ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГОПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ГРАЖДАНСКОЙ АВИАЦИИ»

Кафедра вычислительных машин, комплексов, систем и сетей
Ю.С. Гладышев

ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Часть 1

УЧЕБНОЕ ПОСОБИЕ

для студентов IV курса
специальности 23.01.01
дневного обучения

Москва -2006

ВВЕДЕНИЕ

Основная цель дисциплины «Периферийные устройства» предполагает дать студентам систематизированные сведения о составе, технических характеристиках и принципах действия периферийных устройств, об организации обмена информацией между периферийными и центральными устройствами ЭВМ, а также о системных, локальных и приборных интерфейсах и интерфейсах периферийных устройств.

Несмотря на важность этой дисциплины, всегда существовала и существует проблема по её обеспечению учебно-методической литературой. Изданные в нашей стране учебное пособие Е.Л. Иванова, И.М. Степанова и К.С. Хомякова «Периферийные устройства ЭВМ и систем» (1987г.) и учебное пособие А.М. Ларионова и Н.Н. Горнеца «Периферийные устройства в вычислительных системах» (1991г.) уже не в полной мере соответствуют потребностям настоящего времени.

Учебник (перевод с английского) Ю-Чжен Лю и Г. Гибсона «Микропроцессоры семейства 8086/8088» для студентов высших учебных заведений является наиболее подходящим. В нем содержится много материала, который может быть использован и сегодня в дисциплине «Периферийные устройства». Однако с момента его издания в 1987г. произошла смена нескольких поколений микропроцессоров, были разработаны новые интерфейсы системного уровня (PCI, PCI-X, PCI-Express, AGP), новые периферийные интерфейсы (шины USB, Fire Wire, SCSI).

В этой связи с учетом вышесказанного издание учебного пособия по дисциплине «Периферийные устройства» является актуальным.

Пособие состоит из введения, 5 глав, заключения и списка литературы.

Во введении обосновывается актуальность издания пособия из-за существенного обновления, как аппаратных, так и программных средств вычислительных систем.

В первой главе традиционно определяется роль и место периферийного устройства в вычислительной системе. Используются, ставшие уже традиционными, понятия ядро ЭВМ, СВВ (система ввода/вывода), СУО (средства управления обменом), приводятся примеры устройств, отражающие особенности системы ввода/вывода.

Во второй главе отражены вопросы организации обмена данными в ЭВМ. Определяются функции и каналы ввода/вывода (КВВ). Рассматриваются виды обмена данными: программный по готовности, программный по прерываниям, блоковые передачи и прямой доступ к памяти. Приводятся примеры подсистем ввода/вывода под управлением центрального процессора и под управлением процессора вода/вывода (ПВВ).

В третьей главе рассмотрены интерфейсы периферийных устройств: интерфейс последовательной связи на примере стандарта RS-232C и параллельные интерфейсы «Centronics» и «EPP». Приводятся временные диаграммы взаимодействия, разработанные блок-схемы программ работы процессора ЭВМ и процессора печатающего устройства.

Четвёртая глава включает материал по шинам PCI и PCI-X. В ней отражены вопросы их организации, взаимодействия подключенных к ним устройств, рассмотрены сигнальный протокол и временные диаграммы, команды шины PCI, конфигурационное пространство, организация прерываний и обмена по прямому доступу к памяти.

Пятая глава посвящена популярному интерфейсу USB. Рассматривается архитектура шины USB, режимы и модель передачи данных, структура пакетов, порядок выполнения транзакций. Приводится архитектура хост-контроллера, структура его данных, список кадров, форматы дескрипторов передачи, заголовка очереди, порядок обработки списка дескрипторов. Рассматривается структура устройств с интерфейсом USB, их состояния, а также конфигурирование и управление ими.

Учебное пособие подготовлено на основе курса лекций, читаемого автором студентам факультета «Прикладной математики и вычислительной техники» (ПМВТ) МГТУГА, обучающихся по специальности 230101 «ЭВМ, комплексы системы и сети».

В качестве пожелания расширить объем охватываемого материала, чтобы включить разделы, посвященные непосредственно самим периферийным устройствам.

1. ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА В СОСТАВЕ ЭВМ

1.1 Роль и место периферийных устройств в вычислительной системе.

Независимо от типа любую ЭВМ можно представить в виде центральной части - ядра ЭВМ и совокупности устройств для связи с внешней средой (рис. 1.1, где СУО - средства управления обменом, **УВв**, **УВыв** - устройства ввода и вывода).

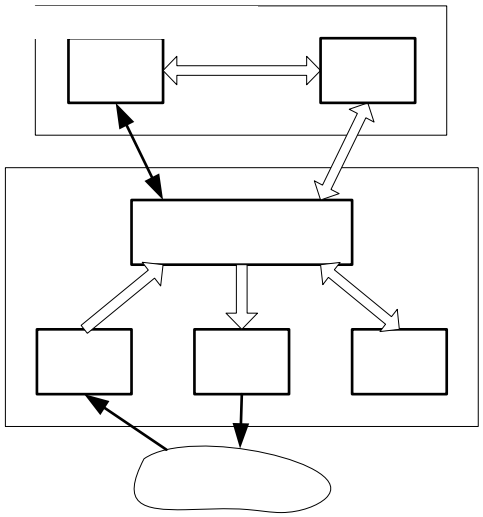


Рис. 1.1. Схема ЭВМ

Ядро ЭВМ включает центральный процессор ЦП (устройство обработки информации) и оперативное запоминающее устройство ОЗУ. Непосредственная связь ядра ЭВМ с внешней средой осуществляется посредством периферийных устройств (ПУ), а организация обменов информацией между ядром ЭВМ и ПУ возлагается на систему ввода-вывода (СВВ) - совокупность аппаратных и программных средств, обеспечивающих обмен информацией между центральной частью ЭВМ и внешней средой, в том числе другими ЭВМ (для многомашинных комплексов). При этом на ПУ, являющиеся компонентами СВВ, возлагаются функции преобразования формы представления информации, используемой в ЭВМ, в форму, пригодную для восприятия объектом или оператором и обратно.

Разнообразие форм представления информации привело к разработке множества ПУ разного функционального назначения, построенных на различных физических принципах, а модульный принцип построения ЭВМ требует такой организации СВВ, которая обеспечивает возможность введения любого ПУ в состав ЭВМ подключением кабеля и загрузки управляющих программ.

Например, результаты вычислений могут быть выведены на экран дисплея, напечатаны на принтере, нарисованы с помощью графического регистрирующего устройства (ГРУ), представлены в виде звука (речевого синтезатора), в виде электрического напряжения и так далее.

При вводе данных в ОЗУ могут использоваться такие ПУ, как клавиатура, планшет, ПУ использующие телевизионный принцип. Имеются ПУ для ввода электрических сигналов, снимаемых с различных датчиков и так далее. В табл.1.1 приведены примеры ПУ.

Таблица 1.1. Примеры периферийных устройств

Примеры периферийных устройств.		
УВв	УВыв	ВЗУ
Клавиатура Планшет Сканер Устройство речевого ввода АЦП Телевизионное устройство	Печатающее устройство Плоттер Дисплей Синтезатор речи ЦАП Модем	НГМД НЖД CD-ROM DVD диск Flash drive

1.2. Особенности системы ввода-вывода

Операции ввода-вывода реализуются с использованием сравнительно медленных ПУ, поэтому на их выполнение затрачивается существенно больше времени, чем на операции обработки, осуществляемые в процессоре. Вычислительную машину недостаточно оценивать по быстродействию ее процессора, измеряемого числом операций, выполняемых в 1сек. Более правильно оценивать ЭВМ по ее эффективной производительности, которая измеряется числом решенных задач заданного класса (включающих как операции обработки, так и операции ввода-вывода) за заданный интервал времени, например за 1час. Повышение эффективной производительности ЭВМ становится одной из основных задач при проектировании, а операции ввода-вывода являются одним из основных сдерживающих факторов ее роста.

Повышение эффективной производительности ЭВМ может быть достигнуто:

1. увеличением номинального быстродействия всех участвующих в операции устройств, в том числе и ПУ;

   Быстродействие ПУ растет медленнее быстродействия процессора и устройств памяти (это связано с использованием электромеханических принципов действия, возможности которых на сегодня практически исчерпаны). Кроме того, быстродействие ряда ПУ ограничено возможностями человека-пользователя;
2. совмещением операций обработки и ввода-вывода;
3. совмещением нескольких операций ввода-вывода, что особенно важно при наличии в системе разнообразных устройств ввода-вывода, значительно отличающихся по быстродействию;
4. совмещением операций, обработки в случае, если операции ввода-вывода не являются сдерживающим фактором.

Наибольшие возможности повышения эффективного быстродействия предоставляют пути совмещения операций обработки и нескольких операций ввода-вывода. Для реализации этих путей современные ЭВМ обладают развитой **СВВ**, т. е. совокупностью аппаратно-программных средств, обеспечивающих выполнение операций ввода-вывода параллельно с операциями обработки. Несмотря на многообразие классов и назначения современных ЭВМ, их СВВ обладают рядом общих черт.

Одно из основных назначений **СВВ** - достижение **совместимости** ЭВМ одного семейства, которая обеспечивается одинаковой функциональной структурой, т. е. совокупностью функциональных элементов и логических связей между ними. Функциональная структура реализуется посредством физической структуры, т. е. совокупности физических компонентов и электрических связей. Между функциональной и физической структурами нет однозначного соответствия, одна функциональная структура может быть реализована различными аппаратно-программными средствами. Функциональная структура **СВВ** одинакова для всех ЭВМ одного семейства.

Отличительная черта функциональной структуры **СВВ** современных ЭВМ-наличие **канала ввода-вывода** (**КВВ**) - функционального элемента, служащего для организации связи и управления обменом между ПУ и внутренней памятью машины.

Совокупность унифицированных правил, реализуемых КВВ для организации связи между ПУ и ядром машины и управления передачей данных, иногда называют **логическим интерфейсом** (**протоколом обмена**).

В зависимости от сложности ЭВМ КВВ может быть реализован в виде:

1. специализированного **процессора ввода-вывода** (**ПВВ**) (в этом случае в физической структуре СВВ имеется специальный подчиненный ПВВ, часто также называемый периферийным процессором или каналом ввода-вывода),
2. совокупности программ и специальной аппаратуры для организации передач информации между модулями ЭВМ (**программный канал**, характерный для мини-ЭВМ и микроЭВМ),
3. совокупности программ и аппаратных средств для организации независимого от процессора доступа в память (**канал прямого доступа** в память, также характерный для мини-ЭВМ и микро-ЭВМ). Дальнейшее развитие указанной выше концепции связано с использованием для организации ввода-вывода машин-сателлитов и построением многомашинных систем.

Физическая структура современных ЭВМ характеризуется принципом **модульности**, который предполагает, что ЭВМ состоит из отдельных устройств-модулей, соединение которых между собой осуществляется по унифицированным правилам. Согласованное сопряжение модулей обеспечивается физическим интерфейсом-системой связей, сигналов и

алгоритмов обмена. Функциональная, электрическая и конструктивная совместимость модулей обеспечивается унификацией интерфейса.

На **СВВ** возлагается решение сложных задач, некоторые из них противоречивые. Такими задачами являются:

1. обеспечение максимальной эффективной производительности ЭВМ, что возможно при использовании автономных средств управления вводом-выводом, ПВВ или отдельных периферийных ЭВМ. Этим достигается возможность практически полного совмещения операций обработки и ввода-вывода, однако значительно возрастают сложность и стоимость аппаратуры;
2. обеспечение минимальной стоимости ЭВМ, что среди прочих мер достигается реализацией функций КВВ программными средствами; при этом возможность совмещения операций существенно меньше, а следовательно, меньше и производительность ЭВМ.

1.3. Совмещение операций обработки и ввода-вывода

Операции ввода-вывода реализуются с использованием сравнительно медленных ПУ, поэтому на их выполнение затрачивается существенно больше времени, чем на операции обработки, осуществляемые в процессоре, поэтому они являются одним из основных сдерживающих факторов роста производительности ЭВМ.

Затраты времени, связанные с формированием кванта информации в ПУ и передачей ее в ОЗУ (или обратно), независимо от типа ПУ можно представить в виде суммы двух интервалов: первый определяет длительность $T_{подг}$ цикла подготовки и преобразования информации в ПУ; второй определяет длительность $T_{прд}$ цикла пересылки подготовленного кванта информации между ПУ и ОЗУ. Для подготовки и преобразования информации характерно использование электромеханического принципа, а пересылка осуществляется электронным путем. Вследствие этого цикл подготовки и преобразования информации значительно продолжительнее цикла пересылки ($T_{подг} \gg T_{прд}$). Именно эта особенность работы ПУ и лежит в основе организации совмещения операций обработки и ввода-вывода.

С точки зрения организации обменов между ПУ и ОЗУ важно деление ПУ на синхронные и асинхронные. Для синхронных ПУ цикл подготовки очередного кванта $T_{Ц}$ информации начинается непосредственно после окончания цикла подготовки предыдущего. Таким образом, длительность интервала между выдачей последовательных квантов информации постоянна и равна ($T_{Ц} = T_{подг} + T_{прд}$).

В асинхронных ПУ подготовка следующего кванта информации начинается по завершении цикла передачи предыдущего, а длительность интервала между выдачей последовательных квантов информации

определяется длительностью подготовки кванта информации, длительностью ожидания обслуживания со стороны КВВ и длительностью передачи ($T_Ц = T_{подг} + T_{прд} + T_{ож}$).

При совмещении операций обработки и ввода-вывода ПУ осуществляет подготовку информации независимо от работы процессора, а связь между ПУ и ОЗУ устанавливается только для передачи подготовленной информации на время $T_{прд}$.

При программной реализации КВВ процессор должен иметь возможность переключаться на выполнение функций канала по сигналу о готовности ПУ; при аппаратной реализации КВВ процессор должен иметь возможность получать информацию от канала о ходе выполнения операции ввода-вывода и об ее завершении. Взаимодействие канала и процессора осуществляется посредством механизма прерываний - процесса переключения процессора с одной программы на другую по внешнему сигналу с сохранением информации для последующего возобновления прерванной программы. Обработка прерывания заключается в запоминании содержимого регистров процессора в специально выделенной области памяти и загрузке этих регистров новой информацией, необходимой для выполнения другой программы.

Важным для организации совместной параллельной работы процессора и ПУ является также **механизм приостановок** - процесс, позволяющий КВВ передавать или получать информацию из ОЗУ без длительной операции обработки прерываний. Приостановки или **занятие цикла ОЗУ** ($T_{озу}$) возникают в том случае, если в ОЗУ должны попеременно обращаться различные устройства (например, процессор и КВВ).

Предположим, что КВВ установил связь между ПУ и ОЗУ и, одновременно в процессоре возникла необходимость обращения к ОЗУ. Поскольку ОЗУ занято, запрос процессора в данный момент удовлетворен быть не может. Работа в процессоре приостанавливается на время, необходимое для освобождения ОЗУ. Максимальное время задержки составляет $T_{озу}$, а среднее время задержки $T_{озу}/2$. Задержки, вызванные приостановками, снижают эффективную производительность процессора, однако это снижение значительно меньше, чем при прерываниях. Приостановки могут быть реализованы только при наличии автономных аппаратных средств обращения к ОЗУ в каждом устройстве (процессоре и КВВ). В том случае, если обращение к ОЗУ со стороны КВВ производится в момент, когда ОЗУ занято процессором, канал получает разрешение на занятие цикла памяти либо по окончании текущей операции процессора, либо при освобождении ОЗУ.

Для сокращения числа прерываний и приостановок в КВВ широко используется **буферизация**. В простейшем случае буферизация достигается уже за счет того, что принимаемые из ПУ байты информации до передачи в ОЗУ каналом объединяются в машинные слова (2, 4 или 8 байт). Наличие

буферных регистров уменьшает вероятность потери информации за счет увеличения допустимого времени ожидания при обращении к ОЗУ.

2. ОРГАНИЗАЦИЯ ОБМЕНА ДАННЫМИ В ЭВМ

2.1. Функции и виды каналов ввода - вывода

Основные функции канала ввода-вывода (КВВ) сводятся к:

1. организации логической связи между ПУ и центральными устройствами на время передачи информации;
2. организации управления автономной работой ПУ;
3. буферизации данных и преобразованию форматов в процессе обмена;
4. контролю передаваемой информации;
5. определению текущих адресов ОЗУ, по которым должна записываться или считываться подлежащая передаче информация;
6. выработке последовательностей синхронизирующих и управляющих сигналов;
7. организации завершения ввода-вывода и отключению ПУ.

Для организации процесса обмена информацией между ПУ и ОЗУ необходимо сформировать и передать компонентам СВВ управляющую информацию, которая определяет тип выполняемой операции ввода-вывода; адрес или номер ПУ, участвующего в операции; внутренний (вторичный) адрес ПУ, т. е. адрес данных или адрес участка носителя информации (см. гл. 3); область ОЗУ, где хранится информация для вывода или куда должна заноситься информация при вводе.

Объем и форма управляющей информации носят специфичный характер для каждого конкретного вида ПУ, а способ передачи ее компонентам СВВ зависит от принципов организации СВВ в машине. Однако принцип модульности требует, чтобы КВВ передавал эту информацию в стандартном для всех ПУ виде; при этом расшифровка информации возлагается на схемы управления самих ПУ.

В процессе выполнения операции может происходить изменение состояния автономно работающих компонентов СВВ, что может влиять на ход выполнения операции. Так, если в какой-то момент возникает неисправность в ПУ или происходит еще какое-либо событие, нарушающее обычный ход выполнения операции ввода-вывода (например, обнаружены ошибка в данных, недопустимая команда, кончилась бумага в принтере и т. п.), то КВВ должен получить соответствующую информацию от компонентов СВВ, проанализировать ее и передать процессору для принятия решения о дальнейшем ходе вычислительного процесса.

По способу аппаратно-программной реализации каналы делят на выделенные (реализованные аппаратными средствами) и встроенные (реализованные программными средствами).

Выделенные каналы позволяют для организации параллельной работы с процессором использовать механизм приостановок и тем самым обеспечить высокую степень совмещения операций обработки и ввода-вывода. Встроенные каналы используют аппаратуру процессора на основе разделения времени, поэтому возможности совмещения операций здесь существенно меньше.

Встроенный КВВ реализует либо **полностью несовмещенный ввод-вывод**, либо совмещение осуществляется посредством **прерывания** программы.

В первом случае все операции по обработке откладываются до завершения операции ввода-вывода, КВВ занимает аппаратуру процессора на все время операции. При этом КВВ позволяет стандартизовать процедуры обмена, упростить программирование ввода-вывода, но не позволяет повысить эффективную производительность ЭВМ.

Во втором случае функции канала реализуются процессором на основе **разделения времени**. Как только ПУ готово к передаче данных, оно передает запрос на передачу данных. Получив этот запрос, процессор прерывает выполнение своей программы, запоминает содержимое регистров для организации возврата и производит загрузку программы ввода-вывода. Затем происходит передача одного слова между ПУ и ОЗУ под управлением программы ввода-вывода, модифицируется содержимое регистров, производится запоминание текущего содержимого регистров для последующего продолжения программы ввода-вывода и осуществляется возврат к программе обработки. При этом на каждое передаваемое между ПУ и ОЗУ слово приходится дважды загружать и разгружать регистры процессоров. Эти «накладные» расходы достаточно велики, и поэтому такой режим целесообразен лишь для медленных ПУ и дешевых ЭВМ.

По способу обращения к ОЗУ КВВ подразделяют на каналы с **прямым** и **косвенным** доступом.

- При **прямом** доступе КВВ имеет собственные схемы обращения к ОЗУ, поэтому приостановки работы процессора происходят только при одновременном обращении к ОЗУ как со стороны канала, так и со стороны процессора.
- При **косвенном** доступе используются общие для процессора и канала схемы обращения к ОЗУ. В этом случае приостановки процессора возникают при каждом обращении КВВ к ОЗУ.

По режиму обслуживания ПУ все КВВ делят на **селекторные** и **мультиплексные**, последние бывают байт- и блок-мультиплексными. Средства селекторного канала монопольно используются одним ПУ в течение всей операции ввода-вывода; средства мультиплексных каналов используются несколькими ПУ на основе разделения времени. Работа этих каналов будет подробнее рассмотрена ниже.

2.2 Программный Ввод/Вывод «ПО ГОТОВНОСТИ»

Каждое периферийное устройство (ПУ) в регистре состояния интерфейса имеет бит готовности, который устанавливается в «1», если ПУ готово к обмену.

ПУ готово к обмену при вводе данных, если буферный регистр полон. ПУ готово к обмену при выводе данных, если буферный регистр пуст.

Обычно состояние бита готовности определяется ассемблеровской командой тестирования (но можно и на любом другом языке).

Типичные операции программного ввода по готовности показаны на рис.2.1.

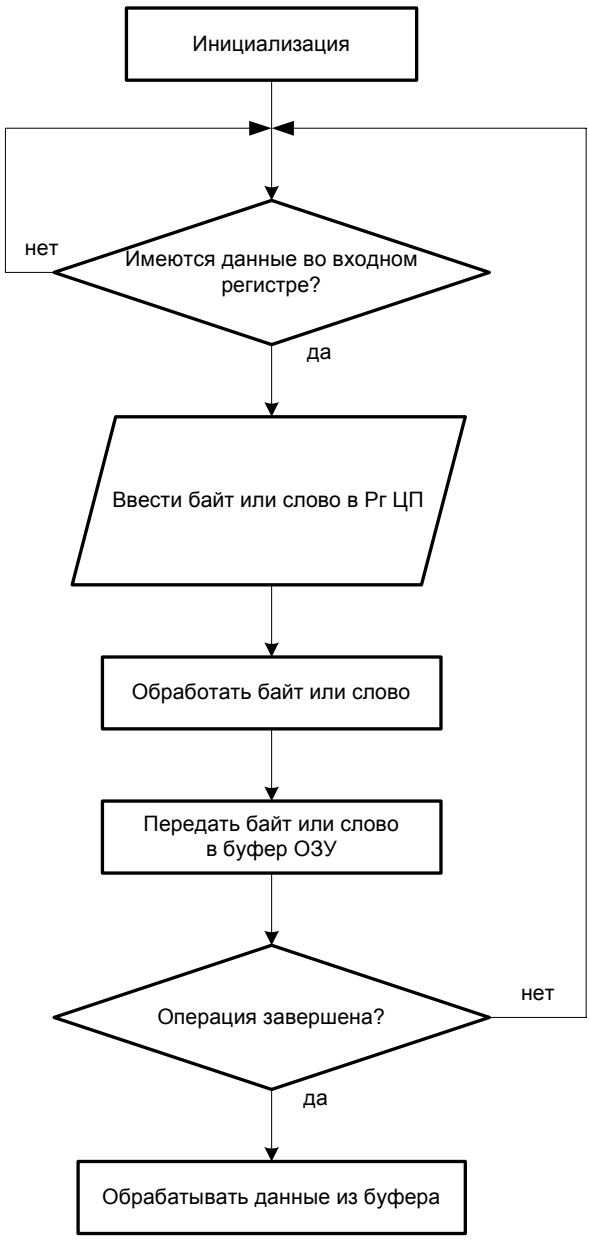


Рис. 2.1. Программный ввод-вывод по готовности.

Характерной особенностью обмена по готовности является наличие петли ожидания. Она образуется при выполнении процессором команды тестирования бита готовности и команды условного перехода. На этом участке программы процессор ждет готовности устройства к обмену. Таким образом, осуществляется синхронизация работы ПУ и ЦП. Аппаратно биты

готовности располагаются в интерфейсной плате, в регистре команд и состояний ПУ.

На рис. 2.2 показано подключение терминала к системной магистрали, содержащей шину данных (ШД), шину управления (ШУ) и шину адреса (ША) посредством интерфейсной платы. Она содержит регистры, которые «прописаны» в адресном пространстве процессора. Интерфейс терминала имеет следующий вид.

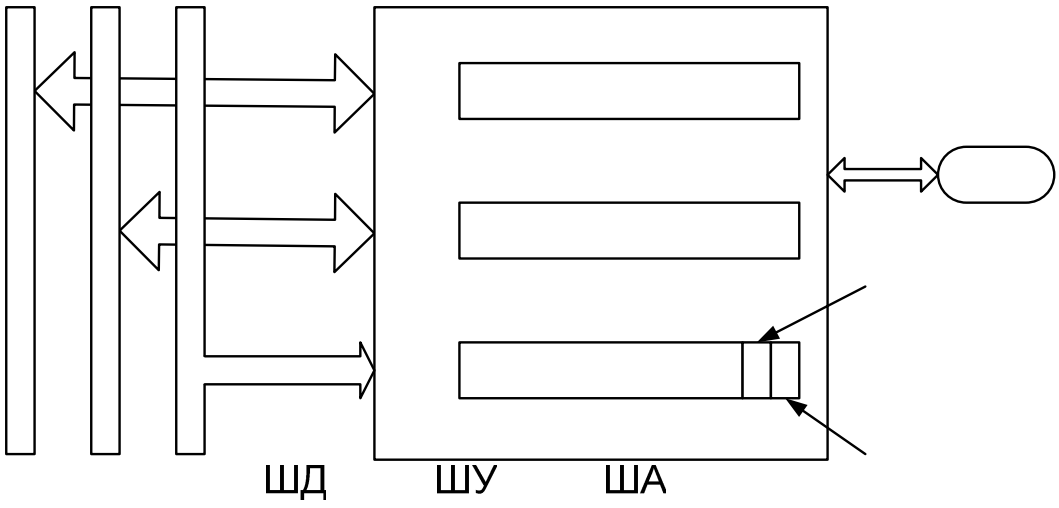


Рис. 2.2. Интерфейс для примера программного ввода-вывода

Пример программирования петли ожидания может иметь следующий вид.

```
        ……….
status      EQU  0054h      ; назначить имена адресам регистров
R_ready     EQU  00000010b  ; интерфейса и битам готовности в регистре
T_ready     EQU  00000001b  ; команд и состояний.
……….
next_in:    in al, status   ;
          test al, R_ready  ;
            jz, next_in     ;
……….
```

Рис. 2.3. Пример программирования петли ожидания

2.3 Программный ввод-вывод «ПО ПРЕРЫВАНИЯМ»

Обмен по готовности полностью занимает ЦП. ЦП не может решать программу пользователя, т. к. тестирует бит готовности. Время тестирования тратится нерационально. Например, если человек печатает на терминале со скоростью 10 символов в секунду, а для ввода каждого символа компьютеру требуется всего 10 мкс, то примерно

$$X = \frac{1000000\text{мкс} - 10\text{мкс} * 10\text{симв}}{1000000\text{мкс}} * 100\% = 99.99\%$$

времени расходуется впустую, поэтому придумали обмен по прерыванию.

При возникновении прерывания микропроцессор вырабатывает следующую последовательность прерывания:

1. Определить тип N.
2. Включить в стек содержимое PSW, CS и IP (именно в таком порядке).
3. Сбросить флаги IF и TF.
4. Передать содержимое ячейки памяти 4*N в IP, а содержимое 4*N+2 в CS.

В результате ЦП перейдёт в режим выполнения подпрограммы обработки прерывания, которая заканчивается командой IRET, по которой осуществляется возврат к прерванной программе.

Существуют два общих класса прерываний: **внутренние** и **внешние**. Типичные внутренние прерывания: деление на нуль, специальные команды. Процедура прерывания аналогична процедуре в том отношении, что переход к ней осуществляется из любой другой программы, а после выполнения процедуры прерывания возврат происходит в прерванную программу.

Последовательность прерывания в МП 8086 можно представить в виде рис. 2.4.

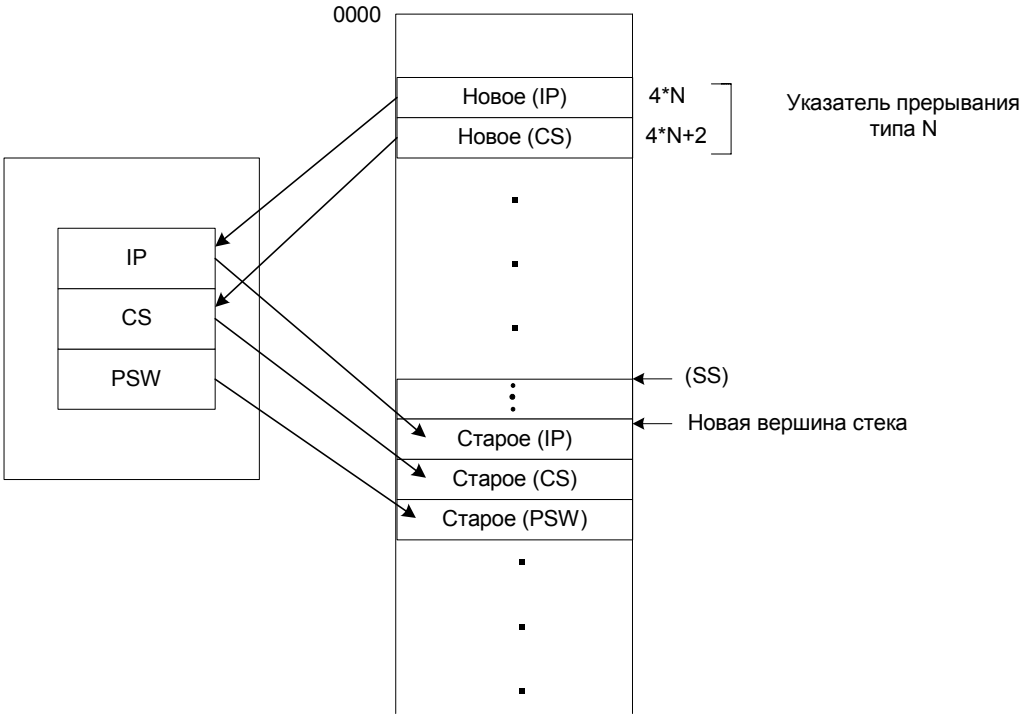


Рис. 2.4. Последовательность прерывания в МП 8086

Двойное слово, в котором находится новое содержимое IP и CS называется **указателем прерывания** (или **вектором прерывания**).

Каждому типу прерывания назначено число N из диапазона 0…255 и адрес указателя прерывания (вектора) получается путём умножения типа (N) на четыре. Для хранения двойного слова требуется 4 байта, поэтому указатели прерываний могут занимать первые 1024 байта памяти и их нельзя использовать для других целей. Некоторые из 256 типов прерываний

резервируются операционной системой и должны инициироваться после включения компьютера.

Виды прерываний и соответствующие им типы изображены на рис. 2.5, показывающим размещение указателей в памяти. Только первые пять типов определены явно, а остальные отведены для команд прерываний или внешних прерываний.

