных узла пытаются друг в друге найти своих родителей, посылая сигнал *Parent_Notify*. При этом столкновение данных сигналов приводит к приему каждым портом состояния *Root_Contention* — признака состязания за роль корня. В этом случае оба узла перестают подавать сигнал на эти порты и через случайный интервал проверяют их состояние. Узел, обнаруживший состояние покоя на данном порте, посылает сигнал *Parent_Notify*. Узел, обнаруживший этот сигнал, посылает сигнал *Child_Notify* и становится корнем. В начале

состязания каждый узел случайным образом устанавливает для себя время задержки из двух вариантов:

- ROOT_CONTEND_FAST (0,24–0,26 мкс) — быстрый участник;
- ROOT_CONTEND_SLOW (0,57–0,6 мкс) — медленный участник.

Тот узел, который окажется быстрее, корнем не станет (он раньше начнет искать родителя). Если оба участника выберут одинаковое время, то они опять столкнутся и повторят состязание с новыми значениями задержки.

На том, что в состязании за роль корня побеждает самый медленный участник, построен механизм принудительного назначения корня. У каждого PHY есть бит RHB, единица в котором заставляет этот узел задерживать подачу сигнала поиска родителей на 83–167 мкс от начала идентификации дерева. Специальный пакет физического конфигурирования позволяет установить этот бит у заданного узла и сбросить у всех остальных, что обеспечит победу данному узлу в последующих выборах корня по ближайшему сигналу сброса. Если вдруг окажется, что бит RHB установлен у нескольких узлов, то они разыграют право стать корнем по вышеописанному правилу случайных состязаний.

На рис. 20.2 приведены пошаговые сценарии роста дерева, на каждом шаге определяется очередной уровень иерархии узлов. Стрелками на рисунке обозначены сигналы *Parent_Notify*, определяющие роли портов и узлов. На рисунке изображены три различных пошаговых сценария вырастания дерева:

- бесконфликтный (рис. 20.2, *а*);
- с состязанием между узлами B и C (рис. 20.2, *б*);
- с принудительным назначением роли корня для узла E (рис. 20.2, *в*). В этом примере показано, как благодаря выдержке узла E из той же топологии соединений, что была и на рис. 20.2, *а*, вырастает иное дерево.

В случае образования петли процесс идентификации дерева «зависает». Эта ситуация выявляется по срабатывании тайм-аута арбитража (167 мкс); она сигнализируется через LINK приложению и пользователю, который должен принять меры к разрыву петли (физическим переключением кабелей). В 1394b приняты меры по автоматической борьбе с петлями путем запрещения работы какого-либо порта по результатам проверки (см. главу 22).

Самоидентификация узлов

На этапе самоидентификации узлы назначают себе уникальные идентификаторы, согласуют скоростные возможности и (не обязательно) выбирают диспетчера изохронных ресурсов. Самоидентификация выполняется под управлением корневого узла, организующего последовательное предоставление прав на самоидентификацию всем узлам сети. Получив это право, узел посылает пакет (или серию пакетов) самоидентификации, которые распространяются по всей шине. Закончив посылку пакетов, узел обменивается со своим родителем информацией о доступной скорости. В результате этого обмена пара соединенных портов становится скон-

фигурированной. *Сконфигурированный порт* — это активный порт, для которого установлен его тип (с-порт или р-порт) и согласована скорость.

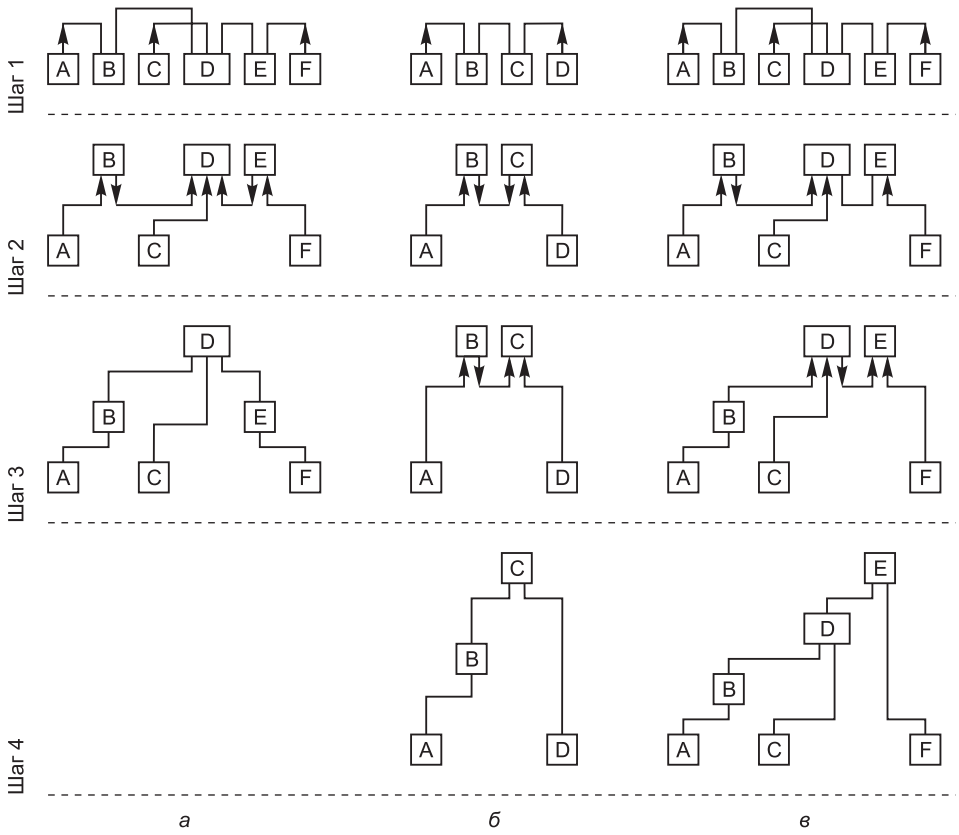
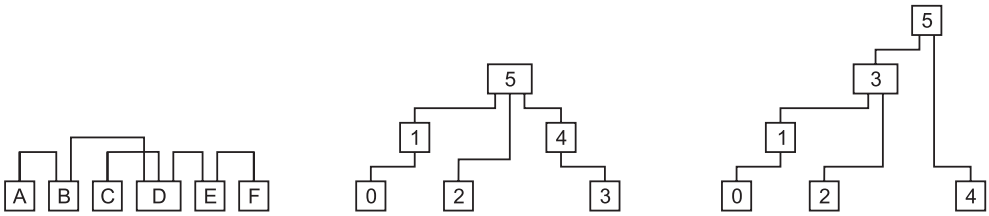


Рис. 20.2. Варианты роста дерева: *а* — бесконфликтный; *б* — с состязанием за роль корня; *в* — с принудительным назначением корня

Промежуточные узлы-ветки и корневой узел организуют передачу права самоидентификации всем узлам по порядку. Порядок, в котором узлы выполняют самоидентификацию, определяется нумерацией портов в промежуточных узлах (ветках и корне). Первым будет самоидентифицироваться конечный узел (лист), который подключен к порту корня с наименьшим номером и путь к которому лежит в узлах-ветках через порты с наименьшими номерами. Этот узел получит `PHY_ID`=0. Последним идентифицируется корень — он получает наибольший идентификатор. На рис. 20.3 приведены два варианта назначения идентификаторов, соответствующие деревьям, взятым из рисунков 20.2, *а* и *в*.

Право на самоидентификацию получает узел, которому по р-порту приходит сигнал *Self_ID_Grant*, а все его с-порты (если таковые есть) уже сконфигурированы. В ответ на предназначенный ему *Self_ID_Grant* узел отвечает передачей (вверх)



а *б* *в*

Рис. 20.3. Назначение идентификаторов в шине: *а* — исходная топология; *б*, *в* — полученные номера PHY_ID

префикса данных. Когда в ответ на это узел увидит снятие сигнала *Self_ID_Grant*, он посылает пакет(ы) самоидентификации. *Пакет самоидентификации* (см. рис. 23.12 на с. 448) всегда передается на скорости S100. Пакет содержит информацию, описывающую физические свойства узла:

- идентификатор (`PHY_ID`), который узел назначает себе сам равным числу услышанных им пакетов самоидентификации (пакетов-0). Самый первый идентифицирующийся узел возьмет `PHY_ID`=0, корень получает наибольший идентификатор;
- число портов и их состояние: неактивен, с-порт или р-порт. Эту информацию узел получает на предшествующем этапе — при конфигурировании дерева;
- зазор арбитража, взятый по умолчанию (при смене топологии) или сохраненный с прошлого сеанса работы (до сброса);
- доступная скорость работы (определяется свойствами узла, заложенными разработчиком);
- отношение к питанию от шины, определяется аналогично;
- состояние LINK-уровня (бит `L`, определяется по факту включения);
- претензии на роль диспетчера (бит `C`), определяемые с участием прикладного ПО.

При числе портов более трех узел посылает серию пакетов самоидентификации. Пакеты самоидентификации слышат все узлы, по ним они могут строить карту топологии и карту скоростей. Прослушивая пакеты самоидентификации, узлы определяют диспетчера изохронных ресурсов — им станет узел с максимальным номером `PHY_ID`, у которого в пакете самоидентификации установлены биты `C` и `L`.

Завершив передачу пакетов самоидентификации, узел посылает в р-порт сигнал *Ident_Done*. На это родительский узел отвечает префиксом данных и помечает свой соответствующий с-порт как сконфигурированный. При подаче сигнала *Ident_Done* узел с помощью смещения уровней сообщает доступную ему скорость работы; его родитель при передаче префикса данных аналогичным образом сообщает свою. Таким образом, на завершающем этапе самоидентификации узла его р-порт и соединенный с ним с-порт родительского узла согласуют и устанавливают максимально возможную скорость обмена друг с другом.

Промежуточный узел-ветка, получивший сверху сигнал *Self_ID_Grant*, транслирует его на свой несконфигурированный с-порт с наименьшим номером; на остальные порты он посылает префикс данных. Получив префикс данных с с-порта, промежуточный узел прекращает подачу сигнала *Self_ID_Grant* (вниз) и транслирует префикс данных вверх. Далее, если после прохождения пакета самоидентификации на с-порт приходит сигнал *Ident_Done*, узел ответит ему префиксом данных с указанием своей скорости и пометит данный порт как сконфигурированный. Когда промежуточный узел, у которого все с-порты сконфигурированы, снова получит сигнал *Self_ID_Grant*, он сам отправит пакет(ы) самоидентификации (во все активные порты), а затем и сигнал *Ident_Done*.

Корневой узел посылает сигнал *Self_ID_Grant* в порт с наименьшим номером из числа активных несконфигурированных. На остальные порты он посылает префикс данных. Как только с порта, на который корень посылает *Self_ID_Grant*, он получит префикс данных, подача сигнала *Self_ID_Grant* прекращается. Когда корень обнаружит покой шины (после прохождения пакетов самоидентификации от узла, которому досталось это право), он снова пошлет сигнал *Self_ID_Grant*, опять же на несконфигурированный порт с наименьшим номером. Если самоидентифицировался узел, подключенный к корню, то корень получит сигнал *Ident_Done* и пометит данный порт как сконфигурированный. Теперь корень пошлет *Self_ID_Grant* на следующий порт. Как только все порты перейдут в сконфигурированное состояние, корень пошлет свой пакет самоидентификации, на чем процесс самоидентификации и закончится — шина переходит в фазу нормального арбитража.

Архитектурные регистры и память конфигурации узла

Пространству регистров узла, в котором располагается и память конфигурации, отводится диапазон адресов FFFF F000 0000–FFFF FFFF FFFFh. В дальнейшем описании в скобках указаны относительные адреса; 48-битный адрес (внутри узла) получается сложением относительного адреса и FFFF F000 0000h. Для получения полного 64-битного адреса этот адрес следует расширить влево 16-битным числом, состоящим из номера шины и идентификатора узла.

Архитектурные регистры CSR

Шина IEEE 1394 основана на стандарте ISO/IEC13213, известном под названием «CSR-архитектура», и наследует его основные регистры и структуры данных в памяти конфигурации. Архитектурные регистры CSR занимают первые 512 байт в начальном адресном пространстве узла. Часть регистров CSR-архитектуры в IEEE 1394 не используется, их описание здесь опущено.

Регистр состояния и управления `STATE` (рис. 20.4) доступен для чтения по двум адресам — `State_Clear` (000h) и `State_Set` (004h). Некоторые биты доступны лишь для чтения (RO), другие биты допускают программный сброс (записью единиц

в биты State_Clear) и аналогичную установку (через регистр State_Set). Назначение полей регистра состояния и управления:

- unit_depended — биты, специфичные для устройства;
- bus_depended — биты, специфичные для шины:
  - gone — бит, устанавливаемый по любому сбросу (устройства или шины) или по команде. Запрещает устройству посылку запросов. Обнуляется программно (возможно, и удаленно) при инициализации, когда устройство становится способным отвечать на запросы;
  - abdicate — бит, предназначенный для смены диспетчера шины. Устанавливается у узла, который должен стать диспетчером шины после грядущего сброса. При этом он не будет ждать положенных 125 мс (как «честные» кандидаты) перед тем, как сделать попытку блокированной транзакции с регистром BUS_MANAGER_ID (см. главу 21);
  - linkoff — состояние питания LINK-уровня, питающегося от шины (1 — отключено). Может быть установлен программно, обнуляется (и включается питание) по приему пакета *Link_On*;
  - cmstr — признак мастера циклов. Может быть установлен только у корневого узла.
- lost — потеря работоспособного состояния (по сбросу питания или фатальной ошибке;
- dreq (disable request) — запрет генерации запросов транзакций;
- atn, off — резервные биты в 1394;
- elog — признак обновления информации в журнале ошибок;
- state (RO) — состояние узла: 00 — рабочее (running), инициализация завершена; 01 — идет инициализация; 10 — тестирование (по запуску через запись в регистр Test_Start); 11 — фатальная ошибка (узел неработоспособен).

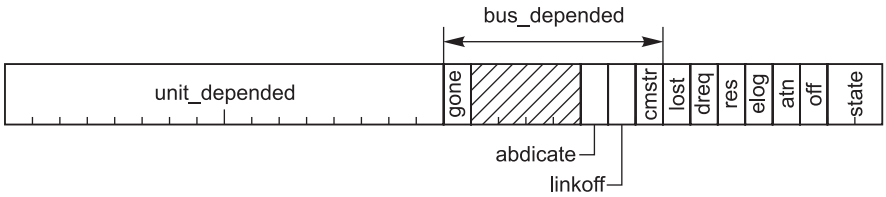


Рис. 20.4. Регистр STATE

Регистр идентификатора узла NODE_IDS (рис. 20.5) содержит поля:

- bus_id — идентификатор шины (по умолчанию 0), команда сброса на него не влияет; этот идентификатор записывается и считывается программно;
- PHY_ID — идентификатор узла, устанавливаемый при автоконфигурировании;
- Bus_dependent — поле, зарезервированное для нужд шины.

Рис. 20.5. Регистр NODE_IDS

Регистр команды сброса `Reset_Start` (00Ch) служит для подачи команды сброса (операцией записи), при этом значим только бит `nt` (no test, бит 31), единица в котором отменяет начальное тестирование узла при сбросе.

Регистры тайм-аута расщепленных транзакций позволяют задать предельное время ожидания ответа на запрос транзакции в пределах 8 с с дискретностью 125 мкс. Начальное значение — 100 мс, по сбросу не изменяется. В младших трех битах `SPLIT_TIMEOUT_HI` (018h) содержится число секунд, в старших 13 битах `SPLIT_TIMEOUT_LO` (01Ch) содержится число 125-микросекундных интервалов.

Регистры тестирования (необязательные) служат для запуска теста (`TEST_START`, 028h), передачи параметров теста (`ARGUMENT_HI`, 020h, `ARGUMENT_LO`, 022h) и сообщения состояния выполнения (`TEST_STATUS`, 02Ch).

Регистры прерываний (необязательные) служат для управления прерываниями, вызываемыми транзакциями записи в регистр `INTERRUPT_TARGET` (050h). Каждый бит этого регистра соответствует одному из условий (событий) прерывания; самый старший бит соответствует прерыванию с самым высоким приоритетом. Регистр `INTERRUPT_MASK` позволяет селективно выбирать условия, вызывающие сигнализацию прерывания приложению (процессору) данного узла (единичное значение бита маски разрешает прерывание по данному условию). Прерывания по шине могут посылаться широковещательно (адресуясь к регистру `INTERRUPT_TARGET` при `Dest_ID` = 63) или направленно, с указанием в целевом адресе идентификатора требуемого узла.

Регистры синхронизации (`Clock_Value`, `Clock_Tick_Period`, `Clock_Strobe_Arrived`, `Clock_Info`) в IEEE 1394, как правило, не используются — их функции выполняет регистр `CYCLE_TIME`).

Регистры сообщений (необязательные) `Message_Request`, `Message_Response` служат для передачи и приема широковещательных сообщений, адресованных всем узлам шины или всем блокам узла.

Специальные регистры последовательной шины

Специальные регистры последовательной шины (Serial Bus Dependent Registers) расположены во втором 512-байтном блоке, специально отведенном в CSR под шинно-зависимые регистры.

Регистры синхронизации изохронных передач `Cycle_Time` (200h) и `Bus_Time` (204h) задают время, измеряемое в секундах, циклах (125 мкс) и тиках (частота 24,576 МГц). Описание их полей см. в главе 21, запись в эти регистры влияет только на поле `second_count_hi`, автоматически инкрементируемое каждые 128 с.

Регистры мониторинга питания (необязательные) позволяют осуществлять предупреждение об угрозе пропадания питания. Узел, обнаруживший угрозу пропада-

ния питания, посылает широковещательное сообщение — выполняет запись в регистр POWER_FAIL_IMMINENT (208h, рис. 20.6, *а*). В этом сообщении единица в pfi_flag указывает на действительность остальных полей. Поле pfi_delay содержит ожидаемое время до пропадания (в сотнях микросекунд); поле pfi_source — идентификатор узла, пославшего сообщение. Регистр POWER_SOURCE (20Ch, рис. 20.6, *б*) в поле power_source содержит идентификатор узла, сообщения от которого принимаются во внимание.

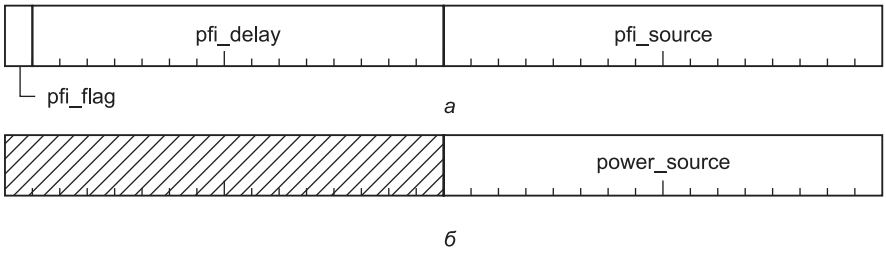


Рис. 20.6. Регистры мониторинга питания: *а* — POWER_FAIL_IMMINENT; *б* — POWER_SOURCE

Регистр тайм-аута повторов по занятости целевого узла BUSY_TIMEOUT (210h) определяет предел времени или количество попыток повторов (см. рис. 18.7). По сбросу устанавливается время ожидания 25 мс и 0 попыток повторов.

Регистр идентификатора диспетчера шины BUS_MANAGER_ID (21Ch) должен присутствовать у любого узла, претендующего на роль диспетчера изохронных ресурсов. В нем у выбранного диспетчера изохронных ресурсов будет значение идентификатора диспетчера шины (в младших 6 битах) или признак отсутствия диспетчера — 3Fh (значение, устанавливаемое по начальному сбросу). Этот регистр должен поддерживать блокированную транзакцию (условную запись) для выбора диспетчера шины (подробнее см. в главе 21).

Регистры доступной пропускной способности BANDWIDTH_AVAILABLE (220h) и *каналов* CHANNELS_AVAILABLE (224h и 228h) должны быть у диспетчера изохронных ресурсов. Эти регистры должны поддерживать блокированную транзакцию (условную запись) для выделения изохронных ресурсов (см. главу 26).

Служебные регистры MAINT_CONTROL (22Ch) и MAINT_UTILITY (230h) служат для отладочных целей. Запись в регистр MAINT_UTILITY не вызывает никаких побочных действий в узле, чтение возвращает результат последней записи. Для проверки связи и протоколов обработки ошибок можно обращаться к этому регистру любого узла, проверяя достоверность обмена (сравнивая данные чтения и записи) и вводя различные ошибки. Регистр MAINT_CONTROL (рис. 20.7) позволяет принудительно вводить ошибки передачи пакетов:

- e_hcrc — искажение CRC заголовка;
- e_dcrc — искажение CRC поля данных;
- no_pkt — запрет генерации следующего пакета;
- f_ack — подмена нормального ответа (квитанции) на поле ack;

- `no_ack` — запрет посылки пакета квитанции;
- `ack` — подставляемое значение кода ответа (если `f_ack` = 1).

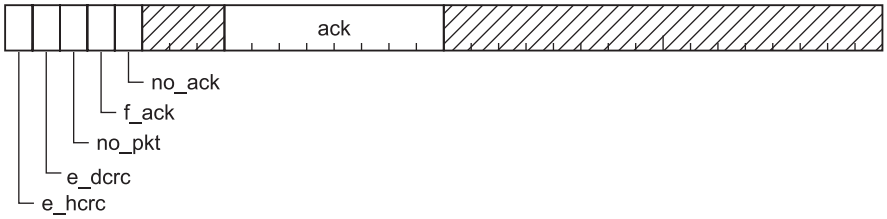


Рис. 20.7. Регистр Maint_Control

Память конфигурации

Память конфигурации для узлов шины 1394 соответствует спецификации ISO/IEC 13213 со специфическими полями для последовательной шины. Эта память в шинных транзакциях допускает только чтение, поэтому она называется ROM (Read-Only Memory). В данном случае применение термина *ПЗУ* (постоянное запоминающее устройство) как синонима *ROM* некорректно, поскольку память конфигурации узла может быть изменена его локальным ПО в процессе работы устройства. Каждое такое изменение отражается инкрементом поля `gen` (generation), входящим в блок параметров последовательной шины, находящийся в этой же памяти.

Память конфигурации может иметь минимальный размер 32 бита или нормальный размер 1 Кбайт. В CSR-архитектуре предусмотрена возможность расширение объема памяти конфигурации (при косвенной адресации содержимого), но в IEEE 1394 эта возможность не используется. Память конфигурации организована в виде иерархической системы каталогов (рис. 20.8). Форматы содержимого памяти приведены на рис. 20.9, где серым цветом выделены обязательные элементы.

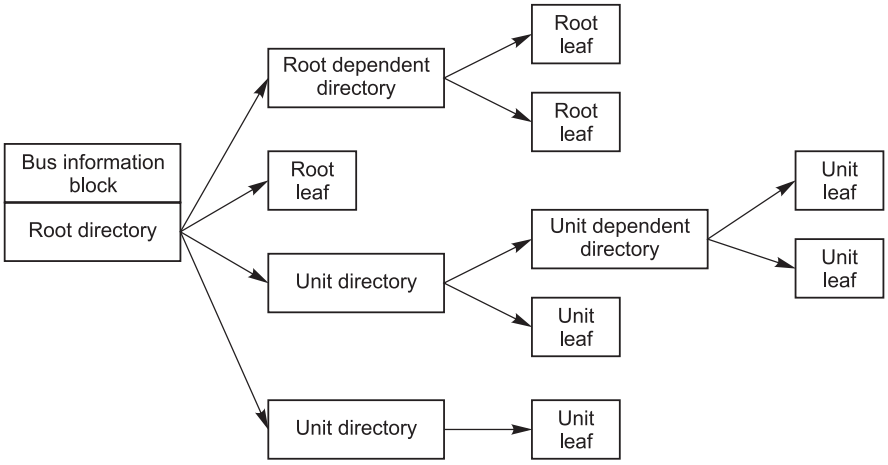


Рис. 20.8. Иерархия каталогов памяти конфигурации

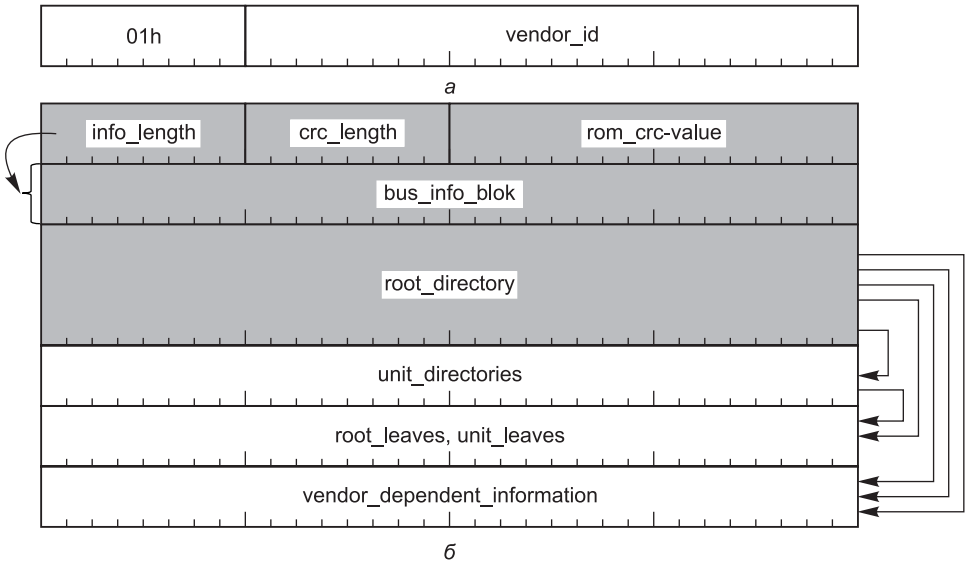


Рис. 20.9. Форматы содержимого памяти конфигурации: *а* — минимальный формат; *б* — общий формат

В минимальном формате память конфигурации содержит только 24-битный идентификатор производителя `vendor_id` (рис. 20.9 *а*). В общем формате (рис. 20.9, *б*) память содержит следующие элементы:

- заголовок (1 квадлет), содержащий три поля:
  - `info_length` — длина информационного блока последовательной шины в квадлетах (01 — признак минимального формата);
  - `crc_length` — длина блока памяти (в квадлетах), защищаемого CRC (255 — весь объем памяти, 1020 байт после заголовка);
  - `rom_crc_value` — значение CRC тела памяти (объявленного числа квадлетов после заголовка);
- `bus_info_block` — информационный блок последовательной шины;
- `root_directory` — корневой каталог, содержащий элементы описаний узла или ссылки на эти описания;
- `unit_directories` — каталоги блоков;
- `root_leaves` и `unit_leaves` — элементы-«листья», на которые ссылаются каталоги;
- `vendor_dependent_information` — специфическая информация (по усмотрению разработчика).

Информационный блок последовательной шины

Информационный блок последовательной шины (`bus_info_block`) занимает 4 квадлета (рис. 20.10). Назначение его полей:

- `bus_name` — имя шины («1394» в ASCII);
- `irmc` — признак способности к роли диспетчера изохронных ресурсов;
- `cmc` — признак способности к роли мастера циклов;
- `isc` — признак способности к изохронным передачам;
- `bmc` — признак способности к роли диспетчера шины;
- `pmc` — признак способности к управлению питанием (введено в 1394a);
- `cyc_clk_acc` — точность генерации интервалов циклов (в миллионных долях, ppm, допустимо 0–100), только для узлов с cmc = 1;
- `max_rec` — указатель максимального размера поля данных, равного $2^{max\_rec + 1}$. Допустимо `max_rec` = 1...13 (длина пакета 4, 8, 16... 16 384 байт);
- `node_vendor_id` — идентификатор производителя (то же, что и `vendor_id` в минимальном формате);
- `chip_id_hi`, `chip_id_lo` — две составные части 40-битного идентификатора микросхемы LINK-уровня, который в совокупности с `node_vendor_id` дает глобально уникальный идентификатор узла;
- `gen` — номер генерации содержимого памяти, инкрементируется с каждой его модификацией (введено в 1394a). Позволяет конфигурационному ПО сократить объем запрашиваемой информации, если считанное значение номера после сброса совпадает с предыдущим;
- `link_speed` — максимальная скорость, поддерживаемая LINK-уровнем; может не совпадать со скоростью PHY, сообщаемой в пакете самоидентификации (введено в 1394a).

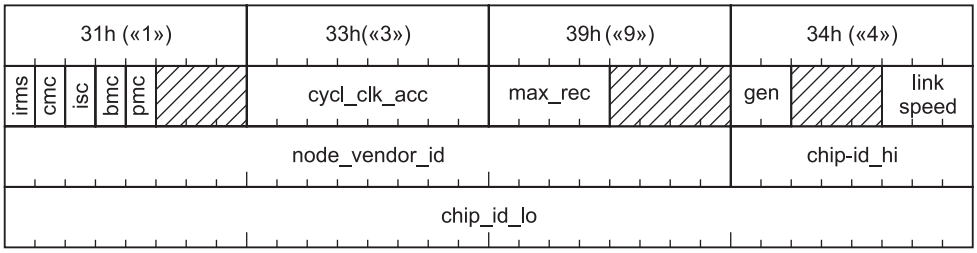


Рис. 20.10. Информационный блок последовательной шины

Корневой каталог

Корневой каталог содержит элементы, состоящие из двух полей:

- `key` (8 бит) — ключ элемента, в котором можно выделить:
  - `key_type` (2 бита) — тип ключа: 0 — непосредственный параметр, 1 — смещение (в начальном пространстве регистров), по которому находится параметр; 2 —смещение (в памяти конфигурации), по которому находится структура данных (лист); 3 — смещение, по которому находится каталог;

- `key_value` (6 бит) — значение ключа, по которому определяется назначение элемента.

◆ `value` или `offset` (24 бита) – значение элемента или смещение ячейки памяти, с которой начинается данный элемент. Смещение указывается в квадлетах относительно положения данного элемента каталога.

Элементы корневого каталога приведены в табл. 20.1. Для всех узлов шины обязательны первые три элемента из приведенных в таблице. Структура элемента, содержащего *уникальный 64-битный идентификатор узла*, приведена на рис. 20.11. Этот идентификатор, называемый EUI-64 (Extended Unique Identifier), состоит из идентификаторов производителя узла и идентификатора микросхемы LINK-уровня. Он может использоваться для идентификации программного драйвера данного узла. В CSR определен еще и 88-битный глобальный идентификатор GUI (Globally Unique Identifier), в котором к EUI-64 в качестве старших 24 бит добавлен идентификатор `Company_Id`. Идентификатор GUI уникально определяет узел во всем мире всех шин, отвечающих CSR-архитектуре.

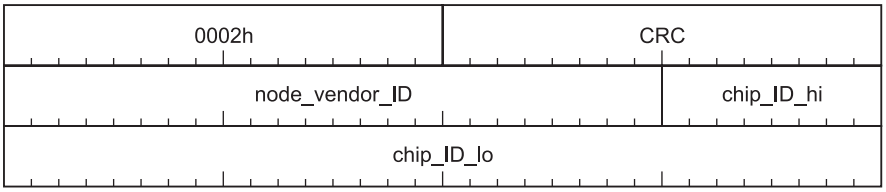


Рис. 20.11. Элемент с уникальным идентификатором узла

Таблица 20.1. Элементы корневого каталога

Ключ	Смещение или значение
03h	`Module_Vendor_Id`, значение идентификатора производителя модуля (может совпадать с идентификатором производителя узла)

0Ch	`Node_Capabilities`, значение свойств узла (рис. 20.12):
	`spt` — признак наличия регистра `SPLIT_TIMEOUT`, должен быть единичным у узла, способного участвовать в транзакциях
	`ms` — признак наличия регистров `Messaging Passing`
	`int` — признак наличия регистров `Interrupt_Traget` и `Interrupt_Mask`
	`ext` — признак наличия регистра `TEST_ARGUMENT`
	`bas` — признак наличия регистров `TEST_START` и `TEST_STATUS`
	`prv` — признак наличия приватного пространства узла
	`64` — признак поддержки 64-битной адресации (в IEEE 1394 всегда установлен)
	`fix` — признак фиксированной (а не расширенной) адресации (в IEEE 1394 всегда установлен)
	`lst` — признак наличия бита `Lost` в регистрах `State_Clear` и `State_Set`
	`drq` — признак наличия бита `Dreq` в регистрах `State_Clear` и `State_Set`
	`elog` — признак наличия регистра `Error_Log` и бита `Elog` в регистрах `State_Clear` и `State_Set`
	`atn`, `off` — признак наличия одноименных битов в регистрах `State_Clear` и `State_Set`
	`ded` — узел поддерживает состояние *Dead*
	`init` — узел поддерживает состояние *Initializing*

Ключ	Смещение или значение
8Dh	`Node_Unique_Id_offset` — смещение элемента-листа с уникальным идентификатором узла. Смещение (в квадлетах) задается относительно положения данного ключа в памяти конфигурации. Формат элемента приведен на рис. 20.11
82h	`Bus_Dependent_Info_Offset` — смещение элемента информации, относящейся к шине
C2h	`Bus_Dependent_Info_Offset` — смещение каталога информации, относящейся к шине
04h	`Module_Hw_Version`, значение версии аппаратных средств модуля
05h	`Module_Spec_Id`, значение идентификатора спецификации модуля
06h	`Module_Sw_Version`, значение версии программных средств модуля
87h	`Module_Dependent_Info_Offset` — смещение элемента информации, относящейся к модулю
C7h	`Module_Dependent_Info_Offset` — смещение каталога информации, относящейся к модулю
08h	`Node_Vendor_Id`, идентификатор производителя узла
09h	`Node_Hw_Version`, значение версии аппаратных средств узла
0Ah	`Node_Spec_Id`, значение идентификатора спецификации узла
0Bh	`Node_Sw_Version`, значение версии программных средств узла
90h	`Node_Dependent_Info_Offset` — смещение элемента информации, относящейся к узлу
D0h	`Node_Dependent_Info_Offset` — смещение каталога информации, относящейся к узлу
D1h	`Unit_Directory_Offset` — смещение каталога информации, относящейся к блоку
F0h	`Node_Power_Directory_Offset` — относительное смещение каталога управления потреблением, относящегося к узлу (введено в 1394a)

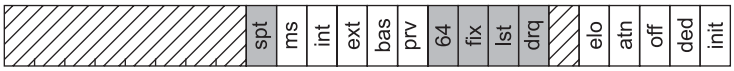


Рис. 20.12. Свойства узла (Node_Capabilities), серым цветом выделены обязательные элементы

Содержимое каталогов блоков применительно к одной из разновидностей устройств 1394 описано в главе 26, посвященной протоколу SBP-2.

ГЛАВА 21

Управление шиной IEEE 1394

Шина IEEE 1394, обеспечивая равноранговые взаимодействия между узлами, нуждается в централизованном управлении некоторыми функциями. Управляющие функции могут брать на себя разные узлы шины; в зависимости от наличия реализации тех или иных функций различают следующие варианты шины IEEE 1394:

- *неуправляемая шина*, нуждающаяся только в корневом узле (root), управляющем арбитражем. Корень, который становится «верховным арбитром», выбирается на этапе идентификации дерева. Первоначальный кандидат на эту «должность» выбирается исходя из топологии соединений, с возможным случайным розыгрышем этого права между двумя победителями предпоследнего тура (см. главу 20). После завершения выборов корня производится самоидентификация (и назначение физических адресов) узлов, после чего шина становится готовой к асинхронным транзакциям между узлами. Впоследствии программным путем (через асинхронные сообщения по шине) возможно переназначение корня (с определением новой структуры дерева и адресов узлов);
- *частично управляемая шина*, которая в дополнение к корню должна иметь узлы, выполняющие роль мастера циклов и диспетчера изохронных ресурсов. Их работа обеспечивает возможность использования шины для изохронных передач;
- *полностью управляемая шина*, которая должна иметь узел-диспетчер шины, обеспечивающий дополнительные сервисы управления.

Мастер циклов

Мастер циклов (Cycle Master) отвечает за регулярную передачу пакетов начала цикла (см. главу 19). Для этого он должен быть устройством с поддержкой изохронных обменов, иметь регистры `CYCLE_TIME` и `BUS_TIME`. В информационном блоке `BUS_INFO_BLOCK` его памяти конфигурации (см. главу 20) должен быть установлен бит `cmc` (Cycle Master Capable) — признак способности к исполнению этой роли. Текущим мастером циклов является узел, у которого в регистре состояния (`STATE`) установлен бит `cmstr` (Cycle Master). Все узлы, кроме корневого, во вре-

мя идентификации дерева (после сброса) должны обнулить у себя этот бит; корневой узел должен сохранять значение, которое было до сброса.

Если выбранный корневой узел не способен быть мастером циклов, а требуются изохронные передачи, то из узлов, способных быть мастером (судя по биту `cmc`), выбирается новый кандидат на роль корня. Для этого посылается широковещательный PHY-пакет конфигурирования с идентификатором нового кандидата и установленным битом `R`. Этот узел установит у себя бит `RHB`, а остальные его сбросят, что и обеспечит выбор данного узла новым корнем во время идентификации, вызванной посылкой этого пакета.

Мастер циклов является источником системного времени; для этого он имеет регистры `CYCLE_TIME` и `BUS_TIME`. Текущее значение регистра `CYCLE_TIME` передается мастером циклов в пакетах начала цикла. Сброс на шине (в любой форме) на значения этих регистров не влияет.

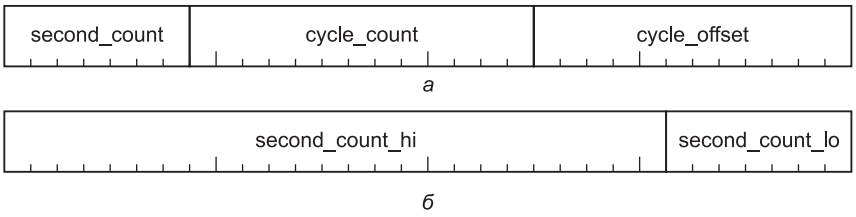


Рис. 21.1. Регистры отсчета времени: *а* — CYCLE_TIME, *б* — BUS_TIME

Регистр `CYCLE_TIME` (32 бита, рис. 21.1, *а*) состоит из трех полей, соответствующих значениям трех счетчиков, соединенных каскадно:

- `cycle_offset` — 12-битный счетчик по модулю 3072 (максимальное значение 3071, после него обнуляется), считающий импульсы с частотой 24,576 МГц. Период этого счетчика соответствует номинальной длительности цикла — 125 мкс;
- `cycle_count` — 13-битный счетчик по модулю 8000, считающий циклы. Период этого счетчика — 1 с;
- `second_count` — 7-битный счетчик, считающий секунды; период счета — 128 с.

Регистр `BUS_TIME` (32 бита, рис. 21.1, *б*) содержит значение системного времени в секундах. Его младшие 7 бит (`second_count_lo`) отображают поле `second_count` предыдущего регистра. Остальные 25 бит (`second_count_hi`) отсчитывают 128-секундные интервалы. Период счетчика составляет $2^{32} = 4\ 294\ 967\ 296$ с (около 136 лет).

Диспетчер изохронных ресурсов

Диспетчер изохронных ресурсов, *IRM* (Isochronous Resources Manager), — узел, ведающий распределением номеров каналов и полосы шины для изохронных передач. Диспетчер требуется, когда на шине появляется хоть одно устройство, спо-

собное к изохронной передаче. Диспетчер выбирается в процессе самоидентификации узлов из числа узлов, претендующих на эту роль. Претендентами являются узлы, которые сообщают в своем пакете самоидентификации единичные значения битов L (LINK активен) и C (Contender — участник состязания). Все узлы, интересующиеся диспетчерами, слушают пакеты самоидентификации и запоминают идентификатор победителя — претендента с наибольшим номером узла. Узел-претендент «сдается», если его идентификатор меньше запомненного идентификатора победителя. Победитель (единственный «не сдавшийся» узел) становится диспетчером изохронных ресурсов. Через него можно будет найти и диспетчера шины. Победителем, скорее всего, станет корневой узел (если он способен к этой роли).

На роль диспетчера изохронных ресурсов годится узел, удовлетворяющий следующему набору требований:

- поддержка изохронных обменов (как источник или приемник);
- активность канального уровня и уровня транзакций во время процесса конфигурирования;
- наличие в памяти конфигурации блока BUS_INFO_BLOCK, в котором установлен бит IRMC (способность исполнять роль IRM);
- наличие регистра BUS_MANAGER_ID, поддерживающего блокированную транзакцию (условную запись);
- наличие регистров доступных каналов и пропускной способности (CHANNELS_AVAILABLE и BANDWIDTH_AVAILABLE).

Специальные регистры диспетчера изохронных ресурсов находятся в области регистров последовательной шины начального пространства регистров узла (см. главу 20).

При отсутствии диспетчера шины IRM отвечает за назначение мастера циклов: если текущий корень не исполняет эту роль, то IRM для этой цели выбирает иной узел (в том числе и себя). Механизм смены корня по этому поводу описан в предыдущем параграфе. При отсутствии диспетчера шины на IRM ложатся еще и минимальные обязанности по управлению питанием — отправка команды на включение LINK-уровня всем узлам (см. главу 23). Отсутствие диспетчера шины IRM определяет по своему регистру Bus_Manager_ID (смещение 2C1h, используются только 6 младших бит): если через 625 мс после сброса его значение не отличается от 3Fh, значит, диспетчера шины нет.

После сброса устройства, нуждающиеся в изохронной передаче, запрашивают у диспетчера изохронных ресурсов выделение номера канала и полосы пропускания.

Для распределения каналов служит 64-битный регистр CHANNEL_AVAILABLE (смещение 224h), представляющий собой битовую карту, в которой каждому возможному изохронному каналу ch# соответствует свой бит (ch0 — в младшем бите). Единичное значение бита соответствует доступности (незанятости) данного канала. Начальное значение регистра — FFFF FFFFh (все каналы доступны). При запросе выделения канала бит сбрасывается, при освобождении — устанавливается. Запрос и освобождение канала выполняются одинаково. Узел, запрашивающий выделение или освобождение канала, первым делом считывает значение регистра

CHANNEL_AVAILABLE, чтобы определить текущую ситуацию, и формирует свое предложение по новому значению этого регистра. Далее, он обращается к регистру CHANNEL_AVAILABLE блокированной транзакцией *compare_swap*, в которой в качестве аргумента (arg_value) передает прежнее (считанное) значение, а в качестве значения (data_value) — свое новое предложение. Данная транзакция в случае совпадения текущего значения регистра с аргументом заменяет его значение на новое, то есть регистр CHANNEL_AVAILABLE принимает новое значение. При этом транзакция возвращает запрашивающему узлу значение регистра, считанное на момент начала выполнения блокированной транзакции. Если это значение совпадает с ранее считанным значением, значит, диспетчер принял новое назначение каналов — выделил узлу или освободил запрошенный канал (можно и группу каналов одновременно). Если совпадения нет, это значит, что какой-то другой узел между нашими двумя транзакциями успел изменить содержимое регистра — занять или освободить канал. В этом случае запрашивающий узел должен повторить попытку, при этом новое текущее значение регистра он уже знает.

Для распределения полосы пропускания служит регистр BW_AVAILABLE (смещение 220h), в котором младшие 13 бит (bw_remaining) содержат число *единиц распределения* полосы шины (bus allocation units), оставшееся доступным. Единица распределения соответствует времени передачи одного квадлета на скорости 1600 Мбит/с (20,345 нс). В 125-микросекундном цикле теоретически доступно 6144 единиц. Как минимум 25 мкс цикла резервируется под асинхронный трафик, поэтому суммарная распределяемая полоса изохронного трафика (начальное значение bw_remaining) по умолчанию составляет 4915 единиц. Сервисным запросом *SB_CONTROL* диспетчеру изохронных ресурсов (или диспетчеру шины) можно урезать полосу, выделяемую под изохронный обмен. Выделение полосы по времени учитывает возможность совместной работы устройств с различными скоростями — в одном цикле соседние пакеты могут передаваться на разных скоростях. Для цифрового видео, например, требуется полоса 30 Мбит/с (25 Мбит/с на видеоданные и 3–4 Мбит/с на аудиоданные, синхронизацию и заголовки пакетов). В S100 такие устройства цифрового видео запрашивают около 1800 единиц, в S200 — около 900. Запрашиваемая полоса должна учитывать не только передаваемые данные, но и накладные расходы на заголовок и на зазор арбитража. Выделение и освобождение полосы выполняется аналогично выделению и освобождению канала — с помощью чтения и блокированного обращения *compare_swap* к регистру BW_AVAILABLE. При сбросе шины передача трафика прекращается и все ресурсы освобождаются, так что узлы, вещавшие в изохронном режиме, должны снова запросить себе ресурсы. Для стабилизации работы шины узлы-новички, не вещавшие до сброса, задерживают свои запросы на 1 с — это позволит прежним вещателям не потерять ресурсы из-за появления новичков и не прерывать передачу.

Диспетчер шины

Диспетчер шины (Bus Manager) обеспечивает полное управление шиной. Им может стать любой узел, способный (и обязанный после избрания диспетчером) выполнять следующие функции:

- назначение (при необходимости) мастера циклов: если текущий корень не исполняет эту роль, то диспетчер шины выбирает на роль корня иной узел;
- управление питанием;
- публикация карты топологии;
- публикация карты скоростей;
- оптимизация трафика шины (изменение значения регистра `BW_AVAILABLE` в диспетчере изохронных ресурсов).

Диспетчер шины может находиться в любом месте шины. Он выбирается из узлов-кандидатов на роль диспетчера изохронных ресурсов. Из этих кандидатов диспетчером шины может стать узел, способный выполнять блокированные транзакции *compare_swap* с регистром `Bus_Manager_ID` уже избранного диспетчера изохронных ресурсов. После самоидентификации каждый узел, претендующий на роль диспетчера шины, пытается с помощью транзакции *compare_swap* записать свой идентификатор в регистр `Bus_Manager_ID`. В качестве аргумента (`arg_value`) этой транзакции передается значение 3Fh, а в качестве значения (`data_value`) — идентификатор. Значение, полученное в ответ, несет информацию о назначенном диспетчере шины: 3Fh — диспетчером стал данный узел, другое значение — идентификатор узла, успевшего стать диспетчером ранее. Если из-за ошибки передачи транзакция не удалась, узел должен ее автоматически повторить. При этом он может получить в ответе свой `PHY_ID` — это означает, что он стал диспетчером еще в предыдущей попытке, но не получил подтверждения.

Для поиска диспетчера шины узел должен прочитать регистр `Bus_Manager_ID` диспетчера изохронных ресурсов. Идентификатор диспетчера изохронных ресурсов узлы запоминают в ходе самоидентификации (см. предыдущий параграф).

Управление питанием

У каждого узла при подключении к шине должен быть включен физический уровень (PHY) и контроллер шины, обеспечивающие инициализацию, самоидентификацию и трансляцию сигналов между портами. Остальные компоненты (LINK и прикладная часть) могут быть отключены, об активности (включении) верхних уровней (LINK и уровень транзакций) узел сообщает битом `L` в пакете самоидентификации. Отношение узла к линиям питания может быть различным, и о нем он сообщает в своем пакете самоидентификации в поле `pwr` (см. главу 22).

Диспетчер шины во время самоидентификации собирает информацию от всех узлов шины об их отношении к питанию (из поля `pwr`). Он подсчитывает баланс питания (сумма запрашиваемых мощностей не должна превышать сумму подаваемых мощностей). Если баланс не сходится — питания для всех узлов недостаточно, то сервис управления питанием посылает управляющему приложению индикацию события *SB_EVENT* с указанием на недостаток питания. Дальнейшие действия зависят от приложения: или пользователю предлагается выбрать часть устройств, которые включать, или не включается ни одно устройство (с уведомлением пользователя). Если питания достаточно, то всем узлам с неактивным LINK-

уровнем сервис *SB_CONTROL* посылает пакет *Link_On* — директиву на включение питания (см. главу 23). Если питанием управляет диспетчер изохронных ресурсов (за неимением диспетчера шины), то он лишь посылает пакеты *Link_On* всем узлам с неактивным LINK-уровнем, не заботясь о бюджете мощности.

Карты топологии и скоростей

Знание топологии шины дает информацию о возможностях ее оптимизации, которая может быть выполнена путем:

- сокращения максимального количества кабельных сегментов между самыми удаленными узлами. Это позволяет снижать задержки распространения сигналов, увеличивая эффективность использования шины (задержки являются бесполезной тратой времени);
- перегруппировки узлов с учетом поддерживаемых ими скоростей передачи, что позволяет более эффективно пользоваться высокоскоростными передачами.

Диспетчер шины собирает информацию о топологии и предоставляет ее по запросам чтения любым узлам сети — публикует *карту топологии* (Topology_Map). В этой карте после заголовка размещаются копии всех пакетов самоидентификации, посланных узлами во время самоидентификации. Диспетчер шины отвечает за контроль целостности карты — число p-портов (родительских) должно совпадать с числом c-портов (дочерних). Если целостность нарушена, диспетчер обнуляет поле длины карты и посылает управляющему приложению уведомление *SB_EVENT*.

По карте топологии диспетчер шины способен определить максимальное число хопов (кабельных сегментов) между узлами сети. Это позволяет определить минимальный зазор арбитража (subaction gap), не конфликтующий с пакетами квитирования, и сообщить его всем узлам сети. Минимальный зазор вычисляется через задержку распространения сигнала в кабельных сегментах и повторителях. Зазор арбитража должен быть больше, чем время оборота по шине: до истечения времени зазора арбитража передающий узел должен успеть получить квитанцию на пакет, посланный самому дальнему узлу. Вместо вычисления задержки диспетчер шины может измерить время оборота, посылая выбранному узлу пробный пакет `Ping` (см. главу 23) и измеряя время до прихода ответа — пакета самоидентификации (или серии этих пакетов). Измерение реальных задержек позволяет использовать кабели длиннее 4,5 м (если у них малое затухание) и узлы-повторители, вносящие задержку более 144 нс. Задание зазора осуществляется широковещательно пакетом конфигурирования с установленным битом `T` (см. главу 23).

Карта топологии располагается с адреса 1000h в начальном пространстве узла-диспетчера шины, ее формат приведен на рис. 21.2. В карте имеются следующие поля:

- `length` — длина карты (в байтах). Нулевая длина — признак некорректности карты. Длина проверяется до и после чтения информационных элементов. Несовпадение этих длин означает, что считанные элементы недостоверны (нужно повторить чтение карты, поскольку она только что изменилась);

- `CRC` — контрольный код;
- `generation_number` — номер генерации карты (увеличивается с каждой ее модификацией);
- `node_count` — число узлов сети;
- `self_id_count` — число пакетов самоидентификации в карте (может быть больше числа узлов, поскольку некоторые узлы могут посылать и по несколько пакетов самоидентификации;
- `self_id_packet[i]` — собственно пакеты самоидентификации (см. главу 23).

