USB для программистов микроконтроллеров.

Предисловие

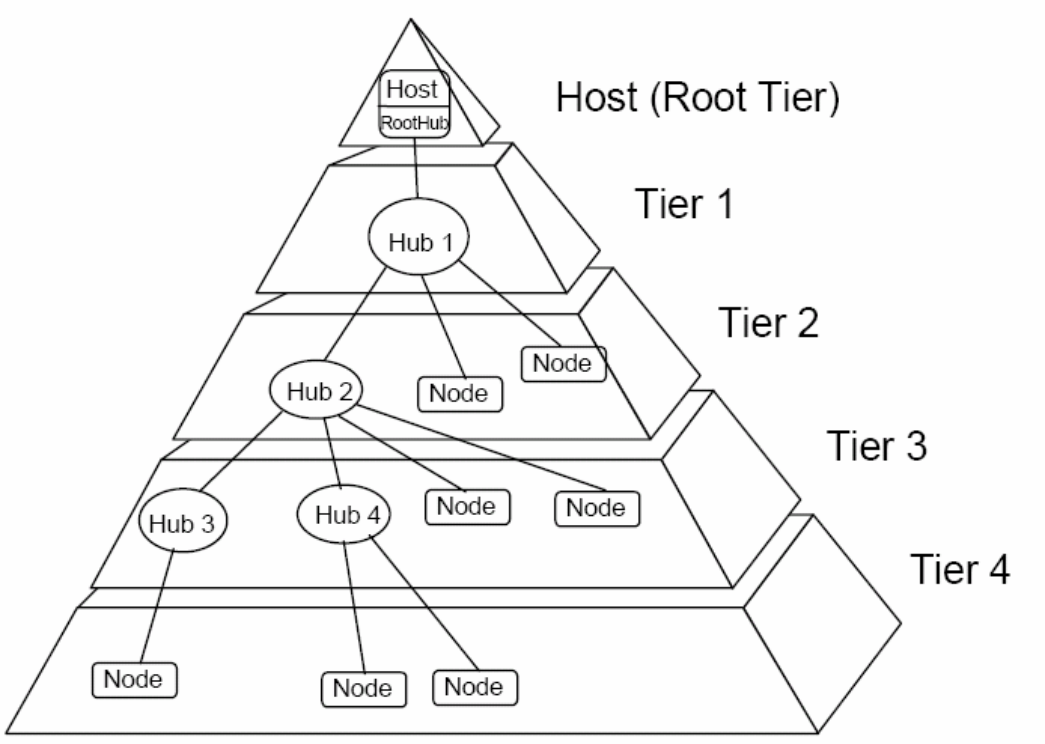
Цель данной статьи помочь программистам микроконтроллеров начать разрабатывать устройства с USB интерфейсом или улучшить свои знания.

Даная статья не претендует на полноту освещения всех вопросов программирования и работы с USB. В ней, надеюсь, простым языком будет рассказано понемногу обо всех аспектах USB.

Введение

В начале, как обычно расшифровка термина, USB – Universal Serial Bus – универсальная последовательная шина. Действительно устройства можно подключать самые разнообразные, данные передаются последовательно, а вот физическое подключение на шину не похоже. Шина это совокупность параллельных проводников, к которым подсоединяются несколько устройств, например ISA или PCI.

В [1] приводиться такой рисунок топологии:



По стандарту такое соединение называется многоярусная звезда. С точки зрения программиста на PC все устройства присоединенные к PC как к шине. Управление и идет через шинный драйвер USB, как через шинный драйвер PCI или ISA.

Взаимодействие между PC и конечным устройством скрыто с помощью драйверов и аппаратуры. В простейшем случае можно выразить таким рисунком из [1]

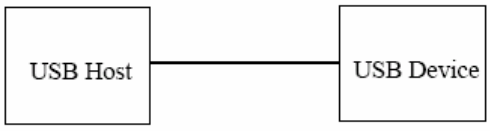


Figure 5-1. Simple USB Host/Device View

PC кабель и устройство – всего три компонента, и две программы одна в устройстве вторая в PC.

