СЕРИЯ
НАРОДНАЯ ЭЛЕКТРОНИКА
ВЫПУСК 1

Популярно о USB

А. В. Немоляев

Екатеринбург 2015

УДК 621.325.5
ББК 32.84

Немоляев А.В.
Популярно о USB. − Екатеринбург.: Живая мысль, 2015. − 53 с.

В сжатой, но доступной форме излагается популярная микрокомпьютерная шина USB, материал ориентирован на начинающих разработчиков микропроцессорных систем и радиолюбителей.

УДК 621.325.5
ББК 32.84

Оглавление

Предисловие

Шина USB – трудный предмет для программистов и разработчиков, в особенности для русскоязычных и в особенности для начинающих. Спецификация шины версии 1.1 1998 года, составляет более 300 страниц, с тех пор спецификация была дополнена и расширена. Надо заметить, что в англоязычных странах существует целая серия книг по этой теме, популярно излагающая предмет. Актуальность темы подтверждается тем, что знаменитая книга «USB Complete» автора Jan Axelson, объёмом более 500 страниц, выдержала уже четыре издания. В сети интернет, много материалов по шине USB на русском языке, но они носят фрагментарный, отрывочный характер, часто эти публикации ещё больше запутывают и дают превратное представление о предмете. Эта книга, вклад в русскоязычный ресурс по шине USB.

Книга ориентированна на начинающих разработчиков и радиолюбителей, кто не знаком с USB, но хотел бы больше узнать. Книга не является исчерпывающим справочным пособием по USB. Примеры основаны на стандарте USB 1.1 для более простого вхождения в тему. Если отдельно не сказано, то подразумевается режим FS (Full speed). В статье нет, широко освещённых в других источниках, сведений об общей топологии USB, о кабелях, хабах и разъёмах. Здесь больше информации о том, что нужно знать разработчику устройств с микроконтроллерами о протоколе USB для своих разработок. Для устройств USB подключаемых к PC, таких: как мышь, клавиатура, микроконтроллер с поддержкой шины USB, использую термин USB устройство. Персональный компьютер, к которому подключается USB устройство, называю хостом. Доступное изложение теории, будет сопровождаться примерами программ на языке C для микроконтроллера AT90USB162, популярной линейки megaAVR фирмы Atmel. Для справочной информации по USB, рекомендую Гук М.Ю. «Шины PCI,USB и FireWire.Энциклопедия.», издательство «Питер».

Обзор USB

Программное обеспечение хоста делится на два отдельных типа. Программное обеспечение инициализации канала связи и программное обеспечение поддержки рабочего режима обмена данными. Программное обеспечение инициализации начинает работать при подключении к хосту нового USB устройства. Происходит обмен служебной информацией между хостом и USB устройством. В результате обменов служебной информацией, хост определяет: тип устройства, его требования к энергопотреблению, возможность поддержки «спящего режима», тип драйверов для правильной работы USB устройства, и даже возможна загрузка необходимых прикладных программ для работы. Эти новые веяния в духе спецификации PNP (plug and play). Устройства могут подключаться и отключаться в горячем режиме. При подключении и отключении происходит автоматическое переконфигурирование программного обеспечения хоста. Процесс настройки хоста на обмен данными, напоминает процесс раскрутки. Первоначально обмениваются простейшими сигналами по шине, затем сообщения усложняются и наконец, выход на рабочий режим.

Программное обеспечение рабочего режима поддерживает обмен данными, когда хост соответственно сконфигурирован и USB устройство вышло на рабочий режим обмена. В спецификации USB этот начальный процесс называется энумерацией.

В последнее время имеется тенденция к унификации не только протоколов обмена, но и унификация устройств, взаимодействующих с персональным компьютером. Точнее, унификация требований к каналу связи. Идея такая. Придумывается универсальная шина для всего, что только можно подключить. Конечно - эта шина сложно устроена, она многоуровневая, гибкая и адаптируемая для разных конфигураций устройств. Унифицируются и драйвера операционной системы персонального компьютера, который взаимодействует с подключаемым устройством. Преимущество - отпадает необходимость в разработке драйвера для ОС, разработчиками USB устройства. Это должно повышать надёжность ОС, так как разработкой драйверов могут заняться разработчики ОС, а не разработчики устройств. В общем, все плюсы унификации и стандартизации. Но есть и минусы. Очевидная сложность и связанная с ней избыточность, громоздкость технических решений. Тот же подход, что и в протоколах коммуникаций на большие расстояния. Академически стек протоколов OSI и знаменитый TCP/IP.

В связи с выше сказанным, в спецификации USB вводится понятие класса устройств. Все электронные устройства, подключаемые к персональному компьютеру, по своим функциональным качествам очень схожи. Например, звуковые платы предоставляют сервис приблизительно одного уровня. Поэтому устройства стали делить на унифицированные классы. Класс - это группа устройств, объединённых общими характеристиками и способных управляться общим для них программным драйвером операционной системы. Отдельное устройство может объединять функциональность сразу нескольких устройств, принадлежащих разным классам. Если функциональность вашего устройства подходит к некоторому классу, и оно поддерживает спецификацию USB для устройств класса, то не нужно писать драйвер для ОС. Вероятнее всего, драйвер уже имеется в ОС. Функциональность устройства, подпадающего под определённый класс, может быть расширена разработчиком устройства добавлением отдельных команд. Точнее говоря, в стандарте USB предусмотрена возможность некоторого расширения функциональности. В стандарте предусмотрено множество возможностей, которые разработчик устройства может использовать для своих целей, добавляя к базовой функциональности, функциональность расширенную. В спецификации USB, есть две большие области, это собственно USB базового уровня и протоколы устройств классов. Протоколы устройств классов – это некоторая надстройка над протоколами нижнего, базового уровня USB. Логично сначала разобраться с базовыми принципами USB, а уж потом всё остальное.

Виртуальные каналы и конечные точки

Все примеры в статье даются в среде операционной системы Linux и Windows. В режиме передачи данных, когда процесс конфигурирования (энумерации) уже закончился, между USB устройством и хостом должны быть налажены мосты. Это некий набор виртуальных каналов, по которым идёт обмен данными и служебной информацией. Англоязычный термин pipe – труба. Ясно, что они разделяют единственную среду передачи, а потому мультиплексированные и соответственно виртуальные. Можно провести аналогию с локальной сетью. Шина Ethernet одна, но соединений TCP может быть несколько. Но в TCP/IP, соединения TCP могут возникать и завершаться много раз в процессе работы. В USB типы каналов и их количество фиксируется на стадии процесса энумерации.

В стандарте определяется четыре типа каналов: управляющие, по прерыванию, массивов, данных и изохронные. Только управляющий канал является двунаправленным, остальные каналы могут быть только однонаправленными. Для двустороннего обмена требуется организация, хотя бы трёх каналов. Обязательный управляющий канал и по одному для каждого направления. В USB принято, что канал называется каналом ввода, если данные передаются в хост и канал вывода, если из хоста. Применяя специальные приёмы, можно использовать канал управления для передачи небольших объёмов данных. Но это не стандартное использование канала и применяется редко. Напоминаем, что все информационные обмены на шине, для всех видов каналов происходят под управлением хоста. Первым делом хост посылает запрос, а затем USB устройство отвечает.