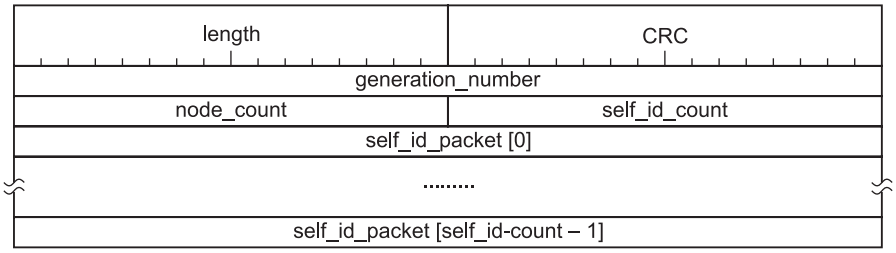


Рис. 21.2. Формат карты топологии

По карте топологии диспетчер шины строит *карту скоростей* (Speed_Map), которая позволяет определить максимальную скорость, возможную при обмене между любой парой узлов. Логически карта представляет собой двухмерную матрицу, в которой каждой паре узлов с идентификаторами *m* и *n* соответствует байт кода скорости (speed_code). Младшие 3 бита этого байта соответствует коду скорости в формате поля `xspd` пакета самоидентификации. Фактически карта — это последовательность байтов speed_code[i], где $I = 64 \times m + n$. Формат карты скоростей приведен на рис. 21.3, ее заголовок аналогичен заголовку карты топологии. Доступ к карте скоростей выполняется аналогично карте топологий, с проверкой поля длины до и после обращения.

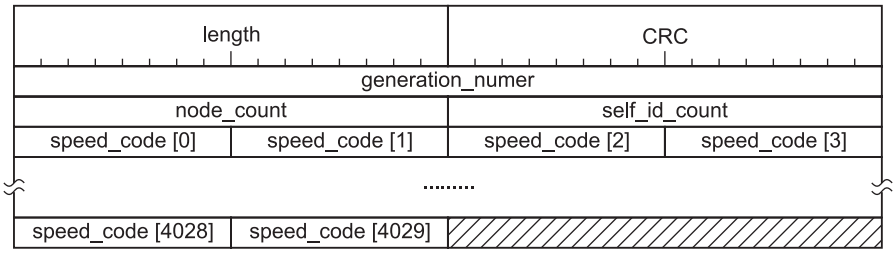


Рис. 21.3. Формат карты скоростей

Сервисы управления шиной

Прикладные драйверы на каждом узле предоставляют специальный интерфейс управления шиной (Bus Management Interface), который взаимодействует с «уров-

нем» управления последовательной шиной (Serial Bus Management Layer). Этот «уровень» проходит по всем этажам архитектуры (рис. 17.4). Он включает:

- контроллер узла, управляющий уровнями PHY и LINK;
- мастер циклов;
- диспетчер изохронных ресурсов;
- диспетчер шины.

Взаимодействие происходит через *сервисы управления последовательной шиной* (Serial Bus Management Services) при помощи:

- запросов управления шиной — *SB_CONTROL.request*;
- подтверждений запросов — *SB_CONTROL.confirmation*, сообщающих результаты их выполнения;
- индикаторов событий шины, происходящих неожиданно для данного узла, — *SB_EVENT.indication*.

Запросы и подтверждения сервисов управления

Для функций управления и определения состояния предусмотрен ряд запросов, (*SB_CONTROL.request*), перечисленных ниже. На все запросы уровень управления отвечает подтверждениями (*SB_CONTROL.confirmation*), указывающими на успех или неудачу выполнения запроса. Сервисы управления включают следующие запросы:

- сброс шины (Bus Reset) — указание своему PHY подать сигнал сброса и реинициализироваться самому; LINK и уровень транзакций отбрасывают все ожидающие транзакции и субакции;
- инициализация узла — LINK и уровень транзакций данного узла отбрасывают все ожидающие транзакции и субакции и становятся готовыми к работе;
- включение LINK-уровня заданного узла. Этот запрос предоставляет только диспетчер шины и диспетчер изохронных ресурсов;
- конфигурирование физического уровня шины, предоставляют только диспетчер шины и диспетчер изохронных ресурсов:
  - принудительное назначение роли корня (*Set Root Force*) — установка бита `RHB` в указанном узле и сброс во всех остальных;
  - установка зазора арбитража (*Set Gap Count*);
  - посылка расширенных пакетов физического конфигурирования — пробных пакетов *ping*, удаленного доступа к регистрам PHY, пакета *resume*;
- Опрос состояния (*Present Status*), по которому в подтверждениях (*SB_CONTROL confirmation*) сообщается следующая информация:
  - идентификатор корневого узла (3Fh — корневой узел не является мастером циклов);
  - идентификатор (физический адрес) данного узла (0...3Eh);
  - состояние бита `RHB`, обеспечивающего победу данному узлу в состязаниях на роль корня;

- текущее значение зазора арбитража (`gap_cnt`);
- идентификатор диспетчера шины (3Fh — нет диспетчера);
- идентификатор диспетчера изохронных ресурсов (3Fh — нет диспетчера);
- идентификатор мастера циклов (3Fh — нет мастера);
- значение, которое можно вычесть из содержимого регистра `BANDWIDTH_AVAILABLE` при перераспределении полосы между изохронным и асинхронным трафиком.

Последние четыре параметра доступны только у узла, являющегося диспетчером шины или диспетчером изохронных ресурсов.

Индикация событий управления шиной

Локальное приложение узла получает следующие сообщения об особых событиях, происходящих на шине и в данном узле:

- нарушение соглашений о времени занятия шины (регистр `MAX_BUS_OCCUPANCY`, переименованный в `MAX_DATA_TIME` в IEEE 1394a);
- начало сброса шины;
- завершение сброса шины (выполнен сброс, идентификация дерева и самоидентификация всех узлов). При этом сообщается ряд параметров и признаков:
  - идентификаторы данного узла, корня, мастера циклов, диспетчеров шины и изохронных ресурсов;
  - ошибки тайм-аута конфигурирования, топологии, самоидентификации, определения зазора, перегрузки питания;
  - значение зазора арбитража;
  - значение остатка полосы (bandwidth set-aside);
- слишком длинный цикл (только для диспетчеров);
- понижение питания в кабеле (ниже 7,5 В);
- обнаружение дублирования номеров изохронных каналов;
- обнаружение ошибки CRC-кода заголовка;
- обнаружение ошибки CRC-кода данных;
- отсутствие квитанции на переданный пакет ответа;
- получение квитанции с указанием на ошибку данных в пакете ответа;
- ошибка формата ответа (получен пакет квитирования с указанием неверного типа ответа);
- ошибка повтора ответа (исчерпан предел повторов или ожидания);
- обнаружен неожиданный канал (диспетчер изохронных ресурсов услышал пакет не выделенного им канала);
- обнаружен неизвестный код транзакции (не поддерживаемой данным узлом);
- обнаружен приход ответа, не ожидаемого данным узлом (проверяется по метке транзакции).

Управление энергопотреблением

Возможность питания устройств от кабельной шины требует управления энергопотреблением узлов. В минимальном варианте, заложенном в изначальной спецификации, управление сводится к простому подсчету баланса мощности на основе классов питания узлов и включением LINK-уровня по команде от диспетчера. В IEEE 1394a управление было усовершенствовано, в частности:

- появилась возможность приостановки и возобновления работы узлов (*suspend* и *resume*);
- появилась возможность управлять уровнем потребления узлов и блоков;
- расширились возможности расчета баланса мощности с учетом батарейного питания и уровней питания узлов и блоков.

Эти новые возможности потребовали введения дополнительных регистров CSR и элементов в памяти конфигурации.

В дополнении IEEE 1394b введено новое энергосберегающее состояние порта и узла — *Standby*, в котором PHY не обеспечивает взаимодействия своего узла с шиной (см. главу 22).

Приостановка и возобновление (Suspend и Resume)

Приостановка (suspend) — это переход пары портов, соединенных кабельным сегментом, в состояние малого энергопотребления. В этом состоянии передача трафика по данному сегменту невозможна. Однако приостановленный порт способен определять события отключения и подключения своего партнера. Для того чтобы стала возможной передача данных по приостановленному сегменту, порты должны выполнить *возобновление* (resume) нормальной работы. Приостановка и запрет портов меняют конфигурацию шины — она может оказаться разбитой на активные домены, изолированные друг от друга (в плане трафика). Возобновление опять-таки меняет конфигурацию. В связи с этим приостановка и возобновление сопровождаются сигнализацией сброса (короткого) и реинициализацией шины.

Приостановка по команде Suspend

Команда *Suspend* приводит к разделению шины на два фрагмента (домена): активный и приостановленный. Инициатор этой приостановки всегда остается в активном домене.

Команда *Suspend* принимается от удаленного узла с помощью расширенного физического пакета или от собственного LINK-уровня. В команде указывается идентификатор узла и номер его приостанавливаемого порта. В ответ на эту команду целевой узел приостановки посылает пакет подтверждения. Получив подтверждение команды приостановки, инициатор приостановки посылает сигнал *TX_SUSPEND* на тот порт, откуда пришло подтверждение. На остальные порты через

короткий зазор узел посылает префикс данных, а за ним — короткий сигнал сброса шины. Это приводит к реконфигурированию активной части шины.

Получив сигнал приостановки *RX_SUSPEND* на одном из своих портов, целевой узел (которому посылали команду) прекращает подачу смещения (TpBias, см. главу 22) на этот порт. На все свои остальные порты этот узел посылает сигнал *TX_SUSPEND*, инициируя приостановку и их партнеров. Таким образом, приостановка распространится на все порты и узлы шины, которые связаны с инициатором приостановки через приостанавливаемый порт.

Узел, обнаруживающий пропадание смещения, также прекращает подачу смещения на этот порт, что завершает согласование приостановки. С этого момента в приостановленных портах остаются работать только детекторы отключения (см. главу 22).

Приостановка по команде запрета порта

Команда запрета порта (локальная или удаленная) переводит в приостановленное состояние только указанный порт и его партнера. Таким образом, в результате шина может оказаться разбитой на два не связанных между собой активных домена.

Получив пакет с командой запрета порта (*Disable Port*), узел посылает во все порты, кроме запрещенного, пакет подтверждения, а за ним — короткий сигнал сброса. На запрещенный порт узел посылает сигнал *TX_DISABLE_NOTIFY*. Его партнер, приняв сигнал *RX_DISABLE_NOTIFY*, посылает на остальные порты префикс данных, а за ним — короткий сброс. Этот порт переводится в приостановленное состояние (снятием смещения, на что ответом будет снятие смещения и у запрещенного порта).

Возобновление нормальной работы

Возобновление работы порта может инициироваться несколькими способами:

- по приему пакета *RESUME*, который может быть послан широковещательно (или направленно) от узла, находящегося в активном домене шины. Этот пакет воспринимается и отрабатывается узлами активного домена, у которых имеются приостановленные порты. Все эти порты переводятся в активное состояние;
- по приему командного пакета с командой *Resume Port*, с указанием ранее приостановленного или запрещенного порта. Адресованный узел подает на указанный порт смещение, что приведет к возобновлению работы всего приостановленного домена (или возобновит работу запрещенного порта);
- по событию, обнаруженному портом. Возобновление могут вызывать события смены состояния подключения, смены уровня смещения, перевод в запрещенное состояние и отказ порта. Каждое из этих событий вызывает пробуждение, если для него установлен соответствующий бит в поле разрешения прерываний.

Уровни потребления узла и блоков

В IEEE 1394a введены понятия *уровней потребления* (Power States), относящиеся к узлу в целом и отдельным его блокам. Узел и блок могут поддерживать до четы-

рех уровней (0...3), из которых нулевой (обязательный) соответствует функционированию в самом полном объеме.

Уровни потребления узла (Node Power States) *N0...N3* определяют состояния уровней PHY и LINK:

- *N0* — состояние полной дееспособности узла: PHY-уровень запитан (может принимать, посылать и транслировать пакеты и сигналы), LINK-уровень способен отвечать на транзакции, обращенные к узлу. Контекст узла (все его конфигурационные регистры CSR и PHY) действителен. При этом возможны два варианта:
  - LINK-уровень полностью запитан — полная функциональность узла (нормальные ответы на все транзакции);
  - LINK-уровень находится в состоянии *Standby* (с пониженным потреблением). Узел способен декодировать адрес обращенных к нему транзакций и ответить на них квитанцией *Ack_TRDY*, что вызовет повтор транзакции инициатором в следующем интервале справедливости. За это время LINK-уровень узла успеет включиться, и на следующую попытку узел ответит нормальным образом;
- *N1* — состояние со включенным PHY (узел транслирует сигналы и пакеты) и отключенным LINK-уровнем. Это соответствует состоянию узла после сброса до получения пакета *Link-On*. Состояние контекста узла стандартом не регламентировано;
- *N2* — состояние с приостановленным (suspended) PHY и отключенным LINK-уровнем. Узел не транслирует пакеты и сигналы, он может только реагировать на внешние сигналы, вызывающие возобновление (или на сигнал от своего приложения). Контекст не определен; информация, связанная с топологией, недействительна (после возобновления будет сброс);
- *N3* — полностью обесточенный узел, контекст потерян.

Уровни потребления блока (Unit Power States) *D0...D3* отражают функциональность и потребление блока:

- *D0* — состояние полной функциональности и полного потребления, обязательное для всех блоков;
- *D1* — состояние пониженного потребления, контекст блока сохраняется, но функциональность может быть ограничена. Переход *D1→D0* может совершаться довольно быстро;
- *D2* — состояние еще меньшего потребления, с меньшей функциональностью и, возможно, потерей контекста блока. Переход в *D0* (или *D1*) может занимать большее время, чем *D1→D0*;
- *D3* — полное обесточивание блока.

Возможные состояния уровней потребления узла и входящих в него блоков связаны между собой: номер уровня потребления узла должен быть не больше, чем наименьший номер уровня потребления его блоков. Например, если в узле два блока и их текущие уровни потребления *D1* и *D2*, то узел может находиться на уровне *N0*

или *N1*. Попытка (запрос) перевода уровня узла с *N0* на *N2* или *N3* приведет к переводу только на уровень *N1*.

Для каждого поддерживаемого уровня потребления узла в памяти конфигурации имеются исчерпывающие описания условий питания и потребляемой (отдаваемой) мощности. Кроме того, введены регистры CSR, отражающие текущие уровни потребления для узла и блоков и управляющие их сменой. Смену уровней запрашивает диспетчер энергопотребления, который может выполнять удаленное управление любыми узлами шины и их блоками (как и своим собственным узлом и его блоками).

Регистры и структуры данных для управления энергопотреблением

Узлы, участвующие в управлении энергопотреблением, должны иметь единичное значение бита `pmc` в информационном блоке последовательной шины. Новые регистры CSR, введенные для управления энергопотреблением, располагаются в начальном пространстве узла, начиная с адреса FFFF F001 0000h или выше. Регистры делятся на две группы:

- регистры, относящиеся к узлу (node-specific CSR);
- регистры, относящиеся к блоку (unit-specific CSR).

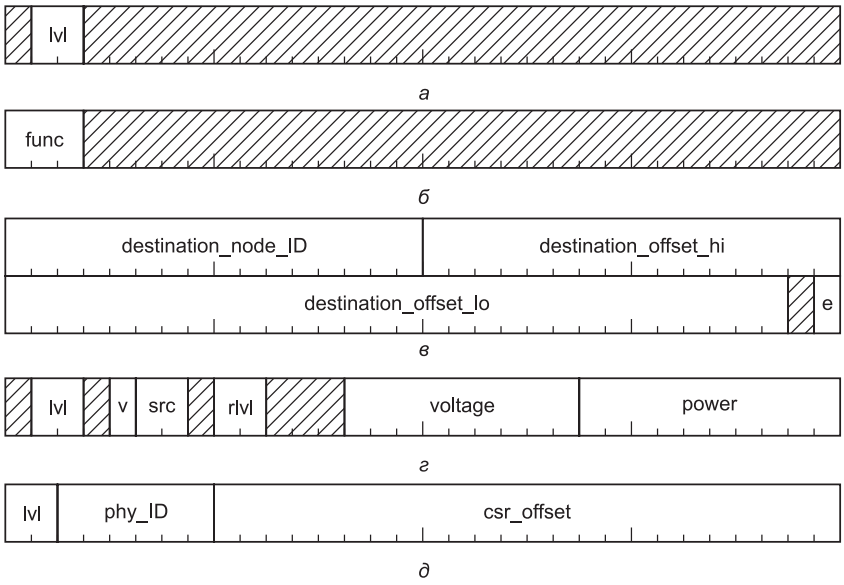


Рис. 21.4. Регистры управления питанием узла: *а* — регистр состояния питания узла `NODE_POWER_STATE`; *б* — регистр управления питанием узла `NODE_POWER_CONTROL` и регистр управления кабельным питанием `CABLE_POWER_SOURCE_CONTROL`; *в* — регистр адреса уведомления `NOTIFICATION_ADDRESS`; *г* — регистр состояния кабельного питания `CABLE_POWER_SOURCE_STATE`; *д* — регистр смены состояния потребления `POWER_CHANGE`

Форматы *регистров*, *относящихся к узлу*, приведены на рис. 21.4. На положение этого блока регистров указывает элемент `Node_Power_Management` в конфигурационной памяти. Ниже в скобках указано смещение регистров от начала блока.

Обязательный *регистр состояния питания узла* `NODE_POWER_STATE` (00h) предназначен для сообщения текущего уровня потребления.

Обязательный *регистр управления питанием узла* `NODE_POWER_CONTROL` (04h) служит только для управления сменой уровня потребления (поле `lvl`). Значения поля `func`: 0 — резерв, 1 (*grant*) — разрешение смены состояния на последнее запрошенное; 2 (*deny*) — запрет смены состояния; 3 (*wait*) — выдержка 5 с перед отработкой последующей команды смены состояния; 4...7 (*Set Level 0... Set Level 3*) — установка указанного уровня (отразится в поле `lvl` в регистре состояния питания узла).

Необязательный *регистр адреса уведомления* `NOTIFICATION_ADDRESS` (08h) содержит полный адрес, по которому следует посылать уведомление о смене уровня потребления (идентификатор узла в поле `destination_node_id`, адрес в полях `destination_offset_hi` и `destination_offset_lo`). Бит `e` разрешает узлу генерировать уведомление.

Необязательный *регистр состояния кабельного питания* `CABLE_POWER_SOURCE_STATE` (0Ch) имеет следующие поля:

- `power` — мощность, отдаваемая узлом в кабель (в десятых долях ватта);
- `voltage` — напряжение питания (в десятых долях вольта);
- `rlvl` — текущий запрашиваемый уровень потребления (0...3);
- `src` — текущее состояние питания узла: 0 — от шины, 1 — от батареи, 2 — от сети, 3 — резерв;
- `v` — признак действительности полей `power` и `voltage`;
- `lvl` — текущий уровень потребления узла (0...3), соответствует состояниям *N0...N3*.

Необязательный *регистр управления кабельным питанием* `CABLE_POWER_SOURCE_CONTROL` (10h) управляет отработкой команд. Значения поля `func`: 0 — резерв, 1 (*grant*) — разрешение смены состояния на последнее запрошенное; 2 (*deny*) — запрет смены состояния; 3 (*wait*) — выдержка 5 с перед отработкой последующей команды смены состояния; 4...7 (*Set Level 0... Set Level 3*) — установка указанного уровня (поле `lvl` в регистре состояния кабельного питания).

Регистр *смены уровня потребления* `POWER_CHANGE`, обязательный для узла, управляющего потреблением, позволяет его приложениям управлять состоянием энергопотребления любого узла или блока. Поле `lvl` задает желаемый (запрашиваемый) уровень потребления узла, заданного полем `PHY_ID`. Поле `csr_offset` задает конкретный блок или весь узел, для которого требуется смена уровня.

Форматы *регистров*, *относящихся к блоку*, аналогичны приведенным на рис. 21.4, *а* и *б*. На положение этого блока регистров указывает элемент `Unit_Power_Management` в конфигурационной памяти.

Регистр состояния питания блока `UNIT_POWER_STATE` (00h) имеет следующие поля:

- `power` — мощность, отдаваемая блоком в кабель или потребляемая (в десятых долях ватта);
- `voltage` — напряжение питания (в десятых долях вольта);
- `k` — признак того, что блок является источником события пробуждения;
- `rlvl` — текущий запрашиваемый уровень потребления (0...3);
- `src` — текущее состояние питания блока: 0 — от узла, 1 — от собственной батареи, 2 — от сети, 3 — блок подает питание в кабель;
- `v` — признак действительности полей power и voltage;
- `lvl` — текущий уровень потребления (0...3), соответствует состояниям *D0...D3*.

Регистр управления питанием блока `UNIT_POWER_CONTROL` (04h) служит для установки уровня потребления с помощью поля `func`: 0...3 — резерв; 4...7 (*Set Level 0... Set Level 3*) — установка указанного уровня (поле `lvl` в регистре состояния питания блока).

Для описания возможностей управления потреблением узла в корневом каталоге памяти конфигурации введен элемент, ссылающийся на *PM-каталог узла Node_Power_Directory*. Аналогичный элемент, ссылающийся на *PM-каталог блока*, может присутствовать в каталоге блока. В PM-каталогах могут присутствовать элементы, приведенные на рис. 21.5.

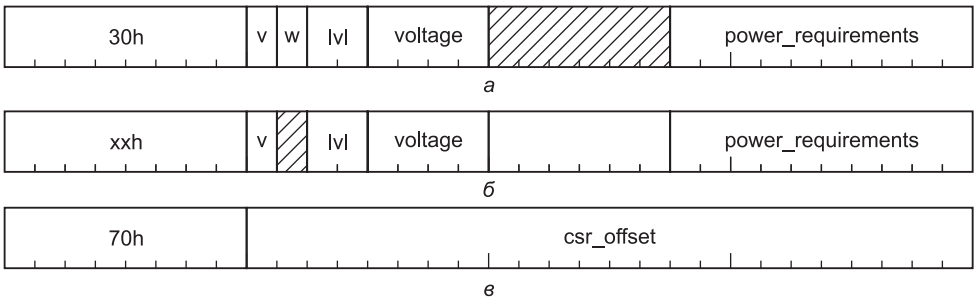


Рис. 21.5. Элементы PM-каталогов: *а* — описатели уровней потребления узла и блока; *б* — описатели кабельного питания; *в* — указатели на блоки регистров

Описатели уровней потребления узла `Node_Power_Level` должны присутствовать для всех уровней потребления, поддерживаемых устройством. Аналогичные *описатели уровней потребления блока* `Unit_Power_Level` могут присутствовать в PM-каталоге блока. В описателях содержатся следующие поля:

- `power_requrements` — мощность, потребляемая узлом или блоком на данном уровне (в десятых долях ватта);
- `voltage` — требуемое напряжение: 0 — в соответствии с 1394–1995 (8–40В), 1 — 3,3 В, 2 — 5 В, 3 — 12 В, 4–Fh — резерв;
- `lvl` — уровень потребления (0–3), описываемый данным элементом (соответствует *N0...N3* для узла и *D0...D3* для блока);

- `w` — способность приостановки (standby) LINK-уровня (бит действителен только в описателе нулевого уровня);
- `v` — признак действительности описателя.

Описатели кабельного питания `Cable_Power_Source_Level` сообщают возможности поставки питания в шину на каждом уровне потребления. Описатели относятся только к узлу. В них содержатся следующие поля:

- `power_requrements` — мощность, подаваемая узлом в кабель на данном уровне (в десятых долях Вт);
- `voltage` — подаваемое напряжение: 0 — в соответствии с 1394–1995 (8–40 В), 1 — 12 В, 2–Fh — резерв;
- `lvl` — уровень потребления (0–3), описываемый данным элементом;
- `v` — признак действительности описателя.

Указатель `Node_Power_Management` в поле `csr_offset` содержит смещение в области регистров CSR, по которому находится группа *регистров управления энергопотреблением узла*.

Указатель `Unit_Power_Management` в поле `csr_offset` содержит смещение в области регистров CSR, по которому находится группа *регистров управления энергопотреблением блока*.

Батареи питания (в том числе и аккумуляторные) могут относиться как к узлу в целом, так и к его отдельным блокам. К каждой батарее относится *регистр состояния батареи* `BATTERY_STATE_REGISTER` со следующими полями (рис. 21.6):

- `capacity` — емкость полностью заряженной батареи в ватт-часах;
- `available` — текущий уровень заряженности (в процентах от полной емкости);
- `voltage` — напряжение батареи (в десятых долях вольта);
- `st` — состояние: 0 — батарея отсутствует, 1 — резерв, 2 — батарея установлена и используется, 3 — батарея заряжается.

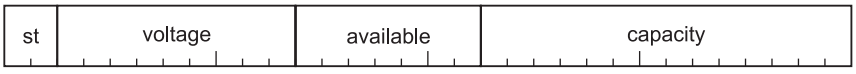


Рис. 21.6. Регистр состояния батареи

На местоположение регистров состояния батарей указывают элементы `Battery_State`, находящиеся в каталогах групп батарей `Battery_Group`. На каталоги групп батарей имеются ссылки в PM-каталоге узла `Node_Power_Directory` (для батарей узла) и/или в PM-каталогах блоков `Unit_Power_Directory` (для индивидуальных батарей блоков).

ГЛАВА 22

Физический уровень шины IEEE 1394

Физический уровень PHY (Physical Layer) IEEE 1394 обеспечивает связь узлов в единую шину. Физический уровень стандарта определяет механические характеристики соединительных разъемов, а также требования к их характеристикам передачи сигналов. В стандарте определено несколько типов разъемов, соответствующих применяемым кабелям. В 1394b в физическом уровне введено дополнительное разделение: введен подуровень PMD (Physical Medium Dependent), зависящий от используемой среды передачи (разновидностей медных и оптических кабелей).

В современной редакции стандарта IEEE 1394 фигурирует два типа сигнализации — традиционная DS-сигнализация шин 1394 и 1394a и бета-сигнализация, введенная в IEEE 1394b. Эти типы используют разные схемы сигнального кодирования — DS-кодирование и кодирование 8B10B; бета-сигнализация допускает и разнообразие типов среды передачи (электрические и оптические кабели). Все эти различия в основном сосредоточены на физическом уровне.

Физический уровень выполняет кодирование и декодирование сигналов состояния шины и потоков данных. В многопортовых узлах он обеспечивают и трансляцию сигналов между своими портами.

Физический уровень узла обеспечивает функционирование шины (включая трансляцию сигналов и отработку автоконфигурирования) без участия и даже при отключенном LINK-уровне (и вышестоящих). Узел может и не иметь LINK-уровня — его многопортовый PHY будет выполнять функции кабельного концентратора-повторителя.

Физический уровень отвечает за кабельное питание — его подачу или потребление. На физическом уровне может быть организована гальваническая развязка узлов. Для DS-сигнализации эта развязка реализуется в интерфейсе PHY-LINK, для бета-сигнализации развязка возможна в сигнальных цепях кабеля.

На физическом уровне решаются задачи конфигурирования шины и арбитража, рассмотренные в главах 19 и 20. Вопросы взаимодействия с физическим уровнем рассмотрены в главе 23.

Физический интерфейс

Первый вариант физического интерфейса шины был определен в IEEE 1394–1995 и в дополнениях 1394a–2000 не претерпел существенных изменений. В этом варианте каждое кабельное соединение состоит из двух пар сигнальных электрических проводов и, дополнительно, пары проводов для подачи питания. Обе сигнальные пары используются для двунаправленной передачи сигналов, дифференциальных и линейных. При передаче данных по одной паре узел передает данные в последовательном коде, по другой — стробы. Этот режим получил название *DS-Mode* (Data-Strobe), он обеспечивает простой механизм синхронизации приемника и передатчика при любой скорости обмена. Для служебной сигнализации (например, об обнаружении подключения/отключения узла) используется сигнализация постоянным током. Кроме того, требуется различать несколько уровней напряжения для сигнализации скорости. Из-за этого гальваническая развязка узлов на уровне кабельного интерфейса оказывается невозможной.

В IEEE 1394b введен режим сигнализации *Beta-Mode*, в котором используются две встречные однонаправленные сигнальные линии. Примененный метод сигнального кодирования 8B/10B избавляет от постоянной составляющей сигнала; избыточность кодирования позволяет использовать «лишние» символы для специальной сигнализации. Приемнику не требуется распознавать несколько уровней сигнала. Это позволяет использовать как электрическую, так и оптическую передачу, а для электрической передачи возможна полная гальваническая развязка приемников и передатчиков через импульсные трансформаторы. Порты 1394b могут быть универсальными «двуязычными» (bilingual), поддерживающими оба режима, или чисто бета-портами.

Кабели и коннекторы для DS-режима

Разъемы, используемые в кабельной шине IEEE 1394, специально разработаны для обеспечения «горячего» подключения/отключения. Контакты, используемые для цепей земли и питания (VG и VP), длиннее других — они при подключении соединяются раньше, а при отключении разъединяются позже сигнальных. На портах устанавливаются гнезда, на кабелях — вилки.

Стандарт IEEE 1394 определяет малогабаритный 6-контактный разъем для подключения устройств, изображенный на рис. 22.1, *а*. Его экранирующий кожух используется как дополнительный контакт и связывается с оплетками витых пар и контактом цепи VG.

В IEEE 1394a дополнительно определен миниатюрный 4-контактный разъем: вилка (на кабель) с плоскими контактами и розетка с пружинными контактами (рис. 22.1, *б*). В качестве дополнительного контакта (цепь VG) используется экранирующий кожух. Подача питания через такой разъем не предусмотрена. Существуют кабели-адаптеры с 4-контактной и 6-контактной вилками на разных концах.

В IEEE 1394b определен миниатюрный 9-контактный разъем: вилка с пружинными контактами и розетка с плоскими контактами. Разъемы имеют две модификации:

- для бета-портов (рис. 22.1, *в*) с широким ключом на кожухе;
- для «двуязычных» портов (рис. 22.1, *г*) с узким ключом.

На вилках имеются соответствующие ключи; к бета-порту можно подключить кабель только с бета-вилкой, к «двуязычному» — с любой вилкой 1394b. Кабель с «двуязычной» вилкой может на противоположном конце иметь вилку любого типа (4-, 6- или 9-контактную). Контакт 7 (SC, Status Contact) пока не используется (остается неподключенным).

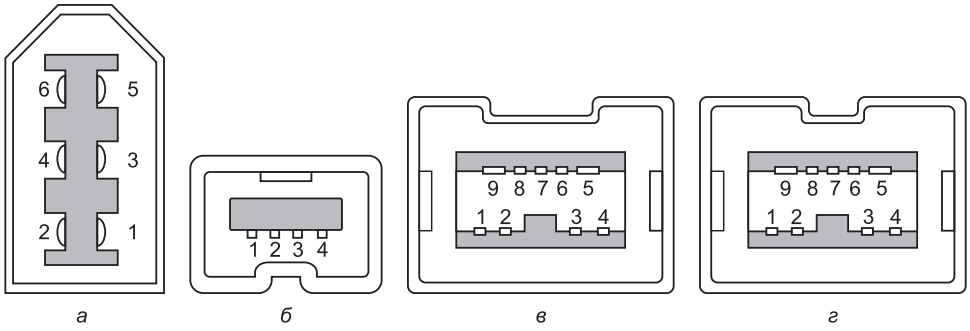


Рис. 22.1. Разъемы IEEE 1394 (размеры увеличены): *а* — 6-контактное гнездо; *б* — 4-контактное гнездо; *в* — 9-контактное гнездо Beta-порта; *г* — 9-контактное гнездо «двуязычного» порта

Стандартный 6-проводный кабель 1394 содержит две экранированные витые пары для передачи сигналов (TPA и TPB) и, дополнительно, два провода для питания устройств. Все эти провода помещаются в общий экран. Сигнальные пары проводников соединяются перекрестно. При необходимости могут использоваться адаптеры-переходники с разъемами разных типов. Соединения контактов вилок кабелей иллюстрирует табл. 22.1.

Таблица 22.1. Соединительные кабели FireWire

Разъем А				Разъем Б			
4-конт.	**6-конт.**	**9-конт.**	**Цепь**	**Цепь**	**9-конт.**	**6-конт.**	**4-конт.**
–	1	8	VP	VP	8	1	–
Экран	2	6	VG	VG	6	2	Экран
1	3	1	TPB–	TPA–	3	5	3
2	4	2	TPB+	TPA+	4	6	4
3	5	3	TPA–	TPB–	1	3	1
4	6	4	TPA+	TPB+	2	4	2
–	–	5	TPA(R)	TPB(R)	9	–	–
–	–	9	TPB(R)	TPA(R)	5	–	–
–	–	7	SC (не соединен)	SC (не соединен)	7	–	–
Экран	Экран	Экран	Экран	Экран	Экран	Экран	Экран

В IEEE 1394b для борьбы с помехами по земляным проводам предпринят ряд мер, возможных с применением 9-контактного разъема. Обратные (общие) провода — экранные оплетки пар TPA(R) и TPB(R) — выведены на отдельные контакты (5 и 9). Цепь TPB(R) соединяется со схемной землей и проводом VG непосредственно, а TPA(R) соединяется со схемной землей через параллельно соединенный резистор (1 МОм) и конденсатор (0,1 мкФ). Общий экран соединяется со схемной землей через такую же RC-цепочку. Схему соединения «земель» узлов иллюстрирует рис. 22.2.

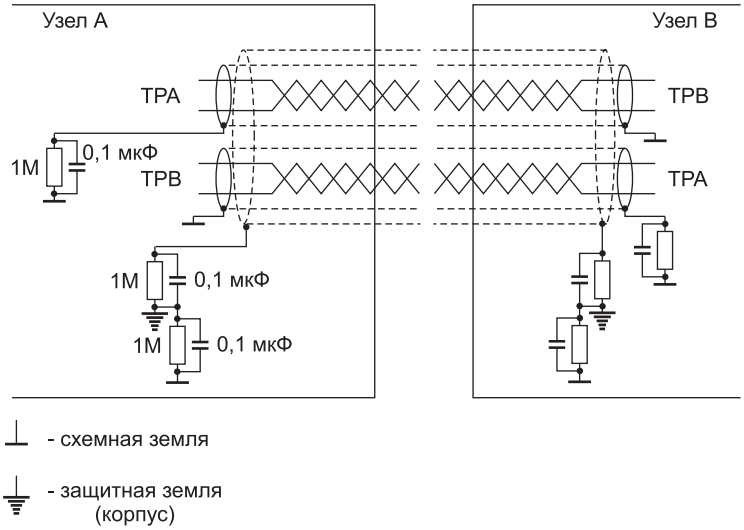


Рис. 22.2. Соединение схемных земель узлов в IEEE 1394b

Кабели и коннекторы бета-режима 1394b

В IEEEE 1394b для бета-режима введены новые варианты среды передачи (табл. 22.2): и соответствующие типы разъемов (рис. 22.3).

Таблица 22.2. Варианты среды передачи для бета-режима IEEE 1394b

Среда	Дальность	Рисунок разъема		S100	S200	S400	S800	S1600
UTP-5	100 м	22.3, *а*	+	–	–	–	–	
POF	50 м	22.3, *б*	+	+	–	–	–	
HPCF	100 м	22.3, *б*	+	+	–	–	–	
MMF	100 м	22.3, *в*	–	–	+	+	+	
STP	4,5 м	22.1, *в*, *г*	–	–	+	+	+	

- *UTP-5* (Unshielded Twisted Pair Cat 5) — неэкранированная витая пара категории 5 со стандартными коннекторами RJ-45, длина сегмента до 100 м. Парамет-

ры линии должны соответствовать стандарту на структурированные кабельные системы TIA-568 (или ISO 11801). Используются две пары проводов: контакты 1, 2 для TPB+ и TPB–; 7, 8 для TPA+ и TPA–. Порт должен поддерживать возможность перекрестного соединения: по умолчанию (*No_Crossover*) он передает сигнал по TPB и принимает по TPA. По управляющему запросу *Crossover* порт меняет назначение пар TPA и TPB. Питание через кабель UTP-5 не предусматривается. Кабель UTP подключается через разделительные трансформаторы, обеспечивающие гальваническую развязку, выдерживающую один из следующих тестов (на выбор):

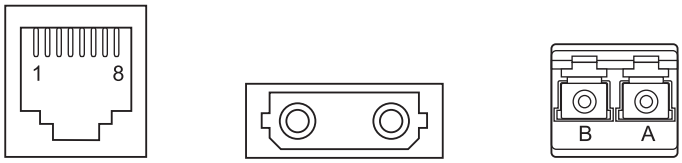
- 1500 В переменного тока 50–60 Гц в течение 1 мин;
- 2250 В постоянного тока в течение 1 мин;
- последовательность из десяти импульсов чередующейся полярности амплитудой 2400 В с временами фронта и спада 1,2/50 мс;

- *POF* и *HPCF* — оптоволоконные варианты с разъемами PN (дуплексные, диаметр стержня 2,5 мм, шаг 10,16 мм). Передатчики — светодиоды с длиной волны 650 нм.:
  - *POF* (Plastic Optical Fiber) — пара пластиковых волокон диаметром 1000 мкм со ступенчатым индексом преломления. Это самый дешевый вариант оптической связи на малых расстояниях;
  - *HPCF* (Hard Polymer Clad Fiber) — пара твердых полимерных волокон диаметром 225 мкм с градиентным индексом преломления;
- *MMF* (Multi Mode Fiber) — пара многомодовых стеклянных волокон 50/125 мкм, Параметры линии должны соответствовать стандарту на структурированные кабельные системы TIA-568 (или ISO 11801). Разъемы — малогабаритные дуплексные LC. Передатчики — коротковолновые лазерные диоды с длиной волны 830–860 нм, выходная мощность не менее –10 дБм[1]. Чувствительность приемников не хуже –16,5 дБм.;
- *STP* (Shielded Twisted Pair) — экранированные витые пары с более высокими характеристиками передачи, чем в прежних версиях IEEE 1394. Кабели для бета-режимов вплоть до S1600, обеспечивающие длину сегмента до 4,5 м, должны иметь витые пары калибра 25AWG; они имеют на оболочке маркировку «1394b 2PR/25AWG 2C/22AWG». Более тонкие пары (30AWG) обеспечивают длину до 2 м, они маркируются «1394b 2PR/30AWG 2C/26AWG». Для питания у них используются провода калибров 22AWG и 26AWG (больший номер калибра соответствует более тонким проводам).

Микросхемы PHY 1394b поддерживают любую среду передачи; соответствующие преобразования сигналов осуществляются в элементах PMD: RC-цепях для STP, трансформаторах для UTP и оптических трансиверах для POF, HPCF и MMF.

---

[1] В оптической связи единичное значение мощности установлено на уровне 1 мВт. Единицей измерения уровня мощности (дБм) является отношение средней мощности оптического излучения к единичному значению мощности, выраженное в децибелах. Таким образом, шкала мощности в единицах дБм является логарифмической, что более удобно, чем использование линейной шкалы в мВт. — *Примеч. ред.*



а *б* *в*

Рис. 22.3. Разъемы IEEE 1394b для бета-режимов: *а* — гнездо RJ-45 (для UTP); *б* —гнездо PN (для OF и HPCF); *в* —гнездо LC (для MMF)

Порт с электрическим интерфейсом в бета-режиме (для STP или UTP) несложно преобразовать в оптический. Для этого требуется оптический трансивер, поддерживающий требуемую скорость передачи, и несложные схемы преобразования уровней сигналов. При использовании трансиверов с низковольтным псевдоЭСЛ интерфейсом (LV PECL) эти схемы содержат всего 8 резисторов, задающих смещение уровней сигналов трансиверов, и 4 разделительных конденсатора.

Электрический интерфейс в DS-режиме

Электрический интерфейс использует комбинацию дифференциальных и линейных сигналов, передаваемых по двум парам сигнальных проводов. *Дифференциальные сигналы* используются для следующих целей:

- подачи сигнала сброса;
- арбитража;
- конфигурирования;
- передачи пакетов.

Линейные сигналы обеспечивают:

- обнаружение подключения/отключения устройств;
- сигнализацию скорости передачи данных;
- сигнализацию приостановки/возобновления работы (suspend/resume)

Схема приемопередатчиков порта IEEE 1394a приведена на рис. 22.4. В версии 1394–1995 отсутствовали элементы, выделенные серым цветом. Для каждой сигнальной пары (A и B) у порта имеются передатчики и приемники дифференциальных и линейных сигналов. При соединении двух узлов кабелем приемопередатчики соединяются перекрестно: пара A порта одного устройства соединяется с парой B другого устройства.

Большая часть интерфейсных функций выполняется с помощью дифференциальной передачи сигналов по витым парам A и B. Высокие скорости передачи предъявляют жесткие требования к электрическим параметрам кабелей, коннекторов, передатчиков и приемников. Для дифференциальной сигнализации обеспечивается волновое согласование портов с линией с помощью пар резисторов-терминаторов (55 Ом). Ограничение на длину кабельного сегмента (4,5 м) диктуется условиями передачи: на частоте 400 МГц затухание сигнала не должно превышать 5,8 дБ

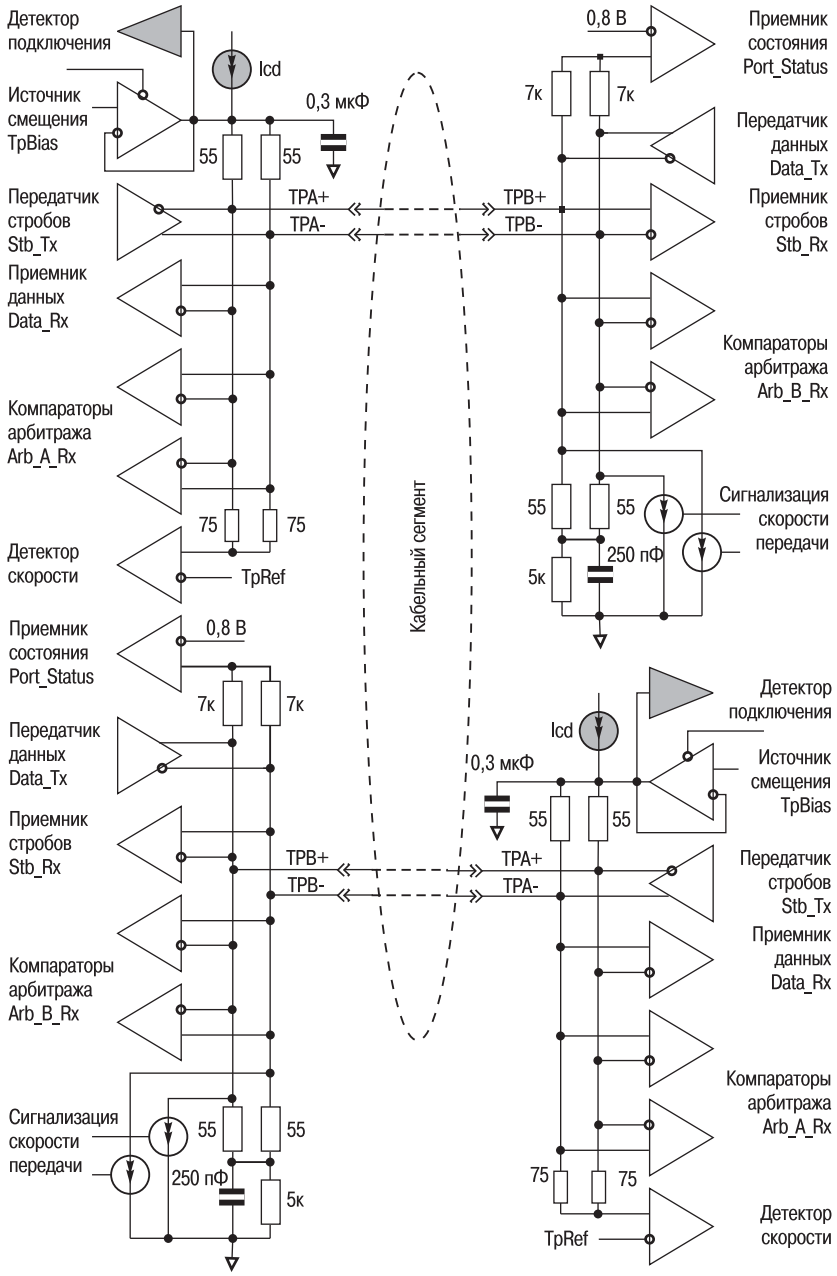


Рис. 22.4. Приемопередатчики порта IEEE 1394a

(почти двукратное снижение амплитуды). Дифференциальным способом передаются как высокочастотные сигналы передачи данных, так и сравнительно низкочастотные сигналы конфигурирования, арбитража и индикации скорости. Пара-

метры дифференциальных сигналов на выходе передатчика и входе приемника приведены в табл. 22.3.

Таблица 22.3. Амплитуда дифференциальных сигналов

Сигнал	Выход (мВ)	Вход S100 (мВ)	Вход S200 (мВ)	Вход S400 (мВ)
Данные	172–265	142–260	132–260	118–260
Арбитраж	172–265	173–260	171–262	168–265

Дифференциальный сигнал может находиться в одном из трех состояний:

- «1» — TPx + > TPx–;
- «0» — TPx + < TPx–;
- «Z» — TPx + ≈ TPx– (передатчик отключен или передатчики обоих портов пытаются передавать противоположные значения).

Эти три состояния воспринимаются и различаются компараторами арбитража, которые имеются на обоих парах порта. Передаются сигналы арбитража с помощью дифференциального передатчика. Сигналы арбитража имеют относительно большую длительность (доли микросекунды).

Распознавание подключения-отключения устройств и состояние порта

Каждый порт через резисторы-терминаторы (55 Ом) подает от источника смещения TpBias напряжение 1,6–2 В на линии пары A и измеряет среднее значение на линиях пары B (приемник состояния Port_Status). Если измеренное напряжение превышает 1 В, это означает, что на противоположном конце кабеля имеется подключенное устройство. Уровень напряжения ниже 0,6 В является признаком отключения устройства. Поскольку кабель соединяет линии A и B пары устройств перекрестно, оба устройства видят присутствие или отсутствие друг друга. Подключение партнера определяется сигналом Port_Status, вырабатываемым детектором состояния (линейным приемником пары B).

В 1394a введена возможность приостановки узла, которая подтверждается снятием напряжения смещения. Для узла 1394 это будет означать отключение устройства. В 1394a введена дополнительная схема обнаружения подключения/отключения: в приемопередатчик пары A введен источник маленького тока Icd (не более 76 мкА) и детектор отключения. Когда противоположный узел подключен, этот ток проходит в землю через нагрузку около 5 кОм (терминаторы на противоположном узле и заземляющий резистор), так что на входе детектора отключения напряжение не превышает 0,4 В. При отсоединении противоположного узла нагрузка снимается и напряжение на входе детектора поднимается выше 0,4 В. Эта схема обнаружения отключения включается только после согласования перехода в режим *Suspend*, когда снимается подача напряжения смещения. В состоянии *покоя шины* (*Bus Idle State*), когда активные порты двух узлов соединены друг с другом, на обоих парах A и B присутствует напряжение смещения (TpBias).

В 1394a введены дополнительные состояния порта:

- *Disabled* — работа порта запрещена, он не передает никаких сигналов (но смещение подает) и не воспринимает сигналы;
- *Suspended* — порт приостановлен, при этом он не подает напряжение смещения и включает детектор отключения.

Сигнализация арбитража

Каждый порт может по своим линиям передавать сигналы Arb_A(Tx) и Arb_B(Tx) и принимать сигналы Arb_A(Rx) и Arb_B(Rx), принимающие значения 0, 1 или Z. Сигнал, передаваемый узлом-1 по линии A, приходит на соседний узел-2 по линии B; при этом он может «сталкиваться» с сигналом, передаваемым узлом-2 по линии B. Для сигнала, передаваемого по линии B, действует симметричное правило. Таким образом, принимаемый сигнал является функцией от передаваемых сигналов двух передатчиков: Arb_A1(Rx) зависит от Arb_A1(Tx) и Arb_B2(Tx), Arb_B1(Rx) зависит от Arb_A2(Tx) и Arb_B1(Tx). Если один из передатчиков находится в состоянии Z, то приемник видит сигнал от другого передатчика без изменений. Если оба передатчика передают один и тот же сигнал, то приемник его так и воспринимает.

При декодировании сигналов арбитража действуют следующее *правило доминирования единицы*:

- если порт передает сигнал 0, а принимает Z (это значит, что соседний узел передает 1), то это состояние декодируется как «1»;
- если порт передает сигнал 1, а принимает Z (это значит, что соседний узел передает 0), то это состояние тоже декодируется как «1».

Сигналы арбитража используются для сброса шины, конфигурирования и собственно арбитража. Состояние передаваемых (Arb_A(Tx), Arb_B(Tx)) и принимаемых (Arb_A(Rx), Arb_B(Rx)) сигналов в различных фазах работы шины приведено в табл. 22.4. В качестве принимаемых значений здесь указаны декодированные состояния (с учетом посылаемых сигналов и правила доминирования единицы).

Таблица 22.4. Сигналы арбитража в различных фазах шины

Передаваемое состояние	Arb_A(Tx), Arb_B(Tx)	Принимаемое состояние	Arb_A(Rx), Arb_B(Rx)
Сброс шины			
Bus Reset	[1 1]	*Bus Reset*	[1 1]
Идентификация дерева			
Tx_Parent_Notify, поиск родителей	[0 Z]	*Rx_Parent_Notify*	[Z 0]
Tx_Child_Notify, подтверждение родительских прав	[1 Z]	*Rx_Parent_Handshake*	[0 1]
Снятие *Tx_Parent_Notify*	[Z Z]	*Rx_Child_Handshake*	[1 Z]
Оба узла посылают *Tx_Parent_Notify*	[0 Z]	*Rx_Root_Contention* — состязание за роль корня	[0 0]

Передаваемое состояние	Arb_A(Tx), Arb_B(Tx)	Принимаемое состояние	Arb_A(Rx), Arb_B(Rx)
Самоидентификация			
Tx_Self_ID_Grant, предоставление права самоидентификации	[Z 0]	*Rx_Self_ID_Grant*	[0 0]
Tx_Ident_Done, завершение самоидентификации	[1 Z]	*Rx_Ident_Done*	[Z 1]
Нормальный арбитраж			
Tx_Request, узел посылает запрос передачи через p-порт	[Z 0]	*Rx_Request*	[0 Z]
Tx_Grant, узел разрешает передачу через c-порт	[Z 0]	*Rx_Grant*	[0 0]
Узел снимает запрос	[Z Z]	*Rx_Request_Cancel* —узел видит снятие запроса	[Z 0]
Префикс данных *Tx_Data_Prefix*	[0 1]	*Rx_Data_Prefix*	[1 0]
Конец пакета данных *Tx_Data_End*	[1 0]	*Rx_Data_End*	[0 1]
Tx_Disable_Notify, сигнал на запрещение порта	[Z 1]	*Rx_Disable_Notify*	[1 Z]
Tx_Suspend, сигнал приостановки порта	[0 0]	*Rx_Suspend*	[0 0]

Сброс шины (Bus Reset) может быть сигнализирован в любой момент любым устройством. Сигнал сброса имеет приоритет над всеми сигналами (благодаря правилу доминирования единиц). Сигнал сброса, сгенерированный или обнаруженный узлом, распространяется на все его порты (кроме запрещенных). Длительность генерируемого сигнала сброса составляет 167 мкс, в 1394a введен и короткий сброс длительностью 1,4 мкс. По сигналу сброса узел инициализируется и переводит все свои порты в состояние покоя (*Bus Idle*). После этого начинается фаза идентификации дерева.