Из рисунка следует, что прерывание типа 0 связано с ошибкой деления. Прерывание типа 1 обеспечивает одношаговую работу и только им управляет флажок **TF**. Если **TF** взведён, по окончании следующей команды возникает прерывание, которое вызывает переход к ячейке, адресуемой содержимым 00004…00007.(B). Так как в последовательности прерывания TF обнуляется, прерывания в процедуре прерывания не возникают, но при возврате восстанавливается исходное **PSW** , в котором бит TF=1, поэтому сразу после команды, выполненной за возвратом, появится прерывание. То есть прерывание будет после каждой команды программы. Одношаговое прерывание используется при отладке. Программист получает при выполнении соответствующие процедуры, содержание основных регистров и ячеек памяти.

Прерывание типа 2 относится к немаскируемому внешнему прерыванию и воспринимается независимо от состояния флажка IF. Они инициируются сигналом, подаваемым на вход **NMI**.

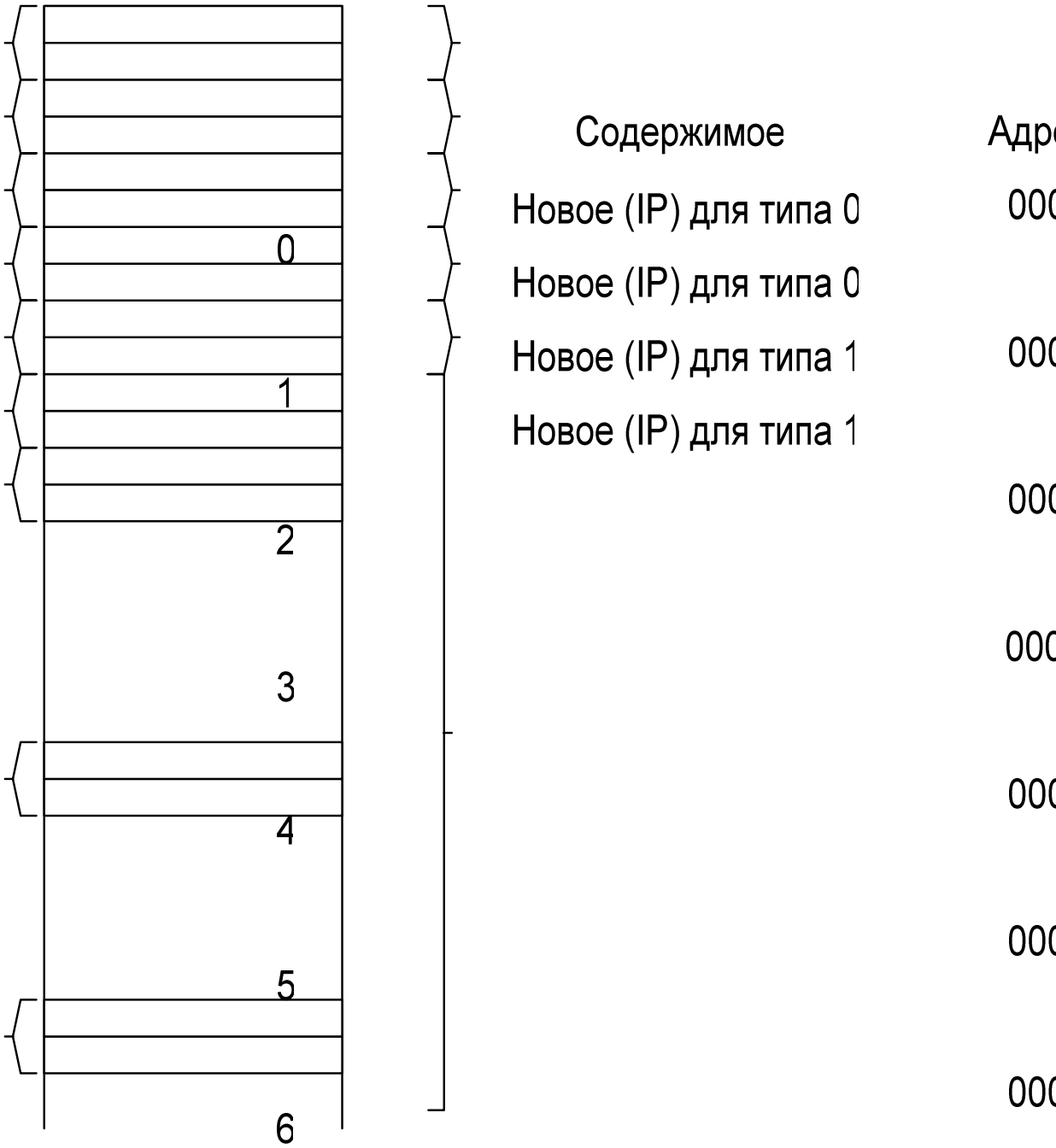


Рис. 2.5. Организация указателей прерываний

Внешние прерывания вызывают сигнал на входе **INT** и **NMI** микропроцессора. Прерывание на входе **NMI** называются немаскируемым: оно вызывает прерывание типа 2 независимо от состояния флажка IF. Немаскируемые прерывания - катастрофические события: отказ питания, внешние события, которые должны обрабатываться немедленно.

Прерывание на входе INT маскируется флажком IF; если IF=0, то прерывание запрещено, если IF=1, то разрешено. ЦП через выход $\overline{INTA}$ возвращает в интерфейс сигнал (подтверждения) разрешения и инициирует последовательность прерывания. Сигнал подтверждения заставляет интерфейс выдать в ЦП (по шине данных) байт, который определяет тип прерывания (N) и, следовательно, адрес указателя прерывания. Указатель, в

свою очередь, даёт начальный адрес процедуры прерывания. Тип N маскируемого внешнего прерывания должен находиться в диапазоне 5…255. Полный процесс обслуживания прерывания представлен на рис. 2.6.

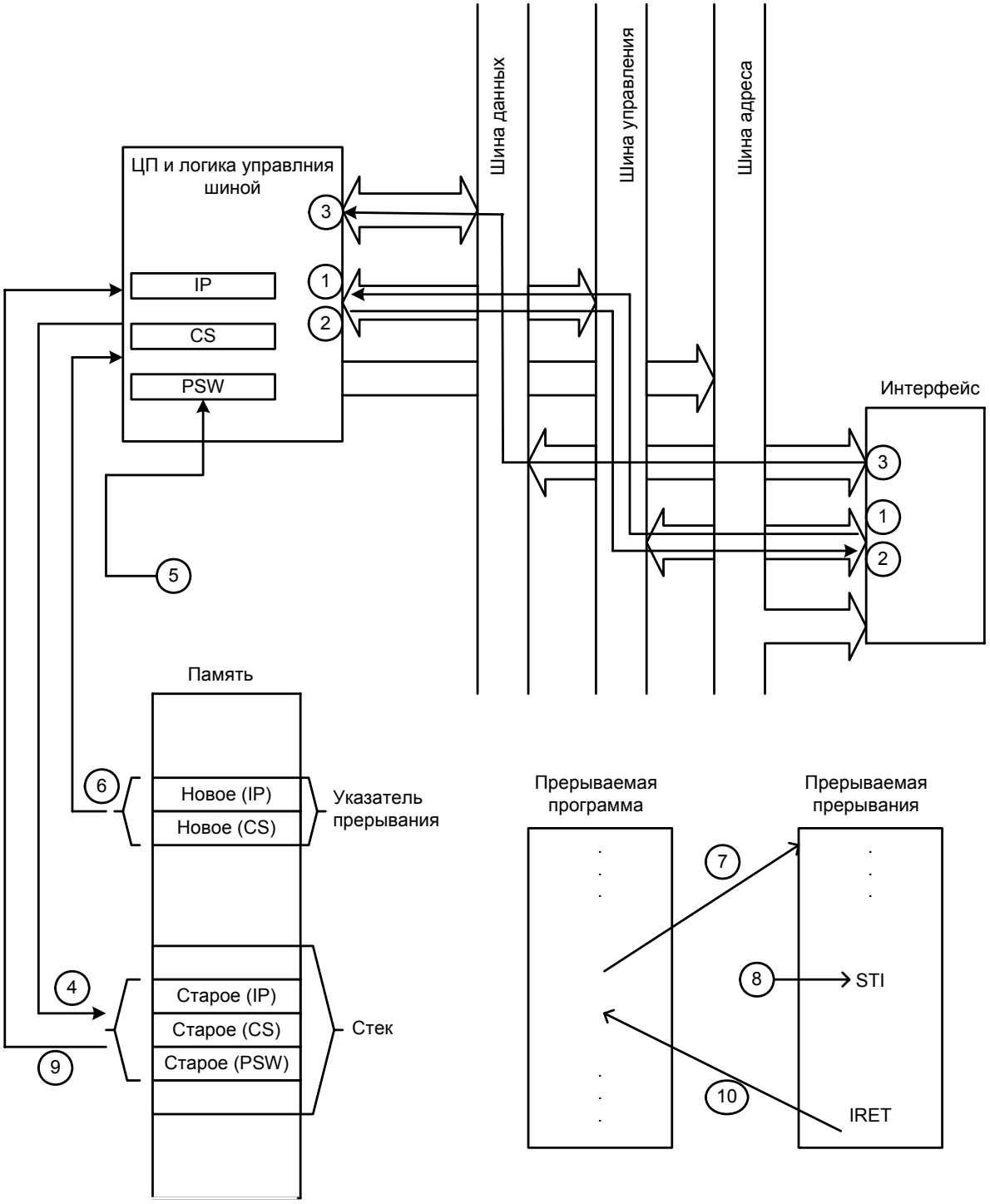


Рис. 2.6. Последовательность событий во время маскируемого прерывания и последующего возврата

1. Интерфейс посылает сигнал прерывания;
2. После завершения текущей команды возвращается подтверждение;
3. В ЦП передаётся тип **N**;
4. Текущее содержимое **PSW**, **CS** и **IP** включаются в стек;

5. Флажки **TF** и **IF** сбрасываются;
6. В **IP** загружается содержимое **4*N**, и в **CS** содержимое **4*N+2**;
7. Начинается процедура прерывания;
8. Прерывания разрешены;
9. Команды **IRET** осуществляет извлечение из стека **IP**, **CS** и **PSW**;
10. Производится возврат в прерванную программу.

2.3.1. Приоритетная цепочка

Имеется несколько способов организации приоритетов при обмене по прерыванию. Рассмотрим следующие средства введения приоритетов в систему прерываний: **опрос**; **приоритетная цепочка**; **схема управления приоритетными прерываниями**.

1. **Опрос** определяет приоритет своим порядком (опроса). Зависит от программы.
2. Простым аппаратным решением введения приоритетов является **приоритетная цепочка** (рис. 2.7).

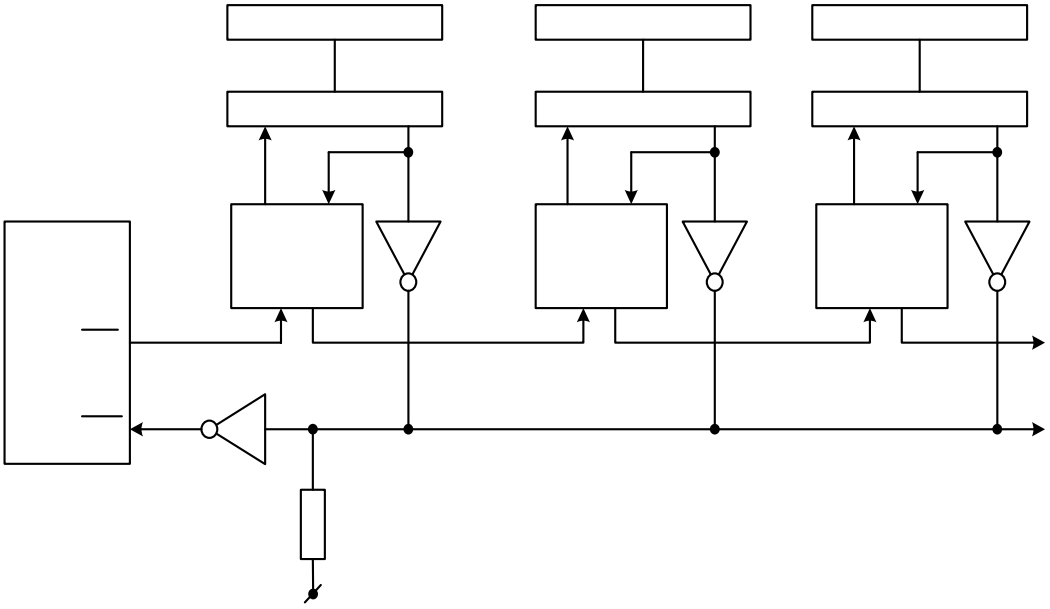


Рис. 2.7. Структурная схема приоритетной цепочки

Если интерфейс выдал запрос, активный сигнал $\overline{INTA}$ заставляет выдать в интерфейс активный сигнал подтверждения прерывания, а передача $\overline{INTA}$ дальше блокируется. Следовательно, ближайший к ЦП запрашивающий интерфейс «перехватывает» сигнал подтверждения, выдаёт тип прерывания N на шину данных и завершает последовательность прерывания. После обслуживания запрос снимается.

Более удалённые интерфейсы, сделавшие запросы, не получают сигнала подтверждения и сохраняют свои запросы.

2.3.2. Программируемая схема управления приоритетными прерываниями

Более гибкий аппаратно-программный механизм приоритетов реализуется программируемой схемой управления приоритетными прерываниями, которая входит в логику управления шиной (рис. 2.8).

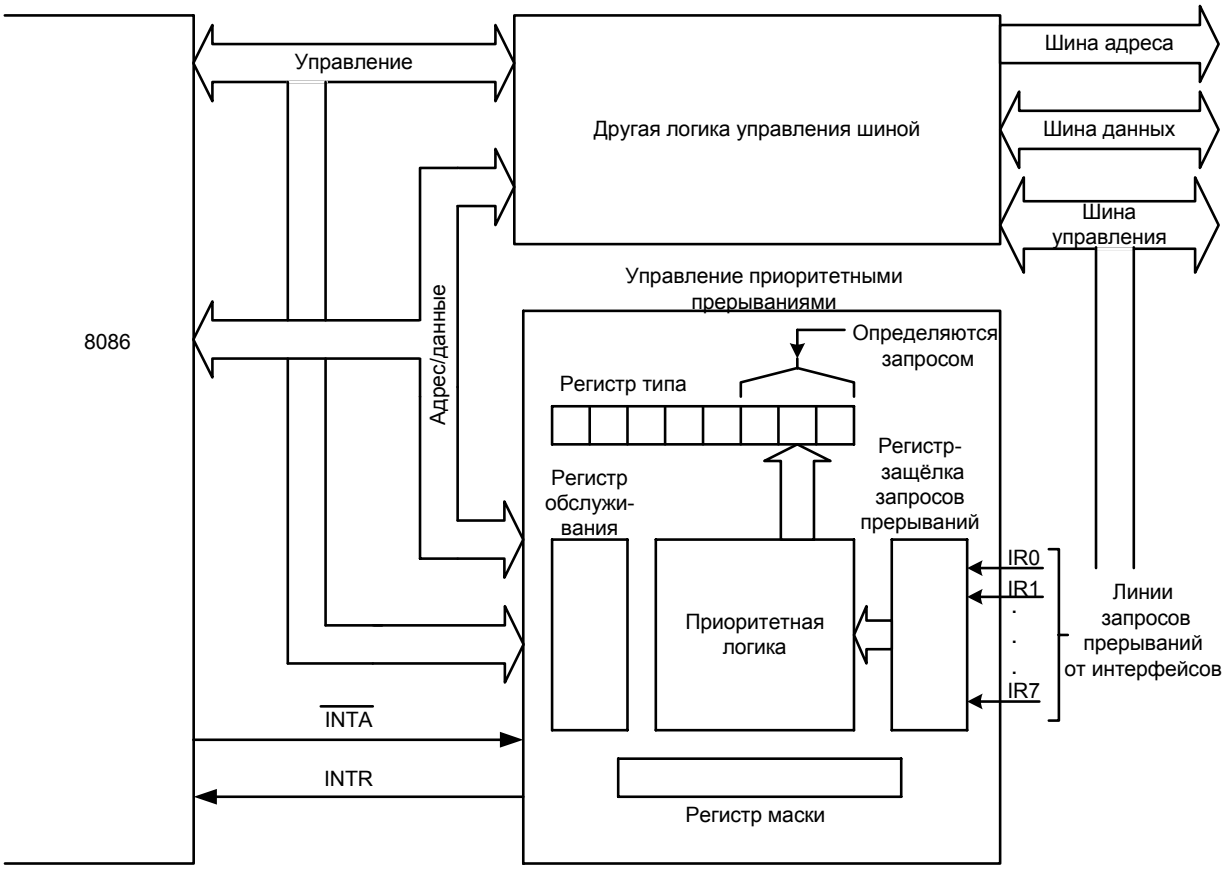


Рис. 2.8. Схема управления приоритетными прерываниями

Блок приоритетной логики (кодирует) («назначает») «распределяет» приоритеты поступающих запросов **IR0**…**IR7**. Когда запрос прерывания распознаётся приоритетной логикой, как имеющий наибольший приоритет, три младших бита в регистре типа устанавливаются на этот номер, устанавливается бит в регистре обслуживания и выдаётся прерывание в ЦП. Если IF=1, ЦП возвращает сигнал подтверждения $\overline{INTA}$ и схема управления выдаёт в ЦП тип прерывания **N**. Все запросы с меньшими приоритетами блокируются до тех пор, пока не будет сброшен бит в регистре обслуживания (это обычно реализует процедура прерывания).

При выполнении процедуры прерывания команда **Sti** повторно разрешает прерывания, и запросы с более высоким приоритетом могут прервать текущую (программу) процедуру прерывания, тогда как запросы с меньшими приоритетами остаются заблокированными до сброса установленного бита в регистре обслуживания. После сброса бита регистра обслуживания процедура прерывания может управлять восприятием запросов с меньшими приоритетами.

Для управления прерываниями биты регистра обслуживания должны быть программно доступны. Кроме того, предполагается, что регистр маски также программно доступен. Бит n в регистре маски маскирует вход IRn. Регистр типа также программно доступен.

Во многих микропроцессорных семействах имеются микросхемы управления приоритетными прерываниями. Для работы с микропроцессорами 8086/8088 предлагается программируемый контроллер прерываний 8259A фирмы Intel. Он аналогичен схеме, показанной на рис.9.

2.4. Блоковые передачи и прямой доступ к памяти

Накопители на лентах и дисках и АЦП работают с высокой скоростью, такой, что быстродействия ЦП при передаче отдельных байт или слов недостаточно. Скорость передачи данных в этом случае должна определяться самими устройствами, а не ЦП.

В дисковом накопителе скорость передачи данных определяется скоростью их прохождения под головкой считывания/записи и часто превышает 200000 байт/с. Следовательно, при передаче в память байта требуется менее 5 мкс. При таком быстродействии требуются блоковые передачи, в которых для непосредственного взаимодействия с памятью применяются КПДП (контроллеры прямого доступа в память).

Действия по передаче по системной шине байта или слова называются **циклом шины**. Выполнение команды может потребовать более одного цикла шины. В течение любого цикла шины системной шиной управляет одна из подключенных к ней компонент. Эта компонента в течение данного цикла является **ведущей**, а компонента, с которой она взаимодействует, - **ведомой**. Обычно ведущим является ЦП, но и другие компоненты могут получить управление шиной, выдавая сигнал в ЦП - запрос шины. После завершения текущего цикла шины ЦП возвращает сигнал разрешения шины, и выдавшая запрос компонента становится ведущей. Получение управления шиной на один цикл шины называется пропуском цикла («**кражей**» цикла). Ведущий во время цикла шины должен помещать адрес на шину адреса и управлять действиями на шине. Компонентами, которые могут стать ведущими шины, являются процессоры (с логикой управления шиной) и контроллеры ПДП.

МП 8086 воспринимает запросы шины по входу **HOLD** и выдает разрешение через выход подтверждения запроса **HLDA**.

Когда запрашивающее устройство получает сигнал разрешения **HLDA**, оно становится ведущим шины и остается им до снятия своего сигнала на входе **HOLD**. Контроллер ПДП, становясь ведущим шины, помещает адрес на шину адреса и посылает в интерфейс необходимые сигналы, чтобы заставить его выдать данные или принять данные с шины данных.

Контроллер ПДП может вернуть управление ЦП после передачи одного данного, а при готовности следующего данного вновь запросить управление. Но он может управлять шиной и до завершения передачи целого блока. Обычно применяется первый способ.

Последовательность действий при выводе одного данного в режиме ПДП иллюстрируется рис.2.9.

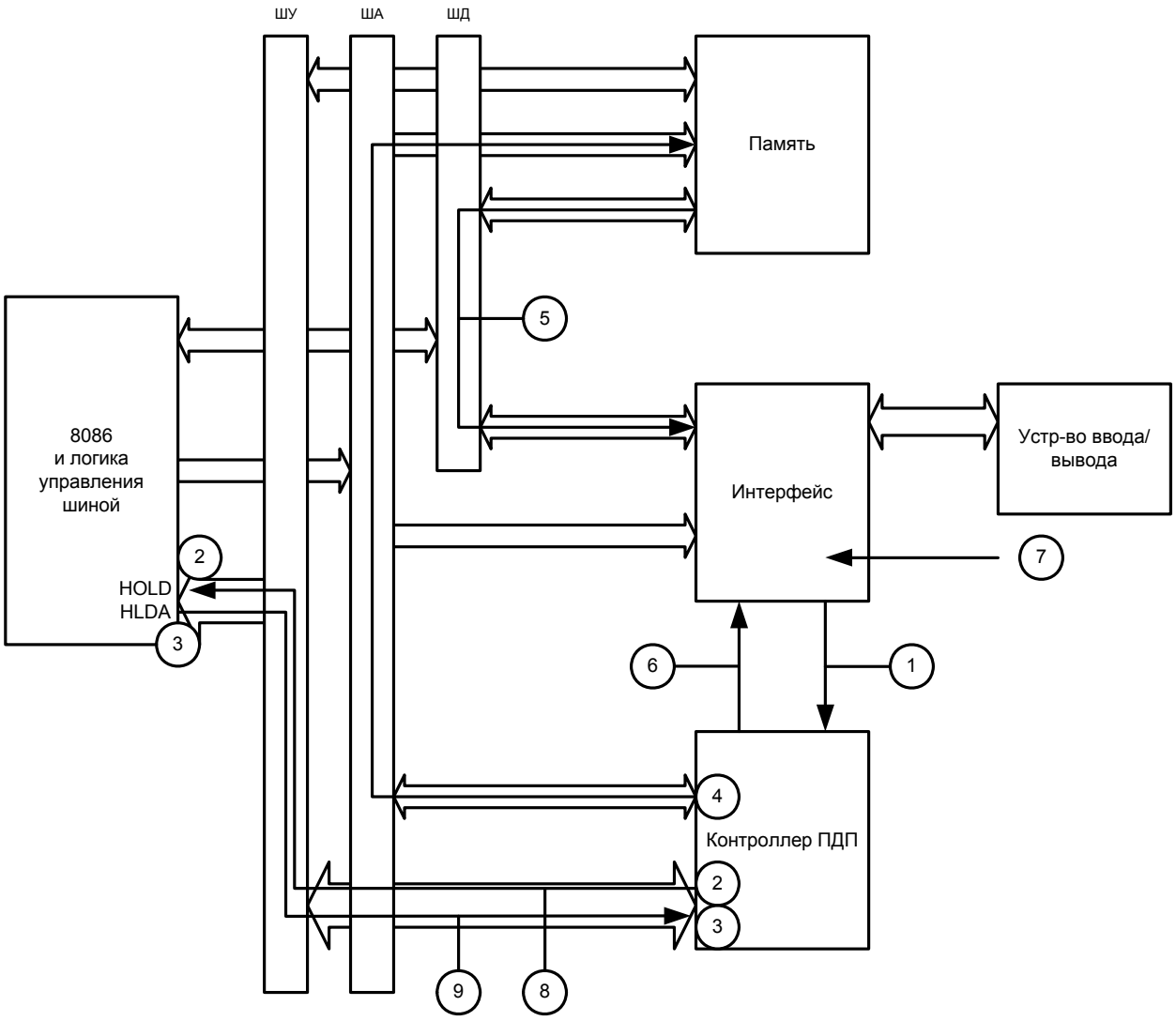


Рис. 2.9. Вывод одного данного в блоковой передаче

1. Интерфейс готов принимать данные и делает запрос ЦП;
2. Сформирован запрос шины;
3. ЦП возвращает разрешение шины;
4. КПДП помещает адрес на шину адреса;
5. Память помещает данные на шину данных;
6. Подтверждается запрос ПДП;
7. Интерфейс фиксирует данные;
8. Снимается запрос шины, и управление возвращается микропроцессору;
9. МП снимает разрешение шины.

Блоковая передача представляет собой последовательность рассмотренных передач одного данного.

Минимальная конфигурация контроллера ПДП и интерфейса представлена на рисунке 2.10

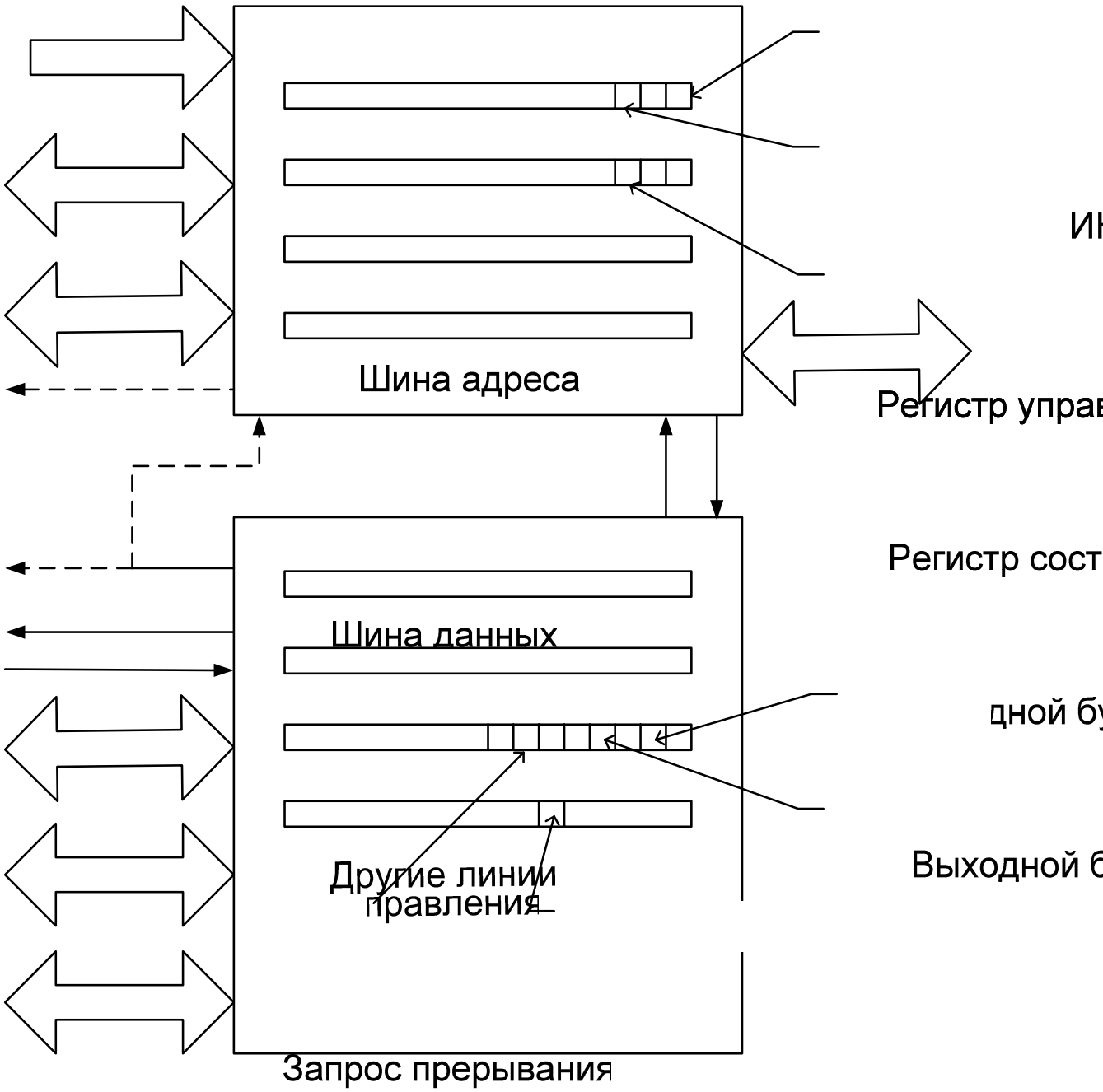


Рис. 2.10 Минимальная конфигурация контроллера ПДП и интерфейса

Во время блоковой передачи одного байта при вводе (в память) имеет место следующая последовательность событий:

1. Интерфейс выдает в контроллер запрос ПДП.
2. Контроллер получает управление шиной.
3. Содержимое регистра адреса помещается на шину адреса.
4. КПДП посылает в интерфейс подтверждение, заставляющее интерфейс выдать данные на шину данных.
5. Байт данных пересылается в ячейку памяти по установленному адресу.
6. КПДП освобождает шину.

7. Инкремент регистра адреса на единицу.
8. Декремент счетчика байт на единицу.
9. Если счетчик байт не содержит нуль, вернуться к шагу №1; в противном случае - остановить передачу.

КПДП имеет **двунаправленную** шину адреса, так как сам формирует адрес. Обмен данными происходит между памятью и регистрами ввода-вывода интерфейса. Двунаправленная шина данных КПДП предназначена для передачи информации в регистры КПДП.

В общем случае интерфейс подключен к устройству, работающему на ввод и вывод. В его регистре управления должен быть бит, указывающий направление передачи. В его регистре управления имеется бит **начать** обмен для инициирования ввода-вывода и бит **устройство занято** в регистре состояния.

В регистрах состояния и управления КПДП необходим бит **разрешения ПДП**, управляющий восприятием запросов ПДП от устройства, и бит, определяющий тип ПДП (освобождается ли шина между передачами, или запрос шины активен во время всей передачи). Имеется бит **направления** передачи.

Для инициирования блоковой передачи необходимо установить управляющие биты интерфейса и контроллера и загрузить начальные значения в регистр адреса и счетчик байт. Последним устанавливается бит **начать** в регистре управления интерфейса.

Типичный фрагмент запуска блоковой передачи при вводе приведен на следующем рисунке. Предполагаются следующие определения бит:

Бит 2 в INSTAT. Бит занятости УВВ (устройства ввода-вывода).