Как видно из двух рисунков приведенных выше одно и тоже присоединенное устройство названо по разному **Node** и **Device**. В [1] можно также встретить наименование устройств как **function** и **peripheral device**,

В этой статье будет использован термин **устройство** и рассмотрен пример, когда к PC устройство подключено напрямую. Так как принцип взаимодействия между программой на PC и устройством (программой в устройстве) не зависит от того, есть ли другие устройства, а также напрямую ли подключено устройство или через хаб(hub).

1 Упрощенная программная модель обмена по USB

Обменом данных по шине управляет хост (host). Хост определяет направление передачи и устройство, которое должно принять или передать данные.

Хост формирует и передает пакет, в котором указан адрес устройства направление передачи, тип пакета и т.д.

Аппаратура хоста и устройства формирует и контролирует CRC, занимается разрешением нештатных ситуаций, например устройство не готово принять данные или произошла ошибка обмена. Программисту в конечном итоге приходиться работать только с корректно полученными пакетами.

Получателем пакетов является не устройство, а конечная точка назначения называемая по стандарту **endpoint**.

Для корректной работы устройства требуется минимум две конечные точки. Они называются **Setup In** и **Setup Out** и имеют нулевые номера.

Для определения характеристик устройства и его конфигурирования используются специальные Setup пакеты, которые приходят от хоста в **Setup Out**, а отправляются в хост контроллером устройства через **Setup In**.

Эти endpoints объединяют в одну логическую абстракцию **Control pipe** или **Default Pipe**. Эта абстракция имеет также синонимы **Message Pipe** и **Setup Pipe**. Надеюсь, в следующем разделе станет понятно, почему столько синонимов.

Дословно pipe переводится как труба. Наиболее близким по смыслу, наверное, будет перевод “канал”. Управления, по умолчанию, сообщений или настройки соответственно.

Другие **endpoint(s)**, также могут быть объединены в pipe(s), для облегчения восприятия архитектуры и принципов работы.

Pipes может быть как однонаправленные – например USB мышь, которая передает информацию о величине перемещения по координатам и состояние клавиш.

Резюме по этому разделу может быть следующее:

Направление передачи данных задается хостом;

Данные передаются по каналам связи, называемыми pipes;

Контроль целостности информации, и адресацию (доставку пакетов) производит аппаратура USB.

2 Как передается информация по кабелю USB.

Теперь, когда с “нудным” и абстрактным теоретическим вступлением закончено, перейдем к физическим принципам передачи.

В [1] приведен следующий рисунок кабеля USB:

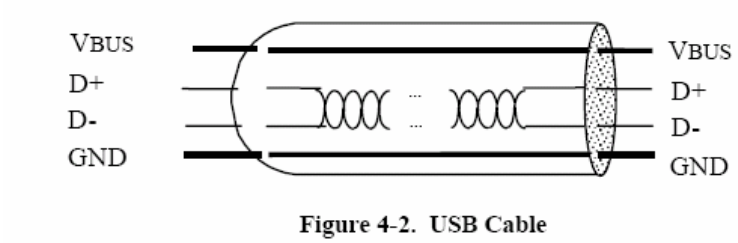


Figure 4-2. USB Cable

Наименование проводов сверху вниз.
Vbus – напряжение питания шины.

D+ линия данных условно названа положительной.
D- линия данных условно названа отрицательной.
GND – общий провод, в англоязычной литературе часто называют return wire – обратный провод.

Назначение проводов следующее:
Vbus – напряжение питания для устройств без собственного источника питания – мышки, клавиатуры, флеш карты и т.д. Номинальное значение плюс 5 вольт. В общем случае можно рассчитывать получить ток около 0,5 ампера после конфигурации. При подключении и конфигурировании устройства 0,1 ампера.

GND – ток от источника питания, пройдя через нагрузку, должен вернутся обратно к источнику через обратный (возвратный провод), как часто принято называть в англоязычной литературе.