Во время фазы *идентификации дерева* (Tree Identification) с помощью сигналов арбитража «дети» ищут своих «родителей» и выстраивается дерево шины. Во время фазы *самоидентификации* (Self Identification) с помощью сигналов арбитража узлам поочередно предоставляется право передать пакет(ы) самоидентификации и сообщить о завершении передачи этих пакетов. Подробнее о процессах идентификации см. в главе 20. Во время нормальной работы сигналы арбитража используются для запроса и предоставления права на передачу пакетов.

Механизм арбитража

В фазе нормальной работы шины (после самоидентификации) механизм арбитража работает следующим образом:

- узел, который собирается отправить пакет, дожидается покоя шины (*Bus Idle*, [Z Z]), длящегося в течение определенного интервала (соответствующего приоритетности данного пакета), и посылает через свой p-порт сигнал запроса передачи *TX_REQUEST*;
- промежуточные узлы-ветки доводят этот запрос до корня, если этому не помешает запрос от другого узла, появившийся раньше;
- корневой узел принимает запрос и отвечает на него сигналом *TX_GRANT*;
- сигнал *TX_GRANT* доводится до источника запроса;
- источник запроса посылает во все свои порты сигнал префикса данных, после которого начинается передача пакета;
- все промежуточные узлы, получившие префикс данных, начинают транслировать его (а затем и собственно данные) во все свои порты.

Сигналы арбитража распространяются промежуточными узлами шины. Запрос передается «вверх» сигналом *TX_REQUEST* (через p-порт); он достигает c-порта соседнего узла сигналом *RX_REQUEST* («переворачиваясь» в кабеле). Сигнал запроса, обнаруженный на c-порте, узел обязан транслировать к корню, если к этому моменту он уже не транслирует запрос от другого порта или же не запрашивает передачу сам. Таким образом, запрос арбитража от какого-либо узла достигает корня.

Корневой узел посылает сигнал предоставления (*TX_GRANT*) на тот свой порт, запрос с которого обнаружен первым (или порту с меньшим номером, если запросы обнаружены одновременно). На остальные порты посылается сигнал префикса данных (*DATA_PREFIX*), который указывает на занятость шины.

Промежуточный узел-ветка, получивший «сверху» предоставление (сигнал *RX_GRANT*), транслирует его «вниз» (сигналом *TX_GRANT*) на тот порт, чей запрос был им принят; на остальные порты посылается сигнал префикса данных (*DATA_PREFIX*). Если узел сам является источником запроса, он на все свои c-порты передает сигнал префикса данных. Таким образом, сигнал предоставления достигает узла-источника запроса, награждая его правом передачи пакета. Узел, который во время передачи сигнала *TX_REQUEST* получает сверху префикс данных, должен прекратить подачу запроса и транслировать вниз префикс данных. Если этот узел сам является источником запроса, он «понимает», что в данном случае арбитраж им проигран и попытку получения доступа придется повторить позже.

Узел-победитель снимает сигнал запроса и устанавливает префикс данных. Получив префикс данных, промежуточный узел прекращает подачу сигнала *TX_GRANT* (вниз) и транслирует префикс данных на все остальные порты. Корень, являясь промежуточным узлом, также прекратит подачу сигнала *TX_GRANT* и продолжит трансляцию префикса данных на другие порты. Таким образом, все узлы увидят сигнал префикса данных (занятость шины), исходящий от победителя. Теперь победитель может начинать передачу пакета.

Передача данных

Данные передаются в дифференциальном виде по обеим парам проводов. При этом по линии A передаются стробы, а по линии B — последовательные данные. Прини-

маются данные по линии B, а стробы — по линии A. Синхронизация данных осуществляется с помощью стробов весьма своеобразным способом. Сами данные передаются в NRZ-коде — уровень сигнала во время битового интервала соответствует значению данного передаваемого бита (рис. 22.5). *Стробы* при передаче формируются из тактового сигнала — меандра с периодом в два битовых интервала — и самих данных. При приеме данные и стробы поступают на логический элемент *Исключающее ИЛИ* (функция XOR). Немного задержанный выходной сигнал этого элемента является сигналом битовой синхронизации, по фронтам и спадам которого «защелкиваются» биты последовательных данных. Для того чтобы все биты пакета были извлечены из декодера приемника, приходится добавлять дополнительные биты (Dribble bits): 1 для S100, 3 для S200 и 7 для S400. Эти биты при передаче вводит PHY, он же их исключает при приеме, так что LINK-уровень с ними дела не имеет. Значение дополнительных битов соответствует тому сигналу, в который должен завершать пакет: *Data_Prefix* или *Data_End* (см. ниже).

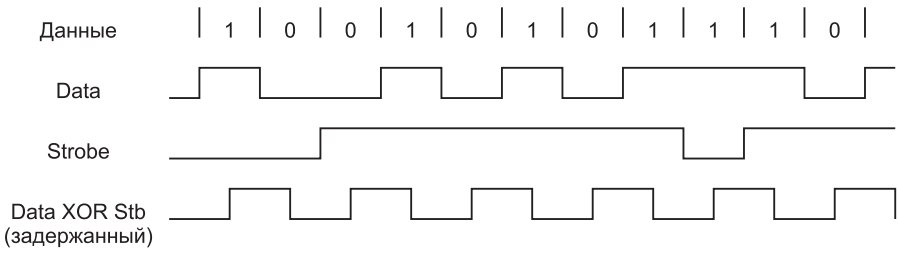


Рис. 22.5. Кодирование данных в DS-режиме

Передача данных начинается с *префикса данных* (*Data Prefix*) — состояния линий [0 1]. Префикс данных длится не менее 140 нс. Во время префикса данных передается и *сигнал идентификации скорости* (*Speed Signal*) — синфазное смещение уровней сигнала на паре B. Смещение вводится с помощью управляемых источников тока (в передатчике B) и измеряется компараторами (в приемнике A). Этот ток смещает среднее напряжение на линии относительно смещения TpBias. Величина тока и, соответственно, величина смещения на TPB кодируют скорость передачи пакета:

- S100 — нет тока смещения, среднее напряжение на TPB — не ниже 1,17 В;
- S200 — ток смещения 3,5 мА, среднее напряжение на TPB — 0,94–1,17 В;
- S400 — ток смещения 10 мА, среднее напряжение на TPB — 0,52–0,94 В.

Скорость передачи пакета заказывает приложение узла, запрашивающего передачу. При трансляции пакетов PHY будет посылать (транслировать) высокоскоростной пакет только на те порты, которые поддерживают данную скорость. На остальные порты на все время передачи пакета будет передаваться префикс данных, чтобы обозначить занятость шины.

Обычным признаком окончания пакета является сигнал *Data End* — состояния линий [1 0] длительностью 0,25 мкс. В соединенных транзакциях пакеты разделяются префиксом данных, который будет играть роль признака конца данных для предшествующего пакета.

На рис. 22.6. приведены временные диаграммы одной субакции. Здесь изображены сигналы на парах TPA, TPB порта узла, посылающего пакет субакции и принимающего пакет подтверждения. В нижней части рисунка показаны сигналы, передаваемые портом (Tx:) и принимаемые им (Rx:). Увеличенная амплитуда дифференциального сигнала на TPA в начале передачи префикса данных обусловлена столкновением сигнала префикса, посылаемого данным портом, с сигналом предоставления права на передачу, еще не снятым узлом-партнером. На рисунке изображена нормальная форма выполнения транзакции (с передачей одиночного пакета). В случае соединенной (concatenated) формы выполнения (например, передачи одним узлом нескольких пакетов самоидентификации) пакеты разделяются лишь префиксами данных, а признак конца (Data_End) вводится только после передачи последнего пакета.

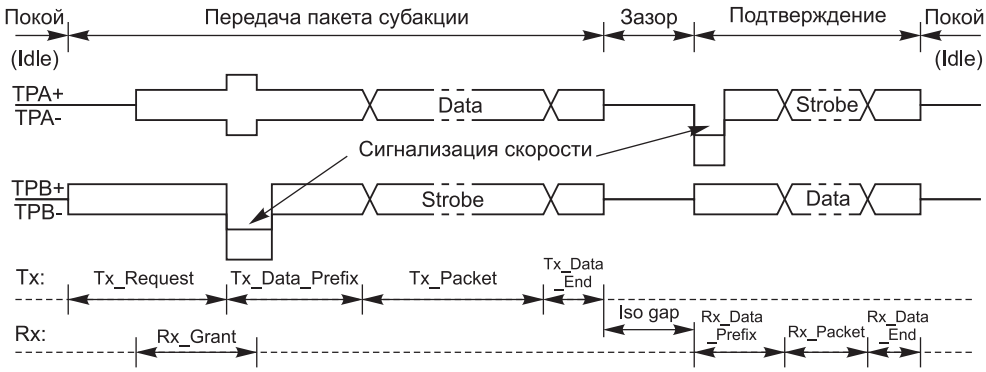


Рис. 22.6. Передача пакетов 1394 и 1394a

Интерфейс в бета-режиме IEEE 1394b

Интерфейс бета-режима построен с учетом возможности использования различных сред передачи сигналов по двум встречным однонаправленным линиям. Здесь исключена сложная дифференциально-линейная сигнализация с распознаванием различных уровней смещения и анализом состояния сигналов арбитража, являющихся результатом одновременной работы передатчиков соединенных устройств на одну и ту же сигнальную пару. Вся информация в бета-режиме передается с помощью 10-битных символов, передаваемых последовательно простым кодированием по методу NRZ. Чтобы избежать электромагнитных помех от шины, сосредоточенных в узких полосах спектра при передаче монотонных данных или служебных сигналов, передаваемые потоки скремблируются. Скремблированные данные проходят через кодер 8B/10B, который из байтов данных делает 10-битные символы. Это избыточное кодирование позволяет сформировать битовый поток, в котором ограничена длина повторяющихся битов (не более пяти нулей или единиц подряд) и среднее число нулей совпадает со средним числом единиц. Такой поток удобно передавать по различным средам передачи (отсутствует постоянная составляющая) и декодировать, синхронизируясь по перепадам сигнала, встречаю-

щимся не реже чем через 5 битовых интервалов. При приеме битовая последовательность подвергается обратным преобразованиям. Аналогичное кодирование-декодирование применяется в Ethernet, FDDI и других коммуникационных технологиях.

В структуре PHY узла 1394b появился сменный нижний уровень PMD (Physical Media Dependent), соответствующий физической среде передачи. Для коротких медных линий (STP) это дифференциальные приемопередатчики, для длинных (UTP) они содержат и разделительные трансформаторы. Для оптоволокна это оптические трансиверы с лазерным или светодиодным излучателем. «Двуязычный» порт должен иметь и традиционный набор дифференциальных и линейных приемопередатчиков, работающих только на традиционный кабель.

В бета-режиме в линию передаются символы, несущие полезную и служебную информацию:

- байты данных — заголовки и тела передаваемых пакетов;
- байты запросов, которые могут содержать:
  - асинхронные и изохронные запросы BOSS-арбитража, упакованные в один байт: 3 бита на код асинхронного запроса (*NONE_ODD*, *NONE_EVEN*, *CURRENT*, *NEXT_ODD*, *NEXT_EVEN*, *CYCLE_START_REQ*, *BORDER*), 3 бита на код изохронного запроса (*ISOCH_NONE*, *ISOCH_EVEN*, *ISOCH_ODD*, *ISOCH_CURRENT*) и два бита нулевые). Асинхронные и изохронные запросы независимы друг от друга;
  - запрос и фазу традиционного арбитража (*LEGACY_REQUEST*, *LEGACY_PHASE*, номер фазы кодируется двумя битами);
  - конфигурационные запросы: *TRAINING*, *DISABLE_NOTIFY*, *CHILD_NOTIFY-IDENT_DONE-PARENT_HANDSHAKE*, *OPERATION*, *STANDBY*, *SUSPEND*, *PARENT_NOTIFY*.
- Управляющие маркеры (4-битные коды): *BUS_RESET*, *CYCLE_START_EVEN*, *CYCLE_START_ODD*, *ASYNC_EVEN*, *ASYNC_ODD*, *ATTACH_REQUEST-ARB_CONTEXT*, *DATA_PREFIX*, *DATA_END*, *DATA_NULL*, *GRANT*, *GRANT_ISOCH*, *SPEEDa*, *SPEEDb*, *SPEEDc*.

Байты данных и запросов скремблируются и кодируются в 10-битные символы единообразно, их можно различать только по контексту. Управляющие маркеры кодируются так, что они в 10-битных символах явно отличимы от данных и запросов. Избыточное кодирование является помехозащищенным — недопустимые комбинации считаются ошибочными. Для обеспечения побайтной синхронизации используются специальные символы, не совпадающие с символами, представляющими байты передаваемой информации.

Традиционные порты с DS-сигнализацией могут передавать и принимать пакеты на любой из доступных им скоростей, поскольку битовая синхронизация автоматически обеспечивается функцией XOR от состояния данных и строба. В бета-режиме передача и прием возможны только на одной скорости, согласованной портами-партнерами. На эту скорость настраивается система фазовой автоподстройки частоты (PLL) приемника. При этом *эффективная скорость* передачи информации может совпадать с *согласованной скоростью* порта или же составлять от

нее 1/2, 1/4, 1/8, ..., 1/32 части. Последний случай относятся к порту, работающему на S3200 и передающему пакет на скорости S100.

Пакеты могут передаваться как в традиционном (Legacy) формате, так и в бета-формате. Пакеты бета-формата передаются только между бета-узлами, в традиционные фрагменты гибридной шины они не транслируются. Передачи на скоростях S800 и выше возможны только в бета-формате. Общая структура формата передачи приведена на рис. 22.7. Для сохранения возможности обмена с узлами, поддерживающими различные скорости и форматы, применяются специальные меры:

- управляющие маркеры размножаются — передаются столько раз, во сколько эффективная скорость меньше скорости работы порта;
- вводится префикс пакета, в котором передается код скорости и определяется формат пакета;
- если эффективная скорость ниже скорости порта, между символами содержимого пакета вводятся символы-заполнители.

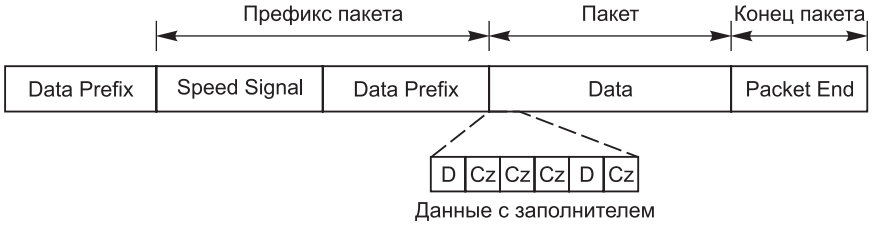


Рис. 22.7. Передача пакетов 1394b

Префикс пакета начинается с кода скорости, за которым следует один или несколько символов-маркеров начала данных *DATA_PREFIX*. Код скорости определяет относительную скорость передачи (отношение эффективной и согласованной скорости) и формат. Отсутствие кода скорости означает передачу традиционного пакета на скорости S100. Кодирование скорости и формата приведено в табл. 22.5. Здесь буквой B обозначен символ *SPEEDb*, буквой C обозначен символ *SPEEDc*, C_m означает *m* повторов символа *SPEEDc*. Букве X соответствует *SPEEDa* для традиционного пакета и *SPEEDb* для бета-пакета, по этим символам и определяется формат пакета. В таблице приведено и время Ts, отводимое для сигнализации скорости.

Содержимое пакета — символы, кодирующие передаваемые байты, — «разбавляется» управляющими маркерами *SPEEDc*. Их количество определяется отношением скоростей: 1 для 1/2, 3 для 1/4, 7 для 1/8, 15 для 1/16 и 31 для 1/32. На рис. 22.7 изображены символы-заполнители Cz для случая, когда скорость порта в 4 раза выше эффективной скорости передачи пакета. По пути к узлу-получателю пакет может проходить через сегменты с различными согласованными скоростями, в результате количество символов-заполнителей может меняться (сокращаться при прохождении через «медленные» сегменты или увеличиваться при прохождении более скоростных). Понятно, что пройти через сегмент, для которого согласованная скорость ниже эффективной, пакет не сможет.

Признаком завершения пакета может быть любой из символов-маркеров (возможно, повторяющийся многократно) *ARB_CONTEXT*, *DATA_END*, *DATA_PREFIX*, *DATA_NULL*, *GRANT*, *GRANT_ISOCH* или *LEGACY_PHASE*. Традиционные пакеты могут завершаться и одним из символов *DATA_END/GRANT/GRANT_ISOCH* (их цепочкой), за которым следует *LEGACY_PHASE* (один или несколько).

Таблица 22.5. Кодирование скорости и формата передач

Скорость порта	Эффективная скорость					
	S100	**S200**	**S400**	**S800**	**S1600**	**S3200**
S100	X					
S200	CX	X				
S400	CCXC	CX	X			
S800	$CCCXC_4$	CCXC	CX	B		
S1600	$CCCCXC_{11}$	$CCCXC_4$	CCXC	CB	B	
S3200	$CCCCCXC_{26}$	$CCCCXC_{11}$	$CCCXC_4$	CCBC	CB	B
T_S, нс	80	40	20	10	5	2,5

Обнаружение подключения и согласование скорости

В бета-режиме обнаружение подключения и согласование скорости портов происходит без использования сигнализации постоянным током, как это было в IEEE 1394 и 1394a. Сигнализация подключения осуществляется тональными сигналами, передаваемыми по линии TPA и прослушиваемыми с линии TPA. Двуязычные узлы имеют оба механизма обнаружения подключения: тональный и с сигнализацией постоянным током. Алгоритм работы обнаружения подключения работает таким образом, чтобы по возможности установить соединение в бета-режиме. Если это не удается (партнер не посылает тональный сигнал, а упорно подает смещение), то включается традиционная схема обнаружения подключения и согласование скорости.

Для согласования скорости в бета-режиме используются серии тональных посылок. Эти серии отличимы от одиночных посылок обнаружения подключения, а также от непрерывных тональных сигналов пробуждения (resume), хотя в них используется одна и та же частота (48–61 МГц). Диаграмма посылок приведена на рис. 22.8. Каждый тональный сигнал представляет один бит (есть тон — бит единичный). Узел передает посылки, в которых закодирована максимальная поддерживаемая им скорость, одновременно прослушивая сигнал от партнера. Обнаружив посылку от партнера, он посылает посылку с битом `ack` = 1 (до того было `ack` = 0), в которой теперь указывает согласованную скорость (минимальную из передаваемой и принимаемой посылок). Приняв посылку с единичным битом `ack`, порт прекращает посылки согласования, и соединение устанавливается на общедоступной скорости. В посылке согласования скоростей два бита — `P` (`PIL_Capable`) и `F` (`FOP_Capable`) относятся к свойствам интерфейса порта (см. главу 23).

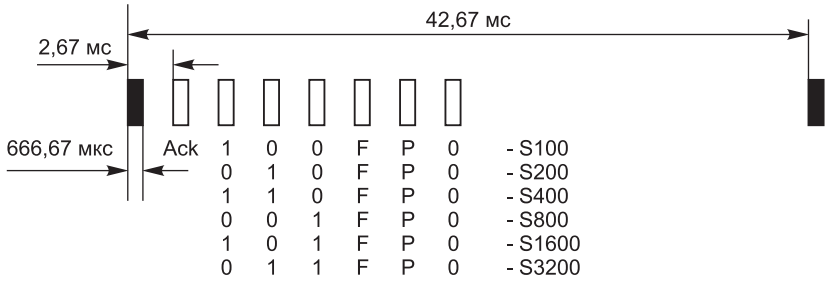


Рис. 22.8. Посылки согласования скоростей

После согласования скорости порты должны взаимно синхронизироваться: установить битовую синхронизацию, определить границы символов и синхронизировать друг с другом свои скремблеры и дескремблеры. Для этого выполняется *процедура тренировки* (training) — порты обмениваются специальными тренировочными последовательностями символов.

Установленная синхронизация в процессе работы может быть потеряна. Порт отслеживает потерю синхронизации: у него имеется счетчик неверных принятых символов. Прием каждого неверного символа инкрементирует счетчик, прием подряд двух верных символов декрементирует счетчик (в пределе до нуля). Если счетчик успевает досчитать до четырех, порт считает, что синхронизация потеряна и ее требуется восстановить с помощью тренировочной последовательности. Тренировку может инициировать и его партнер, послав запрос *TRAINING*.

Предотвращение петлевых соединений

Петлевые соединения узлов недопустимы и запрещены в шине IEEE 1394; за их отсутствие отвечает пользователь, соединяющий устройства. Петлевые соединения приводят к «зависаниям» конфигурирования и арбитража, а также к бесконечно повторяющимся сбросам. В дополнении 1394b введен механизм автоматического исключения петлевых соединений, правда, пригодный только для узлов с портами, работающими в бета-режиме.

При соединении устройств их порты согласуют скорости и выполняют тренировку, после чего переходят в состояние *Untested* (не тестирован). Для каждого нетестированного порта узел должен проверить, не приведет ли разрешение его работы к петле в шине. Если тест дает отрицательный результат (петли не будет), то порт переводится в активное состояние (*Active*); если петля будет, то порт переводится в запрещенное состояние (*Loop-disabled*). Проверка на петлевые соединения производится посылкой тестового пакета *LTP* (Loop-Test Packet) в тестируемый порт и сравнением символов, принимаемых с остальных портов, с тестовым символом, передаваемым в *LTP*. Если принимаемые символы не совпадают, значит, петли нет и можно разрешить работу порта. Если символ совпадает, то делается попытка тестирования с другим значением символа (это может быть случайным совпадением). Если совпадение происходит на нескольких разных символах, это является признаком петли и работу порта разрешать нельзя. Для того чтобы тес-

тировать свои порты, узел должен получить управление шиной, выиграв арбитраж.

Состояние Standby

В новом энергосберегающем состоянии порта и узла — *Standby*, введенном в 1394b, PHY не обеспечивает взаимодействия своего узла с шиной. В режим *Standby* порт узла переходит по получении *пакета Standby* с идентификатором данного узла. Этот пакет посылает узел, управляющий энергопотреблением. Переход произойдет только при соблюдении ряда условий:

- порт является единственным подключенным портом данного узла и работает в бета-режиме;
- узел не является корневым;
- LINK-уровень узла поддерживает бета-режим;
- порту не запрещен переход в состояние *Standby* (флаг `enable_standby` = 1).

Узел, порт которого перешел в состояние *Standby*, получает статус «племянника» (nephew node). Этот узел теряет представление о событиях, происходящих на шине (даже таких, как сброс). Этот узел и сам не может стать инициатором сброса (запись в его регистр `ibr` игнорируется). «Интересы племянника» на шине представляет его «дядя» (uncle node) — узел-партнер его по связи. Порт «дяди», обращенный к «племяннику», также переводится в состояние *Standby* (по приему *символа STANDBY*, который посылает «племянник»). Если теперь произойдет сброс на шине, то «дядя» будет посылать пакеты самоидентификации не только за себя, но и за своего «племянника». Переход порта узла-«племянника» в нормальное рабочее состояние происходит либо по запросу шины от своего LINK-уровня, либо по пробуждающему тональному сигналу, принимаемому по линии от «дяди». При этом нормальную работу узел может начинать только после получения от «дяди» *пакета восстановления Restore*, содержащего идентификатор узла-племянника (он мог поменяться из-за сброса на шине), значение зазора (`gap_count`), указание фазы шины и уведомление о происшедшем сбросе (если он был). Этот пакет LINK-уровню не передается; в ответ на него порт должен выполнить действия в соответствии с функциями BOSS-арбитража (как на последний пакет в субакции). Многопортовый узел, который вернулся из состояния *Standby* по событию подключения к его другому порту, должен в этом случае послать маркер *BUS_RESET*, чтобы шина отработала подключение новых узлов.

Для того чтобы представлять интересы «племянников», узел должен сохранять пакеты самоидентификации, принимаемые им по каждому из портов. Тогда, получив символ *STANDBY* от какого-либо своего порта, он сможет представлять интересы своего появившегося «племянника». «Дядя» должен восстановить нормальную работу «племянника» по получении пакета *Restore* со своим идентификатором, в котором указан номер порта восстанавливаемого «племянника». По этому событию «дядя» посылает в данный порт тональный сигнал восстановления. Если восстановление инициируется «племянником», то он посылает тональный сигнал восстановления. Сгенерировав или получив тональный сигнал, «дядя» и «племянник»

устанавливают взаимную синхронизацию своих портов. Затем «дядя», запросив и выиграв арбитраж на шине, посылает в порт «племянника» пакет восстановления (в остальные порты и в свой LINK он посылает нуль-пакеты). Пакет восстановления завершается символом *DATA_END*, по которому «племянник» становится «боссом» и переводит свой порт в активное состояние.

Интерфейс для кросс-шины (Backplane)

Физический интерфейс для использования в кросс-шине имеет ряд особенностей, отличающих его от кабельного варианта. Эти особенности касаются только PHY-уровня; на работе LINK-уровня и вышестоящих они не сказываются. Интерфейс отличается следующими моментами.

Вместо двух дифференциальных пар (TPA и TPB), соединяющих порты узлов попарно, используются два сигнальных провода STRB и DATA, к которым подключены все соединяемые порты 1394 на шасси. Уровни сигналов на этих линиях стандартом не оговариваются. Для работы шины существенно лишь, чтобы в случае столкновения передач логического «0» и «1» состояние линии воспринималось бы всеми узлами как «1». То есть шина должна обеспечивать логическую функцию «*Проводное ИЛИ*». В случае ТТЛ-интерфейса шины логической единице соответствует низкий уровень напряжения и передатчиками являются элементы с открытым коллектором. Приемопередатчики узлов должны быть способными одновременно и передавать, и принимать сигнал.

Здесь также используется DS-кодирование, но с более низкими скоростями: формальной скорости S100 соответствует скорость около 50 Мбит/с, S200 — 100 Мбит/с.

Механизм арбитража для такой физической шины изменен. Для получения доступа к шине устройство должно послать 10-битную последовательность арбитража. Последовательность арбитража передается с использованием DS-кодирования, причем всегда на низкой скорости (49,152 Мбит/с). Последовательность начинается с 4-битного кода приоритета, за которым следует 6-битный арбитражный номер узла (его PHY_ID). Посылая эту последовательность, узел следит за состоянием линии (принимает эту последовательность). Как только узел видит, что очередной принимаемый бит отличается от передаваемого, он понимает, что арбитраж в данной попытке проигран. Последовательность арбитража узел имеет право посылать, только дождавшись покоя шины. При этом последовательность одновременно могут начать посылать несколько узлов. Выиграет арбитраж тот узел, который посылает последовательность с самым большим значением кода приоритета, а из узлов с одинаковым приоритетом — тот, у которого самый большой номер. Исходя из этого код приоритета 0000 используется для запроса справедливого арбитража, код 1111 — для запроса передачи пакета мастером циклов.

Межпакетные зазоры в кросс-шине значительно сокращены; здесь они определяются в арбитражных тактах (частоты 49,152 МГц):

- зазор пакета подтверждения *Acknowledge Gap* — 4 такта на обнаружение (около 183 нс), на шину пакет подтверждения выдается после 8 тактов покоя;

- зазор между субакциями (*Subaction Gap*) — 16 тактов на обнаружение, пакет передается после 20 тактов покоя (около 410 нс);
- зазор сброса арбитража (*Arbitration Reset Gap*) — 28 тактов на обнаружение, шину можно занимать после 32 тактов покоя (около 651 нс).

Сигнал сброса по кросс-шине не передается. Подключение/отключение узлов на ходу допускается, но механизм автоконфигурирований (самоидентификации узлов) не используется — физические идентификаторы узлам назначаются какими-либо сторонними способами.

Кросс-шина 1394 используется как дополнительная шина в составе параллельных шин VME64, FutureBus, GTLP. Здесь ее роль является вспомогательной (как SMBus в PCI). Несмотря на низкую (по отношению к кабельному варианту) скорость, последовательная шина остается высокопроизводительным каналом связи, поддерживающим все передачи 1394.

Трансляция сигналов (функции повторителя)

Соединение более двух узлов в шину IEEE 1394 возможно только при использовании многопортовых узлов. Многопортовость реализуется в PHY-уровне узла, который для этого должен иметь в своем составе несколько интерфейсных портов и повторитель. *Повторитель* (Repeater) служит для передачи пакетов между портами и кодером-декодером своего узла:

- пакет, исходящий с другого узла и не предназначенный для данного узла, транслируется с порта, на который он пришел, во все остальные активные порты, для которых согласованная скорость достаточна для передачи данного пакета;
- пакет, исходящий с другого узла и предназначенный данному узлу, транслируется в другие порты (с учетом скорости), а также декодируется и передается LINK-уровню (если он поддерживает эту скорость);
- пакет, исходящий из данного узла, посылается во все порты (с учетом скорости).

Повторитель узла выполняет ресинхронизацию данных по своему тактовому генератору. Поскольку тактовая частота узлов не может совпадать абсолютно точно, ресинхронизация требует применения эластичного буфера (с FIFO-организацией). Принимаемая битовая последовательность помещается в буфер на частоте входного сигнала. Биты на передачу начинают выбираться из FIFO-буфера после того, как там наберется некоторое начальное количество бит. Это вводит неизбежную задержку трансляции данных повторителем. В процессе трансляции число бит в FIFO может колебаться: если частота вывода (определяется частотой данного узла) чуть ниже частоты принятого сигнала, то в буфере биты будут накапливаться. Если частота вывода выше, то буфер будет опустошаться. Переопустошение буфера, естественно, недопустимо — оно приведет к ошибке данных. Начальный

порог заполнения выбирается исходя из максимально допустимого расхождения частот и максимальной длины передаваемого пакета.

Питание от шины

Для питания узлов постоянным током в кабелях IEEE 1394 предусмотрена отдельная пара проводов — VG (общий провод, GND) и VP (положительный полюс питания) с напряжением 8–40 В при токе до 1,5 А. В 1394a диапазон напряжений сужен до 8–33 В. Узлы могут быть источниками, потребителями питания или не пользоваться питанием от шины; их отношение к питанию сообщается в поле pwr пакета самоидентификации (см. главу 23). В узлах-повторителях линии питания всех портов объединены. Узлы-источники питания подают напряжения на линию питания через ограничитель тока и диод, так что мощность, подаваемая от нескольких узлов, суммируется. При подключении узел сначала может потреблять от шины не более 1 Вт (1394a — 3 Вт), при этом обязательно должен быть включен уровень PHY. Уровни LINK и выше, как и прикладная часть устройства, могут потреблять дополнительную мощность, заявленную в поле pwr. Однако эти уровни будут включаться только по команде от диспетчера шины или диспетчера изохронных ресурсов. Каждый узел контролирует питание от шины: при напряжении выше 7,5 В он устанавливает бит PS (power status), сообщаемый в пакете самоидентификации. При падении уровня напряжения ниже 7,5 В физический уровень должен обнулить бит PS и уведомить об этом событии управляющее ПО.

Понятие *классов питания* (Power Class) относится к питанию узлов от кабельной шины IEEE 1394. Класс питания узла сообщается им в поле pwr пакета самоидентификации. В соответствии с классом питания возможны следующие конфигурации узла:

- *узел с автономным питанием* (Self-Power), для которого возможно несколько вариантов:
  - однопортовый узел не питается от шины и не подает питание сам (pwr = 0);
  - многопортовый узел не питается от шины, не транслирует питание с других портов и не подает питание сам (pwr = 0);
  - многопортовый узел не питается от шины, но транслирует питание с других портов (pwr = 4);
- *поставщик питания* (Power Provider, pwr = 1, 2 или 3) — узел питается самостоятельно и подает на шину питание с указанием минимальной мощности;
- *альтернативный поставщик питания* (Alternate Power Provider, pwr = 4) — узел может питаться от шины, а также может поставлять питание;
- *потребитель* (Power Consumer, pwr = 5, 6 или 7) — узел (только однопортовый) питается от шины. До завершения конфигурирования мощность потребляет только PHY (не более 1 или 3 Вт). Верхние уровни (LINK и выше), требующие дополнительной мощности, узел включает только по команде от диспетчера.

Характеристики классов питания приведены в табл. 22.6. В 1394a и 1394b параметры потребления несколько отличаются от первоначальных определений 1394–1995 и зарезервирован класс 5. Для класса 4 поставляемая мощность указывается в памяти конфигурации узла.

Таблица 22.6. Классы питания

Клас (pwr)	Поставляет, Вт	Потребляет, Вт (1394–1995)		Потребляет, Вт (1394a, b)	
		PHY	LINK	PHY	LINK
0	0	0	0	0	0
1	15	0	0	0	0
2	30	0	0	0	0
3	45	0	0	0	0
4	0[1]	1	0	3	0
5	0	1	2	Резерв	Резерв
6	0	1	5	3	3
7	0	1	9	3	7

[1] Для класса 4 поставляемая мощность указывается в памяти конфигурации узла.

Каждой конфигурации соответствует своя схема соединения линий питания портов, собственного PHY и внутреннего (батарейного или сетевого) источника питания. Обобщенная схема питания узла приведена на рис. 22.9, для каждой из вышеприведенных конфигураций на схеме будут отсутствовать те или иные элементы. Общие идеи следующие:

- для многопортового узла следует предусмотреть питание PHY-уровня до последней возможности — это обеспечивает целостность шины (отключение питания PHY сегментирует шину, поскольку узел не сможет транслировать сигналы и трафик). Питание на PHY может подаваться и от кабеля, и от внутреннего источника (через диоды);
- поставщик питания должен подавать питание на каждый порт через индивидуальные разделительные диоды и ограничители тока (рис. 22.9, *а*). При этом каждый порт представляет отдельный домен питания (Power Domain);
- альтернативный поставщик, обеспечивающий напряжение ниже 20 В, может и не обеспечивать разделения доменов (но индивидуальные ограничители тока обязательны);
- узел с автономным питанием, транслирующий питание, при числе портов более двух должен иметь ограничители тока на каждый порт.

На рисунке в качестве ограничителей тока условно изображены плавкие предохранители. Реально, конечно же, используются или электронные ограничители, или самовосстанавливающиеся полупроводниковые предохранители.

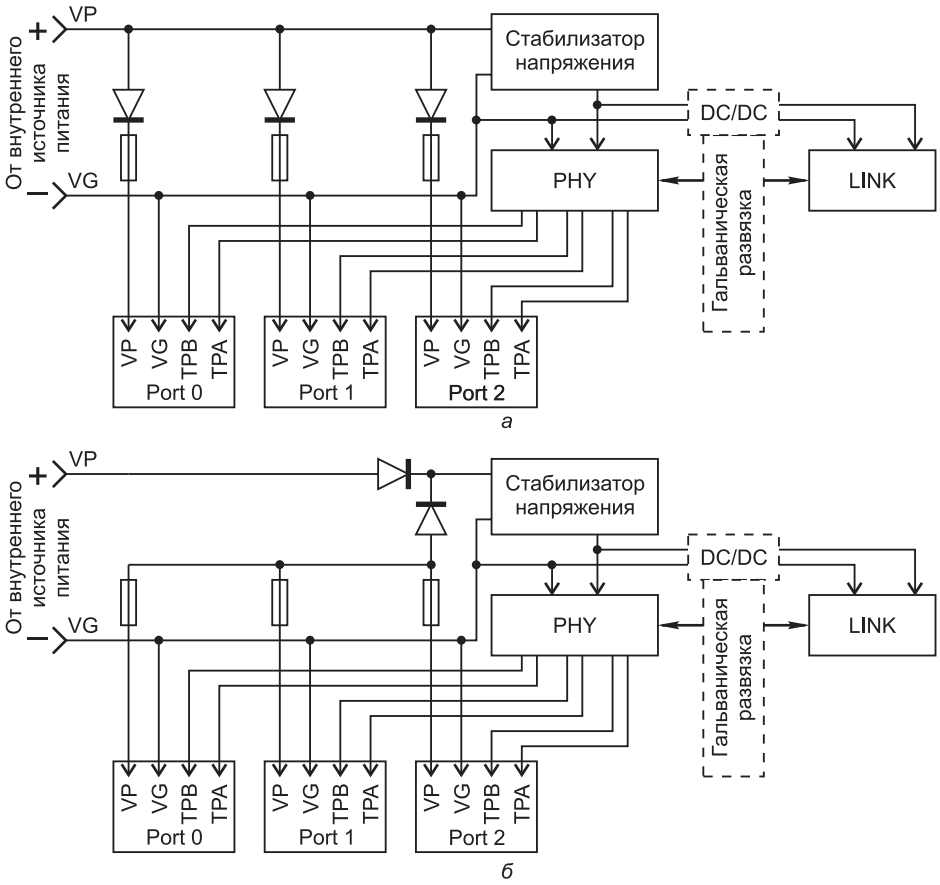


Рис. 22.9. Схема подачи питания от шины: *а* — узел-поставщик питания; *б* — узел-потребитель питания

Гальваническая развязка

Возможность гальванической развязки узлов шины IEEE 1394 позволяет решить ряд проблем, связанных с объединением «схемных земель» соединяемых устройств. В аудиотехнике из-за наводок в контуре заземления возникают помехи, неприемлемые для работы высококачественной аппаратуры, на которую и ориентирована шина.

В первых спецификациях, в которых используется только DS-сигнализация, гальваническая развязка возможна только в интерфейсе PHY-LINK. Этот параллельный интерфейс содержит значительное число сигнальных цепей, по которым передаются сигналы с уровнями логики КМОП или ТТЛ. Большая часть этих сигналов имеет весьма высокую частоту переключения (около 50 МГц), что не позво-

ляет применить дешевую оптронную развязку[1]. Оптроны удобно применять только для медленных сигналов LinkOn и LPS. Кроме того, ряд сигнальных цепей используется для двунаправленной передачи, что осложняет схему развязки. Самый простой и дешевый способ развязки — введение разделительных конденсаторов. Однако при этом развязка осуществляется только по постоянному току (типичное напряжение развязки до 60 В); высокочастотные помехи, вызывающие пульсации на общем проводе, будут воздействовать на интерфейс. Полную развязку обеспечивают только импульсные трансформаторы (1:1), устанавливаемые в каждой цепи; при этом достижимо напряжение изоляции развязки до 500 В. Однако это слишком дорогой и громоздкий вариант, который используется редко. И конденсаторы, и трансформаторы фактически выполняют дифференцирование сигналов (пере-

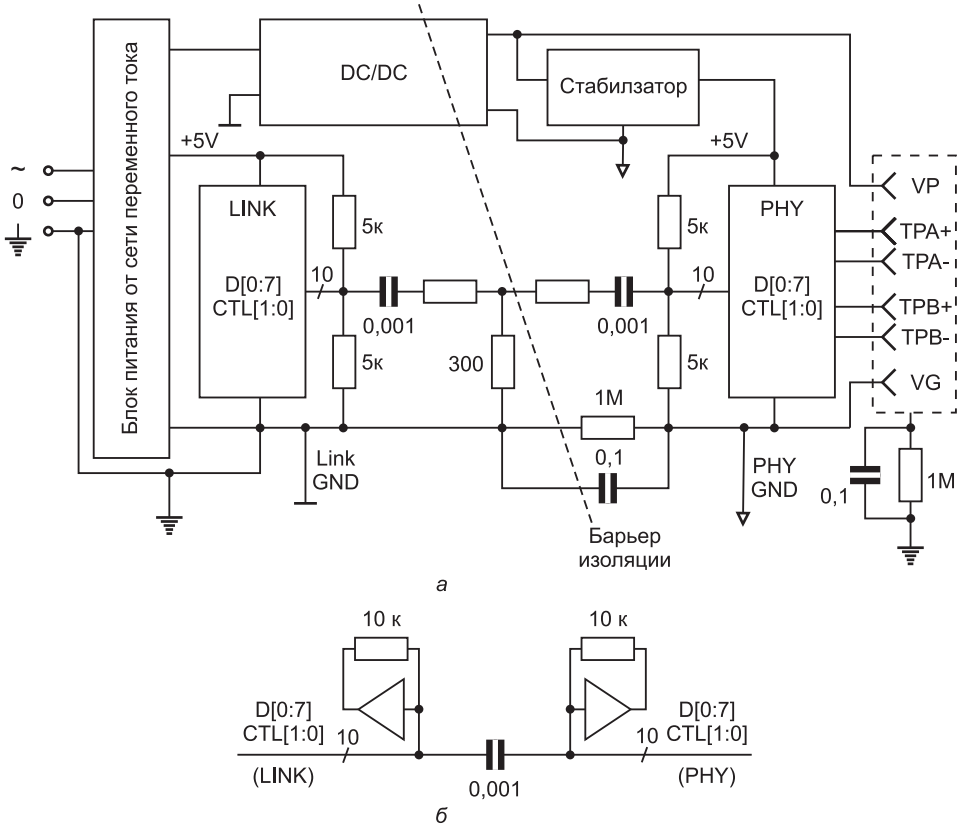


Рис. 22.10. Гальваническая развязка цепей интерфейса PHY-LINK: *а* — схема развязки по IEEE 1394; *б* — развязка сигнальных цепей по IEEE 1394b

[1] Обычные оптроны слишком инерционны, быстродействующие оптические приемники, применяемые, например, в сетях передачи данных, слишком дороги.

дают только перепады). Специально для этого и LINK, и PHY имеют конфигурирующие сигналы Direct, по которым интерфейсные цепи настраиваются на прием непосредственных сигналов (без развязки) или дифференцированных (с развязкой).

В приложении стандарта IEEE 1394 и в основном тексте IEEE 1394a приводится довольно громоздкая схема конденсаторной развязки (рис. 22.10, *а*), содержащая большое число резисторов, «подтягивающих» уровни сигналов. В IEEE 1394b приведена более элегантная схема установки привязки состояний входов, использующая КМОП-вентили (повторители) с высокоомной обратной связью (рис. 22.10, *б*). На рисунках изображены цепи развязки для одной из десяти сигнальных линий.

Для портов, работающих в бета-режиме (IEEE 1394b), полная гальваническая развязка стала возможной в цепях сигналов внешнего электрического интерфейса. Параметры этой развязки приводятся в начале данной главы. Использование оптоволоконной связи позволяет обеспечить бескомпромиссную развязку с любым требуемым напряжением изоляции.

Гальваническая развязка сигнальных цепей подразумевает и развязку для кабельного питания. Эта развязка осуществляется с помощью импульсных преобразователей напряжения (DC/DC Converter).

ГЛАВА 23

Взаимодействие с физическим уровнем шины IEEE 1394

Физический уровень (PHY) обеспечивает LINK-уровню возможность обращения к шине для посылки и приема пакетов и исполнения некоторых служебных действий. Физический уровень имеет свои регистры, служащие для управления выполнением функций и определения состояния шины и узлов. К этим регистрам имеется возможность локального доступа — приложение узла через LINK-уровень может общаться с регистрами PHY. Содержимое некоторых регистров может изменяться по приему пакетов физического конфигурирования. В 1394a появилась возможность удаленного обращения к регистрам чужого узла на данной шине. Физический уровень не только обеспечивает передачу и прием пакетов для LINK-уровня, но и сам является источником и получателем служебных пакетов.

Интерфейс с канальным уровнем

Физический и канальный уровни (PHY и LINK) в стандарте IEEE 1394 отделимы друг от друга. Это обеспечивает модульность построения устройств и возможность гальванической развязки основной (прикладной) части устройства от кабельного интерфейса. В устройствах физический и канальный уровни могут реализоваться раздельными микросхемами, причем даже разных производителей. Интерфейс между ними был описан в приложении к стандарту 1394, но в рекомендательной (необязательной) форме. В 1394a этот интерфейс (с небольшими изменениями) объявлен стандартом. Связь физического и канального уровней обеспечивается интерфейсом PHY-LINK. По этому интерфейсу канальный уровень получает сервисы доступа к шине для передачи и приема пакетов данных, а также доступ к внутренним регистрам PHY.

В IEEE 1394b в связи с особенностями бета-режима подход к организации интерфейса связи с физическими портами изменился. Здесь появился новый промежуточный интерфейс PIL-FOP, который позволяет соединять микросхему, в кото-

рой LINK интегрирован с однопортовым PHY, с многопортовым разветвителем. Таким образом, в 1394b имеются два варианта интерфейса:

- *параллельный интерфейс B PHY-LINK*, являющийся развитием своего традиционного предшественника и обеспечивающий работу с портами на скоростях от S100 до S800 как в традиционном, так и в бета-режиме;
- *последовательный интерфейс PIL-FOP*, по сигналам соответствующий электрическому интерфейсу порта в бета-режиме, подключающий многопортовый PHY-разветвитель (называемый FOP) к однопортовому PHY, интегрированному с LINK-уровнем (эта комбинация называется PIL). Последовательный интерфейс поддерживает все режимы и скорости (от S100 до S3200).

Интерфейс PHY-LINK 1394 и 1394a

Интерфейс PHY–LINK изображен на рис. 23.1, назначение сигналов раскрыто в табл. 23.1. Интерфейс содержит небольшое количество сигналов, и именно для этих сигналов и возможно введение гальванической развязки (см. главу 22).

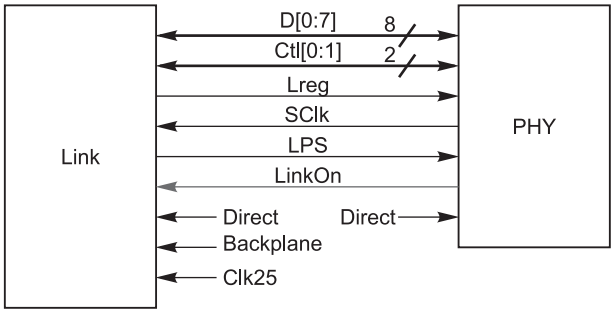


Рис. 23.1. Интерфейс PHY-LINK 1394 и 1394a

На рисунке приведены сигналы для обоих вариантов шины — кабельной и кросс-шины. Описание интерфейса, приведенное ниже, относится к самому распространенному варианту — кабельной шине. В этом варианте не используются (отсутствуют) сигналы Clk25 и Backplane.

Для кросс-шины в интерфейсе LINK-PHY сигнал SClk имеет пониженную частоту (24,756 или 12,288 МГц для S50 и S25 соответственно); для идентификации используемой частоты введен сигнал Clk25 (0 — 12,288 МГц, 1 — 24,576 МГц). Признаком подключения LINK-уровня к Backplane PHY является дополнительный сигнал Backplane (от PHY к LINK'у). Особенности PHY-уровня Backplane-варианта рассмотрены в главе 22.

Таблица 23.1. Сигналы интерфейса LINK-PHY

Сигнал	Назначение
D[0:7]	Двунаправленная шина данных. На S100 используются только линии D[0:1], на S200 — D[0:3], на S400 — D[0:7]

Сигнал	Назначение
Ctl[0:1]	Состояние интерфейса (табл. 23.2)
LReq	Link Request, последовательный интерфейс, по которому LINK инициирует запрос к PHY (для передачи пакетов и обращения к регистрам)
SClk	Синхронизация (49,152 МГц) для вышеприведенных сигналов. Источник меняет состояние сигнала по спаду SClk, приемник фиксирует состояние по фронту SClk
LPS	Link Power Status, признак активности (включения) LINK-уровня
LinkOn	Сигнал для включения LINK-уровня
Direct[1]	Признак непосредственного соединения LINK-PHY (нет дифференцирования сигналов)

[1] В микросхемах этот вывод может называться «Iso» (изоляция).

Интерфейсом управляет PHY; LINK может посылать запросы только в определенных состояниях интерфейса. Право передачи информации по двунаправленным линиям данных LINK получает от PHY. Текущее состояние интерфейса (использование шины данных) определяется сигналами Ctl[0:1] в соответствии с табл. 23.2.

Таблица 23.2. Состояние интерфейса LINK-PHY

Ctl[0:1]	Состояние интерфейса
	Линиями Ctl[0:1] управляет PHY:
00	*Idle* — нет активности (LINK может посылать запрос)
01	*Status* — PHY по шине данных посылает информацию о событиях или ответ на запрос чтения регистра
10	*Receive* — PHY передает LINK-уровню пакет, принимаемый с шины, или генерируемый им (PHY-пакет), или пакет ответа на запрос к регистрам PHY
11	*Grant* — PHY передает LINK'у право использования шины данных
	Линиями Ctl[0:1] управляет LINK:
00	*Idle* — нет активности: LINK завершил передачу и отдает управление интерфейсом уровню PHY
01	*Hold* — LINK удерживает шину, готовясь к передаче пакета (или после передачи пакета, собираясь послать следующий без повторного арбитража)
10	*Transmit* — LINK передает данные PHY (для передачи в шину или обращения к регистрам PHY)
11	Не используется

В состоянии *Status* PHY по линиям данных D[0:1] за два такта передает 4 бита состояния (Stat[0:1] в первом такте и Stat[2:3] во втором (табл. 23.3).

Таблица 23.3. Состояние, сообщаемое PHY

Бит	Значение
0	*ARB_RESET_GAP* — обнаружение зазора сброса арбитража (указание на начало нового интервала справедливости)

продолжение

Таблица 23.3 (*продолжение*)

Бит	Значение
1	*SUBACTION_GAP* — обнаружение нормального зазора арбитража (можно запрашивать право посылки очередного асинхронного пакета)
2	*BUS_RESET_START* — обнаружение сброса на шине
3	*PHY_INTERRUPT* — прерывание от PHY, причина уточняется чтением PHY-регистра 0101b. Возможные события: обнаружение петли во время идентификации дерева; обнаружение снижения питающего напряжения (ниже 7,5 В); обнаружение тайм-аута арбитража; обнаружение изменения напряжения смещения

Запросы LINK к PHY

Запрос от LINK к PHY посылается в последовательном виде по линии LReq. Запрос начинается с единичного старт-бита (`LReq0` = 1) и завершается нулевым стоп-битом (`LReq6` в 1394 и `LReq7` в 1394a). За старт-битом следует *код типа запроса* (биты `LReq[1:3]`), за которым следуют биты тела запроса. Типы запросов, их коды и состояния интерфейса, во время которых LINK их имеет право посылать, приведены в табл. 23.4.