Бит 1 в DMACON. Информирует контроллер о направлении передачи (при вводе содержит единицу).

Бит 3 в DMACON. Разрешает («включает») контроллер, после чего контроллер будет воспринимать запросы ПДП.

Бит 6 в DMACON. Содержит нуль, если шина должна освобождаться между передачами.

Бит 0 в INTCON. Информирует интерфейс о направлении передачи (при вводе содержит единицу).

Бит 2 в INTCON. Бит **начать** ввод-вывод, который запускает действия ввода-вывода.

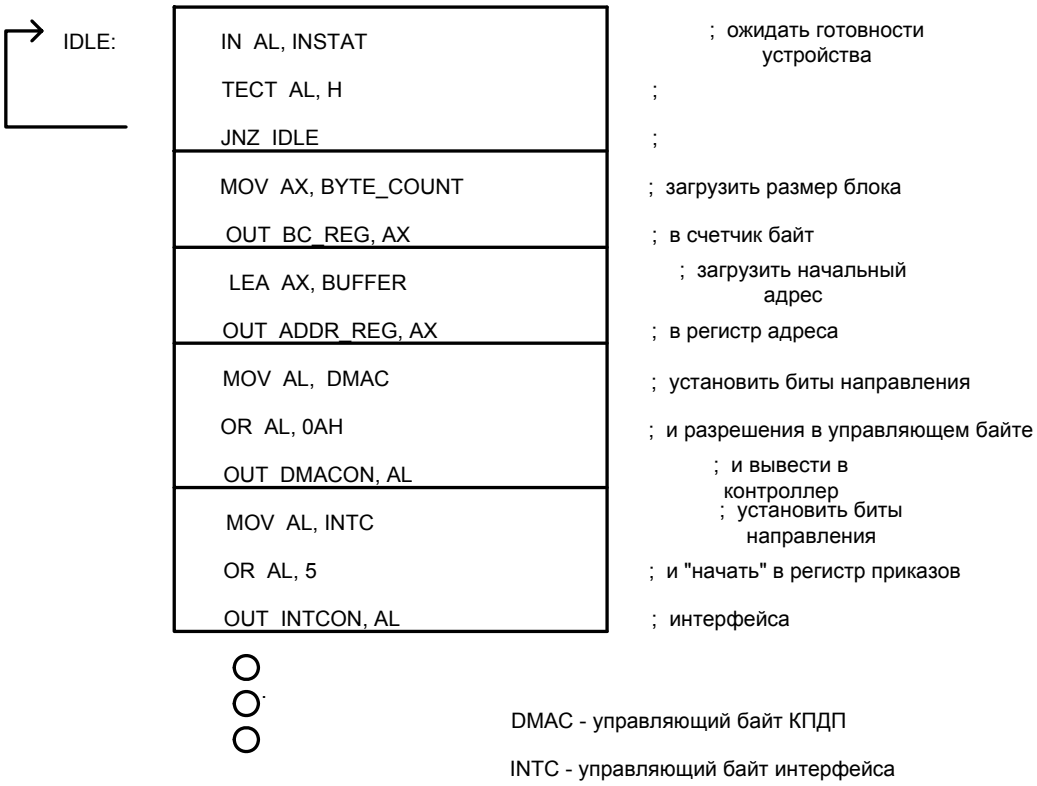


Рис. 2.11. Типичный фрагмент инициирования блоковой передачи

После выполнения рассмотренного фрагмента устройство ввода-вывода начинает ввод данных, а контроллер ПДП запрашивает цикл шины и передает байт из интерфейса в память каждый раз, когда байт помещается в буферный регистр данных интерфейса.

По завершению блоковой передачи в регистре состояния КПДП **DMASTAT** устанавливается бит **закончено**, а на один из контактов контроллера выдается сигнал **закончено**. Текущая программа может обнаружить конец блоковой передачи по этому биту периодическим его тестированием. Можно этот выходной сигнал использовать для инициирования прерывания. В этом случае он подается либо в логику управления шиной, либо посылается в интерфейс, который и обрабатывает прерывание.

После завершения передачи или аварийного прекращения текущая программа или процедура прерывания должны проверить регистры состояния и предпринять соответствующие действия: повторный ввод или сообщение об ошибке. Процедура, выполняющая эти проверки и действия, называется процедурой завершения.

Рассмотрим подробнее возможные схемы реализации арбитража.

2.4.1 Блок последовательного распределенного арбитража

Простейшая схема последовательного распределенного арбитража показана на рис.2.12.

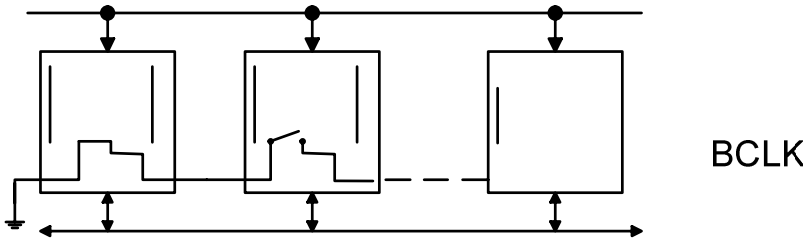


Рис. 2.12. Схема последовательного распределенного арбитража

Вход сигнала $\overline{BPRN}$ арбитра, которому присвоен наивысший приоритет, подключается к точке с потенциалом земли, его выход ($\overline{BPRO}$) подключается к входу $\overline{BPRN}$ арбитра с более низким приоритетом и т. д. Сигнал $\overline{BPRN}$ подается в цепочку устройств постоянно и достигает устройства, желающего стать задатчиком (ЗДТ).

Каждое устройство имеет право выставлять запрос, то есть снимать сигнал $\overline{BPRO}$ на своем выходе по отрицательному фронту тактирующего сигнала BCLK. Снятие сигнала $\overline{BPRO}$ i-ым устройством (что на рисунке иллюстрируется разомкнутым состоянием ключа) становится доступным всем устройствам с более низким приоритетом.

Устройство по отрицательному фронту тактирующего сигнала BCLK формирует сигнал на линии BUSY, то есть «захватит» магистраль при одновременном выполнении условий: отсутствии сигнала на его выходе $\overline{BPRO}$ (данное устройство запрашивает шину), наличии сигнала $\overline{BPRN}$ на его входе (ни одно из более приоритетных устройств не запросило шины), отсутствии сигнала на линии BUSY (шина свободна).

Очевидно, что для правильной работы такой схемы арбитража за один интервал тактирующих сигналов BCLK сигнал запроса (снятие $\overline{BPRO}$) от устройства с высшим приоритетом должен распространяться до устройства с низшим приоритетом, что ограничивает число устройств, подключенных к интерфейсу.

В приведенной схеме арбитража центральный арбитр отсутствует, а взаимодействие схем в отдельных устройствах координируется сигналом BCLK.

2.4.2. Блок параллельного арбитража

Второй возможной схемой является схема параллельного арбитража, реализуемого блоком параллельного приоритета, как показано на рис.2.13.

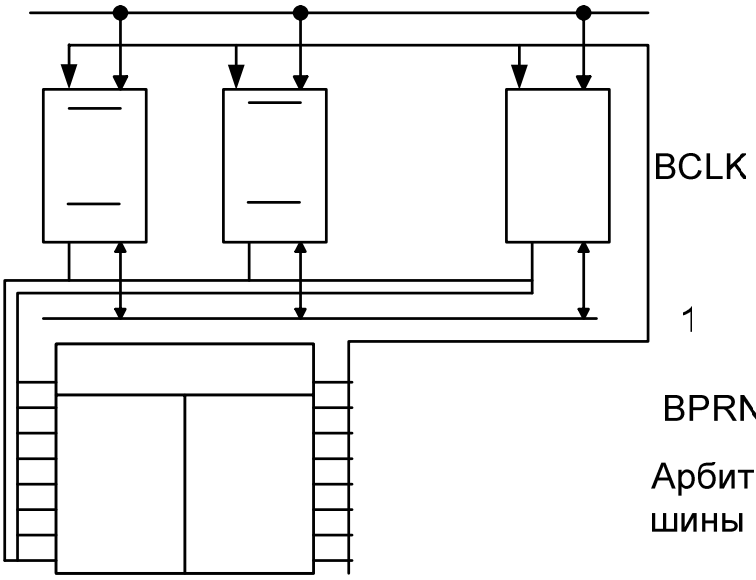


Рис. 2.13. Схема параллельного арбитража

ЗДТ может «захватить» магистраль при наличии сигнала разрешения $\overline{BPRN}$ на его входе и отсутствии сигнала $\overline{BUSY}$ от других устройств. Все устройства могут посылать запросы на использование магистрали в центральный блок параллельного арбитража по индивидуальным линиям $\overline{BREQ}$.

Блок параллельного приоритета состоит из двух частей – приоритетного шифратора CD, определяющего номер наиболее приоритетного устройства, приславшего запрос, и дешифратора DC, выходы которого индивидуальными линиями соединены со входами $\overline{BPRN}$ устройств. Разрешающий сигнал $\overline{BPRN}$ может присутствовать лишь на одном выходе дешифратора. Число устройств ограничено числом входов и выходов блока параллельного приоритета (обычно 8).

Процесс захвата шины, то есть смены ЗДТ, показан на рис.2.14. Все действия тактируются сигналом BCLK.

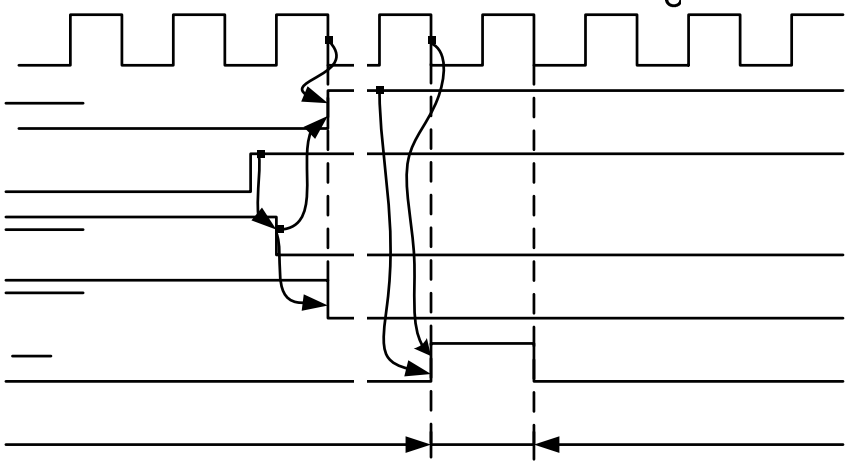


Рис. 2.14. Процесс захвата шины устройством (В)

1. По отрицательному фронту BCLK блок параллельного приоритета, получив сигнал $\overline{BREQ}$(B) от устройства B, приоритет которого превышает приоритет других устройств, снимает сигнал на линии $\overline{BPRN}$ (A) и
2. выдает разрешение потенциальному задатчику B (ЗДТ(B)), то есть сигнал на линию $\overline{BPRN}$ (B). После завершения текущей операции устройством A, текущий задатчик распознает, что у него нет больше наивысшего приоритета (нет сигнала $\overline{BPRN}$ на его входе).
3. Тогда он по отрицательному фронту BCLK снимает сигнал $\overline{BUSY}$, при этом он переводит в состояние высокого выходного сопротивления формирователи адресных, информационных и управляющих сигналов, т.е. отключается от магистрали.
4. После снятия сигнала $\overline{BUSY}$ устройством A, на эту линию выдается сигнал от устройства B.

Задатчик A может удерживать сигнал $\overline{BUSY}$ до завершения монопольного режима обмена.

2.5. Процессор ввода-вывода

Контроллеры и интерфейсные микросхемы значительно упрощают проектирование интерфейсов, но, за исключением передач ПДП, подготовку и саму передачу данных осуществляет ЦП. Для быстродействующих устройств данные передаются с применением ПДП, но все же ЦП должен подготовить контроллер ПДП, инициировать передачу и контролировать состояние по завершению каждой операции ПДП.

Схема общего взаимодействия, когда вводом-выводом управляет ЦП, имеет следующий вид (рис. 2.15):

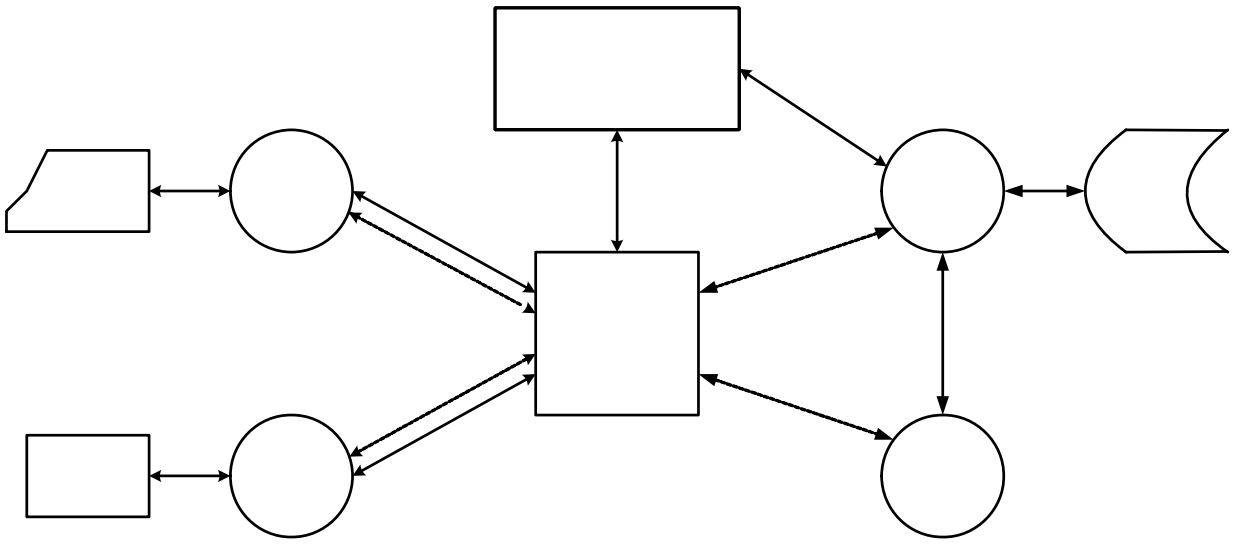


Рис. 2.15. Подсистема ввода-вывода под управлением ЦП

Для еще большего освобождения ЦП от операций ввода-вывода используют подсистему ввода-вывода с процессором ввода-вывода (ПВВ). Структура подсистемы с ПВВ 8089 имеет следующий вид (рис. 2.16).

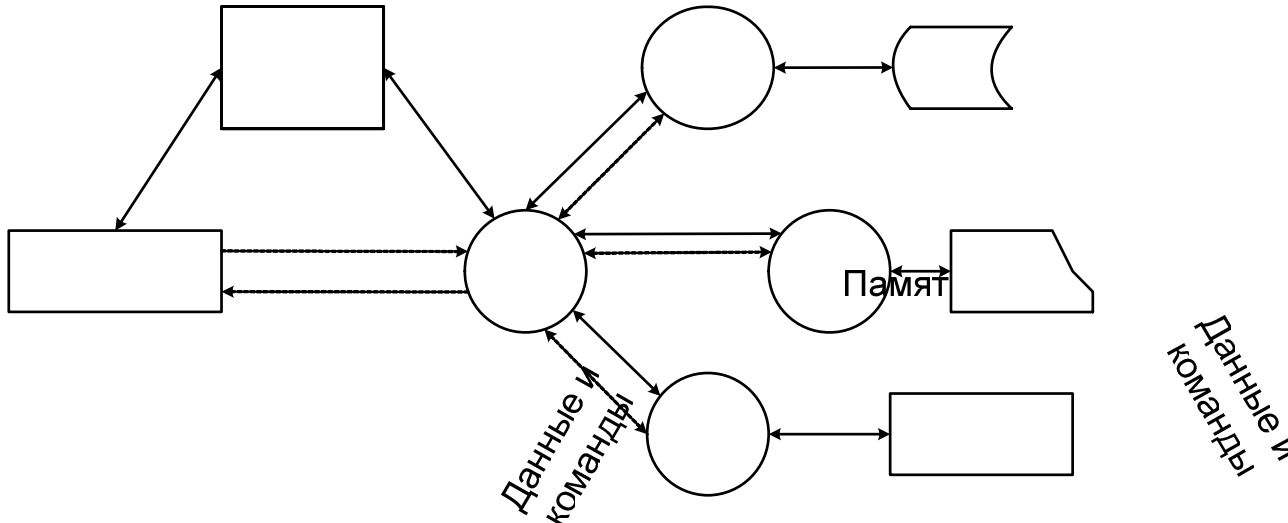


Рис. 2.16. Подсистема ввода-вывода под управлением ПВВ

Процессор ввода-вывода, в отличие от контроллера ПДП, может выбирать и выполнять свои команды. Эти команды ориентированы на операции ввода-вывода, но, кроме того, имеются арифметические и логические команды и т. д.

Совокупность команд ПВВ (составляет) образует «**канальную программу**», под управлением которой работает ПВВ. Канальная программа располагается в ОП. Конфликт ЦП и ПВВ разрешается путем механизма **приостановок**.

ЦП взаимодействует с ПВВ посредством управляющих блоков в памяти (в системной области памяти выделены специальные ячейки). Он готовит управляющие блоки, описывающие канальную программу, ячейку с первой командой программы, а затем передает управление (задачу) ПВВ сигналом, напоминающим прерывание.

ПВВ считывает управляющие блоки для определения стартовой команды (начала канальной программы) и начинает ее выполнять. О факте завершения обмена ПВВ извещает ЦП посредством прерывания или модификации ячейки состояния в памяти.

Таким образом, ПВВ выполняет все действия передач ввода-вывода, включая настройку устройства, программный ввод-вывод и операции ПДП, освобождая ЦП от операций ввода-вывода. В такой конфигурации на ЦП возлагаются задачи более высокого уровня, а ПВВ специализируется на реализации ввода-вывода. Распределенная обработка упрощает разработку аппаратных и программных средств и улучшает производительность и гибкость системы.

3. ИНТЕРФЕЙСЫ ПЕРИФЕРИЙНЫХ УСТРОЙСТВ

3.1. Понятие интерфейса

Для наиболее простого соединения ПУ с шиной компьютера следует использовать порт ввода-вывода. Порт ввода представляет собой ряд тристабильных вентилей, а порт вывода является регистром. Это самая простая аппаратная реализация интерфейса, представлена на рис. 3.1.

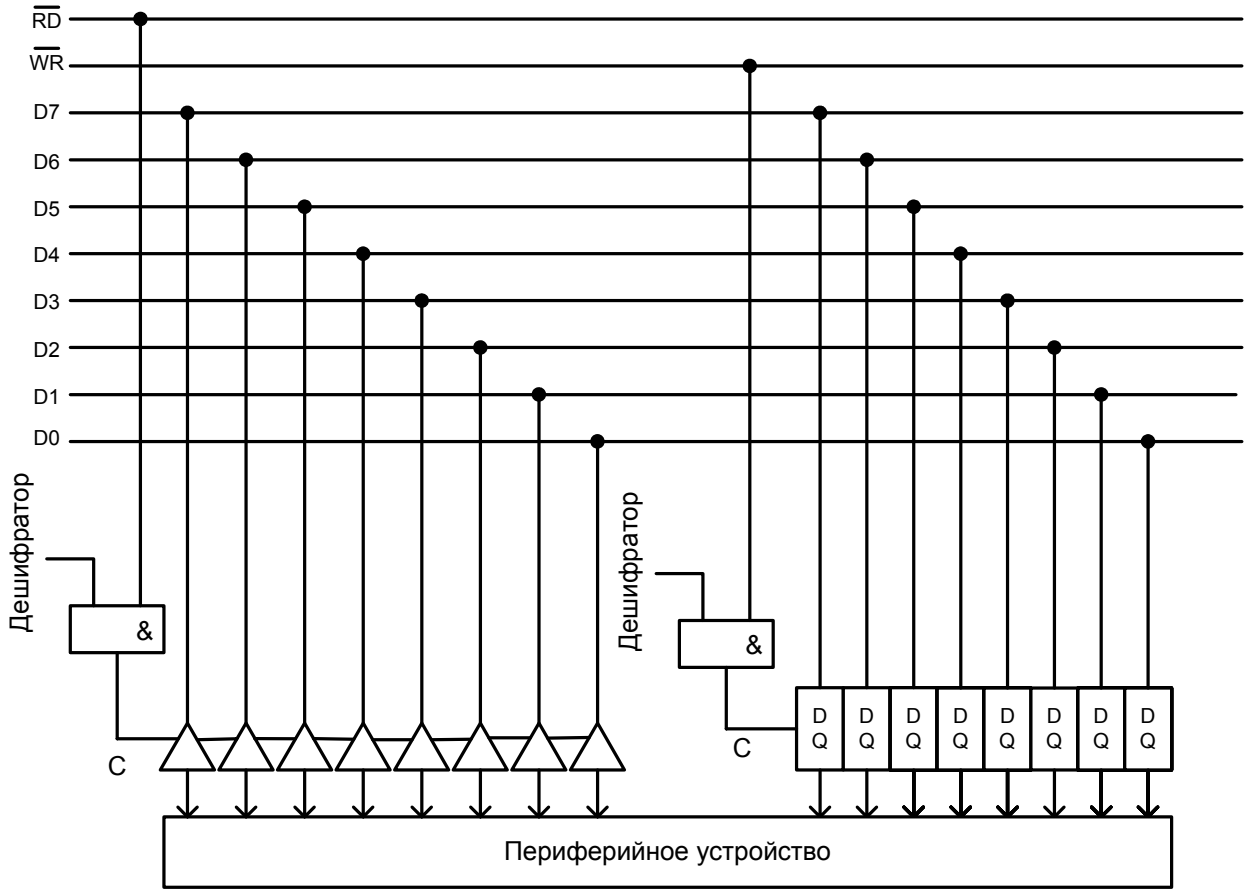


Рис. 3.1. Порт ввода-вывода

В этом случае приём и передача сигналов должен осуществляться под управлением программных средств. Такой способ усложняет программное обеспечение и, кроме того, снижает производительность всей системы. Поэтому разрабатываются специальные аппаратные средства, автоматически выполняющие приём и передачу сигналов. В этом случае программное обеспечение упрощается, а производительность возрастает. Эти аппаратные средства реализуются на кристаллах, называемых интерфейсными.

Если рассмотреть архитектуру вычислительной системы, то можно заметить, что интерфейсы могут быть двух типов (рис. 3.2):

- со стороны шины компьютера
- со стороны периферийного устройства

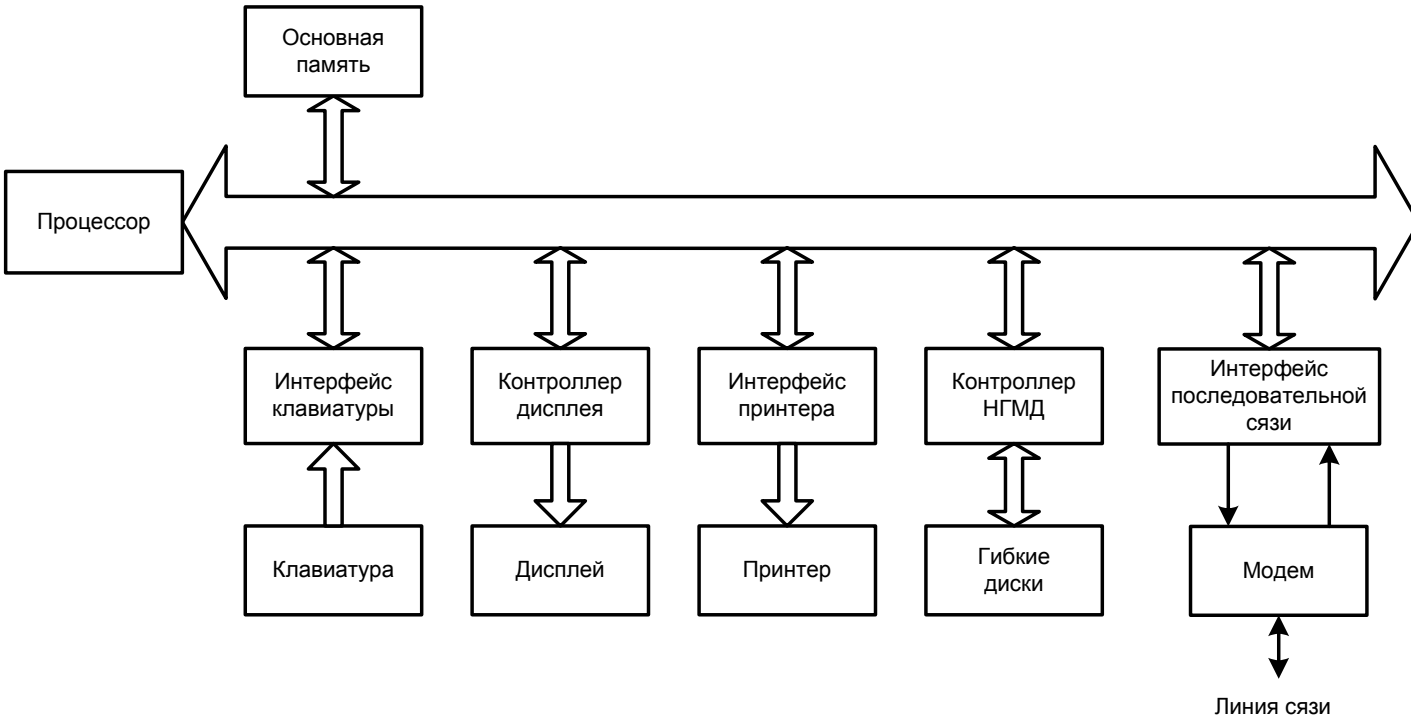


Рис. 3.2. Архитектура вычислительной системы

Характеристики интерфейса со стороны шины компьютера должны удовлетворять процедурам (временным диаграммам) обмена на шине. Характеристики интерфейса со стороны периферийного устройства должны быть согласованы с характеристиками этого устройства.

Каждому ПУ нужен свой интерфейсный кристалл, однако, похожие устройства имеют индивидуальные отличия, которые учитывают программным путём в унифицированном кристалле, пригодным для групп устройств. Отличия определяются с помощью записи соответствующих битовых комбинаций в предусмотренные для этого регистры интерфейсного кристалла.

Для приведения в действие периферийных устройств (например, НГМД или дисплея) обычно требуется сложное управление, при этом желательно, чтобы оно осуществлялось автоматически. Для этого имеется несколько команд, одна из которых указывается компьютером, который записывает её в специальный регистр команд интерфейса. Остальное выполняется интерфейсом автоматически. Интерфейс такого рода часто называют контроллером.

3.2. Стыки

Стыки используются в сетях ЭВМ, а также при непосредственном подключении терминала (клавиатуры и дисплея) к ЭВМ.

Рассмотрим простую сеть, в которой используются модемы и которая представляет собой двухточечный канал.

Модем (модулятор-демодулятор) - устройство, преобразующее последовательные цифровые сигналы в аналоговые и наоборот.

Двухточечный канал - его примером является сеть, в которой два модема соединены с помощью одной линии связи (рис. 3.3).

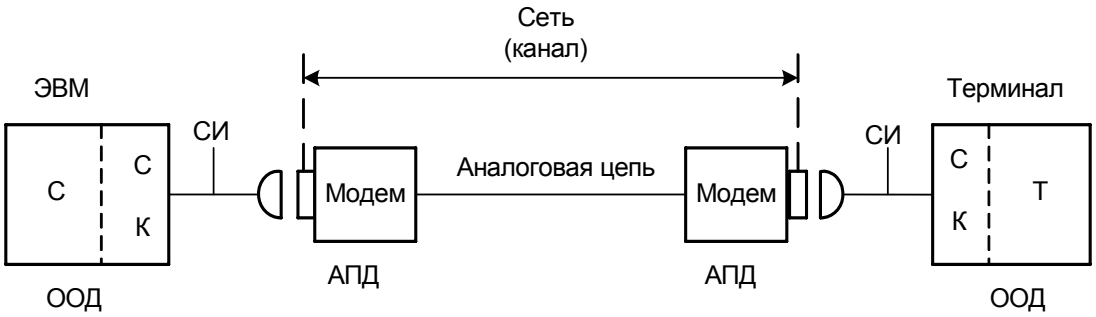
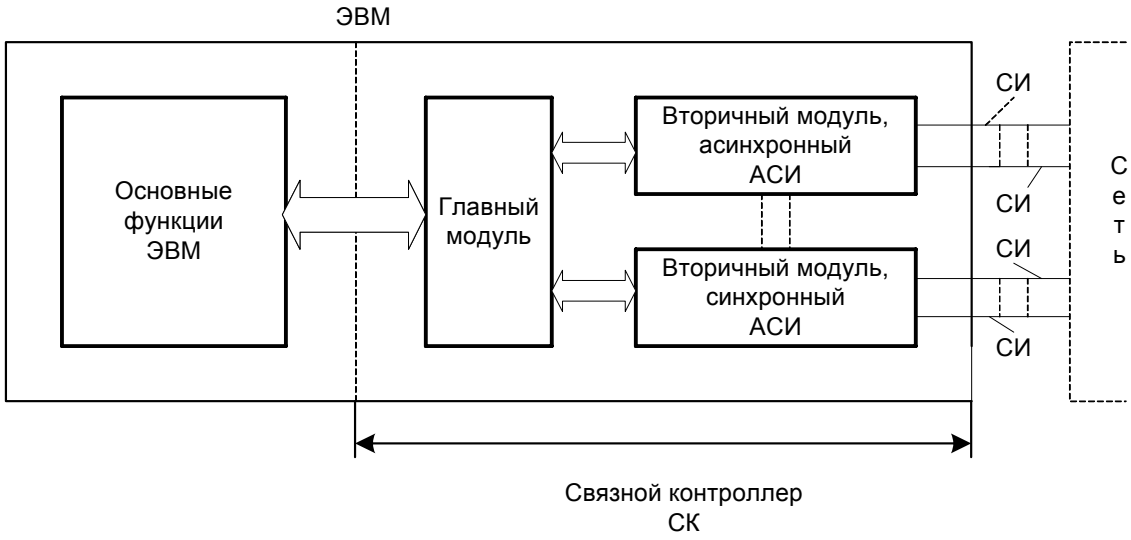


Рис. 3.3. Двухточечный канал

На рисунке 3.3 показаны следующие обозначения:

- **ООД** - конечное оборудование данных; (**DTE** - Data Terminal Equipment).
- **АПД** - аппаратура передачи данных; (**DCE** - Data Communications Equipment).
- **СИ** - связной интерфейс. СИ используется для подключения ЭВМ/терминала к сети передачи данных.
- **СК** - связной контроллер. СК - аппаратура, реализующая часть функций **СИ**, входящая в состав ЭВМ. СК обычно строится по модульному принципу. СК представлен на рис. 3.4.



