Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра Системного Программирования

Абрамов Иван Александрович

Исследование режимов гарантированной доставки данных контроллеров USB последних версий

Бакалаврская работа

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Терехов А. Н.

Научный руководитель:
ст. преп. Баклановский М. В.

Рецензент:
к. ф.-м. н., доцент Герасимов М. А.

Санкт-Петербург
2015

SAINT-PETERSBURG STATE UNIVERSITY

Software Engineering Chair

Ivan Abramov

Investigation of guaranteed data delivery modes for latest versions of USB controllers

Bachelor's Thesis

Admitted for defence.
Head of the chair:
professor Andrey Terekhov

Scientific supervisor:
assistant professor Maxim Baklanovsky

Reviewer:
associate professor Mikhail Gerasimov

Saint-Petersburg
2015

Оглавление

Введение

Стремительный рост числа периферийных устройств, подключаемых к персональному компьютеру, стал причиной, по которой крупнейшие компании [1] в 1994 году приступили к разработке нового стандарта, описывающего универсальный последовательный интерфейс взаимодействия. Представленный стандарт USB (Universal Serial Bus, универсальная последовательная шина) определил единый для всех устройств протокол конфигурирования и смог успешно завоевать рынок благодаря простоте в использовании, поддержке составных устройств и возможности передачи данных в различных режимах.

На сегодняшний день большинство периферийных устройств спроектированы в соответствии со стандартом USB. Представители семейств операционных систем Unix и Windows уже содержат полноценные реализации интерфейса USB, но есть задачи, для решения которых операционные системы общего назначения использоваться не могут. Так, при тестировании производительности аппаратного обеспечения важно, чтобы проводимые измерения были максимально точными и чтобы фоновые процессы не оказывали влияние на результаты тестирования, что не гарантируется в случае использования операционных систем общего назначения. Для решения описанной задачи создается специализированное программное обеспечение. Примером такого проекта может служить тестовая операционная система TestOS, разрабатываемая на кафедре системного программирования СПбГУ, где в качестве основного запоминающего устройства планируется использовать распространённые USB-флеш-накопители. Кроме этого, разработчики операционной системы BareMetalOS [2] хотели бы иметь возможность производить загрузку системы с USB-флеш-накопителя. По этой причине возникает необходимость обеспечить поддержку интерфейса USB.

Перед тем как начать непосредственную реализацию поддержки, требуется выделить ключевые элементы шины USB. Основной составляющей шины является хост-контроллер — аппаратное средство, управляющее работой подключённых устройств. USB-устройством является либо хаб, обеспечивающий дополнительные порты для подключения, либо функция — устройство, передающее информацию по шине.

Реализуемая программная поддержка, обеспечивающая взаимодействие с USB-устройствами, должна согласовываться с выбранной версией стандарта. Современные устройства спроектированы по стандартам USB 2.0 [7] и USB 3.0 [6], поэтому при разработке логично ориентироваться именно на них. Принятое решение подразумевает работу с реализациями хост-контроллера версий EHCI (Enhanced Host Controller Interface) [1] и xHCI (eXtensible Host Controller Interface) [2]. Выбранная версия реа-

[1] Intel Corporation, Microsoft Corporation, IBM, Compaq Computer Corporation, Digital Equipment Corporation, NEC Corporation, Northern Telecom Limited

[2] Операционная система BareMetalOS — URL: `https://github.com/ReturnInfinity/BareMetal-OS` (дата обращения: 18.04.2015)

лизации определяет интерфейс взаимодействия, включает в себя описание процесса инициализации хост-контроллера и требования к формируемым структурам данных. Используя функциональность, предоставленную хост-контроллером, требуется осуществить идентификацию и нумерацию USB-устройств, учитывая, что устройство могло быть подключено к компьютеру через промежуточный хаб.

В данной работе рассматривается стандарт USB 2.0 и соответствующая ему реализация хост-контроллера версии EHCI. Стандарт USB 2.0 определяет группу режимов передачи, для которых доставка данных гарантируется протоколом: режим управляющей передачи данных, режим передачи массивов данных, режим передачи по прерыванию. Каждое USB-устройство должно поддерживать режим управляющей передачи данных, в котором происходит конфигурирование подключенных устройств. Дополнительные поддерживаемые режимы передачи зависят от класса конкретного устройства. Основное внимание будет уделено устройствам хранения данных (mass storage class), так как модуль для работы с устройствами данного класса должен стать базовым для операционной системы TestOS. Устройства хранения данных работают в режиме передачи массивов данных, реализация программной поддержки которого может основываться на различных подходах к организации процесса передачи данных, которые требуется исследовать.

Постановка задачи

Целью данной работы является исследование режимов гарантированной доставки данных для хост-контроллера версии EHCI. Для достижения поставленной цели были сформулированы следующие задачи.

1. Реализовать взаимодействие с хост-контроллером версии EHCI:
   - идентифицировать хост-контроллер;
   - реализовать модуль, обеспечивающий доступ к регистрам хост-контроллера;
   - инициализировать хост-контроллер;
   - реализовать драйвер хост-контроллера.
2. Реализовать поддержку режима управляющей передачи данных.
3. Реализовать поддержку режима передачи массивов данных для устройств хранения данных и исследовать подходы к организации процесса передачи.
4. Провести апробацию реализованной функциональности:
   - получить и проверить конфигурационную информации для USB-устройств;
   - провести сравнение различных подходов к организации процесса передачи данных для USB-флеш-накопителей.

1. Обзор решений

Для того чтобы провести полноценное исследование интерфейса и режимов передачи данных USB, требуется программно реализовать взаимодействие с хост-контроллером. Решение данной задачи подразумевает, что будет проводиться низкоуровневое конфигурирование аппаратного обеспечения. Ввиду специфики задачи было принято решение осуществлять разработку программного обеспечения на языке ассемблера. Учитывая эту особенность, были проанализированы проекты, где решалась описанная задача в схожих условиях.

К рассматриваемым решениям предъявлялись следующие требования:

- реализована поддержка хост-контроллера версии EHCI;
- реализована работа с USB-хабами;
- реализована работа с устройствами хранения данных;
- есть возможность отделить функциональность, отвечающую за работу с USB, от основных составляющих системы.

1.1. Существующие решения

1.1.1. BIOS

BIOS (Base Input Output System) — базовая система ввода-вывода. В зависимости от конкретной реализации BIOS предоставляет ограниченную функциональность для работы с USB-устройствами. Чаще всего речь идёт об обработке данных, полученных от USB-клавиатуры или мыши. Отдельно стоит выделить возможность работы с USB-накопителем как с жёстким диском. Эта особенность даёт возможность реализовать загрузку системы, например, с флеш-накопителя.

Но есть и ряд ограничений, из-за которых описанную функциональность можно считать недостаточной. Во-первых, большинство сервисов BIOS доступны только в реальном режиме. Во-вторых, нет возможности узнать конфигурационную информацию используемого USB-устройства. Кроме этого, нет возможности подключить USB-оборудование к системе во время работы и приступить к использованию оборудования без остановки системы. В рамках исследования BIOS [3] при попытке копировать файл размером 22 ГБ с USB-флеш-накопителя на жёсткий диск, используя стандартную функциональность, было обнаружено зависание программного обеспечения.

[3]Intel BIOS версии BGP6710J.86A — URL: `https://downloadcenter.intel.com/download/21687/BIOS-Update-BGP6710J-86A-` (дата обращения: 18.05.2015)

1.1.2. TatOS

TatOS [4] — любительская операционная система, написанная на языке ассемблера. Основным плюсом tatOS, по заявлениям разработчика, является наличие драйвера для хост-контроллеров версий UHCI (Universal Host Controller Interface) и EHCI. Также в рамках данной операционной системы реализован драйвер для USB-мыши и базовая функциональность для работы с флеш-накопителями. Большим плюсом tatOS является то, что программный код детально описан. Для некоторых запросов приведены результаты тестирования на конкретном аппаратном обеспечении и выделены особенности, найденные в процессе разработки. Полезными являются примеры транспортных пакетов, содержащих SCSI-команды для устройств хранения данных.