D+ и **D–** Информационные двунаправленные линии данных.

Внимание!
Эти линии для передачи информации всегда используются вместе. По USB в один конкретный момент информация передается в одном направлении. По терминологии связистов такой обмен данными называется полудуплексный.

Напряжение на этих линиях, измеренное относительно GND указывает состояние на шине. Теперь обратный провод удобнее назвать общим, так как относительно него измеряются потенциалы на линиях D+ и D–.

В стандарте [1] есть таблица **7–1. Signaling Levels**, где перечислены все возможные состояния линий.

Упрощенно можно таблицу трактовать так:

Если на линиях D+ и D– часто изменяются уровни, значит, идет передача данных. Частота изменения, естественно, равна скорости передачи.
Если напряжение на линии D+ больше D– то передается логическая единица. Если напряжение на линии D+ меньше D–, то передается логический ноль.

Если на линиях определенное время отсутствуют изменения, значит, канал связи находится в состоянии покоя (Idle), сброса (Reset) и т.д.

Естественно сигналы идут не только по кабелю, но и по разъемам и проводникам печатных плат и заходят в интегральные схемы приемопередатчиков USB.

3 Что происходит при подключении устройства.

При подключении устройства к хосту, получается следующая схема:

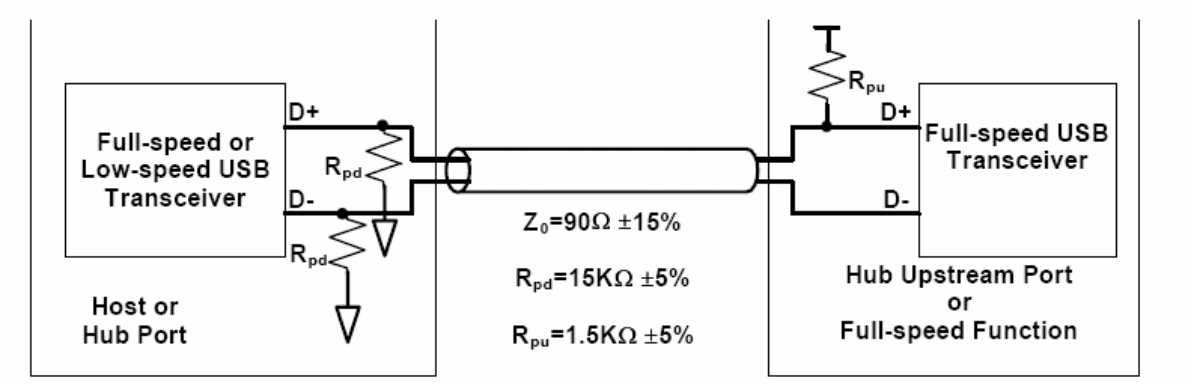


Figure 7-10. Full-speed Device Cable and Resistor Connections

Если мысленно отсоединить кабель, то на линиях хоста будет низкое напряжение и согласно таблице 7-1. Signaling Levels из [1] линия находится в состоянии сброса, поскольку резисторы Rpd “притягивают” линии к общему проводу.

При подсоединении устройства, линия D+ оказывается, притянута к питанию от 3 до 3,6 вольт через резистор 1,5килоома. Таким образом, хост детектирует присутствие устройства работающего в режиме **Full Speed** (**FS**, скорость обмена информацией 12мбит/с).

Если устройство работает в режиме **Low Speed** (**LS**, скорость обмена информацией 1,5мбит/с), то резистор 1,5килоом должен присоединяться к линии D-.

При разработке стандарта USB ревизии 2.0[2] была добавлена возможность работы со скоростью 480 мбит/с, и режим обмена получил название **High Speed (HS)**. В этом режиме детектирование и конфигурирование происходит как в режиме FS, но после конфигурирования резистор должен отключатся, для выравнивания импеданса линии.

Внимание!