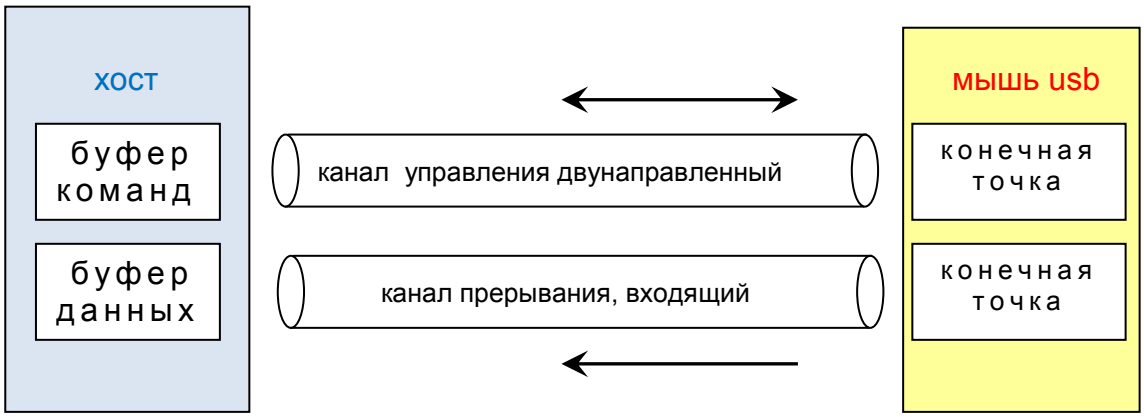


Рисунок 1

Управляющий канал используется для передачи команд протокола USB, передача данных с использованием канала управления, не является стандартным использованием канала. Хотя можно приспособить канал управления для передачи данных прикладной программы. В каждом USB устройстве должен быть хотя бы один управляющий канал.

Канал передачи по прерыванию используется для передачи небольших объёмов данных, но с гарантированными задержками. Хост опрашивает USB устройство на предмет готовности порции данных и если USB устройство готово для обмена, то обмен происходит. Время реакции USB устройства задаётся при конфигурировании и лежит в диапазоне от 1 до 255 миллисекунд. Так что это не связано напрямую с прерываниями в общепринятом понимании. Один канал по прерыванию типа ввода, используется в мыши USB. По каналу передаются клики и координаты

указателя. В моей клавиатуре USB используется два канала по прерыванию и один канал управления.

Изохронные каналы – применяются для передачи потока данных, например аудио и видео. Характерной особенностью, является отсутствие повторной передачи данных в случае ошибок. Повреждённые пакеты просто отбрасываются без запроса повторной передачи. Такая политика позволяет воспроизводить поток данных в реальном времени, без временных задержек. С периодом в 1 миллисекунду хост запрашивает данные, и буфер USB устройства передаётся на хост. В веб камерах используются изохронные каналы.

Канал передачи массивов данных используется в типах устройств, не требующих временной привязки при передаче данных и оперативной реакции на внешние события. Принципиальное отличие от двух предыдущих типов каналов, что не гарантирован временной интервал, по истечении которого данные будут доставлены. В изохронном канале хост опрашивает устройство с частотой 1 миллисекунда. В канале передачи по прерыванию, хост может опрашивать USB устройство с частотой от 1 до 255 миллисекунд. В канале передачи массивов, хост начинает запрашивать данные для приёма в свободное от всех остальных передач время. В случае искажения данных при передаче, происходит повторная передача искажённых данных. В моей флэшке используется два канала передачи массива данных, один на передачу и один на приём, не считая канала управления.

У USB устройства, приёмником и передатчиком данных служит буфер, который называется конечная точка. Тип конечной точки определяет тип канала, который связывает её с хостом. Например, контроллер USB микроконтроллера AT90usb162 фирмы Atmel, имеет в общей сложности 4 конечные точки, не считая конечной точки управления, 4 возможных буфера обмена. Программа микроконтроллера, должна соответственно сконфигурировать необходимое количество конечных точек для своих нужд. Все конечные точки контроллера USB перенумерованы. Нулевая конечная точка всегда используется для нужд канала управления и не может быть переконфигурирована для других целей. Если не создаётся драйвер операционной системы, то на стороне хоста расположение буфера знать не обязательно, так как взаимодействие программы на стороне хоста с USB устройством происходит через системные вызовы операционной системы и скрыто от пользовательской программы.