Таблица 23.4. Запросы LINK к PHY

Код запроса LReq[1:3]	Назначение	Состояния, в которых допустима подача запроса
000	*ImmReq*, немедленный запрос (без задержки арбитража) для передачи пакета квитирования	*Idle*, *Receive*
001	*IsoReq*, запрос изохронной передачи	Все (и Idle, когда интерфейсом владеет LINK)
010	*PriReq*, запрос передачи пакетов начала цикла и приоритетных асинхронных пакетов	*Idle*, *Status*
011	*FairReq*, запрос передачи обычных асинхронных пакетов	*Idle*, *Status*
100	*RdReg*, запрос чтения регистра PHY	Все
101	*WrReg*, запрос записи регистра PHY	Все
110	*Acceleration Control*, управление ускорением арбитража: LReq[4]= 1 — разрешить, 0 — запретить	Все
111	Резерв	

Запросы на передачу

Тело запросов передачи пакетов содержит код скорости, за который отвечают биты `LReq[4:5]`; в 1394a он занимает биты `LReq[4:6]`. Дополнительный бит в запросе 1394a появился ради введения дополнительных скоростей; назначение кодов ско-

рости обеспечивает совместимость запросов 1394 и 1394a (на скоростях S100–S400 бит LReq6=0, он может восприниматься как стоп-бит). Коды скоростей в запросе 1394a приведены далее:

- S100 — 000 (00 в 1394); S800 — 110;
- S200 — 010 (01 в 1394); S1600 — 001;
- S400 — 100 (10 в 1394); S3200 — 011.

Послав запрос на передачу пакета, LINK должен следить за состоянием интерфейса (по состоянию сигналов Ctl[1:0]), дожидаясь разрешения передачи пакета — состояния *Grant*. Это состояние появляется, когда PHY выигрывает арбитраж для данного запроса. Обнаружив состояние *Grant*, LINK берет на себя управление интерфейсом и шиной, установив на линиях Ctl[1:0] признак состояния *Hold*. Во время передачи по шине данных пакета LINK устанавливает признак состояния *Transmit*. Завершив передачу пакета, LINK устанавливает признак состояния *Idle*, чем отдает управление интерфейсом PHY-уровню. Если LINK посылает подряд несколько пакетов (в соединенных транзакциях или при ускоренном арбитраже), то между пакетами он удерживает интерфейс (и шину), сигнализируя состояние *Hold*.

Состояния *Idle* и *Status*, наблюдаемые LINK'ом после подачи запроса передачи, означают, что запрос обслуживается (LINK должен ждать дальше). Состояние *Receive* для обычных и приоритетных запросов означает, что запрос отвергнут PHY (LINK его должен повторить позже). Запрос передачи пакета квитирования не отвергается (для него *Receive* означает ожидание). Запросы изохронных передач отвергаются только в случае получения в состоянии *Status* уведомления об обнаружении нормального зазора арбитража (истекло время, отведенное на изохронные передачи в данном цикле).

Прием пакетов

Пакеты, принимаемые по шине и предназначаемые данному узлу, PHY передает по шине данных D[0:7] в состоянии *Receive*. При приеме PHY передает по D[0:7] некоторое количество байтов FFh (признак *Data-On*), байт-индикатор скорости и собственно пакет данных. Признаком конца пакета будет переход интерфейса в состояние *Idle*. При подсчете CRC-кода LINK учитывает только байты пакета (без индикатора скорости). Байт-индикатор скорости указывает LINK'у, сколько линий D[0:7] будет использоваться при передаче данного пакета. Если LINK не поддерживает данную скорость, он должен игнорировать весь трафик до обнаружения состояния *Idle*. Значение индикатора скорости приведено в табл. 23.5.

Таблица 23.5. Индикация скорости от PHY к LINK

Скорость	D[0:7]	Скорость	D[0:7]
S100	00xxxxxx	S800	01010001
S200	0100xxxx	S1600	01010010
S400	01010000	S3200	01010011

Доступ к регистрам PHY

Обращение к регистрам PHY (см. далее) через запросы от LINK разрешено в любом состоянии интерфейса, за исключением времени, когда PHY занят обработкой предыдущего запроса чтения. Запросы обращения к регистрам выполняются всегда, их может отменить только сигнал сброса. Запрос передается в последовательном коде по линии LReq. Для запросов обращения к регистрам PHY в теле передается адрес регистра, а в запросе записи — и сами данные записи. В запросе чтения в битах [4:7] передается адрес запрашиваемого регистра, бит 8=0 — стоп бит. В запросах записи после адреса в битах [8:15] передается записываемый байт. Ответ на запрос чтения регистра передается по линиям D[0:1] в состоянии *Status* за 8 тактов (передаются 16 бит). За первые 2 такта передается текущее значение состояния (см. табл. 23.3), далее передается 4-битный адрес регистра и 8 бит считанных данных.

Параллельный интерфейс B PHY-LINK 1394b

В IEEE 1394b параллельный интерфейс между физическим уровнем и LINK-уровнем модифицирован для поддержки высоких скоростей и бета-режима, и этот вариант интерфейса называется *B PHY-LINK*. Схема интерфейса приведена на рис. 23.2, назначение сигналов — в табл. 23.6. Этот интерфейс обеспечивает передачу данных на скоростях вплоть до S800. Здесь в шине данных всегда используются все 8 бит, а при передаче на более низких скоростях байты данных растягиваются до 2, 4 или 8 тактов. Интерфейс может работать и в традиционном режиме (1394/1394a), при этом он синхронизируется от линии PClk (она будет играть роль SCLK), частота этого сигнала становится 49,152 МГц. Сигналы LClk и PInt при этом не используются.

Таблица 23.6. Сигналы интерфейса LINK-PHY 1394b

Сигнал	Назначение
D[0:7]	Двунаправленная шина, по которой передаются данные, код скорости и тип запроса. Данные от PHY синхронизируются по PClk, данные от LINK — по LClk
Pint	PHY Interrupt, линия для последовательной передачи информации о состоянии PHY, содержимого регистров и прерываний по событиям
Ctl[0:1]	Состояние (фаза) интерфейса (табл. 23.7) синхронизируется по PClk, когда интерфейсом управляет PHY, и по LClk, когда интерфейсом управляет LINK
LReq	Link Request, последовательный интерфейс, по которому LINK инициирует запрос к PHY (для передачи пакетов и обращения к регистрам) Синхронизируется по LClk
PClk	Синхронизация (98,304 МГц) для сигналов, передаваемых от PHY. Источник синхронизации — PHY
LClk	Синхронизация для сигналов, передаваемых от LINK. Сигнал вырабатывается LINK'ом из PClk и совпадает с ним по частоте
LPS	Lint Power Status, признак активности (включения) LINK-уровня
LinkOn	Сигнал для включения LINK-уровня

Сигнал	Назначение
Beta_Mode_link	Признак работы интерфейса в режиме 1394b (при отсутствии сигнала интерфейс работает в режиме 1394a)
Direct[1]	Признак непосредственного соединения LINK-PHY (нет дифференцирования сигналов)

[1] В микросхемах этот вывод может называться Iso (изоляция).

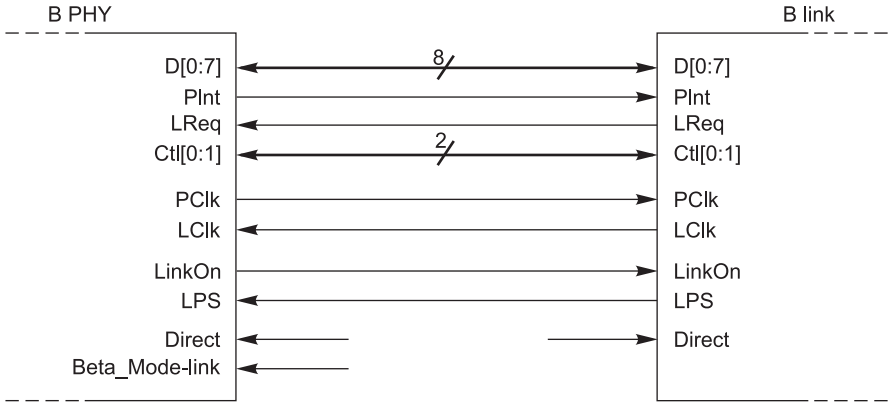


Рис. 23.2. Интерфейс B PHY-LINK 1394b

Интерфейсом управляет PHY или LINK, текущее состояние интерфейса (использование шины данных) определяется сигналами Ctl[0:1] в соответствии с табл. 23.7. По сравнению с 1394a интерфейс изменен в том, что все обращения к регистрам PHY происходят без использования шины данных: записываемые данные передаются по линии LReq, а считываемые данные и прерывания от PHY приходят по линии PInt.

Таблица 23.7. Состояние интерфейса B PHY-LINK (1394b)

Ctl[0:1]	Состояние интерфейса
	Линиями Ctl[0:1] управляет PHY:
00	*Idle* — нет активности (дежурное состояние)
01	*Status* — PHY по шине данных посылает информацию о состоянии шины
10	*Receive* — PHY передает LINK-уровню пакет, принимаемый с шины
11	*Grant* — PHY передает LINK'у право использования шины данных
	Линиями Ctl[0:1] управляет LINK:
00	*Idle* — нет активности: LINK завершил передачу и отдает управление интерфейсом уровню PHY
01	*Transmit* — LINK передает данные PHY
10	Не используется
11	*Hold*/*More_Info* — LINK удерживает шину, готовясь к передаче пакета. В конце передачи пакета указывает на передачу следующего запроса

Данные, принимаемые и передаваемые по параллельной шине, растягиваются на n-тактов, в зависимости от скорости передачи: для S100 n = 8, для S200 n = 4, для S400 n = 2, для S800 каждый байт передается только в одном такте (n = 1). Более высокие скорости данным интерфейсом не поддерживаются.

Запросы LINK к PHY в 1394b

Запрос от LINK к PHY посылается в последовательном виде по линии LReq. Обобщенный формат запроса приведен на рис. 23.3, по сравнению с запросами традиционного интерфейса он усложнен. Запрос начинается с единичного старт-бита (`LReq0` = 1) и завершается нулевым стоп-битом. За старт-битом следует *код типа запроса* (биты `RT[0:3]`, табл. 23.8), за которым следуют поля, необходимые для данного типа запроса:

- бит `RFMT` (Request Format) определяет формат запроса: 0 — явно не задан, 1 — B-формат;
- поле `RS[0:3]` (Request Speed) определяет скорость: 0000 — S100, 0010 — S200, 0100 — S400, 0110 — S800 (остальные значения зарезервированы);
- поле `RA[0:3]` содержит адрес регистра PHY;
- поле `RD[0:7]` (Request Data) содержит записываемые данные.

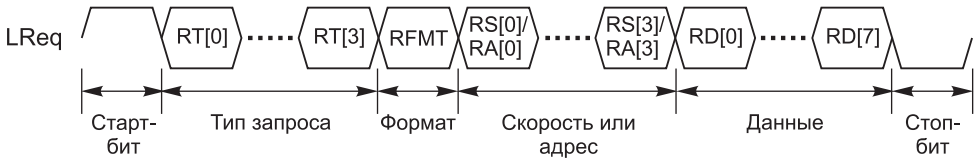


Рис. 23.3. Формат запроса к PHY

В ответ на запросы передачи PHY, выиграв арбитраж, предоставляет LINK'у право на передачу пакета. По запросу записи в регистр PHY выполняется соответствующая операция, никакого ответа не предусматривается. В ответ на запрос чтения регистра PHY данные (и адрес) будут переданы по линии Pint.

Таблица 23.8. Запросы LINK к PHY

Код запроса RT[0:3]	Назначение	Требуемые поля
0000	Резерв	
0001	*PH_IMMED_REQ*, немедленный запрос передачи (для посылки пакета квитирования)	RF, RS
0010	*PH_NEXT_EVEN*, запрос очередной передачи асинхронного пакета в четной фазе интервала справедливости	RF, RS
0011	*PH_NEXT_ODD*, запрос очередной передачи асинхронного пакета в нечетной фазе интервала справедливости	RF, RS
0100	*PH_CURRENT*, запрос очередной передачи асинхронного пакета в текущей фазе интервала справедливости	RF, RS

Код запроса RT[0:3]	Назначение	Требуемые поля
0101	Резерв	
0110	*PH_ISOCH_REQ_EVEN*, запрос передачи изохронного пакета в четном периоде	RF, RS
0111	*PH_ISOCH_REQ_ODD*, запрос передачи изохронного пакета в нечетном периоде	RF, RS
1000	*PH_CYC_START_REQ*, запрос передачи пакета начала цикла	RF, RS
1001	Резерв	
1010	*PH_REG_READ*, запрос чтения регистра PHY	RA
1011	*PH_REG_WRITE*, запрос записи в регистр PHY	RA, RD
1100	*PH_ISOCH_PHASE_EVEN*, сообщение от LINK'а (не мастера циклов) о приеме пакета начала цикла с четной фазой, обеспечивает синхронизацию LINK'а и PHY, если последний из-за ошибки не принял маркер *Cycle Start*	
1101	*PH_ISOCH_PHASE_ODD*, то же для нечетной фазы	
1110	*PH_CYCLE_START_DUE*, сообщение от LINK'а (не мастера циклов) о приеме уведомления о начале цикла	
1111	Резерв	

Прием пакета

Прием пакета иллюстрирует рис. 23.4. После индикации начала приема (*Data_On*) принимается байт скорости `SPD`, определяющий также и формат (табл. 23.9). Далее принимаются байты данных — пакет, передаваемый по шине.

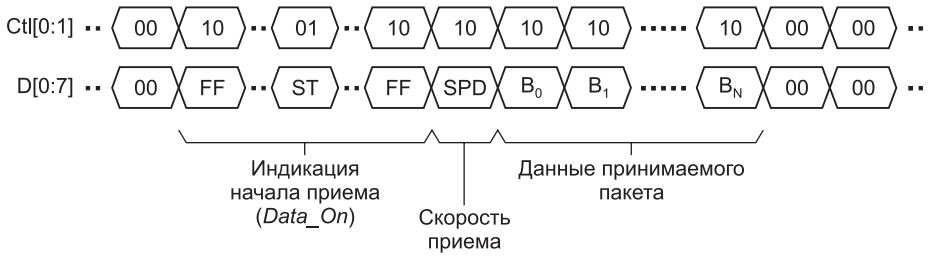


Рис. 23.4. Прием пакета по интерфейсу B PHY-LINK

Во время индикации начала приема по шине данных передается байт `ST`, сообщающий состояние шины:

- `ST[0]` — Bus Reset, сброс шины;
- `ST[1]` — Arbitration Reset Gap Odd, признак зазора сброса арбитража и начала нового нечетного интервала справедливости;
- `ST[2]` — Arbitration Reset Gap Even, признак зазора сброса арбитража и начала нового четного интервала справедливости;

- ST[3] — Cycle Start Odd, признак начала нечетного цикла;
- ST[4] — Cycle Start Even, признак начала четного цикла;
- ST[5] — Subaction Gap, признак зазора арбитража. После сброса означает завершение сброса и инициализации шины.

Таблица 23.9. Кодирование скорости и формата принимаемого пакета

SPD[0:7]	Скорость, формат
0000 0000	S100 Legacy
0000 0001	S100 Beta
0000 0100	S200 Legacy
0000 0101	S200 Beta
0000 1000	S400 Legacy
0000 1001	S400 Beta
0000 1101	S800 Beta
1111 1111	DATA_ON, признак начала приема
Остальные значения	Резерв

Передача пакета

В ответ на запрос *передачи пакета* PHY, сигнализируя состояние *Grant* (Ctl[0:1] = = 11), посылает LINK-уровню байт GT, в котором сообщает формат, скорость и тип разрешенной передачи. После этого он отдает управление интерфейсом LINK-уровню, указав состояние *Grant* (Ctl[0:1] = 11), а затем и прекращая управлять линиями Ctl[0:1] и D[0:7]. Теперь LINK устанавливает состояние *Hold* (Ctl[0:1] = 11), обозначая занятость интерфейса и шины, и начинает передавать пакет данных (состояние *Transmit*, Ctl[0:1] = 01). После передачи пакета LINK сигнализирует состояние *MORE_INFO* (Ctl[0:1] = 11), в котором по шине данных передается байт дополнительной информации AI[0:7]. В этом байте может присутствовать встроенный последующий запрос шины с указанием его формата, типа и скорости. После этого LINK указывает состояние *Idle* (Ctl[0:1] = 00), затем перестает управлять линиями Ctl[0:1] и D[0:7] и отдает управление PHY.

Получив право передачи пакета, LINK может послать только пакет, полностью удовлетворяющий свойствам, описанным байтом GT. Этот байт декодируется следующим образом:

- бит GT[0] задает тип формата: 0 — произвольный, 1 — B-формат;
- биты GT[1:3] задают тип гранта (пакета, разрешенного к передаче): 010 — изохронный, 101 — асинхронный, 110 — пакет начала цикла, 111 — немедленный (пакет квитирования), остальные значения зарезервированы;
- биты GT[5:6] задают скорость: 00 — S100, 01 — S200, 10 — S400, 11 — S800.

Байт дополнительной информации AI[0:7]содержит следующие поля:

- `AI[0]` — признак формата последующего запроса: 0 — произвольный, 1 — B-формат;
- `AI[1:3]` — тип встроенного запроса: 000 — нет запроса, 001 — *PH_ISOCH_REQ_ODD*, 010 — *PH_ISOCH_REQ_EVEN*, 011 — *PH_CURRENT*, 100 — *PH_NEXT_EVEN*, 101 — *PH_NEXT_ODD*, 110 — *PH_CYC_START_REQ*, 111 — резерв;
- `AI[4]` — признак последнего пакета в субакции;
- `AI[5:6]` — скорость для встроенного запроса: 00 — S100, 01 — S200, 10 — S400, 11 — S800;
- `AI[7]` — резерв.

Прием состояния PHY, прерываний и результатов чтения регистров

Состояние PHY, результаты чтения регистров и прерывания передаются последовательно по линии Pint (рис. 23.5). Значения кодов индикации состояния раскрыты в табл. 23.10. Поля адреса считываемого регистра (`RA`) и считанных данных (`RD`) используются только при индикации результатов чтения. Индикация результатов чтения может быть как запрошенной LINK'ом, так и неожиданной для него (чтение по инициативе PHY).

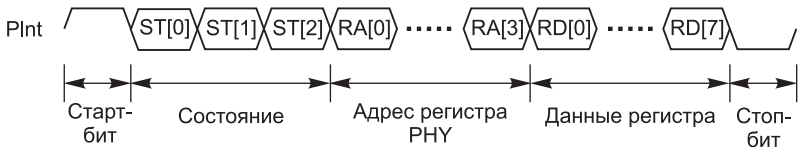


Рис. 23.5. Формат посылки состояния PHY

Таблица 23.10. Посылки состояния PHY

Код индикации состояния ST[0:2]	Назначение	Требуемые поля
000	*NOP*, нет индикации состояния	–
001	*PHY_INTERRUPT*, прерывание от PHY (тайм-аут конфигурирования из-за петли, пропадание кабельного питания, событие порта или тайм-аут арбитража)	–
010	*PHY_REGISTER_SOL*, результат чтения регистра по запросу LINK'а	`RA, RD`
011	*PHY_REGISTER_UNSOL*, результат неожидаемого чтения регистра	`RA, RD`
100	*PH_RESTORE_NO_RESET*, инициализация интерфейса с сообщением об отсутствии сбросов шины, о которых не было сообщено LINK'у (или PHY начал процесс восстановления интерфейса)	–

продолжение

Таблица 23.10 (*продолжение*)

Код индикации состояния ST[0:2]	Назначение	Требуемые поля
101	*PH_RESTORE_RESET*, инициализация интерфейса с сообщением о сбросе шины, происшедшем за время пребывания интерфейса в состоянии *Standby*	–
110	Не используется (игнорируется LINK'ом)	–
111	Резерв	–

Последовательный интерфейс PIL-FOP

Последовательный интерфейс подключает к однопортовому узлу с интегрированным портом, называемому *PIL* (Port Integrated Link), физический разветвитель *FOP* (Fan-Out PHY). Схема подключения с использованием этого интерфейса приведена на рис. 23.6. Узел PIL содержит один бета-порт, соединяемый с разветвителем FOP двумя дифференциальными однонаправленными сигнальными парами. В этих сигнальных парах может присутствовать и гальваническая развязка (конденсаторная или трансформаторная). Дополнительно от FOP должен передаваться сигнал LinkON, управляющий включением LINK-уровня по приему соответствующего пакета. Узел PIL и разветвитель FOP могут находиться в разных доменах питания. Внешние порты разветвителя FOP могут быть как бета-портами, так и универсальными, поддерживающими DS-сигнализацию.

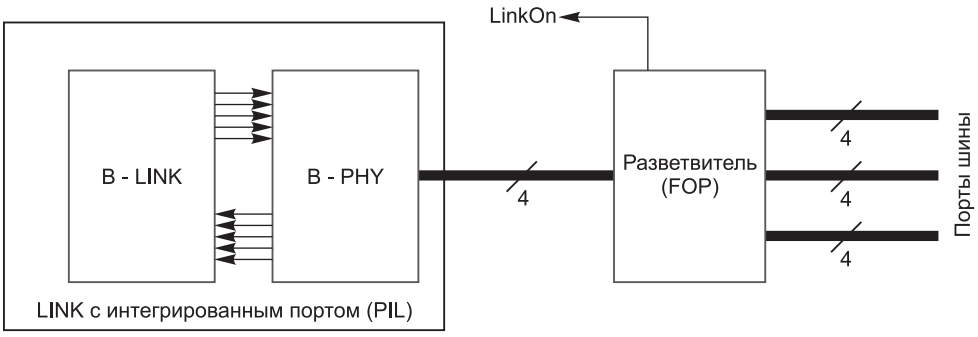


Рис. 23.6. Интерфейс PIL-FOP 1394b

Логически уровень PHY комбинации PIL-FOP выглядит как один многопортовый PHY, число его портов равно числу портов разветвителя с учетом дополнительного порта, обращенного к PIL. Этому порту назначается самый большой номер; он никогда не представляется как активный порт и его нельзя перевести в режим *Standby*. В пакетах самоидентификации этот порт отмечается как отсутствующий, удаленное обращение к его регистрам дает тот же результат, как обращение к отсутствующему порту.

PHY, предназначенный для роли FOP, может и не подключаться к PIL, а играть роль автономного повторителя-разветвителя (с отсутствующим LINK'ом). При этом порт, предназначенный для подключения PIL, может работать в традиционном, бета- или «двуязычном» режиме.

PIL и FOP обнаруживают присутствие (включение) друг друга, согласуют скорости и устанавливают синхронизацию так же, как и пара обычных бета-портов (см. главу 22). В посылках согласования скоростей FOP будет передавать бит `FOP_Capable`, а порт интегрированного узла — бит `PIL_Capable`. Протокол согласования позволяет установить им друг с другом правильные взаимоотношения.

Для того чтобы порты FOP и PIL логически связались в единый набор портов многопортового узла, в интерфейсе PIL-FOP определен специальный двухточечный протокол обмена со своими специальными *P2P*-пакетами (point-to-point). Пакеты P2P передаются только по интерфейсу между PIL и FOP, на общую шину они не выходят. Пакеты P2P передаются между пакетами основного трафика шины. Если во время передачи пакета P2P требуется передача основного трафика, то пакеты P2P прерываются, а пакеты основного трафика передаются беспрепятственно. Формат пакетов P2P приведен на рис. 23.7, назначение полей приведено ниже:

- `PP1` и `PP2` — байты-префиксы пакетов (00011xx1 и 00111xx1 соответственно);
- `D1` — байт, определяющий операцию:
  - 0000xxxx — нет операции;
  - 0001aaaa — запись в регистр FOP, aaaa — адрес, `D2` содержит данные;
  - 0010aaaa — чтение регистра aaaa;
  - 0011aaaa — неожидаемое (для PIL) чтение регистра aaaa, `D2` содержит считанные данные;
  - 0100aaaa — ожидаемые (для PIL) результаты чтение регистра aaaa, `D2` содержит считанные данные;
  - 0101iiii — уведомление о прерывании от FOP, iiii — код прерывания (0001 — прерывание от физического уровня, остальные коды пока не определены);
  - 0110xxxx — Cycle Start Due, если в FOP установлен бит `enab_accel`, PIL посылает это уведомление после начала каждого изохронного периода (по своим часам);
  - 0111xxxx — Restore_Notify, уведомление восстановления, посылает FOP, начинающий восстановление как «племянник»;
  - 10000000 – 11111111 — резерв;
- `D2` — данные записи или чтения (присутствуют не во всех пакетах);
- `DE1`, `DE2` — маркеры конца данных *DATA_END*.

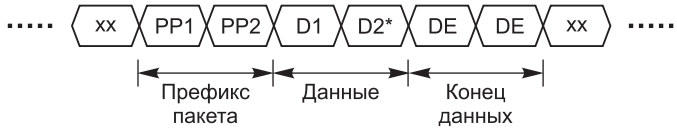


Рис. 23.7. Пакеты P2P интерфейса PIL-FOP

Каждый байт пакета на рисунке представлен в виде одного символа, который после кодирования 8B/10B представляется десятью битами сигнала на шине. Пакеты P2P могут быть прерваны в любом месте управляющими символами префикса данных или кода скорости, что означает начало пакета основного трафика, передаваемого по шине. Если пакет прерывается до `DE1`, то он повторяется передатчиком и игнорируется приемником. Пакет, прерванный после `DE1`, используется по назначению, поскольку все его содержательные поля и разделитель уже переданы.

Регистры PHY

Регистры PHY предназначены для управления функциями физического уровня, определения возможностей и состояния узла и состояния его портов. Регистры PHY имеют разрядность 8 бит. Они не отображаются на адресное пространство шины IEEE 1394. Доступ к регистрам PHY осуществляется по запросам его локального приложения (запросами по интерфейсу LINK-PHY). Некоторые поля регистров могут быть удаленно модифицированы пакетами физического уровня от узлов-диспетчеров. В 1394a появилась возможность удаленного доступа к регистрам с помощью асинхронных транзакций.

В 1394 все регистры PHY отображаются на плоскую карту (рис. 23.8), в которой 4-битный адрес позволяет выбрать любой из 16 регистров. Реальное число используемых регистров зависит от числа портов, которые имеются в данном PHY. Назначение полей регистров приведено ниже:

- `Physical_ID` — 6-битный идентификатор узла на шине;
- `R` (Root) — признак того, что данный узел является корневым;
- `PS` (Power State) — состояние питания: 1 — питание от шины в норме (>7,5 В);
- `RHB` (Root Hold Off Bit) — бит, принуждающий узел задерживать свое участие в идентификации дерева, чтобы стать корнем (управляется PHY-пакетом физического конфигурирования от диспетчера);
- `IBR` (Initiate Bus Reset) — инициирование сброса шины; установка бита вызывает немедленную подачу сигнала сброса на 166 мкс, после чего бит автоматически обнуляется;
- `Gap_cnt` — значение счетчика, используемого для определения зазоров арбитража (`Subaction_gap`) и сброса арбитража (`Arb_reset_gap`); по сбросу, связанному с изменением топологии, устанавливается `Gap_cnt` = 63. Управляется PHY-пакетом физического конфигурирования от диспетчера;
- `SPD` (Speed) — поддерживаемая скорость: 00 — S100, 01 — S200, 10 — S400, 11 — резерв;
- `E` (Enhanced) — признак использования регистров в расширенном пространстве (расположенных после адреса `#ports` + 0100);
- `#ports` (5 бит) — число портов PHY;
- `AStat[i]`, `BStat[i]` — состояние линий TPA, TPB *i*-го порта: 11 — Z, 01 — «1», 10 — «0», 00 — недопустимо;

- `Ch[i]` — признак дочернего *i*-го порта: 0 — p-порт, 1 — c-порт;
- `Con[i]` — состояние *i*-го порта: 0 — не подключен, 1 — подключен;
- `ENV` (Environment) — тип оборудования (используется только с регистрами в расширенном пространстве): 00 — кросс-шина (backplane), 01 — кабельная шина, 10 и 11 — резерв;
- `Reg_count` — число регистров в расширенном пространстве.

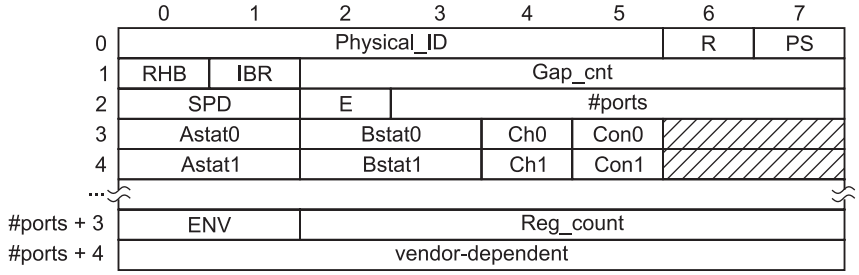


Рис. 23.8. Карта регистров PHY 1394

В варианте кросс-шины карта регистров имеет специфику. Здесь всегда имеется только один порт (`#ports` = 1), биты `R`, `PS` и `RHB` отсутствуют. В расширенном пространстве имеется один регистр (`Reg_count` = 1), в котором старший бит `TD` (Transceiver disable) запрещает работу приемопередатчика узла, младшие 4 бита `Priority` задают значение приоритета, вводимое узлом при арбитраже.

В 1394a набор регистров PHY был расширен, и для того чтобы сохранить совместимость с 4-битной адресацией, часть регистров адресуется косвенно. В этой карте в 1394b определены дополнительные поля. Карта регистров PHY 1394a/b приведена на рис. 23.9. Здесь адреса 1000–1111 отведены под восемь страничных регистров; какие именно регистры на них отображаются в данный момент, определяется полем `Page_select`, а для портов — еще и полем `Port_Select`. Первые два регистра полностью совпадают с регистрами 1394; третий регистр несколько отличается, и по нему можно определить, что остальные регистры являются регистрами 1394a/b. Новые поля регистров приведены ниже:

- `Extended` (3 бита) — признак формата 1394a/b (111);
- `Total_Ports` (4 бита) — число портов;
- `Max_Speed` (3 бита) — максимальная поддерживаемая скорость;
- `E_SB` (`Enable_Standby`) — разрешение узлу-«племяннику» перейти в режим Standby;
- `Repeater_delay` (4 бита) — максимальная задержка, вносимая повторителем (T = 144 + `Repeater_delay`×20 нс);
- `Link_Act` — активность LINK-уровня (соответствует биту `L` пакета самоидентификации), в 1394b называется `LCtrl`;
- `Contend` — признак кандидата на роль диспетчера (соответствует биту `C` пакета самоидентификации);

- `Rep_del_jit` (3 бита) — максимальное колебание задержки повторителя (`Rep_del_jit` + 1)×20 (нс), в 1394b называется `Jitter`;
- `Power_Class` (3 бита) — отношение к питанию от шины (соответствует полю `pwr` пакета самоидентификации);
- `Resume_int` (1394a) — разрешение прерывания по возобновлению работы приостановленного порта (при возобновлении установится бит `Port_event`). В 1394b на месте этого бита расположен бит `Watchdog`;
- `Watchdog` (1394b) — разрешение индикации обнаружения петли, пропадания питания, тайм-аутов и начала возобновления (resume) при неработающем интерфейсе PHY-LINK;
- `ISBR` (Initiate Short Bus Reset) — инициирование «короткого» сброса (через арбитраж);
- `Loop` — признак обнаружения петли, сбрасывается записью «1» в этот бит;
- `Power_Fail` — признак отказа питания, сбрасывается записью «1» в этот бит;
- `Time-out` — признак тайм-аута арбитража (200–400 мкс), сбрасывается записью «1» в этот бит;
- `Port_event` — признак смены состояния, конкретные источники разрешаются и запрещаются своими битами;
- `Enab_accel` — разрешение ускорения арбитража;
- `Enab_multi` — разрешение соединения пакетов, передаваемых на разных скоростях (не поддерживается узлами 1394);
- `Max_legacy_path_speed` (1394b) — максимальная скорость, обнаруженная в пакетах самоидентификации, переданных или транслированных данным узлом во время инициализации шины (000 —S100, 001 — S200, 010 —S400);
- `B_link` (1394b) — к PHY подключен LINK, поддерживающий бета-режим;
- `Bridge` (1394b) — поле, задающее значение `brdg` в пакете самоидентификации;
- `Page_select` (3 бита) — выбор страницы: 0 — страница регистров состояния порта, 1 — страница идентификатора производителя, 2–7 — резерв;
- `Port_select` (4 бита) — выбор порта.

Страница состояния порта 1394a (рис. 23.10) содержит следующие поля:

- `AStat`, `BStat` — состояние линий TPA, TPB: 11 — Z, 01 — «1», 10 — «0», 00 — недопустимо;
- `Child` — признак дочернего порта: 0 — p-порт, 1 — c-порт;
- `Con` (`Connected`) — состояние порта: 0 — не подключен, 1 — подключен;
- `Bias` — обнаружение смещения на линии TPB (признак подключения устройства);
- `Dis` (`Disabled`) — порт запрещен;
- `Neg_Spd` (Negotiated Speed) — скорость порта, согласованная с партнером;

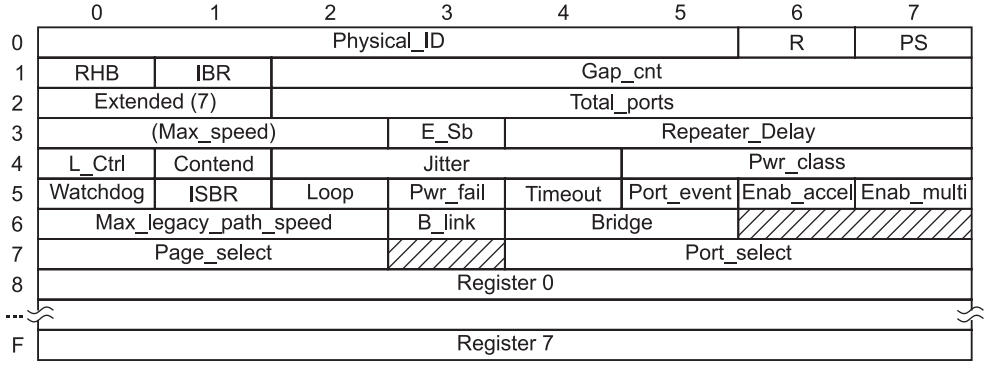


Рис. 23.9. Карта регистров PHY 1394a/b

- `Int_Enable` — разрешение прерывания (установки бита `Port_evnt`) по событию подключения, смены состояния смещения, запрету и отказу порта;
- `Fault` — обнаружение ошибки во время операции приостановки (suspend) и возобновления (resume), сбрасывается при выходе из режима приостановки, возобновлении нормальной сигнализации и при записи «1» в этот бит.

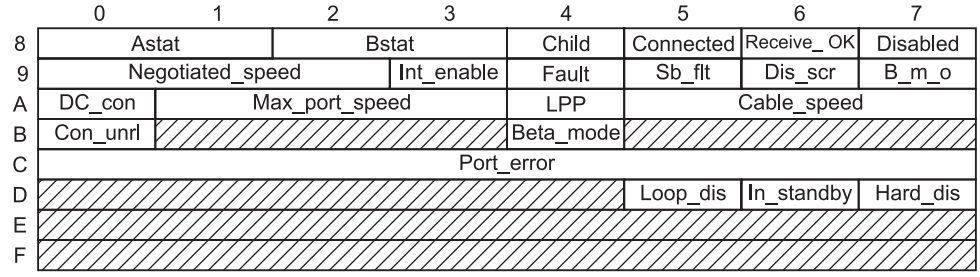


Рис. 23.10. Страница состояния порта PHY 1394a/b

В 1394b в странице состояния порта появились новые поля:

- `Receive_OK` — прием в норме: в DS-режиме — обнаружено установившееся (без дребезга) значение смещения, в бета-режиме — обнаружен устойчивый сигнал;
- `Sb_flt` (`Standby_fault`) — обнаружение ошибки в состоянии *standby*, сбрасывается при записи «1» в этот бит и по соответствующей команде удаленного управления;
- `B_m_o` (`Beta_mode_only_port`) — порт не поддерживает DS-режим;
- `Disable_scrambler` — запрет скремблирования данных (служебные посылки — запросы, состояния, маркеры — скремблируются всегда). Применяется для отладочных целей, в нормальной работе бит обнулен;
- `Cable_speed` — максимальная скорость, на которой может работать соединительный кабель. Если для кабеля скорость определить невозможно, то в этом поле устанавливается то же, что и в `Max_port_speed`;

- `Max_port_speed` — максимальная скорость, разрешенная для порта; 000 — S100, 001 — S200, 010 — S400, 011 — S800, 100 — S1600, 101 — S3200, 110 — резерв, 111 — порт может работать только в DS-режиме. Поле допускает запись (в пределах физических возможностей порта);
- `LPP` (Local_plug_present) — к порту подключена вилка кабеля (на другом конце его может и не быть ничего). Если нет возможности определить это состояние, то бит всегда единичный;
- `DC_con` (`DC_connected`) — обнаружено подключение партнера сигнализацией постоянным током;
- `Beta_mode` — порт работает в бета-режиме;
- `Con_unrl` (`Connection_unreliable`) — в бета-режиме не удалось согласование скоростей или потеряна синхронизация, сбрасывается записью «1»;
- `Port_error` — счетчик принятых неверных кодовых слов (инкрементируется до 255, обнуляется при чтении);
- `Hard_dis` (`Hard_disable`) — аппаратный запрет порта;
- `In_standby` — признак состояния *Standby*;
- `Loop_dis` (`Loop_disable`) — порт отключен для предотвращения петли.

В странице идентификатора производителя (рис. 23.11) определены следующие поля:

- `Compliance_level` — уровень совместимости: 0 — не определено, 1 — IEEE 1394a, 2 — IEEE 1394b, остальные значения зарезервированы;
- `Vendor_ID` — идентификатор разработчика;
- `Product_ID` — идентификатор продукта.

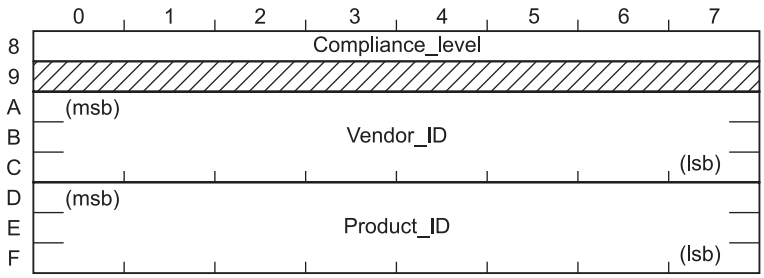


Рис. 23.11. Страница идентификатора производителя

Пакеты PHY

Физический уровень сам является источником и конечным получателем специальных пакетов, используемых для ряда целей:

- самоидентификации узлов шины;

- физического конфигурирования узлов;
- удаленного управления узлами;
- тестирования портов на образование петель в топологии (1394b).

Пакеты PHY-уровня по формату отличаются от пакетов транзакций и потоков: PHY-пакеты имеют длину 64 бита, первые 32 несут информацию, а в следующих 32 битах содержится инверсное значение информационных бит. Такой способ контроля позволяет отличить PHY-пакет от изохронного пакета с нулевой длиной данных: CRC-контроль для пакета PHY позволит сформировать сигнал об ошибке, что в совокупностью с длиной в 64 бита и является признаком PHY-пакета. Несоответствие контрольных бит информационным заставляет получателя игнорировать пришедший пакет. Адресация PHY-пакетов тоже иная — здесь используется только 5-битный идентификатор узла (PHY-пакеты за пределы шины не выходят). Назначение PHY-пакета определяется по первым двум битам.

Форматы *пакетов самоидентификации* приведены на рис. 23.12. В 1394–1995 допускалось до четырех пакетов (0, 1, 2 и 3), что позволяло описывать свойства до 27 портов одного узла. В 1394a сократили число портов до 16, в результате остались только пакеты 0, 1 и 2. Назначение полей пакетов самоидентификации приведено ниже:

- `phy_ID` — идентификатор узла;
- `L` (Link Active) — признак активности LINK-уровня;
- `gap_cnt` — счетчик для определения зазоров арбитража;
- `sp` (PHY Speed) — скорость, поддерживаемая PHY-уровнем данного узла (может быть и выше, чем поддерживает уровень LINK): 00 — S100, 01 — S200, 10 — S400, 11 — скорость определяется по регистру PHY;
- `brdg` (bridge) — поле зарезервировано для IEEE 1394.1 (мосты между шинами). В 1394 на этом месте было поле `del` (PHY Delay) — максимальная задержка повторителя: 00 — ≤ 144 нс, остальные значения зарезервированы;
- `c` (contender) — претендент на роль диспетчера;
- `pwr` (Power Class) — отношение к питанию (см. главу 21);
- `p0`...`p6` (Port status, 2 бита на порт) — состояние порта: 11 — активен (или в состоянии Standby), подключен к c-узлу, 10 — активен (или в состоянии *Standby*), подключен к p-узлу, 01 — неактивен (запрещен, не подключен, приостановлен), 00 — нет порта;
- `i` (Initiated reset) — признак того, что текущий сброс на шине вызвал этот узел;
- `m` (More packets) — признак наличия последующих пакетов самоидентификации;
- `r` — зарезервированные поля (на рисунке заштрихованы).

Форматы управляющих пакетов физического уровня приведены на рис. 23.13.

Пакеты физического конфигурирования являются широковещательными, их принимают все узлы шины. Пакеты используются двояко:

- для выборов нового корня (`R` = 1, `T` = 0) пакет посылается с указанием в поле `root__ID` идентификатора узла, которому надлежит стать корнем после гряду-

Рис. 23.12. Пакеты самоидентификации: *а* — пакет-0; *б* — пакет-1; *в* — пакет-2

щего сброса. Этот узел должен установить бит `RHB` (задержка при идентификации дерева), остальные узлы — его сбросить;

- для оптимизации зазоров арбитража (`R` = 0, `T` = 1) пакет посылается с указанием `phy_ID` корневого узла. В пакете указывается значение `gap_cnt`, рассчитанное для текущей топологии.

Длительность зазора арбитража и зазора сброса арбитража определяется через значение `gap_cnt` по формулам:

$T_{Subaction\_gap}$=(29+16×gap_cnt)/98,304 (мкс), диапазон значений 0,3–10,6 мкс;

$T_{Arb\_reset\_gap}$=(51+32×gap_cnt)/98,304 (мкс), диапазон значений 0,5–21 мкс.

По сбросу, связанному с изменением топологии шины (и по включению питания), принимается максимальное значение `gap_cnt`=63. В зависимости от топологии шины диспетчер шины может вычислить минимально допустимое значение `gap_cnt` и сообщить его всем узлам. Это новое значение будет действовать до следующего изменения топологии (PHY способен определить, связан ли сброс с подключением/отключением устройства; если не связан — значение `gap_cnt` сохраняется).

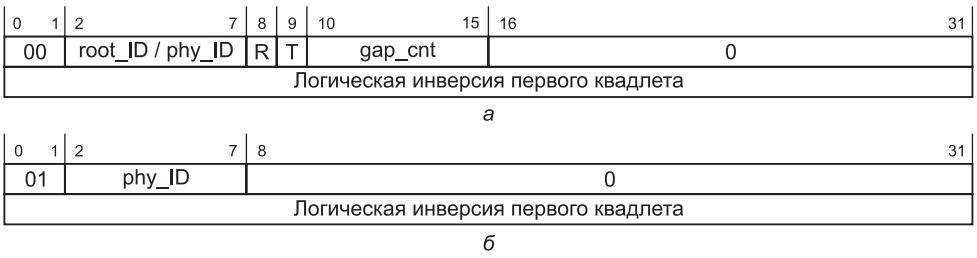


Рис. 23.13. Управляющие пакеты физического уровня: а — пакет конфигурирования; *б* — пакет *Link_On*

Пакет *Link_On* является директивой включения питания LINK-уровня (и вышестоящих) узлу, адресованному полем `phy_ID`.

Расширенные физические пакеты были введены в 1394a для удаленного доступа к PHY-регистрам, посылки пробного пакета и пакета возобновления:

- *пробный пакет* (Ping, рис. 23.14, *а*) заставляет узел с идентификатором `phy_ID` послать пакет(ы) самоидентификации. Диспетчер шины посылает эти пакеты всем узлам шины и измеряет время отклика, что позволяет ему рассчитать допустимое (минимальное) значение зазоров арбитража;
- *пакет удаленного доступа* (Remote Access Packet, рис. 23.14, *б*) позволяет прочитать PHY-регистр удаленного узла. *Пакет удаленного ответа* (Remote Access Reply, рис. 23.14, *в*) возвращает считанные данные;
- *пакет удаленной команды* (Remote Command Packet, рис. 23.14, *г*) позволяет подать команду узлу. В ответ на него узел через короткий интервал `ack_gap` посылает *пакет подтверждения* выполнения, в котором передается и текущее состояние (Remote Confirmation Packet, рис. 23.14, *д*);
- *пакет возобновления* (Resume Packet, рис. 23.14, *е*) предназначен для инициирования возобновления работы всех приостановленных портов адресованного устройства или всех устройств (может передаваться с широковещательным адресом 3Fh). Этот пакет не предполагает подтверждений.

Форматы расширенных физических пакетов 1394a приведены на рис. 23.14, назначение полей приведено ниже:

- `phy_ID` — идентификатор узла;
- `type` — тип команды: 0 — пробный пакет, 1 — чтение одного из базовых регистров, 5 — чтение страничного регистра, 8 — удаленная команда, код команды в поле `cmd`, Ah — удаленное подтверждение, Fh — возобновление;
- `page` — номер страницы;
- `port` — номер порта;
- `reg` — адрес регистра;
- `cmd` — код удаленной команды: 0 — *NOP* (нет операции), 1 — передать уведомление *TX_DISABLE_NOTIFY* и запретить порт, 2 — инициировать приостановку порта (*Suspend Port*), 4 — сбросить биты отказа порта (`Fault` и `Standby_Fault`), 5 — разрешить порт, 6 — инициировать возобновление работы порта (*Resume Port*); 7 — расширенная команда (код в поле `e_cmnd`);
- `e_cmnd` (в 1394b) — расширенная команда: 0 — *NOP* (нет операции), 1 — инициировать переход в *Standby* для партнера; 2 — инициировать выход из *Standby* для партнера, 3–7 — резерв;
- `s` (standby_fault, 1394b) — состояние одноименного бита страницы состояния порта;
- `f` (Fault) — состояние одноименного бита страницы состояния порта;
- `c` (Connected) — состояние одноименного бита страницы состояния порта;
- `b` (bias) — состояние одноименного бита страницы состояния порта;

- `d` (disabled) — состояние одноименного бита страницы состояния порта;
- `ok` — состояние команды: 1 — принята, 0 — отвергнута.

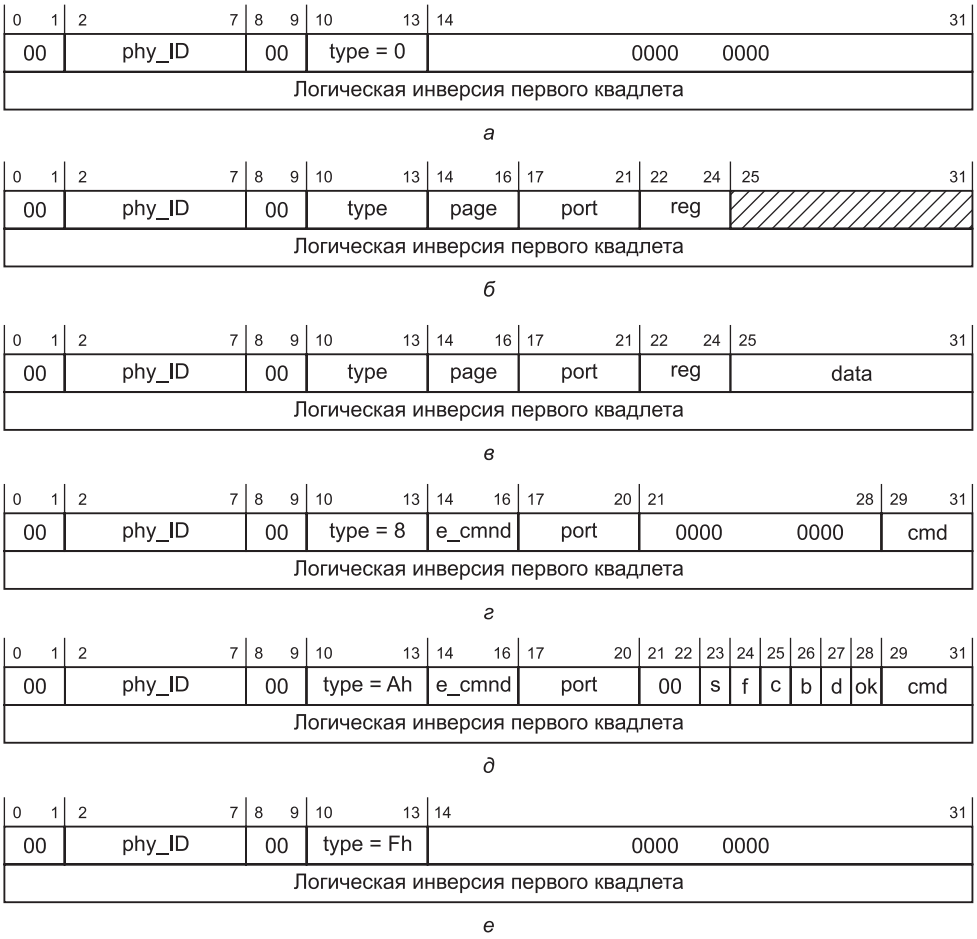


Рис. 23.14. Расширенные пакеты физического уровня: *а* — пробный пакет (Ping); *б* — пакет удаленного доступа; *в* — пакет удаленного ответа; *г* — пакет удаленной команды; *д* — пакет подтверждения удаленной команды; *е* — пакет возобновления

В IEEE 1394b для восстановления из состояния *Standby* используется пакет физического конфигурирования, который передается только от «дяди» к «племяннику» (рис. 23.15, *а*). При этом назначение полей становится следующим:

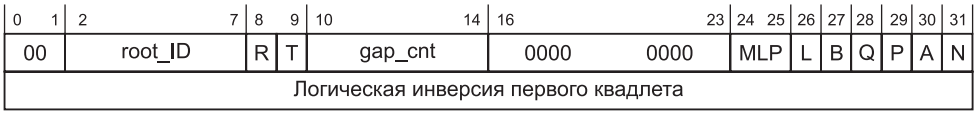
- `phy_ID` — идентификатор узла (возможно, изменившийся);
- `R` = 1 (игнорируется);
- бит `T` содержит копию переменной `gap_count_reset_disable` «дядиного» узла. Переменная `gap_count_reset_disable` — запрет сброса счетчика зазора в ис-

ходное значение, устанавливается по приему пакета физического конфигурирования, задающего длительность зазора;

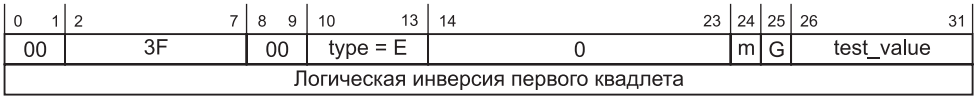
- `gap_cnt` – новое значение зазора;
- `MLP` — восстанавливаемое значение `max_Legacy_path_speed`;
- `L` — признак наличия корня в локальном B-облаке;
- `B` (B_bus) — признак отсутствия старых узлов на шине;
- `Q` — копия бита `R` из `R0` передающего узла;
- `P` — копия бита `PS` из `R0` передающего узла;
- `A` — текущая асинхронная фаза: 1 — нечетная, 0 — четная;
- `N` — признак сбросов шины, происшедших с момента перехода в состояние *Standby*.