Стандарт 2.0 объединил ранее утвержденные скорости обмена FS и LS, была *добавлена скорость HS. Кроме этого были добавлены изменения в протокол обмена для ускорения передачи информации и управления шиной. Устройства, работающие со скоростью LS и FS и поддерживающие эти изменения, могут маркироваться USB 2.0. Но скорость обмена у них естественно не HS.*

Вопросы детектирования и конфигурирования со стороны PC, берет на себя операционная система. Программисту контроллера устройства нужно самостоятельно решить, как будет происходить процесс присоединения, и обеспечить правильное функционирование.

Если резистор Rpu будет постоянно запитан от Vbus и устройство имеет собственное питание, то может возникнуть ситуация, когда питание на компьютер будет подано намного раньше, чем на устройство. При этом ОС попытается сконфигурировать устройство и после нескольких неуспешных попыток заблокирует работу устройства.

По этому желательно предусмотреть возможность управлять процессом присоединения и отсоединения резистора.

Внимание!

В случае сброса устройства сторожевым таймером или монитором питания, устройство не сможет просигнализировать хост о том, что конфигурация заданная хостом, пропала и требуется переинициализация.

Возможно также, по логике работы устройства понадобится детектирование включения/отключения хоста по напряжению на линии Vbus.

4. В каком состоянии должен находится контроллер устройства перед подключением к шине USB.

Перед подсоединением к шине USB (перед присоединением резистора Rpu), Контроллер должен иметь адрес 0. В некоторых контроллерах USB, адрес может быть просто запрещен сбросом бита в соответствующем регистре.

Все конечные точки кроме пары Endpoint 0 – Setup In и Setup Out запрещены. Буферы и флаги прерываний очищены.

В зависимости от того, каким образом решено реагировать на события на шине должны быть разрешены/запрещены прерывания называемые Bus Reset (или End Bus Reset) и по приходу Setup пакета.

После этого можно присоединяться к шине USB.
Желательно проанализировать схему, чтобы убедиться, при неактивном контроллере (сбросе контроллера) Rpu отсоединяется от проводов D+ или D-.

В противном случае нужно добавит функцию, которая при старте контроллера принудительно отсоединяет резистор от USB на 1..2 секунды.

5. Конфигурирование устройства.

После детектирования устройства хост производит сброс шины(Bus Reset). После выдержки паузы хост снимает состояние сброс шины и производит конфигурирования устройств на шине.

В литературе встречается термин “перечисление” или калька с английского языка enumeration – енумерация.

Процесс конфигурирования состоит в посылке стандартизированных запросов устройству и получение стандартизированных ответов от него. Под словом стандартизированный подразумевается жестко заданные стандартом USB длины посылок назначения и порядок полей в посылках.

Эти посылки называются по стандарту **Message**, и обмен ими идет, как было сказано выше через Message Pipe. В литературе также распространены и синонимы, например Default Control Pipe..

Конечная цель конфигурирования, получить информацию об устройстве, настроить режим его работы, если устройство имеет несколько конфигураций. Если устройство не принадлежит ни одному стандартному классу (мышь, клавиатура, и т.д.) также определить код производителя и устройства, чтобы корректно установить драйвера.

Информация об устройстве позволяет хосту (драйверу хоста) корректно поделить пропускную способность шины так, чтобы удовлетворить потребности всех устройств. О распределении пропускной способности шины чуть ниже.

Через сообщения также можно получить информацию о токе потребления устройством в сконфигурированном режиме. Если устройство подключается к ноутбуку или к хабу, без собственного питания, возможна ситуация, когда из за недостатка тока в системе драйвер (программный) со стороны хоста отключит устройство.

В [1] есть рисунок, показывающий как устройство переходит из состояния в состояние **Figure 9-1. Device State Diagram.**

Стандарт различает состояние **Attached** и **Powered.** Устройства, которые имеют питания от USB, при присоединении сразу же запитываются, их состояние называется Powered. Принтеры и фотокамеры имеют собственное питание, по этому они подсоединены, но не запитаны их состояние называется Attached. После нажатия клавиши включения питания эти устройства тоже переходят в состояние Powered.