Недостатком tatOS является то, что в данном проекте не реализована работа с USB-хабами. Проблема заключается в том, что изначально в спецификации [1], разработанной Intel, предполагалось использовать наряду с основным EHCI-контроллером еще и контроллер-компаньон версии UHCI или OHCI (Open Host Controller Interface). Контроллер-компаньон отвечал за работу с низкоскоростными устройствами. Позже в Intel отказались от архитектуры с двумя хост-контроллерами, а вместо этого к корневому хабу присоединили промежуточный хаб, который получил название RMH (Rate Matching Hub). Именно через RMH теперь происходит обмен данными с периферийными USB-устройствами. Из-за отсутствия поддержки USB-хабов в tatOS нет возможности взаимодействовать с устройствами, подключенными к RMH.

1.1.3. KolibriOS

KolibriOS [5] — популярная операционная система, написанная на языке ассемблера, работающая на платформе x86. В данном проекте реализованы драйверы для хост-контроллеров версий UHCI, OHCI, EHCI. KolibriOS содержит модуль для работы с устройствами хранения данных и с устройствами ввода-вывода. В данном решении реализована полноценная поддержка USB-хабов. Функциональность, отвечающая за работу с USB в KolibriOS, заточена по конкретную систему, поэтому данное решение сложно переиспользовать.

1.2. Выводы

Рассмотренные проекты не удовлетворяют предъявленным требованиям, поэтому было принято решение самостоятельно реализовывать взаимодействие с хост-контроллером и осуществлять программную поддержку требуемых режимов передачи данных.

[4] Операционная система TatOS — URL: `https://code.google.com/p/tatos/` (дата обращения: 15.04.2015)

[5] Операционная система KolibriOS — URL: `http://kolibrios.org/` (дата обращения: 10.05.2015)

2. Архитектура шины USB

Архитектура шины USB подразумевает, что в состав персонального компьютера входит хост-контроллер — аппаратное средство, управляющее работой всех подключенных к портам устройств. USB-устройством является либо хаб, обеспечивающий дополнительные порты на шине, либо функция — устройство, подключенное к порту и передающее информацию по шине. Хаб, встроенный в хост-контроллер, называется корневым. Для физического соединения устройств используется топология многоярусной звезды, где центром каждой звезды является хаб. Архитектура шины USB схематически представлена на рис. 1.

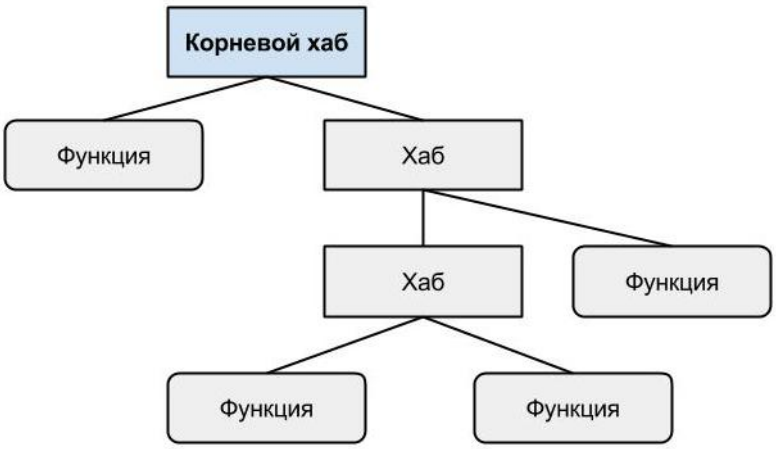

Рис. 1: Физическая архитектура шины USB

Операции обмена данными по шине USB инициируются хост-контроллером, а требуемое периферийное устройство лишь реагирует на полученные команды. Для описания процесса обмена требуется определить логические уровни, на которых происходит взаимодействие. Согласно распространенному подходу, описанному в [8], выделяют три логических уровня, характеризующих взаимодействие на стороне пользовательского компьютера (далее хоста), и три уровня, характеризующих взаимодействие на стороне USB устройства (рис. 2).

Для хоста соответствующими уровнями являются:

- уровень драйвера клиентского программного обеспечения;
- уровень системного драйвера;
- уровень хост-контроллера.

На уровне драйвера клиентского программного обеспечения происходит взаимодействие пользовательского приложения с операционной системой. На этом этапе вы-

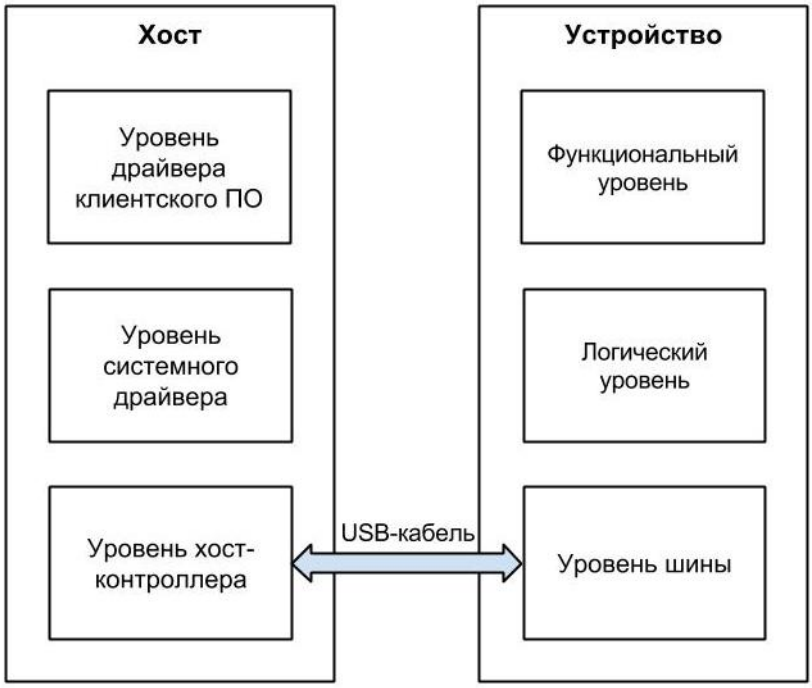

Рис. 2: Логические уровни взаимодействия шины USB

ясняется тип передачи, а требуемые данные переходят на уровень системного драйвера.

На уровне системного драйвера происходит назначение адресов подключённым USB-устройствам. Поэтому, когда клиентские данные поступают на данный уровень, системный драйвер формирует транзакционные последовательности для конкретного устройства и передает их драйверу хост-контроллера. Задачей драйвера хост-контроллера, который принадлежит уровню системного драйвера, является планирование исполнения полученных транзакций и отслеживание статуса их выполнения.

Уровень хост-контроллера характеризует используемая реализация хост-контроллера. Данная реализация описывает структуры данных, которые создает и заполняет драйвер хост-контроллера, обрабатывая транзакции. Транзакции разбиваются на небольшой набор пакетов, которые хост-контроллер отправляет принимающему устройству. Пакеты уже не содержат адреса, но снабжаются контрольной суммой, вычисляемой хост-контроллером для обеспечения помехоустойчивости.

Для периферийного USB-устройства выделяют следующие логические уровни:

- функциональный уровень;
- логический уровень;
- уровень шины.

На функциональном уровне периферийное устройство воспринимается как функ-

ция, которая может иметь несколько интерфейсов, определяющих возможности устройства. На этом уровне устройство оперирует данными, полученными от клиентского программного обеспечения, или направляет запрашиваемые данные на логический уровень устройства.

На логическом уровне USB-устройство представлено как набор конечных точек, с которыми взаимодействует уровень системного драйвера со стороны хоста. Конечная точка является пунктом назначения для отдельной транзакции. Транзакционные последовательности, соответствующие различным типам передачи, обрабатываются различными конечными точками.

На уровне шины происходит обмен пакетами между хост-контроллером и периферийным устройством. Корректно полученные пакеты передаются на логический уровень.