-

Рис. 3.4. Модульная структура связного контроллера

- **АСИ** - аппаратный связной интерфейс.

Главный модуль в мини-ЭВМ используется не всегда.

Модем имеет два интерфейса (рис. 3.5):

1. Интерфейс между АПД и аналоговой линией.
2. Многопроводной цифровой интерфейс между АПД и ООД (это и есть **стык**).

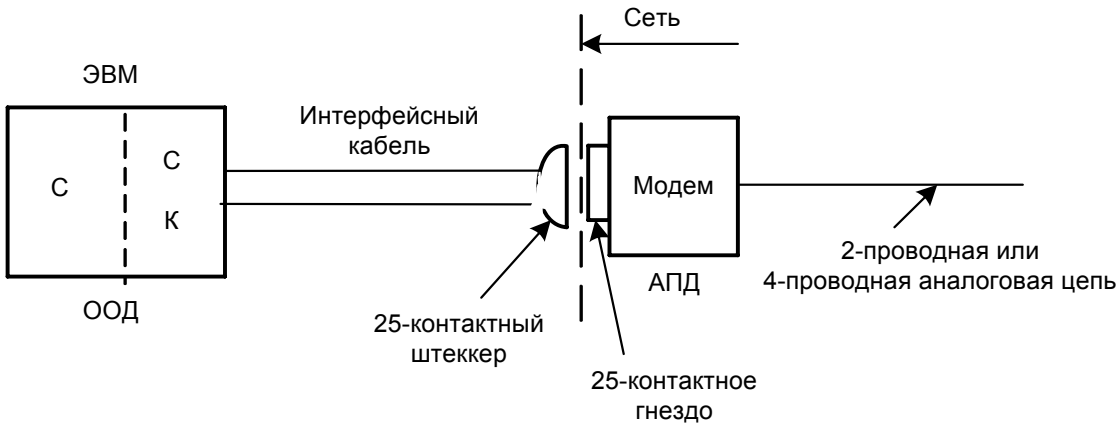


Рис. 3.5. Два интерфейса модема

3.3. Интерфейс последовательной связи

Последовательная связь предусматривает последовательную (по битам) передачу данных. Такая связь может осуществляться между компьютерами или компьютером и другими устройствами. Этот вид связи используется для передачи данных на большие расстояния (в пределах, а с использованием терминальных линий на любые расстояния).

Мы рассматриваем связь между двумя устройствами (1→1). В локальных сетях данные также передаются последовательно, однако, существует отличие от данного случая, оно состоит в том, что между множеством устройств, соединённых каналами связи организуется связь N→N (каждое устройство соединено со всеми остальными).

Существуют 2 режима последовательной связи:

1. дуплексный (рис. 3.6);
2. полудуплексный (рис. 3.7).

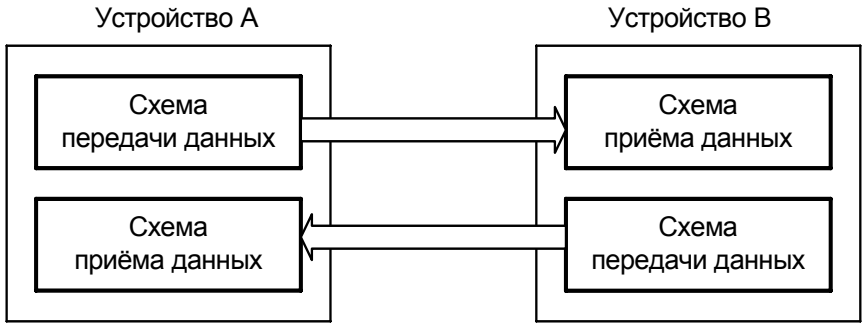


Рис. 3.6. Дуплексный режим

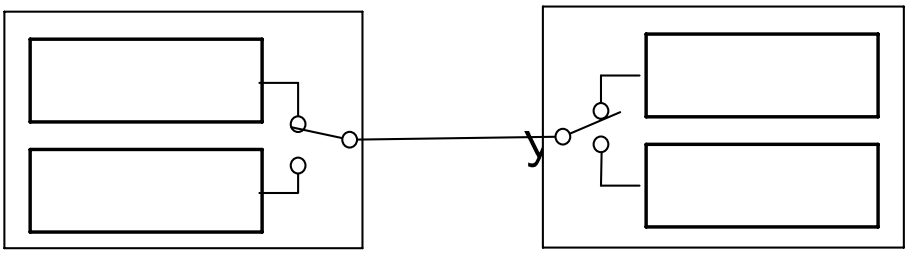


Рис. 3.7. Полудуплексный режим

Наиболее часто последовательная связь используется для передачи данных между компьютером и информационным терминалом (рис. 3.8). В состав информационного терминала обычно входят клавиатура, дисплей (или принтер) и интерфейс последовательной связи.

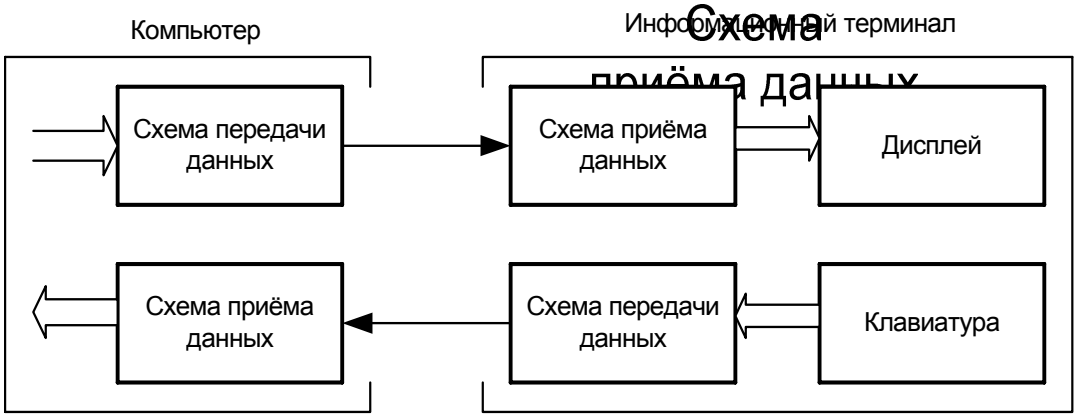


Рис. 3.8. Последовательная связь между компьютером и информационным терминалом

3.4. Асинхронная передача данных.

В асинхронном режиме необходимо указывать начало и конец единицы информации, поэтому в начале передаваемых данных помещают стартовый бит, а в конце - стоповый бит. Когда передачи нет, на линии устанавливается состояние логической «1», при передаче данных стартовый бит равен 0, а стоповый бит равен 1.

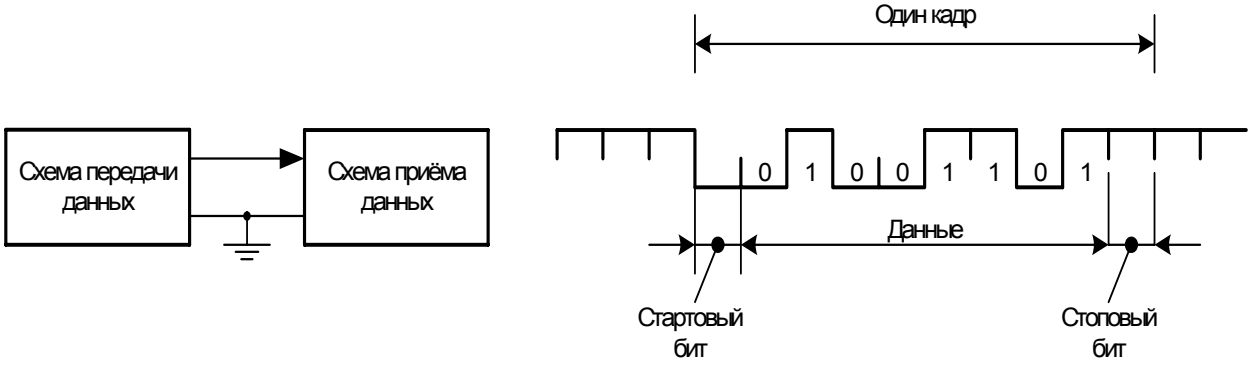


Рис. 3.9. Передача восьми разрядного числа 10110010

На рис. 3.9 показан случай передачи 8-разрядного числа 10110010. Единица передаваемой информации называется кадром.

Приёмник и передатчик не могут использовать одни и те же тактовые сигналы.

Предположим, что скорость передачи данных установлена заранее и равна 300 бит/с, что соответствует интервалу между битами 3,33 мс. В приёмнике проверяются значения входных сигналов несколько короче 3,33 мс. Если в какой-то момент времени входной сигнал переходит из состояния логической единицы в 0, этот момент считается началом кадра. Спустя 1,67 мс после этого проверяется значение входного сигнала. В середине первого бита оно должно быть нулевым (в противном случае переход $1 \rightarrow 0$, зарегистрированный вначале считается сбоем). После этого каждые 3,33 мс отсчитываются остальные 8 сигналов, представляющих собой информацию. Затем проверяется наличие стопового бита и с него идентифицируется момент перехода входного сигнала из 1 в 0. Такая система последовательной связи называется стартстопной системой.

В стартстопной системе тактовые частоты передатчика и приёмника могут отличаться на несколько процентов. Время обнаружения на принимающей стороне часто смещается от положения середины бита (рис. 3.10), и если оно не перекрывает положение соседнего бита в кадре, информация воспринимается правильно.

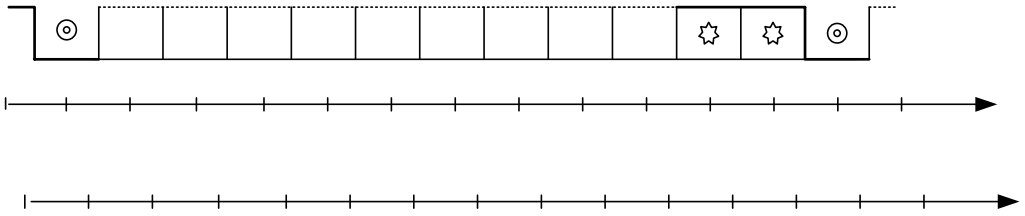


Рис. 3.10. Смещение во времени при передаче данных в стартстопной системе

3.5. Модем

Модем используется для передачи данных по телефонным линиям. Этот вид связи имеет ограничения:

- Используется только пара сигнальных проводов.
- Диапазон частот составляет 300 - 3000 Гц.

Модем, как устройство, имеет передатчик и приёмник. В передатчике двоичные сигналы модулируют периодический сигнал, а в приёмнике демодулируются, то есть восстанавливаются.

Можно привести следующий пример распределения частот при телефонной связи: (табл. 3.1)

Таблица 3.1. Распределение частот при телефонной связи

Режим работы	Передача - приём	Частота метки (частота пропуска) Гц
Вызов (активное) <**CALL**>	Передача Приём	1080±100 750 ± 100
Ответ (пассивное) <**ANSWER**>	Передача Приём	1750±100 1080±100

В этом примере логической единице соответствует сигнал $f_0 + \Delta f$ (частота метки), логическому нулю - сигнал $f_0 - \Delta f$ (частота пропуска).

Модем работает в двух режимах: режим вызова (**CALL**) и режим ответа (**ANSWER**). Если один в режиме вызова, то другой в режиме ответа.

Существуют два способа сопряжения модема с телефонной линией: акустический (рис. 3.11) и электрический (рис. 3.12).

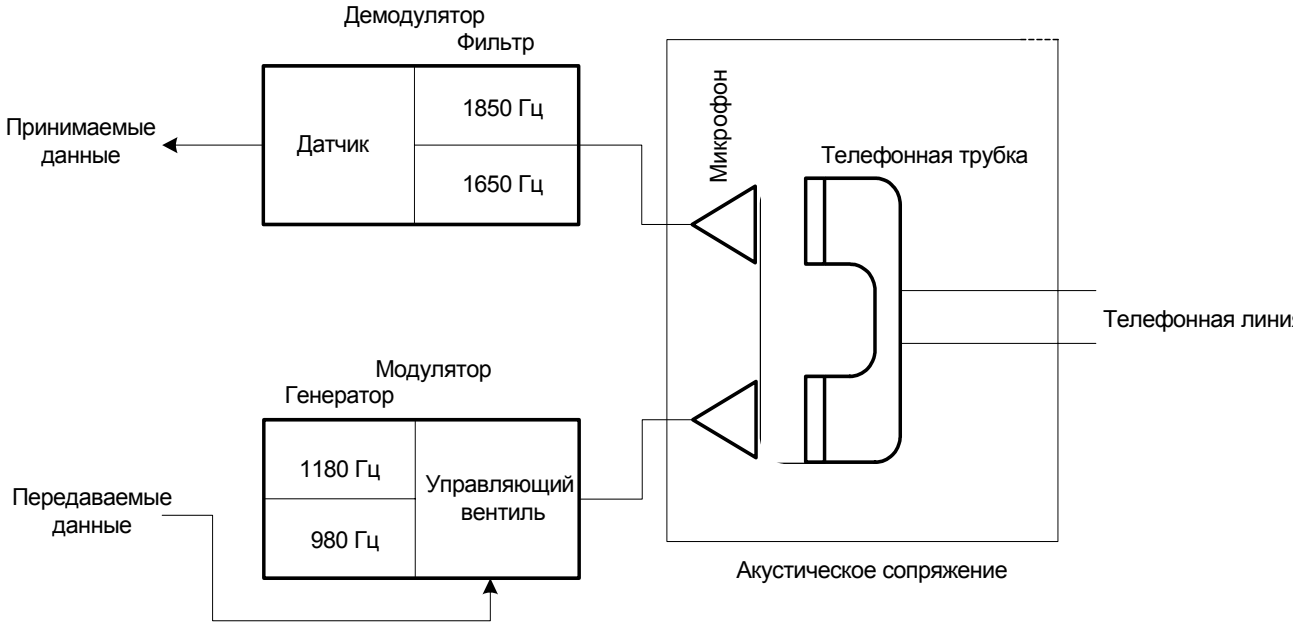


Рис. 3.11. Акустическое сопряжение

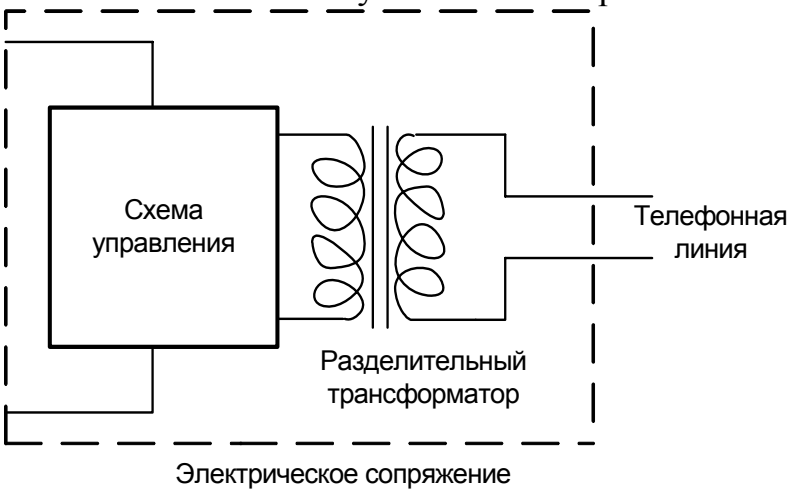
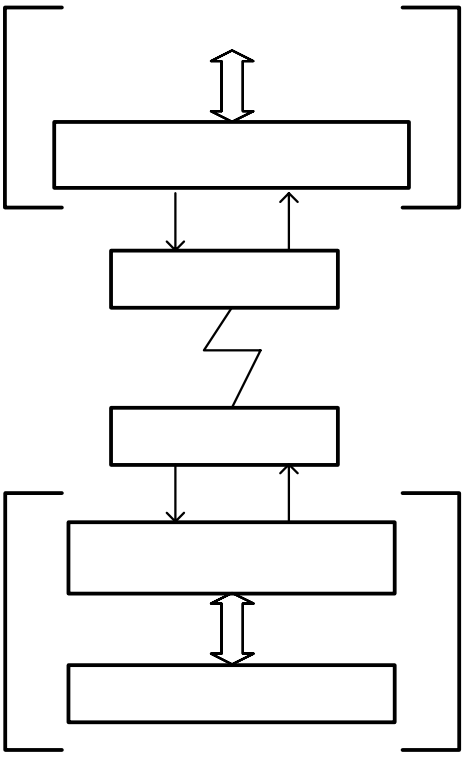


Рис. 3.12. Электрическое сопряжение

3.6. Стандарт RS232C

Модем подключается к информационному терминалу, и, для того, чтобы любой терминал можно было соединить с любым модемом, желательно стандартизировать систему соединений. В настоящее время широко используется американский стандарт RS232C. Модем в этом стандарте называется устройством сопряжения (рис. 3.13).



Процессор

Интерфейс последовательной

Рис. 3.13. устройство сопряжения

Модем

Уровень напряжения на входе устройства сопряжения свыше +3В считается высоким, а ниже -3В - низким уровнем. Уровни ТТЛ получаются использованием схем преобразования уровней.

Напряжение от +5В до +15В на выходе устройства сопряжения соответствует высокому логическому уровню сигналов, а напряжение от -5В до -15В - низкому уровню. Высокий уровень означает, что биты данных равны 0, а управляющие сигналы активные. Низкий уровень означает, что биты данных равны 1, а управляющие сигналы пассивные.

Модем

Имеются 13 соединительных проводов, но из них обычно используются только 9 (рис. 3.14).

Интерфейс последовательной

Клавиатура и дис

Рис. 3.14. Провода, используемые в устройстве сопряжения

По ним передаются данные в линию, принимаются данные с линии и организуется аппаратное управление потоком (flow control по $\overline{RTS}/\overline{CTS}$) в отличие от программного управления потоком (XON/XOFF).

Физический смысл сигналов следующий:

1. Если данные не передаются, то на выходе терминала устанавливается логическая 1. Кроме того, пока все управляющие сигналы ($\overline{DSR}$, $\overline{DTR}$, $\overline{RTS}$, $\overline{CTS}$) не установятся в активное состояние (включено) передача данных не осуществляется.
2. $\overline{RXD}$ - принимаемые данные. Данные передаются из устройства сопряжения в терминал.
3. «Подвешенная земля» - общий земляной провод.
4. $\overline{DSR}$ (**Data Set Ready** - готовность АДП) - готовность устройства сопряжения (источник питания включён, и устройство находится в рабочем состоянии). Терминал по этому сигналу передаёт сигнал запроса на передачу.
5. $\overline{DTR}$ **(Data Terminal Ready) -** готовность терминала, (источник питания терминала включён, терминал находится в рабочем состоянии). В устройстве сопряжения этот сигнал можно использовать для контроля за соединением с телефонной линией (рис. 3.14).
6. $\overline{RTS}$ **(Request To Send** - запрос передачи) - запрос на передачу. Управляющий сигнал, требующий начала передачи данных из терминала в устройство сопряжения. Когда этот сигнал становится активным, включаются средства передачи данных устройства сопряжения (то есть передача несущей частоты).

7. $\overline{CTS}$ **(Clear To Send** – свободно для передачи) - разрешение передачи. Управляющий сигнал, разрешающий начало передачи данных из терминала в устройство сопряжения.
8. Обнаружение несущей частоты (**DCD**). Если несущая из телефонного канала правильно принимается в устройство сопряжения, то этот сигнал становится активным.

На рис. 3.15 показано соединение устройства сопряжения с телефонной линией. Эта схема позволяет уточнить взаимосвязь сигналов $\overline{RTS}/\overline{CTS}$, используемых для управления потоком.

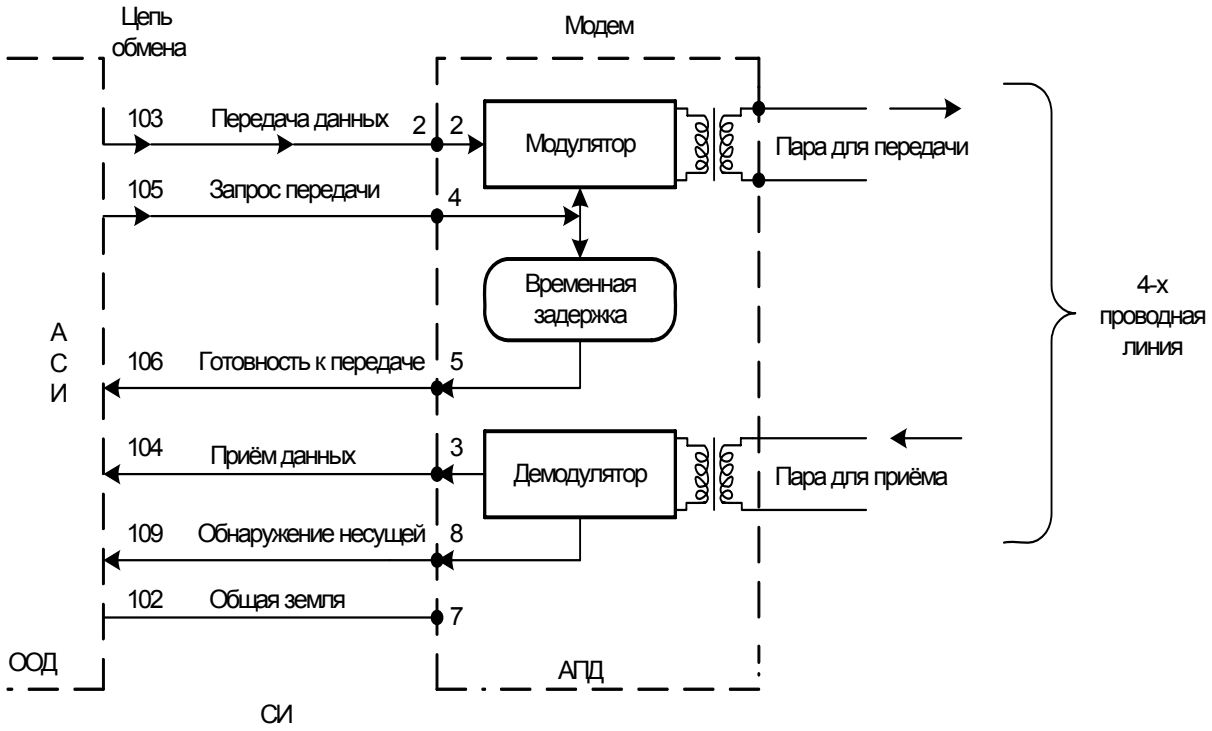


Рис. 3.15. Соединение устройства сопряжения с телефонной линией

103 - передача данных TXD;

104 - приём данных RXD;

109 - обнаружение несущей DCD. Цепь меняет состояние «Выключено» (-V) на «Включено» (+V)

105 - запрос передачи ($\overline{RTS}$ – **Request To Send**). $\overline{RTS} \rightarrow$ разрешение модулятору на вход в линию

106 - готовность к передаче ($\overline{RFS}$ **- Ready For Sending**) или ($\overline{CTS}$ **- Clear To Send**)

Эта схема также помогает понять конструирование нуль-модемных кабелей. Дело в том, что интерфейс RS-232C широко используется для подключения к ЭВМ периферийных устройств (например мышей). В этом случае модем не требуется, поэтому для удовлетворения требованиям интерфейса его нужно смоделировать, говоря другими словами «обмануть» интерфейс так, чтобы он считал, что модем подключён и всегда готов к передаче. Это делается с помощью нуль-модемных кабелей, варианты возможных структур которых приведены на рис. 3.16.

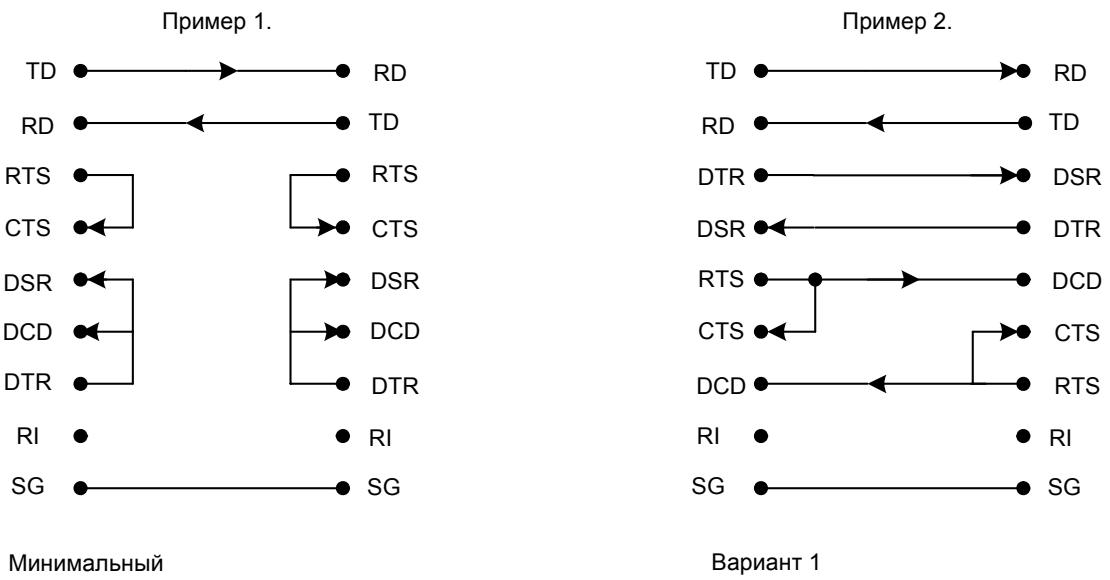


Рис. 3.16. Схемы нуль-модемных кабелей

При этом временные диаграммы вывода данных в линию и ввода данных с линии имеют вид как на рис. 3.17 и 3.18 соответственно.

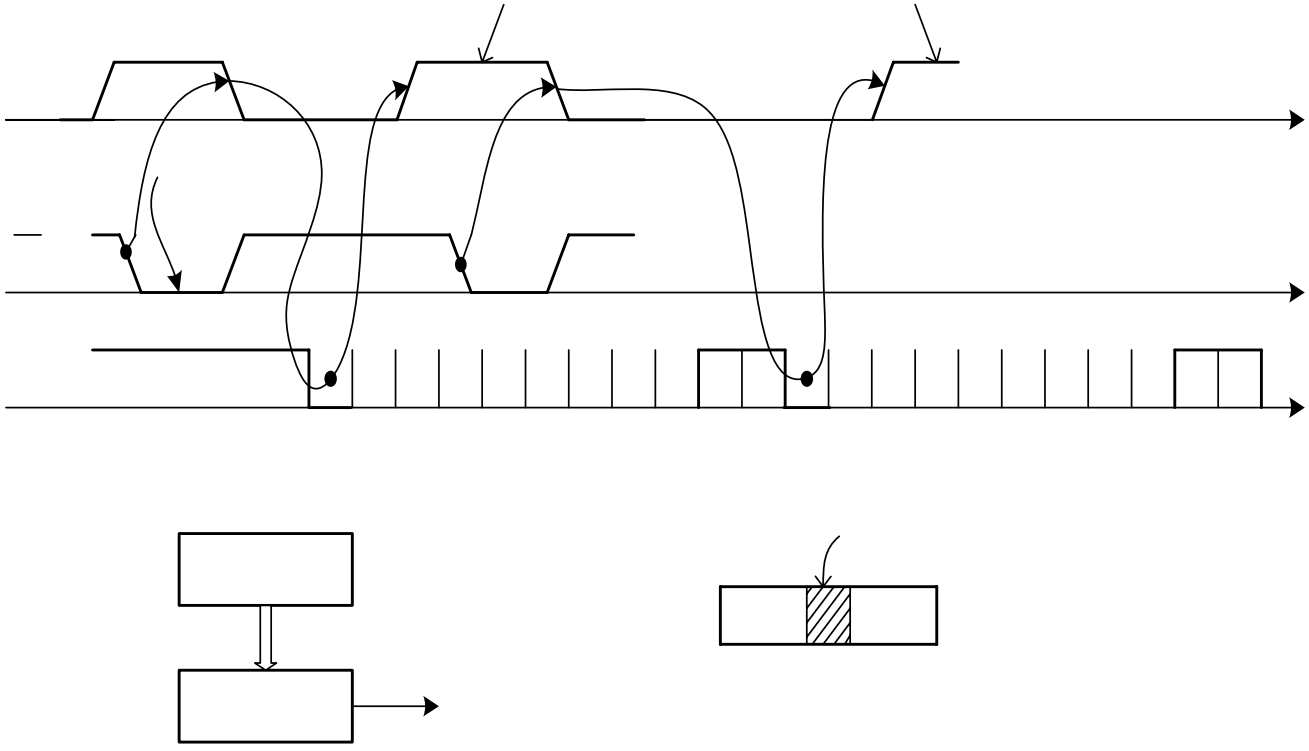


Рис. 3.17. Временная диаграмма «Вывод данных» в линию.