Хабы, адреса, хост-контроллеры

Кратко опишем аппаратуру хоста. На стороне хоста должен обязательно присутствовать, хотя бы один хост-контроллер и связанный с этим хост-контроллером корневой хаб. В современных PC имеется несколько хост-контроллеров и соответственно несколько корневых хабов. Основателем каждой шины USB является корневой хаб, поэтому, если на PC несколько корневых хабов, то и несколько шин. В Linux, имеющиеся шины USB, можно посмотреть с помощью команды lsusb. Ниже приведён дамп вывода на моём компьютере:

```
alex@big:~$ sudo  lsusb
Bus 005 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 004 Device 002: ID 1a2c:0021
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 003 Device 002: ID 0458:003a KYE Systems Corp. (Mouse Systems)
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 003: ID 13fe:4100 Kingston Technology Company Inc.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

В колонке слева перенумерованы шины USB имеющиеся в компьютере, а значит и контроллеры хабов и корневые хабы. Каждое USB устройство на шине имеет уникальный адрес, диапазон адресов от 1 до 127. Нулевой адрес выполняет служебные функции и не может быть назначен USB устройству постоянно. Адреса распределяются по устройствам на шине хостом, в

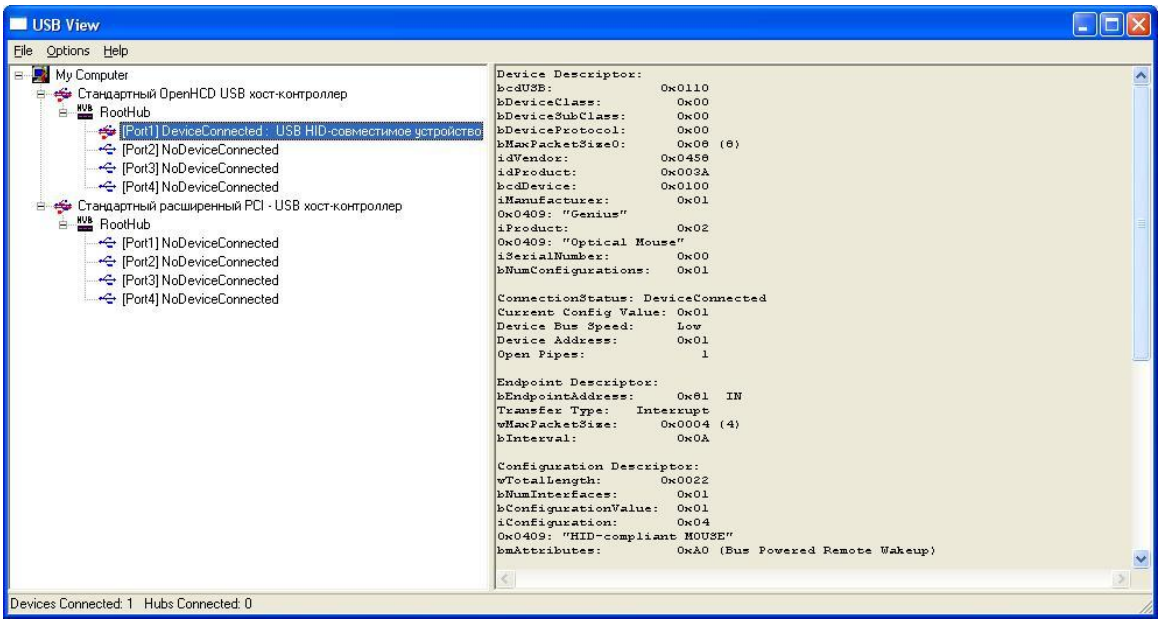


Рисунок 2

процессе энумерации, и сохраняются на все время работы устройства на шине. Из работы программы lsusb видно, что мышь USB подключена к шине 03 и на этой шине ей назначен адрес 02. Устройство хранения данных подключено к шине 01 с адресом 03. Можно поэкспериментировать, поменять последовательность включения и посмотреть, как это отразится

на выводе lsusb. В следующей колонке указан идентификационный код устройства ID. Этот код состоит из 2 частей, идентификатора изготовителя (vendor ID) и идентификатора изделия (product ID). Они представляют собой два уникальных числа, используемых для идентификации конкретного устройства. Операционная система, по этим кодам может определять, какой драйвер требуется загрузить для работы. Значение кода изготовителя назначается форумом разработчиков USB по заказу фирмы. Код устройства устанавливает сам производитель. Программист микроконтроллеров может задать произвольно эти числа. Другое дело, что такие устройства, поступившие в широкую продажу, могут вызвать возражения фирмы собственника кода. Кроме этих чисел, для идентификации, можно использовать номер версии устройства (ID Device).

Для ОС Windows имеется программка usbview от Microsoft. Программа имеет графический интерфейс, слева в виде иерархического дерева представлены хост-контроллеры, корневые хабы и USB устройства. Если в левом окне выделить элемент, то в правом окне увидим развёрнутую информацию. Смотри рисунок 2. Порт хаба (шина) к которой подключено устройство, можно увидеть в левом окне, а адрес можно найти в правом, это атрибут «Device Address». Программа «USB Device Tree Viewer» от Uwe Sieber из Германии, более удобна в использовании.

Дескрипторы

Известно, что к проектированию архитектуры шины USB подходили весьма основательно. Хотели создать архитектуру, которая сможет удовлетворить потребности самых сложных периферийных устройств, широкого спектра применения. Для этого придумали достаточно сложную структуру иерархических данных, называемую набор дескрипторов. В этой структуре, формат которой строго регламентируется стандартом, должна храниться вся информация, с помощью которой хост может автоматически конфигурироваться и начинать нормальную работу с USB устройством. Именно набором дескрипторов USB устройства обменивается с хостом во время процесса энумерации. Имеется 4 основных типа дескрипторов: дескриптор устройства, дескриптор конфигурации, дескриптор интерфейса и дескриптор конечной точки. Все эти виды дескрипторов должны обязательно присутствовать в USB устройстве. Рисунок 3 поясняет сказанное.

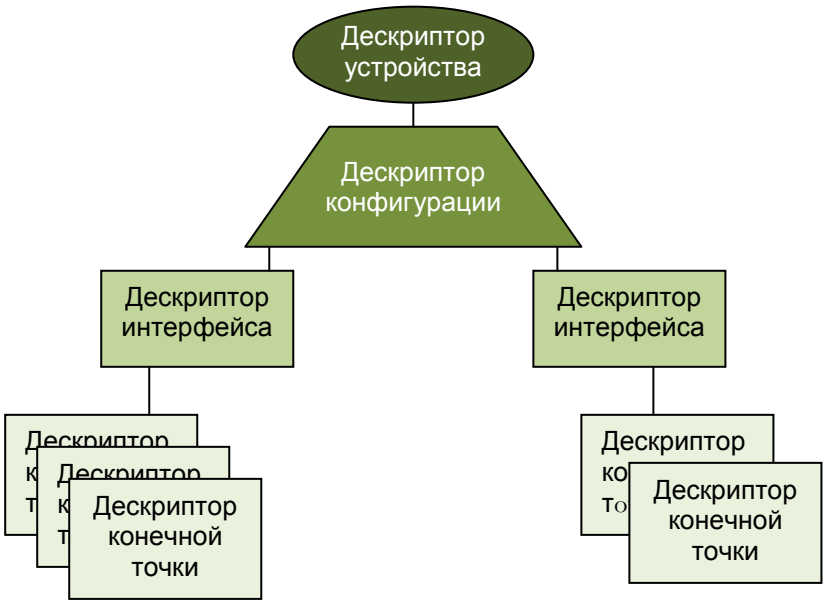


Рисунок 3

Конечные точки объединяются в интерфейс. Каждому интерфейсу соответствует драйвер операционной системы (ОС) хоста. Некоторые устройства могут иметь несколько интерфейсов, которые могут функционировать одновременно. Телефонный аппарат VOIP может иметь интерфейс клавиатуры и аудио интерфейс, тогда ОС хоста будет использовать 2 различных драйвера, для каждого интерфейса. А прикладная программа, будет взаимодействовать с этими драйверами для функционирования. Несколько интерфейсов, могут объединяться в конфигурацию. У USB устройства может быть несколько конфигураций, но не может быть активными несколько конфигураций одновременно, возможно переключение между конфигурациями. В наших примерах не будем использовать USB устройства с несколькими дескрипторами конфигураций и несколькими дескрипторами интерфейсов.

Кроме базовых дескрипторов, которые являются основными, для создания работоспособного устройства, имеются и прочие дескрипторы. Об этих дескрипторах упомянем после.

С начала для сторонников Linux, потом для Windows. Что представляют дескрипторы мыши USB, можно увидеть, воспользовавшись утилитой lsusb для Linux. Эта команда не в

точности выдаёт структуру дескрипторов USB устройства, но очень близкую. Введём команду и перенаправим её вывод в текстовый файл, а затем файл откроем в любом текстовом редакторе.

```
alex@big:~$ sudo lsusb -d 0458:003a -v > dsc.txt
```

Здесь в качестве параметра указан идентификатор устройства, дескрипторы которого требуется вывести и дополнительные ключи. Ищем и открываем файл dsc.txt. Как узнать идентификатор мыши PC, писалось выше.

Посмотрим на содержимое файла dsc.txt. Легко распознать секции с дескрипторами устройства, конфигурации, интерфейса и конечной точки. В названиях полей дескрипторов имеются префиксы, соответственные типу данных.

```
Bus 003 Device 002: ID 0458:003a KYE Systems Corp. (Mouse Systems)
Device Descriptor:
  bLength                 18
  bDescriptorType          1
  bcdUSB                1.10
  bDeviceClass            0 (Defined at Interface level)
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0         8
  idVendor           0x0458 KYE Systems Corp. (Mouse Systems)
  idProduct          0x003a
  bcdDevice             1.00
  iManufacturer           1 Genius
  iProduct                2 Optical Mouse
  iSerial                 0
  bNumConfigurations      1
  Configuration Descriptor:
    bLength                 9
    bDescriptorType         2
    wTotalLength           34
    bNumInterfaces          1
    bConfigurationValue     1
    iConfiguration          4 HID-compliant MOUSE
    bmAttributes         0xa0
      (Bus Powered)
      Remote Wakeup
    MaxPower              100mA
    Interface Descriptor:
      bLength                 9
      bDescriptorType         4
      bInterfaceNumber        0
      bAlternateSetting       0
      bNumEndpoints           1
      bInterfaceClass         3 Human Interface Device
      bInterfaceSubClass      1 Boot Interface Subclass
      bInterfaceProtocol      2 Mouse
      iInterface              0
        HID Device Descriptor:
          bLength                 9
          bDescriptorType        33
```