Представленное описание дает общее понимание того, как должен быть организован процесс обмена данными по шине USB и какие элементы системы в нем задействованы. Кроме того, для каждого уровня определены его функции и задачи. Но есть в этом описании и ряд недостатков. Во-первых, нет объяснения того, как различные уровни должны взаимодействовать. Во-вторых, нет четкого разделения сфер ответственности программного и аппаратного обеспечения. Ответы на некоторые вопросы дает статья [6], в которой описан альтернативный взгляд на логические уровни USB. Если дополнить статью примерами из спецификации [7], то можно получить более целостную картину происходящего.

Предлагается рассмотреть стек USB, используя уровни сетевой модель OSI:

- прикладной уровень;
- уровень передачи;
- транзакционный уровень;
- пакетный уровень;
- физический уровень.

На физическом уровне кодируется битовый поток. Данная функиональность аппаратно реализуется хост-контроллером и для разработчика программного обеспечения является прозрачной. Подробное описание уровня представлено в 7-й главе [7].

На пакетном уровне между хост-контроллером и USB-устройством передаются пакеты следующих типов:

- пакет-маркер (token packet);

[6]Статья: «Разбираем и собираем обратно стек USB» — URL: `http://habrahabr.ru/post/236401/` (дата обращения: 15.05.2015)

- пакет данных (data packet);
- пакет-подтверждение (handshake packet);
- специальный пакет (special packet).

Каждый пакет имеет собственный формат. Например, на рис. 3 представлен формат пакета данных.

Рис. 3: Формат пакета данных

Поле SYNC (SYNChronisation) используется для синхронизации тактов приемника, поле PID (Packet IDentifier) является идентификатором пакета, поле Data содержит передаваемые данные, поле CRC16 (Cyclic Redundancy Checksum) является контрольной суммой, поле EOP (End Of Packet) — символ конца пакета. Описание структуры пакетов остальных типов представлено в 8-й главе [7]. Пакеты требуемого типа формируются хост-контроллером на основе полученной транзакции. Хост-контроллер автоматически заполняет поля пакета, поэтому на пакетном уровне вмешательство программного обеспечения не требуется.

На транзакционном уровне происходит обмен транзакциями между драйвером хост-контроллера и конечной точкой. Транзакция — небольшой набор пакетов, следующих друг за другом. Для описания транзакции конкретная реализация хост-контроллера требует, чтобы были сформированы связанные структуры — начало очереди (queue head) и дескриптор передачи (transfer descriptor). Данные структуры содержат информацию о номере конечной точки устройства, которой адресована транзакция, а также указатель на данные, которые необходимо передать. Драйвер хост-контроллера формирует начало очереди и дескрипторы передач, а потом уведомляет об этом хост-контроллер, который на основе перечисленных структур формирует пакеты. Разработчик должен реализовать драйвер хост-контроллера.

На уровне передачи рассматриваются различные типы конечных точек. Для каждого типа определён один из четырёх режимов передачи:

- управляющие передачи (control transfers) — используются для конфигурирования устройств;
- передачи массивов данных (bulk data transfers) — применяются для гарантированной доставки больших объемов данных;
- передачи по прерываниям (interrupt transfers) — используются для передачи малых объёмов данных за фиксированное время;

- изохронные передачи (isochronous transfers) — применяются для негарантированной доставки данных, которые должны быть обработаны за определённое время.

Каждый режим передачи имеет приоритет, влияющий на резервирование пропускной способности шины, который учитывает планировщик транзакций. Произвольное USB-устройство должно поддерживать режим управляющей передачи, который связан с нулевой конечной точкой. Информация об остальных конечных точках выясняется в процессе конфигурирования. Полученная информация характеризует возможности устройства.

На прикладном уровне решаются конкретные задачи: идентификация устройства, нумерация устройства, запись данных на устройство-накопитель и т.д. Системный драйвер формирует запросы, которые отправляются на конечную точку заданного типа. Например, для выяснения класса устройства системный драйвер отправляет на нулевую контрольную точку запрос на дескриптор устройства (device descriptor), в котором одно из полей содержит данные о классе. Аналогичным образом передаются SCSI-команды для устройств хранения данных. Реализуя работу на прикладном уровне, разработчик должен следовать указаниям в спецификации, регламентирующей правила взаимодействия.

3. Реализация взаимодействия с хост-контроллером

Взаимодействие с хост-контроллером организуется при помощи управляющих регистров. Спецификация [1] разделяет регистры на два пространства:

- регистры конфигурационного пространства PCI (Peripheral Component Interconnect), которые используются для нумерации компонентов системы;
- регистры, проецируемые в память, которые характеризуют возможности хост-контроллера и его текущее состояние.

Адрес, указывающий на регистры, проецируемые в память, определен в одном из регистров конфигурационного пространства PCI. Чтобы сконфигурировать хост-контроллер, необходимо:

1. идентифицировать хост-контроллер, используя механизмы конфигурационного пространства PCI, и определить местоположение регистров, проецируемых в память;
2. перевести процессор в гибридный режим, чтобы получить доступ к регистрам, проецируемым в память;
3. инициализировать хост-контроллер.

Когда хост-контроллер сконфигурирован, требуется предоставить механизм, отвечающий за обработку транзакций. Данную задачу решает драйвер хост-контроллера, учитывающий особенности реализации хост-контроллера рассматриваемой версии.

3.1. Идентификация хост-контроллера

Шина PCI — основная шина персонального компьютера, которая предназначена для соединения контроллеров, расположенных на системной плате или подключенных в слоты расширения, друг с другом, с процессором и памятью. С каждым устройством, подключенным к шине PCI, связана структура из 256 байт — конфигурационное пространство PCI (PCI configuration space), в которой содержится информация о устройстве и то, какие ресурсы занимает устройство. Для обращения к конфигурационному пространству PCI используются два порта ввода-вывода:

1. порт адреса, через который задаётся адрес PCI-устройства;
2. порт данных, через который происходит чтение и запись данных в конфигурационном пространстве.

Таблица 1: Адрес PCI-устройства

Бит 31	Биты 30-24	Биты 23-16	Биты 15-11	Биты 10-8	Биты 7-2	Биты 1-0
Всегда 1	Зарезервированы	Номер шины	Номер устройства	Номер функции	Номер регистра	Всегда 0

Адрес PCI-устройства представляется в виде 32-х битной структуры (таблица 1).

Номер шины, устройства и функции определяют адрес, по которому располагается конфигурационное пространство физического устройства, подключенного к шине PCI. Используя номер регистра, можно читать и записывать данные конфигурационного пространства PCI (рис. 7 в приложении). Полное описание структуры конфигурационного пространства PCI содержится в спецификации [3]. Для нас основной интерес представляет поле класса (class code), по которому определяется тип устройства. Чтобы идентифицировать конфигурационное пространство PCI, соответствующее хост-контроллеру, нужно иметь значения данного поля. Поле класса состоит из трех частей: базового класса, подкласса и интерфейса. В спецификации [1] сказано, что для хост-контроллера версии EHCI описанные поля должны принимать значения, представленные в таблице 2.

Таблица 2: Значения поля класса для хост-контроллера версии EHCI

Имя поля	Значение
Базовый класс	0x0C
Подкласс	0x03
Интерфейс	0x20

После того как конфигурационное пространство конкретного хост-контроллера версии EHCI было найдено, оттуда необходимо извлечь регистры базовых адресов (base address registers). Ненулевой регистр базовых адресов указывает на область в памяти, где находятся проецируемые в память регистры.

3.2. Реализация модуля для доступа к регистрам хост-контроллера

Область памяти, где располагаются проецируемые регистры, не доступна из режима реальных адресов, в котором находится процессор на этапе запуска компьютера. Для того, чтобы иметь доступ к требуемой памяти, необходимо перевести процессор в защищенный режим, в котором работает значительное число операционных системы общего назначения. Для этого была определена глобальная таблица дескрипто-

ров GDT (Global Descriptor Table) — служебная структура данных, характеризующая сегменты, через которые осуществляется доступ к памяти. Сегмент описывается дескриптором. Исчерпывающая информация о том, как заполняются поля дескрипторов, описывающие базовый адрес и предельное смещение в сегменте, тип сегмента и уровень привелегий, предоставляется ресурсом [10]. Сформированные дескрипторы последовательно размещаются в глобальной таблице дескрипторов. После этого при помощи регистров управления (control registers) осуществляется переход в защищённый режим.