Флаг

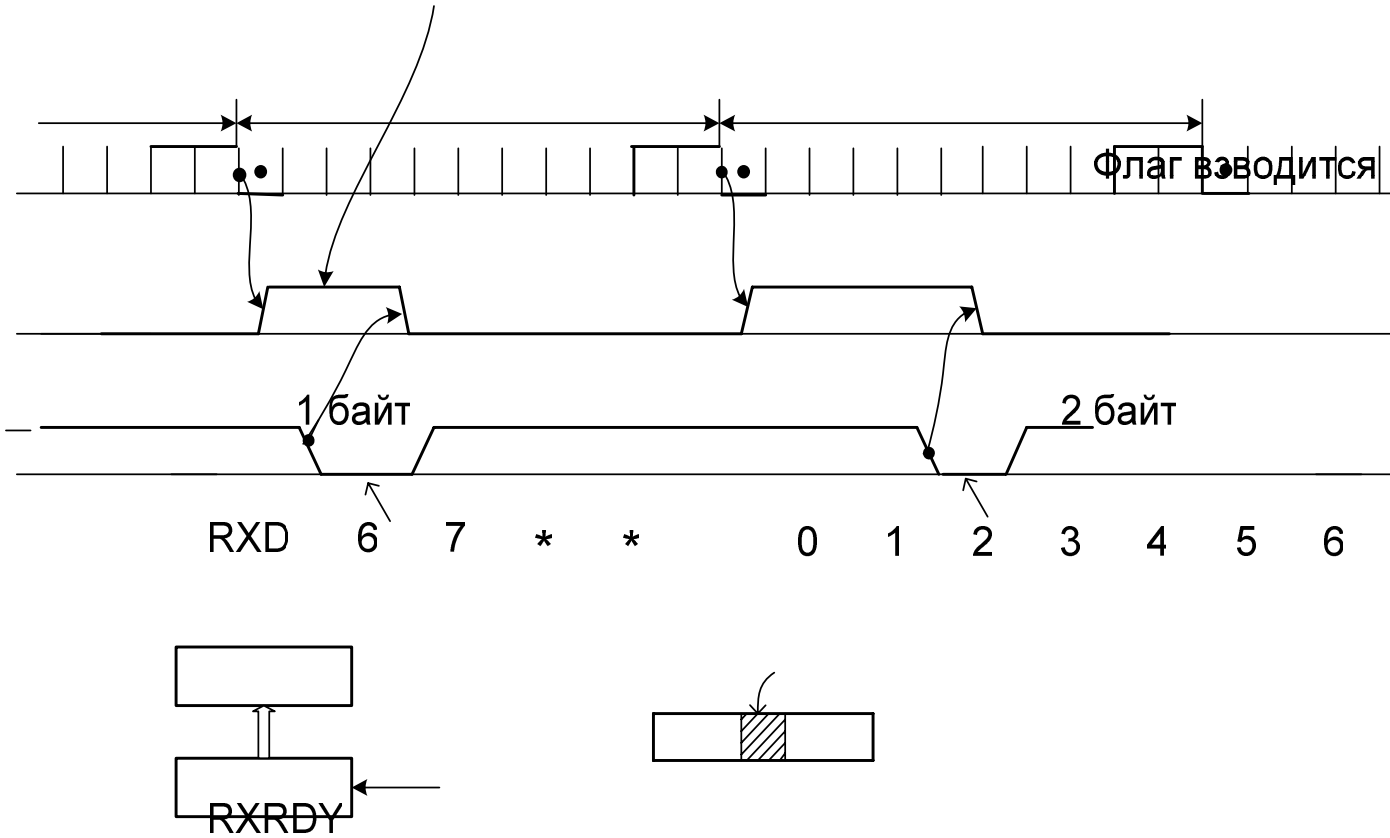


Рис. 3.18. Временная диаграмма «Ввод данных с линии»

В этом случае управление потоком осуществляется на основании состояния бита готовности ввода и бита готовности вывода (и связанных сними сигналов прерываний), находящихся в регистре состояния линии интерфейса RD.

Если к последовательному порту подключен модем, то при выводе данных в линию необходимо также учитывать готовность модема, определяемую сигналом $\overline{CTS}$.

Считывание 1-го байта
из БР Передатчика

3.7. Параллельный интерфейс «CENTRONICS» (ИРПР-М)

Интерфейс Centronics и соответственно параллельный порт персонального компьютера ориентированы на подключение принтера. Все сигналы интерфейса (табл. 3.1), можно разделить на четыре группы:

1. Однонаправленная восьмиразрядная шина данных для записи из компьютера (сигналы D0...D7); БР Передатчика
2. Четырехразрядная шина управления для записи из компьютера (сигналы -$\overline{STROBE}$, $\overline{AUTOFD}$, $\overline{INIT}$ и $\overline{SLCTIN}$);
3. Пятиразрядная шина состояния для чтения в компьютер (сигналы $\overline{ACK}$, BUSY, $\overline{PE}$, SLCT и -$\overline{ERROR}$); RXD
4. Шина "земли". Сдвиговый Рг.

Таблица 3.2. Линии интерфейса (с точки зрения печатающих устройств)

Наименование	Обозначение		Направление
	Русское	Международное	
Нуль	0В	0V	
Экран	Э	CG	
Питание	+5	+5V	
Линии управления и состояния			
Готовность приёмника	ГП	SLCT	П→И
Строб	$\overline{\text{СТР}}$	$\overline{\text{STROBE}}$	И→П
Подтверждение	$\overline{ПТВ}$	$\overline{\text{ACK}}$	П→И
Занят	ЗАН	BUSY	П→И
Сброс	$\overline{СБР}$	$\overline{\text{INIT}}$	И→П
Выбор	$\overline{\text{ВЫБОР}}$	$\overline{\text{SLCTIN}}$	И→П
Ошибка	$\overline{ОШ}$	$\overline{\text{ERROR}}$	П→И
Конец бумаги	$\overline{КБМ}$	$\overline{\text{PE}}$	П→И
Автоперевод строки	$\overline{АПС}$	$\overline{\text{AUTOFD}}$	И→П
Информационные линии			
Данные (1…8)	Д1÷Д8	D0÷D7	–

Обозначения: П-приемник, И-источник.

Все сигналы программно доступны, что позволяет реализовать произвольные протоколы информационного обмена в рамках имеющегося их набора и быстродействия компьютера.

Клавишей [ON LINE] для EPSON или [READY] для D100 принтер (PRN) можно привести в состояние «ON LINE» или в состояние «OFF LINE».

Если принтер отключён от линии, то передача данных на него прекращается.

Если принтер клавишей [ON LINE] или [READY] (для D100) подключён к линии (режим «ON LINE»), то в зависимости от сигнала $\overline{\text{SLCTIN}}$, поступающего на вход принтера могут быть следующие случаи:

1. $\overline{\text{SLCTIN}}$ ↓
   a. сигналы программного управления потоком **DC1/DC3** игнорируются
   b. печать возможна при соответствующем сигнале $\overline{\text{ERROR}}$ и BUSY.
2. $\overline{\text{SLCTIN}}$ ↑

   сигналы программного управления потоком **DC1/DC3** могут включать/выключать принтер в/из линии.

Если $\overline{\text{SLCTIN}}$ ↑, то командой **DC3** ЭВМ может запретить принтеру выполнять печать, а командой DC1 снова разрешить.

Команда **DC3** только запрещает печать, но не отключает PRN от линии. PRN в этом случае может принимать коды из ЭВМ для обнаружения кода DC1.

Такой способ управления потоком является программным. Говорят, что принтер работает в режиме **XON/XOFF**. Напомним, что существует аппаратный способ управления потоком **RTS/CTS**, с помощью соответствующих одноименных сигналов.

В общем виде, контакты ЭВМ и PRN для интерфейса CENTRONICS могут быть соединены так, как показано на рис. 3.19.

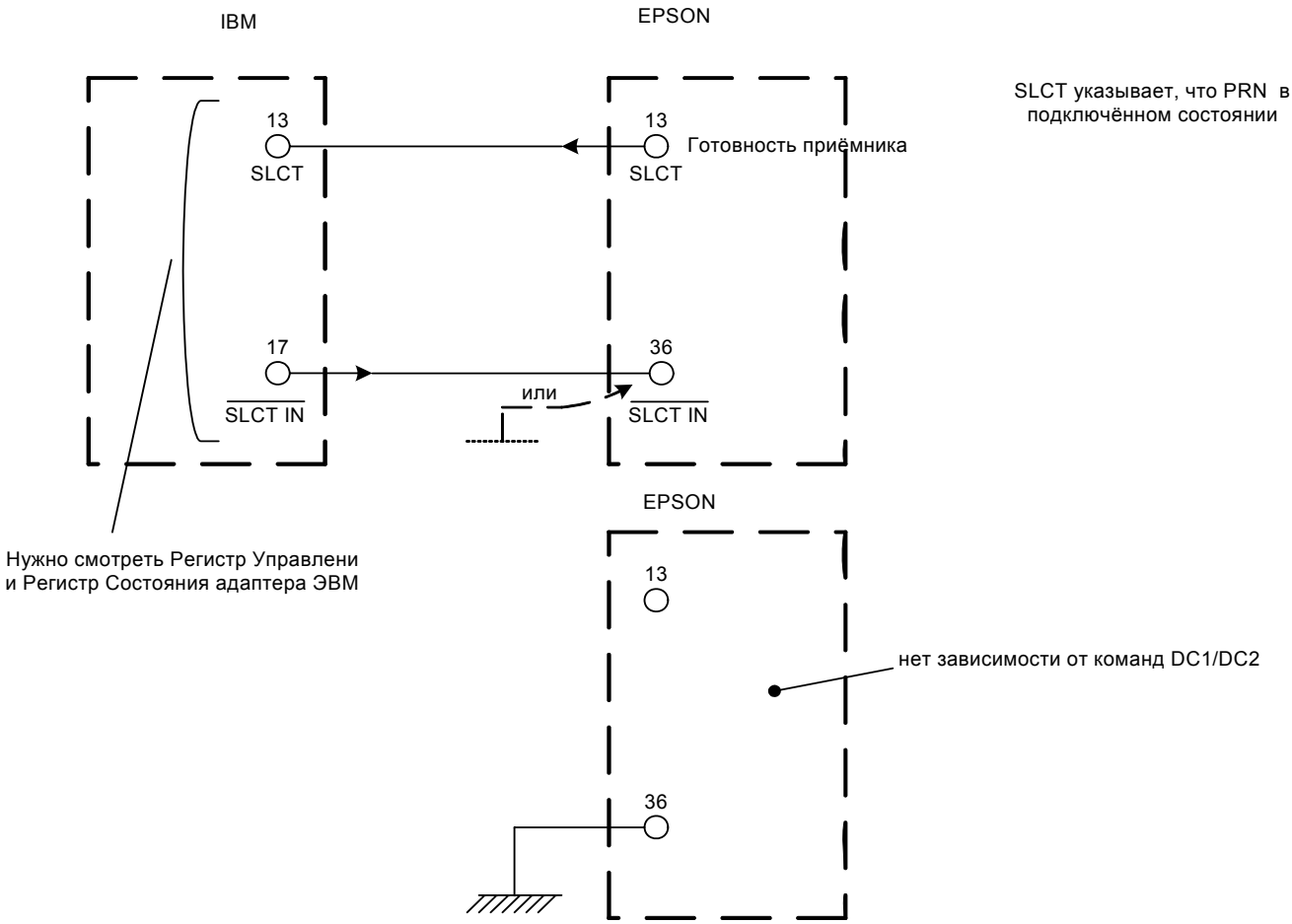


Рис. 3.19. Общий вид контактов ЭВМ и PRN для интерфейса CENTRONICS

Если PRN подключен к ЭВМ посредством интерфейса CENTRONICS, причем PRN находится в режиме «ON LINE», то передача байт происходит под управлением квитирующих сигналов $\overline{\text{STROBE}}$ и BUSY (или $\overline{\text{ACK}}$). Схема подключения изображена на рис. 3.20.

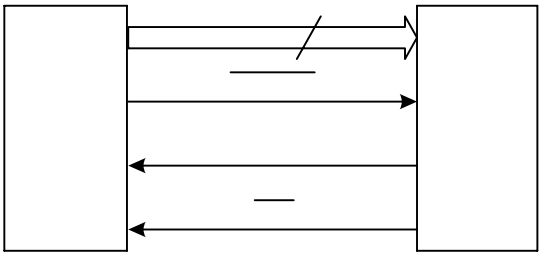


Рис. 3.20. Соединение по интерфейсу CENTRONICS

При выводе данных на печать **происходит передача данных из ОЗУ1 ЭВМ в ОЗУ2 принтера**. Этот путь можно разбить на три участка передачи данных

1. На первом участке (между ОЗУ1 и выходным буферным регистром) обменом «руководит» процессор ЭВМ (ПРЦ1).
2. На третьем участке (между входным буферным регистром и ОЗУ2) обменом «руководит» процессор принтера (ПРЦ2).
3. На втором участке (между выходным буферным регистром, “прописанным” в адресном пространстве ПРЦ1 и входным буферным регистром, “прописанным” в адресном пространстве ПРЦ2) обменом руководит интерфейс.

Блок-схемы программ работы ПРЦ1 и ПРЦ2 при условии, что

1. ПРЦ 1 «работает» в интерфейсе по готовности
2. ПРЦ 2 кроме ввода данных из входного буфера в ОЗУ2 печатающего устройства осуществляет анализ режима работы печатающего устройства «OFF LINE» или «ON LINE»,

приведены на рис. 3.21 и 3.22 соответственно.

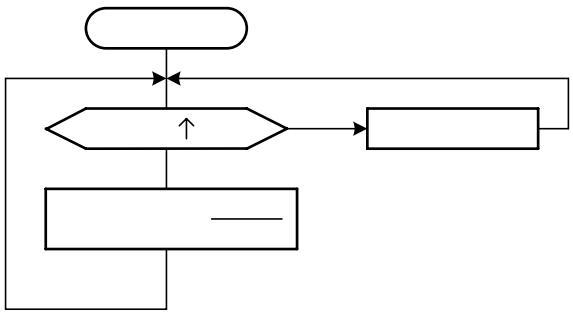


Рис. 3.21. Блок-схема программы для ПРЦ 1

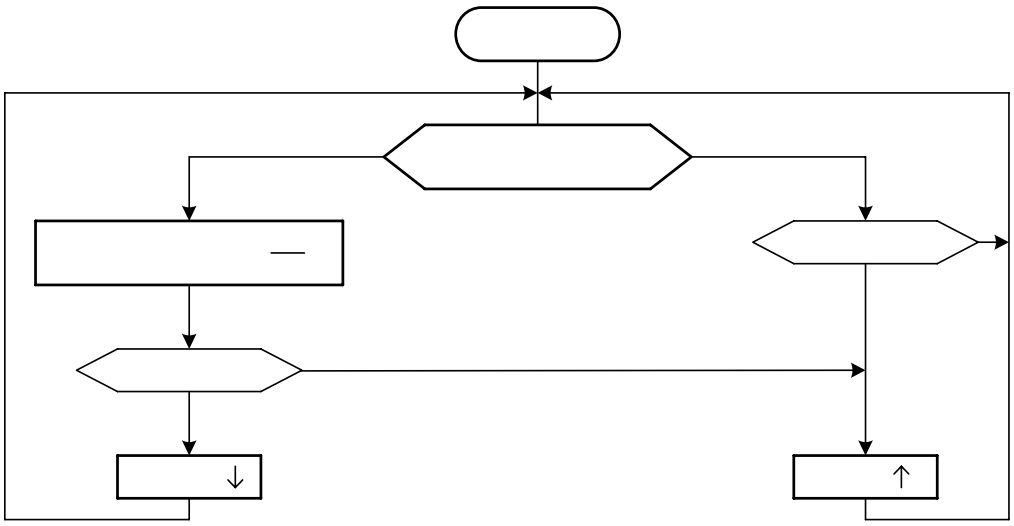


Рис. 3.22. Блок-схема для ПРЦ 2

Приведенные блок-схемы отражают **взаимодействие двух процессоров** при передаче данных из ОЗУ1 ПРЦ1 в ОЗУ2 ПРЦ2. Следует помнить, что управление переходом из состояния «ON LINE» в состояние «OFF LINE» осуществляет оператор.

Физический смысл сигналов квитирования отражает временная диаграмма взаимодействия ПРЦ1 и ПРЦ2 на рис. 3.23.

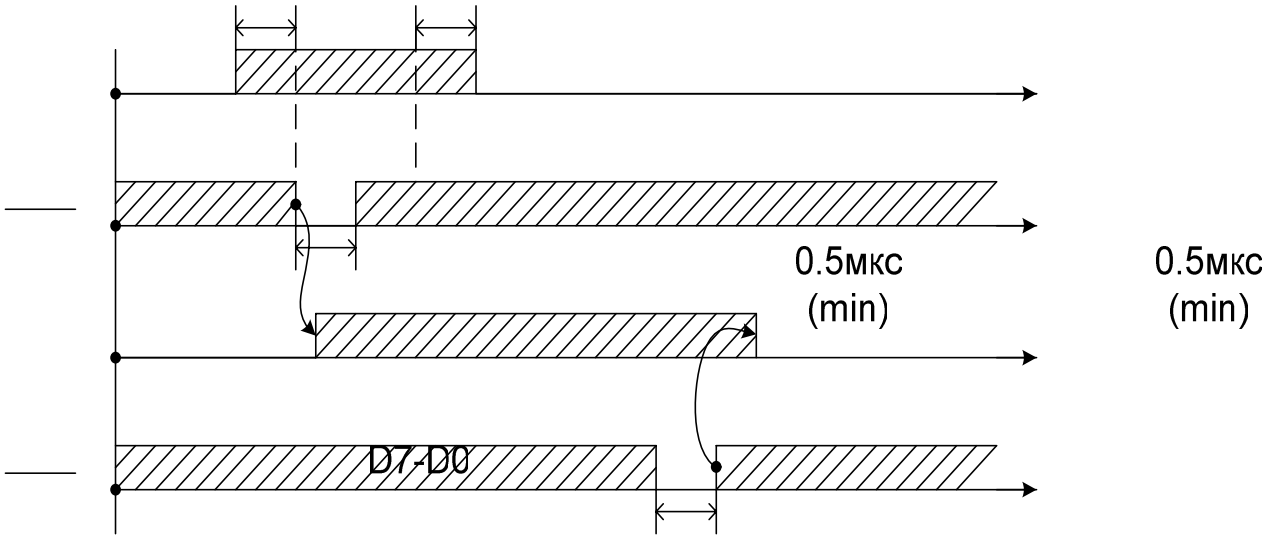


Рис. 3.23 Временная диаграмма взаимодействия ПРЦ1 и ПРЦ2

Форматы регистра управления и регистра состояния интерфейса CENTRONICS представлены на рис. 3.24 и 3.25 соответственно.

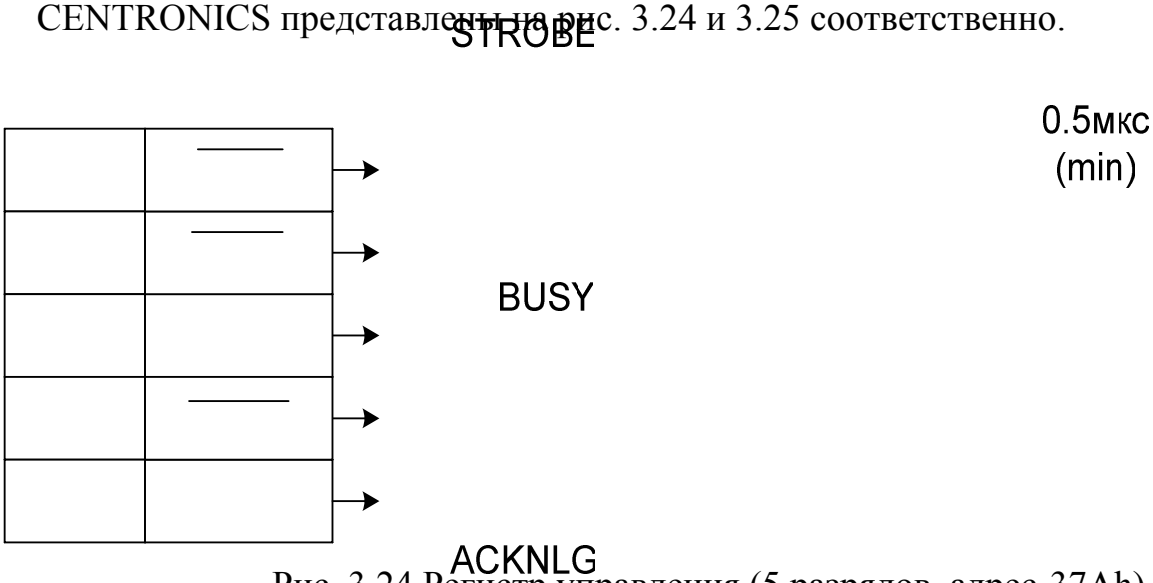


Рис. 3.24 Регистр управления (5 разрядов, адрес-37Ah)

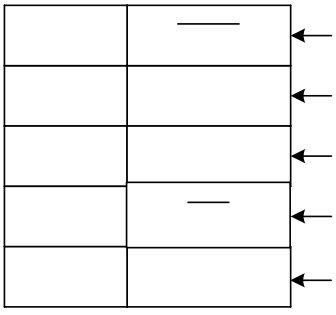


Рис. 3.25 Регистр состояния (5 разрядов, адрес-379h)

3.8. Параллельный интерфейс EPP

Стандарт на параллельный интерфейс IEEE 1284, принятый в 1994 году, определяет порты **SPP, EPP** и **ECP**.

- SPP - Standard Parallel Port
- EPP - Enhanced Parallel Port
- ECP - Extended Capability Port

Стандарт определяет 5 режимов обмена данными. Согласно IEEE 1284, возможны следующие режимы обмена данными через параллельный порт:

- **Режим совместимости (Compatibility Mode)** - однонаправленный (вывод) по протоколу Centronics. Этот режим соответствует стандартному порту SPP.
- **Полубайтный режим (Nibble Mode)** - ввод байта в два цикла (по 4 бита), используя для приема линии состояния. Этот режим обмена может использоваться на любых адаптерах.
- **Байтный режим (Byte Mode)** - ввод байта целиком, используя для приема линии данных. Этот режим работает только на портах, допускающих чтение выходных данных **(Bi-Directional** или **PS/2 Type 1).**
- **Режим EPP (Enhanced Parallel Port)** - двунаправленный обмен данными. Управляющие сигналы интерфейса генерируются аппаратно во время цикла обращения к порту. Эффективен при работе с устройствами внешней памяти и адаптерами локальных сетей.
- **Режим ECP (Extended Capability Port)** ***(*ECP Mode*)*** - двунаправленный обмен данными с возможностью аппаратного сжатия данных по методу **RLE** (Run Length Encoding) и использования FIFO-буферов и DMA. Управляющие сигналы интерфейса генерируются аппаратно. Эффективен для принтеров и сканеров.

В данном разделе рассмотрим EPP - усовершенствованный параллельный порт. К свойствам этого порта относится следующее:

- Передача данных со скоростью до 2 Мбайт/с;
- Работа в режиме 8- разрядного двунаправленного обмена (у стандартного параллельного порта для обмена имеется только 4 входных разряда);
- Адресация для поддержки периферийных устройств, подключенных шлейфом к одному параллельному порту ПК;
- Аппаратное формирование стробов, при этом для передачи каждого байта программному обеспечению ПК не требуется устанавливать или сбрасывать сигналы на линии стробов, а следовательно, ЦП ПК может использовать команду rep outst для повторного вывода строки байт из расположенных по порядку ячеек памяти в порт ввода-вывода.

Протокол **EPP** (Enhanced Parallel Port- расширенный параллельный порт) был разработан для обеспечения высокоскоростной двунаправленной связи через параллельный адаптер. Чтобы достигнуть этого, стандарт предусматривает для **EPP** два дополнительных порта ввода/вывода:

- порт данных

- порт адреса

3.8.1. Программные регистры EPP

Базовая система ввода/вывода (BIOS) может работать с тремя параллельными адаптерами. В процессе тестирования и инициализации системы BIOS находит работоспособные адаптеры и записывает их базовые адреса в таблицу, которая располагается в области данных BIOS по адресу **0000:0408h** и может содержать следующие значения:

378h - параллельный адаптер **LPT1;**

278h - параллельный адаптер **LPT2;**

3BCh - параллельный адаптер **LPT3.**

Эти адреса представляют собой базовые адреса портов, предназначенных для работы с соответствующими параллельными адаптерами. Для работы с каждым из них выделено по 3 порта: **БА** (базовый адрес), **БА+1**, **БА+2**. В режиме EPP имеются 2 дополнительных порта : **БА+3, БА+4**

- порт адреса
- порт данных

EPP-порт имеет расширенный набор регистров представленных в табл. 3.3.

Таблица 3.3. Регистры порта EPP

Имя регистра	Смеще-ние	Режим	R/W	Описание
SPP Data Port	+0	SPP/EPP	W	Регистр данных SPP
SPP Status Port	+1	SPP/EPP	R	Регистр состояния SPP
SPP Control Port	+2	SPP/EPP	W	Регистр управления SPP
EPP Address Port	+3	EPP	R/W	Регистр адреса EPP. Чтение или запись в него генерирует связанный цикл чтения или записи адреса EPP
EPP Data Port	+4	EPP	R/W	Регистр данных EPP. Чтение (запись) генерирует связанный цикл чтения (записи) данных EPP

Если программа пишет или читает из порта с адресом **БА+4** (порт данных), то происходит цикл записи или чтения данных.

Порт **БА+3**(порт адреса) выделен для передачи адреса, команд и управляющей информации, и при обращении к нему происходит цикл записи или чтения адреса. Для передачи байта в качестве адреса или данных необходимо записать его в соответствующий регистр.

Разработчик может интерпретировать информацию, полученную из портов адреса и данных любым образом, который имеет смысл для конкретного проекта, так как их разделение является условным (технически циклы передачи данных через порты **БА+3** и **БА+4** отличаются, но функционально эти порты никак не разделяются).

SPP Data Port (БА)

Этот порт, доступный как для записи, так и для чтения, предназначен для вывода данных. Программа может прочитать байт, только что записанный в порт **БА**. Биты этого порта называются **DATA**.

SPP Status Port(БА+1)

Порт состояния принтера, доступен только для чтения:

7	6	5	4	3	2	1	0
Wait	Interrupt	SPARE	SPARE	SPARE	0	0	0

SPARE- не используется.

Interrupt- принтер готов к печати.

Wait- низкий уровень разрешает начало цикла(установку строба в низкий уровень), переход в высокий - разрешает завершение цикла.

SPP Control Port(БА+2)

Порт применяется для управления принтером, подключенным к параллельному адаптеру. Он доступен для чтения и записи. Описание разрядов этого порта:

7	6	5	4	3	2	1	0
0	0	0	0	AddrStrobe	Reset	Data Strobe	$\overline{\text{WRITE}}$

Write- низкий уровень- цикл записи, высокий - цикл чтения
Data Strobe- строб данных. Низкий цикл устанавливается в циклах передачи данных. Когда компьютер посылает данные на принтер, он должен в течении 5мс активизировать (перевести в низкий уровень) данный сигнал, этим принтеру сообщается о готовности данных на соответствующих шинах.
AddrStrobe - строб адреса. Низкий уровень устанавливается в адресных циклах.

Каждому из битов портов, приведенных выше, соответствует одна из линий кабеля, соединяющего адаптер с устройством. Приведем табл. 3.4 соответствия контактов разъемов параллельного адаптера и принтера и их назначение.

Таблица 3.4. Соответствие разъемов параллельного адаптера и принтера

Контакты разъема адаптера	Контакты разъема принтера	Назначение
1	1	$\overline{\text{WRITE}}$
2	2	Данные, бит 0
3	3	Данные, бит 1

4	4	Данные, бит 2
5	5	Данные, бит 3
6	6	Данные, бит 4
7	7	Данные, бит 5
8	8	Данные, бит 6
9	9	Данные, бит 7
10	10	INTERRUPT
11	11	$\overline{\text{WAIT}}$
12	12	SPARE
13	13	SPARE
14	14	$\overline{\text{DataStrobe}}$
15	32	SPARE
16	31	RESET
17	36	$\overline{\text{AddressStrobe}}$
18-25	16,17,19-30,33	Земля

На рис. 3.26 представлена схема взаимодействия компьютера с периферийным устройством.

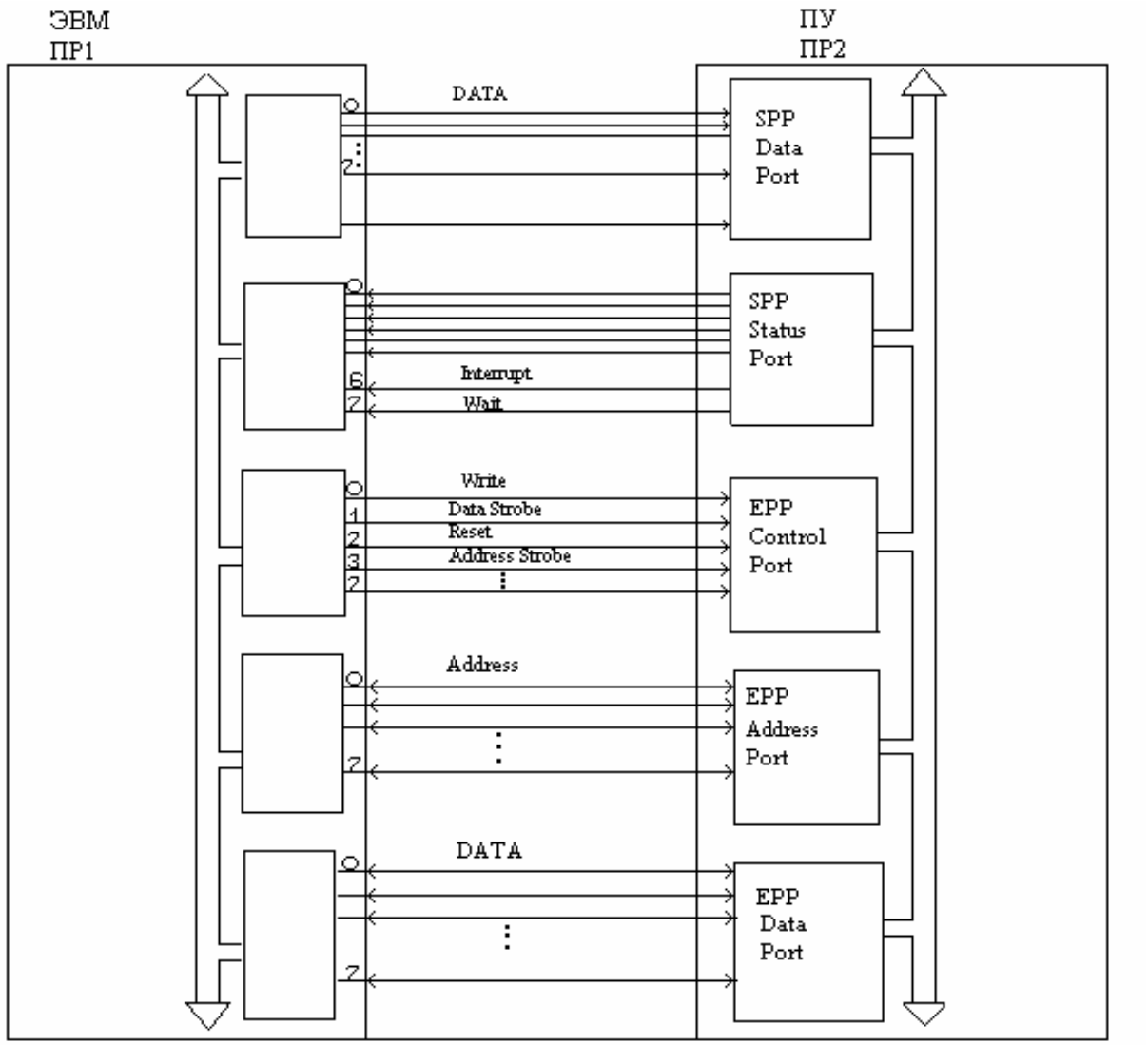


Рис. 3.26. Взаимодействие ЭВМ с ПУ

3.8.2. Протоколы обмена EPP

Чтобы обмен данными происходил корректно, мы должны чётко следовать протоколу передачи данных EPP. Так как "железо" выполняет всю работу, этот способ управления передачей требует использовать оборудование порта вместо программного обеспечения, как это происходит в случае SPP. Для выполнения цикла EPP программное обеспечение должно произвести одну операцию ввода-вывода по отношению к соответствующему регистру EPP.