```
        bcdHID                 1.10
        bCountryCode           0 Not supported
        bNumDescriptors        1
        bDescriptorType        34 Report
        wDescriptorLength      62
       Report Descriptors:
         ** UNAVAILABLE **
    Endpoint Descriptor:
      bLength                  7
      bDescriptorType          5
      bEndpointAddress         0x81  EP 1 IN
      bmAttributes             3
        Transfer Type          Interrupt
        Synch Type             None
        Usage Type             Data
      wMaxPacketSize  0x0004  1x 4 bytes
      bInterval                10
Device Status:                 0x0000
  (Bus Powered)
```

Начнём с описания полей дескриптора устройства.

поле	размер	описание
bLength	1	размер дескриптора в байтах
bDescriptorType	1	тип дескриптора
bcdUSB	2	номер спецификации USB, которой удовлетворяет устройство, в двоично-десятичном формате
bMaxPacketSize	1	максимальный размер пакета для конечной точки 0, из ряда 8,16,32,64
idVendor	2	код разработчика
idProduct	2	код разработки
bcdDevice	2	код версии разработки, в двоично-десятичном виде
iManufacturer	1	индекс текстовой строки производителя
iProduct	1	индекс текстовой строки изделия
iSerial	1	индекс текстовой строки серийного номера
bNumConfigurations	1	количество возможных конфигураций

Поле bMaxPacketSize задаёт размер максимального пакета конечной точки управления. Конечная точка управления используется для приёма команд от хоста. Эта конечная точка не имеет дескриптора, только задаётся размер её буфера. Дескрипторы могут содержать не только информацию в двоичном коде, а так же и текст. Эта информация не обязательна для работы USB устройства, но весьма ценна для человеческого восприятия. Для передачи этой информации служат строковые дескрипторы. Дескриптор строки состоит из 2 байтных символов UNICODE-16 v1. Такая кодировка используется в Windows. Структура строковых дескрипторов не описана в дампе вывода, lsusb извлекла текст по умолчанию и подставила в отчёт. Поля с индексами, указывают на номер строкового дескриптора в пуле строковых дескрипторов. Если в индексном поле задан 0, то считается, что строковый дескриптор не задан. Из дампа информации выведенной lsusb видно, что iSerial не задано, а описатель iProduct имеет индекс 2.

Поля: bDeviceClass, bDeviceSubClass, bDeviceProtocol применяются в USB устройствах, относящихся к стандартизованным классам устройств. Ранее уже сообщалось о существовании унифицированных классах устройств, существующих в рамках стандарта USB. Классы устройств – это надстройка над базовым USB. По аналогии, протокол прикладного уровня HTTP – это надстройка над TCP/IP. Рассматриваемая мышь относится к классу устройств HID. Но это

отдельная большая тема. Важно знать, что совсем не обязательно разбираться в протоколах типа HID, для того чтобы создавать устройства на микроконтроллерах с USB.

Далее идёт дескриптор конфигурации.

поле	размер	описание
wTotalLength	1	полная длина возвращаемых данных
bNumInterfaces	1	количество интерфейсов
bConfigurationValue	1	условный номер конфигурации
iConfiguration	1	индекс строкового дескриптора к конфигурации
bmAttributes	1	атрибуты конфигурации
bMaxPower	1	максимальный ток потребления в единицах по 2 mA

Поле bMaxPower задаёт максимальный ток, потребляемый USB устройством. Задаётся в единицах кратных 2 миллиамперам. В дампе lsusb этот параметр уже переведён в миллиамперы. Поле bmAttributes – набор битов, способ одним байтом задать несколько параметров. Этим полем задаются энергетические характеристики устройства. Поэтому сделаем некоторое введение.

Все USB устройства делятся на 3 класса по потребляемой энергии: питаемые от шины, питаемые от шины с высоким потреблением энергии и имеющие собственный источник энергии. Напомним, что USB устройства подключаемые к шине USB, могут получать энергию по самой шине. Те что потребляют меньше 100 миллиампер, относятся к классу с низким потреблением (Low power). А USB устройства потребляющие до 500 миллиампер от самой шины, относятся к мощным (High power). Третья категория это те, что имеют собственный источник (Self powered). Имеющие собственный источник питания, могут потреблять дополнительно и от шины, но не боле 100 миллиампер.

Если установлен бит 6 поля bmAttribute, то это указывает, что USB устройство имеет собственный источник питания. Бит 5, что USB устройство может являться источником сигнала пробуждения, выводящим вышестоящий хаб из состояния низкого потребления. Теперь ясно, почему моя мышь может разбудить компьютер. Остальные биты зарезервированы.

В дескрипторе интерфейса, пока только интересным являются поля bNumEndpoints и поле iInterface. Из названия ясно, что одно поле задаёт количество конечных точек в интерфейсе, не считая управляющей конечной точки. Другое поле - индекс строкового дескриптора, указывающий на строковый дескриптор интерфейса. Поля относящиеся к секции HID Device Descriptor, рассматривать не будем.

Последним идёт дескриптор единственной конечной точки.

поле	размер	описание
bEndpointAddress	1	адрес конечной точки
bmAttributes	1	битовое поле атрибутов
wMaxPacketSize	1	максимальный размер пакета
bInterval	1	интервал опроса в миллисекундах

Адрес конечной точки состоит из 2-х частей, номер и направление. Для номера выделены биты 0..3 , а для направления предназначен бит 7. Если этот бит установлен, то конечная точка ввода. Остальные биты зарезервированы. Всего можно адресовать до 30 точек, 15 на ввод и 15 на вывод. Нулевой адрес зарезервирован для точки управления. Из дампа вывода видно, что у мыши конечная точка типа ввода, посылает данные на хост и имеет номер 1.

Поле bmAttributes , биты 0..1 задают тип точки: 0 – управления, 1 – изохронной передачи, 2 – массовой передачи, 3 – передачи по прерыванию. Биты 2..5 имеют смысл только для изохронных типов конечных точек. Биты 2..3 задают тип синхронизации для изохронной конечной точки: 0 – нет синхронизации, 1 – асинхронная, 2 – адаптивная, 3 – синхронная. Биты 4..5 задают тип использования изохронной конечной точки: 0 – для данных , 1 – обратная связь, 2 – данные с предоставлением неявной обратной связи, 3 – зарезервировано. Смысл этих параметров относится к изохронным передачам, которые не будем рассматривать подробно. Поле wMaxPacketSize , задаёт максимальный размер пакета, который способна принять точка. Мышь

передаёт хосту пакеты данных по 4 байта. Поле bInterval задаёт максимально возможную задержку опроса, для конечных точек передач по прерыванию. При поступлении запроса от хоста на передачу данных, данные будут переданы с возможной задержкой не более чем значение в bInterval. Для изохронных точек всегда 1. Для остальных не имеет значения. Мышь опрашивается с частотой 10 миллисекунд.

В ОС Windows дамп вывода программы UsbTreeView для мыши USB, приведён ниже, нет только информации специфичной для Windows. По моему мнению, информация более удобная для восприятия. Строковые дескрипторы показываются отдельно.

```
        ------------------ Device Descriptor ------------------
bLength                : 0x12 (18 bytes)
bDescriptorType        : 0x01 (Device Descriptor)
bcdUSB                 : 0x110 (USB Version 1.10)
bDeviceClass           : 0x00 (defined by the interface descriptors)
bDeviceSubClass        : 0x00
bDeviceProtocol        : 0x00
bMaxPacketSize0        : 0x08 (8 bytes)
idVendor               : 0x0458 (KYE Systems Corp.)
idProduct              : 0x003A
bcdDevice              : 0x100
iManufacturer          : 0x01
 Language 0x0409       : "Genius"
iProduct               : 0x02
 Language 0x0409       : "Optical Mouse"
iSerialNumber          : 0x00
bNumConfigurations     : 0x01

        ------------------ String Descriptors ------------------
            ------ String Descriptor 0 ------
bLength                : 0x04 (4 bytes)
bDescriptorType        : 0x03 (String Descriptor)
Language ID[0]         : 0x0409 (English - United States)
            ------ String Descriptor 1 ------
bLength                : 0x0E (14 bytes)
bDescriptorType        : 0x03 (String Descriptor)
Language 0x0409        : "Genius"
            ------ String Descriptor 2 ------
bLength                : 0x1C (28 bytes)
bDescriptorType        : 0x03 (String Descriptor)
Language 0x0409        : "Optical Mouse"
            ------ String Descriptor 4 ------
bLength                : 0x28 (40 bytes)
bDescriptorType        : 0x03 (String Descriptor)
Language 0x0409        : "HID-compliant MOUSE"

        -------------- Configuration Descriptor ---------------
bLength                : 0x09 (9 bytes)
bDescriptorType        : 0x02 (Configuration Descriptor)
wTotalLength           : 0x0022 (34 bytes)
bNumInterfaces         : 0x01
bConfigurationValue    : 0x01
iConfiguration         : 0x04
 Language 0x0409       : "HID-compliant MOUSE"
bmAttributes           : 0xA0
```