Сервисы BIOS и MS-DOS, используемые при отладке программы, доступны только в реальном режиме. Для того, чтобы повторно не создавать программное обеспечение для отладки и работы с периферийными устройствами на низком уровне для защищённого режима, было принято решение работать в гибридном режиме процессора, также известном под названием unreal mode. Для этого потребовалось осуществить переход из защищенного режима в реальный, но с сохранением возможности адресовать память, доступную в защищённом режиме, через теневые регистры. Данная техника, предложенная Томасом Роденом, описана в книге [9].

3.3. Инициализация хост-контроллера

Проецируемые в память регистры делятся на две группы:

- регистры возможностей (capability registers), описывающие возможности реализации хост-контроллера;
- операционные регистры (operational registers), управляющие поведением хост-контроллера и характеризующие его состояние.

Чтобы привести регистры хост-контроллера в состояние по умолчанию и начать процесс инициализации, требуется произвести сброс хост-контроллера. На этом этапе важно учесть, что поддержка интерфейса USB может осуществляться на уровне BIOS. Чтобы урегулировать процесс обслуживания хост-контроллера между BIOS и реализуемым программным обеспечением, разработчиками были определены вспомогательные регистры USB Legacy Support, располагающиеся в конфигурационном пространстве PCI по смещению, указанному в регистре из группы регистров возможностей. После того, как права на обслуживание хост-контроллера были получены от BIOS, производится инициализация хост-контроллера согласно плану, определённому в 4-й главе [1]. План требует, чтобы в операционных регистрах были указаны типы прерываний, которые обрабатывает хост-контроллер, адреса используемых структур данных и права на управление портами корневого хаба.

3.4. Реализация драйвера хост-контроллера

Обмен данными между устройством и хост-контроллером происходит при помощи транзакций. Для описания транзакций режимов управляющей передачи данных и передачи массивов данных реализация хост-контроллера версии EHCI требует, чтобы были созданы связанные структуры данных, на основе которых хост-контроллер формирует пакеты. Этими структурами данных являются начало очереди (QH, Queue Head) (рис. 4) и транспортный дескриптор очереди (qTD, Queue element Transfer Descriptor) (рис. 5). Каждый транспортный дескриптор очереди содержит указатель на следующий транспортный дескриптор, если тот существует. Начало очереди указывает на первый транспортный дескриптор, а указатель на начало очереди должен размещаться в операционном регистре хост-контроллера (в регистре ASYNCLISTADDR, Asynchronous List Address Register).

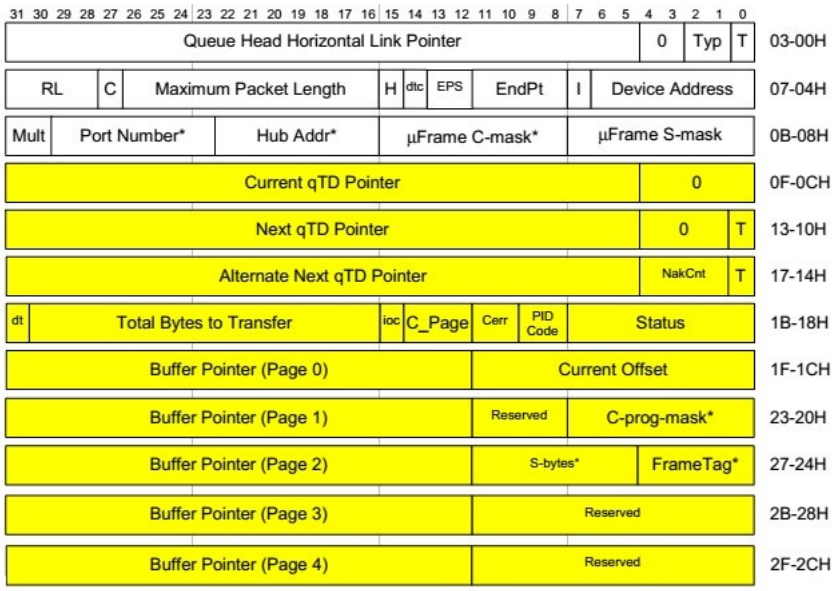

Рис. 4: Структура начала очереди (рис. из [1])

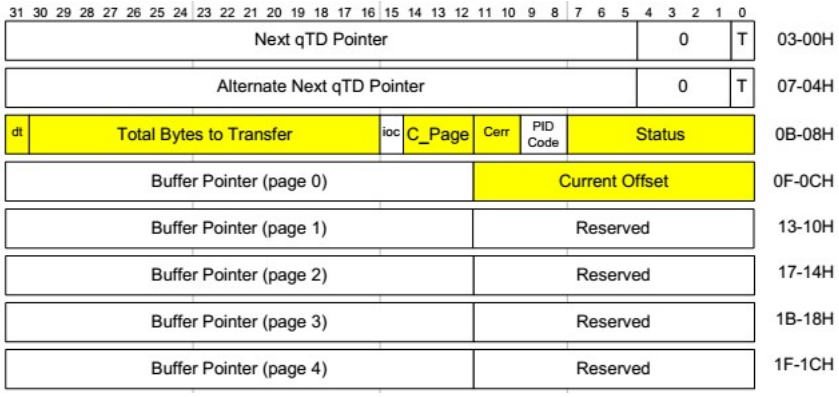

Рис. 5: Структура транспортного дескриптора очереди (рис. из [1])

Для обработки транзакций потребовалось реализовать драйвер хост-контроллера, в задачи которого входит:

- создание начала очереди и цепочки транспортных дескрипторов очереди для конкретной транзакции;
- отслеживание статуса выполнения транзакции;
- мониторинг состояния хост-контроллера.

Информация о том, какому устройству и для какой конечной точки предназначена транзакция, должна быть указана в структуре начала очереди. Также начало очереди должно располагать информацией о характеристиках устройства. Транспортные дескрипторы очереди описывают данные, которые необходимо передать или получить от конечной точки устройства. Правила для заполнения полей основных структур данных представлены во 2-й главе [1].

После обработки транзакции хост-контроллер обновляет поля, отмеченные на рис. 4 и рис. 5. Если транзакция была корректно обработана и передана устройству, то в поле статуса начала очереди и транспортных дескрипторов будет занесена отметка о успешном завершении. При возникновении ошибки в статусное поле будет занесено значение, описывающее природу неполадки. В случае возникновения серьёзной проблемы работа хост-контроллера будет прервана. Состояние хост-контроллера можно будет выяснить, прочитав операционный регистр USBSTS (USB Status Register).

Реализованный драйвер хост-контроллера создаёт начало очереди и транспортные дескрипторы очереди для транзакций. В случае возникновения ошибки при выполнении некоторой транзакции драйвер интерпретирует поле статуса и отображает уведомление о неполадке на экране.

В дальнейшей работе функциональность драйвера хост-контроллера использовалась для организации взаимодействия в режиме управляющей передачи данных и в режиме передачи массивов данных.

4. Реализация поддержки режима управляющей передачи данных

Режим управляющей передачи данных используется для конфигурирования периферийных USB-устройств. USB-устройством может быть хаб, который предоставляет дополнительные порты на шине для подключения устройств, или функция — устройство, передающее информацию по шине. Режим управляющей передачи данных связан с нулевой конечной точкой. Произвольное USB-устройство должно корректно обрабатывать служебные транзакции, посылаемые на нулевую конечную точку.