Цикл записи данных представлен на рис. 3.27.

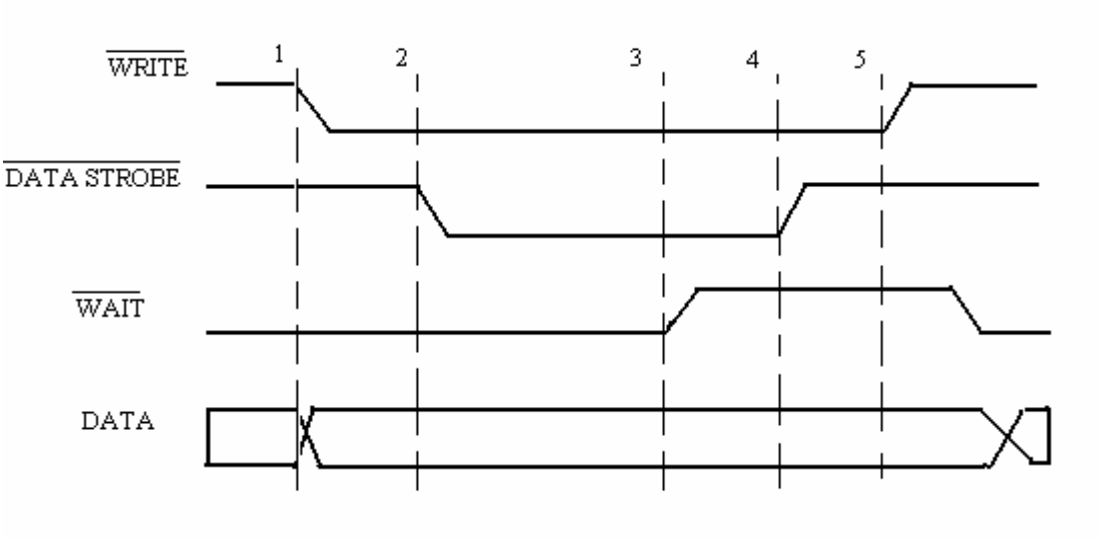


Рис. 3.27. Цикл записи данных EPP

Алгоритм программы записи данных в регистр данных EPP (базовый адрес + 4) имеет следующий вид:

1. Перевод $\overline{\text{WRITE}}$ в состояние низкого уровня означает начало операции записи, при этом данные помещаются на линии Data 0-7.

2. При низком уровне $\overline{\text{WAIT}}$ устанавливается строб данных, т.е. разрешается начало цикла передачи данных.

3. Компьютер ждёт подтверждения через переход $\overline{\text{WAIT}}$ в состояние высокого уровня, что означает завершение цикла передачи данных. Данные записываются из входного буфера в ОЗУ ПУ.

4. $\overline{\text{DataStrobe}}$ переходит на высокий уровень.

5. $\overline{\text{WRITE}}$ переводится в состояние высокого уровня, цикл записи завершается.

Цикл записи адреса EPP показан на рис. 3.28.

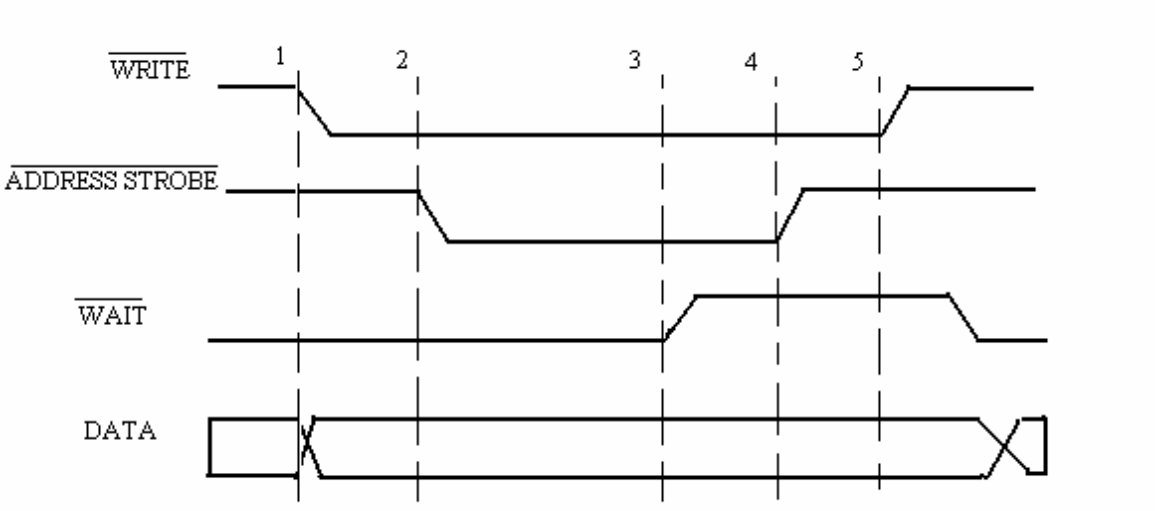


Рис. 3.28. Цикл записи адреса EPP

В соответствии с ним программа записи адреса в регистр адреса EPP (БА+ 3) может быть представлена в следующем виде:

1. Перевод $\overline{\text{WRITE}}$ в состояние низкого уровня означает начало операции записи. Адрес выводятся на линии Data 0-7.

2. Низкий уровень $\overline{\text{AddressStrobe}}$ при низком уровне состояния $\overline{\text{WAIT}}$ разрешает начало цикла.

3. Хост ждёт подтверждения через переход $\overline{\text{WAIT}}$ в состояние высокого уровня (завершение цикла).

4. $\overline{\text{AddressStrobe}}$ восстанавливается.

5. Цикл заканчивается.

Цикл чтения данных EPP представлен на рис. 3.29.

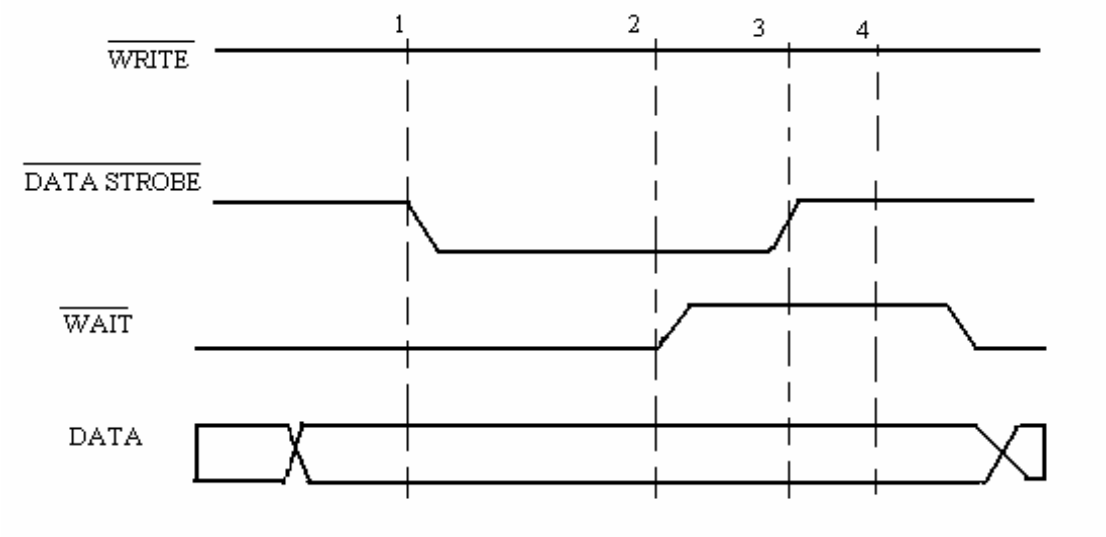


Рис. 3.29. Цикл чтения данных EPP

Ему соответствует следующая программа чтения регистра данных EPP (БА. + 4):

1. При низком уровне $\overline{\text{WAIT}}$ сигнал $\overline{\text{DataStrobe}}$ разрешает начало цикла.

2. Хост ждёт подтверждения через переход $\overline{\text{WAIT}}$ в состояние высокого уровня.

3. Данные считываются с контактов параллельного порта.

4. $\overline{\text{DataStrobe}}$ восстанавливается.

Цикл чтения адреса EPP представлен на рис. 3.30.

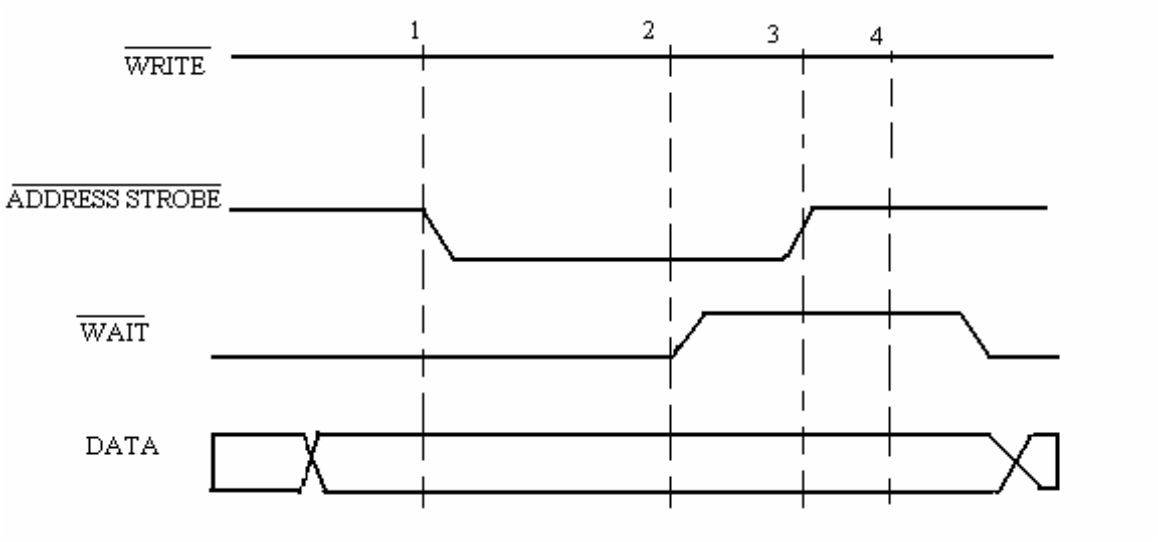


Рис. 3.30. Цикл чтения адреса EPP

Ему соответствует следующий алгоритм программы чтения регистра адреса EPP (Б. А. + 3):

1. При низком уровне $\overline{\text{WAIT}}$ сигнал $\overline{\text{AddressStrobe}}$ разрешает начало цикла.
2. Хост ждёт подтверждения через переход $\overline{\text{WAIT}}$ в состояние высокого уровня.
3. Данные считываются с контактов параллельного порта.
4. $\overline{\text{AddressStrobe}}$ восстанавливается.

4. ШИНА 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, а использоваться для подключения устройств в интегрированных системных платах.

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 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. 4.1.

Таблица 4.1

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
AGP1x/2x/ 4x/8x	266/533/ 1066/2132	-	+	+		32/64	66
PCMCIA	10/20	+	-	+		26	10
Card Bus	132	-	+	+		32	33

[1] Поддержка автоматического конфигурирования.

4.1. Организация шин PCI и PCI-X

PCI и PCI-X - синхронные параллельные шины расширения ввода-вывода, обеспечивающие надежный высокопроизводительный обмен и автоматическое конфигурирование устройств.

Шина PCI позволяет **объединять одноранговые 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 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 в компьютере выполнять 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 работы на эти усложнения приходится идти. В системе с несколькими шинами PCI возможно параллельное функционирование 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 приходится выполнять последовательно.

Каждая **физическая шина PCI** позволяет объединять лишь небольшое число устройств (обычно не более шести). 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 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 др.

Шина PCI/PCI-X имеет несколько вариантов конструктивного оформления, некоторые из них 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 конфигурирования устройств существует специальный набор функций PCI BIOS.

Блок-схема (рис. 4.1) показывает типичную системную организацию локальной шины PCI. В этом примере подсистема [**процессор** / **кэш 2-го уровня** / **память**] соединена с PCI через PCI-мост. Этот мост обеспечивает малое время задержки, за которое процессор может непосредственно обращаться к PCI устройствам, отображенным где-нибудь в адресных пространствах ввода-вывода или памяти. Он также обеспечивает широкую пропускную способность, позволяя управителям PCI напрямую обращаться в главную память.

Мост по необходимости исполняет такие функции, как буферизацию данных, регистрацию и главные функции PCI (например, арбитраж).

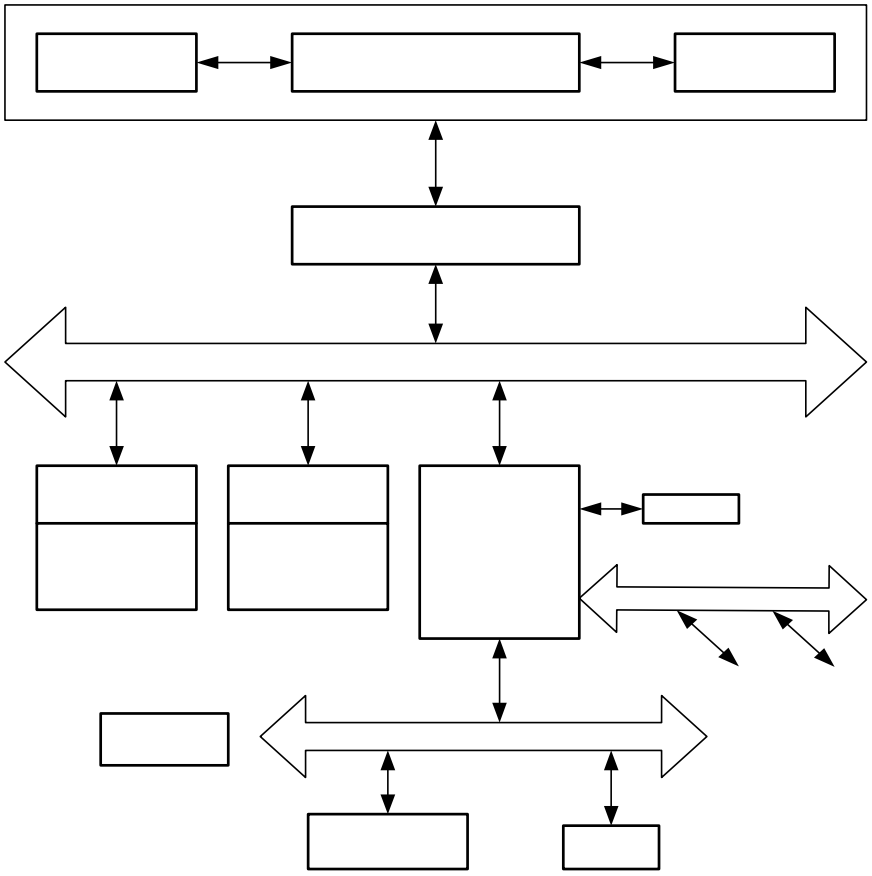


Рис. 4.1 Типовая конфигурация системы с шиной PCI

4.2. Взаимодействие устройств

Процессор Контро

С программной точки зрения устройство PCI может иметь следующие компоненты:

- **конфигурационные регистры**, используемые для идентификации начального конфигурирования устройства при инициализации системы (для всех устройств предусмотрен обязательный набор конфигурационных регистров, остальные регистры могут применяться для текущего управления);
- **операционные регистры** (необязательные), отображенные на пространство памяти или/и 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 (компонентам 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- 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} 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 задействована только 16 младших битов. В системе адресации ввода-вывода реализована 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, отображая операционные регистры устройств на пространство памяти (Memory-Mapped I/O).

Каждому устройству (точнее, каждой функции сложного устройства) выделяется 256-байтный блок конфигурационных регистров; в спецификации PCIX 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; кроме того, через них конфигурируются прерывания.

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.

Конфигурационные регистры устройств PCI расположены в обособленном пространстве адресов (отдельном от пространства адресов памяти и ввода-вывода, табл. 4.2).

Табл._4.2. Заголовок конфигурационного пространства одной функции устройства PCI

Байты конфигурационного пространства				
3	2	1	0	
Device ID		Vendor ID		00
Status Register		Command Register		01
Class Code		Revision ID		02
BIST	Header Type	Lat Timer	CL Size	03
Base Address 0				04
Base Address 1				05
Base Address 2				06
Base Address 3				07
Base Address 4				08
Base Address 5				09
Card Bus CIS Pointer				0A
Subsystem ID		Subsystem Vendor ID		0B
Expansion ROM Base Address				0C
Зарезервировано			Cap. Pointer	0D
Зарезервировано				0E
Max_Lat	Min_Gnt	Int_Pin	Int_Line	0F

Примечания:

- Device ID - идентификационный номер устройства, назначаемый

производителем.

- Vendor ID - идентификатор производителя устройства.
- Revision ID - версия продукта, назначенная производителем.
- Header Type - тип заголовка (биты [6:0]), определяющий формат ячеек в диапазоне 10-3Fh и несущий признак многофункционального устройства (если бит 7 установлен).
- Class Code - код класса, определяющий основную функцию устройства. Старший байт (адрес 0Bh) определяет базовый класс. Средний – подкласс, младший - программный интерфейс.
- BIST - **B**uilt-**I**n **S**elf **T**est - встроенный тест.

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

Регистр команд - Command (R/W) - служит для управления поведением устройства на шине PCI.

Регистр состояния - Status - служит для определения состояния и свойств устройства.

Регистр- Cache Line Size (CL 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#. 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).

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 IDnVendor ID). В поле 2Ch ставится идентификатор производителя карты PCI (может совпадать со значением в поле 0, если фирма выпускает и микросхемы, и карты).

Регистр Capabi1ity 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:

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

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 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 безусловно может подать любую команду любому устройству PCI. Только хост всегда 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**:

- **обычные маскируемые** - для сигнализации событий в устройстве; эти прерывания могут сигнализироваться как традиционным способом по специальным сигнальным линиям, так и передачей сообщений (MSI);
- **немаскируемые** - для сигнализации о серьезных ошибках;
- **прерывания системного управления** (System Management Interrupt, 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.

4.3. 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 **номеров шины** (bus), **устройства** (device) и **функции** (function). Идентификатор задает положение блока конфигурационных регистров заданной функции выбранного устройства в общем конфигурационном пространстве системы. Идентификаторы фигурируют при обращениях к регистрам конфигурационного пространства, а также при обмене сообщениями между устройствами.

Шина PCI представляет собой набор сигнальных линий, непосредственно соединяющих интерфейсные выводы группы устройств (слотов, микросхем на плате). В системе может присутствовать несколько

шин PCI, соединенных **мостами PCI** . 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 которых 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 из номеров функций, либо только на номер функции 0.

Нумерацией и конфигурированием всех устройств PCI занимается хост - «хозяин» шины PCI. Роль хоста, как правило, исполняет 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.

Конфигурирование выполняется для каждой **функции**; как уже отмечалось, полный идентификатор функции состоит из трех номеров: шины, устройства и 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, 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, но разными номерами шин). В устройствах PCI, интегрированных в системную плату, имеет место та же система нумерации. Их номера «запаяны намертво», в то время как номера устройств на картах расширения можно менять, 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 (такова, 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.

4.4. 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, у 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.

4.4.1. Сигнальный протокол шин PCI и PCI-X

Состояния всех сигнальных линий воспринимаются по положительному перепаду CLK, и именно эти моменты в дальнейшем описании подразумеваются под тактами шины (на рис. 4.3 отмечены вертикальными 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.

Знак - (минус) перед названием сигнала означает, что активный уровень этого сигнала логический ноль, обозначение {XX:0} означает группу сигналов с номерами от 0 до XX.

AD{31:0} - Address/Data - мультиплексированная шина адреса/данных. Адрес передается по сигналу -FRAME (в начале транзакции), в последующих тактах передаются данные.

-C/BE{3:0}# - Command/Byte Enable - команда/разрешение обращения к байтам. Команда, определяющая тип очередного цикла шины (чтение-запись памяти, ввода/вывода или чтение/запись конфигурации, подтверждение прерывания и другие) задается четырехбитным кодом в фазе адреса по сигналу - FRAME.

-FRAME# Кадр - индикатор фазы адреса (иначе - передача данных).

-DEVSEL# Device Select - выбор инициатором устройства назначения (ответ ЦУ на адресованную к нему транзакцию).

-IRDY# Initiator Ready - готовность инициатора к обмену данными.

-TRDY# Target Ready - готовность устройства назначения к обмену данными.

-STOP - запрос устройства назначения к инициатору на останов 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# {3:0} Request - запрос от PCI-устройства на захват шипы (для слотов 3:0).

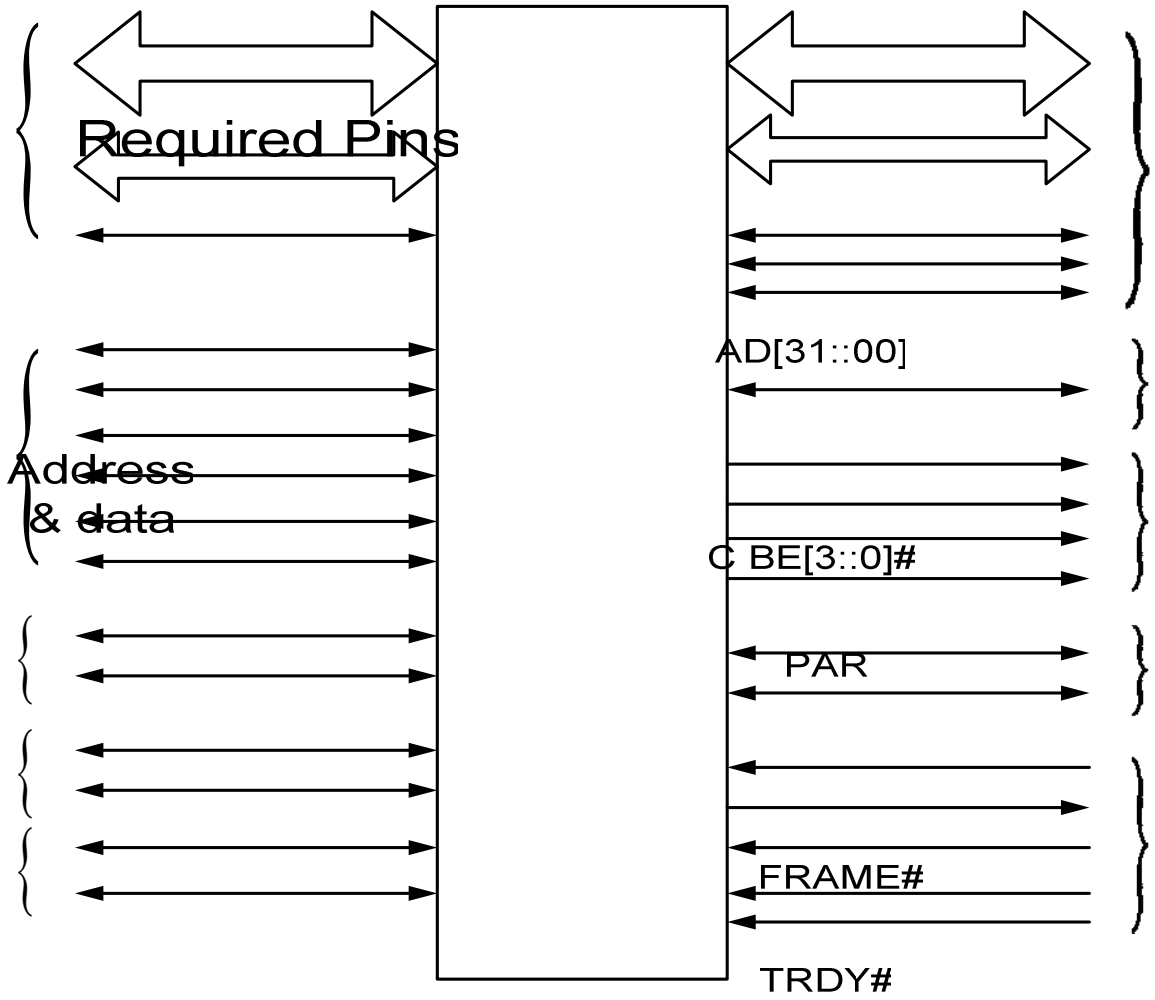


Рис. 4.2. Сигналы шины PCI

-GNT# Grant {3 0} - разрешение мастеру на использование шины.

PAR - общий бит четности для линий **AD{31:0}** и **C/BE{3:0}.**

-PER Parity Error - сигнал об ошибке по четности (от устройства, ее обнаружившего).

-RST Reset - 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, активизируется любым устройством PCI и вызывает немаскируемое прерывание процессора (NMI).

-REQ64 - Request 64 bit - запрос на 64-битный обмен. Сигнал вводится 64-битным устройством, по времени он совпадает с сигналом FRAME#. Во время окончания сброса (сигналом RST#) синхронизирует 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.

-ACK64# Подтверждение 64-битного обмена. Сигнал вводится 64-битным ПУ, опознавшим свой адрес, одновременно с DEVSEL#. Отсутствие этого подтверждения заставит инициатор выполнять обмен с 32-битной разрядностью

-INTR#A, B, C, D - линии запросов прерывания, направляются на доступные линии IRQ BIOS компьютера. Запрос по низкому уровню допускает разделяемое использование линий прерывания.

Clock - сигнал синхронизации на тактовой частоте шины.

Test Clock, -TSTRES, TestDO, TestDI - сигналы для тестирования адаптеров по интерфейсу JTAG (на системной плате обычно не задействованы).

TSTJMSLCT - перевод в режим тестирования.

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 заставить 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. 4.3 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.

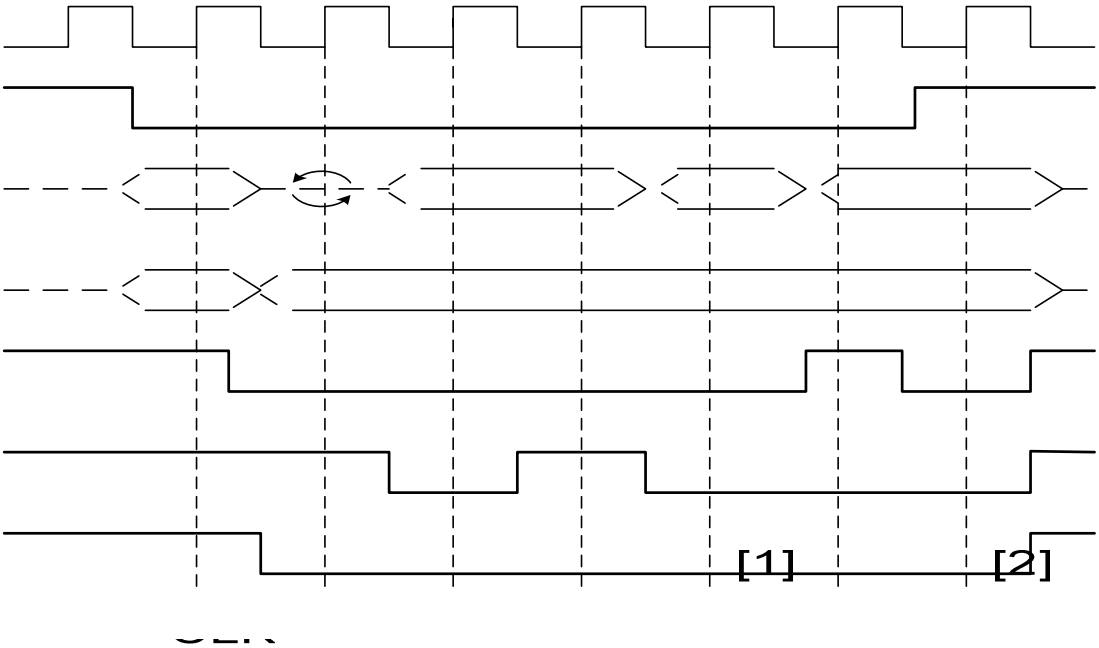


Рис. 4.3. Цикл передачи данных на PCI, включает 3 фазы передачи данных , с тактами ожидания. Данные передаются по переднему фронту сигнала CLK

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. Поддержка второго типа смежных транзакций необязательна. 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.

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 байты C/BE#). 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:

1. **comletion - нормальное завершение** по окончании обмена данными;
2. **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;
3. **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:

1. **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;
2. **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;
3. 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 устройство обнаруживает фатальную ошибку или иные условия, по которым оно уже никак не сможет обслужить данный запрос (в том числе и не поддерживаемую 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 транзакцию (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.

4.4.2 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, не пакетных) 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;

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 «за интересованные» 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. 4.3).

Таблица 4.3 Декодирование команд шин PCI и PCI-X

Код C/BE[3:0]	PCI Команда	PCI-X 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 сообщения 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.

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. 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.

4.5. 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#.

Традиционная схема формирования запросов прерываний с использованием пары PIC (Peripheral Interrupt Controller) изображена на рис.4.3.

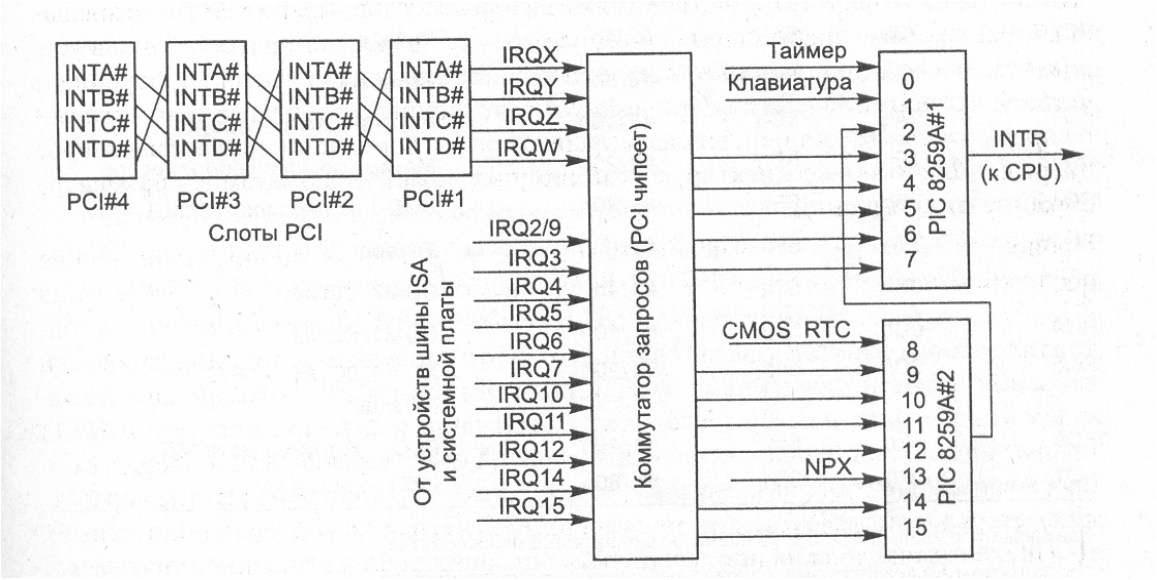


Рис. 4.3. Коммутация запросов прерываний

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/ ElSA. Эти линии обозначаются как IRQx и имеют общепринятое назначение (табл. 4.4).