Если исключить из рассмотрения несущественное состояние Suspend (приостановлено), то остаются три состояния, через которые драйвер хоста должен провести(сконфигурировать) устройство, чтобы с ним можно было работать пользовательской программе на PC:

Default–Address–Configured.

Работу по корректной обработке сообщений со стороны хоста выполняет ОС, со стороны устройства эту задачу должен выполнить программист устройства.

6 Обработка конфигурационных запросов со стороны устройства.

Контроллер устройства может быть выполнен отдельными процессором и внешним интерфейсом шины USB, например PDIUSBD12 или USBN9604. На данный момент много микроконтроллеров имеют встроенное ядро (core, engine) USB и интегрированные приемопередатчики (bus driver).

Далее по тексту, фразы, например “чтение регистра прерывания” нужно понимать как чтение регистра во внешнем интерфейсе или во внутреннем ядре, в зависимости от конструкции устройства.

Когда контроллер устройства произвел собственную настройку как описано в разделе 4 этой статьи, его состояние соответствует Default.

После подсоединения резистора на информационную шину (D+ или D– в зависимости от скорости обмена) хост начинает процесс конфигурации.

В регистре состояния (прерывания) USB начинают взводиться флажки о изменении состоянии шины и приходе Setup пакетов.

Внимание!

Программист, разрабатывающий обработчик прерываний от USB, не может предположить, какие сообщения будет присылать ОС, какие изменения или ошибки могут произойти на шине. Кроме того, разработчики ОС имеют полное право посылать некорректные запросы для проверки правильного реагирования устройством на такие запросы.

В очень хорошей обзорной статье [3], всего в 30 страницах дан обзор стандарта USB 2.0, который в свою очередь занимает 650 страниц.
На последней странице описан типовой процесс конфигурирования, приведу его не с дословным переводом, а по смыслу.

1. The host or hub detects the connection of a new device via the device's pull up resistors on the data pair. The host waits for at least 100ms allowing for the plug to be inserted fully and for power
to stabilize on the device.

После детектирования устройства, хост делает задержку минимум на 100 миллисекунд, для стабилизации напряжения питания на устройстве

2. Host issues a reset placing the device is the default state. The device may now respond to the default address zero.

Хост производит сброс устройства, что приводит устройство в состояние по умолчанию. Устройство теперь должно реагировать на сообщения с адресом по умолчанию равным 0.

3. The MS Windows host asks for the first 64 bytes of the Device Descriptor.

Драйвер хоста MS Windows запрашивает первые 64 байта дескриптора устройства.

4. After receiving the first 8 bytes of the Device Descriptor, it immediately issues another bus reset.

После приема первых 8 байт дескриптора устройства хост генерирует еще один шины сброс.

Наверное, правильнее было – после приема первой порции ответа размером в конечную точку. Размер конечной точки для обмена Setup пакетами стандартом оговаривается следующими размерами 8, 16 , 32, 64 для FS устройств. По моему мнению, аппаратура хоста не сможет определить, корректна ли принятая информация, если пакет будет разорван.

5. The host now issues a Set Address command, placing the device in the addressed state.

Хост посылает команду Set Address, переводя устройство в адресуемое состояние (Address), состояние в котором устройство имеет адрес отличный от 0.

6. The host asks for the entire 18 bytes of the Device Descriptor.
Хост запрашивает все 18 байт Device Descriptor.

7. It then asks for 9 bytes of the Configuration Descriptor to determine the overall size.
8. The host asks for 255 bytes of the Configuration Descriptor.

Хост запрашивает 9 байт Configuration Descriptor (дескриптора/описателя конфигурации), для определения его размера, а затем 255 байт Configuration Descriptor.

9. Host asks for any String Descriptors if they were specified.

Хост запрашивает описатели в виде строк, если они указаны. Индексы этих строк указываются в дескрипторе конфигурации.

At the end of Step 9, Windows will ask for a driver for your device. It is then common to see it request all the descriptors again before it issues a Set Configuration request.

После шага 9 Windows выводит приглашение указать путь к драйверу вашего устройства. Это общий взгляд на конфигурирование устройства.