При реализации поддержки режима управляющей передачи данных важно обратить внимание на физическую архитектуру шины USB (рис. 1). Для того, чтобы сконфигурировать периферийное USB-устройство, первоначально необходимо инициализировать корневой хаб, встроенный в хост-контроллер. Корневой хаб хост-контроллера версии EHCI заточен на работу с высокоскоростными устройствами (high speed) и не обеспечивает работу с устройствами, работающими в скоростных режимах full speed и low speed, которые были определены в стандарте USB 1.1. В спецификации [1] предполагалось, что взаимодействовать с низкоскоростными устройствами будут контроллеры-компаньоны версий UHCI и OHCI, но позже от архитектуры с контроллерами-компаньонами отказались. В Intel было принято решение поставлять хост-контроллер совместно с промежуточным хабом, подключаемым к корневому хабу. Промежуточный хаб получил название RMH (Rate Matching Hub), и обмен данными с USB-устройствами осуществляется только через него. Отметим, что в данной архитектуре промежуточный хаб — это единственное устройство, подключаемое к корневому хабу.

Чтобы инициализировать корневой хаб, достаточно произвести сброс его портов, используя операционные регистры PORTSC (Port Status and Control Register). Регистр PORTSC отображает информацию о конкретном порте корневого хаба. Если к порту было подключено устройство, то после сброса порта устройство переходит в состояние по умолчанию и становится доступным для конфигурирования.

Процесс конфигурирования периферийного устройства можно разделить на два этапа:

1. общий для всех устройств этап конфигурирования;
2. дополнительный этап для USB-хабов.

Сначала рассматривается общий для всех устройств этап конфигурирования. Предполагается, что на этом этапе устройство находится в состоянии по умолчанию и готово обрабатывать транзакции, направляемые на нулевую контрольную точку. Для выполнения операций, например, для чтения характеристик устройства, требуется

несколько транзакций, поэтому их формируют в группы. Группы транзакций можно разделить на 3 типа, в зависимости от операций, которые они выполняют.

- Группа чтения данных (control read):
  1. транзакция, содержащая запрос;
  2. транзакция, читающая данные с устройства;
  3. транзакция-подтверждение.
- Группа записи данных (control write):
  1. транзакция, содержащая запрос;
  2. транзакция, записывающая данные на устройство;
  3. транзакция-подтверждение.
- Группа без данных (no-data control):
  1. транзакция, содержащая запрос;
  2. транзакция-подтверждение.

Первая транзакция каждой группы должна содержать запрос, характеризующий группу транзакций и операцию, которую они выполняют. Все запросы имеют одинаковую структуру, представленную в таблице 3.

Таблица 3: Структура запроса

Поле	Описание	Размер в байтах
bmRequestType	Тип запроса	1
bRequest	Код запроса	1
wValue	Параметр	2
wIndex	Индекс	2
wLength	Число байт для передачи	2

Информация о том, как заполнять поля конкретного запроса, предоставлена в 9-й главе [7].

Используя группы транзакций, необходимо присвоить устройству адрес, а потом перевести устройство в сконфигурированное состояние. Для этого необходимо получить данные об устройстве в целом, либо о его частях. Требуемая информация хранится в дескрипторах устройства. Полное описание структуры дескрипторов USB-устройства содержится в книге [8].

Согласно указаниям, данным в [7], была выполнена последовательность операций, после которой устройство получило адрес и перешло в сконфигурированное состояние. На рис. 6 представлена реализованная последовательность действий. Каждый элемент рис. 6 обозначает группу транзакций, реализующих конкретную операцию. Название элемента состоит из названия запроса, который присутствует в первой транзакции каждой группы, и из типа группы транзакции.

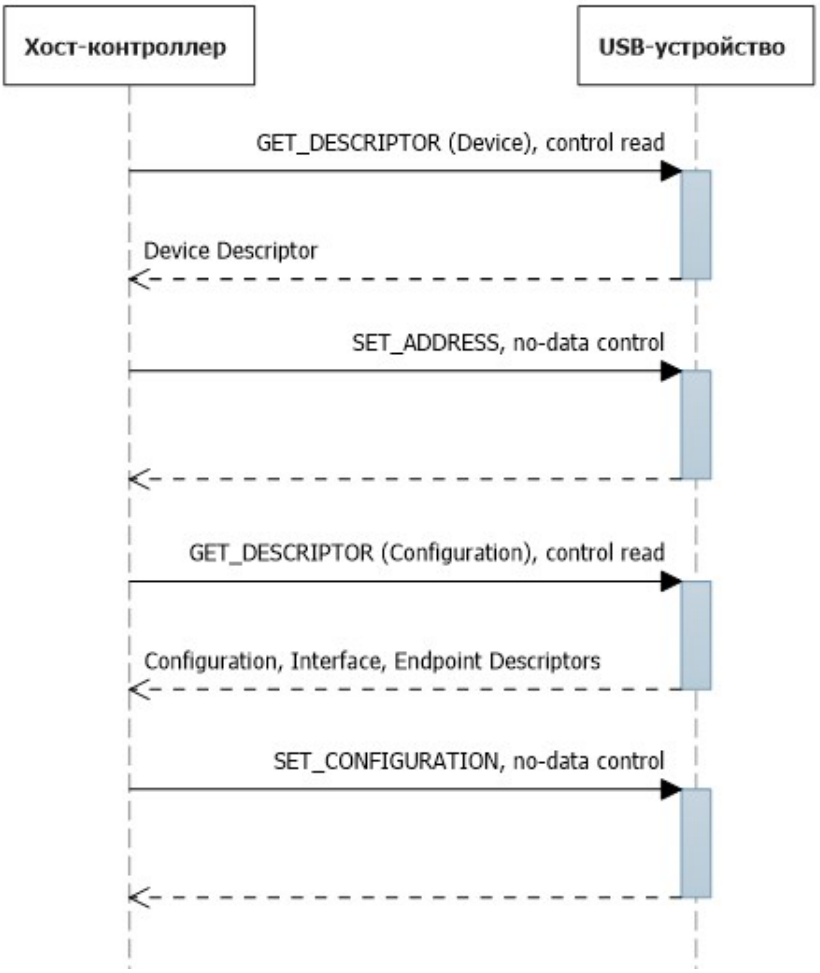

Рис. 6: Последовательность действий при конфигурировании USB-устройства

Проверить, является ли конфигурируемое устройство хабом, можно, используя информацию, полученную из дескриптора устройства. При работе с USB-хабом нужно выполнить дополнительные операции для каждого порта хаба, представленные на рис. 8 в приложении. Реализованные операции подают команды хабу для включения и сброса порта, к которому подключено устройство. После выполнения данных операций устройство, подключённое к порту, должно пройти общий для всех устройств этап конфигурирования.

5. Реализация поддержки режима передачи массивов данных для устройств хранения данных

Режим передачи массивов данных (bulk data transfer) используется для гарантированной доставки больших объёмов данных. В данном режиме работает большинство устройств хранения данных. С режимом передачи массивов данных на стороне устройства связаны конечные точки типа:

- OUT, через которые хост-контроллер передаёт данные USB-устройству;
- IN, через которые хост-контроллер принимает данные от USB-устройства.

Для того, чтобы начать работу с устройством в режиме передачи массивов данных требуется, чтобы устройство прошло общий этап конфигурирования. Далее необходимо проверить, поддерживает ли данное устройство работу в режиме передачи массивов данных. Подтверждающая информация содержится в дескрипторе интерфейса, полученном на этапе конфигурирования. Определяемые спецификацией [5] значения полей дескриптора интерфейса приведены в таблице 4.

Таблица 4: Значения полей дескриптора интерфейса для устройств хранения данных

Поле	Размер в байтах	Значение	Описание
bInterfaceClass	1	0x08	Код класса интерфейса
bInterfaceProtocol	1	0x50	Код протокола

Если значения полей дескриптора интерфейса совпали с требуемыми, то можно переходить на этап реализации поддержки режима передачи массивов данных. Спецификация [5] определяет структуры данных, которые должны использоваться при взаимодействии с устройством хранения данных:

- CBW (Command Block Wrapper) — структура, в которой передаётся команда устройству хранения данных (рис. 9 в приложении);
- CSW (Command Status Wrapper) — структура, в которой хост-контроллер получает информацию о статусе переданной команды (рис. 10 в приложении).

Команды, которые может выполнять устройство хранения данных, объявлены в спецификации [4]. Основной интерес представляют команды чтения и записи данных. В процессе реализации для чтения использовалась команда READ10, а для записи команда WRITE10.