```
D7: Reserved, set 1     : 0x01
D6: Self Powered        : 0x00 (no)
D5: Remote Wakeup       : 0x01 (yes)
D4..0: Reserved, set 0  : 0x00
MaxPower                : 0x32 (100 mA)

        ---------- Interface Descriptor ----------
bLength                 : 0x09 (9 bytes)
bDescriptorType         : 0x04 (Interface Descriptor)
bInterfaceNumber        : 0x00
bAlternateSetting       : 0x00
bNumEndpoints           : 0x01 (1 Endpoints)
bInterfaceClass         : 0x03 (HID - Human Interface Device)
bInterfaceSubClass      : 0x01 (Boot Interface)
bInterfaceProtocol      : 0x02 (Mouse)
iInterface              : 0x00

        ---------- Endpoint Descriptor ----------
bLength                 : 0x07 (7 bytes)
bDescriptorType         : 0x05 (Endpoint Descriptor)
bEndpointAddress        : 0x81 (Direction=IN  EndpointID=1)
bmAttributes            : 0x03 (TransferType=Interrupt)
wMaxPacketSize          : 0x04
bInterval               : 0x0A (10 ms)
```

Здесь всё те же дескрипторы, только в ином виде. Без труда можно разобраться, с учётом информации приведённой выше для Linux.

Даже такое поверхностное знание организации USB, позволяет решать некоторые проблемы. В ОС Windows, сведения об устройствах USB, которые подключались, и с которыми работала система, записываются в системный реестр и там хранятся. При подключении устройства, которое уже подключалось, информация о конфигурации берётся именно оттуда. Иногда происходит сбой и USB устройство система не видит. У меня так бывало с принтерами, сканерами, конвертерами USB⇔RS-232. Лучше утилитой типа USBview, посмотреть подключенные устройства и дескрипторы проблемного устройства. Часто, даже беглого взгляда достаточно, чтобы понять, что устройство неверно сконфигурировано. В большинстве случаев, для решения проблемы требуется удалить испорченные записи об устройстве в реестре. Создайте командный файл следующего содержания:

```
SET  DEVMGR_SHOW_NONPRESENT_DEVICES=1
MMC  DEVMGMT.MSC
```

Этот набор команд файла, запустит диспетчер устройств и установит значение переменной среды. Далее установить галочку в меню «Вид -> Показать скрытые устройства ». Записи о драйверах скрытых устройств будут приглушённого цвета, удалите ненужную запись и вставьте штекер USB проблемного устройства. Проблемное устройство будет сконфигурировано с белого листа. Те же задачи, но более комфортно можно решить с программой USBDeview от Nir Sofer.

LibUSB

Напишем простейшее приложение на стороне хоста, работающее с USB устройством. В качестве устройства, будем использовать мышь USB, хотя устройство может быть любым. Примеры программ для Windows и Linux. Для Windows программа создана в среде Microsoft Visual C++ 6.0 , а для Linux на C для GCC.

Проблема написания приложения для устройства USB, состоит в необходимости создания драйвера. Создание драйвера – задача не рядовая. Получается, что для изучения USB, требуется предварительно научиться писать драйвера для ОС.

Другой подход – это использовать существующие драйвера ОС. Уже говорилось о USB устройствах, имеющих архитектуру, относящуюся к определённым стандартизованным классам. Для большинства таких типов устройств уже имеются драйвера в составе установленной ОС. Например, мышь относится к классу HID , для неё имеются стандартные драйвера в ОС. Такой подход имеет право на жизнь. Через API, из своей программы обращаетесь к ОС, а она через драйвер к USB устройству. Но тогда создаваемое USB устройство должна поддерживать протокол HID. Это достаточно сложная надстройка над базовой функциональностью USB.

Оба этих метода не подходят для начинающих изучать USB. Так как быть? К счастью для новичков, имеется библиотека libusb. Первоначально она создавалась для Linux, теперь имеются порты и для Windows. Уже известная утилита lsusb использует библиотеку libusb. Эта библиотека позволяет писать прикладные программы, напрямую обращающиеся к USB устройству, устраняя необходимость использования специальных драйверов ОС.

Для работы нужно установить libusb-win32 для Windows. Эта библиотека в Windows может применяться в двух разных режимах, как обыкновенный драйвер устройства и как фильтрующий драйвер для уже установленных устройств.

В первом случае, необходимо удалить USB устройство, все его драйвера и скрытые записи об установке как описывалось ранее, если они конечно имелись. При подключении USB устройства Windows попросит драйвер, выдать ей файл INF , который заранее заготовили для этого устройства. В составе пакета поставляется нехитрый мастер, с помощью которого, легко создать требуемый INF файл. Теперь при всяком подключении разработанного устройства, она будет использовать libusb-win32 как полноценный драйвер. Из своей прикладной программы, обращаясь к libusb-win32, будете работать с разработанным USB устройством.

Второй вариант, как фильтрующий драйвер. В этом случае никаких файлов INF не нужно, в этом режиме можно работать с любым USB устройством, подключенным к хосту, выбирая их по кодам производителя. Эти устройства могут быть сконфигурированы, и иметь подключенные драйвера. Есть некоторые ограничения, нельзя менять конфигурацию этих USB устройств, но в этом необходимость возникает редко. Такой режим больше рекомендуется для экспериментов при разработке. Не исключена возможность нарушения работы системы.

С сайта sourceforge.net скачайте бинарный архив, в моём случае, это libusb-win32-bin-1.2.6.0.zip . Архив содержит всё необходимое для установки библиотеки и для разработки, в том числе утилиты для тестирования и конфигурирования.

В подкаталоге bin архива, содержатся подкаталоги для всех основных архитектур Windows. У меня Windows XP SP3 , поэтому используем каталог x86. Файл libusb0_x86.dll переименовываем в libusb0.dll и копируем в %SystemRoot%\system32. Файл libusb0.sys копируем в %SystemRoot%\system32\drivers. Подключаем мышь USB, с которой будем экспериментировать, затем запускаем установщик фильтра install-filter-win.exe и выбираем нашу мышь из списка. Далее запускаем тестовую программу testlibusb-win.exe , если всё правильно работает, должны увидеть в окне дескрипторы мыши. Устанавливать фильтр нужно единожды, сведения о USB устройствах, с которыми предполагается работать, запоминаются.

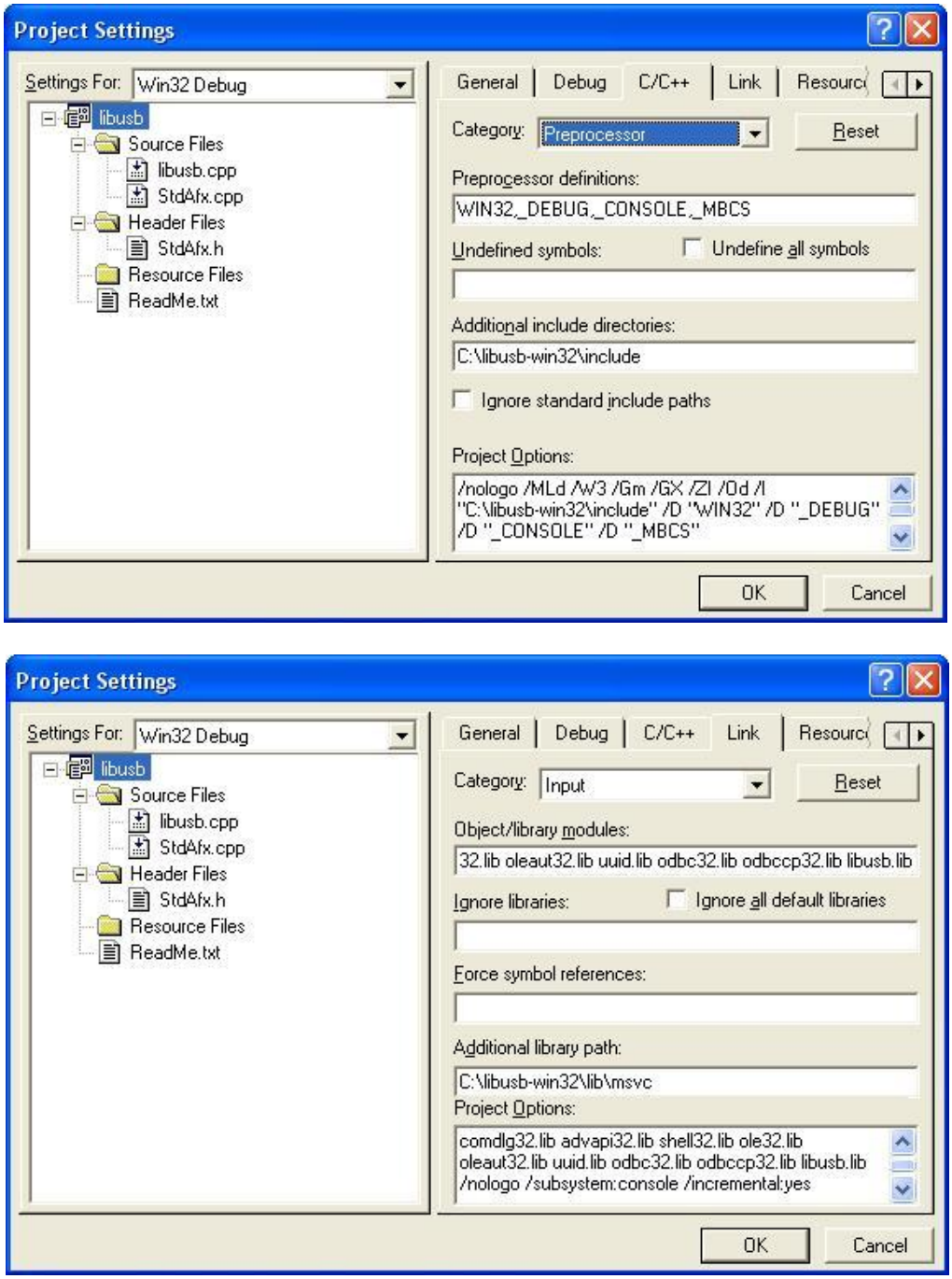


Рисунок 4

Создайте шаблонный, консольный проект «Hello, world!» в MSVC. Затем нужно настроить этот проект, подключить заголовочный файл и статическую библиотеку среды libusb-win32. В моём случае, архив распакован в C:\libusb-win32 , в подкаталоге include находится заголовочный файл lusb0_usb.h , а в подкаталоге C:\libusb-win32\lib\msvc находится файл библиотеки libusb.lib. Там же можно обнаружить каталоги с библиотеками для 64-разрядных платформ.

В главном меню выбирается, «Project -> Settings…» , вкладка C/C++ , в поле ввода «Additional include directories» , ввести путь к каталогу, где лежит заголовочный файл lusb0_usb.h .

На вкладке «Link» , в поле «Object/library modules» , через пробел добавить библиотеку libusb.lib , а в поле ввода «Additional library path», добавить путь к этой библиотеке.

Посмотрите коды производителя мыши USB и подставьте в текст программы, затем откомпилируйте. Программа открывает USB устройство по кодам производителя и продукта,

считывает некоторые строковые дескрипторы и выводит на экран. Запускать из командной строки. Имея работающий шаблон легко создать программу с графическим интерфейсом.

Функции usb_find_busses и usb_find_devices находят все шины хоста и все устройства на шинах и сохраняют эти сведения во внутренних структурах данных. Функция usb_get_busses возвращает ссылку на глобальную структуру, где хранятся сведения о шинах, устройствах на шинах и дескрипторах этих устройств. Описание функций библиотеки libusb дано в приложении. На момент написания материала, существовали 2 версии библиотеки, версия 0.1 и версия 1.0 . Они достаточно сильно различаются, libusb-win32 перенесена из версии 0.1 для Linux.

На рисунке 4 приведены формы проекта MSVC, для настройки среды разработки с библиотекой libusb. С сайта vk.com/protocols можно скачать готовый проект для Visual C 6.0. для Windows.

В Ubuntu для программирования требуется установить пакет libusb-dev, или установить библиотеку из исходных кодов. Для понимания работы с библиотекой необходимо знание работы со структурами и со списками структур на языке С. Для компилирования необходимо иметь на машине компилятор GCC. На моём компьютере была установлена версия библиотеки libusb - 0.1.12. Утилитой lsusb, посмотрите коды производителя и продукта своей мыши или иного устройства, к которому нужно подключиться. Распакуйте архив с программой для Linux в любой доступный каталог. Откройте файл c.c любым редактором, измените эти коды на коды своего USB устройства в тексте программы. Из командной строки зайдите в каталог с makefile и выполните команду make all. Запуск компилированной программы осуществлять с правами суперпользователя.

Пакет – базовое понятие USB

Коммуникационный стек USB, в отличие от TCP/IP «несимметричный». Поэтому попытки проводить прямую аналогию со стеком TCP/IP и подобными стеками, нужно сразу отбросить. Уровни есть, но уровни разные на стороне хоста и USB устройства. Хотя без некоторой аналогии с сетями не обойтись.