Пакет тестирования на петли (Loop Test Packet, рис. 23.15, *б*), введенный в 1394b, содержит следующие поля:

- `m` (mode) — режим: 0 — тест, 1 — идет процесс подключения;
- `G` (Generation) — номер генерации, чередующиеся 0 и 1 указывают на изменение значения `test_value`
- `test_value` — значение, используемое для сравнения переданного и принятого пакетов.



а



б

Рис. 23.15. Новые пакеты физического уровня 1394b: *а* — пакет восстановления из состояния *Standby*; *б* — пакет тестирования на петли

ГЛАВА 24

Применение шины IEEE 1394

Шина IEEE 1394 широко применяется в устройствах различных классов, как связанных с компьютерами и являющихся их периферией, так и работающих без подключения к компьютерам. Принципиальное преимущество шины 1394 — ее идеология равноранговых взаимоотношений узлов и отсутствие необходимости в централизованном контроллере. Широкий круг применений и разные подходы к идеологии построения систем с использованием последовательной шины породили (и продолжают порождать) множество стандартов и спецификаций, часть из которых упоминается и бегло рассматривается в данной главе.

IEEE 1394 в компьютерах

Шина IEEE 1394 (FireWire) является «родным» интерфейсом компьютеров Macintosh фирмы Apple. В PC-совместимых ПК поддержка IEEE 1394 была провозглашена в спецификации PC'99 (Personal Computer Design Guidelines 1999, http://www.microsoft.com/hwdev/desguid.htm). В современных версиях ОС для PC-совместимых компьютеров поддерживаются спецификации OHCI, SBP-2, IEC61883 и другие.

Порты шины IEEE 1394 выводятся на внешние разъемы системного блока, что позволяет подключать внешние устройства с помощью кабелей. Для подключения съемных устройств с интерфейсом IEEE 1394 (и USB 2.0), устанавливаемых пользователем в отсеки корпуса ПК, имеются отдельные спецификации (http://www.device-bay.org/):

- Device Bay — для съемных периферийных устройств, устанавливаемых в 5" или 3,5" отсеки системного блока;
- CardBay — для съемных периферийных устройств, устанавливаемых слоты конструктива PC Card блокнотных ПК. Возможна совместимость с CardBus.

В IBM PC-совместимых ПК контроллеры IEEE 1394 чаще всего встречаются в виде карт расширения, но они уже встраиваются и в некоторые модели систем-

ных плат. Современные блокнотные ПК, как правило, имеют встроенный контроллер 1394.

Контроллер IEEE 1394 для PC, как правило, является устройством PCI, поскольку только шина PCI (и ее современные «родственники») способна обеспечить производительность обмена периферийного устройства с памятью, достойную шины FireWire. Для контроллера IEEE 1394 существует стандартная спецификация OHCI.

OHCI (Open Host Controller Interface) — интерфейс «открытого» хост-контроллера 1394, определяющий программную модель адаптера FireWire для PC-совместимых компьютеров с использованием прямого доступа к памяти. Версия 1.0 определяет базовый интерфейс, 1.1 — его уточнения и стандартный механизм управления энергопотреблением (ftp://ftp.austin.ibm.com/pub/chrptech/1394ohci). Подробно интерфейс описан в главе 25.

SBP-2 (Serial Bus Protocol-2) — транспортный протокол асинхронного обмена данными и командами и изохронной доставкой данных, широко используемый Microsoft и Apple. Протокол использует механизм прямого доступа к памяти и позволяет уменьшить необходимое число прерываний центрального процессора. Часть области системного ОЗУ ПК отображается на адресное пространство узла 1394. В нем формируется связанный список команд и указателей на буферы данных. Адрес этого списка передается устройству, и оно исполняет требуемые команды и обмены данными, самостоятельно обращаясь к памяти ПК транзакциями шины IEEE 1394. Спецификация SBP-2 не задает конкретных форматов команд — они специфичны для устройств разных классов. Черновик стандарта можно найти на ftp://ftp.symbios.com/pub/standards/io/x3t10/drafts/sbp2. Подробно протокол описан в главе 26.

Управление бюджетом питания для мобильных компьютеров описано в Intel Mobile Power Guidelines 2000 (http://developer.intel.com/design/mobile/Intelpower/int_mpg.htm).

Механизмы автоконфигурирования и загрузки драйверов и приложений для устройств IEEE 1394 определены в 1394 Plug & Play Specification от Microsoft (http://www.microsoft.com/hwdev/respec/pnpspecs.htm).

Использование IEEE 1394 для *управления аудио- и видеоустройствами* регламентируется стандартом *IEC 61883* (см. далее).

IEEE 1394 в локальной сети

Шина IEEE 1394 может быть основой небольших (домашних) локальных сетей с естественной поддержкой передачи изохронного трафика для аудио- и видеотехники. На физический и канальный уровень и промежуточные протоколы для домашних сетей имеется спецификация VESA «Video Electronics Standards Association Home Network».

Существует спецификация *IP/1394*, описывающая передачу IP-пакетов (IP v.4) по последовательной шине и поддержку протокола DHCP — динамического выделения IP-адресов (http://www.ietf.org/html.charters/ip1394-charter.html).

Для регистрации, обнаружения устройств в локальной сети и коммуникаций между ними имеется набор спецификаций *UPnP* (Universal Plug and Play, http://www.upnp.org/). Спецификации описаны для IP-сетей, но предполагается и прямое подключение устройств IEEE 1394.

IEEE 1394 в инструментальных устройствах

Для подключения измерительных приборов и устройств промышленной автоматики имеется спецификация «Industrial & Instrumentation Control Protocol» (ftp://fcext3.external.hp.com/dist/mxd/ieee1394/ii-wg/), описывающая коммуникационный протокол, похожий на AV/C. Для транспортировки команд приборного интерфейса IEEE 488 по шине IEEE 1394 имеется спецификация «IEEE 488 over 1394 Industrial & Instrumentation Control Protocol» (ftp://fcext3.external.hp.com/dist/mxd/ieee1394/ii-wg/).

IEEE 1394 для устройств хранения данных

С интерфейсом 1394 выпускаются разнообразные устройства хранения данных — жесткие диски, приводы CD и DVD, AV-диски (винчестеры, оптимизированные для записи и чтения мультимедийных данных). Выпускаются и преобразователи интерфейсов 1394–IDE, оформленные в виде корпусов для стандартных IDE-устройств форматов 5" или 3,5". В эти корпуса можно установить обычные винчестеры, приводы CD и DVD, получая переносные устройства хранения данных.

В устройствах хранения, подключаемых к компьютерам, шину IEEE 1394 обычно используют как транспортное средство передачи пакетов команд и данных SCSI. При этом для ОС и приложений устройства хранения выглядят как SCSI-устройства соответствующих классов. Спецификация *RBC* (Reduced Block Commands) описывает сокращенный набор команд SCSI, относящихся к устройствам хранения, с использованием протокола SBP-2 (ftp.symbios.com/pub/standards/io/x3t10/drafts/rbc/). Эта спецификация применяется для устройств хранения в ОС Windows, Mac OS и других.

Для дисковых устройств хранения в спецификациях *AV/C* (http://www.1394ta.org) имеется общая спецификация (disk general specification) и следующие расширения:

- AV/C MiniDisk subunit — расширения для мини-дисков;
- AV/C hard disk drive subunit — расширения для винчестеров;
- AV/C compact disk subunit — расширения для компакт-дисков;
- AV/C storage object descriptor subunit — абстрактное описание класса устройств хранения.

IEEE 1394 для передачи и печати изображений

Для печати (и передачи) неподвижных изображений по IEEE 1394 есть несколько разных подходов со своими протоколами:

- *1394 Printer Working Group (PWG) Imaging Device Communication Specification* — спецификация на использование транспортного протокола SBP-2 для принтеров, сканеров и прочих устройств ввода/вывода изображений, подключаемых к компьютерам. Из-за отсутствия единых спецификаций на форматы данных и команд для каждой модели принтера требуется свой драйвер;
- *протокол DPP* (Direct Printing Protocol) предназначен для непосредственного соединения устройств ввода и вывода изображений — сканеров, принтеров, фотокамер[1]. Протокол обеспечивает равноранговое соединение устройств и передачу масштабируемых изображений. (http://www.1394ta.org/Technology/Specifications/, http://www2.tokyoweb.or.jp/ieee1394pwgc/);
- *расширение AV/C Printer Sub-unit*, относящееся к принтерам, управляемым AV/C-командами.

IEEE 1394 для аудио- и видеоустройств

Интерфейс IEEE 1394 является общепринятым для современных цифровых устройств профессиональной и бытовой аудио- и видеотехники, которые используют эту шину и без участия компьютера. Кроме цифровых устройств, имеющих встроенные адаптеры 1394, к шине FireWire возможно подключение и традиционных аналоговых и цифровых устройств (плееры, камеры, мониторы) через адаптеры-преобразователи интерфейсов и сигналов.

Для шины 1394 привлекательна возможность соединения устройств бытовой электроники в «домашнюю сеть», причем как с использованием PC, так и без. При этом стандартные однотипные кабели и разъемы 1394 заменяют множество разнородных соединений устройств бытовой электроники с PC. Разнотипные цифровые сигналы (сжатые видеосигналы, цифровые аудиосигналы, команды MIDI и управления устройствами, данные) мультиплексируются в одну шину, проходящую по всем помещениям. Используя одни и те же источники данных (приемники вещания, устройства хранения, видеокамеры и т. п.), можно одновременно в разных местах просматривать (прослушивать) разные программы с высоким качеством, обеспечиваемым цифровыми технологиями. Применение компьютера с адаптером 1394 и соответствующим ПО значительно расширяет возможности этой сети. Компьютер становится виртуальным коммутатором домашней аудио-видеостудии. Приложения для аудио- и видеоустройств используют логические «вилки» (plugs) и «розетки» (sockets), которые являются аналогами разъемов, применяемых в обычной аппаратуре. Вилки соответствуют выходам, розетки — входам соответствую-

[1] В USB по назначению ему аналогично расширение OTG (On The Go).

щих устройств. «Вставляя» эти «вилки» в «розетки», можно собрать требуемую систему.

Для аудио- и видеоустройств существует ряд спецификаций:

- DVC Blue Book — первый стандарт на аппаратные средства и протоколы, используемые в цифровых видеоустройствах, выпускаемых консорциумом Digital Video Consortium Devices. Этот стандарт должен учитываться при разработке ПО для совместимости со старыми устройствами;
- 1394 Digital Video Conferencing Camera Specification — определение функций, сервисов и набора регистров камер для видеоконференций без использования компрессии (http://www.1394ta.org/Technology/Specifications/);
- *HAVi* (Home AV Interoperability) — протоколы, обеспечивающие взаимодействие бытовой аудиовидеотехники с поддержкой PnP, как с участием ПК, так и без. Спецификации разработаны объединенными усилиями производителей бытовой техники Grundig, Hitachi, Matsushita, Philips, Sharp, Sony, Thomson и Toshiba (http://www.havi.org). Спецификации совместимы с современными устройствами, соответствующими спецификациям AV/C (см. ниже), и ориентированы на следующее поколение бытовой техники и ее инфраструктуры. Набор команд, используемых в HAVi, определен спецификацией HAVi CTS (Command Transaction Set);
- спецификации IEC 61883 — управление аудио- и видеоустройствами (см. ниже);
- спецификации AV/C (Audio/Video Compatibility) — обеспечение совместимости аудио-видеоустройств с шиной IEEE 1394;
- Music LAN (MLAN) — спецификации для цифровых музыкальных инструментов (http://www.yamaha.co.jp/tech/1394mLAN/mlan.html).

Шина IEEE 1394 фигурирует в спецификациях OpenCable (http://www.opencable.com), описывающих интегрированные сервисы, предоставляемые с помощью сетей кабельного телевидения (в США): телевещание (аналоговое и цифровое), мультимедийные сервисы (голосовая и видеосвязь), интерактивные приложения и т. п. В этой сети у пользователя устанавливается устройство STB (set-top box), подключаемое к TV-кабелю, декодирующее сигналы и обеспечивающее предоставление пользователю оплаченных услуг. STB имеет внешние интерфейсы для подключения аудио-видеотехники: видеоинтерфейсы (композитный сигнал на RCA-разъемах, S-видео), аудиоинтерфейсы (аналоговые, SP/DIF). IEEE 1394 является цифровым интерфейсом STB-устройств, через который подключаются цифровые аудио-видеоустройства, DVD-плееры и устройства хранения данных.

Спецификации IEC 61883

Спецификации IEC 61883 описывают использование IEEE 1394 для управления аудио- и видеоустройствами (http://www.iec.ch). Спецификация разбита на части:

- *part 1.* Транспортировка изохронных данных при использовании AV/C, DVC, MPEG, AMP. Приводятся описания регистров управления изохронными подключениями (Isochronous Plug Control Registers), протокола управления со-

единениями CMP (Connection Management Protocol), протокола управления функциями FCP (Function Control Protocol), заголовков изохронных пакетов CIP (Common Isochronous Packet);

- *part 2*. Протокол транспортировки данных современных камкордеров (SD-DVCR Transport Protocol);
- *part 3*. Протокол транспортировки потоков данных HD камкордеров (HD-DVCR Transport Protocol);
- *part 4*. Протокол транспортировки потоков MPEG (включая DVB TS) для цифрового, спутникового и тому подобного телевидения;
- *part 5*. Протокол транспортировки потоков SDL-DVCR (цифровое телевидение со сжатием данных);
- *part 6*. Протокол транспортировки форматированных аудио- и музыкальных потоков.

Спецификации AV/C

AV/C (Audio/Video Compatibility) — спецификации, обеспечивающие совместимость аудио-видеоустройств с шиной IEEE 1394. Общая часть (General Specification) определяет команды управления бытовыми аудио-видео устройствами, используя архитектуру узла 1394. Расширение AV/C VCR Sub-unit относится к управляемым VCR и цифровым домашним видеосистемам. Эти спецификации призваны сократить многообразие индивидуальных описаний устройств, группируя их в классы (например, ленточных и дисковых устройств). Спецификации AV/C находятся на сайте http://www.1394ta.org/Technology/Specifications, в их составе имеются следующие:

- AV/C Asynchronous Connection Management — управление асинхронными соединениями (аналогичное управлению изохронными) для аудио-видеоустройств, нуждающихся в асинхронном обмене;
- AV/C Asynchronous Connections — передача данных через асинхронные соединения;
- AV/C Asynchronous Flow Control — управление потоком при асинхронной передаче;
- AV/C Isochronous Rate Control — управление скоростью изохронных данных;
- AV/C Printer Sub-unit — работа с принтерами;
- AV/C Camera Sub-unit — работа с цифровыми видеокамерами;
- AV/C Panel Sub-unit — использование экрана в качестве дисплея устройства (On-Screen Display);
- AV/C Monitor Sub-unit — управление видеомониторами;
- AV/C Audio Control Model — управление аудиоустройствами;
- AV/C Super Audio CD Model — управление функциями Super Audio CD;
- AV/C Changer Sub-unit — управление устройствами смены носителей (типа CD-changer);

- AV/C Tape Recorder Sub-unit — унифицированное управление ленточными записывающими устройствами. Заменяет спецификацию VCR sub-unit, имеет профили для стандартов DV, 8-миллиметровых лент, DVHS и т. п.;
- AV/C Timer Operation — управление таймерами и планированием для событий и заданий;
- AV/C Preset Sub-unit — установка предопределенных настроек для устройств;
- AV/C Diagnostics Sub-unit — запуск и сообщение результатов тестирования устройств.

Защита передаваемой информации

Одной из проблем цифровой передачи мультимедийной информации является защита авторских прав. Пользователь должен иметь возможность высококачественного воспроизведения принимаемых программ или приобретенных дисков, но их авторы (производители) должны иметь возможность защитить свои права, по своему усмотрению вводя ограничения на цифровое копирование. К этой стороне шины относятся следующие спецификации:

- «5C» — механизм защиты информации при цифровой передаче (digital transmission copy protection mechanism), разработанный объединением пяти компаний: Sony, Matsushita, Intel, Hitachi и Toshiba. Механизм включает в себя методы шифрования данных и управления ключами. Технология защищена патентами, информация по которым предоставляется только легальным производителям оборудования. Реализация технологии не требует слишком сложной логики и больших вычислительных ресурсов. Лицензиями управляет организация 5C Digital Transmission Licensing Authority (http://www.dtcp.com);
- XCA Digital Transmission Copy Protection Mechanism — альтернативный вариант защиты с использованием карт Smart Media в качестве ключевых элементов, предложенный Zenith/Thomson;
- OCPS (Open Copy Protection system) — открытая система защиты с использованием шифрования DES, системы NDS, MRJ и др. В спецификациях AV/C есть субблок условного доступа (Conditional Access Sub-unit), обеспечивающий дешифрацию скремблированных видео-аудио потоков (http://www. 1394ta.org/).

ГЛАВА 25

Интерфейс «открытого» хост-контроллера IEEE 1394 — OHCI

«Открытый» хост-контроллер (OHC 1394) представляет собой реализацию канального уровня (LINK) шины IEEE 1394 с дополнительными средствами поддержки уровней транзакций и управления шиной. Для высокопроизводительного обмена данными OHC содержит контроллеры прямого доступа к памяти (DMA). Контроллер поддерживает все типы пакетов, передаваемых по шине 1394. Данное описание основано на спецификации 1394 Open Host Controller Interface Specification, версия 1.1, 2000 год (ftp://ftp.austin.ibm.com/pub/chrptech/1394ohci).

Со стороны шины 1394 *хост* — узел с контроллером OHC — выглядит как обычный узел шины, способный выполнять функции мастера циклов и диспетчера изохронных ресурсов. Контроллер позволяет хосту быть инициатором любых транзакций шины 1394 и отвечать на любые транзакции, адресованные узлу хоста. В адресном пространстве этого узла расположены архитектурные регистры CSR и память конфигурации; большая часть пространства доступна для обращений в виде обычных транзакций шины. Часть пространства узла может отображать пространство физических адресов памяти хоста. Часть обращений к хосту может отрабатываться исключительно аппаратными средствами контроллера; остальные обращения хост отрабатывает программно. Принимаемые пакеты запросов для программной отработки OHC своими каналами DMA помещает в буферы, размещенные в памяти хоста. Пакеты ответов программа размещает в других буферах, из которых OHC организует их передачу в шину, опять же с помощью каналов DMA.

Для *асинхронных транзакций* контроллер обеспечивает чтение пакетов из системной памяти хоста и их передачу в шину; пакеты, принимаемые из шины, контроллер записывает в системную память. Обмен производится с помощью каналов DMA. Контроллер может функционировать и как шинный мост, аппаратно отрабатывая запросы транзакций чтения и записи шины 1394 как обращения к пространству памяти хоста.

Для *изохронных операций* OHC может исполнять роль мастера циклов, синхронизируясь от внутреннего генератора синхронизации или (необязательно) от внешнего источника. Если OHC не исполняет роль мастера циклов, то он поддерживает синхронизацию внутреннего таймера циклов с таймером узла-мастера циклов (по приему пакетов начала цикла). Для изохронных операций OHC имеет два контроллера DMA — для приема и для передачи данных. Каждый из этих контроллеров может поддерживать до 32 каналов DMA, называемых *контекстами DMA*. Передающий контроллер в каждом цикле может передавать данные из каждого контекста для связанного с ним изохронного канала. Принимающий контроллер способен в каждом цикле принимать данные в каждый контекст из связанного с ним канала. Кроме того, один из принимающих контекстов может быть настроен на прием данных из множества каналов.

По обнаружении *сброса на шине* OHC автоматически очищает все очереди асинхронных пакетов для передачи; прием пакетов не прерывается, но в потоке пакетов запросов появляется маркер, индицирующий факт сброса. Новый физический идентификатор узла (`PHY_ID`), получаемый от PHY, контроллер записывает в соответствующий регистр. Контроллер возобновит асинхронные передачи только по указанию программы, при этом повторное использование старых запросов в общем случае невозможно: физический идентификатор узла назначения может измениться. Изохронный прием и передача по сбросу не прекращаются — они возобновляются сразу по завершении инициализации.

Устройство контроллера OHC

Упрощенная структурная схема OHC приведена на рис. 25.1.

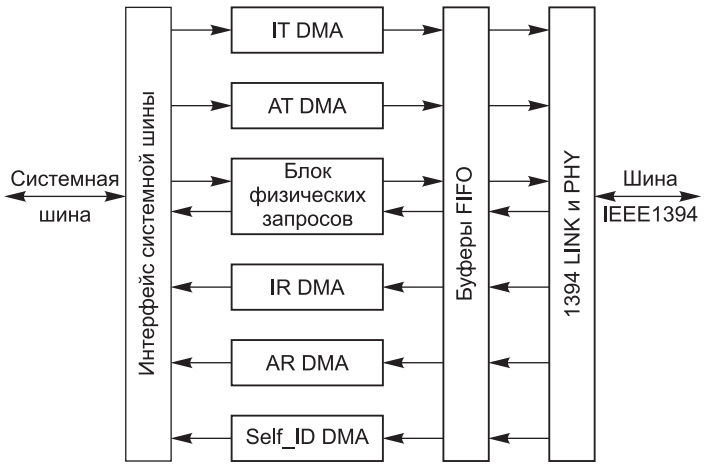


Рис. 25.1. Структура контроллера OHC 1394

Интерфейс системной шины (Host Bus Interface) обеспечивает взаимодействие с контроллером в двух режимах:

- ведомый режим (PCI Target), обеспечивающий программный доступ к регистрам контроллера со стороны центрального процессора хоста;
- ведущий режим (PCI Bus Master), обеспечивающий контроллеру возможность прямого доступа к системной памяти хоста. В этом режиме интерфейс системной шины должен выдерживать поток данных, по крайней мере, базовой скорости S100 (100 Мбит/с) с накладными расходами на организацию прямого доступа к памяти.

Контроллеры DMA обеспечивают обмен данными между шиной и системной памятью. В OHC имеются семь типов контроллеров DMA:

- контроллер асинхронной передачи (AT DMA);
- контроллер асинхронного приема (AR DMA);
- блок физических запросов, в который входят два контроллера:
  - контроллер приема аппаратно-обрабатываемых запросов (Physical Receive);
  - контроллер ответов для аппаратно-обрабатываемых запросов (Physical Response);
- контроллер изохронной передачи (IT DMA);
- контроллер изохронного приема (IR DMA);
- контроллер приема пакетов самоидентификации (Self_ID DMA).

Каждый контроллер работает со своими *контекстами* — наборами регистров, управляющих работой канала и выборкой запросов из списков, расположенных в системной памяти. Контроллеры асинхронной передачи и приема имеют отдельные контексты для запросов и ответов шинных транзакций. Контроллеры изохронного приема и передачи могут иметь до 32 контекстов каждый. Назначение и работа контроллеров подробно описано ниже.

LINK-уровень OHC передает пакеты из FIFO-буферов передающих каналов и отдает в FIFO принятые пакеты с корректным адресом, предназначенные данному узлу. Уровень выполняет следующие действия:

- передает и принимает пакеты форматов IEEE 1394;
- генерирует соответствующие пакеты квитирования для принятых асинхронных пакетов, отрабатывая однофазный или двухфазный протокол повторов (см. главу 18);
- выполняет функции мастера циклов;
- генерирует и проверяет корректность 32-битных CRC-кодов;
- обнаруживает пропуски пакетов начала цикла;
- взаимодействует с регистрами PHY;
- принимает изохронные пакеты (все время);
- игнорирует асинхронные пакеты во время изохронной фазы цикла.

Буферы FIFO, находящиеся между каналами DMA и LINK-уровнем, выполняют промежуточную буферизацию данных, считываемых из системной памяти для передачи в шину и принятых из шины для записи в память. Буферы FIFO обеспечивают и согласование выравнивания данных, побайтного для хоста и поквадлетно-

го для шины 1394. При необходимости буферы FIFO вставляют байты-заполнители, выравнивающие данные до границ квадлетов. Переполнение (overflow) или переопустошение (underrun) буферов (по вине интерфейса системной шины и памяти), приводящее к потерям принимаемых или передаваемых пакетов, контролируется аппаратными средствами OHC.

Буферы могут «на лету» выполнять преобразование форматов представления квадлетов. Шина IEEE 1394 и, соответственно, LINK-уровень работают с квадлетами, представленными в формате Big Endian (старший байт передается первым). Передача данных через хост-интерфейс может выполняться по выбору:

- в формате Big Endian, используемом на платформах фирмы Apple;
- в формате Little Endian (младший байт передается первым), используемом на платформах фирмы Intel.

Для поддержки функций управления OHC имеет 64-битный *регистр уникального идентификатора* (GUID, он же IEEE EUI-64), автоматически загружаемый из энергонезависимой памяти по сбросу контроллера (или однократно программируемый в самом контроллере).

Для выполнения функций диспетчера изохронных ресурсов контроллер имеет четыре *автономных регистра*, реализующих блокированные операции (*compare_swap*) как со стороны шины, так и со стороны хоста.

Адресное пространство узла OHC

Карта адресного пространства узла 1394, являющегося контроллером OHC, приведена на рис. 25.2. Здесь приведены 48-битные адреса — идентификатор узла, составляющий 16 старших бит адреса, опущен. Запросы со стороны шины к узлу отрабатываются либо аппаратно, блоком физических запросов, либо программно, средствами ПО узла. Программа хоста может фильтровать запросы, ограничивая список идентификаторов узлов, чьи обращения будут отрабатываться, и список узлов, чьи запросы будут отрабатываться аппаратно.

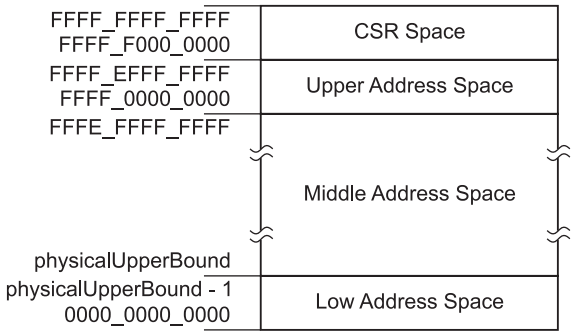


Рис. 25.2. Карта адресов узла OHC 1394

В карте имеются следующие области:

- нижние адреса (Low Address Space), ограниченные сверху адресом `physicalUpperBound`. Обращение по шине к этим адресам отрабатывается блоком физи-

ческих запросов в соответствии с критериями, заданными фильтрами асинхронных запросов. Запись в эту область может выполняться как *отправленная* (Posted Write): контроллер может сразу ответить квитанцией *ack_completed*, не дожидаясь физической записи в память. Отправленная запись повышает производительность отработки транзакций. Если требуется надежное подтверждение, то контроллер отвечает квитанцией *ack_pending* и по реальному выполнению записи пошлет пакет завершения транзакции. Верхняя граница задается в необязательном регистре `physicalUpperBound`; при его отсутствии размер области принимается равным 4 Гбайт. Регистр позволяет увеличить размер области. Область нижних адресов является окном, в котором хост-контроллер работает в качестве моста между шинами 1394 и системной шиной хоста (PCI).

- средние адреса (Middle Address Space) в диапазоне `physicalUpperBound`–FFFE FFFF FFFFh блоком физических запросов не обслуживаются. Обращения к ним отрабатываются программно, на запись в эту область контроллер квитанцией *ack_completed* не отвечает. Эта область используется для обмена данными в пакетных протоколах со своими механизмами подтверждения (например, TCP/IP).
- верхние адреса (Upper Address Space) FFFF 0000 0000–FFFF EFFF FFFFh также обслуживаются программно; на запрос записи в эту область контроллер ответит квитанцией *ack_pending*. О завершении записи программа должна сообщить посылкой пакета *resp_complete*. Эта область используется протоколами, требующими подтверждений.
- область CSR FFFF F000 0000–FFFF FFFF FFFFh, в которой расположены регистры CSR и память конфигурации. Почти все запросы обрабатываются программно, за исключением запросов к автономным регистрам и памяти конфигурации, которые отрабатываются как физические запросы.

Регистры OHC 1394

Регистры OHC предназначены для взаимодействия программ, исполняемых хостом, с контроллером шины 1394. Регистры OHC занимают 2048 байт в адресном пространстве хоста, их положение определяется из заголовка конфигурационного пространства PCI-устройства. Регистры доступны по чтению и записи только двойными словами (32 бит). Для регистров OHC принята нумерация бит, обычная для устройств PCI: старший бит — 31 (изображается слева), младший бит — 0 (справа). Карта регистров приведена в табл. 25.1, назначение регистров приводятся в описании работы узлов контроллера.

Таблица 25.1. Карта регистров OHC 1394

Смещение	Назначение
000	`Version`, версия и возможности OHCI
004	`GUID_ROM`, доступ к ПЗУ идентификатора
008	`ATRetries`, параметры автоповтора при занятости

продолжение

Таблица 25.1 (*продолжение*)

Смещение	Назначение
00Ch, 010h, 014h	`CSRData`, `CSRCompareData`, `CSRControl`, доступ к автономным ресурсам
018h	`ConfigROMhdr`, заголовок памяти конфигурации
01Ch	`BusID`, идентификатор шины («1394» в ASCII)
020h	`BusOptions`, опции шины из `Bus_Info_Block`
024h, 028h	`GUIDHi`, `GUIDLo`, уникальный идентификатор узла
034h	`ConfigROMmap`, адрес отображения памяти конфигурации
038h, 03Ch	`PostedWriteAddressLo`, `PostedWriteAddressHi`, адрес для неудавшейся отправленной записи
040h	`Vendor_ID`, идентификатор производителя
050h, 054h	`HCControlSet`, `HCControlClear` — `HCControl`, управление хост-контроллером
064h	`SelfIDBuffer`, адрес буфера пакетов самоидентификации
068h	`SelfIDCount`, число принятых пакетов самоидентификации
070h, 078h	`IRMultiChanMaskHiSet`, `IRMultiChanMaskLoSet`, установка масок мультиканального изохронного приема
074h, 07Ch	`IRMultiChanMaskHiClear`, `IRMultiChanMaskLoClear`, сброс масок мультиканального изохронного приема
080h, 084h	`IntEventSet`, `IntEventClear` —`IntEvent`, признаки прерываний
088h, 08Ch	`IntMaskSet`, `IntMaskClear` — `IntMask`, маски прерываний
090h, 094h	`IsoXmitIntEventSet`, `IsoXmitIntEventClear` —`IsoXmitIntEvent`, события изохронной передачи
098h, 09Ch	`IsoXmitIntMaskSet`, `IsoXmitIntMaskClear` — `IsoXmitIntMask`, маски событий изохронной передачи
0A0h, 0A4h	`IsoRecvIntEventSet`, `IsoRecvIntEventClear` — `IsoRecvIntEvent`, события изохронного приема
0A8h, 0ACh	`IsoRecvIntMaskSet`, `IsoRecvIntMaskClear` — `IsoRecvIntMask`, маски событий изохронного приема
0B0h, 0B4h, 0B8h	`InitialBandwidthAvailable`, `InitialChannelsAvailableHi`, `InitialChannelsAvailableLo`, регистры диспетчера изохронных ресурсов
0DCh	`FairnessControl`, управление приоритетным асинхронным доступом
0E0h, 0E4h	`LinkControlSet`, `LinkControlClear`, управление LINK-уровнем
0E8h	`NodeID`, идентификатор узла
0ECh	`PhyControl`, доступ к регистрам PHY
0F0h	`IsochronousCycleTimer`, таймер циклов
100h, 104h, 108h, 10Ch	`AsynchronousRequestFilterHiSet`, `AsynchronousRequestFilterHiClear`, `AsynchronousRequestFilterLoSet`, `AsynchronousRequestFilterLoClear`, фильтр асинхронных запросов

Смещение	Назначение
110h, 114h, 118h, 11Ch	`PhysicalRequestFilterHiSet`, `PhysicalRequestFilterHiClear`, `PhysicalRequestFilterLoSet`, `PhysicalRequestFilterLoClear`, фильтр физических запросов
120h	`PhysicalUpperBound`, верхняя граница адресов запросов, отрабатываемых аппаратно
180h... 18Ch	Контекст передачи асинхронных запросов (*AT_Request_DMA*)
1A0h... 1ACh	Контекст передачи асинхронных ответов (*AT_Responce_DMA*)
1C0h... 1CCh	Контекст приема асинхронных запросов (*AR_Request_DMA*)
1E0h... 1ECh	Контекст приема асинхронных ответов (*AR_Responce_DMA*)
200h + 16×n... 20Ch + 16×n	*n*-й контекст передачи изохронных пакетов (n = 0, 1, 2...31)
400h + 32×n... 41Ch + 32×n	*n*-й контекст приема изохронных пакетов (n = 0, 1, 2...31)

Взаимодействие хоста и OHC

Взаимодействие хоста и OHC производится несколькими способами:

- программные обращения процессора хоста к регистрам контроллера. Эти обращения обеспечивают общее управление контроллером и узлом хоста, управление контроллерами прямого доступа и их контекстами, управление прерываниями и их идентификацию, доступ к автономным регистрам;
- прямой доступ к памяти хоста (DMA), в котором различают:
  - DMA по контекстным программам. Структуры данных, расположенные в памяти хоста, описывают списки буферов данных. Контроллер автоматически последовательно обходит эти списки, выполняя передачи данных и обновляя в этих структурах информацию о состоянии выполнения. Каждый контекст DMA имеет набор регистров, через которые программа центрального процессора управляет контекстом;
  - DMA физических обращений к памяти. Контроллер аппаратно преобразует транзакции чтения и записи определенного диапазона адресов узла 1394 в транзакции чтения и записи определенного диапазона физической памяти. Для программы хоста эти обращения невидимы и при нормальной работе прерываний не вызывают;
- прерывания центрального процессора хоста, вырабатываемые контроллерам по различным событиям: исполнение (или отказ) передачи или приема, завершение обработки контекстной программы, прием потока пакетов самоидентификации и т. п. Контроллер имеет регистры идентификации и маскирования событий, вызывающих прерывания.

Контроллеры DMA

Контроллеры DMA OHC по способу управления разделяются на два типа:

- контроллеры, работающие по программам контекстов. *Программа контекста DMA* — это связанный список дескрипторов, описывающих команды и связанные с ними буферы данных. Программу контекста, расположенную в системной памяти хоста, формирует программа (драйвер), исполняемая процессором хоста. По контекстным программам работают контроллеры асинхронной передачи и приема, обслуживающие пакеты запросов и ответов, генерируемые и отрабатываемые программой хоста. По контекстным программам работают и контроллеры изохронного приема и передачи. Управление каждым контекстом (фактически — каналом DMA) производится с помощью своего блока регистров;
- Контроллеры, обслуживающие аппаратно обрабатываемые внешние запросы к узлу. Эти контроллеры управляются регистрами; никаких контекстных программ для них не создается. К данному типу относятся следующие контроллеры:
  - контроллеры блока физических запросов на внешние обращения к памяти хоста и автономым регистрам OHC;
  - контроллер записи пакетов самоидентификации.

Контроллер асинхронной передачи (Asynchronous Transmit), *AT DMA*, имеет по одному контексту для передачи запросов (*AT Request*) и ответов (*AT Responce*). Для каждого отправленного пакета контроллер ожидает пакет квитирования; в случае получения квитанции *ack_busy* (занято) контроллер организует программно заданное число попыток повтора. Контроллер может реализовать однофазный или двухфазный протокол повторов (см. главу 18). Если контроллер реализует изменение порядка исполнения, то он в случае занятости узла-получателя может продвигаться дальше по контекстной программе, возвращаясь к повтору данного пакета позже. Контроллер AT DMA отвечает и за ответы на запросы к физической памяти, обнаруженные принимающим контроллером DMA.

Прием асинхронных пакетов выполняют блок физических запросов и контроллер асинхронного приема.

Блок физических запросов, *Physical Request Unit*, включается в работу, когда приходит пакет запроса одного из трех типов:

- запрос к системной памяти хоста, доступной для узлов шины (лежащий в диапазоне нижних адресов);
- запрос блокированной транзакции (*compare_swap*), адресованный к одному из автономных регистров (для диспетчера изохронных ресурсов);
- запрос по адресу памяти конфигурации.

Остальные асинхронные пакеты обслуживает *контроллер асинхронного приема* (Asynchronous Receive), *AR DMA*, который имеет по одному контексту для приема запросов (*AR Request*) и ответов (*AR Responce*). Этот же контроллер обслуживает

и запросы блокированных транзакций, адресованных не к регистрам изохронных ресурсов, помещая их в контекст *AR Request*.

Контроллер DMA изохронной передачи (Isochronous Transmit), *IT DMA*, поддерживает от 4 до 32 контекстов, каждый из которых обеспечивает передачу одного канала.

Контроллер DMA изохронного приема (Isochronous Receive) поддерживает от 4 до 32 контекстов, каждый из которых обеспечивает прием одного канала. Один из контекстов можно запрограммировать на прием множества каналов.

Специальный контроллер, *Self_ID DMA*, принимает *пакеты самоидентификации* и укладывает их последовательно в один выделенный буфер. После каждого обнаруженного сброса контроллер начинает укладку пакетов с начала буфера, затирая предыдущие пакеты.

Фильтрация асинхронных запросов

Каждый приходящий асинхронный запрос, не относящийся к 1-килобайтной области памяти конфигурации, фильтруется трехступенчатым фильтром. Первая ступень фильтра по идентификатору узла-источника определяет судьбу запроса:

- запрос игнорируется (без посылки каких-либо пакетов квитирования);
- запрос направляется на вторую и третью ступени фильтрации, где по адресу памяти и идентификатору источника определяется способ обработки:
  - аппаратный, без привлечения программы хоста;
  - программный, помещением запроса в контекст асинхронного приема и дальнейшей обработкой программой хоста.

Запросы чтения памяти конфигурации принимаются от любых источников и отрабатываются аппаратно (если в хост-контроллере определен корректный образ памяти конфигурации).

Фильтрацией асинхронных запросов управляют 64-битные регистры `AsynchronousRequestFilter` и `PhysicalRequestFilter`, каждый из которых представлен регистрами установки и сброса старшей (`Hi`) и младшей (`Lo`) половин.

Первая ступень фильтрации выполняется в соответствии с содержимым регистра `AsynchronousRequestFilter`. В этом 64-битном регистре единица в старшем бите (`asynReqResourceAll`) разрешает отработку асинхронных запросов от всех источников. При его нулевом значении остальные биты разрешают отработку запросов от узлов с `PHY_ID`, соответствующих номерам бит (младший бит соответствует `PHY_ID` = 0). Управление данным фильтром осуществляется через регистры `AsynchronousRequestFilterHiSet` (100h), `AsynchronousRequestFilterHiClear` (104h), `AsynchronousRequestFilterLoSet` (108h), `AsynchronousRequestFilterLoClear` (10Ch).

Вторая ступень фильтрует прошедшие запросы по адресу смещения, указанному в пакете запроса. Кандидатами на физическую отработку являются запросы, адресованные к нижней области памяти и к автономным регистрам. Граница нижней области задается регистром `PhysicalUpperBound` (120h) — в нем содержатся стар-

шие 32 бита 48-разрядного адреса начала средней области памяти, которая уже не попадает в кандидаты на физическую отработку запросов. Младшие 16 бит считаются нулевыми. Если данный регистр отсутствует в OHC, то его чтение дает нули, что должно трактоваться как указание на размер нижней области 4 Гбайт.

Последний фильтр, управляемый регистром `PhysicalRequestFilter`, определяет способ обработки запроса-кандидата на физическую обработку. В этом 64-битном регистре единица в старшем бите (`physReqResourceAllBuses`) разрешает физическую обработку запросов от узлов всех шин. Остальные биты относятся к узлам локальной шины (на которой находится узел с OHC), они разрешают физическую отработку запросов от узлов с соответствующими номерами. Запросы, не прошедшие через данный фильтр, направляются в контекст *AR_Request DMA* и обрабатываются программно. Управление данным фильтром осуществляется через регистры `PhysicalRequestFilterHiSet` (110h), `PhysicalRequestFilterHiClear` (114h), `PhysicalRequestFilterLoSet` (118h), `PhysicalRequestFilterLoClear` (11Ch).

Контексты DMA

Контекст DMA, образующий независимый канал DMA, состоит из контекстной программы и регистров контроллера. *Контекстная программа* — это список команд, отрабатываемых контроллером для передачи и приема пакетов данных. Каждый контекст DMA представлен в контроллере своим *блоком регистров*, в который входят регистры `ContextControl` (управление и состояние) и `CommandPtr` (указатель на команду). В дополнение к этому контексты изохронного приема имеют свои регистры шаблонов совпадений `ContextMatch` и общий регистр масок мультиканального приема. В последующем описании указывается смещение регистров относительно начального адреса своего блока регистров.

Управляющий регистр `ContextControl` контекста представлен парой регистров `ContextControlSet` (+0h) и `ContextControlClear` (+4h), обеспечивающих установку, сброс и чтение битов и полей. Формат регистра для асинхронных контекстов приведен на рис. 25.3, форматы регистров для изохронных контекстов описаны в соответствующем разделе. Назначение полей, используемых в регистрах всех контекстов, приведено далее:

- `run` — программное разрешение (1) и запрет (0) отработки дескрипторов;
- `wake` — семафор, установкой которого программа уведомляет о добавлении нового дескриптора в контекст. Хост-контроллер обнуляет бит после выборки каждого дескриптора;
- `dead` — хост-контроллер устанавливает этот бит, обнаружив фатальную ошибку. Программный сброс бита `run` сбрасывает и этот бит;
- `active` — признак обработки дескрипторов (управляется хост-контроллером);
- `spd` — скорость, на которой был принят пакет (только для контекстов приема). Значение некорректно, если установлен бит `wake` или `active`;
- `event_code` — код события, раскрытый в табл. 25.2.

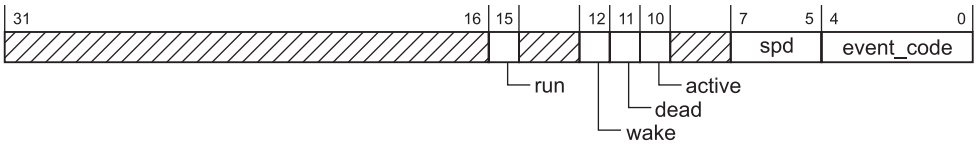


Рис. 25.3. Формат управляющих регистров асинхронных контекстов

Таблица 25.2. Коды событий для контекстов DMA

Код	События	Контексты
00	*evt_no_status*, нет индикации события	AT, AR, IT, IR
01	reserved	
02	*evt_long_packet*, длина данных в принятом пакете больше, чем размер буфера	IR
03	*evt_missing_ack*, потерян пакет подтверждения *ack*	AT
04	*evt_underrun*, недостаточно данных в FIFO, переданный пакет усечен	AT
05	*evt_overrun*, переполнение FIFO при изохронном приеме	IR
06	*evt_descriptor_read*, неисправимая ошибка при чтении дескриптора контроллером	AT, AR, IT, IR
07	*evt_data_read*, неисправимая ошибка при чтении контроллером из памяти данных для передачи	AT, IT
08	*evt_data_write*, неисправимая ошибка при записи данных в память хоста	AR, IR, IT
09	*evt_bus_reset*, признак приема пакета, синтезированного по обнаружении сброса на шине	AR
0A	*evt_timeout*, не удалась своевременно отправка пакета асинхронного ответа или изохронный контекст не смог записать число пропущенных циклов	AT, IT
0B	*evt_tcode_err*, неверный код транзакции в принятом пакете	AT, IT
0C-0D	Резерв	
0E	*evt_unknown*, неизвестная ошибка	AT, AR, IT, IR
0F	*evt_flushed*, пакет был отброшен из-за сброса шины	AT
10	Резерв	
11	*ack_complete*, пакет запроса или ответа от хоста успешно принят узлом назначения и на этом транзакция завершена. Для пакетов, не требующих подтверждений, этот признак устанавливается автоматически	AT, AR, IT, IR
12	*ack_pending*, пакет запроса от хоста принят успешно узлом назначения, субакция ответа последует позже	AT, AR
13	Резерв	
14	*ack_busy_X*, переданный пакет не принимается узлом назначения (после исчерпания лимита попыток повторов), последний полученный код подтверждения — *ack_busy_X*	AT

продолжение

Таблица 25.2 (*продолжение*)

Код	События	Контексты
15	*ack_busy_A*, переданный пакет не принимается узлом назначения (после исчерпания лимита попыток повторов), последний полученный код подтверждения — *ack_busy_A*	AT
16	*ack_busy_B*, переданный пакет не принимается узлом назначения (после исчерпания лимита попыток повторов), последний полученный код подтверждения — *ack_busy_B*	AT
17-1A	Резерв	
1B	*ack_tardy*, узел назначения не может принять пакет, поскольку его LINK приостановлен (в состоянии *suspended*)	AT
1C	Резерв	
1D	*ack_data_error*, AT-контекст принял пакет *ack_data_error* или изохронный контекст обнаружил ошибку CRC или длины данных (при приеме каждого пакета в отдельный буфер)	AT, IR
1E	*ack_type_error*, недопустимый тип транзакции	AT, AR
1F	Резерв	

Регистр `CommandPtr` (+Ch) содержит указатель на блок дескрипторов (старшие 28 бит адреса в поле `descriptorAddress`) и индикатор длины этого блока (поле `Z`). Длина (поле `Z`) указывается в 16-байтных блоках; дескрипторы выровнены по границе параграфа (младшие 4 бита — нулевые). Индикатор `Z` = 0 означает недействительность указателя — признак окончания контекстной программы. Допустимое число блоков в дескрипторе зависит от типа контекста. При инициализации контекста в регистр заносится указатель на начальный блок дескрипторов и их число в блоке. Дальнейшая программная модификация регистра допустима лишь при нулевом значении признаков `run` и `active`. Чтение регистра, в зависимости от состояния признаков `run`, `dead`, `active` и `wake`, дает различные значения указателей:

- указывает на последний исполненный, текущий или следующий дескриптор;
- указывает на блок с `Z` = 0, вызвавший прекращение активности контекста или блок, вызвавший фатальную ошибку.

Инициализация контекста начинается с проверки состояния — биты `run`, `active` и `dead` должны быть сброшены. При этом условии в регистр `CommandPtr` помещается указатель на блок дескрипторов (и длина блока), после чего можно программно установить бит `run`.

Добавление дескрипторов в список возможно в любое время. Для этого в памяти формируется дескриптор или связанный список дескрипторов, и указатель на него (и поле `Z`) помещается в адрес перехода, находящийся в последнем дескрипторе прежнего списка. После этого необходимо программно установить бит `wake` — указать контроллеру на обновление списка.

Остановка контекста выполняется программным сбросом бита `wake`, но это может и не привести к немедленной остановке. Признаком остановки контекста после сброса `run` является обнуленный бит `active`.

Контексты асинхронной передачи

Контроллер асинхронной передачи работает с двумя контекстами:

- *контекст передачи асинхронных запросов*, *AT_Request*, который используется для посылки пакетов запросов транзакций чтения, записи и блокированных, а также пакетов физического уровня;
- *контекст передачи асинхронных ответов*, *AT_Response*, который используется для посылки пакетов ответов на запросы транзакций чтения, записи и блокированных, посланные другими узлами.

Контекстная программа асинхронной передачи является списком команд, являющихся для контроллера инструкциями по сборке отправляемых пакетов. Каждый передаваемый асинхронный пакет описывается непрерывным списком команд, называемым *блоком дескрипторов*. Начальные адреса и длина этих списков фигурируют в регистре `CommandPtr`. В каждом блоке дескрипторов содержится адрес перехода (`branchAddress`), связывающий данный блок со следующим в цепочке. Контекстная программа асинхронной передачи является линейной (неветвящейся). В контекстах асинхронной передачи используются команды следующих типов:

- *OUTPUT_MORE* — не последняя команда в блоке, задающая адрес и длину буфера данных, которые следует поместить в собираемый пакет (рис. 25.4, *а*). Эта команда используется для помещения блока данных в середину пакета;
- *OUTPUT_MORE-Immediate* — аналогичная команда, буфер (1 или 2 квадлета) находится в самом теле дескриптора (рис. 25.4, *б*). Эта команда используется для формирования заголовков пакетов 1394, за которыми еще следуют данные;
- *OUTPUT_LAST* — последняя команда в блоке, задающая адрес и длину буфера данных, которые следует поместить в собираемый пакет, а также адрес перехода — адрес следующего блока дескрипторов в контекстной программе (рис. 25.4, *в*). Эта команда используется для помещения блока данных в конец пакета;
- *OUTPUT_LAST-Immediate* — аналогичная команда, но буфер (1 или 2 квадлета) находится в самом теле дескриптора (рис. 25.4, *г*). Эта команда используется для формирования коротких пакетов фиксированной длины.

Для того чтобы послать асинхронный пакет, программа хоста должна в соответствующем контексте асинхронной передачи сформировать блок команд. Возможны следующие варианты блока:

- блок, состоящий из одной команды *OUTPUT_LAST-Immediate* (`z` = 2);
- блок, содержащий цепочку команд (`z` = 4...9). Цепочка начинается с команды *OUTPUT_MORE-Immediate*, за которой следуют 0–5 промежуточных фрагментов *OUTPUT_MORE* и завершающая команда *OUTPUT_LAST*.

Отрабатывая команды *OUTPUT_MORE-Immediate* или *OUTPUT_LAST-Immediate*, контроллер подсчитывает и автоматически вставляет в пакет CRC-код заголовка. Отрабатывая команду *OUTPUT_LAST*, контроллер вставляет в пакет CRC, подсчитанный для всех фрагментов поля данных. По отработке команды контроллер помещает состояние выполнения (из регистра управления соответствующим контекстом) в дескриптор последней команды.