Рассмотрим подробнее операцию чтения данных с устройства хранения данных. Команда READ10 предоставляет возможность читать данные секторами. Для этого

в команде требуется указать стартовый сектор, с которого будет проводиться чтение, а также количество секторов, которые требуется прочитать. Размер сектора, а также число доступных секторов для чтения для конкретного устойства можно узнать, воспользовавшись командой READ CAPACITY. После того, как параметры команды READ10 были определены, требуется передать её устройству на конечную точку типа OUT, используя структуру CBW. В случае, если устройство успешно обработает команду, запрашиваемые данные станут доступны на конечной точке типа IN.

На данном этапе работы требуется провести исследование. Проблема заключается в том, что в спецификациях [4] и [5] не указано, как именно должен быть организован процесс передачи данных. В случае, когда требуется прочитать данные с конечной точки типа IN, мы можем предложить несколько подходов к решению данной задачи, используя транспортные дескрипторы очереди, создаваемые драйвером хост-контроллера.

Во-первых, следует обратить внимание на транспортный дескриптор очереди, представленный на рис. 5. В стандартном случае транспортный дескриптор очереди использует один буфер для обработки данных. Размер буфера равен 4 КБ. Но спецификация [1] предоставляет возможность использовать пять буферов для одного транспортного дескриптора, поэтому суммарный объем обрабатываемых данных для одного транспортного дескриптора возрастает до 20 КБ. Во-вторых, важно учесть возможность создавать очердь из нескольких транспортных дескрипторов, которые будут обрабатываться последовательно.

Чтобы сравнить различные подходы к организации процесса передачи данных, были определены и реализованы следующие тестовые случаи:

- Последовательное чтение, реализованное с помощью команды READ10, запрашивающей 4 КБ данных:
    1. для чтения используется очередь из транспортных дескрипторов, каждый из которых обрабатывает 512 байтов данных;
    2. для чтения используется один транспортный дескриптор, обрабатывающий 4 КБ данных.
- Последовательное чтение, реализованное с помощью команды READ10, запрашивающей 20 КБ данных:
    1. для чтения используется очередь из транспортных дескрипторов, каждый из которых обрабатывает 512 байтов данных;
    2. для чтения используется один транспортный дескриптор, обрабатывающий 20 КБ данных.

- Последовательное чтение, реализованное с помощью команды READ10, запрашивающей 64 КБ данных:

    1. для чтения используется очередь из транспортных дескрипторов, каждый из которых обрабатывает 512 байтов данных;
    2. для чтения используется очередь из транспортных дескрипторов, каждый из которых обрабатывает 20 КБ данных.

Причина, по которой для тестовых случаев с очередью были выбраны транспортные дескрипторы, обрабатывающие 512 байтов данных, заключается в том, что для USB-флеш-накопителей максимальный размер пакета передачи составляет 512 байтов. Так как тестирование проводилось для USB-флеш-накопителей, то было бы разумно учесть эту особенность.

Также было сделано предположение, что в случае последовательного чтения, когда используется команда READ10, стоит запрашивать большее количество данных, чтобы сократить суммарное время, которое уходит на передачу команды устройству.

Чтобы отследить, что взаимодействие с устройством хранения данных организовано корректно, были проведены следующие проверки:

- проверка статусного поля каждого обработанного транспортного дескриптора;
- проверка статуса выполненной команды при получении структуры CSW;
- сравнение объёма запрашиваемых и полученных данных.

Используя побайтовое сравнение, была проведена проверка того, что полученные данные совпадают с запрашиваемыми.

6. Апробация

Для того, чтобы проверить, что реализация поддержки режима управляющей передачи данных проведена корректно, было принято решение создать утилиту для операционной системы MS-DOS, которая строит дерево подключенных к хост-контроллеру устройств, а также отображает информацию о USB-устройствах. Данная утилита подтверждает, что хост-контроллер был правильно инициализирован и что при помощи разработанного драйвера хост-контроллера удалось сконфигурировать подключенные USB-устройства.

При реализации поддержки режима передачи массивов данных для устройств хранения данных были предложены различные подходы к организации процесса передачи данных. Чтобы сравнить описанные подходы, потребовалось провести тестирование. В качестве тестового оборудования использовались распространенные USB-флеш-накопители. По результатам тестирования были сделаны выводы, которые помогли определить лучший подход к организации процесса передачи данных для собственной реализации.

6.1. Получение конфигурационной информации для USB-устройств

Информация, характеризующая возможности USB-устройства, определяется изготовителем и размещается в дескрипторах устройства. Данная информация используется в процессе конфигурирования USB-устройства, и поэтому связана с режимом управляющей передачи данных. Была создана утилита, которая строит дерево устройств, подключенных к хост-контроллеру, и отображает основные дескрипторы устройства на экране. В таблицах 10 и 11 в приложении представлены полученные значения дескриптора устройства и дескриптора интерфейса для USB-флеш-накопителя ADATA C905 8GB.

Чтобы проверить, что полученные дескрипторы USB-флеш-накопителя корректны, использовалась спецификация [7], где определены стандартные значения полей, которые являются одинаковыми для всех устройств. Например, значение типа дескриптора строго определено. Проверка значений полей, которые являются специфичными для конкретного устройтсва, например, идентификатор производителя, осуществлялась при помощи утилиты lsusb [7], доступной для Unix-подобных операционных систем. Результаты, полученные с помощью собственной утилиты, совпали с результатами, полученными утилитой lsusb.

[7]Утилита lsusb, отображающая информацию о подключенных USB-устройствах — URL: `http://linuxcommand.org/man_pages/lsusb8.html` (дата обращения: 10.05.2015)

6.2. Сравнение различных подходов к организации процесса передачи данных для USB-флеш-накопителей

Для сравнения различных подходов к организации процесса передачи данных использовались следующие USB-флеш-накопители:

1. ADATA C905 8GB (idVendor = 0x125F, idProduct = 0xC95A, USB 2.0), далее ADATA 2.0;
2. Silicon Power LuxMini 320 64GB (idVendor = 0x13FE, idProduct = 0x4200, USB 2.0), далее Silicon Power 2.0;
3. Transcend JetFlash 780 16GB (idVendor = 0x8564, idProduct = 0x1000, USB 3.0), далее Transcend 3.0.

Тестирование проводилось на ноутбуке HP Pavilion G6-1216 с процессором 2.1 GHz Intel Core i3-2310M, на операционной системе MS-DOS 6.22. Использовался хост-контроллер версии EHCI.

В рамках одного теста последовательно читалось 256 МБ данных с USB-флеш-накопителя. Для проведения теста требовалось определить составляющие:

1. USB-флеш-накопитель:
   - ADATA 2.0;
   - Silicon Power 2.0;
   - Transcend 3.0.
2. Размер одной передачи, определяемый командой READ10:
   - 4 КБ;
   - 20 КБ;
   - 64 КБ.
3. Подход к организации одной передачи:
   - с использованием очереди из транспортных дескрипторов, обрабатывающих 512 байтов данных (Очередь 512 Б);
   - с использованием одного транспортного дескриптора, обрабатывающего 4 КБ данных (Дескриптор 4 КБ);
   - с использованием одного транспортного дескриптора, обрабатывающего 20 КБ данных (Дескриптор 20 КБ);
   - с использованием очереди из транспортных дескрипторов, обрабатывающих 20 КБ данных (Очередь 20 КБ).

Для каждого теста вычислялось время работы. Суммарное количество данных, читаемых во время теста, фиксировано, поэтому возможно вычислить скорость чтения в МБ в секунду. Результирующее значение скорости вычислялось как среднее на основе результатов запусков одинаковых тестов.

Для тестового случая, когда размер одной передачи равнялся 4 КБ, были получены результаты, представленные в таблице 5. В этом тесте подход с очередью задействовал 8 транспортных дескрипторов для обработки данных одной передачи.