Там же[3], обращается внимание на то, что действия Windows на шаге 4 приводит в недоумение начинающих программистов USB устройств.

Даже при правильной структуре отсылаемых дескрипторов Windows реагирует как на ошибочную структуру.

Теперь еще нужно обратить внимание, что обычно устройства имеют Setup endpoint размером 8 или 16 бит. Таким образом, при отсылке информации в PC возникают еще события – окончания передачи очередной порции данных через Setup Endpoint.

Резюмируя этот раздел можно предложить следующий порядок обработки событий на шине USB со стороны контроллера.

1. Есть ли сброс шины?
1.1 Сбросить адрес устройства в 0.
1.2 Очистить Setup endpoints как на передачу, так и на прием.
1.3 Запретить прием/передачу данных в остальных endpoints. Снять все флаги других событий, окончание передачи очередной порции данных через Setup Endpoint – обязательно.
1.4 Обработка событий завершена.

2. Пришел ли Setup пакет?
2.1 Провести действия от 1.1 до 1.3.
2.2 Определить тип запроса и сформировать ответ через Setup IN.
2.3 В зависимости от соотношения длины Setup IN, длины дескриптора и длины запроса запомнить длину остатка и указатель, откуда выводить следующую порцию данных, а также взвести программный флаг о том, что идет передача дескриптора. Если длина ответа меньше длины Setup IN, то флаг нужно сбросить.
2.4 Обработка событий завершена.

3. Окончание передачи очередной порции данных через Setup Endpoint?
3.1 Очистить флаг передачи.
3.2 Здесь возможны такие варианты дальнейшей обработки:
3.2.1 Если программный флаг передачи сброшен, то просто перейти к рассмотрению следующих событий.
3.2.2 Если текущий остаток кратен Setup Endpoint, то длину для последующей передачи установить в 0.
3.2.3 Остался вариант, когда остаток ответа, по-прежнему больше Setup Endpoint. Нужно сформировать следующую часть данных и запомнить параметры для последующей передачи как в 2.3.

На некоторые запросы требуется отвечать нулевым пакетом или пакетом длиной короче, чем Setup Endpoint. При этом достаточно после отправки только сбрасывать программный флаг передачи.

Такая сложная манипуляция требуется стандартом USB для определения конца передачи данных. Хорошие диаграммы как реагировать на события есть в [4]

4. Рассмотреть все следующие события.

Внимание!

На все запросы, которые не обрабатываются программным обеспечением контроллера, устройство нужно отвечать переводом Setup Endpoints в состояние **STALL** (останов). При последующем обращении, хост получит аппаратный ответ STALL, а драйвер шині на PC должна проанализировать эту ошибку и больше не передавать таких запросов или произвести сброс и повторное перечисление устройств на шине. Выход из такого состояния, только по приему нового Setup пакета обеспечивается

аппаратурой USB для Setup Endpoints, для других Endpoints после приема пакета ***Set Configuration***.

При правильной манипуляции с корректными дескрипторами устройство получит запрос Set Adress, и после корректной установки адреса его состояние будет называтся Adressed. После корректной установки драйверов, устройству будет послан запрос ***Set Configuration.*** Это запрос имеет параметр. Нулевое значение – запретить конфигурацию, единичное значение – разрешить конфигурацию.

При разрешении конфигурации хостом, ПО в контроллере должно разрешить работу всех конечных точек устройства, что соответствует состоянию устройства – Configured. При запрещении конфигурации, соответственно запретить работу, такое состояние называется Address. В [4] есть подробные диаграммы, как отвечать на запросы хоста.

7 Interrupt, Bulk, Isochronous.

Шина USB предназначена для замены разнообразных устройств. Клавиатура и мышка, например, требуют, чтобы их данные передавались в компьютер с небольшой задержкой. Так как большая задержка будет заметна пользователю и будет его раздражать. Потеря данных при обмене, конечно же, не желательна. Для таких устройств, требующих передачи небольших объемов данных с гарантированной задержкой программист может (или обязан) назначить конечную точку на обмен **Interrupt**. Для этого программисту нужно сделать два действия:

1. В соответствующий регистр управления USB устройства записать корректное значение бит;
2. В дескрипторе конфигурации объявить конечную точку как Interrupt и назначить время опроса.