Часть этих линий отдается в распоряжение шины PCI. В табл. 4.4 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 4.4. Аппаратные прерывания (в порядке убывания приоритета)

Имя (номер[1])	Вектор [2]	Вектор [3]	Контрол-лер/маска	Описание
NMI	02h			Контроль канала, четность памяти
				(в XT - сопроцессор)
IRQ0	08h	50h	#1/1 h	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	#2/1 h	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.

Устройство, подключаемое к **PCI** и представляющее одну функцию, должно использовать только линию **INTA#.**

Многофункциональные устройства могут использовать комбинации из четырёх линий, начиная с INTA#. Единственное ограничение состоит в том, что каждая из восьми функций (возможных в одном устройстве) может использовать только одну линию прерывания. Соответственно, устройство с внутренними восемью функциями может задействовать имеющиеся линии INTA#, INTB#, INTC#, INTD# следующим образом:

1. Все восемь функций подключены к INTA#.
2. Семь подключены к INTA#, одна к INTB#.
3. Две подключены к INTA#, две к INTB#, две к INTC# и две к INTD#.
4. Четыре подключены к INTA#, четыре к INTB# и т.п.

Спецификация PCI относительно безразлична к приоритетам прерываний. Приоритет в данном случае зависит от внешнего контроллера, который переадресует запрос на прерывание PCI устройства в соответствующую линию системных прерываний.
Например, на персональных компьютерах **редиректор** (коммутатор запросов) может преобразовать запрос функциональной единицы PCI по линиям INTA#-INTD# в запрос по одной из линий IRQ0-IRQ15.

Чтобы можно было это сделать, любая функция внутри PCI устройства, которая генерирует прерывание, должна задействовать два конфигурационных регистра (в своём конфигурационном пространстве):

1. **Регистр вывода прерывания** (Interrupt pin register). Является регистром «только для чтения», который идентифицирует линию прерывания PCI (**INTA#-INTD#**), используемую данной функцией данного PCI устройства.
2. **Регистр линии прерывания** (Interrupt line register). Является регистром «только для записи», который указывает приоритет и вектор (номер системного прерывания), которые редиректор прерываний должен присвоить данной функции. В персональных компьютерах Intel x86 значения 0x00-0x0F соответствует **IRQ0-IRQ15**.

Такая схема является достаточно гибкой, поскольку не навязывает никаких ограничений, обусловленных спецификой механизма прерываний в системе, конструктору устройств или системы, что делает возможным использование этой архитектуры в процессорных средах, отличных от Intel x86.

4.6 Возможности DMA

Спецификация PCI не включает понятия **slave DMA**. Вместо этого, собственные функции PCI (функциональные единицы) являются либо «хозяевами шины» (bus masters), выполняющими свой собственный DMA перенос данных, либо они используют программируемый ввод/вывод.

Единственным типом устройств, которые могут использовать DMA перенос данных с использованием системных контроллеров (slave DMA), являются не-PC1 платы, подключенные через (E)ISA мост.

В DMA операциях собственно PCI шины, участники называются агентами (agents), и в каждой транзакции всегда участвуют два из них, а именно:

1. **Инициатор (Initiator)**. Это «хозяин шины» (bus master), который выиграл право доступа к шине и хочет определить операцию переноса.
2. **Цель (Target)**. Это PCI функция (функциональная единица PCI устройства), адресуемая в настоящий момент инициатором с целью выполнения передачи данных.

Поскольку любой «хозяин шины» PCI может быть инициатором, то возможна передача данных напрямую между двумя PCI-устройствами без промежуточной остановки данных в оперативной памяти. Эта возможность просто предназначена для использования в высокоскоростных сетевых и видео-приложениях.

Необходимо также упомянуть, что спецификация PCI не определяет стратегию, которая должна быть использована при арбитрировании доступа к шине. Определены только временные диаграммы арбитрирующих сигналов в шине. Метод по нахождению того, «кто будет следующим» в использовании шины, является определяемым конкретной системой, где реализованы шины PCI.

4.7. 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 «заинтересованная» программа может найти требуемое устройство и выбрать подходящий вариант драйвера.

4.8. 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 lAh**; 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 lAh) 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, 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-разрядном режиме указывает на сегмент с физическим адресом F0000. 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. 4.5). При возврате в структуре

параметров буфера возвращается его реальная длина; если при вызове указан буфер, не вмещающий весь результат, устанавливается код ошибки 89h.

Таблица 4.5. Описание опций прерывания для одного устройства PCI

Смещени	Размер	Назначение
0	byte	PCI Bus number - номер шины PCI
1	byte	PCI Device number - номер устройства PCI (в старших пяти битах)
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). Для устройств на материнской плате номер слота = 0, для дополнительных устройств номер слота равен значению, которое соответствует физическому расположению слота на материнской плате.
15	byte	Резерв

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 регистрах всех затрагиваемых устройств и их функций.

5. ИНТЕРФЕЙС USB

5.1. Общие сведения

Возникновение интерфейса USB в первую очередь связано с потребностью людей в едином и универсальном интерфейсе. В начале 1996 года была опубликована первая версия этого стандарта USB 1.0. В группу разработчиков вошли такие известные компании, как IBM, Intel, Microsoft, DEC, Compaq и многие другие. Создавая универсальный интерфейс, они больше ориентировались на то, чтобы с его помощью осуществить взаимодействие компьютерных и телефонных систем. Но очень скоро поняли, что USB может удовлетворить потребности многих приложений и все сферы

компьютерной телефонии. И уже осенью 1998 года вышла спецификация USB 1.1, исправляющая проблемы, обнаруженные в первой редакции. Спецификации 1.0 и 1.1 обеспечивали работу на скоростях 12 Мбит/с и 1.5 Мбит/с. Вместе с тем, к этому времени были достигнуты следующие скорости обмена: с флэш-памятью - 4 Мбайт/с, с дисками Iomega Zip -750 Мбайт/с, с дисками CD ROM - 7 Мбайт/с. В связи с этим в 2000 году вышла новая версия USB 2.0, которая обеспечивала работу на скоростях 1.5; 12 и 480 Мбит/с и была полностью совместима с ранними версиями.

Спецификация USB определяет следующие функциональные возможности интерфейса:

1. Простота использования для конечного пользователя:
   - скрытие подробностей электрического подключения от конечного пользователя;
   - самоидентификация устройства и автоматическое конфигурирование.
2. Широкие возможности работы:
   - поддержка одновременных операций со многими устройствами;
   - поддержка до 127 устройств на шине;
   - передача разнообразных потоков данных и сообщений;
   - поддержка одновременно как изохронной, так и асинхронной передачи данных.
3. Равномерная пропускная способность:
   - гарантированная пропускная способность и низкие задержки голосовых и аудиоданных;
   - возможность использования всей полосы пропускания.
4. Гибкость:
   - поддержка разных размеров пакетов, которые позволяют настраивать функции буферизации устройств;
   - управление потоком данных на уровне протокола.
5. Надежность:
   - контроль ошибок и восстановление на уровне протокола;
   - поддержка идентификации неисправных устройств.
6. Объединение с архитектурой Plug and Play.
7. Возможность простого обновления.

В табл. 5.1 приведено сравнение интерфейса USB с другими интерфейсами персонального компьютера.

Таблица 5.1. Сравнение USB с другими интерфейсами.

Интерфейс	Число устройств / Число проводов / Длина провода (м)	Скорость	Использование
USB 2.0	127/3/10	1.5 Мбит/с 12 Мбит/с 480 Мбит/с	Любые устройства с USB 1.x/2.0

RS-232	1/6/50-100	115.2 Кбит/с	Модем, мышь, ключи защиты
RS-485	32/2/4000	10 Мбит/с	Промышленные устройства (COM-порт через преобразователь)
FireWire	64/3/15	400 Мбит/с	Видеоданные, дисковые массивы
Ethernet	1024/3/1600	10 Мбит/с 100 Мбит/с 1 Гбит/с	Сетевые соединения (сетевая карта)
MIDI	1/3/50	31.5 Кбит/с	Музыкальные устройства
LPT	1/9/10-30	0.8-16 Мбит/с	Принтеры, сканеры, дисковые устройства.

Из таблицы видно, что достаточной альтернативы USB не существует. Интерфейсы сравнимые с USB по скорости обмена требуют специальных преобразователей. Интерфейсы, не требующие дополнительных элементов, либо низкоскоростные, либо узконаправленные. Кроме того, к несомненным плюсам USB относится организация помехозащищенности на уровне аппаратного и шинного протоколов, а также отсутствие дополнительных элементов для подключения устройств (например, терминаторы для SCSI-интерфейса). Но единственным минусом можно считать довольно короткое кабельное соединение.

5.2. Основные понятия USB

Для того, чтобы осуществить подключение USB-устройств к ПК, в последнем должен быть установлен USB-интерфейс. Материнские платы, которые собраны на базе ATX форм-фактора уже содержат USB хост-контроллеры, и в компьютеры встроены USB-порты. Однако материнские платы, собранные до 2000 года, по технологии AT такого интерфейса не содержат.

Для корректной работы USB- интерфейсу должны быть выделены следующие системные ресурсы:

- номер линии прерывания;
- номер канала прямого доступа к памяти (номер DMA);
- диапазон адресов ввода/вывода.
- Номер прерывания для USB выделяется из числа незанятых другими устройствами прерываний и из числа вообще возможных. Например, очень часто USB- контроллер использует линии прерываний IRQ5 и IRQ9. Но для устройств, подключенных к шине USB, настоящего механизма прерываний (как, например, для последовательного порта) не существует. Вместо этого хост-контроллер опрашивает подключенные устройства на предмет наличия данных о прерывании. Опрос происходит в фиксированные интервалы времени, обычно каждые 1-32мс. Таким образом, возможности

работы с прерываниями фактически определяются и реализуются хост-контроллером через выделенные ему линии прерываний.

Устройства USB-шины также не имеют прямого доступа к системной памяти, то есть они полностью изолированы от системных ресурсов USB-интерфейсом хост-контроллера. Тем не менее, USB-интерфейс обеспечивает «иллюзию» поддержки DMA для логических каналов, обеспечивающих доступ к конечным точкам внутри подключенных к шине устройств. По мере получения данных от подключенных USB-устройств, интерфейс хост-контроллера использует DMA доступ для того, чтобы поместить полученные от устройств данные в системную память. Номер канала DMA также назначается из числа свободных, чаще это может быть 5, 6 или 7 каналы DMA. Диапазон адресов ввода-вывода назначаются автоматически для каждого хост-контроллера.

Спецификация USB регламентирует наличие следующих компонентов:

- **Хост-контроллер** - главный контроллер, который входит в состав системного блока компьютера и управляет работой всех устройств на шине USB.
- **Хаб**- устройство, которое обеспечивает дополнительные порты на шине USB.
- **Корневой хаб** - хаб, входящий в состав хост-контроллера.
- **Порт**-точка подключения.
- **Функция** - периферийное устройство или отдельный блок периферийного устройства, способный передавать и принимать информацию по шине USB.
- **Логическое устройство** USB - набор конечных точек.

Хост-контроллер реализует следующие функции:

1. обнаруживает подключение и отключение устройств USB;
2. управляет потоками данных;
3. осуществляет сбор статистики;
4. управляет энергопотребление подключенных к шине ПУ.

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

1. **Адресация**-устройство должно отзываться на назначенный ему уникальный адрес и только на него.
2. **Конфигурирование** - после включения или сброса устройство должно предоставить нулевой адрес для возможности конфигурирования его портов.
3. **Передача данных** - устройство имеет набор конечных точек для обмена данными с хостом. Для конечных точек, допускающих разные типы передач, после конфигурирования доступен только один из них.
4. **Управление энергопотреблением** - любое устройство при подключении не должно потреблять от шины ток, превышающий 100мА, при конфигурировании устройство заявляет свои потребности тока, но не более 500мА.

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

Конечная точка имеет следующие основные параметры:

- частота доступа к шине;
- допустимая величина задержки обслуживания;
- требуемая ширина полосы пропускания;
- номер конечной точки;
- максимальный размер пакета, который конечная точка может принимать или отправлять;
- тип передачи;
- направление передачи (IN-к хосту, OUT- от хоста).

Физическая архитектура USB-шины представлена на рис. 5.1

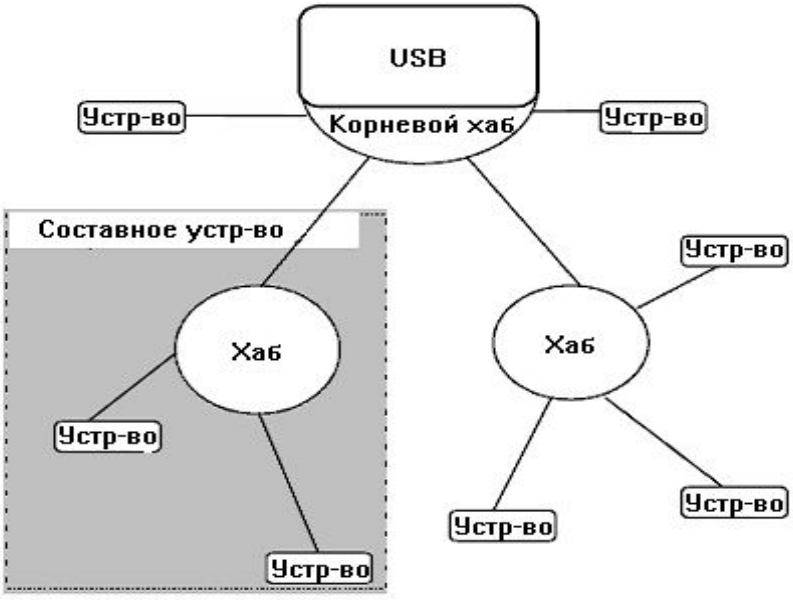


Рис. 5.1. Физическая архитектура USB

Логическая архитектура выглядит как обычная звезда, центром которой является прикладное ПО, а вершинами - набор конечных точек. Прикладная программа ведет обмен информацией с каждой конечной точкой посредствам организации канала.

Логическая архитектура шины представлена на рис. 5.2

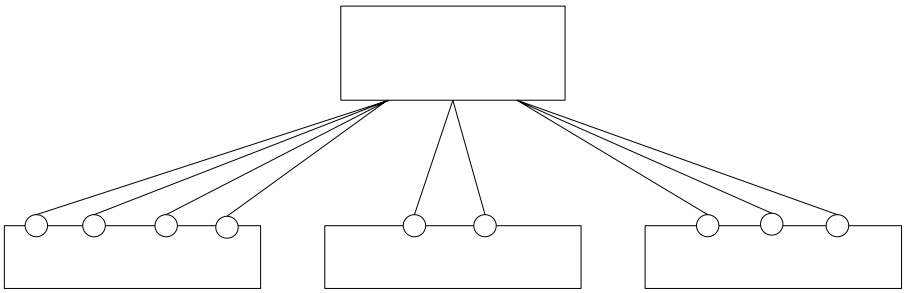


Рис. 5.2. Логическая архитектура USB.

5.3 Физический интерфейс

Стандартный съемный кабель служит для соединения хоста или хаба с устройством. С одной стороны, он заканчивается разъёмом типа «А» для подключения к хосту или хабу, а, с другой стороны, - разъёмом типа «В» для подключения к устройству. В таблице 2 приведена цветовая гамма проводников разъёма «А» и «В» и их описание.

Таблица 5.2. Цветовая гамма проводников разъёма «А» и «В»

Номер контакта	Цвет	Описание
1	Красный	+5 В, питание
2	Белый	D-, данные «минус»
3	Зеленый	D+, данные «плюс»
4	Черный	GND, земля
Корпус	Медная оплётка	Экран

Для передачи сигналов используются два провода **D+** и **D-**. На каждой стороне интерфейса (порте хаба и подключенного устройства, рис. 5.3) 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($15_K O_M$) 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 Rzl 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.

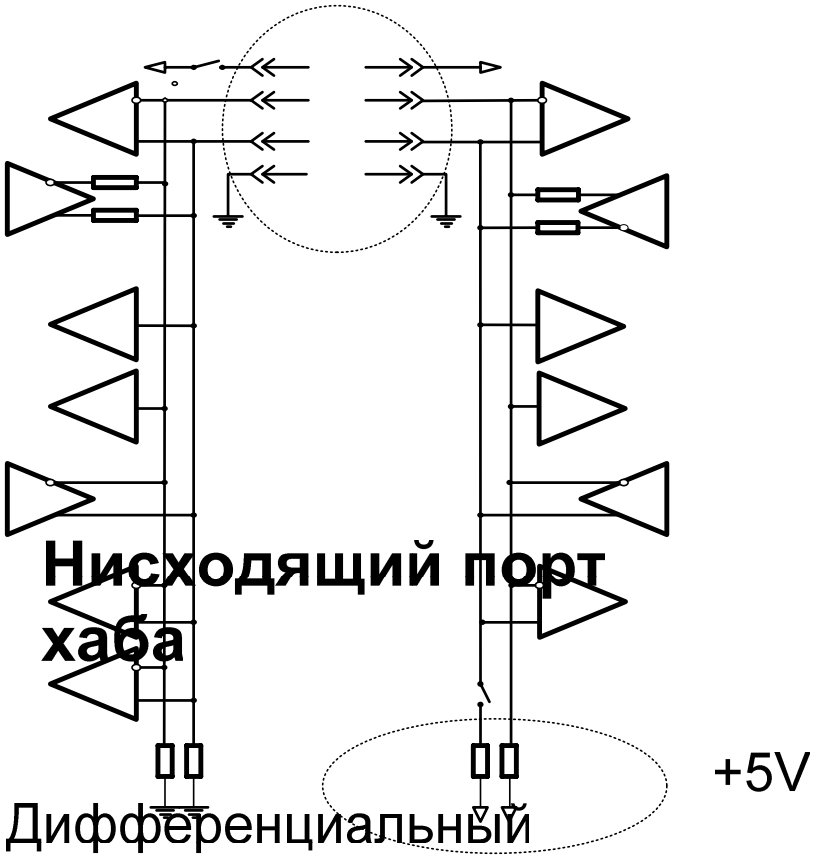


Рис. 5.2. Физический интерфейс USB

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;
- **SEO** (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. 5.4): 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 синхропоследовательность 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.

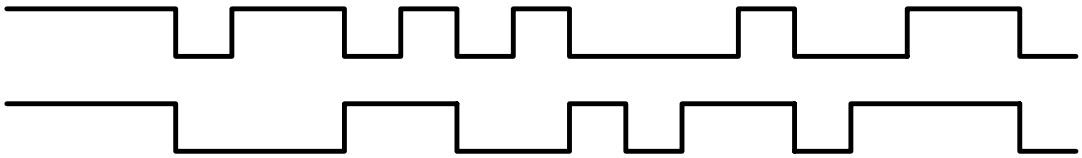


Рис. 5.4. Кодирование данных по методу NRZI

Идентификация устройства

USB- устройство должно указать свою скорость путем подтяжки линии **D+** или **D-** по средствам сопротивления 1.5 кОм к напряжению 3.3 В. Это подтягивающее сопротивление хост и хаб используют для обнаружения подключения устройств к своему порту. То есть, без этого резистора ни хост, ни хаб не поймут, что к шине USB подключено новое устройство. Лишь когда напряжение, снимаемое с регистра R2 составит 3.3В, хост считает, что на шине появилось новое устройство, и теперь хост может инициировать сброс устройства и запрос дескрипторов, которые описывают его функциональные особенности и возможности.

5.4. Взаимодействие ПУ и ПК посредством шины USB

Ранее было оговорено, что для реализации подключения USB-устройств необходимо, чтобы в системе присутствовал USB-интерфейс, под которым подразумевалось наличие в системе хост-контроллера USB (он может быть не один, в зависимости от количества шин USB) и корневого хаба USB. В данном же пункте будут рассмотрены программные компоненты, которые также необходимы для осуществления взаимодействия ПУ и ПК посредствам шины USB. На рис. 5.5 представлена структурная схема взаимодействия компонентов USB.

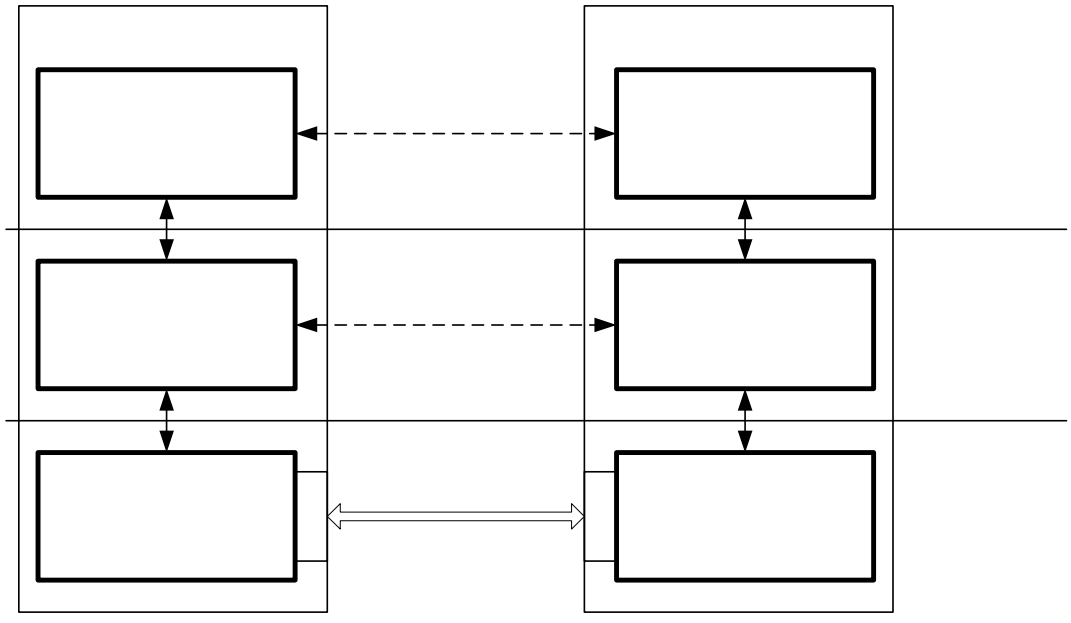


Рис. 5.5. Взаимодействие компонентов USB

Рассмотрим каждый из компонентов:

1. **Клиентское программное обеспечение** (**CSw**-Client Software) - драйверы устройств USB (**USB device driver**), обеспечивающие доступ к устройствам со стороны прикладного ПО. Эти драйверы взаимодействуют с устройствами только через программный интерфейс с общим драйвером

шины USB (**USBD**). Непосредственного обращения к каким-либо регистрам аппаратных средств драйверы устройств USB не выполняют. CSw отвечает за формирование **пакета запроса на ввод/вывод** (**IRP**, Input/output Request Packet), который содержит сведения об адресе и длине буфера в оперативной памяти, куда или откуда эти данные будут помещаться/извлекаться.

2. Компонент **системное обеспечение USB** состоит из драйвера шины USB (**USBD**, Universal Serial Bus Driver) и драйвера хост-контроллера (**HCD**, Host Controller Driver).

USBD «заведует» всеми USB-устройствами системы, их нумерацией, конфигурированием, предоставлением служб, распределением пропускной способности шины, мощности питания и т. д.;

Когда клиентский компонент передаёт IRP запрос компоненте системного обеспечения, то USBD преобразует его в одно или несколько транзакций шины и затем передаёт получившийся перечень транзакций HCD. **HCD** выполняет с этим перечнем транзакций следующее:

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

Также системный драйвер (USBD+HCD) отвечает за выполнение следующих действий:

- распределение полосы пропускания шины USB,
- назначение логических адресов каждому физическому USB-устройству.

3. Компонент **хост-контроллер интерфейса шины USB** полученные отдельные транзакции преобразует в соответствующую последовательность операций шины. В результате формирует кадры. Кадры передаются последовательной передачей бит по методу NRZI.

4. Компонент **интерфейса шины периферийного устройства** взаимодействует с интерфейсным компонентом шины USB на стороне хоста и передаёт кадры данных от хоста периферийному устройству в формате, определяемом спецификацией USB. Затем он передаёт эти кадры вверх - компоненту логического USB- устройства.

5. Компонент **логическое USB- устройство** 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.

6. Компонент **функциональное устройство** может выполнять несколько 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.

На рис. 5.6 представлена более наглядная схема взаимодействия компонентов при передаче данных.

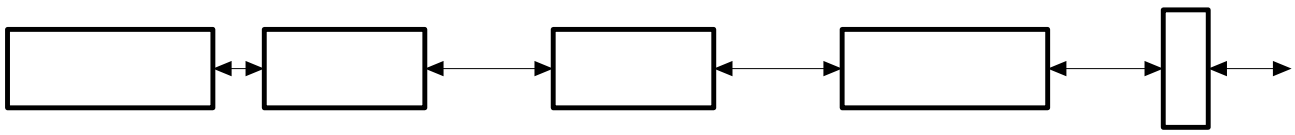


Рис. 5.6. Схема передачи данных между компонентами

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.

5.5 Модель передачи данных

Каждая единица клиентского ПО (обычно представляемая драйвером) связывается с одним интерфейсом своего устройства (функции) монопольно и независимо (рис. 5.7). 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.

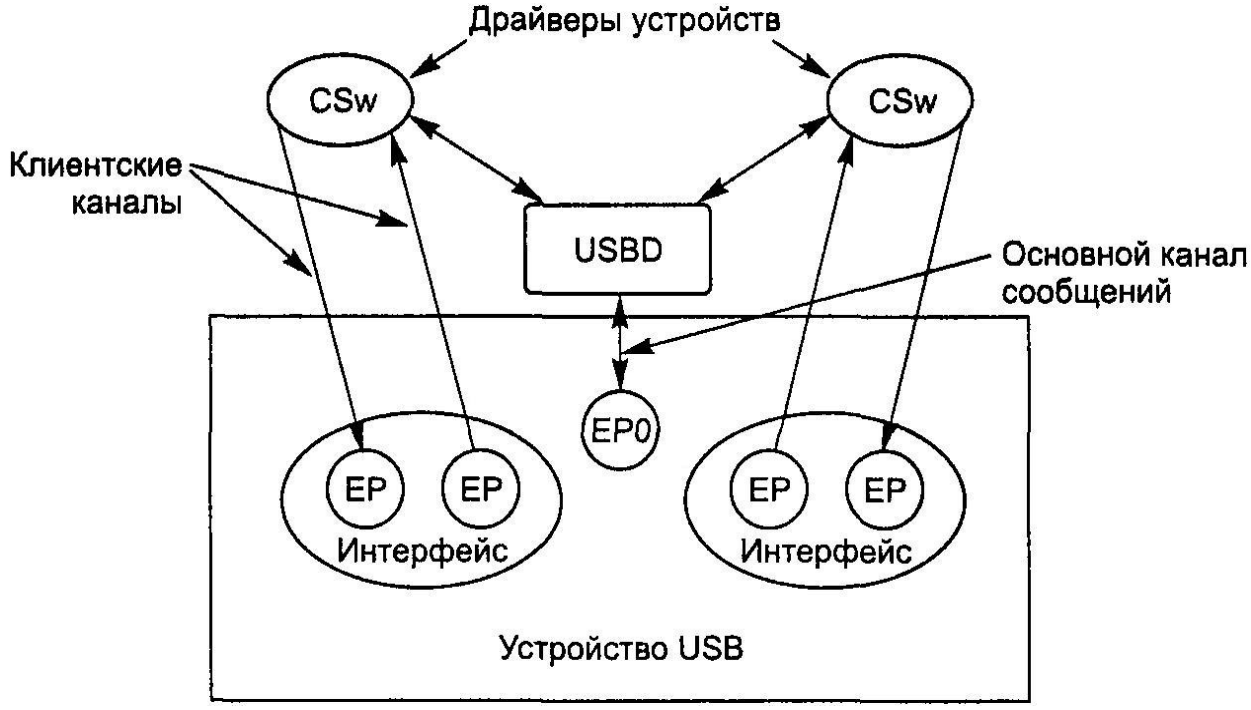


Рис. 5.7. Отношения клиентского ПО (CSw) с интерфейсами устройств USB

5.5.1 Каналы

Коммуникационные каналы USB разделяются на два типа:

- **потоковый капал** (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 в 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 разработчиком 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 не обслуженных 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 капал 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 (передаются три признака в формате регистра состояния LPT-порта: ошибка ввода/вывода, принтер выбран, отсутствие бумаги).

5.5.2 Кадры и микрокадры

Любой обмен по шине USB инициируется хост-контроллером. 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. 5.8).

Каждый кадр начинается с посылки пакета-маркера SOF (Start Of Frame), который является синхронизирующим сигналом для изохронных устройств, а также для хабов. Каждый кадр имеет свой номер. Хост-контроллер оперирует 32-битным счетчиком, но маркер SOF передает только младшие 11 бит. Номер кадра циклически увеличивается во время EOF.

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

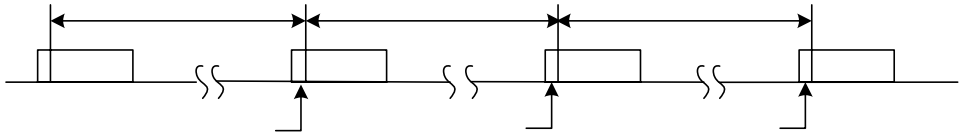


Рис. 5.8. Кадры шины USB

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, 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.