Таблица 5: Результаты тестирования скорости последовательного чтения с USB-флеш-накопителей для передачи размером 4 КБ (МБ/с)

USB-флеш-накопитель	Очередь 512 Б	Дескриптор 4 КБ
ADATA 2.0	11,91	12,00
Silicon Power 2.0	10,11	11,84
Transcend 3.0	22,28	22,54

Отметим, что представленные результаты близки, учитывая, что стандартное отклонение менее 0,20 МБ/с. Вывод о том, какой подход к организации передачи лучше, сделать сложно.

Рассмотрим тестовый случай, когда размер одной передачи равнялся 20 КБ (таблица 6), и подход с очередью использовал 40 транспортных дескрипторов.

Таблица 6: Результаты тестирования скорости последовательного чтения с USB-флеш-накопителей для передачи размером 20 КБ (МБ/с)

USB-флеш-накопитель	Очередь 512 Б	Дескриптор 20 КБ
ADATA 2.0	21,32	23,52
Silicon Power 2.0	19,27	21,11
Transcend 3.0	29,43	32,45

Если сравнить первый тестовый случай и второй, то заметен значительный прирост в скорости. Отдельно стоит отметить, что подход с транспортным дескриптором, обрабатывающим 20 КБ данных, показал лучший результат. Поэтому было принято решение протестировать подход с очередью из транспортных дескрипторов, обрабатывающих 20 КБ данных. Но во время тестирования данного подхода появились ошибки передачи, связанные с некорректной работой устройства, при этом часть запрашиваемых данных была потеряна. Предполагается, что устройство не успевало обрабатывать запросы на чтение, отправляемые хост-контроллером.

В случае, когда размер одной передачи равнялся 64 КБ и подход с очередью использовал 128 транспортных дескрипторов, были получены результаты, представленные в таблице 7.

Таблица 7: Результаты тестирования скорости последовательного чтения с USB-флеш-накопителей для передачи размером 64 КБ (МБ/с)

USB-флеш-накопитель	Очередь 512 Б
ADATA 2.0	29, 78
Silicon Power 2.0	24, 59
Transcend 3.0	34, 12

Данный результат является лучшим среди всех проведённых тестов, поэтому подход к организации передачи с использованием очереди из транспортных дескрипторов, обрабатывающих 512 байтов данных, можно считать оптимальным для собственной реализации.

Чтобы оценить полученный результат, было проведено тестирование скорости чтения и записи USB-флеш-накопителей при помощи специализированных средств:

- IOmeter [8];
- HD Speed [9];
- FlashBenchmark [10].

Описанные средства запускались на операционной системе Windows 7 Professional (32-bit), и для каждого инструмента указывалось, что размер одной передачи должен быть равен 64 КБ. Собственная реализация запускалась на MS-DOS 6.22, использовался подход с очередью из 128 транспортных дескрипторов. Результаты измерения скорости последовательного чтения представлены в таблице 8, последовательной записи в таблице 9.

Таблица 8: Результаты измерения скорости последовательного чтения с USB-флеш-накопителей для передачи размером 64 КБ (МБ/с)

USB-флеш-накопитель	Очередь 512 Б	IOmeter	HD Speed	FlashBenchmark
ADATA 2.0	29, 78	28, 06	27, 30	24, 14
Silicon Power 2.0	24, 59	22, 74	20, 50	18, 69
Transcend 3.0	34, 12	32, 74	30, 08	25, 33

По полученным результатам можно сделать следующие выводы:

[8]IOmeter — инструмент для анализа производительности накопителей — URL: `http://www.iometer.org/` (дата обращения: 20.05.2015)

[9]HD Speed — утилита для измерения скорости чтения и записи жестких дисков и флеш-накопителей — URL:`http://www.steelbytes.com/?mid=20/` (дата обращения: 22.05.2015)

[10]FlashBenchmark — приложение, измеряющее скорость чтения и записи флеш-накопителей — URL: `http://usbflashspeed.com/` (дата обращения: 20.05.2015)

Таблица 9: Результаты измерения скорости последовательной записи на USB-флеш-накопители для передачи размером 64 КБ (МБ/с)

USB-флеш-накопитель	Очередь 512 Б	IOmeter	HD Speed	FlashBenchmark
ADATA 2.0	8, 78	12, 74	9, 40	4, 94
Silicon Power 2.0	8, 95	13, 01	11, 32	4, 74
Transcend 3.0	26, 19	25, 05	24, 50	6, 62

- реализованный подход для последовательного чтения с USB-флеш-накопителя показал отличные результаты;
- в случае последовательной записи на USB-флеш-накопитель, скорее всего, можно улучшить используемый подход.

Отдельно стоит отметить тот факт, что отставание на последовательной записи получено только для USB-флеш-накопителей, спроектированных по стандарту USB 2.0. Предполагается, что при работе с USB 2.0 устройствами, есть некоторая особенность, которая помогает увеличить скорость записи. Скорее всего, для USB 3.0 устройств реализация данной особенности не потребовалась, так как скорость работы USB 3.0 устройств и без этого значительно выше.

Заключение

В рамках данной работы были достигнуть следующие результаты.

1. Реализовано взаимодействие с хост-контроллером версии EHCI: проведена идентификация хост-контроллера, выполнена реализация модуля, обеспечивающего доступ к регистрам хост-контроллера, проведена инициализация хост-контроллера, реализован драйвер хост-контроллера.
2. Выполнена реализация поддержки режима управляющей передачи данных.
3. Выполнена реализация поддержки режима передачи массивов данных для устройств хранения данных и предложены различные подходы к организации процесса передачи данных.
4. Проведена апробация реализованной функциональности: разработана утилита, отображающая конфигурационную информацию для USB-устройств, выполнено сравнение подходов к организации процесса передачи данных для USB-флеш-накопителей.

Список литературы

[1] Intel Corporation. Enhanced Host Controller Interface Specification for Universal Serial Bus. — 2002. — URL: http://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/ehci-specification-for-usb.pdf (дата обращения: 8.02.2015).

[2] Intel Corporation. eXtensible Host Controller Interface for Universal Serial Bus. Revision 1.1. — 2013. — URL: http://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/extensible-host-controler-interface-usb-xhci.pdf (дата обращения: 25.05.2015).

[3] PCI Special Interest Group. PCI Local Bus Specification. Revision 2.3. — 2002.

[4] Seagate Technology. SCSI Commands Reference Manual. Rev. A. — 2006. — URL: http://www.seagate.com/staticfiles/support/disc/manuals/scsi/100293068a.pdf (дата обращения: 2.05.2015).

[5] USB Implementers Forum. Universal Serial Bus Mass Storage Class. Bulk-Only Transport. Revision 1.0. — 1999. — URL: http://www.usb.org/developers/docs/devclass_docs/usbmassbulk_10.pdf (дата обращения: 22.04.2015).

[6] Hewlett-Packard Company, Intel Corporation, Microsoft Corporation и др. Universal Serial 3.0 Bus Specification. Revision 1.0. — 2008. — URL: http://www.usb.org/developers/docs/documents_archive/ (дата обращения: 15.03.2015).

[7] Compaq, Hewlett-Packard, Intel и др. Universal Serial Bus Specification. Revision 2.0. — 2000. — URL: http://www.usb.org/developers/docs/usb20_docs/ (дата обращения: 10.03.2015).

[8] Агуров П.В. Интерфейс USB. Практика использования и программирования. — БХВ-Петербург, 2004.

[9] Кулаков В. Программирвоание на аппаратном уровне: специальный справочник. 2-е издание. — Питер, 2003.

[10] Семенко А. Ассемблер для 32-разрядных процессоров: от i386 до Pentium 4. Защищённый режим. — 2015. — URL: http://sasm.narod.ru/docs/pm/pm_in/chap_4.htm (дата обращения: 18.03.2015).