Так как устройство не может вызвать прерывание в классическом смысле, то хост с периодичностью заданной в дескрипторе конфигурации будет запрашивать данные из указанной конечной точки.

Если при обмене с такой конечной точкой произойдет сбой, то драйвер шины обязан повторить обмен.

Если проектируемое устройство предполагает большой объем данных и целостность их важна, то нужно использовать тип обмена **Bulk.**

Для этого нужно также выполнить два действия, как и для Interrupt обмена.

Недостаток Bulk в том, что время доставки не гарантировано. В случае если требуется передать большой объем с высокой скоростью нужно предусмотреть буферирование. А также исключить другие устройства с шины USB. Естественно, что исключение устройств из шины, возможно только для проведения экспериментов или проверочных действий. В реальной конфигурации компьютера такие действия будут раздражать пользователя.

Осталось рассмотреть режим передачи **Isochronous**. Этот режим используют в тех случаях, когда поздно доставленные данные такие же бесполезные, как и испорченные. Типичными устройствами являются аудио и видео устройства.

При ошибке обмена(NAK) данные повторно не передаются.

Кроме выполнения стандартных двух действий по настройке конечной точки, нужно еще согласование скорости обмена. Есть несколько вариантов согласования о них можно почитать, например в [5]

Обычно в контроллере (USB engine) есть возможность настроить конечные точки на разнообразные режимы. Для передачи больших объемов в режиме Bulk нужно использовать конечные точки с двойным буферированием и с наибольшим размером.

8 Как выбрать драйвер на PC?

Теперь можно перейти к вопросу о том, как реализовать доступ из пользовательской программы к устройству и получение данных из устройства.

Есть два подхода к решению этого вопроса.

1. Использовать стандартные предопределенные типы устройств.
2. Написать свой драйвер.

Для предопредеоленных устройств опередедены коды:
http://www.usb.org/developers/defined_class/
Эти коды должны быть указаны в дескрипторе устройства.
Для большинства из них, под Windows есть стандартные (классовые) драйвера, да и под Linux очевидно тоже.

Таким образом, программисту нужно выбрать какой-нибудь, наиболее подходящий из них, изучить принцип взаимодействия с классовым драйвером и реализовать взаимодействие в контроллеры в соответствии с требованиями и протоколами Microsoft Windows.

В сети Интернет достаточно много примеров реализации HID, CDC, Mass Storage устройств для микроконтроллеров.

Такой подход позволяет уйти от изучения среды проектирования драйверов для Windows(DDK), но естественно обладает меньшей гибкостью.

Для получения большей гибкости можно использовать драйвера из примеров DDK или взять наиболее подходящий из них и переделывать под свои задачи.

Некоторые примеры можно использовать без переделки.
В Windows DDK есть пример WDM драйвера BULKUSB. Этот драйвер удобен для обмена данных с микроконтроллерами. Так как гарантирует целостность данных.
Современные Windows DDK (для XP и NT2003) не требуют среды проектирования типа Visual C++.
Достаточно DDK. Последовательность действий примерно такая

1) Устанавить DDK
2) В ПУСК\Программы\Developments Kit\NT2003\
   запускаете bat файл Free Build Enviroment.
   Или другой каталог/DDK
3)В командной строке переходите в каталог, где установлен DDK.
Можно перейти с помощью Norton/Volcov commander, набираете "build -ce"
4)Ожидаете около 30 минут, пока скомпилируются все примеры и библиотеки.
5) В каталоге src\wdm\usb\bulkusb\sys исходники драйвера.
Глубже objfree\i386 собственно драйвер bulkusb.sys.

9 Достижение максимальной скорости обмена.