Для передачи данных в обоих направлениях и разного рода служебной информации используется только 2 проводника, не считая нулевого и питания. Соответственно и сложная система сигнализации на шине. Классический подход к изучению USB, сверху вниз или снизу вверх, мало подходит. Точнее будет трудно понять существо дела. Хотя формально всё будет верно, но для новичка в USB непонятно. Поэтому лучше начать с середины. Предположим, что по этим проводникам как-то передаются биты информации. Первоначально не будем заострять внимание на механизмах реализации передачи сигналов по проводникам. Отдельно нужно заметить, что в работе любого USB устройства, есть две различные стадии, стадия конфигурирования и рабочий режим обмена. Здесь не будем рассматривать процесс энумерации, а только рабочий режим обмена данными.

Основной элемент протокола USB - это пакет. Пакет, в свою очередь состоит из 3-х частей: преамбула, тело и концевик. Преамбула служит для подстройки частоты генератора приёмника, в английской транскрипции SYNC. Тело пакета - последовательность байт, от одного до 1025. Первый байт тела пакета - это идентификатор пакета, в английской транскрипции PID. Идентификатор пакет задаёт тип пакета, его функциональное назначение. Только первые 4 бита PID, кодируют тип пакета, а остальные служат для защиты от ошибок и дублируют первые 4 бита, в виде инверсной копии. Четырьмя битами, кодируется 10 типов пакетов, остальные 6 зарезервированы. Эти десять типов пакетов делятся на четыре категории: пакеты маркеры, данных, квитирования и специальные. Тело пакета содержит полезную информацию. Концевик - должен сигнализировать о завершении пакета. В английской транскрипции End-Of-Packet или EOP. Из этих трёх частей состоят все пакеты, которые снуют туда-сюда по шине.

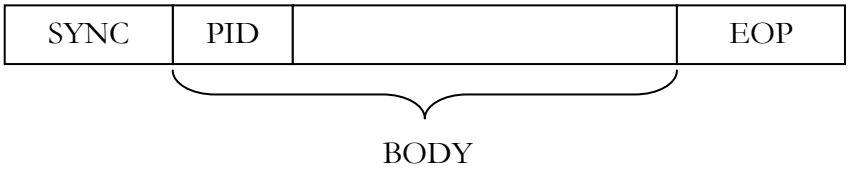


Рисунок 5

Маркерный пакет – старт временного окна

Все виды пакетов шины USB делятся на 4 категории: маркерные пакеты, пакеты данных, квитирования и специальные. Маркерный пакет, как следует из названия, выполняет служебную роль, он сообщает приёмнику о том, какие пакеты последуют за маркерным пакетом. Из дальнейшего будет ясно, для чего они служат.

Пропускная способность шины делится на временные окна. Начало временного окна отмечается служебным пакетом SOF (Start Of Frame). Пакет SOF транслируется каждую 1 миллисекунду корневым хабом и широковещательно распространяется по всему дереву хабов, достигая каждого из USB устройств. За 1 миллисекунду, при частоте 12 мегагерц, теоретически можно передать максимально 1500 байт данных. Заметим, что тактовая частота в 12 мегагерц является стандартной для шины USB. Дополнительные затраты на SYNC и EOP, снижают максимально возможный размер данных передаваемых в одном временном окне, до 1200 байт. Пакет SOF состоит из идентификатора пакета PID, данных и контрольной суммы. Контрольная сумма пакета SOF имеет размер 5 бит. Данные пакета имеют размер 11 бит. Это число, длиной 11 бит монотонно увеличивается с каждым пакетом, при переполнении обнуляется и вновь возрастает. Цикл длится 2048 ms. Некоторые устройства используют этот счётчик для привязки к

Рисунок 6

реальному времени. Пакет SOF не содержит адресной части, так как является широковещательным и не требует квитирования. Не надо забывать что, как и всякий пакет, пакет SOF содержит преамбулу и концевик. Завершение временного окна, отмечается особым состоянием сигнальных проводов, которое получило название EOF(End of frame). Это состояние молчания.

С помощью пакетов SOF создаётся временная сетка синхронизации по всему дереву устройств. Источником пакетов SOF всегда является хост, как и для всех маркерных пакетов. Пакеты SOF можно увидеть на экране осциллографа, если подключить щуп к одной из линий данных. Нужно помнить, что инициатором любых обменов на шине является хост. Обмены на шине происходят в течение временного окна в обоих направлениях.

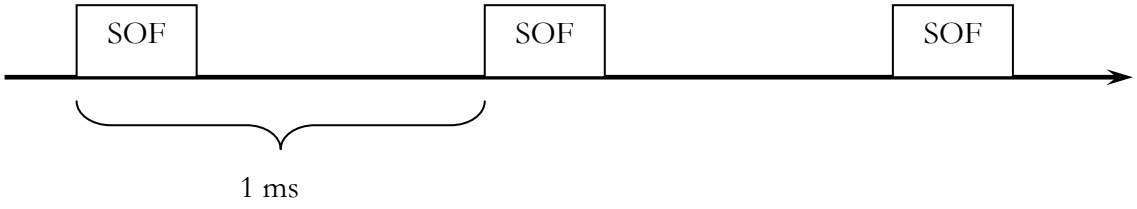


Рисунок 7

Маркерные пакеты – SETUP, IN , OUT

В отличие от маркерного пакета SOF, пакеты SETUP, IN и OUT, не являются широковещательными, а адресованы конкретному устройству на шине. Каждый из этих пакетов состоит из, PID, адреса устройства, адреса конечной точки и контрольной суммы. Три типа этих маркерных пакетов применяются при инициализации обмена данными между хостом и USB устройством. Как не трудно догадаться, маркерные пакеты сообщают адресуемому устройству, что нужно готовиться к приёму данных, либо команд, которые последуют за маркерными пакетами. Пакет IN применяется при инициализации передачи данных от устройства к хосту, а OUT в противоположном направлении. Пакеты IN и OUT могут адресовать любое USB устройство и любую конечную точку USB устройства на шине. Маркерный пакет SETUP, специальный вариант пакета OUT , он всегда адресован нулевой конечной точке и имеет наивысший приоритет. Любое устройство, обязано немедленно принять этот адресованный ему пакет, даже если требуется прервать выполнение предыдущей команды. Рисунок 8 иллюстрирует сказанное.

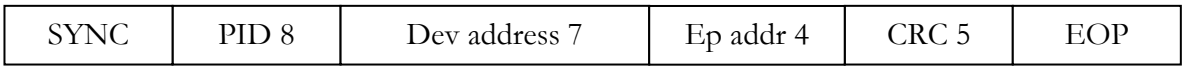


Рисунок. 8

Пакеты данных

Маркерные пакеты выполняют вспомогательную роль, сами данные передаются отдельным типом пакетов. Существует два вида пакетов данных, они называются DATA0 и DATA1. Эти пакеты состоят из PID, прикладных данных и контрольной суммы. Прикладные данные или полезный груз, могут иметь размер от 0 до 1023 байт для одного пакета данных. Различные типы этих пакетов, попеременно передаются для дополнительного контроля возможного искажения передачи. Передатчик транслирует, DATA0, DATA1, DATA0. Приёмник отслеживает это чередование и если чередование нарушается, то сигнализирует об ошибке в приёме данных. Эта дополнительная мера, усиливает защиту от некоторых видов ошибок. В отличие от маркерных пакетов, в пакетах данных, контрольная сумма 16 бит, а не 5, что направлено на усиление защиты от помех.

Пакеты квитирования

Для подтверждения приема, управления потоком и сигнализации об ошибках используются пакеты квитирования, так же как и в других протоколах обмена. Для режима FS (full speed) имеется три пакета квитирования: NAK, ACK и STALL. Все пакеты квитирования не содержат ни контрольной суммы, ни каких либо частей, кроме PID. Как упоминалось выше пакет PID, делится на два полубайта, с одинаковым значением, это и служит защитой от искажения самого пакета квитирования. Пакет ACK подтверждает, успешный приём переданного маркерного пакета или пакета данных. Пакет NAK сообщает о невозможности принять маркер пакет или пакет данных, прямо сейчас. Например, USB устройство занято другой работой, или не успевает обрабатывать поступающие данные, занято обработкой предыдущего пакета. Протоколом допустимо, что USB устройство может квитировать любой принятый пакет данных или пакет маркера, пакетом NAK, кроме пакета SETUP. На принятый пакет SETUP, устройство не должно отвечать пакетом NAK, пакет SETUP обязателен к рассмотрению. Образно можно сказать, что пакетом NAK USB устройство сообщает о необходимости подождать и не торопиться.

Пакет STALL передаётся USB устройством в случае серьёзных ошибок, чтобы сообщить хосту о невозможности дальнейшей работы. Например, в принтере кончилась бумага и попытки передавать данные лишены смысла. Для таких ситуаций USB устройство отправит пакет STALL, сообщая этим, что адресат заблокирован, и бессмысленно загружать шину попытками передачи данных.

Остаётся невыясненным, а как обрабатывается ситуация, когда пакет повреждён? Если проверка целостности пакета покажет, что он искажён? На принятые, искажённые пакеты, ни USB устройство, ни хост, не должны никак отвечать. Точнее говоря, они должны выдержать некоторый временной промежуток молчания, тайм-аут. По отсутствию пакета квитирования передатчик догадается, что пакет искажён и повторит передачу. Если несколько повторов передача будет неудачной, хост перейдёт в аварийный режим.

тип пакета	обозначение	значение
маркер	OUT	0001
	IN	1001
	SOF	0101
	SETUP	1101
квитирования	ACK	0010
	NAK	1010
	STALL	1110
данных	DATA0	0011