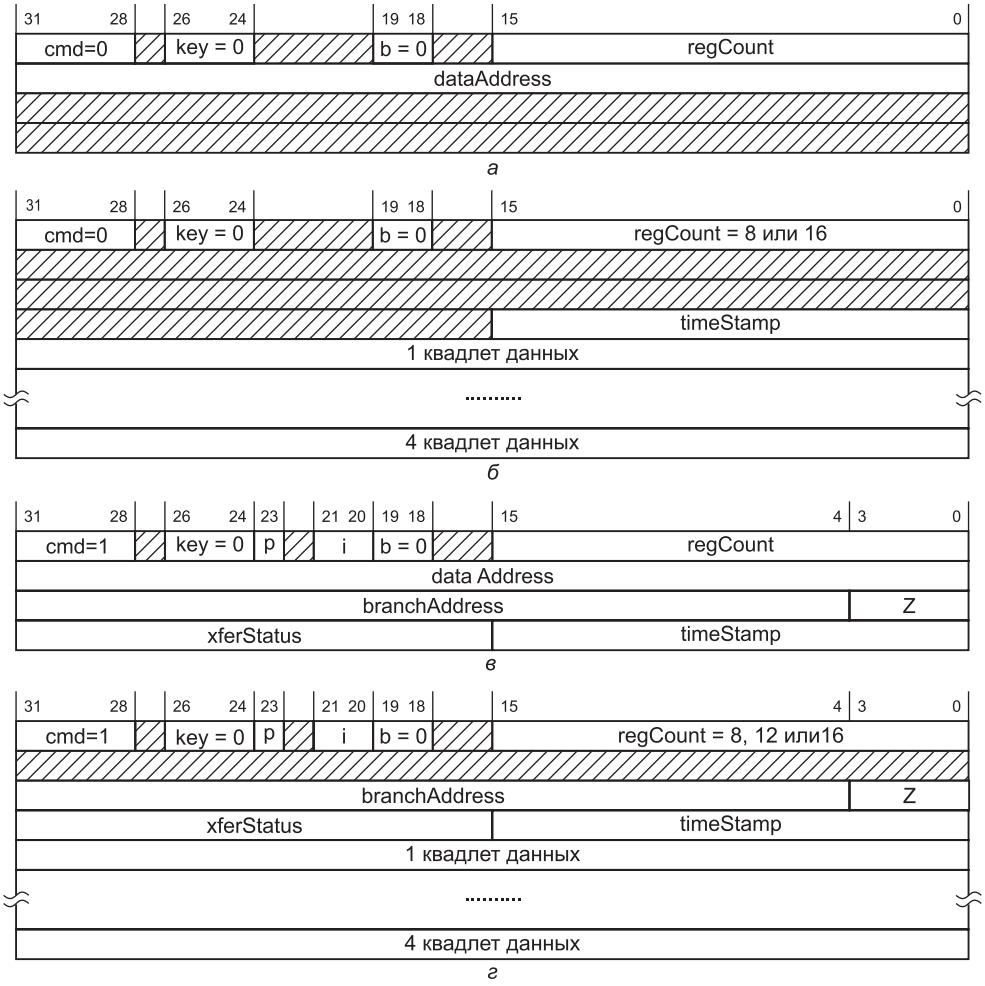


Рис. 25.4. Форматы дескрипторов команд асинхронной передачи: *а* — OUTPUT_MORE; *б* — OUTPUT_MORE-Immediate; *в* — OUTPUT_LAST; *г* — OUTPUT_LAST-Immediate

Назначение полей в дескрипторах команд асинхронной передачи приведено далее:

- `cmd` — код команды;
- `key` — ключ команды (можно рассматривать как расширение кода);
- `b` (branch control) — управление переходом: 0 — нет указателя перехода, 1 и 2 — недопустимо, 3 — дескриптор содержит указатель перехода;
- `reqCount` — длина буфера (в байтах);
- `dataAddress` — адрес буфера данных (нет требования выравнивания);

- `timeStamp` — метка времени, имеющая разное назначение:
  - в пакете запросов (кроме *ping*) — время фактической отправки пакета (3 младших бита `second_count` и 13 бит `cycle_count`), устанавливаемое аппаратно при передаче;
  - в пакетах ответов — время, позже которого пакет передавать не нужно (устанавливается программно);
  - в пакетах ping-запросов — промежуток времени, измеренный от момента передачи последнего бита до начала приема ответа (в тактах частоты 49,152 МГц);
- `i` (interupt control) — управление прерываниями: 0 — нет прерываний, 1 — прерывание только при неполучении квитанции *ack_complete* или *ack_pending*; 2 — недопустимо, 3 — прерывание по исполнении команды;
- `branchAddress` — адрес перехода, указывающий на начало следующего блока дескрипторов;
- `Z` — длина следующего блока дескрипторов (в 16-байтных блоках). Нулевая длина является указанием на останов контекстной программы;
- `xferStatus` — состояние передачи, значения младших 16 бит регистра управления контекстом на момент завершения команды;
- `p` (Ping Timing) — признак пакета *ping*.

Форматы данных, которые подготавливает программа хоста для посылки асинхронных пакетов, соответствуют форматам пакетов транзакций IEEE 1394 (см. главу 18), но с некоторыми особенностями:

- заголовки и поля данных асинхронных пакетов формируются без CRC-кодов. Место под эти коды не резервируется, контроллер подсчитывает и добавляет эти коды в процессе передачи;
- в заголовках асинхронных пакетов поля идентификаторов узлов источника и назначения поменяны местами (рис. 25.5, *а*). При этом вместо 16-битного идентификатора узла-источника подставляется поле, в котором указываются скорость передачи пакета (`spd`) и указание для подстановки номера шины (`srcBusID`: 0 — номер шины принимается 3FFh, 1 — берется из поля `busNumber` регистра `Node_ID`). Контроллер передаст пакет в формате шины 1394, подставив номер шины и физический идентификатор узла;
- пакеты физического уровня формируются программой в соответствии с рис. 25.5, *б*. Здесь введен управляющий квадлет, определяющий скорость и тип транзакци (`tcode` = Eh). Первый и второй квадлеты пакета физического уровня полностью формируются программой (второй должен быть инверсией первого) и контроллером передаются без изменений, CRC не формируется. Длина передаваемого пакета будет всегда 2 квадлета, независимо от длины, указанной в дескрипторе команды. Такой особенный способ обработке дескриптора команды *OUTPUT_LAST-Immediate* контроллеру предписывает значение Eh кода типа транзакции.

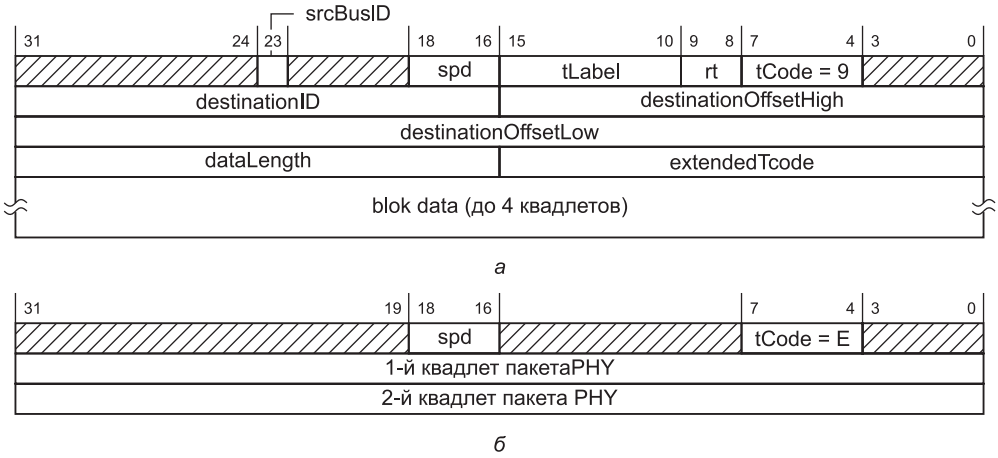


Рис. 25.5. Форматы данных для контекстов асинхронной передачи: *а* — начало заголовка асинхронного пакета; *б* — формат данных для физического пакета

Контекст асинхронного приема

Контроллер асинхронного приема работает с двумя контекстами:

- *контекст приема асинхронных запросов, AR_Request*, который используется для приема пакетов запросов транзакций чтения, записи и блокированных, а также пакетов физического уровня (кроме пакетов самоидентификации, принятых в фазе инициализации шины);
- *контекст приема асинхронных ответов, AR_Response*, который используется для приема пакетов ответов на запросы транзакций чтения, записи и блокированных, посланных к другим узлам.

Контекстная программа асинхронного приема является списком команд, описывающих цепочку буферов, которыми контроллер заполняет принятыми асинхронными пакетами, не обрабатываемые блоком физической обработки. Эта программа является линейной. Цепочка буферов, описанная контекстной программой, для контроллера является логически непрерывной областью памяти. Границы принятых пакетов и границы буферов не связаны друг с другом (кроме начала первого пакета, совпадающего с началом первого буфера). Такой режим приема пакетов называется *bufferFill mode*. Извлечением пакетов из этого буфера занимается программа хоста.

В контекстах асинхронного приема используется только один тип команды — *INPUT_MORE*, формат ее дескриптора приведен на рис. 25.6. Назначение полей в дескрипторе команды приведено далее:

- `cmd` — код команды (0010b);
- `s` (status control) — управление статусом. Программа должна устанавливать этот бит, контроллер записывает состояние независимо от значения бита;
- `key` — ключ команды (0);

- `i` (interupt control) — управление прерываниями: 0 — нет прерываний, 3 — прерывание по исполнении команды (заполнении указанного в ней буфера), 1 и 2 — недопустимо. Прерывания по приему пакетов данным полем не управляются (для них есть свои биты в регистре запросов и масок прерываний контроллера);
- `b` (branch control) — управление переходом: 3 — дескриптор содержит указатель перехода, остальные значения недопустимы;
- `reqCount` — длина буфера (в байтах). Буфер должен вмещать не менее одного пакета максимальной длины (его размер определяется в `bus_info_block`), включая 5 квадлетов его заголовка и концевика. При этом любой пакет может пересекать не более одной границы буфера;
- `dataAddress` — адрес буфера данных (должен быть выровнен по границе квадлета);
- `branchAddress` — адрес перехода, указывающий на начало следующего блока дескрипторов;
- `Z` — длина следующего блока дескрипторов (допустимо `Z` = 0 или 1). Нулевая длина является указанием на останов контекстной программы;
- `xferStatus` — состояние передачи, значения младших 16 бит регистра управления контекстом на момент модификации поля `resCount`;
- `resCount` — счетчик байтов свободного места в буфере.

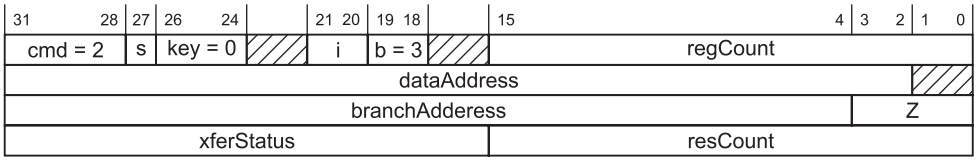


Рис. 25.6. Формат дескриптора команды *INPUT_MORE*

Программа хоста должна инициализировать контекстные программы и следить за наличием свободного места в буферах. Контроллер укладывает в буферы только корректно принятые пакеты. Пакеты, принятые с ошибками CRC заголовков и данных и с ошибками FIFO (переполнение), для программы хоста невидимы. По каждому успешному приему пакета контроллер модифицирует поля `resCount` и `xferStatus` в дескрипторах, в буферы которых попал пакет.

На каждый успешно принятый асинхронный пакет контроллер автоматически отвечает соответствующим пакетом квитирования. На пакет, принятый с ошибкой, он отвечает подтверждением *ack_busy_X*, вынуждающим отправителя повторить передачу. Если из-за сброса шины контроллер не успевает послать пакет подтверждения, то успешно принятый (но не подтвержденный) пакет выбрасывается из буфера (поле `resCount` по приему этого пакета не будет модифицировано).

С асинхронным приемом могут быть связаны *события прерываний*, отраженные в регистрах запросов и масок:

- `RQPkt` и `RSPkt` — прерывания по приему пакетов в контексты *AR_Request* и *AR_Response* соответственно;
- `ARRQ` и `ARRS` — прерывания по заполнении буферов (в контекстах *AR_Request* и *AR_Response* соответственно), для которых в дескрипторах установлено поле `i` = = 3.

Форматы данных в буферах контекстов асинхронного приема соответствуют форматам пакетов, определенных стандартом IEEE 1394. В отличие от буферов асинхронной передачи здесь идентификаторы источника и назначения присутствуют оба и находятся на своих местах. Однако и здесь есть ряд особенностей:

- квадлеты с CRC-кодами заголовка и поля данных в буферах отсутствуют;
- вместо последнего поля CRC (заголовка или данных, смотря по формату) в буфер помещается *концевик* (packet trailer) с меткой времени и состоянием контекста (рис. 25.7, *а*):
  - `timeStamp` — метка времени приема пакета (3 младших бита `second_count` и 13 бит `cycle_count`);
  - `xferStatus` — состояние приема, значения младших 16 бит регистра управления контекстом на момент приема пакета;
- принятые PHY-пакеты помещаются в четыре квадлета буфера в соответствии с рис. 25.7, *б*.

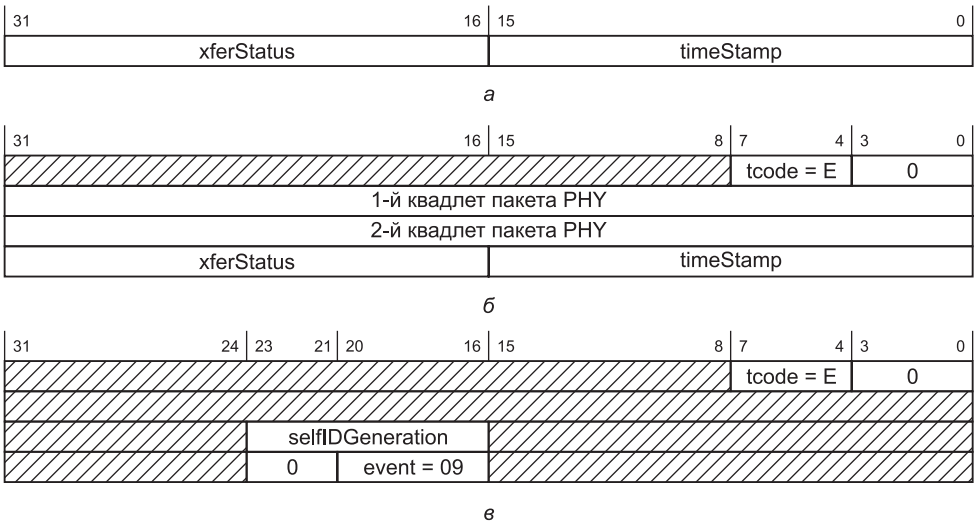


Рис. 25.7. Форматы данных в буферах асинхронного приема: *а* — концевик пакета; *б* — пакет физического уровня; *в* — синтезированный пакет сброса шины

Пакеты физического уровня, принятые с шины, попадают в контекст *AR_Request* (сюда попадают и пакеты самоидентификации, принятые не в фазе инициализации шины). Признаком PHY-пакетов являются длина в два квадлета и неверный

CRC-код. Корректность пакета (соответствие прямой и инверсных копий информации) проверяется программой хоста. Прием пакетов физического уровня разрешается установкой `rcvPhyPkt` = 1 в регистре `LinkControl`.

Сброс на шине отмечается помещением в буфер контекста *AR_Request* (если там есть место) синтезированного *пакета сброса шины*. Это позволяет программе определить, какие пакеты пришли до сброса шины, а какие — после. Такая пометка необходима, поскольку после сброса могут поменяться физические идентификаторы узлов. Формат пакета приведен на рис. 25.7, *в*, назначение полей приведено далее:

- `tcode` — метка транзакции (Eh), указывающая на PHY-пакет;
- `selfIDGeneration` — номер генерации, соответствующий новому значению поля `selfIDGeneration` регистра `selfIDCount` на момент создания пакета;
- `event` — код события (09h), по которому идентифицируется данный пакет.

Контексты изохронной передачи

Для передачи изохронных пакетов контроллер может иметь от 4 до 32 контекстов, с каждым из которых связан свой изохронный канал шины. Число контекстов можно определить, записав FFFF FFFF в регистр масок прерываний от контекстов и прочитав его значение: единицы останутся только в битах, соответствующих существующим контекстам. Контексты изохронных передач отличаются от контекстов асинхронных тем, что требуют привязки начала передачи к определенному моменту времени, а также реакции на невозможность своевременной передачи пакета.

Контекстные программы изохронной передачи содержат команды описания пакетов, передаваемых данным контекстом в изохронном периоде каждого цикла шины. Контекстная программа может быть ветвящейся, и для каждого пакета указываются два адреса перехода:

- *адрес перехода* в случае нормальной передачи пакета (`branchAddress`);
- *адрес пропуска* — адрес перехода в случае пропуска цикла (`skipAddress`). По этому адресу программа переходит в случае, если в данном цикле передача пакета не удалась.

Каждый пакет описывается блоком из 1–8 дескрипторов, в которых используются следующие команды:

- *OUTPUT_MORE* — не первая и не последняя команда в блоке, задающая адрес и длину буфера данных, которые следует поместить в собираемый пакет (рис. 25.8, *а*). Эта команда используется для помещения блока данных в середину пакета;
- *OUTPUT_MORE-Immediate* — первая команда в блоке, используемая для передачи заголовка пакета с ненулевой длиной данных. Буфер (2 квадлета) находится в самом теле дескриптора (рис. 25.8, *б*). В команде задается адрес пропуска;
- *OUTPUT_LAST* — последняя команда в блоке, задающая адрес и длину буфера данных, которые следует поместить в собираемый пакет, а также адрес перехо-

да или пропуска (рис. 25.8, *в*). В дескриптор команды при передаче контроллер помещает состояние контекста и метку времени. Команда используется двояко:

- для помещения данных (но не заголовка) из буфера в конец пакета, при этом длина буфера ненулевая и в команде содержится *адрес пропуска*. В этом случае данной команде должна предшествовать команда *OUTPUT_MORE* или *OUTPUT_MORE-Immediate*;
- как признак отсутствия пакета для передачи в данном цикле, при этом указывается нулевая длина буфера и в команде содержится *адрес перехода*. В этом случае команде не должны предшествовать команды *OUTPUT_MORE* или *OUTPUT_MORE-Immediate*;

- *OUTPUT_LAST-Immediate* — команда, используемая для формирования пакетов с нулевой длиной поля данных; буфер (2 квадлета) находится в самом теле дескриптора (рис. 25.8, *г*). Для такого пакета переход в любом случае происходит по одному и тому же адресу. В дескриптор команды при передаче контроллер помещает состояние контекста и метку времени;
- *STORE_VALUE* — команда, обеспечивающая программный мониторинг исполнения контекста без использования прерываний. Эта команда может быть введена в начало блока дескрипторов, при ее исполнении контроллер выполняет запись определенного слова данных по указанному адресу в случае успешной отработки блока. В дескрипторе указываются адрес пропуска, адрес записываемых данных и сами данные (рис. 25.8, *д*). В случае пропуска цикла (перехода по адресу пропуска) запись данных не производится.

Назначения полей в дескрипторах команд изохронной передачи следующие:

- `cmd` — код команды;
- `key` — ключ команды (расширение кода);
- `b` (branch control) — управление переходом: 0 — для *OUTPUT_MORE* или *OUTPUT_MORE-Immediate* , 3 — для *OUTPUT_LAST* или *OUTPUT_LAST -Immediate*, 1 и 2 — недопустимо;
- `reqCount` — длина буфера (в байтах);
- `dataAddress` — адрес буфера данных (нет требования выравнивания) или адрес записи для команды мониторинга;
- `i` (interrupt control) — управление прерываниями: 0 — нет прерываний, 1 и 2 — недопустимо, 3 — прерывание при переходе по адресу пропуска;
- `branchAddress` — адрес нормального перехода, указывающий на начало следующего блока дескрипторов;
- `skipAddress` — адрес перехода по пропуску;
- `Z` — длина следующего блока дескрипторов. Нулевая длина является указанием на останов контекстной программы;
- `xferStatus` — состояние передачи, в котором содержатся значения младших 16 бит регистра управления контекстом на момент завершения команды;
- `timeStamp` — метка времени помещения пакета в FIFO-буфер (3 младших бита `second_count` и 13 бит `cycle_count`), устанавливается аппаратно при передаче;

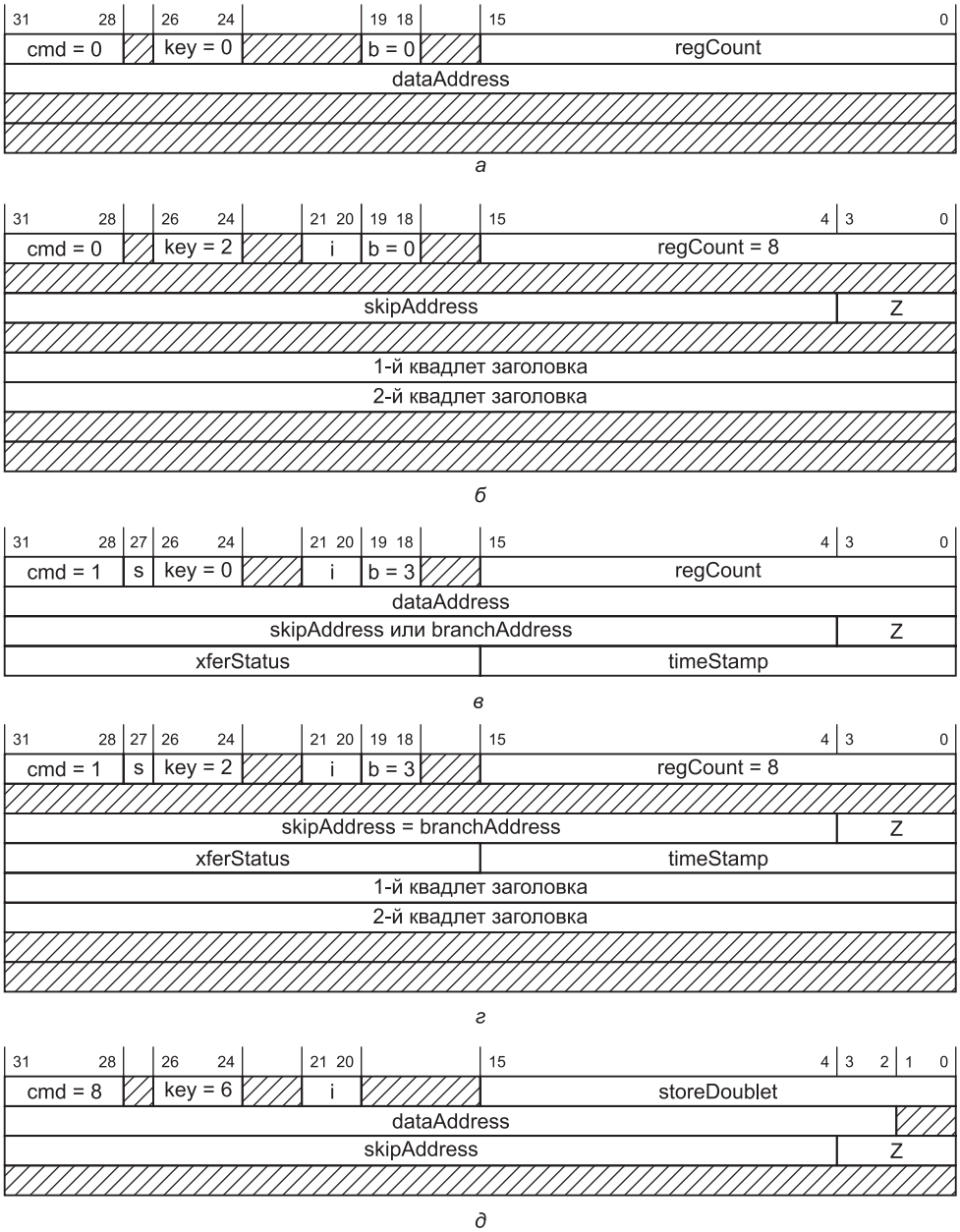


Рис. 25.8. Форматы дескрипторов команд изохронной передачи: *а* — OUTPUT_MORE; *б* — OUTPUT_MORE-Immediate; *в* — OUTPUT_LAST; *г* — OUTPUT_LAST-Immediate; *д* — STORE_VALUE

- `s` (status control) — признак модификации полей `xferStatus` и `timeStamp`, устанавливается контроллером;

- `storeDoublet` — младшее слово записываемых данных мониторинга (старшее слово — нулевое).

В контекстной программе должны быть описаны пакеты для каждого цикла. Если для данного цикла нет данных, в контексте должен быть описан пакет нулевой длины (командой *OUTPUT_LAST-Immediate*). Если канал в каком-то цикле не должен передавать пакет, это описывается единственной командой *OUTPUT_LAST* с нулевой длиной буфера. Останов контекстной программы происходит при переходе, для которого указан `z` = 0.

Для определения цикла, с которого должна активизироваться работа контекста, в управляющем регистре контекста (рис. 25.9) введены дополнительные поля:

- `cycleMatchEnable` — разрешение активизации контекста по началу определенного цикла (только для изохронных контекстов);
- `cycleMatch` — указатель цикла, в котором активизируется контекст (только для изохронных контекстов).

Остальные поля имеют то же назначение, что и для асинхронных контекстов (см. выше).

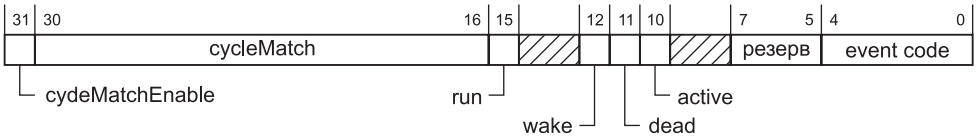


Рис. 25.9. Формат управляющих регистров изохронных контекстов

Каждая контекстная программа обычно связана со своим изохронным каналом, номер канала присутствует в заголовках передаваемых пакетов (формируется программно).

Пакеты всех контекстов DMA изохронной передачи укладываются в один и тот же буфер FIFO, из которого они выбираются LINK-уровнем для передачи в шину. Пакеты соседних циклов в FIFO отделяются друг от друга специальным разделителем. Контроллер может укладывать пакеты в FIFO с опережением до двух циклов. В каждом цикле шины LINK-уровень из FIFO забирает на передачу пакеты до очередного разделителя, а контроллер DMA укладывает в FIFO очередные пакеты из контекстов. Возможна ситуация, когда за очередной цикл LINK-уровень не может выбрать все пакеты до разделителя. При этом контроллер DMA не может поместить в буфер очередные пакеты — это и есть пропуск цикла. Как правило, такая ситуация возникает из-за сброса шины. Ветвления контекстной программы позволяют отрабатывать эту ситуацию различными способами, как это иллюстрирует рис. 25.10:

- пропускать передачи пакета в случае пропуска цикла (контекст A) — адрес пропуска и адрес перехода указывают на один и тот же блок дескрипторов пакета для следующего цикла;

- настойчиво передавать каждый пакет (контекст B) — адрес пропуска в дескрипторе указывает на начало того же блока дескрипторов. При этом, естественно, появляется задержка доставки данных;
- в случае пропуска любого цикла послать (настойчиво) специальный завершающий пакет и остановить контекстную программу (контекст C) — адрес пропуска во всех дескрипторах цепочки указывает на дескриптор завершающего пакета;
- останавливать передачу по пропуску любого цикла (контекст D) — адрес пропуска во всех дескрипторах цепочки указывает на нуль-дескриптор.

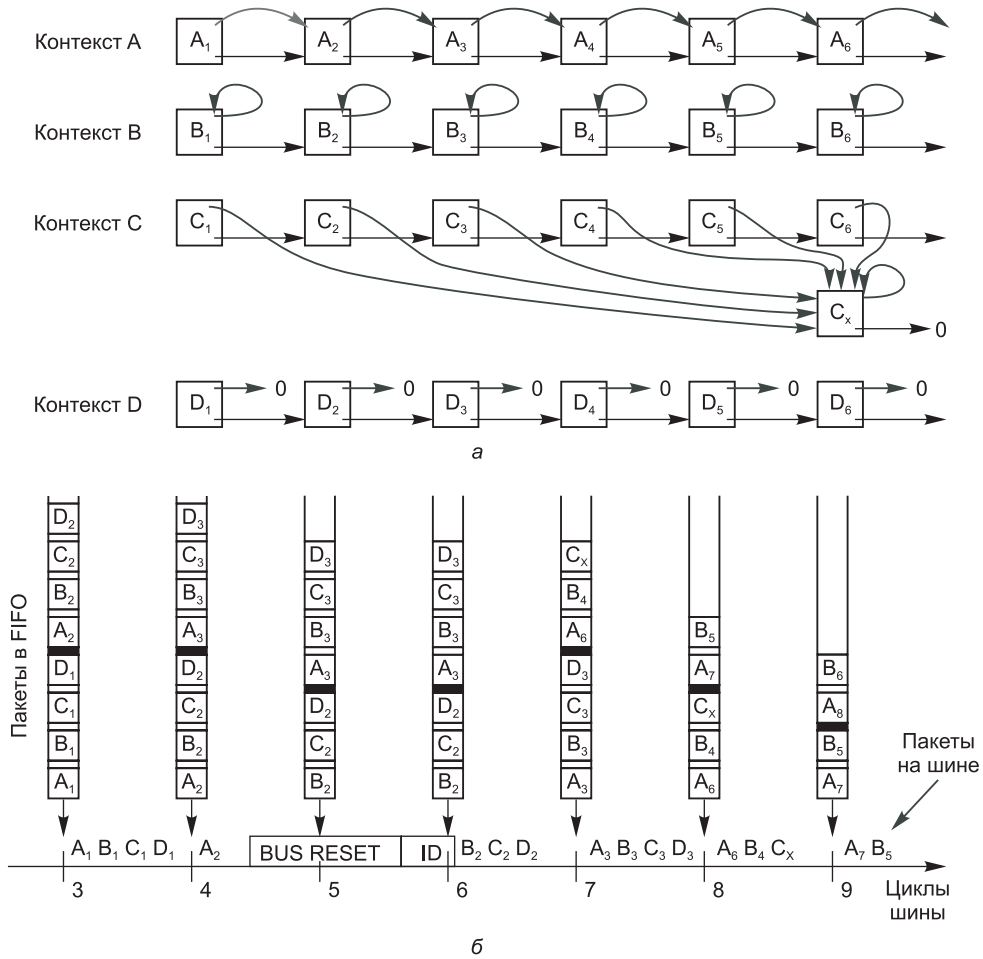


Рис. 25.10. Поведение контекстов изохронной передачи при пропуске цикла из-за сброса шины

Квадраты на рис. 25.10, *а* обозначают блоки дескрипторов контекста; выходящие из них прямые стрелки (снизу) указывают на нормальные переходы, изогнутые (сверху) — на переходы по пропускам. На рис. 25.10, *б* показаны состояния FIFO-буфера на моменты начала каждого цикла. Здесь показаны пакеты, собранные по соответствующим блокам дескрипторов, и разделители циклов. Как видно из рисунка, сброс на шине (BUS RESET) и процесс идентификации дерева и узлов (ID) прерывают передачу изохронных пакетов. В результате этого по шине передаются изохронные пакеты следующим образом:

- для контекста A: A_1, A_2, <пропуск двух циклов>, A_3, A_6, A_7, — из-за сброса пропущены два цикла и потеряны (позже!) два пакета;
- для контекста B: B_1, <пропуск двух циклов>, B_2, B_3, B_4, B_5, — из-за сброса пропущены два цикла (задержка доставки), но пакеты не потеряны;
- для контекста C: C_1, <пропуск двух циклов>, C_2, C_3, C_x (специальный пакет, сигнализирующий останов), останов — из-за сброса пропущены два цикла, чуть позже передача останавливается с предварительным уведомлением слушателей;
- для контекста D: D_1, <пропуск двух циклов>, D_2, D_3, останов — из-за сброса пропущены два цикла, чуть позже передача останавливается без предупреждения.

Если во время передачи FIFO-буфер переопустошается (underrun), то передаваемый пакет теряется (в его дескрипторе поле состояния не обновляется), а программы всех оставшихся контекстов продвигаются по ветке пропуска цикла.

Программа хоста помещает пакеты изохронной передачи данных в буферы, описанные блоками дескрипторов. Формат пакетов данных (рис. 25.11) почти соответствует формату потоковых пакетов IEEE 1394 (см. главу 18). Отличие заключается в переносе поля длины пакета `data_length` во второй квадлет ради помещения в первый квадлет кода скорости `spd` (эта информация при передаче нужна раньше всего). Поля CRC заголовка и данных контроллер строит сам при передаче.

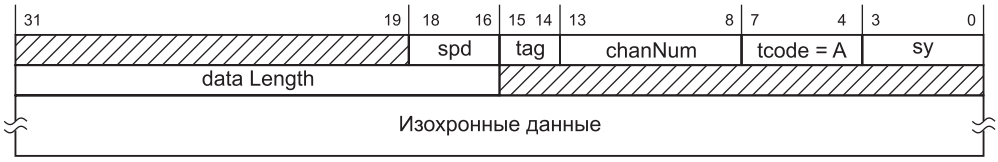


Рис. 25.11. Формат изохронного пакета в буфере передачи

Контекст изохронного приема

Для приема изохронных пакетов контроллер может иметь от 4 до 32 контекстов, каждый из которых обеспечивает прием одного канала. Один из этих контекстов можно запрограммировать на мультиканальный прием.

Контекстная программа содержит список буферов, заполняемых принимаемыми пакетами. Каждый контекст может быть сконфигурирован на определенный способ обработки принятых пакетов. В буфер пакеты могут помещаться как целиком, так и освобожденные от заголовков и концевиков, содержащих метку времени и состояние контекста. По способу заполнения буферов возможны следующие режимы:

- *packet-per-buffer* — укладывать в каждый буфер по одному пакету;
- *buffer-fill* — соединять пакеты в поток, полностью заполняющий каждый буфер цепочки;
- *dual-buffer* — соединять первые части поля данных пакетов в поток, укладываемый в одну цепочку буферов, а вторые части — в другой поток и другую цепочку. Этот режим позволяет при приеме из пакетов выделять два потока (например, аудио и видео).

В контекстных программах используются следующие команды:

- *INPUT_MORE* (рис. 25.12, *а*) — команды, используемые в режимах *packet-per-buffer* и *buffer-fill*. В дескрипторах этих команд задается адрес перехода и описывается один буфер данных;
- *INPUT_LAST* — команды аналогичного формата, используемые как обязательное завершение блока дескрипторов в режиме *packet-per-buffer*;
- *DUALBUFFER* (рис. 25.12, *б*) — команды, используемые в одноименном режиме. В дескрипторах этих команд кроме адреса перехода задается пара буферов, в которые раскладывается принятый пакет.

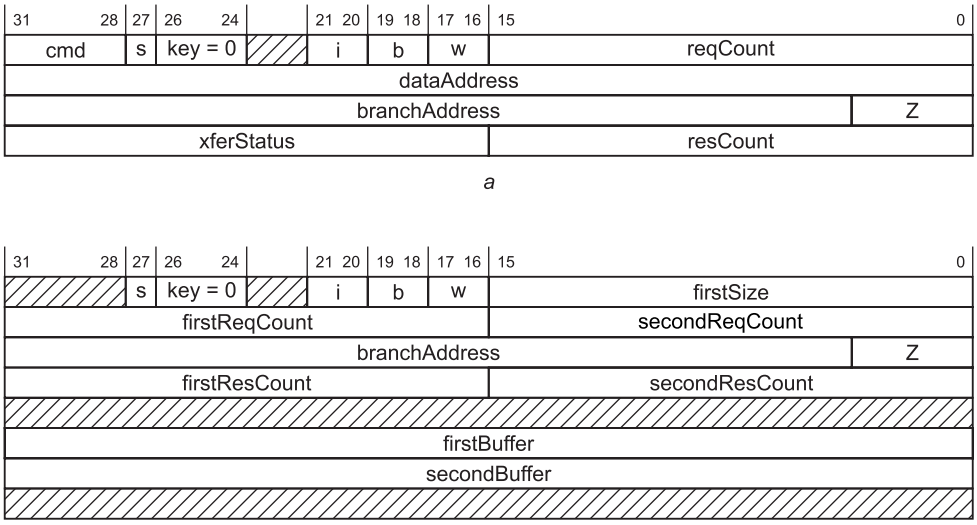


Рис. 25.12. Формат дескрипторов команд изохронного приема: *а* — INPUT_MORE и INPUT_LAST; *б* — DUALBUFFER

Назначения полей дескрипторов следующие:

- `cmd` — код команды: 0 — *DUALBUFFER*, 2 — *INPUT_MORE*, 3 — *INPUT_LAST*;
- `s` (status control) — управление статусом: в режиме *packet-per-buffer* разрешает запись состояния, в остальных режимах должно быть «1»;
- `key` — ключ команды (0);
- `i` (interupt control) — управление прерываниями: 0 — нет прерываний, 3 — прерывание по исполнении команды, 1 и 2 — недопустимо;
- `b` (branch control) — управление переходом: 3 — дескриптор содержит указатель перехода (для *INPUT_LAST* и *DUALBUFFER*), 0 — не содержит (для *INPUT_MORE*), остальные значения недопустимы;
- `w` (wait control) — управление ожиданием совпадения поля `sync` с шаблоном: 3 — ожидать совпадения, 0 — не ожидать. В режиме *packet-per-buffer* значение «3» используется только в первом дескрипторе блока, в остальных режимах это значение применимо к любым дескрипторам блока;
- `reqCount` — длина буфера (в байтах). Буфер должен вмещать не менее одного пакета максимальной длины (его размер определяется в `bus_info_block`), включая 5 квадлетов его заголовка и концевика. При этом любой пакет может пересекать не более одной границы буфера;
- `dataAddress` — адрес буфера данных (должен быть выровнен по границе квадлета);
- `branchAddress` — адрес перехода, указывающий на начало следующего блока дескрипторов;
- `Z` — длина следующего блока дескрипторов (допустимо `Z` = 0 или 1). Нулевая длина является указанием на останов контекстной программы;
- `xferStatus` — состояние передачи, значения младших 16 бит регистра управления контекстом на момент модификации поля `resCount`;
- `resCount` — счетчик байтов свободного места в буфере;
- `firstSize` — размер части пакета, укладываемой в первый буфер в режиме *dual buffer*;
- `firstReqCount` — размер первого буфера;
- `secondReqCount` — размер второго буфера;
- `firstResCount` — свободное место в первом буфере;
- `secondResCount` — свободное место во втором буфере;
- `firstBuffer` — адрес первого буфера;
- `secondBuffer` — адрес второго буфера.

В контекстах изохронного приема управляющий регистр `ContextControl` (рис. 25.13, *а*) имеет дополнительные биты:

- `bufferFill` — признак режима соединения пакетов в единый поток;

- `isochHeader` — разрешение записи заголовка пакета в буфер;
- `cycleMatchEnable` — разрешение ожидания запрошенного номера цикла;
- `multiChanMode` — признак многоканального приема;
- `dualBufferMode` — признак режима двойной буферизации. При установленном бите недопустима установка бит `bufferFill` и `multiChanMode`.

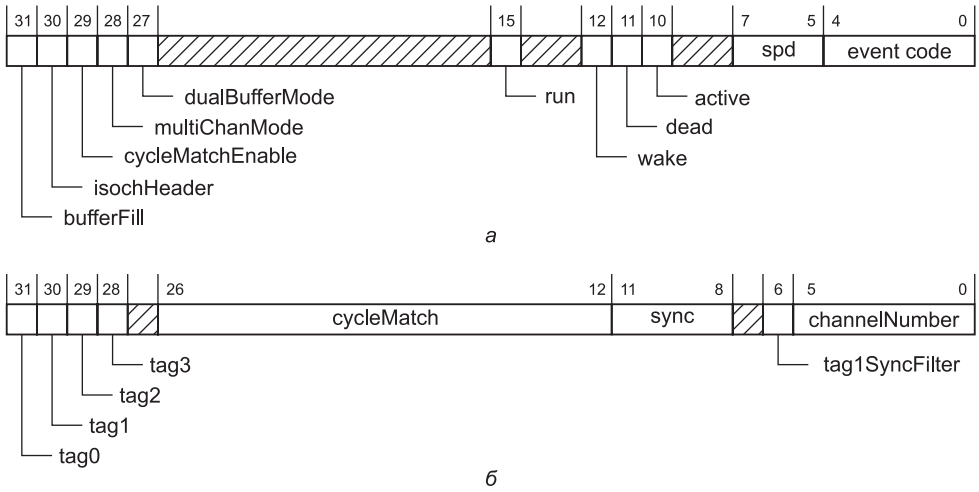


Рис. 25.13. Формат регистров контекстов изохронного приема: *а* — ContextControl, *б* — ContextMatch

Один из контекстов изохронного приема можно запрограммировать на мультиканальный прием (установкой бита `multiChanMode` в его регистре `ContextControl`). Каналы, данные которых должны приниматься в этот контекст, отмечаются единицами в соответствующих им битах 64-битного регистра `IRMultiChanMask` (старший бит соответствует каналу 63). Биты данного регистра можно устанавливать и сбрасывать через регистры `IRMultiChanMaskHiSet` (070h), `IRMultiChanMaskLoSet` (078h), `IRMultiChanMaskHiClear` (074h) и `IRMultiChanMaskLoClear` (07Ch).

Регистр `ContextMatch` (+10h) используется для активации контекста в указанном цикле, фильтрации пакетов по полю тега (`tag`) и ожидания пакета с указанным значением поля синхронизации (`sy`). Запись в регистр допустима лишь при неактивном состоянии контекста. Формат регистра приведен на рис. 25.13, *б*; назначения полей следующие:

- `tag3` — разрешение приема пакетов с тегом `tag` = 11;
- `tag2` — разрешение приема пакетов с тегом `tag` = 10;
- `tag1` — разрешение приема пакетов с тегом `tag` = 01;
- `tag0` — разрешение приема пакетов с тегом `tag` = 00;

- `cycleMatch` — 15-битное поле, задающее значение двух старших бит счетчика секунд и 13-битного счетчика циклов, при котором контекст активизируется (если в его управляющем регистре бит `cycleMatchEnable` = 1).
- `sync` — значение, которое ожидается в поле `sy` принимаемых пакетов, когда в дескрипторе команды установлено поле `w` = 3;
- `tag1SyncFilter` — признак дополнительной фильтрации по полю `sy`. При установке этого бита и `tag1` = 1 пакеты с тегом `tag` = 01 будут приниматься лишь при `sy` = 00xx; пакеты с другими тегами фильтруются по обычным правилам;
- `channelNumber` — номер изохронного канала, с которым связан данный контекст.

С каждым номером канала должно быть связано не более одного контекста приема; в противном случае неизвестно, в какой контекст попадет принятый пакет из FIFO-буфера. Если номер канала для какого-либо контекста используется в мультиканальном режиме другого контекста, то пакет попадет в контекст с мультиканальным приемом.

Формат данных в приемных буферах зависит от режима приема:

- в режиме *bufferFill* с заголовком формат соответствует формату изохронного пакета, но поле CRC заголовка отсутствует, а вместо CRC данных записывается концевик с меткой времени и состоянием контекста;
- в режиме *bufferFill* без заголовка в буферы укладываются только поля данных из пакетов;
- в режимах *Packet-per-buffer* и *dual-buffer* с заголовками буфер начинается с квадлета, в котором младшие 16 бит содержат метку времени (старшие 16 бит не используются). Далее следует пакет, освобожденный от полей CRC и заполнителя данных (его длина может быть не выровненной по границе квадлета);
- в режимах *Packet-per-buffer* и *dual-buffer* без заголовков в буферы укладываются только поля данных.

Блок физических запросов

Ряд запросов к памяти и регистрам узла отрабатывается на аппаратном уровне OHC, без привлечения программного обеспечения хоста. Отработкой этих запросов занимается блок физических запросов, в котором имеется контроллер DMA принимающий с шины запросы на транзакции, и контроллер DMA, посылающий на них ответы. Запросы в зависимости от смещения, указанного в адресе назначения, отрабатываются по-разному.

Если смещение попадает в область нижних адресов узла, то запрос направляется к памяти хоста. При этом смещение трактуется как физический адрес памяти в пространстве хоста. В этой области физически (обменом с памятью по каналу DMA) отрабатываются запросы чтения, записи и блокированных транзакций с нулевым кодом расширенной команды, прошедшие фильтр по иден-

тификатору источника (см. выше). Остальные запросы будут переданы в контекст *AR DMA Request*.

Запросы блокированных транзакций *compare_swap* и чтения квадлета по адресам автономных регистров диспетчера изохронных ресурсов направляются к этим регистрам. На другие запросы по этим адресам OHC отвечает квитанцией ошибки типа запроса (*ack_type_error*). К адресам автономных регистров относятся следующие:

- FFFFF000021Ch — регистр идентификатора диспетчера шины `BusManagerID`;
- FFFFF0000220h — регистр доступной полосы пропускания `BandwidthAvailable`;
- FFFFF0000224h — старший квадлет регистра доступных каналов, `ChannelsAvailableHi`;
- FFFFF0000228h — младший квадлет регистра доступных каналов, `ChannelsAvailableLo`.

Запросы чтения квадлета по специальным адресам памяти конфигурации направляются к регистрам OHC. На другие запросы по этим адресам, а также при некорректности образа памяти OHC отвечает квитанцией ошибки типа запроса *ack_type_error*. К специальным адресам памяти конфигурации относятся следующие:

- FFFFF0000400h — заголовок памяти конфигурации (`Config ROM header`), запрос направляется к регистру `ConfigROMhdr`;
- FFFFF0000404h — идентификатор шины, первый квадлет `Bus_Info_Block`, запрос направляется к регистру `BusID`;
- FFFFF0000408h — опции шины, второй квадлет `Bus_Info_Block`, запрос направляется к регистру `BusOptions`;
- FFFFF000040Ch и FFFFF0000410h — глобальный идентификатор, 3-й и 4-й квадлеты `Bus_Info_Block`, запрос направляется к регистрам `GlobalIDHi` и `GlobalIDLo`.

Запросы чтения памяти конфигурации по адресам FFFFF0000414–FFFFF00007FFh направляются к памяти хоста. Память конфигурации отображается на 1-килобайтный блок системной памяти в соответствии со значением регистра `ConfigROMmap`. При некорректности образа памяти (в регистре `HCControl` бит `BIBimageValid` == 0) OHC на эти запросы отвечает квитанцией ошибки *ack_type_error*.

Если принятый пакет запроса содержит ошибку CRC поля данных или имеет неправильную длину, контроллер ответит на него квитанцией *ack_busy_** (вместо * подставляется буква *A*, *B* или *X* в соответствии с используемым однофазным или двухфазным протоколом). Это вынудит инициатора повторить запрос.

Генерируемые ответы на аппаратно-обрабатываемые запросы содержат метку транзакции, соответствующую запросу, и адрес назначения, соответствующий

адресу источника запроса. Если на ответ контроллер получает квитанцию *ack_busy*, то число повторов ограничивается полем `MaxPhysRespRetries` регистра `ATRetries`.

Запросы записи могут выполняться как *отправленные записи* (Posted Writes) — подтверждение на них посылается сразу, не дожидаясь фактической записи. Если выполнение фактической записи не удается, то в регистре `IntEvent` устанавливается бит `PostedWriteErr`, а 48-битное смещение из адреса неудавшегося шинного запроса фиксируется в регистрах `PostedWriteAddressLo`, `PostedWriteAddressHi` (038h, 03Ch).

Запросы прерываний при нормальном выполнении аппаратно-обрабатываемых запросов *не вырабатываются*; прерыванием сигнализируется только ошибка при выполнении отправленной записи и невозможность доставки ответа на блокированные транзакции.

После *сброса на шине* все аппаратно-обрабатываемые запросы, на которые OHC должен был ответить, аннулируются. После сброса OHC автоматически начинает отрабатывать только запросы к регистрам диспетчера изохронных ресурсов (в CSR), остальные запросы будут отрабатываться только после инициализации фильтров.

Прием пакетов самоидентификации

Во время фазы инициализации после сброса шины контроллер автоматически принимает *пакеты самоидентификации узлов* и специальным каналом DMA помещает их в буфер. Положение буфера в памяти хоста задается регистром `SelfIDBuffer` (064h). Число пакетов, принятых после очередного сброса, можно прочитать в регистре `SelfIDCount` (068h). Автоматический прием пакетов самоидентификации разрешается битом `rcvSelfID` регистра `LinkControl`.

В буфере первый квадлет содержит *номер генерации* (`selfIDGeneration`) — счетчик числа принятых потоков пакетов самоидентификации и *метку времени* (`timeStamp`), состоящую из трех младших бит счетчика секунд и 13-битного счетчика циклов. За первым квадлетом помещаются все принятые пакеты самоидентификации, причем вместе с контрольными (инверсными) квадлетами. Проверка корректности пакетов (соответствие прямого и инверсного квадлетов) должна выполняться программно. Прием каждого потока (в момент модификации первого квадлета буфера) вызывает запрос прерывания — признаки `selfIDComplete` и `selfIDComplete2` в регистре запросов.

Пакеты самоидентификации, принимаемые не в фазе инициализации шины, и другие пакеты физического уровня отправляются в контекст *AR_Request*.

Организация автоповторов передачи

Хост-контроллер автоматически выполняет *повторные попытки передачи* пакетов при получении квитанции занятости (*busy*). Превышение лимита автоповто-

ров приводит к отбрасыванию пакета; этот факт отражается в дескрипторе передачи. Для аппаратно отрабатываемых запросов превышение лимита индицируется (прерыванием) только при невозможности передачи ответа блокированной транзакции. Параметры автоповтора задаются в регистре `ATRetries` (008h, рис. 25.14). Поля `secondLimit` и `cycleLimit` определяют предел времени автоповтора при двухфазном протоколе. Они соответствуют полям `seconds_limit` и `cycle_limit` в регистре `BUSY_TIMEOUT`, описанном в главе 18. Поле `maxPhysRespRetries` определяет максимальное число попыток повтора ответов для аппаратно отрабатываемых запросов. Поля `maxATRespRetries` и `maxATReqRetries` задают предел числа попыток передачи пакетов асинхронных ответов и запросов из соответствующих контекстов DMA.

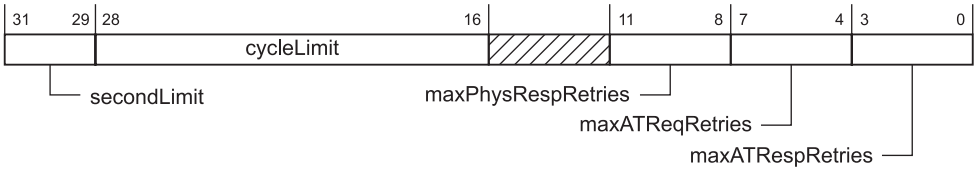


Рис. 25.14. Регистры параметров автоповтора

Регистры управления контроллером

Кроме основной работы с контекстами DMA, обеспечивающей взаимодействие хоста с другими узлами шины FireWire, программа должна идентифицировать хост-контроллер, конфигурировать и управлять его поведением. Эти задачи решаются обращениями к регистрам хост-контроллера, описанным ниже.