Для удобства изложения материала рассмотрим режим обмена Full Speed с устройством, имеющим одну конечную точку типа Bulk. Таким устройством может быть экспериментальная установка, снимающая непрерывные данные.

Драйвер хоста планирует обмен с устройствами фреймами. То есть разбивает какой-то промежуток времени, на части. Для Full Speed это части по 1 миллисекунде. Длительность промежутка времени, которое берется для планирования, зависит от количества буферов имеющихся или выделенных хосту. В [1] в таблице 5-1 указано, что за один фрейм можно переслать 832 байта собственно данных, а за 1 секунду 832кбайта. На современных компьютерах и ОС реальные значения для Full Speed составляют около 1 мегабайта в секунду.

Для того чтобы обеспечить пересылку таких данных нужно хосту дать задание на прием и ли передачу данных еще до начала фрейма. Так как после его начала механизм ПДП в хосте начинает обмен и не реагирует на новые задания.

То есть программист после открытия устройства и нужной pipe стандартной функцией Windows API CreateFile, должен сформировать запрос на чтение/запись большого массива информации. Стандартными функциями ReadFile/WriteFile.

Размер буфера для чтения или записи для вызова функций ReadFile/WriteFile в программе на PC нужно создавать кратным размеру конечной точки в устройстве.

Со стороны микроконтроллера достаточно использовать конечную точку с двойным буферированием. В то время когда идет передача одного буфера, контроллер должен успеть заполнить другой.

Максимальный размер буфера для чтения/записи в программе PC зависит от операционной системы.

Для драйвера bulkusb.sys из DDK описанного выше есть экспериментальные данные по скорости одностороннего обмена.

При компиляции исходного текста драйвера и использования его для связки PDIUSBD12+AT89C52 под Windows удалось достичь скорости обмена 240 килобайт в секунду.

При исследовании причины почему скорость не получается выше, найдена следующая в драйвере.

В файле blk82930.h есть строка
#define BULKUSB_MAX_TRANSFER_SIZE 256

После ее замены на
#define BULKUSB_MAX_TRANSFER_SIZE 4096

И последующей перекомпиляции/замены драйвера скорость обмена поднялся до 808 Кбайт в секунду

Число 4096 это максимальный размер пакета драйвера USBD.SYS, который принимает пакеты от пользовательских драйверов и передает его дальше. Драйверу нижнего уровня. Это же число и ограничивает размер буфера для обмена между программой пользователя и драйвером в системе Windows Me. Точнее размера данных запрашиваемых/передаваемых функциями Read/Write File. Так как буфер может быть больше, нужно только передвигать указатель на размер данных.

При запросе обмена блоком большего размера драйвер останавливается и завешивает программу. Очевидно, что в Win95 и возможно Win98 можно было обмениваться блоками по 256байт.

С подкорректированным драйвером на плате с процессором AT91SAM7S128 и операционными системами NT2000+SP4 и Windows XP.

В NT2000+SP4 на фрейм выделяется 4 пакета (проверено осциллографом) в результате 4*64*1000=256КБ теоретический максимум даже при одном устройстве на шине USB, обмен производился блоками по 4096 байт.

Под Windows XP экспериментально установлен размер блока для обмена – он равен 64 килобайта, скорость при этом достигла 1 мегабайта за секунду.

Попытка чтения блоками в 2 раза больше по 131КБ, привела к той же ошибке, что и Windows Me при увеличении размера блока с выше 4096 байт.

20 мая 2008г. misyachniy@mail.ru

Список использованных материалов

1 Universal Serial Bus Specification Revision 1.1 September 23, 1998
2 Universal Serial Bus Specification Revision 2.0 April 27, 2000
3 USB in a Nutshell Craig Peacock (Craig.Peacock@beyondlogic.org)
Second Release 9th May 2002
4 Firmware Programming Guide for PDIUSBD12 version 1.0 Philips Semiconductors – Asia Product Innovation Centre Visit http://www.flexiusb.com

5 Интерфейс USB. Практика использования и программирования. Павел Агуров, БХВ Петербург 2005.