Приложения

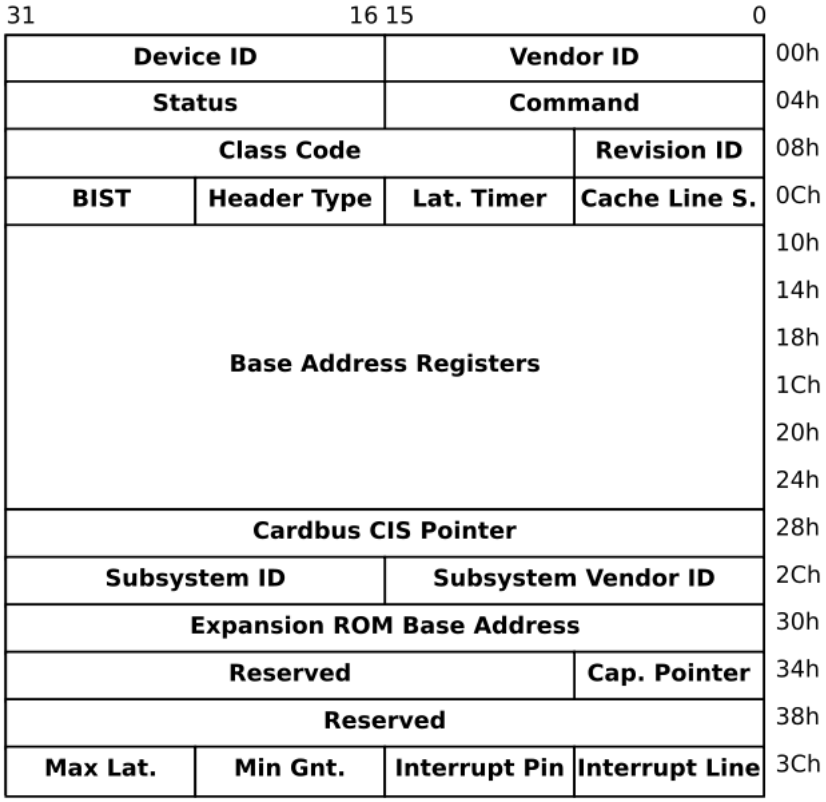

Рис. 7: Структура конфигурационного пространства PCI (рис. из [3])

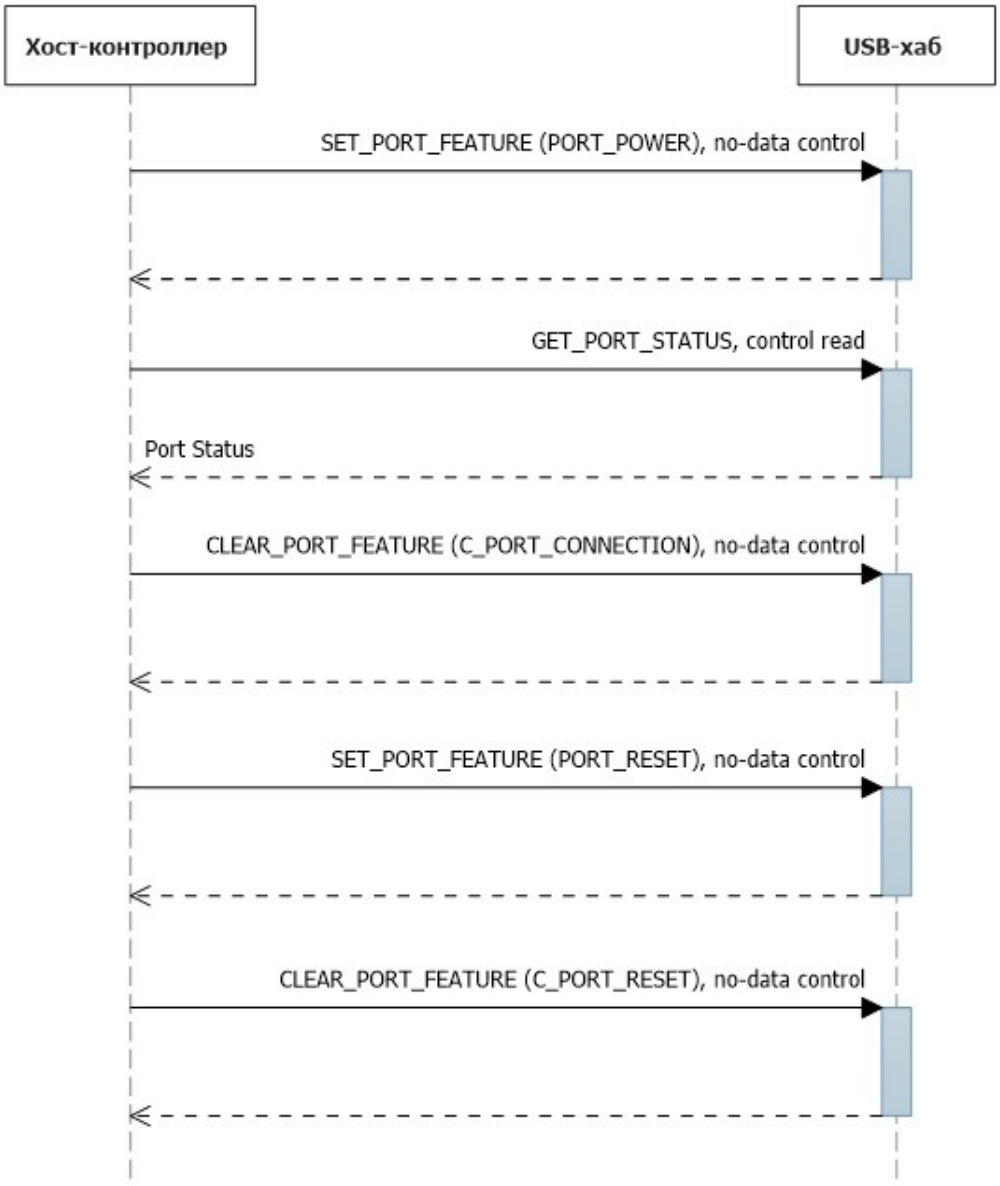

Рис. 8: Последовательность действий при конфигурировании порта USB-хаба

bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCBWSignature*							
4-7	*dCBWTag*							
8-11 (08h-0Bh)	*dCBWDataTransferLength*							
12 (0Ch)	*bmCBWFlags*							
13 (0Dh)	Reserved (0)				*bCBWLUN*			
14 (0Eh)	Reserved (0)			*bCBWCBLength*				
15-30 (0Fh-1Eh)	*CBWCB*							

Рис. 9: Структура Command Block Wrapper (рис. из [5])

bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCSWSignature*							
4-7	*dCSWTag*							
8-11 (8-Bh)	*dCSWDataResidue*							
12 (Ch)	*bCSWStatus*							

Рис. 10: Структура Command Status Wrapper (рис. из [5])

Таблица 10: Дескриптор устройства USB-флеш-накопителя ADATA C905 8GB

Поле	Размер в байтах	Значение	Описание
bLength	1	0x12	Размер дескриптора
bDescriptorType	1	0x01	Тип дескриптора
bcdUSB	2	0x0200	Версия спецификации USB
bDeviceClass	1	0x00	Код класса
bDeviceSubClass	1	0x00	Код подкласса
bDeviceProtocol	1	0x00	Код протокола
bMaxPacketSize	1	0x40	Максимальный размер пакета для нулевой конечной точки
idVendor	2	0x125F	Идентификатор производителя
idProduct	2	0xC95A	Идентификатор продукта
bcdDevice	2	0x0100	Версия устройства
iManufacturer	1	0x01	Индекс дескриптора строки производителя
iProduct	1	0x02	Индекс дескриптора строки продукта
iSerialNumber	1	0x03	Индекс дескриптора строки серийного номера
bNumConfigurations	1	0x01	Количесво конфигураций

Таблица 11: Дескриптор интерфейса USB-флеш-накопителя ADATA C905 8GB

Поле	Размер в байтах	Значение	Описание
bLength	1	0x09	Размер дескриптора
bDescriptorType	1	0x04	Тип дескриптора
bInterfaceNumber	1	0x00	Номер интерфейса
bAlternateSetting	1	0x00	Альтернативный номер интерфейса
bNumEndpoints	1	0x02	Число конечных точек
bInterfaceClass	1	0x08	Код класса интерфейса
bInterfaceSubClass	1	0x06	Код подкласса интерфейса
bInterfaceProtocol	1	0x50	Код протокола
iInterface	1	0x00	Индекс дескриптора строки интерфейса