Идентификация контроллера

Контроллер OHC обеспечивает возможность идентификации со стороны хоста двояко:

- идентифицируясь как устройство PCI в конфигурационном пространстве шины PCI;
- предоставляя информацию о своих возможностях и уникальном идентификаторе через свои регистры, отображенные в пространство памяти хоста.

Идентифицировать себя со стороны шины IEEE 1394 контроллер позволяет, обеспечивая доступ к своей памяти конфигурации.

На шине PCI контроллер IEEE 1394 с интерфейсом OHCI представляется устройством (функцией) с кодом *класса* 0Ch (контроллеры последовательных шин), кодом *подкласса* 00h кодом и *интерфейса* 10h.

Версия и функциональность OHCI определяются регистром `Version` (000h, рис. 25.15, *а*). Поля `version` и `revision` определяют старшую и младшую части номера версии, бит `GUID_ROM` указывает на реализацию одноименного регистра и автоматическую загрузку уникального идентификатора.

Уникальный идентификатор хранится в энергонезависимой памяти (ПЗУ). В этом ПЗУ содержится описание контроллера в формате, определенном разработчиком. Там же может присутствовать область *miniROM* со структурами данных, похожими на память конфигурации узла IEEE 1394. Доступ к ПЗУ идентификатора обеспечивает регистр `GUID_ROM` (004h, рис. 25.15, *б*). К данным ПЗУ идентификатора возможен только последовательный доступ. Для получения данных из ПЗУ программа первым делом должна обнулить адрес считывания, установив бит `addrReset`. Сбросом этого бита контроллер сигнализирует о выполнении данной операции. Далее, установкой бита `rdStart` хост инициирует чтение байта; по завершении чтения бит сбрасывается контроллером и программа может считать данные из поля `rdData`. Очередная операция чтения инкрементирует внутренний счетчик адреса данных ПЗУ, что позволяет последовательно считать все данные. Поле `miniROM` определяет положение одноименной области в ПЗУ, нулевое значение поля означает отсутствие структуры MiniROM.

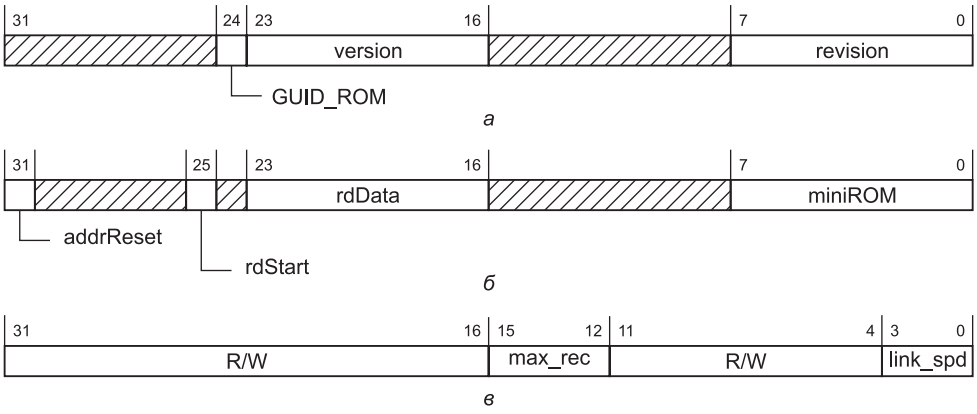


Рис. 25.15. Регистры идентификации: *а* — регистр версии; *б* — регистр доступа к идентификатору; *в* — регистр опций шины

Память конфигурации реализуется отображением ее адресов на блок в памяти хоста; начальная область памяти конфигурации отображается регистрами контроллера. Запросы со стороны шины на чтение памяти конфигурации отрабатываются аппаратно. К памяти конфигурации узла относятся следующие регистры:

- регистр `ConfigROMhdr` (018h) содержит заголовок памяти конфигурации, определяющий ее размер (см. главу 20);
- регистр `BusID` (01Ch) идентифицирует шину — содержит константу 31333934h — «1394» в символах ASCII;

- регистр BusOptions (020h, рис. 25.15, *в*) содержит второй квадлет блока Bus_Info_Block. Поле max_rec задает максимальный размер обслуживаемых пакетов запросов, поле link_spd — скорость LINK-уровня узла. Остальные биты допускают программную запись, их значение должно соответствовать описанию данного блока в главе 20;
- регистры GUIDHi, GUIDLo (024h, 028h) содержат глобально уникальный 64-битный идентификатор узла — 3-й и 4-й квадлеты блока Bus_Info_Block;
- регистр ConfigROMmap (034h) задает адрес памяти хоста, на которую отображаются обращения со стороны шины к памяти конфигурации (младшие 10 бит — нулевые);
- регистр Vendor_ID (040h) задает идентификатор фирмы-производителя и назаченный ей идентификатор устройства.

Общее управление контроллером

Общее управление контроллером обеспечивает регистр HCControl (рис. 25.16). Отдельные биты регистра устанавливаются и сбрасываются установкой соответствующих бит в регистрах HCControlSet (050h) и HCControlClear (054h). Назначение битов:

- BIBimageValid — признак действительности образа блока Bus_Info_Block и памяти конфигурации, разрешающий доступ к ним со стороны шины;
- noByteSwapData — управление перестановкой байтов, соотетствует формату данных на хост-шине: 0 — Little Endian (перестановка выполняется), 1 — Big-Endian (нет перестановки);
- ackTardyEnable — разрешение ответа ack_Tardy на обращения к памяти конфигурации;
- programPhyEnable — признак возможности программного управления расширенными (1394a) свойствами PHY;
- aPhyEnhanceEnable — признак разрешения использования всех расширений физического уровня, принятых в 1394a;
- LPS — разрешение взаимодействия по интерфейсу LINK-PHY;
- postedWriteEnable — разрешение выполнения физических обращений в форме отправленных записей;
- linkEnable — разрешение работы LINK-уровня (после установки бита необходимо запросить сброс на шине);
- softReset — программный сброс хост-контроллера (очистка всех FIFO, приведение регистров в исходное состояние), сброс на шине при этом не генерируется. При чтении бит указывает на выполнение сброса (программного или аппаратного). По завершении сброса бит обнуляется.

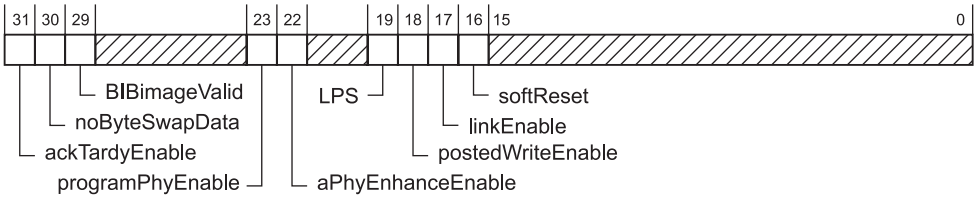


Рис. 25.16. Регистр управления контроллером

Управление прерываниями

Прерывания для процессора хоста вызываются по событиям каналов DMA (асинхронных и изохронных), событиям LINK-уровня и по ошибкам. Для наблюдения и управления источниками прерываний в контроллере имеется регистр запросов и регистр масок прерываний. События, вызывающие аппаратное прерывание процессора хоста, отражаются в регистре `IntEvent`. Этот регистр считывается через регистр `IntEventSet` (080h), биты запросов сбрасываются при записи единиц в соответствующие биты регистра `IntEventClear` (084h). Чтение регистра `IntEventClear` показывает незамаскированные запросы — результат функции И (`IntEvent & IntMask`). Маски событий задаются регистром `IntMask`, установка — через регистр `IntMaskSet` (088h), сброс — через `IntMaskClear` (08Ch). Назначение бит регистров запросов и масок раскрыто в табл. 25.3.

Таблица 25.3. Назначение битов регистров запросов и масок прерываний

Бит	Назначение
0	`reqTxComplete`, передача пакета запроса с командой *OUTPUT_LAST**
1	`respTxComplete`, передача пакета ответа с командой *OUTPUT_LAST**
2	`ARRQ`, завершение команды из контекста *AR_Request*
3	`ARRS`, завершение команды из контекста *AR_Response*
4	`RQPkt`, прием пакета в контекст *AR_Request*
5	`RSPkt`, прием пакета в контекст *AR_Response*
6	`isochTx`, признак запроса прерываний от какого-либо контекста изохронной передачи
7	`isochRx`, признак запроса прерываний от какого-либо контекста изохронного приема
8	`postedWriteErr`, ошибка отправленной записи
9	`lockRespErr`, невозможность отправки результата блокированной транзакции из-за превышения допустимого числа повторов
10–14	Резерв
15	`selfIDcomplete2`, завершение фазы самоидентификации (не обнуляется по сбросу шины)
16	`selfIDcomplete`, завершение фазы самоидентификации (обнуляется по сбросу шины)
17	`busReset`, индикация состояния сброса PHY, при этом запись в регистры фильтров и управляющий регистр контроллера не вызывает никаких эффектов

Бит	Назначение
18	`regAccessFail`, отказ операции обращения к регистрам PHY из-за отсутствия сигнала синхронизации SCLK
19	`Phy`, запрос прерывания от PHY
20	`cycleSynch`, начало нового изохронного цикла (по внутреннему таймеру)
21	`cycle64Seconds`, признак изменения бита 7 счетчика секунд
22	`cycleLost`, признак потери пакета начала цикла (он не принят между двумя событиями *cycleSynch*)
23	`cycleInconsistent`, признак приема пакета начала цикла, в котором значения полей счетчиков не соответствуют внутреннему таймеру
24	`unrecoverableError`, признак неисправимой ошибки какого-либо блока (например, состояние *dead* для какого-либо контекста)
25	`cycleTooLong`, слишком длинный изохронный цикл (через 115–120 мкс после начала цикла не встретился зазор для асинхронного арбитража)
26	`phyRegRcvd`, прием байта данных регистра PHY
27	ack_tardy, при установленном бите `ackTardyEnable` регистра `HCControl` имеет место одно из условий: в приемном FIFO имеются данные для доставки хосту; блок аппаратного ответа занят обслуживанием запроса или посылкой ответа; хост-контроллер послал пакет квитирования ack_tardy
28	Резерв
29	`softInterrupt`, программно вызываемое прерывание
30	`vendorSpecific`, специфическое прерывание
31	Резерв

Идентификацию и маскирование *контекстов изохронной передачи*, вызывающих прерывания, обеспечивают регистры `IsoXmitIntEvent` и `IsoXmitIntMask`. Каждый бит этих регистров соответствует событию прерывания и маске для своего контекста изохронной передачи. Все запросы считываются через регистр `IsoXmitIntEventSet` (090h); биты запросов сбрасываются при записи единиц в соответствующие биты регистра `IsoXmitIntEventClear` (094h). Чтение регистра `IsoXmitIntEventClear` показывает запросы незамаскированных контекстов. Маски контекстов устанавливаются через регистр `IsoXmitIntMaskSet` (098h), сбрасываются — через `IsoXmitIntMaskClear` (09Ch).

Идентификацию и маскирование *контекстов изохронного приема*, вызывающих прерывания, аналогичным образом обеспечивают регистры `IsoRecvIntEvent` и `IsoRecvIntMask`. Эти регистры доступны через `IsoRecvIntEventSet` (0A0h), `IsoRecvIntEventClear` (0A4h), `IsoRecvIntMaskSet` (0A8h) и `IsoRecvIntMaskClear` (0ACh). Каждый бит этих регистров соответствует событию прерывания и маске для своего контекста изохронного приема.

Специальные регистры для изохронного режима

Для поддержки изохронного режима шины контроллер имеет специальные регистры:

- *таймер циклов*, обеспечивающий синхронизацию изохронных операций. Таймер доступен через регистр `IsochronousCycleTimer` (0F0h). Его поля-счетчики `cycleOffset`, `cycleCount` и `cycleSeconds` описаны в главе 20;
- автономные *регистры диспетчера изохронных ресурсов*:
  - регистр идентификатора диспетчера шины;
  - регистр доступной полосы;
  - регистры доступных изохронных каналов.

CSR-архитектура допускает модификацию автономных регистров только через блокированные транзакции *compare_swap*. Контроллер OHC аппаратно реализует эти блокированные транзакции по запросам от других узлов шины. Со стороны хоста выполнить блокированные транзакции *compare_swap* позволяют специальные *регистры доступа к автономным ресурсам*. Для этих транзакций данных записи помещаются в регистр `CSRData` (00Ch), данные сравнения — в регистр `CSRCompareData` (010h). Записью в регистр `CSRControl` (014h, рис. 25.17) выбирается автономный ресурс и инициируется операция. Признаком завершения операции является бит `csrDone` (при записи в регистр `CSRControl` он должен обнуляться). Ресурс выбирается полем `csrSel`: 0 — `BUS_MANAGER_ID`, 1 — `BANDWIDTH_AVAILABLE`, 2 — `CHANNELS_AVAILABLE_HI`, 3 — `CHANNELS_AVAILABLE_LO`. После выполнения операции регистр `CSRData` содержит данные, которые были в выбранном регистре до начала операции. Таким образом обеспечивается чтение автономных регистров. Об успешности выполнения операции можно судить по совпадению этих считанных данных с данными, записанными в регистр `CSRCompareData`. Чтение автономных регистров возможно и по их прямым адресам в пространстве регистров хоста: `InitialBandwidthAvailable` (0B0h) и `InitialChannelsAvailableHi`, `InitialChannelsAvailableLo` (0B4h, 0B8h). Регистр `BUS_MANAGER_ID` не имеет прямого отображения.

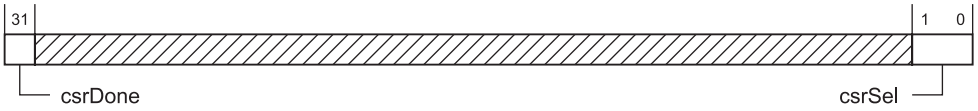


Рис. 25.17. Регистр доступа к автономным ресурсам CSR control

Управление уровнями LINK и PHY

Управления LINK-уровнем обеспечивает регистр `LinkControl` (рис. 25.18, *а*), биты которого могут устанавливаться и сбрасываться через регистры `LinkControlSet` (0E0h) и `LinkControlClear` (0E8h). Регистр содержит следующие биты:

- `cycleSource` — источник синхронизации счетчика `cycleCount`: 0 — внутренний генератор, 1 — внешний источник;
- `cycleMaster` — признак мастера циклов. При установленном бите данный узел будет передавать пакеты начала циклов, если он является корневым;
- `cycleTimerEnable` — разрешение работы счетчика циклов;
- `rcvPhyPkt` — разрешение приема PHY-пакетов и помещения их в контекст *AR_Request* (на прием пакетов самоидентификации не влияет);
- `rcvSelfID` — разрешение приема пакетов самоидентификации;
- `tag1SyncFilterLock` — единичное значение бита эквивалентно единицам в поле `tag1SyncFilter` регистра `ContextMatch` всех *IR*-контекстов.

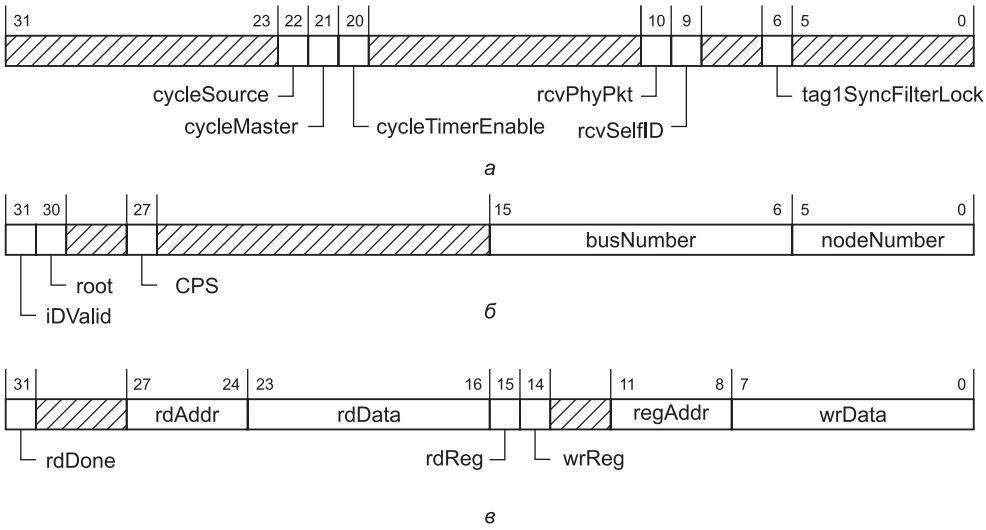


Рис. 25.18. Управление LINK и PHY: *а* — регистр LinkControl; *б* — регистр идентификатора узла; *в* — регистр PhyControl

Для *управления приоритетным арбитражем* (1394a) служит регистр `Fairness-Control` (0DCh, необязательный). Его поле `pri_req` (биты [7:0]) задает число приоритетных запросов, которые LINK-уровень имеет право послать за один интервал справедливости.

Физический идентификатор и состояние узла определяются по регистру `NodeID` (0E8h, рис. 25.18, *б*):

- `iDValid` — признак действительности идентификатора;
- `root` — признак того, что узел — корень шины;
- `CPS` — признак нормального питания от кабеля;

- `busNumber` — номер шины, устанавливается программно (по умолчанию 3FFh);
- `nodeNumber` — номер узла, получаемый автоматически.

Программный доступ к регистрам PHY предоставляется через регистр `PhyControl` (0ECh, рис. 25.18, *в*). Для чтения адрес регистра указывается в поле `rdAddr` и устанавливается признак `rdReg`. Завершение чтения определяется по биту `rdDone` (установится контроллером), считанные данные будут в поле `rdData`. Для записи адрес и данные задаются в полях `regAddr` и `wrData`, признак операции — бит `wrReg`. К PHY-регистру с номером 0 следует обращаться только через регистр `NodeID`.

ГЛАВА 26

Протокол SBP-2

Протокол SBP-2 описывает транспортировку команд, данных и состояния между устройствами, подключенными к шине IEEE 1394 с использованием расщепленных транзакций. Протокол соответствует требованиям *архитектурной модели SAM* (SCSI-3 Architecture Model), что позволяет реализовать шину SCSI в последовательном варианте. В протоколе имеются расширения (относительно SAM) для обеспечения управления изохронными соединениями и передачи изохронных данных.

Протокол описан в документе Information technology — Serial Bus Protocol 2 (SBP-2) комитета X3T10.

При разработке SBP-2 преследовался ряд целей:

- обеспечить возможность инкапсуляции команд, данных и статуса выполнения для разнообразных наборов команд, как старых (определенных в SCSI), так и новых;
- обеспечить возможность для инициатора формировать большой набор выполняемых заданий без оглядки на ограничения, присущие целевым устройствам;
- обеспечить возможность для инициатора динамического добавления новых заданий целевому устройству, без побочных влияний на исполнение ранее выданных заданий;
- реализовать поддержку различных уровней производительности и функциональности целевых устройств без влияния этих уровней на аппаратные и программные средства инициатора;
- использовать преимущества последовательной шины (функциональность, возможность изохронных обменов);
- обеспечить масштабируемость общей производительности системы; для этого протокол должен распределять контекст DMA от инициатора к целевым устройствам. При таком подходе возможность обслуживания большого количества целевых устройств не будет упираться в возможности аппаратно-программных средств инициатора (компьютера).

Организация взаимодействия устройств

Протокол SBP-2 определяет процедуры взаимодействия инициатора с целевыми устройствами. *Инициатор* (*Initiator*) — устройство, заинтересованное в выполне-

нии каких-либо действий целевым устройством и формирующее запросы на выполнение этих действий.

Целевое устройство (target) способно исполнять адресованные ему *команды*. Примером команд может быть чтение или запись данных устройством хранения, воспроизведение аудиотрека и т. п. Каждое целевое устройство имеет определенный набор поддерживаемых команд (систему команд).

Задание (task) — понятие, определяющее действия целевого устройства, связанные с выполнением команды. Для выполнения задания целевое устройство оперирует *контекстом задания*, в который входят, например, адреса и длина передаваемых данных, статус выполнения, отношения с другими заданиями. Время жизни задания начинается с момента, когда целевому устройству сигнализировано данное задание, и завершается, когда целевое устройство сигнализирует инициатору статус завершения. За время своей жизни задание использует ресурсы как инициатора, так и целевого устройства.

Для выполнения каждого задания инициатор формирует *запрос*, оформленный в виде структуры данных в памяти узла шины. Запросы могут быть связаны в цепочки, каждая цепочка представляет при этом *набор заданий* (task set).

Целевое устройство само выбирает задания из памяти и исполняет их; при определенных условиях оно имеет право изменять порядок исполнения заданий в пределах набора. За *передачу данных*, необходимых для выполнения заданий, *отвечает целевое устройство*. Это как раз и обеспечивает масштабируемость системы: добавление новых периферийных устройств не будет перегружать процессор инициатора (хост-компьютера). Состояние выполнения задания (успешное или с ошибкой) целевое устройство сообщает записью по адресу, указанному инициатором в запросе. Инициатор может динамически добавлять задания, а также управлять выполнением заданий (принудительно завершать).

Структура целевого устройства

Протокол SBP-2 используется для работы с целевыми устройствами, подключенными к шине IEEE 1394, то есть входящими в состав узла шины (см. главу 17).

Блок (Unit) — часть узла шины, обеспечивающая некоторую функциональность (память, ввод/вывод, хранение, обработку данных и т. п.). После инициализации узла блок предоставляет CSR-интерфейс, которым обычно пользуется программный агент инициатора. Узел может содержать несколько блоков, как правило, независимых друг от друга. Узел шины, реализующий протокол SBP-2 (целевое устройство), для каждого своего блока должен иметь *каталог блока* (unit directory), в котором идентифицируется его присутствие и описываются его свойства. Каталог находится в памяти конфигурации узла. В каталоге блока присутствует его *уникальный идентификатор EUI-64*, по которому блок можно однозначно идентифицировать. Идентификация через EUI-64 независима от физического идентификатора узла, в котором значение `Phy_ID` может меняться после любого сброса.

Блок узла содержит один или несколько *логических блоков* (Logical Unit), каждый из которых представляет модель устройства (например, устройство хранения, прин-

тер и т. п.). Каждый логический блок имеет идентификатор LUN (Logical Unit Number), уникальный в пределах данного целевого устройства. Логический блок с LUN=0 должен быть обязательно. Разные логические блоки в пределах одного блока могут представлять устройства с разнотипными моделями. Присутствие логических блоков в узле может быть описано в памяти конфигурации или же определяться с помощью запросов, адресованных целевому устройству.

С каждым логическим блоком должен быть связан *сервер устройства* (device server), отвечающий за исполнение команд, посланных устройству. Потоковое устройство должно иметь еще и *контроллер потока* (stream controller), один или несколько. Сервер должен обслуживать *набор заданий* (task set), один или несколько, в котором содержатся команды для исполнения сервером устройства или контроллером потока.

Запросы

Целевые действия (например, чтение с диска, которое передает данные с носителя в системную память) определяются *запросами*, формируемыми инициатором и сигнализируемыми им целевому устройству. Запрос содержится в структуре данных, называемой *блоком запроса операции*, *ORB* (Operation Request Block). Конечный статус запроса содержится в *блоке состояния* (status block), который целевое устройство записывает по адресу, предоставленному инициатором. Стандарт SBP-2 описывает различные форматы ORB, используемых для следующих целей:

- получения доступа к ресурсам целевого устройства (*login requests*);
- транспортировки командных блоков, обычных и потоковых (*command block requests*);
- управления наборами заданий и освобождения целевых ресурсов (*management requests*);
- управления потоком изохронных данных (*stream control requests*).

Запросы получения доступа и управления заданиями направляются агентам, которые обслуживают по одному запросу. Остальные запросы содержат специальные поля-указатели на следующий ORB (или нулевые указатели). Это позволяет строить *очереди* — связанные списки запросов (linked request list), представляющие наборы заданий.

Агенты целевого устройства

Агенты целевого устройства — это программные компоненты, занимающиеся приемом и отработкой запросов. Взаимодействие инициатора с агентами осуществляется через регистры агентов целевого устройства. Агенты целевого устройства разделяются на два основных типа:

- агент, обслуживающий *одиночные запросы*. Такие запросы сигнализируются инициатору транзакцией записи (или блокированной транзакцией *compare_swap*), в которой передается адрес запроса;

- агент, обслуживающий *очередь запросов*. Инициатор добавляет новые запросы в список, а целевой агент выбирает эти запросы из области памяти, указанной инициатором, по мере освобождения своих ресурсов.

Запросы из очереди обслуживаются *выбирающим агентом* (fetch agent) целевого устройства, который читает запросы из памяти инициатора, когда инициатор сигнализирует об их доступности. Целевое устройство имеет право на *упреждающее чтение* (read ahead) последующих доступных запросов, еще до завершения обслуживания ранее считанных. Целевому устройству ради повышения производительности позволяется изменять порядок обслуживания запросов очереди. По завершении обслуживания запроса, успешному или аварийному, целевое устройство записывает блок состояния по адресу, указанному инициатором в данном запросе.

Для обслуживания запросов из очередей инициатор и целевое устройство работают с *контекстом очереди*, который должен состоять как минимум из трех элементов:

- связанный список запросов;
- адрес текущего запроса;
- «звонок» (doorbell), которым инициатор информирует целевое устройство о поступлении новых запросов (обновлении списка заданий).

По назначению различают три типа агента целевых устройств:

- *агент управления* (management agent), принимающий запросы подключения (*login*), управления заданиями (*task management*) и отключения (*logout*). Этот агент отрабатывает только одиночные запросы. До того как инициатор успешно выполнит защищенное подключение (*sequre login*), никакие другие запросы к целевому устройству выполняться не будут;
- *агент командных блоков*, обслуживающий основной поток целевых запросов (обычных и потоковых), в соответствии с правами, полученными при подключении. Этот агент работает с очередями;
- *агент управления потоком*, связанный с изохронными операциями. Агент управления потоком связывается с агентом, обеспечивающим передачу потоковых командных блоков. Этот агент управления работает с очередями.

Потоки

Поток в SBP-2 — это объект, базирующийся на возможностях изохронной передачи 1394. Поток использует ресурсы целевого узла, необходимые для передачи данных. В блоке-«слушателе» (listener) данные одного или нескольких изохронных каналов 1394 передаются в среду целевого устройства (пример — цифровые аудиоколонки). В передающем блоке (talker) данные из среды передающего целевого устройства передаются в один или несколько изохронных каналов (пример — цифровой микрофон, аудио CD).

Потоки отличаются от обычных передач:

- для потока в контексте не используется адрес памяти, по которому записывается или считывается текущая порция данных. Потоковые данные идентифицируются номером канала и позицией данных (порядком по времени поступления) в потоке;
- потоки требуют управления скоростью с синхронизацией по времени или некоторым событиям.

Эти особенности требуют для полного управления потоком наличия двух компонент: набора потоковых заданий и контроллера потока.

Набор потоковых заданий состоит из цепочки команд обмена данными, в которых не используются адреса системной памяти, но подразумевается строгая упорядоченность данных.

Контроллер потока выполняет передачу данных между набором потоковых заданий и последовательной шиной. Формат передаваемых данных аналогичен формату изохронных пакетов 1394. Данные идентифицируются *меткой времени* (time stamp) и *номером канала*; размер данных определяется с точностью до байта. Контроллер потока фильтрует изохронные данные согласно номерам каналов, преобразует номера каналов и метки времени и синхронизирует поток данных с внешними событиями. Запросы, управляющие потоком (*Stream Control ORB*), и запросы команд потоковых заданий независимы друг от друга.

Во время изохронных передач возможны ошибки:

- потеря изохронного пакета или пакета начала цикла;
- ошибка в принятом пакете (по CRC-контролю);
- нехватка данных передатчику (underflow) — очередной ORB не сформирован к моменту, когда его данные уже должны передаваться по шине;
- переполнение буферов приемника (overflow).

Поведение целевого устройства при ошибках может быть различным:

- игнорировать ошибки, продолжая работу;
- останавливаться по первой же ошибке и сообщать о ней;
- регистрировать ошибки, продолжая работу. Для этого каждый ORB управления потоком должен определять буфер, в который целевое устройство помещает сообщения об ошибках (error log). Буфер будет доступен инициатору, когда данный ORB будет исполнен.

Выполнение нормальных заданий

Работа инициатора с целевым устройством начинается с *запроса входа* (*LOGIN*), указатель на который инициатор записывает в регистр управляющего агента. После успешного входа инициатор получает указатели на базовые регистры требуемых ему выбирающих агентов. Это позволяет инициатору взаимодействовать с агентом целевого устройства для организации выполнения заданий и их наборов.

Понятие «задание» соответствует одному запросу (ORB), а «набор заданий» — связанной цепочке запросов.

Агенты требуют *инициализации*: инициатор должен сформировать в своей памяти пустой ORB с нулевым указателем на следующий блок и транзакцией записи в регистр `CURRENT_ORB` передать указатель на этот блок агенту. Агент, выбрав и выполнив пустой блок, перейдет в состояние *SUSPENDED*.

Для *постановки заданий в очередь* инициатор формирует в памяти цепочку блоков запросов и помещает ссылку на первый блок цепочки в поле `next_ORB` пустого блока, заданного при инициализации. Об этой модификации памяти инициатор сообщает агенту записью в его регистр `DOORBELL`, что заставит агента снова считать блок запроса и перейти к исполнению первого запроса цепочки. В дальнейшем к существующей цепочке можно динамически добавлять следующие запросы (или цепочки). Для этого инициатор помещает указатель на новый запрос в поле `next_ORB` блока, бывшего последним в предыдущей цепочке, и сигнализирует об этом обновлении записью в регистр `DOORBELL`. Если новые запросы добавляются когда агент приостановился (в состоянии *SUSPENDED*), то добавление можно выполнять и записью в регистр `CURRENT_ORB` указателя на новый блок (это исключает повторную выборку старого ORB).

За *передачу данных*, необходимых для выполнения запросов, отвечает целевое устройство. В ORB описывается передача блока данных, которые могут и не умещаться в одну транзакцию шины. Целевое устройство разбивает передачу на транзакции с размером поля данных пакета, не превышающим определенного в ORB, и выполняет транзакции чтения или записи на заданной скорости.

Состояние завершения запроса целевое устройство сообщает инициатору, записывая в FIFO блок состояния (транзакцией записи блока). Состояние сообщается только для запросов, у которых в ORB установлен бит уведомления (`n`), а также запросов, завершающихся с ошибкой. Сообщение состояния является указанием инициатору на возможность повторного использования памяти, занятой данным ORB (если только он не последний в цепочке), и всех предшествующих ему в наборе заданий.

Управление заданиями может быть реализовано в различных объемах. Для базовой модели управления все задания в наборе имеют одинаковые возможности в плане изменения порядка исполнения или строгой упорядоченности. Эти возможности неявно определяются природой целевого устройства и инициатором не управляются. Для наборов потоковых заданий обязательна строгая упорядоченность, для нормальных наборов возможность изменения порядка указывается в памяти конфигурации целевого устройства. Все задания в пределах набора однозначно идентифицируются адресом своего ORB. Для базовой модели обязательна поддержка функций управления *ABORT_TASK*, *ABORT_TASK_SET* и *TARGET_RESET*; функция *TERMINATE_TASK* должна отвергаться.

В более сложной модели управления установленный порядок выполнения запросов может быть нарушен запросами управления. Конкретно указанное (по адресу ORB) задание может быть принудительно завершено (функцией *TERMINATE_*

TASK) на том этапе выполнения, на котором оно находится. При этом в блоке состояния будет отражена данная причина завершения (*function rejected*), если завершение оказалось преждевременным.

В случае *возникновения ошибки* при выполнении задания целевое устройство должно сбросить весь остаток набора задания: остановить выборку запросов (перейдя в состояние *DEAD*), дождаться успешной записи в память инициатора состояний выполнения всех предыдущих запросов и передать состояние задания, завершенного с ошибкой. Получение состояния с ошибкой для инициатора означает, что оставшиеся задания в наборе отвергнуты.

Изохронные операции

При работе с изохронными устройствами выполняется ряд операций:

- *выделение* инициатору *ресурсов целевого устройства* запросом изохронного входа (*isochronous login)*;
- *управление соединениями* между целевым устройством и другими изохронными устройствами;
- *передача изохронных данных* к целевому устройству или от него с помощью запросов потоковых передач;
- *управление потоком* — запуск, останов и синхронизация передачи данных в шину или приема из шины, выполняемое с помощью запросов управления потоком;
- *распределение ресурсов* последовательной шины — номеров каналов и полосы пропускания, выполняемое через обращения к регистрам диспетчера изохронных ресурсов (см. главу 21).

Управление соединениями

Управление соединениями осуществляется манипуляциями с регистрами управления штекерами (см. главу 18), которые обязательны для всех изохронных устройств, как реализующих SBP-2, так и не реализующих (устройств бытовой электроники). Эти манипуляции физически могут реализоваться блокированными шинными транзакциями обращений к управляющим регистрам штекеров. Протокол SBP-2 позволяет манипулировать этими регистрами опосредованно, с помощью управляющих запросов *CONFIGURE PLUG*.

Штекер может находиться в одном из четырех состояний, видимых по состоянию бита `o` (online) и состоянию подключения. Бит `o` отражает текущую способность устройства к изохронному обмену, он может устанавливаться по запросу *START*, а сбрасываться по запросу *PAUSE*, *STOP* и блокированной транзакцией, обращенной к регистру управления штекером. Штекер считается *подключенным* (connected), если в его управляющем регистре установлен признак широковещания `b` или счетчик двухточечных соединений `point_to_point` имеет ненулевое значение. Состояния штекеров и переходы между ними следующие:

- *IDLE* — исходное состояние, o = 0, штекер не подключен, передача и прием запрещены. В это состояние штекер переходит из любого состояния по начальному включению и по выходу (*logout*). Управляющий запрос *START* переведет устройство в состояние готовности. Подключение штекера переведет его в состояние приостановки;
- *READY* — готовность, o = 1, штекер не подключен, передача и прием запрещены. Запросы *PAUSE* или *STOP* переведут штекер в исходное состояние. Подключение штекера переведет его в активное состояние;
- *ACTIVE* — активность, o = 1, штекер подключен и ведется передача или прием. Запросы *PAUSE* или *STOP* переведут штекер в состояние приостановки. Отключение штекера или сброс шины переведут его в состояние готовности;
- *SUSPENDED* — приостановка, o = 0, штекер подключен, но передача (прием) не ведется. Запрос *START* переведет штекер в активное состояние, отключение или сброс шины — в исходное состояние.

При установлении изохронного соединения устанавливается признак широковещания `b` или инкрементируется счетчик двухточечных соединений `point_to_point`. Первое же установленное соединение переводит штекер в подключенное состояние. Разрыв соединения сопровождается сбросом признака `b` или декрементом счетчика `point_to_point`. Последнее разорванное соединение переводит штекер в отключенное состояние.

Сброс на шине меняет состояние штекера, обнуляя в его управляющем регистре бит `b` и счетчик `point_to_point`. При этом штекер, активный перед сбросом, переходит в состояние готовности, но продолжает передачу или прием еще секунду. За эту секунду инициатор обмена успеет снова активизировать изохронное устройство, получив к нему доступ, так что изохронный обмен не прервется. Если после сброса устройство окажется недоступным, то по истечении этой секунды штекер станет готовым к последующему использованию. Штекер, приостановленный перед сбросом, по сбросу переходит в исходное состояние.

До подключения штекера для него должны быть получены ресурсы шины — номер канала и полоса. Для подключенного выходного штекера менять номер канала или скорость нельзя. Для входного штекера номер канала нельзя менять, если у него есть двухточечные соединения.

Передача потоковых данных

Организация передачи потоковых данных — постановка потоковых запросов в очередь, их выборка и исполнение агентом целевого устройства — выполняется аналогично нормальным запросам. Здесь, естественно, целевое устройство не имеет права менять порядок исполнения запросов в наборах заданий (и между наборами). Отработка заданий может притормаживаться управляющими запросами к контроллеру потока (пауза может быть неопределенно долгой). Целевое устройство, передающее поток в шину, сообщает состояние завершения, как только его контроллер потока передаст указанные данные. Целевое устройство, принимаю-

щее поток с шины, сообщает состояние завершения либо по приему указанных данных его контроллером потока, либо после выполнения записи этих данных на свой носитель (выбирается при конфигурировании). В отличие от нормальных запросов состояние, сообщаемое при исполнении потока запроса (успешном или ошибочном), не дает инициатору никакой информации о том, какое количество данных запроса реально было записано на носитель или считано с него.

Управление потоком

Управление потоком осуществляется запросами, связанные списки которых строит инициатор в памяти узла. Агент контроллера потока в целевом устройстве должен выбирать эти запросы достаточно быстро, поскольку управляющие действия привязываются к событиям синхронизации, для которых дискретность времени составляет всего 125 мкс (период цикла). Контроллер потока должен выполнять ряд действий:

- синхронизацию, пуск и останов приема (передачи) данных от шины (на шину) по определенному моменту времени или по специальным меткам, обнаруженным в потоке принимаемых данных;
- управление каналами — селективное разрешение их работы с помощью масок каналов;
- преобразование заголовков изохронных данных, представляемых CIP-пакетами, передаваемыми по шине и записываемыми на носитель устройства (считывания с носителя). Эти преобразования затрагивают номера каналов и метки времени, а также добавление или удаление *NULL*-пакетов (заполнителей).

Регистрация ошибок

Во время активной работы контроллер потока может обнаруживать ошибки (пропущенные пакеты данных или пакеты начала цикла, ошибки данных, переполнение или переопустошение буферов). Для изохронных передач сообщение о состоянии выполнения каждого запроса передачи данных неприемлемо, поэтому обработка ошибок выполняется особым способом. Полем `rpt` запроса управления потоком (*SET ERROR MODE*) контроллеру потока задают поведение при ошибке:

- остановиться по первой же ошибке;
- зарегистрировать ошибку и продолжать работу;
- игнорировать ошибку.

Ошибки регистрируются в журнале `error_log` — области памяти, которую выделяет инициатор. Ошибки приводят к потере непрерывности потока, и в журнале регистрируются события, приводящие к прерыванию и возобновлению нормального потока. Каждое из этих событий регистрируется с соответствующей меткой времени.

Структуры данных SBP-2

В протоколе SBP-2 фигурируют три класса структур данных:

- блоки запросов операций (*ORB — operation request block*);
- таблицы страниц (*page tables*), описывающие блоки передаваемых данных, если они располагаются не в физически непрерывной области памяти;
- блоки статуса (*status block*) выполнения запросов.

Инициатор обмена располагает и инициализирует эти структуры данных в системной памяти узлов шины. Таблицы страниц должны находиться в памяти тех узлов, в которых расположены буферы данных, описываемые этими таблицами. Все эти структуры выравниваются по границе квадлета; они адресуются 64-битными указателями, соответствующими формату адреса в IEEE 1394 (см. главу 17). Поле идентификатора узла `Node_ID` (номер шины и физический идентификатор) в ряде случаев является избыточным и не используется. Младшие 2 бита 48-битного смещения всегда нулевые (из-за выравнивания). На рисунках, приведенных в данной главе, биты нумеруются в порядке Big Endian, принятом в 1394.

Блоки запросов операций (ORB)

Блоки запросов операций (ORB) описывают действия целевого устройства, которые ему предписывает исполнить инициатор. Блоки ORB *выбираются целевым устройством* посредством транзакций чтения. ORB являются блоками переменного размера, их длина и формат определяются полем `rq_fmt`. Размер ORB определяется типом устройства и его набором команд, этот размер указывается в памяти конфигурации. С точки зрения оптимизации производительности предпочтительны блоки, кратные 32 байтам (размер строки кэша). Обобщенный формат ORB приведен на рис. 26.1, *а*. Указатель `next_ORB` имеет формат, изображенный на рис. 26.1, *б*. Бит `n` (`notify`) указывает на необходимость сигнализации инициатору об исполнении данного запроса. Поле `rq_fmt` определяет формат блока: 0 — стандартный (SBP-2), 1 — резерв, 2 — специфический (vendor-dependent), 3 — пустой блок (*Dummy ORB*). Содержимое остальных полей (`rq_fmt-dependent`) зависит от типа запроса.

Пустой блок ORB используется лишь как указатель на следующий ORB или признак конца цепочки запросов. В этом блоке значимы лишь два начальных квадлета, в которых содержится `next_ORB` — указатель на следующий ORB. В этом указателе поле `Node_ID` по прямому назначению не используется, его старший бит указывает на действительность указателя (нуль в старшем бите является признаком конца цепочки).

Блоки запросов команд `Command Block ORB` делятся на два типа:

- нормальный (асинхронный) командный блок (Normal command block), в котором описаны буферы данных в системной памяти;

- потоковый (изохронный) командный блок (Stream Command Block), в котором описывается только длина передаваемого блока.

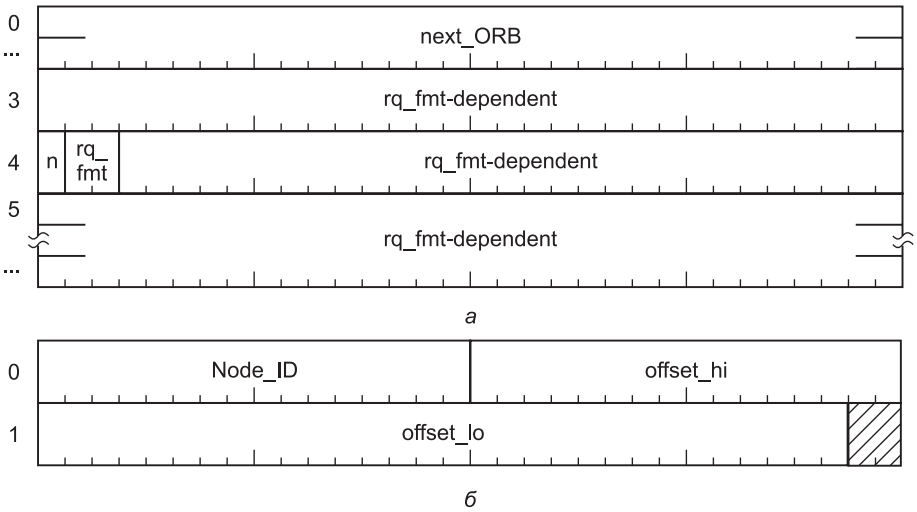


Рис. 26.1. Формат запроса SBP-2: *а* — общий формат ORB; *б* — формат указателя

Нормальный командный блок

Нормальный командный блок используется для описания действия целевого устройства, связанного с асинхронными транзакциями шины IEEE 1394. Примером этих действий может быть чтение (или запись) блоков данных с устройств хранения, передача данных на принтеры, ввод данных со сканера и т. п. Формат командного блока приведен на рис. 26.2.

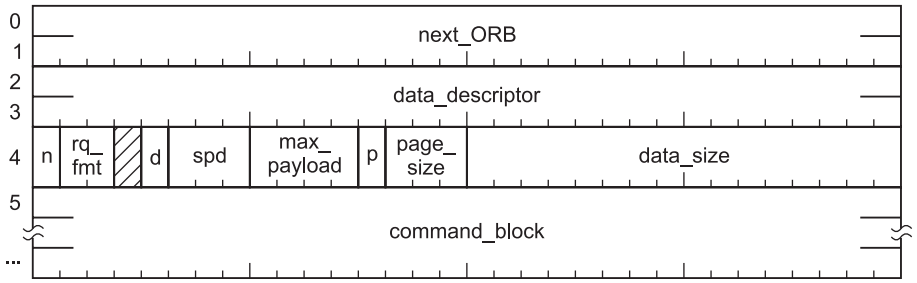


Рис. 26.2. Формат нормального командного блока

Поле `next_ORB` может указывать только на следующий нормальный блок или являться признаком конца цепочки (по нулевому старшему биту `Node_ID`).

Поле `data_descriptor` (стандартный 64-битный адрес IEEE 1394) прямо или косвенно указывает на местоположение буфера данных, в зависимости от бита `p` (page_table_present): при `p` = 0 `data_descriptor` указывает на сам буфер, при `p` = 1 — на таблицу страниц (см. ниже).

Бит `d` (direction) задает направление передачи данных: `d` = 0 — целевое устройство читает данные из буфера, `d` = 1 — записывает в буфер.

Поле `spd` указывает скорость передачи: 0 — S100, 1 — S200,...5 — S3200.

Поле `max_payload` определяет максимальный размер передачи за одну транзакцию (размер поля данных в пакете не более $2^{max\_payload + 2}$).

Поле `page_size` определяет размер страниц памяти в буфере данных (при `p` = 0) или таблице страниц (при `p` = 1). При ненулевом `page_size` размер страницы составляет $2^{page\_size + 8}$ байт, `page_size` = 0 означает неопределенность размера страницы.

Поле `data_size` описывает длину блока данных или количество элементов в таблице страниц.

Поле `command_block` содержит команду целевому устройству, его содержимое протоколом SBP-2 не регламентируется.

Потоковый командный блок

Потоковый командный блок используется для описания действия целевого устройства, связанного с изохронными передачами шины IEEE 1394. Примером может быть передача аудиопотока, воспроизводимого с проигрывателя компакт-дисков, передача видеопотока от цифровой видеокамеры и т. п. Формат командного блока приведен на рис. 26.3.

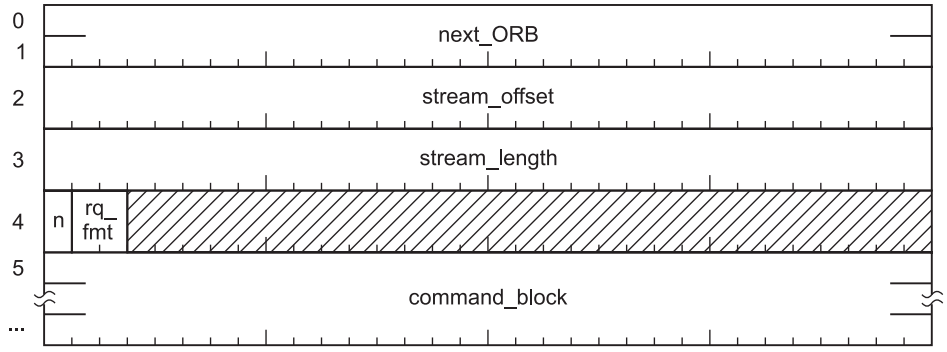


Рис. 26.3. Формат потокового командного блока

Поле `next_ORB` может указывать только на следующий потоковый блок или являться признаком конца цепочки (по нулевому старшему биту `Node_ID`).

Поле `stream_offset` указывает на позицию (в байтах) начала изохронных данных относительно начала элемента, на который указывает команда в поле `command_block`. Позиция должна быть кратна 4 байтам.

Поле `stream_length` задает длину передаваемого элемента в байтах.

Поле `command_block` содержит команду целевому устройству, его содержимое протоколом SBP-2 не регламентируется.

Блок управления потоком

Блок управления потоком служит для управления контроллером потока логического блока. Формат командного блока приведен на рис. 26.4.

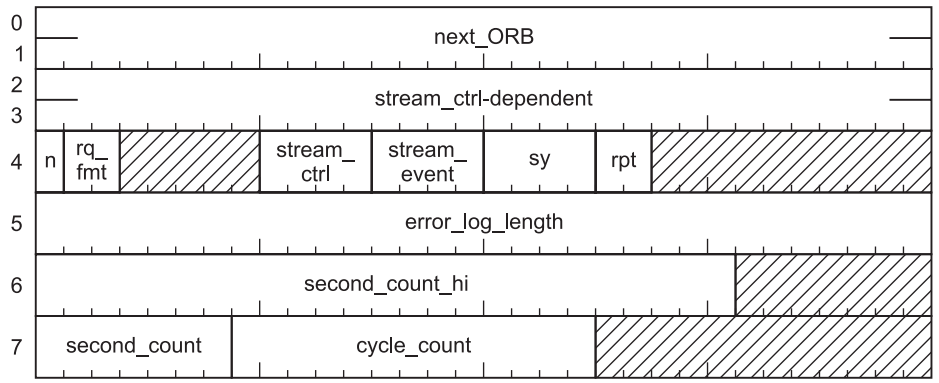


Рис. 26.4. Формат блока управления потоком

Поле `next_ORB` может указывать только на следующий блок управления потоком или являться признаком конца цепочки (по нулевому старшему биту `Node_ID`).

Поле `stream_ctrl-dependent` несет информацию, необходимую для выполнения управляющих функций.

Поле `stream_ctrl` определяет функцию управления:

- 0 — *START*, начало (или возобновление после паузы) приема или передачи потока;
- 1 — *STOP*, останов приема или передачи потока;
- 2 — *PAUSE*, пауза в приеме или передаче потока;
- 3 — *UPDATE CHANNEL MASK*, изменение значения маски каналов. Маска содержится в поле `stream_ctrl-dependent`;
- 4 — *CONFIGURE PLUG*, управление штекерами (см. ниже), позволяет задать поля регистра управления штекером (задать характеристики изохронного канала шины и соединения) и связать с данным штекером внутренний канал целевого устройства. Управляющая информация передается поле `stream_ctrl-dependent` (см. рис. 26.5). Здесь первые 32 бита соответствуют формату регистра

PCR; в поле `plug` задается номер PCR (0–Eh — регистры `OUTPUT_PLUG[plug]`, 10–1Eh — `NPUT_PLUG[plug-16]`); поле `int_channel` задает номер внутреннего канала;

- 5 — *SET ERROR MODE*, управление реакцией на ошибки и, если требуется, определение буфера для сообщений об ошибках. Реакция определяется полем `rpt`: 0 — сообщить об ошибке и остановить поток; 1 — сообщить, не останавливая; 2 — игнорировать ошибки; 3 — резерв. Указатель на буфер сообщений об ошибках содержится в поле `stream_ctrl-dependent`, поле `error_log_length` задает длину буфера;
- 6 — *QUERY STREAM STATUS*, опрос состояния потока;
- 7...Fh — резерв.

Поле `stream_event` служит для определения событий, по наступлении которых выполняются команды *START*, *STOP*, *PAUSE* и *UPDATE CHANNEL MASK*:

- 0 — *IMMEDIATE*, немедленно;
- 1 — *CYCLE MATCH*, в цикле, определяемом полями `second_count_hi`, `second_count` и `cycle_count`;
- 2 — *SY MATCH*, в том цикле, в котором значение поля `SY` совпадет с одноименным полем изохронного пакета любого разрешенного канала (это условие применимо только для приемников потока);
- 3 — *FIRST DATA*, по получении первого изохронного пакета любого разрешенного канала (это условие применимо только для приемников потока);
- 4...Fh — резерв.