5.6 Протокол шины USB

Кадр N-1

На рис. 5.9 представлена общая схема составляющих USB-протокола

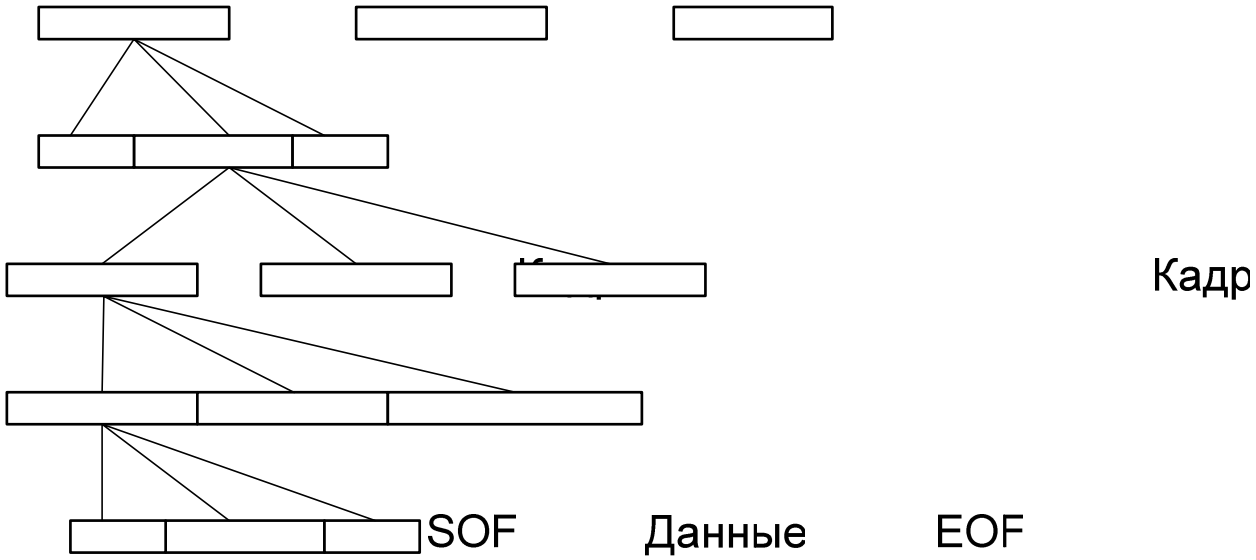


Рис. 5.9. Общая схема составляющих USB-протокола

5.7 Типы передач данных.

Спецификация USB- определяет четыре типа передач данных для конечных точек (после конфигурирования конечной точки доступен лишь один тип передач):

Транзакция Транзакция

1. **Управляющие передачи** (Control Transfers)- используются хостом для конфигурирования устройства во время подключения, для управления устройством и получения статусной информации в процессе работы. Протокол обеспечивает гарантированную доставку таких посылок. Длина поля данных не может превышать 64 байт на полной скорости и 8 байт на низкой скорости. Для таких посылок хост гарантированно выделяет 10% полосы пропускания.

Маркер-пакет Пакеты данных Пакет по

2. **Передачи массивов данных** (Bulk Data Transfers)- применяются при необходимости обеспечения гарантированной доставки данных от хоста к функции или наоборот, но время доставки не ограничено. Такая передача занимает всю допустимую полосу пропускания шины. Пакеты имеют поле данных размером 8, 16, 32, 64 байт. Приоритет у таких передач самый низкий, они могут приостанавливаться при большой занятости шин. Такие типы передач используются принтерами и сканерами.

PID Данные CRC

3. **Передачи по прерываниям** (Interrupt Transfers) - используются в том случае, когда требуется передавать одиночные пакеты данных небольшого размера. Каждый пакет требуется передать за ограниченное время, и операции передачи носят спонтанный характер, при этом должны обслуживаться не медленнее, чем этого требует устройство. Поле данных может содержать до 64 байт на полной скорости и до 8 байт на низкой. Предел времени обслуживания устанавливается в диапазоне 1-255 мс. Такие передачи используются в устройствах ввода (мышь, клавиатура).

4. **Изохронные передачи** (Isochronous Transfers) - применяются для обмена данными в «реальном времени», когда на каждом временном интервале требуется передать строго определенное количество данных, но

доставка информации не гарантирована (передача данных ведется без подтверждения, и при сбоях допускается потеря пакетов). Такие передачи занимают предварительно согласованную часть пропускной способности шины и имеют заданную задержку доставки. Их используют для передачи в мультимедийных устройствах для передачи аудио- и видео-данных.

Для реализации любого типа передач между программным обеспечением хост-контроллера и конечной точкой должно быть организовано логическое соединение- канал (pipe). Каналы образуются на этапе конфигурации конечной точки и могут быть двух видов: потоковые каналы и каналы сообщений.

Вся информация по каналу передается в виде **пакетов**. Каждый пакет начинается с поля синхронизации **SYNC**, за которым следует идентификатор пакета **PID**, поле контрольной суммы идентификатора пакета Check, далее поле данных пакета и заканчивается пакет концевиком **EOF**.

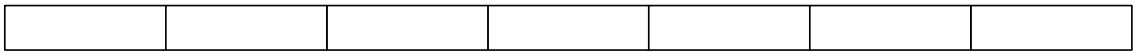
В табл. 5.3 приведен список всех возможных идентификаторов пакетов (PID).

Таблица 5.3. Список кодов PID

Обозначение	КодPID	Источник	Описание
Идентификаторы маркер - пакетов (Token Pocket)			
OUT	0001	Хост	Маркер транзакции вывода, передает адрес и номер конечной точки при передаче от хоста к конечной функции.
IN	1001	Хост	Маркер транзакции ввода, передает адрес и номер конечной точки при передаче от функции хосту.
SOF	0101	Хост	Маркер начала кадра, содержит номер кадра.
SETUP	1101	Хост	Маркер транзакции управления, передает адрес и номер конечной точки при передаче команды от хоста к функции.
Идентификаторы пакетов данных (Data Pocket)			
Data0	0011	Хост, устройство	Пакеты данных с четным и нечетным PID, чередуются для точной идентификации подтверждений.
Data1	1011	Хост, устройство	
Data2	0111	Хост, устройство	Дополнительные типы пакетов данных, используемые в транзакциях с широкополосными изохронными точками.
MData	1111	Хост, устройство	
Идентификаторы пакетов подтверждений (Handshake)			
Ack	0010	Хост, устройство	Подтверждение безошибочного приема пакета
Nak	1010	Устройство	Приемник не сумел принять или передатчик не сумел передать данные. Может использоваться для управления потоком данных («ответ на запрос не готов»)
Stall	1110	Устройство	Произошел сбой в конечной точке или запрос не поддерживается
Nyet	0110	Устройство	Подтверждение безошибочного приема, но указание на отсутствие места для приема следующего пакета.

Рассмотрим форматы некоторых пакетов.

1. Формат пакетов-маркеров **IN**, **OUT**, **SETUP**



PID может принимать значения: 1001, 0001, 1001.

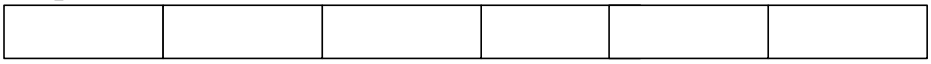
Func [7]- содержит адрес функции (номер физического устройства , который может быть в диапазоне от 0-127)

SYNC[8] PID[4] Check[4]

Endpoint[4]- адрес конечной точки (в диапазоне от 0-15)

CRC[5] – циклический контрольный код.

2. Формат пакета **SOF**

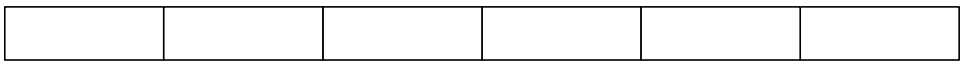


Пакет SOF используется для отметки начала кадра. И хотя хост контроллер оперирует 32-битным счетчиком кадров, в маркере SOF передается только младшие 11 бит, остальная часть номера кадра содержится в регистре ввода-вывода хост-контроллера (FRNUM).

Frame[11]- содержит номер кадра

CRC[5] - циклический контрольный код, вычисленный по полю Frame.

3. Формат пакета данных **Data0**, **Data1**, **Data2**, **MData**

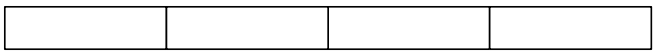


Data [0..8192]- содержит данные размером от 0 до 1023 байт.

SYNC[8] PID[4] Check[4]

CRC[16] - циклический контрольный код, вычисленный по полю Data.

4. Формат пакета подтверждения **Ack**, **Nak**, **Stall**, **Nyet**

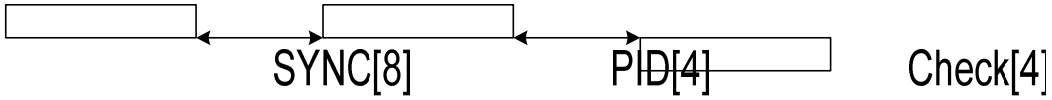


Поле данных пакетов подтверждения пустое.

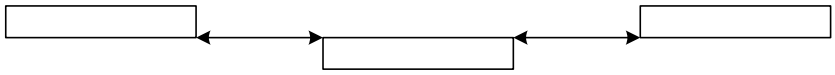
5.8 Протоколы транзакций для различных типов передач

Все обмены (транзакции) по USB состоят из трех пакетов. Хост-контроллер, инициирующий обмен, посылает **маркер-пакет** то есть пакет типа token. Он описывает тип и направление передачи, адрес устройства USB и номер конечной точки. В каждой транзакции возможен обмен только между устройством и хостом. Адресуемое маркером устройство распознает свой адрес и готовится к обмену. Источник данных, определенный маркером, передает **пакет данных** или уведомление об отсутствии данных для передачи. После успешного приема пакета приемник данных посылает **пакет подтверждения** (Handshake).

Типовые последовательности пакетов в транзакциях приведены на рис. 5.10.



а)



Источник б)

Рис. 5.10. Последовательность пакетов в транзакциях на шине USB:
а - вывод; б - ввод данных Хост IN

Устройство

В спецификации USB определены следующие типы транзакций:

Хост ждет

1. Передача команды:

- хост посылает маркер Setup, содержащий номер функции и номер конечной точки, для которой предназначена команда;
- хост посылает выбранной конечной точке пакет данных со сброшенным битом синхронизации (то есть пакет типа Data0), содержащий 8-байтовый код команды;
- функция посылает хосту пакет подтверждения.

2. Изохронная передача данных:

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

3. Изохронный прием данных:

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

4. Передача массивов данных:

- хост посылает маркер Out, содержащий номер функции и номер конечной точки, для которой предназначены данные;
- хост посылает выбранной конечной точке пакет данных Data0;
- функция посылает хосту пакет подтверждения.

5. Прием массивов данных:

- хост посылает маркер In, содержащий номер функции и номер конечной точки, от которой запрашиваются данные;
- выбранная конечная точка передает хосту пакет данных Data0 или пакет подтверждений Nak-данные не готовы;
- если хост получил пакет данных, он посылает пакет подтверждения Ack.

При приеме или передаче каждого блока данных происходит переключение триггера данных. Первый передаваемый (или принимаемый) блок имеет тип Data0, следующий Data1 и так далее. Максимальный размер

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

5.9. 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 данном разделе рассматриваются общие вопросы поведения устройства на шине и механизмы автоматического конфигурирования.

5.9.1. Структура устройства с интерфейсом USB

Периферийное устройство с интерфейсом USB можно разделить на две части - интерфейсную и функциональную (рис. 5.11). Физически они могут объединяться и на одной микросхеме, но логически их функции четко разделимы.

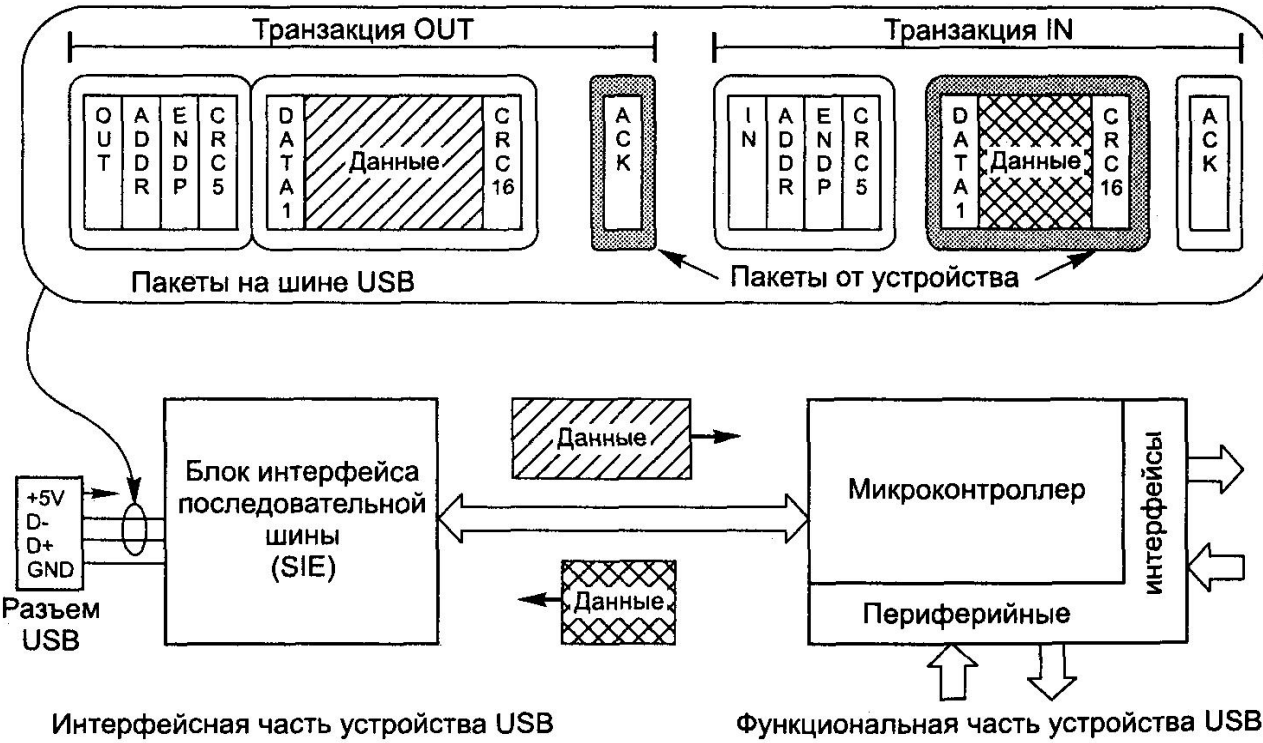


Рис. 5.11. Структура устройства USB

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). 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. 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.

5.9.2. 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 **SetjConfiguration** 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» (SuspendedMode)** - 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.

5.9.3. 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.

5.9.4. Автоматическое конфигурирование

Все устройства подключаются через порты хабов. Хабы определяют подключение и отключение устройств к своим портам 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 назначению 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.

5.9.5. 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 0lh-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 устройствами. 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. 5.4

Таблица 5.4. 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	
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 *(***EPO, Bulk-OUT, Bulk-IN, Interrupt-IN***)* 01 - CBI-транспорт без прерываний *(***EPO, Bulk-OUT, Bulk-IN, Interrupt-IN***),* только для FS 50h - BO-транспорт (**Bulk-OUT, Bulk-IN, EPO**)
Класс 09h - хабы	
00 - деления на подклассы нет	00 -xa6USB 1.x; 01 - хаб USB 2.0 с одним транслятором транзакций; 02 -xa6USB2.0 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

5.9.6. 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**, описывающий устройство в целом (версия USB, класс, производитель, модель, протокол, число возможных конфигураций). Для HS-устройств общее описание дополняется **дескриптором-квалификатором, Device Qualfier Descriptor,** 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**, 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**, описывающие те же параметры с тем же форматом;
- **дескрипторы интерфейсов, Interface Descriptor** , 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** определяют номер и направление точки, атрибуты (тип передач), максимальный размер поля данных и интервал обслуживания (для точек периодических передач);
- **строковые дескрипторы, String Descriptor** - 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 (**LanguagelD**). Строковый дескриптор, вызываемый по нулевому индексу (с любым кодом языка), в своем теле содержит список поддерживаемых идентификаторов языка. Значения **LanguagelD** 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, **Inter/ace 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. 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 подчиняться 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 неизвестна, 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.

5.9.7. 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. 5.5.

Таблица 5.5. Стандартные запросы к устройствам

Запрос	bmRequestType	bRequest
Clear_Feature	00000000b	1
	00000001b	
	00000010b	
Get_Configuration	10000000b	8
Get_Descriptor	10000000b	6
Getjnterface	10000001b	10
Get_Status	10000000b	0
	10000001b	
	10000010b	
Set_Address	00000000b	5

Set_Configuration	00000000b	9
Set_Descriptor	00000000b	7
Set_Feature	00000000b 00000001b 00000010b	3
Setjnterface	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 wVaIue в старшем байте содержит тип дескриптора (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 *(***Ge***t)* 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 wlndex. 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 wlndex 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 GetJStatus может адресоваться к устройству, интерфейсу или конечной точке. Здесь поле wlndex 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); Dl (Remote Wakeup) - возможность подачи сигнала удаленного пробуждения (0 - нет); D2 (Port Test): I - 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 SetJFeature (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 Bi t);
- управление тестовым режимом приемопередатчиков (Test_Mode), адресат - устройство, wValue = 2. Здесь используется и поле wlndex, 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.

5.10. 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.

5.10.1. структура Хаба

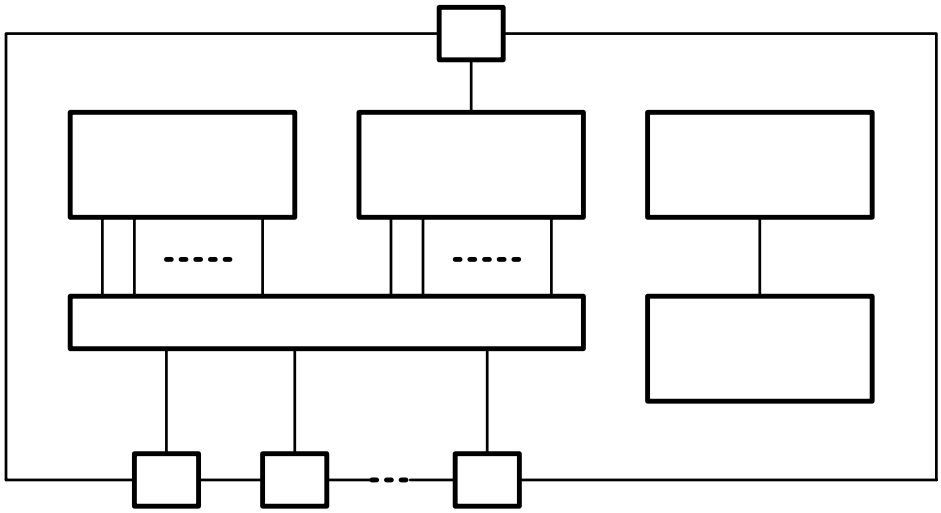


Рис. 5.12. Структура хаба USB 2.0

Структура хаба USB 2.0 приведена на рис. 5.12. 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.

5.10.2. Порты

Со стороны **восходящего порта** хаб выглядит как и любое устройство USB (см. правую часть рис. 5.13). 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.

Транслятор транзакций(ТТ)

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***)*

Маршрутизирую

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 ниже перечисленных **состояний***,* наблюдаемых и управляемых хостом с помощью запросов к хабу:

- **не запитан (Notpowered)** - 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 **SetJPortJEnabl***e,* из состояния **приостановлен** – запросом **ClearJPortJSuspend***;*
- **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 **SetJPort_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.

5.10.3. 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. 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 дает полное представление о возможностях управляемости и наблюдаемости хаба.

5.10.4. 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. 5.13).

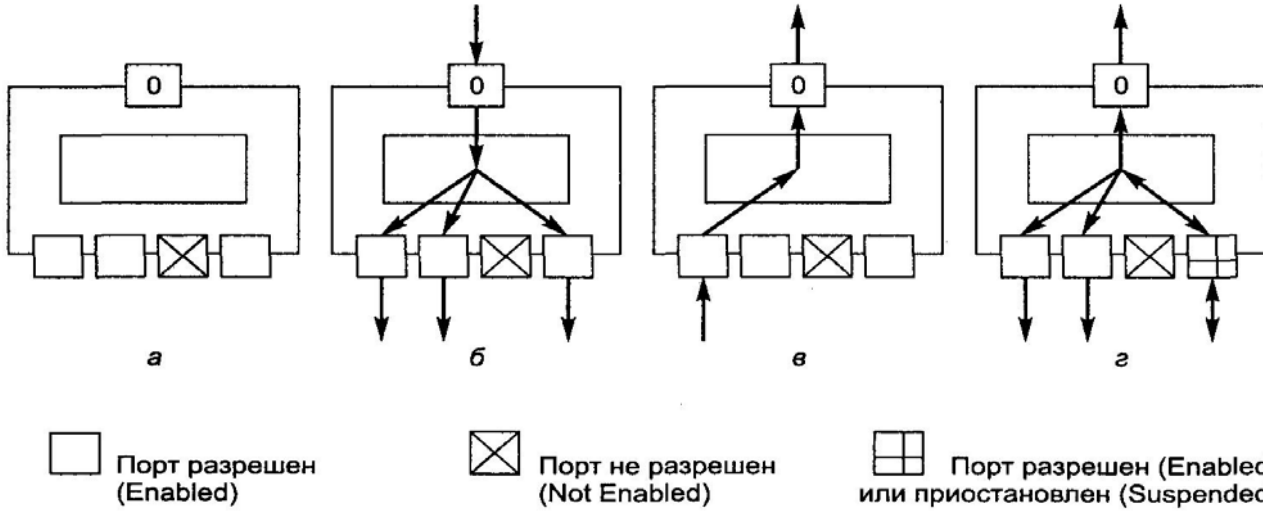


Рис. 5.13. Соединения, обеспечиваемые повторителем хаба: а - покой; б - трансляция пакета с восходящего порта; е - трансляция пакета с нисходящего порта; г - трансляция восходящего возобновления от разрешенного порта

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. 5.13 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.

5.10.5. Запросы к хабам

Хаб имеет всего один интерфейс, в котором используется (кроме нулевой) только одна конечная точка типа **Intermpt-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 пакетом **NAK** (не передает данных). По получению признака изменения состояния хаба хост выполняет запрос чтения состояния хаба (**GetHubStatus**)*,* по получению 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. 5.6).

Таблица 5.6. Классовые запросы хаба

Запрос	bmRequestType	bRequest
ClearHubFeature	00100000B	1
ClearPortFeature	00100011B	1

ClearTTBuffer	00100011B	8
GetHubDescriptoi	10100000B	6
GetHubStatus	10100000B	0
GetPortStatus	10100011B	0
ResetTT	00100011B	9
SetHubDescriptot	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.**

5.11. 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.

«Классический» хост USB делится на три основных уровня:

1. **интерфейс шины USB** обеспечивает физический интерфейс и протокол шины. Интерфейс шины реализуется хост-контроллером, имеющим встроенный корневой хаб, обеспечивающий точки физического подключения к шине (гнезда USB типа «А»). Хост-контроллер отвечает за генерацию 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;

2. **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-контроллера - HCD** (Host Controller Driver) - модуль, привязанный к конкретной модели контроллера, обеспечивающий абстрагирование драйвера USB и позволяющий в одну систему включать несколько разнотипных контроллеров;
- **драйвер USB - USBD** (USB Driver) - обеспечивает основной интерфейс (**USBDI**) между клиентами и устройствами USB. Интерфейс **HCDI** (Host Controller Driver Interface) между **USBD** и **HCD** спецификацией USB не регламентируется. Он 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.

3. **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.

5.11.1. 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 (имеется в виду порядок выполнения запросов к разным конечным точкам. Для каждой конечной точки (канала) порядок завершений всегда соответствует порядку помещения запросов), 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. 5.14.

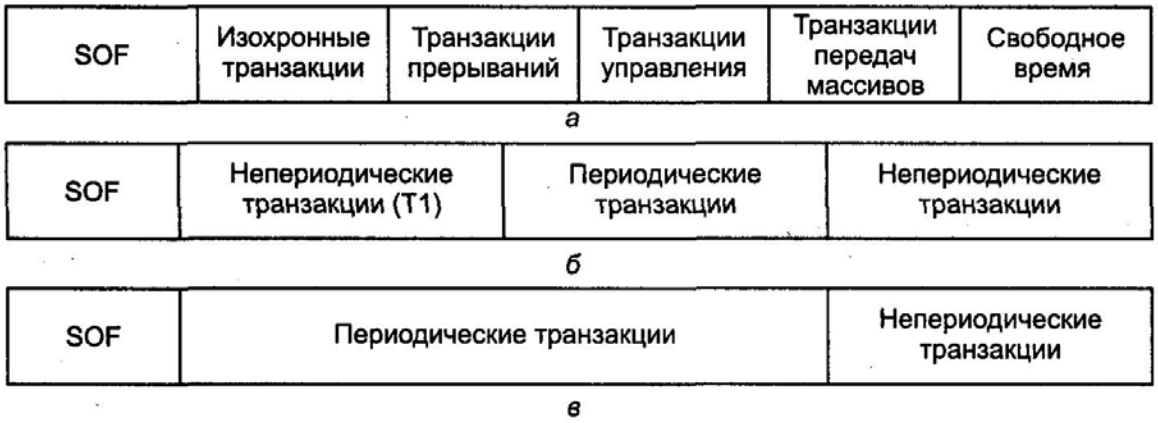


Рис. 5.14. План распределения времени в кадре: а) - UHC; б) - OHC; в) - EHC (в микрокадре)

5.11.2. «Универсальный» хост-контроллер - UHC

Хост-контроллер UHC от Intel появился в микросхеме PHX3 (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.

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. Для транзакций передач с гарантированной доставкой дескрипторы TD приходится организовывать в **очереди**. Очереди нужны для таких передач, поскольку заранее неизвестно, 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 **FrameList** на 1024 кадра. Хост-контроллер обходит списки дескрипторов, начиная с точки, на которую указывает **FrameList** 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.

5.11.3. Структуры данных и регистры контроллера UHC

Драйвер в системной памяти создает **список кадров FrameList**, 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. 5.15. 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).

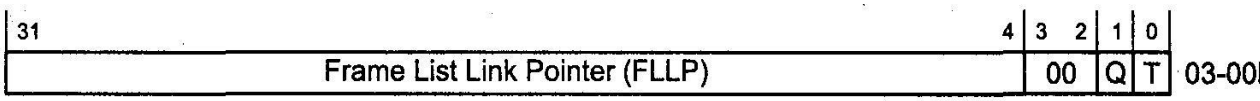


Рис. 5.15. Формат элемента списка кадров для UHL

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**)

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. 5.16. Серым цветом выделены поля, модифицируемые хост-контроллером.

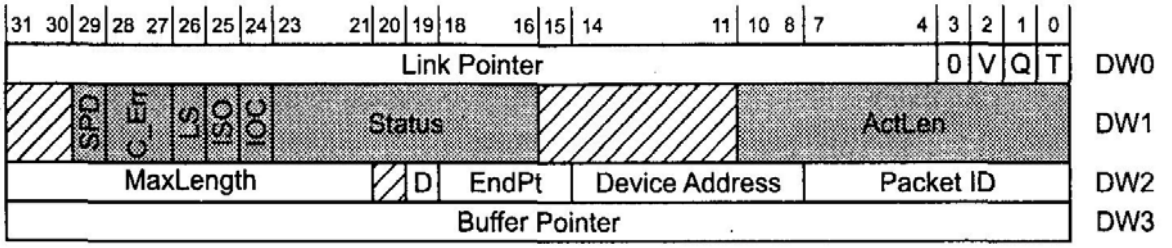


Рис. 5.16. Формат дескриптора передачи для UHC

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 переопустошении 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** (Elh) или **SETUP** (2Dh); DeviceAddress -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 BufferPointer - указатель на буфер в ОЗУ, используемый для данных этой передачи.

- **Заголовок очереди** (**QH**)

Заголовок очереди связывает очереди друг с другом (по горизонтали) и ссылается на первый элемент (TD) данной очереди. Хост-контроллер использует два 32-битных слова (рис. 5.17). 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.

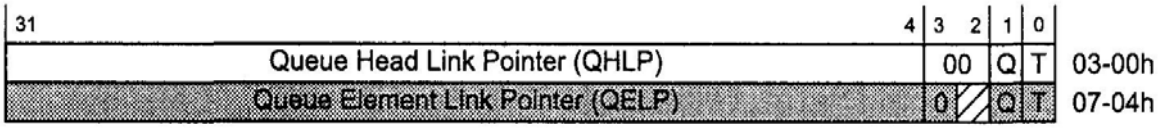


Рис. 5.15. Формат заголовка очереди для UHC

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**

Регистровая модель UHC поясняется в табл. 5.7, 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 5.7. Регистры контроллера UHC

Ад	Назначение
Bas e + (0(0lh)	**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 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

Bas
e + (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

Bas
e + (04
05h)

USBINTR - регистр разрешения прерываний

Биты [15:4] – резерв

Бит 3: Short Packet Interrupt Enable - разрешение прерываний п приему короткого пакета

Бит 2: IOC (Interrupt On Complete Enable) - разрешение прерываний п завершении транзакции

Бит 1: Resume Interrupt Enable - разрешение прерываний по прием сигнала возобновления

Бит 0: Timeout/CRC Interrupt Enable - разрешение прерываний п ошибке тайм-аута и CRC-контролю

Bas
e + (06
07h)

FRNUM - регистр номера кадра

Bas
e + (08
0Bh)

FRBASEADD - регистр базового адреса списка кадров

Bas
e +
0Ch

SOFMOD - регистр управления частотой кадров

Биты [6:0] - управление длительностью кадра: 0 - 11936 бит, 1 - 1193 бит, 63-11999 бит, 64 -12000 бит (номинал), 65- 12001 бит, 127 - 12063 бит

Bas
+ (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

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+

Бит3: (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

Bas + (12-13h **P0RTSC2 - регистр управления и состояния порта 2** (аналогично предыдущему)

Литература

1. Иванов Е.Л. и др. Периферийные устройства ЭВМ и систем. - М.: Высшая школа, 1987.

2. Ларионов А.М., Горнец Н.Н. Периферийные устройства в вычислительных системах. - М.: Высшая школа, 1991

3. Лю Ю-Чжен, Гибсон Г. Микропроцессоры семейства 8086/8088. Архитектура, программирование и проектирование микрокомпьютерных систем: Пер. с англ.- М.: Радио и связь, 1987.

4. Дженнингс Р. Практическая передача данных. Модемы, сети и протоколы. - М.: Мир, 1989.

5. Гук М.Ю. Шины PCI, USB и FireWire. Энциклопедия.-СПб.: Питер,2005.

6. Кулаков В. Программирование на аппаратном уровне: специальный справочник. 2-е изд.- СПб.: Питер, 2003.

7. Джордейн Р. Справочник программиста персональных компьютеров типа IBM PC, XT и AT. – М.:Финансы и статистика, 1992.

СОДЕРЖАНИЕ Стр