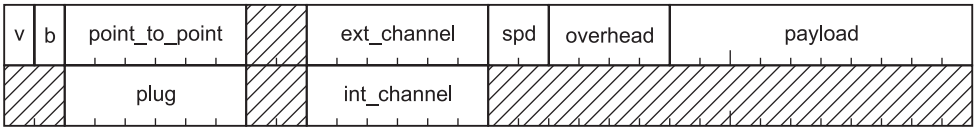


Рис. 26.5. Формат поля управления штекером

Блоки управляющих запросов

Блоки управляющих запросов представляют собой структуры фиксированной длины (32 байта), используемые для целей защиты (login, logout) и управления заданиями. В этих блоках явно указывается их привязка к конкретным наборам заданий (очередям) или потокам, хотя они и так неявно привязаны к агентам, выбирающим командные блоки и блоки управления потоками. Каждое задание однозначно идентифицируется адресом, по которому находится командный блок, инициирующий выполнение этого задания. Общий формат блоков управляющих запросов приведен на рис. 26.6. Эти блоки не могут связываться в цепочки. Поля `n` и `rq_fmt` имеют обычное значение. Поле `status_FIFO` указывает на адрес, по которому располагается FIFO-буфер состояния выполнения запросов.

Поле `function` определяет выполняемую функцию, от назначения которой зависит использование полей `function-dependent`. Определены следующие функции:

- 0 — *SECURE LOGIN*, защищенный вход, который должен быть выполнен до начала любых действий (кроме опроса состояния подключений) с целевым устройством. В этом запросе указывается номер логического блока, положение и длина буферов памяти, в одном из которых находится пароль доступа к носителю, а в другой должен помещаться ответ на запрос входа. В ответе передаются 16-битный идентификатор входа `login_ID`, базовый адрес регистра агента командного блока `command_block_agent` и длина возвращаемых данных;

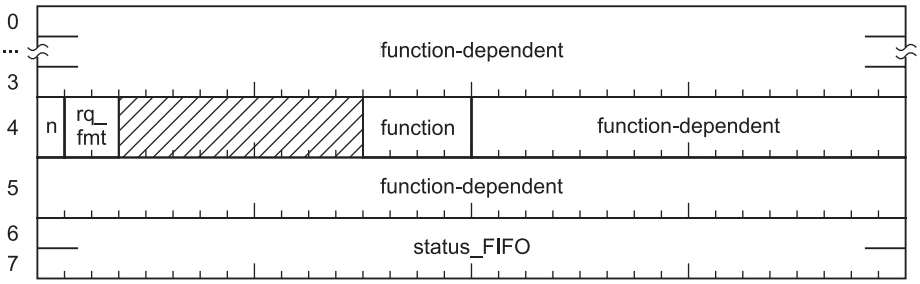


Рис. 26.6. Формат блока управляющего запроса

- 1 — *QUERY LOGINS*, опрос состояния подключений, возвращающий для указанного логического блока один из двух списков:
  - список идентификаторов (EUI_64) всех зарегистрированных владельцев логического блока и код используемого протокола защиты для носителя, установленного в логический блок;
  - список всех текущих инициаторов обмена с логическим блоком (идентификаторов узлов, идентификаторов входа и EUI_64);
- 2 — *ISOCHRONOUS LOGIN*, вход для доступа к потоковым запросам. В запросе указывается роль блока (передатчик или приемник), максимальное число одновременно работающих изохронных каналов, максимальный размер поля изохронных данных, идентификатор входа (`login_ID`, полученный ранее), местоположение и длина возвращаемых данных. В случае успешного входа в возвращаемых данных содержатся:
  - 16-битный идентификатор изохронного потока, выделенный целевому устройству, используемый для последующей работы;
  - указатель на базовый регистр агента командного блока `command_block_agent`;
  - указатель на базовый регистр агента управления потоком `stream_control_agent`;
  - выделенные штекеры `allocated_plugs` (битовая маска для доступных регистров `OUTPUT_PLUG` или `INPUT_PLUG`);
  - минимальная длина передаваемого блока данных `min_transfer_length`, достаточная для того, чтобы не было переполнения или нехватки данных;
- 3 — *RESECURE LOGIN*, восстановление защищенного входа после сброса на шине, когда идентификаторы узлов могут измениться. В запросе передаются

старый идентификатор входа. Целевое устройство проверяет, совпадает ли идентификатор EUI-64 инициатора с тем, который был зарегистрирован для данного идентификатора. В случае успешного восстановления инициатор может пользоваться адресами агентов, сообщенными ему при первоначальном входе;

- 4 — *SET REGISTERED EUI-64*, задание списка идентификаторов (EUI-64) инициаторов, разрешенных для работы с данным целевым устройством. В запросе передается идентификатор входа, а также указатель и длина буфера, в котором находится обновленный список разрешенных инициаторов. Этот список целевое устройство получает транзакцией чтения;
- 5–6 — резерв;
- 7 — *LOGOUT*, выход с потерей права дальнейшего доступа. В запросе передается идентификатор входа;
- 8–9 — резерв;
- Ah — *TERMINATE TASK*, преждевременное завершение указанного задания (возможно, с передачей не всех затребованных данных);
- Bh — *ABORT TASK*, отказ от выполнения указанного задания;
- Ch — *ABORT TASK SET*, отказ от выполнения набора заданий;
- Dh — *CLEAR TASK SET*, сброс набора заданий;
- Eh — *LOGICAL UNIT RESET*, сброс логического устройства;
- Fh — *TARGET RESET*, сброс целевого устройства.

Таблицы страниц

Буферы данных, требуемые для выполнения транзакций по последовательной шине, считаются непрерывными в логическом адресном пространстве узла. При этом они могут отображаться на физическую память, видимую как память узла IEEE 1394, как непрерывной областью, так и фрагментированной на постраничном базисе. Эта фрагментация обусловлена страничной трансляцией адресов, применяемой для организации виртуальной памяти в большинстве современных процессоров. Если буфер данных является непрерывным в пространстве физических адресов узла, то в командных блоках на этот буфер делается прямая ссылка: задается начальный адрес, длина буфера и признак прямой ссылки (p = 0). Если буфер фрагментирован, то ссылка ведет на *таблицу страниц*, указывается число элементов в таблице, в запросе обязательно указывается размер страницы (популярный размер — 4 Кбайт) и признак косвенной ссылки (p = 1). Таблица страниц (рис. 26.7) должна находиться в памяти того же узла, в котором расположены буферы данных.

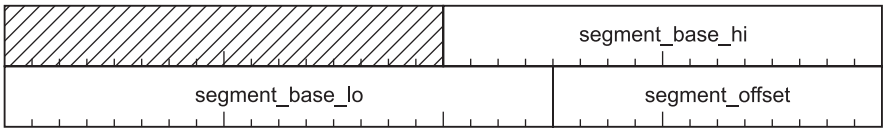


Рис. 26.7. Элемент таблицы страниц

Таблица страниц — массив элементов, описывающих *сегменты*, в которых расположен буфер данных. Каждый элемент таблицы (рис. 26.7) имеет размер 2 квадлета и имеет формат, соответствующий 64-битному адресу IEEE 1394. В этом адресе младшие n-бит являются смещением в сегменте (`segment_offset`), число этих бит определяется размером страницы (на рисунке изображено 12-битное смещение, используемое для для страниц размером 4 Кбайт). Старшие биты (`segment_base_hi` и `segment_base_lo`) являются физическим адресом начала каждой страницы. Длина буфера определяется числом описателей таблицы страниц и значениями полей `segment_offset` описателя первого и последнего сегментов:

- в первом сегменте буфер располагается, начиная с байта, заданного полным адресом, и до конца страницы;
- в промежуточных сегментах буфер занимает всю страницу (поле `segment_offset` в его описателе нулевое);
- в последнем сегменте буфер располагается от начала и до байта, заданного полем `segment_offset` его описателя.

Блок состояния

По исполнении запроса, при установленном бите оповещения (`n`) или в случае обнаружения ошибки целевое устройство помещает *блок состояния* (Status block) в FIFO-буфер, адрес которого указан в блоке запроса. Целевое устройство может помещать в FIFO и блок состояния, неожиданный для инициатора. Блок состояния может иметь размер от 8 до 32 байт (кратный квадлету), его формат приведен на рис. 26.8.

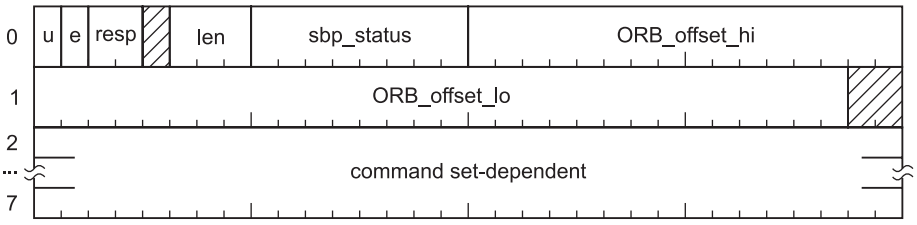


Рис. 26.8. Формат блока состояния

Бит `u` (`unsolicited`) является признаком неожиданного сообщения.

Бит `e` (`end_of_list`) является признаком того, что данное состояние относится к последнему запросу в цепочке.

Поле `resp` определяет тип завершения:

- 0 — *REQUEST COMPLETE*, завершение запроса без транспортной ошибки;
- 1 — *TRANSPORT FAILURE*, завершение из-за неисправимой транспортной ошибки;
- 2 — *ILLEGAL REQUEST*, недопустимый тип запроса;
- 3 — *VENDOR DEPENDENT*, специфическое сообщение.

Поле sbp_status содержит дополнительную информацию:

- 0 — нет дополнительной информации;
- 1 — недопустимый тип запроса;
- 2 — указанная скорость не поддерживается;
- 3 — указанный размер страницы не поддерживается;
- 4 — доступ запрещен;
- 5 — не поддерживается данное логическое устройство;
- 6 — слишком мал допустимый размер поля данных;
- 7 — слишком много каналов;
- 8 — ресурс недоступен;
- 9 — функция отвергнута (rejected);
- Ah — нераспознанный идентификатор входа;
- FFh — неопределенная ошибка.

Поля ORB_offset_hi и ORB_offset_lo определяют адрес блока запроса, к которому относится завершение (при u = 1 поля игнорируются).

Содержательная часть блока состояния (поле command set-dependent) определяется набором команд устройства. Длина блока состояния (в квадлетах) равна len + 1.

Форматы пакетов для изохронных передач

Формат пакетов, в которых изохронные данные записываются на носитель устройства, происходит от формата изохронного пакета (см. рис. 18.8 на стр. 359), передаваемого по шине. Поле tcode (тип транзакции) в пакетах на носителе используется как поле type для определения типа пакета:

- *NULL* (type = 0), пакет-заполнитель (рис. 26.9, *а*). В заголовке этого пакета используется только поле длины data_length, значение которого должно быть кратно четырем (допускается и нулевая длина). Поле data может содержать любые значения, при воспроизведении данные пакета *NULL* игнорируются. Пакеты-заполнители могут использоваться для предотвращения переполнения или опустошения буферов;
- *CYCLE MARK* (type = 1), маркер цикла (рис. 26.9, *б*). Этот пакет записывается на носитель, когда на каком-либо разрешенном канале обнаруживается пакет начала цикла. Поля cycle_count и second_count соответствуют одноименным полям последнего пакета начала цикла. Для воспроизведения изохронного потока эти маркеры не требуются; их используют лишь для приложений, в которых требуется навигация по времени;
- *DATA* (type = 2), пакет изохронных данных (рис. 26.9, *в*). Его формат совпадает с форматом пакета изохронных данных, передаваемого по шине, но в поле chan-

nel указывается внутренний номер канала (int_channel), в который преобразуется номер шинного канала (ext_channel). Преобразование (отображение) номеров задается управляющей функцией *CONFIGURE PLUG*. При воспроизведении потока выполняется обратное преобразование номеров. Использование поля sy, предназначенного для синхронизации, стандартом SBP-2 не регламентируется. Поле tag определяет формат данных: 0 — неформатированные данные; 1 —*общий формат CIP* (см. ниже); 2, 3 — резерв. Данные могут иметь произвольную длину, определяемую полем data_length; при необходимости в последний квадлет добавляются нулевые байты-заполнители.

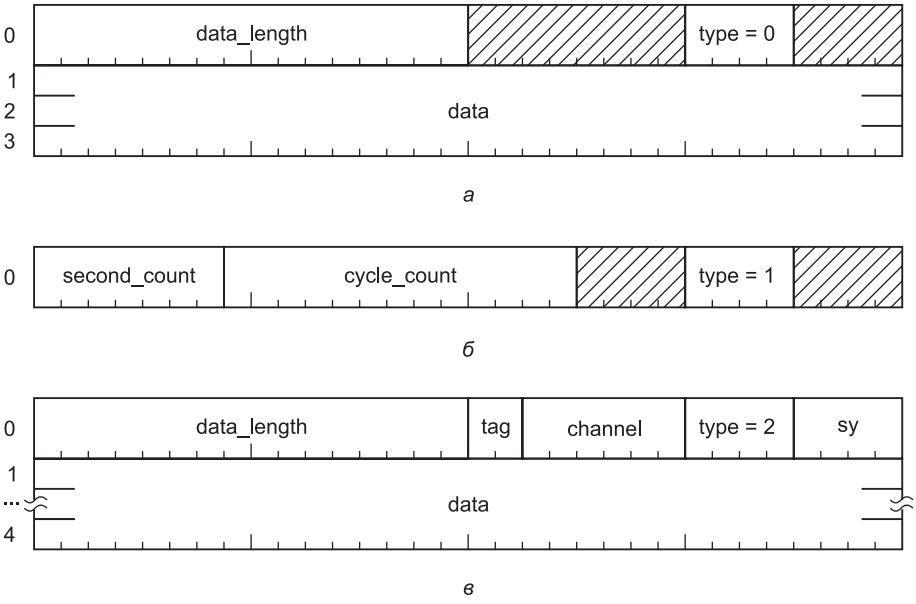


Рис. 26.9. Форматы изохронных пакетов SBP-2: *а* — пакет-заполнитель; *б* — маркер цикла; *в* — пакет данных

Остальные типы (3–Fh) зарезервированы.

В пакете общего формата (CIP-пакете) поле данных, длина которого указана в заголовке пакета, содержит один или несколько *блоков прикладных данных* (*application data block*), каждому из которых предшествует свой *CIP-заголовок* (CIP header). Один или несколько блоков прикладных данных составляют *исходный изохронный пакет* (isochronous source packet) — объект верхнего уровня, несущий специфичные прикладные данные.

CIP-заголовок представляет собой цепочку квадлетов, длина заголовка произвольна, хотя первоначально определен формат только двухквадлетного заголовка (рис. 26.10, *а*). Признаком конца заголовка является квадлет, у которого бит eoh=1. Следующий за ним квадлет является началом блока данных.

Формат каждого квадлета заголовка определяется парой старших бит (`eoh` и `form`). Назначение полей двухквадлетного CIP-заголовка:

- поле `sid` (source ID) содержит физический идентификатор источника (`PHY_ID`), при воспроизведении в него заносится идентификатор узла, передающего данные в шину;
- поле `dbs` (data block size) указывает размер блока данных в квадлетах (`dbs` = 0 соответствует 256 квадлетам);
- поле `fn` (fraction number) задает число блоков, составляющих исходный пакет. Это число определяется как 2^{fn}, при `fn` = 0 каждый блок данных представляет один исходный пакет;
- поле `qpc` (quadlet padding count) определяет, сколько квадлетов-заполнителей добавлено к исходному пакету перед его делением на блоки;
- бит `s` (sph, source packet header) является признаком начала исходного пакета. Если `s` = 1, то последующий блок данных начинается с квадлета-*заголовка исходного пакета* (Source packet header, рис. 26.10, *б*). В этом заголовке передается метка времени — поля `cycle_count` и `cycle_offset`, соответствующие 25 младшим битам регистра `CYCLE TIME`;
- поле `dbc` (data block continuity counter) определяет последовательный номер блока данных в исходном пакете;
- поле `fmt` определяет формат остальной части данного квадлета заголовка и формат прикладных данных в блоке. Для данных с кодами формата `fmt` = 0–1Fh в заголовке присутствует метка времени `syt` (Synchronization time), состоящая из полей `cycle_count` и `cycle_offset` (рис. 26.10, *в*), соответствующий 16 младшим битам регистра `CYCLE TIME`. Для данных с кодами формата `fmt` =

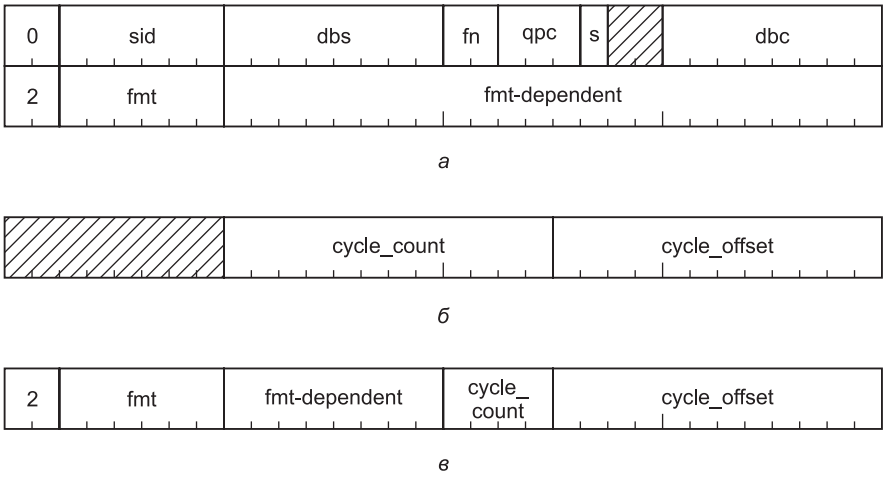


Рис. 26.10. Форматы CIP-заголовков: *а* — двухквадлетный заголовок; *б* — заголовок исходного пакета; *в* — квадлет заголовка с меткой времени

20–3Eh формато-зависимая часть заголовка не определена. Значение `fmt` = 3Fh означает отсутствие прикладных данных и недействительность полей `dbs`, `fn`, `qpc`, `sph` и `dbc`.

Формат сообщений об ошибках изохронных потоков

Сообщения о событиях, регистрируемых в журнале ошибок изохронных передач, имеют формат, приведенный на рис. 26.11. Поля `second_count_hi`, `second_count` и `cycle_count` представляют метку времени — цикл, во время которого произошло событие. Поле `stream_error` раскрывает само событие:

- 0 — первый цикл с действительными изохронными данными после прерывания потока;
- 1 — начало прерывания потока из-за неизвестной ошибки;
- 2 — начало прерывания потока из-за невозможности приема или передачи данных целевым устройством;
- 3 — начало прерывания потока из-за пропуска пакета (только для принимающего устройства);
- 4 — начало прерывания потока из-за CRC-ошибки данных (только для принимающего устройства);
- 5–FFh — резерв.

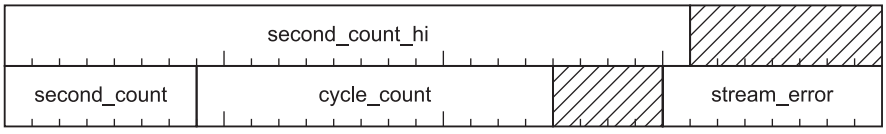


Рис. 26.11. Формат сообщения об ошибке изохронного потока

Описание блоков в памяти конфигурации

Каждый блок узла и его логические устройства описываются в памяти конфигурации. В корневом каталоге памяти конфигурации имеется ссылка на *каталог блока* (Unit Directory). Каталог блока содержит элементы, перечисленные в табл. 26.1. Элементы, описывающие логическое устройство (`command_set_spec_ID`, `command_set_version`, `Logical_Unit_Characteristics`, `Logical_Unit_Number`), присутствуют в каталогах простых блоков (состоящих из одного логического устройства). Для сложных блоков, состоящих из нескольких логических устройств, создаются *каталоги логических устройств* (Logical Unit Directory). Каждый из этих каталогов содержат элементы `command_set_spec_ID`, `command_set_version`, `Logical_Unit_Characteristics`, `Logical_Unit_Number`, описывающие логическое устройство.

Таблица 26.1. Элементы каталогов блока и логических устройств

Ключ	Назначение
12h	`Unit_Spec_ID`, идентификатор спецификации блока (00 609h — блок поддерживает протокол SBP-2),
13h	`Unit_SW_Version`, идентификатор версии ПО (01 048h для протокола SBP-2)
38h	`command_set_spec_ID`, идентификатор спецификации системы команд блока, назначается IEEE/RAC
39h	`command_set_version`, версия системы команд
54h	`Management_Agent`, смещение (в квадлетах) базового адреса управляющего агента относительно адреса начального пространства регистров
3Bh	`Logical_Unit_Characteristics`, характеристики логического устройства (рис. 26.12, *а*): `i` (isochronous) — признак поддержки изохронных операций; `o` (ordered) — упорядоченность асинхронных операций: 0 — порядок исполнения команд произвольный, 1 — строгий; q (queuing) — поддерживаемая модель управления заданиями: 0 — базовая, 1 — полная; `prot` (sequrity protocol) — протокол безопасности: 0 — резервирование логических устройств, 1 — пароль на носитель, 2 — резервирование носителя, 3 — резерв на будущее. `login_timeout` — тайм-аут выполнения (успешного или нет) процедуры входа (в 0,5-секундных интервалах); `ORB_size` — размер (в квадлетах) блока запроса, выбираемого целевым устройством из памяти инициатора
D4h	`Logical_Unit_Directory`, относительное смещение каталога логического устройства
14h	`Logical_Unit_Number`, описание логического устройства (рис. 26.12, *б*): `lun` — логический номер устройства, к которому относится описание; `device_type` — тип устройства (по SPC)
8Dh	`Unit_Unique_ID`, смещение уникального идентификатора блока (необязательный элемент), формат идентификатора приведен на рис. 26.12, *в*. Тело блока имеет длину 2 квадлета, что указано в его заголовке; оно защищено 16-битным полем CRC. Сам идентификатор состоит из идентификатора производителя узла (`node_vendor_ID`) и идентификатора блока (`unit_ID_hi`, `unit_ID_lo`)

Регистры агентов целевого устройства

Регистры агентов целевого устройства обеспечивают инициатору доступ к сервисам целевого устройства. Регистр управляющего агента обеспечивает подключение к сервисам (и отключение) и управление заданиями; регистры выбирающих агентов позволяют ставить задания в очереди на выполнение. Адрес регистра управляющего агента (`MANAGEMENT_AGENT`) инициатор узнает из элемента `Management_Agent` памяти конфигурации целевого устройства. Базовые адреса регистров выбирающих агентов сообщаются инициатору по запросам входа (*secure login*

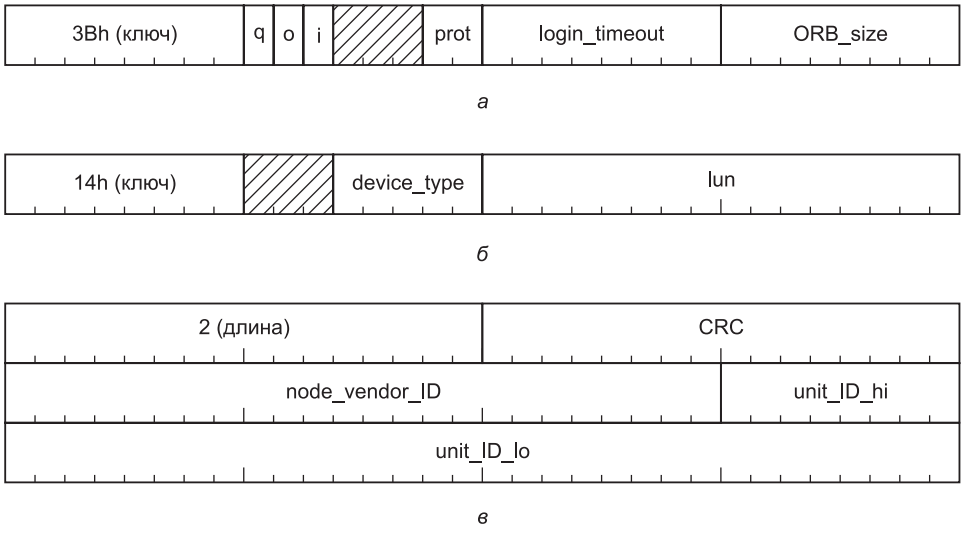


Рис. 26.12. Элементы каталогов блока и логических устройств: *а* — характеристики логического устройства; б — описание логического устройства; *в* — уникальный идентификатор блока

и *isochronous login*). Регистры агентов располагается в целевом устройстве по адресу FFFF F001 0000h или выше.

Регистр управляющего агента

Регистр управляющего агента целевого устройства `MANAGEMENT_AGENT` предназначен для записи 64-битного адреса управляющего запроса. Эту запись выполняет инициатор в виде одной 8-байтной транзакции записи. Успешная запись в регистр приводит к тому, что управляющий агент целевого устройства считает управляющий ORB из памяти инициатора (по указанному адресу) и исполнит его. Если агент оказался занят, то запись не удастся; инициатор об этом получит уведомление (в коде возврата) и должен будет повторить попытку позже. Неудавшиеся попытки записи целевым устройством игнорируются.

Регистры выбирающих агентов

Регистры агентов, выбирающих командные запросы (нормальные и потоковые) и запросы управления потоками, располагаются по адресам FFFF F001 0000h или выше. Каждый выбирающий агент имеет следующие регистры:

- `AGENT_STATE` (смещение 00), регистр состояния. Поле `st` (младшие 2 бита) определяют текущее состояние агента:
  - 0 — *RESET*, сброшен;
  - 1 — *ACTIVE*, активен (выполняет запросы);
  - 2 — *SUSPENDED*, приостановлен (последний запрос цепочки исполнен);
  - 3 — *DEAD*, «умер» (прекращает выборку заданий);

- AGENT_RESET (смещение 04), регистр сброса. Запись любого значения в регистр вызывает сброс агента;
- CURRENT_ORB (смещение 08), адрес текущего ORB. Запись в регистр вызывает выборку ORB по указанному адресу (в состоянии *DEAD* запись игнорируется);
- DOORBELL (смещение 10h), регистр сигнализации обновления списка запросов. Записью любого значения в регистр инициатор указывает агенту целевого устройство на обновление списка;
- STATUS_ACKNOWLEDGE (смещение 14h), записью в этот регистр инициатор подтверждает прием неожиданного блока состояния.

Регистры со смещением 18h–1Ch зарезервированы на будущее.

ГЛАВА 27

Подключение нестандартных периферийных устройств

В этой главе подразумевается знакомство читателя с организацией шин PCI, USB и FireWire, их особенностями, проблемами и путями их решения. Все эти вопросы достаточно подробно описаны в предыдущих главах книги, их можно найти по содержанию и предметному указателю. В этой главе рассматриваются общие вопросы взаимодействия программного обеспечения с периферийными устройствами, интересующие разработчиков нестандартной (несистемной) периферии. В данном контексте нестандартными и несистемными устройствами будем называть те, которые привносят в компьютер новые полезные функции, не имеющие поддержки стандартными средствами операционных систем. Так, например, к системным устройствам относятся устройства хранения данных, коммуникационные устройства (сетевые адаптеры, модемы), принтеры, сканеры, клавиатура, указатели (мыши). Эти устройства могут использоваться любыми приложениями, для чего ОС предоставляет стандартные сервисы, позволяющие приложениям абстрагироваться от конкретных устройств. К нестандартным устройствам можно отнести, например, различные измерительные приборы и комплексы, устройства связи с технологическим оборудованием и прочие устройства, выполняющие ввод-вывод информации в различных вариантах представления. Как правило, для использования этих устройств требуется специальное прикладное ПО и драйверы. Заметим, что драйверы устройств для современных ОС являются весьма сложными конструкциями. При этом сложность написания драйвера обусловлена отнюдь не реализацией функций оперативного взаимодействия приложения и устройства — идеология организации транзакций в рассматриваемых шинах достаточно проста и понятна. Более сложной (и объемной) задачей является реализация вспомогательных функций, чтобы устройство со своим драйвером органично включилось в систему PnP, управления потреблением и прочие подсистемы современных ОС.

Выбор интерфейса

При разработке нестандартных устройств возникает вопрос выбора подходящего интерфейса подключения. Этот вопрос следует решать исходя из принципа разумной достаточности, по возможности отдавая предпочтение внешним интерфейсам. Следует помнить, что разработка аппаратной части устройства (hardware) тесно связана и с программной поддержкой устройств — как модулями ПО, исполняемыми процессором компьютера (software), так и программами встроенного микроконтроллера (firmware), на базе которого, как правило, строятся все современные устройства. Промышленностью выпускается множество моделей микроконтроллеров, имеющих популярные периферийные интерфейсы (USB, FireWire, RS-232, I_2C и другие). Однако в ряде случаев приходится использовать и стандартизованные шины расширения ввода/вывода (ISA, PCI). Эти шины предоставляют более широкие возможности для взаимодействия процессора с аппаратурой, не скованные жесткими ограничениями внешних интерфейсов. Однако за универсальность и производительность внутренних шин расширения приходится расплачиваться более замысловатой реализацией интерфейсных схем и сложностями при обеспечении совместимости с другим установленным в компьютер оборудованием. Здесь ошибки могут приводить к потере работоспособности компьютера (хорошо, если временной). Недаром серьезные производители компьютеров гарантируют работоспособность своих изделий только при установке сертифицированных (ими или независимыми лабораториями) карт расширения. При использовании внешних интерфейсов неприятности в случае ошибок чаще всего имеют отношение только к подключаемому устройству.

Существенным свойством при выборе интерфейса является возможность «горячего» подключения/отключения или замены устройств (Hot Swap), причем в двух аспектах. Во-первых, это безопасность переключений «на ходу» как для самих устройств (их интерфейсных схем), так и для целостности хранящихся и передаваемых данных и, наконец, для человека. Во-вторых, это возможность использования вновь подключенных устройств без перезагрузки системы, а также продолжения устойчивой работы системы при отключении устройств. Далеко не все внешние интерфейсы поддерживают «горячее подключение» в полном объеме: так, например, зачастую сканер с интерфейсом SCSI должен быть подключен к компьютеру и включен до загрузки ОС, иначе он не будет доступен системе. С новыми шинами USB и FireWire проблем «горячего подключения» не возникает. Для внутренних интерфейсов «горячее подключение» не свойственно. Это касается и шин расширения, и модулей памяти, и даже большинства дисков ATA и SCSI. «Горячее подключение» поддерживается для шин расширения промышленных и блокнотных компьютеров, а также в специальных конструкциях массивов устройств хранения.

ВНИМАНИЕ

Карты расширения, модули памяти и процессоры можно устанавливать и извлекать только при выключенном питании компьютера. Но выключения блока питания ATX основной кнопкой недостаточно, поскольку при этом на системной плате остается напряжение 3,3 В. Эти блоки должны обесточиваться по входу (посредством отсоединения питающего кабеля).

Все рассматриваемые в книге интерфейсы имеют достаточно сложные протоколы обмена. Для их аппаратной реализации ряд фирм выпускают специализированные микросхемы. При разработке собственного устройства (и, естественно, его программной поддержки) следует обратить внимание на принципиальное различие возможностей взаимодействия программ и устройств, которые предоставляет подключение по шинам PCI, USB и FireWire.

Реализация интерфейса PCI

При рассмотрении шины PCI становится ясно, что разработка собственных PCI-устройств на логике малой и средней степени интеграции (как это можно было сделать для шины ISA) — занятие неблагодарное. Собственно протокол шины не так уж и сложен, но реализация требований к конфигурационным регистрам проблематична. Серийные устройства PCI, как правило, являются однокристальными — в одной микросхеме размещается и интерфейсная, и функциональная части устройства. Разработка таких микросхем весьма дорогостояща и имеет смысл лишь с перспективами массового выпуска. Для создания отладочных образцов и мелкосерийных изделий ряд фирм выпускают интерфейсные микросхемы PCI различного назначения. Довольно широкий выбор микросхем представлен на сайтах www.ti.com, www.plxtech.com, этой темой занимаются и иные фирмы. Со стороны PCI практически все эти микросхемы поддерживают одиночные целевые транзакции (target transactions), более совершенные модели допускают и пакетные циклы. Более сложные микросхемы выполняют и функции ведущего устройства шины, организуя каналы DMA для обмена с системной памятью. Обмены по этим каналам могут инициироваться как программно со стороны хоста (host initiated DMA), так и с периферийной стороны микросхемы (target initiated DMA), в зависимости от возможностей микросхем. С периферийной стороны встречаются интерфейсы для подключения периферийных микросхем, микроконтроллеров и распространенных семейств микропроцессоров, универсальных и сигнальных. Отметим, что реализовать потенциально высокую скорость обмена по PCI возможно только в режиме прямого управления шиной (bus mastering), организуя длинные пакетные обмены с памятью.

Интересно решение построения интерфейса PCI на конфигурируемой логике FPGA (Field Programmable Gate Array — программируемый массив вентилей). Здесь PCI-ядро, а также функции целевого и ведущего устройств занимают 10–15 тысяч вентилей в зависимости от требуемых функций (см. www.xilink.com, www.altera.com). Микросхемы FPGA выпускаются на 20, 30 и 40 тысяч вентилей — оставшаяся часть может быть использована для реализации функциональной части устройства, буферов FIFO и т. п.

Быстро перевести старые разработки для шины ISA на PCI можно с помощью микросхем-мостов PCI-ISA (см., например, www.iss-us.com).

Подключение устройства к шине PCI/PCI-X позволяет организовать для устройства программное *управление в реальном времени* (по отношению к исполняемому программному коду). Обращением к регистрам устройства можно организовывать

программно-управляемую подачу управляющих воздействий на устройство с временным разрешением порядка единиц микросекунд. Если при формировании воздействий не требуется анализ ответа устройства, то программно можно формировать и более быстрые управляющие сигналы (доли микросекунды). Однако большие объемы данных таким способом передавать не рекомендуется — слишком велика будет загрузка и шины, и процессора. Для интенсивных передач выгодно использовать DMA, для чего устройство должно быть мастером шины. При использовании DMA программные действия (инициализация сеансов обмена) теряют жесткую синхронизацию с физическим обменом с устройством.

Реализация интерфейса USB

Интерфейс USB, несмотря на довольно сложный протокол обмена, вполне доступен для периферийных устройств собственной разработки. Рядом фирм выпускается широкий ассортимент микросхем, со стороны USB различающихся скоростями обмена (LS, FS или HS), числом и возможностями конечных точек (тип передач, размер буфера). Реализация интерфейса USB в периферийном устройстве может быть различной, но во всем многообразии можно выделить два основных подхода:

- микроконтроллер (или программируемая логическая матрица), на котором реализованы основные прикладные функции устройства, имеет встроенный порт USB. Выбор микросхемы для такого решения определяется требованиями прикладной части устройства к производительности процессора, необходимой периферией (аналоговые и дискретные порты ввода-вывода, таймеры, дополнительные интерфейсы) и, наконец, возможностями USB-порта (скорость и характеристики конечных точек). В этом случае разработчику устройства приходится кроме прикладной части заниматься и вопросами взаимодействия по шине USB. Эти вопросы включают как программированием микроконтроллера, так и разработку USB-драйверов устройства на хост-компьютере;
- для организации интерфейса USB используется отдельная микросхема (программируемый микроконтроллер или микросхема с фиксированными функциями), соединяемая с прикладной частью устройства, как правило, параллельной шиной. В этом случае микросхема с USB выполняет сугубо транспортные функции: потоки, посылаемые через конечные точки USB, направляет в прикладную часть устройства (или из нее). В этом случае можно обойти вопросы интерфейса USB (микропрограмму и драйверы), воспользовавшись готовым транспортным решением. Для этого выпускаются микросхемы с фиксированными функциями преобразования интерфейсов, например USB–ATA (включая и режимы UltraDMA), USB–RS-232, USB–Centronics, USB–ECP и другие.

С портом USB выпускаются микроконтроллеры на ядре MCS51, M68HC05, M68HC11 или RISC-архитектуры; они различаются объемом памяти (оперативной и энергонезависимой), производительностью, питанием, потреблением. Микроконтроллеры могут иметь встроенные устройства АЦП/ЦАП, дискретные ли-

нии ввода/вывода общего назначения, последовательные и параллельные порты различных типов. Из этого ассортимента можно выбрать подходящую микросхему, на базе которой разрабатываемое устройство будет реализовано с минимальным числом дополнительных элементов. К микроконтроллерам прилагаются и средства разработки их встроенного ПО (firmware) — самой сложной части такого устройства.

Есть микроконтроллеры с USB, способные работать без программирования энергонезависимой памяти. Так, например, микроконтроллеры серии EzUSB фирмы Cypress Semiconductor позволяют использовать EEPROM малого размера, чтобы хранить в нем только идентификаторы. По включении питания прикладного микропрограммного обеспечения в устройстве нет, и до его загрузки с к компьютера устройство остается функционально «мертвым». Однако при подключении к шине USB устройство предоставляет дескрипторы со своими идентификаторами (VID, PID), по которым с ними связываются специализированные драйверы. Кроме того, устройство «умеет» по специфическому запросу от хоста (*Firmware_Load*) загружать по шине USB микропрограмму из хост-компьютера в свое ОЗУ и запускать исполнение этой программы. В процессе исполнения загруженной микропрограммы устройство может «обрасти» новыми свойствами и дескрипторами. Чтобы хост воспринял устройство в его новом качестве, выполняется так называемая *ренумерация* (ReNumeration™): микропрограмма отключает, а потом снова включает порт USB. Хост это воспринимает как подключение нового устройства, считывает его новые дескрипторы и конфигурирует устройство. Конечно, такая гибкость нужна не всегда, но это особенно удобно при разработке и совместной отладке всех аппаратных и программных компонентов устройства, включая и его микропрограммную поддержку.

Есть и *периферийные микросхемы USB* — порты USB, подключаемые к микроконтроллерам параллельной 8/16-битной шиной данных с обычным набором управляющих сигналов (CS#, RD#, WR#...), линией запроса прерывания и, возможно, сигналами каналов DMA.

Есть и микросхемы USB, сочетающие в себе и USB-функции, и хабы. Все варианты не перечислить, тем более что все время появляются новые микросхемы. Информацию о них можно найти в Сети (www.cypress.com, www.devasys.com, www.iged.com, www.microchip.com, www.netchip.com, www.motorola.com, www.semiconductor.philips.com, www.ti.com, www.natsemi.com, www.intel.com, www.ftdichip.com, www.gigatechnology.com).

Немаловажная часть разработки собственных устройств — программное обеспечение для хост-компьютера, которое доносит до пользователя всю функциональность устройства. В ряде случаев удается воспользоваться готовыми драйверами (например, драйвером виртуального COM-порта для преобразователя интерфейса). В других случаях ПО приходится писать самостоятельно, и хорошо, когда изготовитель микросхем с USB заботится о предоставлении инструментальных

средств разработки всех частей ПО. Так, например, фирма Cypress для своих микросхем с USB (микроконтроллеров и периферийных ИС) поставляет универсальный драйвер, поддерживающий обращения к конечным точкам всех типов (управление по *EP0*, точки для передачи массивов, изохронных обменов и прерывания). При установке этого драйвера в реестре Windows автоматически прописываются предопределенный список кодов производителя и устройства (VID и PID), с которыми будет связан данный драйвер. В дескрипторе устройства его разработчик должен поместить один из вариантов сочетаний идентификаторов, и по подключении данного устройства к шине USB ОС автоматически загрузит его драйвер. Прикладное ПО, связанное с данным устройством, должно обращаться к этому драйверу (напрямую или через промежуточные библиотечные функции). Если разработчик позаботился о создании строкового дескриптора устройства, ОС отобразит его содержимое по подключении данного устройства.

Подключение через USB имеет свою специфику, обусловленную природой USB — хост-центрической шины, подключающей множество устройств через один контроллер. Устройство с интерфейсом USB всегда сугубо подчиненно, всем обменом управляет программа, исполняемая центральным процессором. При этом моменты физического обмена данными с устройством (реального управления устройством) отделены от моментов формирования этих данных программой (то есть подачи управляющих воздействий). Интервал времени между этими событиями — неопределенный и весьма ощутимый: драйвер USB не может (и не будет) мгновенно отдать хост-контроллеру в работу свежепоступивший запрос на передачу (IRP). Во-первых, драйверу приходится должным образом оформлять запрос в виде дескриптора передачи (а для контроллера UHC его требуется еще и разбить на транзакции), на что уходит время. Во-вторых, драйвер должен согласовывать момент времени, в который он вводит новый дескриптор, с циклом работы контроллера, привязанным к кадрам шины USB. В-третьих, запрос попадает в очередь, которая к этому моменту может быть и не пустой — когда именно контроллер доберется до данной передачи, заранее не известно. Таким образом, задержка от момента формирования выводимых данных до их поступления в устройство может в лучшем случае исчисляться единицами-десятками миллисекунд, а в худшем (при большой загрузке шины и процессора) — и того больше. Если программное взаимодействие с устройством требует двунаправленного обмена информацией, то надо принимать во внимание и *взаимную несинхронизированность* обменов с разными конечными точками устройства (см. главу 9). Синхронизацию ввода и вывода обеспечивают только конечные точки типа *Control*, но они не обеспечивают высокой скорости обмена и не всегда достаются разработчику для прикладного использования. Пример-подсказка решения задачи синхронизации — протокол *Bulk-Only Transport* для устройств хранения — описан в главе13. Синхронизация изохронных конечных точек — несколько иная и довольно сложная тема.

Уже из этого поверхностного описания особенностей подключения по USB становится ясно, что программно-управляемые интерфейсы (аналогичные, например,

программной реализации протокола Centronics для параллельного порта) будут работать слишком медленно. Если очередной шаг протокола интерфейса (вывод) выполняется по результату ввода текущего состояния устройства, то разрешение порядка единиц миллисекунд — это слишком оптимистичная оценка его быстродействия, даже для устройств HS (USB 2.0). Для диалогов, критичных к времени, можно использовать конечные точки типов *Interrupt-IN* и *Interrupt-OUT*, что и делается, например, в HID-устройствах. Однако пропускная способность этих точек высока только на HS в USB 2.0. Шина USB предоставляет довольно высокую пропускную способность для потоковых обменов, что и следует учитывать, разрабатывая концепцию взаимодействия с устройством. Взаимодействие будет эффективным, если оно ведется *пакетами*, а не одиночными операциями чтения-записи байтов. Для взаимодействия с регистро-ориентированными устройствами удобно использовать точки типа *Control*.

Прерывания от устройств USB, как это ни парадоксально звучит, планируются хостом: в дескрипторе передачи устанавливается специальный бит-признак, заставляющий хост-контроллер выработать запрос прерывания по ее выполнении. При этом хост-контроллер может быть запрограммирован на некоторую задержку подачи запроса прерывания, что позволяет сократить число прерываний процессора: один запрос от контроллера может сигнализировать прерывания от нескольких устройств.

Передача данных по прерываниям от устройства (через точки типа *Interrupt*) опять-таки планируются хостом. В соответствии с параметрами точки (периодом обслуживания) хост циклически пытается выполнить передачу (ввод или вывод данных). Реально передача происходит по готовности устройства. При этом генерация прерывания процессору может быть как разрешена, так и запрещена. Отметим, что по прерываниям можно как вводить данные (через точки *Interrupt-IN*), так и выводить (через точки *Interrupt-OUT*). О возможности вывода по прерываниям порой забывают.

Изохронные передачи на USB возможны только между памятью хоста и периферийным устройством; ими управляет только хост-контроллер. Непосредственная изохронная передача между устройствами (как в FireWire) на USB невозможна.

Реализация интерфейса FireWire

Разработка собственных устройств с интерфейсом FireWire может оказаться несколько сложнее, чем с интерфейсами PCI или USB. Этому есть несколько причин: круг производителей микросхем FireWire и их ассортимент у **II** же, чем USB; многие спецификации, связанные с FireWire, не являются общедоступными; идеология взаимодействия по шине FireWire существенно отличается от других интерфейсов. Последний тезис развернем подробнее.

Шина FireWire построена для организации равнорангового взаимодействия устройств, среди которых компьютер не является чем-то особенным. В шинах PCI и USB хост-компьютер выполняет конфигурирование всех устройств, а в USB он является еще и единственным планировщиком и исполнителем транзакций. Для устройств PCI и USB системная память хост-компьютера является единственной «перевалочной базой» для всех обменов данными между устройствами. Шина FireWire объединяет узлы, каждый из которых представляет остальным участникам часть своей памяти. Любой узел может выполнять по шине прямой доступ к памяти любого узла. Таким образом, исключается повышенная нагрузка на память (и ее контроллер) какого-то одного узла (хоста).

Кроме вышеописанного распределения памяти шина FireWire располагает к распределению процессорных ресурсов. Периферийные устройства, подключаемые к компьютеру, как правило, используют протокол SBP-2 (см. главу 26). Согласно этому протоколу, забота об исполнении передач, запрошенных инициатором обмена, ложится на целевое устройство. Инициатором, как правило, является компьютер; периферийное устройство является целевым. Это периферийное устройство должно обладать определенным интеллектом, которого должно хватать на реализацию управляющих и выбирающих агентов. Так, например, принтер FireWire, использующий протокол SBP-2, должен сам «добывать» себе данные из памяти узла-инициатора печати. Конечно же, эти данные (связанные списки буферов) подготавливает инициатор. Для устройств бытовой электроники[1] (и для профессиональной цифровой аудио и видеотехники) на FireWire используются и иные протоколы обмена (см. главу 24).

Что касается синхронизации программных действий, происходящих в узлах, и фактических передач данных, то здесь ситуация аналогична USB[2]: между помещением заданий в очередь и их исполнением имеется задержка, неопределенная по длительности. Шина FireWire ориентирована на пакетные передачи; ее пропускная способность будет выше при передачах значительных объемов данных.

Изохронные передачи, используемые для различных мультимедийных приложений, на FireWire существенно отличаются от аналогичных передач по USB. На FireWire они являются широковещательными — одного «вещателя» может слушать произвольное число узлов. «Слушатель» может собирать в один поток несколько вещательных каналов. Изохронные передачи «живучи» — их не прерывает даже сброс на шине. На шине USB возможности изохронных передач значительно скромнее, что обусловлено ее хост-центричностью.

Богатый выбор микросхем FireWire предлагает фирма Texas Instruments (www.ti.com). Ниже перечислены семейства микросхем, реализующих LINK-уровень (это

---

[1] Это не очень удачный эквивалент понятия «consumer electronics», буквально означающего «потребительская электроника». По смыслу сюда относятся устройства, которые в первоначальном назначении не являются периферийными для компьютеров: видеокамеры, плееры, аудиоустройства, телевизоры, аппаратура спутникового и кабельного телевидения и т. п.

[2] Исторически USB появилась значительно позже FireWire.

отражено фирменным названием Lynx), часть которых имеет и интегрированный PHY-уровень:

- iOHCI-Lynx — семейство интегрированных (LINK+PHY) контроллеров OHCI с 1–3 портами 1394/1394a S400;
- iceLynx-Micro — семейство интегрированных (LINK+PHY) микроконтроллеров на процессоре ARM для поддержки аудио- и видеоустройств, включая MPEG2 декодеры;
- PCILynxII — LINK-уровень (S400) с интерфейсом PCI для хост-контроллера или периферийных устройств;
- GP2Lynx — LINK-уровень (S400) общего назначения для камер, принтеров, сканеров;
- CameraLynx — комбинация LINK-уровня (S400) и специализированных схем (ASIC) для камер;
- ceLynx — LINK-уровень (S400) для работы с MPEG2, DirecTV и DV, поддерживает защиту от копирования 5C;
- ceLynx-DV — то же для бытовых устройств, но без защиты от копирования;
- StorageLynx — LINK-уровень (S400) для устройств хранения данных;
- OHCI-Lynx — LINK-уровень (S800) для OHCI.

Все семейства LINK-уровней могут работать с PHY-микросхемами, которые фирма уже (в 2004 году) выпускает для скоростей вплоть до S800, с поддержкой кабельных и шинных (backplane) интерфейсов 1394a/b, с числом портов 1–6. Заметим, что микросхема многопортового PHY может работать повторителем (кабельным концентратором) даже без соединения с микросхемой LINK.

Микросхемы FireWire выпускают и другие фирмы. Например, фирма Oxford Semiconductor (www.oxsemi.com) выпускает микросхемы OXUF922, в которых имеется:

- микроконтроллер ARM Micro (тактовая частота 50 МГц и 8 кбайт памяти, близкой к ядру);
- LINK-уровень IEEE1394b (S800), совместимый и с IEEE1394a для более дешевых устройств;
- интерфейс USB 2.0 (со встроенным физическим уровнем);
- интерфейс ATA/ATAPI-7 с пропускной способностью 80 Мбайт/с для непосредственного подключения устройств хранения;
- приемник изохронного потока со встроенным последовательным аудиоконтроллером;
- высокоскоростной UART для подключения модема и радиоканала (Bluetooth), множество линий ввода/вывода общего назначения, контроллер последовательной шины (2 и 3-проводной) для подключения периферии.

Предшествующие модели (OXF911, 911plus и 912) не имели интерфейса USB, а в плане FireWire поддерживали интерфейсы 1394, 1394a и 1394b соответственно.

Та же фирма выпускает контроллер аудиоустройств OXFW970, в котором имеется интегрированный интерфейс 1394a/b (LINK и PHY S800) и 4 последовательных выходных аудиоинтерфейса, каждый из которых может передавать данные пары каналов. К этим интерфейсам подключаются ЦАП'ы разрядностью 16/24/32 бит; микросхема поддерживает частоты выборок 32, 44,1 и 48 кГц. Микросхема поддерживает форматы аудиопотоков в соответствии со стандартом 61883-6 (см. главу 24).

Микросхемы для FireWire выпускаются и другими фирмами, их можно найти в Сети, например, по ключевым словам «1394 LINK chip».

Алфавитный указатель

I

L

M

N

O

P

Q

S

T

U

V

W

Z

А

Б

В

Г

Д

З

И

К

М

Н

О

П

Р

С

Т

У

Ц

Ш

Э

Гук Михаил Юрьевич

Шины PCI, USB и FireWire. Энциклопедия

Главный редактор	*Е. Строганова*
Заведующий редакцией	*А. Кривцов*
Руководитель проекта	*А. Пасечник*
Технический редактор	*В. Шендерова*
Литературный редактор	*А. Пасечник*
Иллюстрации	*О. Гук*
Корректоры	*Н. Викторова, А. Моносов*
Верстка	*Л. Харитонов*

Лицензия ИД № 05784 от 07.09.01.
Подписано к печати 25.02.05. Формат 70×100/16. Усл. п. л. 43,86. Тираж 3000. Заказ
ООО «Питер Принт», 194044, Санкт-Петербург, пр. Б. Сампсониевский, 29а.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.
Отпечатано с готовых диапозитивов в ФГУП «Печатный двор» им. А. М. Горького Министерства РФ по делам печати, телерадиовещания и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.