	DATA1	1011

Транзакции

Из пакетов строятся транзакции. Транзакция - это логическая единица обмена данными. Транзакции, как и пакеты, имеют структуру и находятся на более высокой степени абстракции. Транзакции совершаются в рамках временных окон, о которых уже говорилось. Каждая транзакция должна начинаться и завершаться в рамках единственного окна и недопустимо, чтобы транзакция началась в одном временном окне и закончилась в другом. Напомним, что начало временного окна отмечается пакетом SOF и завершается особым состоянием шины EOF.

Существует 4 типа транзакций: прерывания, массовой передачи, изохронных передач, и управления. Сообщалось о каналах, связывающих конечную точку с хостом. Поток транзакций и составляет канал. Транзакции для различных типов передач имеют протокольные различия, обусловленные гарантированием или не гарантированием пропускной способности, времени отклика, надежности доставки и синхронизации ввода и вывода. К примеру, транзакции изохронных передач, гарантируют скорость обмена, но не обеспечивают надёжности доставки. При приёме повреждённых пакетов, повторная передача повреждённого пакета не осуществляется, в то время как транзакции прерываний, обеспечивают гарантированное время реакции USB устройства на запрос хоста, но не предназначены для передачи больших объёмов данных.

Рассмотрим транзакцию прерывания чтения. Любая транзакция инициируется хостом, посредством передачи пакета маркера IN, который ранее описывался. Пакет маркера описывает тип транзакции, адрес USB устройства, номер конечной точки. Адрес распознаётся USB устройством, и оно возвращает пакет данных. Затем хост отвечает квитированием. В транзакции прерывания может быть только один пакет данных. Для передачи нескольких пакетов используются дополнительные транзакции. Пакет маркера защищён от искажений, и если USB устройство будет принят искажённый пакет маркера, USB устройство выдерживает тайм-аут, сигнализируя хосту о необходимости повторной передачи маркерного пакета. В случае не готовности данных для передачи, посылается пакет NAK. Типичное USB устройство распознав принятый от хоста пакет IN, заполняет буфер конечной точки данными и изменив значение некоторого флага, запускает отправку содержимого буфера хосту. Но до тех пор пока данные не отправлены, SIE автоматически посылает пакеты NAK хосту. Если USB устройству требуется значительное время на ожидание готовности данных, то пропускная способность шины будет расходоваться впустую, на приём потока пакетов квитирования. В этом случае микропрограммное обеспечение USB устройства переводит конечную точку в блокированное состояние. В блокированном состоянии конечная точка на пакет IN от хоста, отвечает пакетом квитирования STALL. Приняв этот пакет, хост узнаёт, что конечная точка находится в блокированном состоянии. Этим пакетом сообщается хосту о необходимости длительного ожидания. Если пакет данных, переданный USB устройством, принят хостом с искажениями, то хост выдерживает тайм-аут, сигнализируя USB устройству о необходимости повторной передачи. Рисунок поясняет сказанное.

В транзакции прерывания записи, хост посылает пакет маркера OUT транзакции записи и пакет данных. В случае искажения маркерного пакета или пакета данных, переданных хостом, USB устройство выдерживает тайм аут, свидетельствуя об ошибке. При успешном принятии данных, USB устройство возвращает пакет положительного квитирования ACK. Если буфер ещё не пуст, возвращается NAK.

Логика транзакций массовой передачи данных (bulk) аналогична рассмотренной выше транзакции прерывания. Понятно, что для транзакций массовой передачи используется свой маркерный пакет. Отличия в основном касаются планирования транзакций массовой передачи хостом. Проще говоря, транзакции массовой передачи посылаются в последнюю очередь.

Транзакции изохронной передачи устроены совсем просто, хост посылает маркерный пакет, затем принимает или посылает пакет данных. Пакеты квитирования не используются, искажённые пакеты просто отбрасываются.

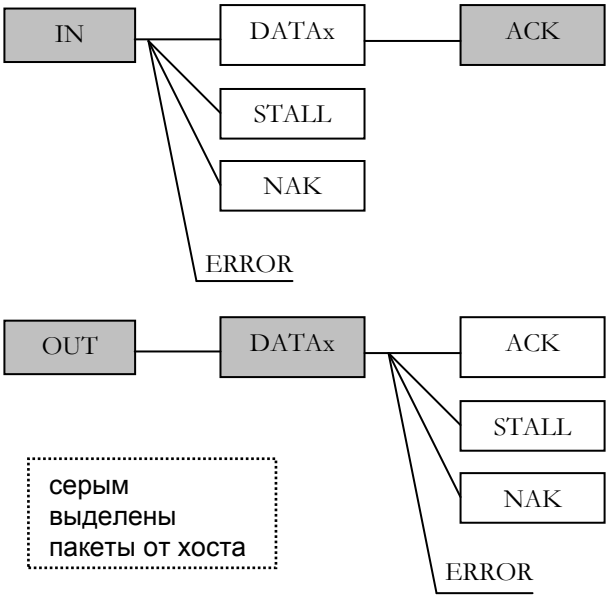


Рисунок 9

Запросы

В терминологии USB, команды, которые посылает хост USB устройству, называются запросами или requests по-английски. Для передачи запросов используется канал управления. В подавляющем большинстве случаев для приёма запросов используется нулевая конечная точка. Но стандартом допускается существование нескольких точек управления. По умолчанию все запросы управления принимаются нулевой конечной точкой. Запросы могут менять конфигурацию и некоторые свойства USB устройства, запрашивать служебную информацию. В примере программы, который был приведён, хост посылает специальный запрос USB устройству на получение строкового дескриптора от USB устройства. В ответ USB устройство, возвращает строковый дескриптор.

Стандартом USB, определён набор запросов, которые должно обрабатывать любое USB устройство. Эти запросы называются стандартными запросами и всего их 11, каждому стандартному запросу присвоен номер. Стандартом предусмотрена возможность создания разработчиком своих собственных запросов. А для USB устройств относящихся к стандартизованным классам, например мышь USB, кроме стандартных запросов, имеется своя система запросов, которую рассматривать не будем, но о существовании которой знать надо. Средствами библиотеки libusb, можно посылать любой запрос USB устройству.

Для подачи запроса USB устройству, используется механизм управляющих передач - это последовательность обмена пакетами, похожая на рассмотренные ранее транзакции, но только сложнее. Кратко рассмотрим стандартные запросы.

Запрос GET_DESCRIPTOR (06h) позволяет хосту получить дескриптор устройства, конфигурации, конечной точки или строки. Во время процесса энумерации, хост получает сведения о USB устройстве с помощью этого запроса.

Запрос SET_DESCRIPTOR (07h), позволяет добавить новый дескриптор или расширить имеющийся.

Запрос SET_ADDRESS (05h), присваивает адрес USB устройству. Каждое USB устройство имеет уникальный адрес на шине. Адрес назначает хост, в процессе энумерации.

Запрос SET_FEATURE (03h), позволяет установить значение свойства или состояния. Данные не возвращаются. Запрос может быть адресован устройству, интерфейсу или конечной точке. Адресовав запрос к конечной точке можно перевести её в блокированное состояние. Конечная точка, находящаяся в блокированном состоянии, на попытки обращения хоста, отвечает пакетом STALL.

Хост может перейти в режим с пониженным энергопотреблением (suspended mode), а USB устройство может выводить хост из этого состояния специальной сигнализацией на шине. Это свойство USB устройства называется – удалённое пробуждение (remote wakeup). Специальным запросом SET_FEATURE, можно дать возможность USB устройству пробуждать хост.

Запрос CLEAR_FEATURE (01h), позволяет сбросить значение свойства или состояния. Запрос аналогичен запросу SET_FEATURE и противоположен по действию. Этим запросом выводится из блокированного состояния конечная точка, если она была заблокирована. Соответственно сбрасывается свойство пробуждать хост (remote wakeup).

Запрос GET_STATUS (00h) позволяет определить состояние USB устройства, интерфейса или конечной точки. Если запрос направлен к USB устройству, то можно получить информацию, установлено или сброшено свойство «wakeup» USB устройства. Дополнительно можно определить, способ питания USB устройства, от шины или от собственного источника.

При направлении запроса GET_STATUS к конечной точке, можно определить состояние конечной точки, заблокирована она или нет.

Запросы GET_CONFIGURATION (08h) и SET_CONFIGURATION (09h), используются хостом для работы с USB устройствами, имеющими несколько конфигураций, а так же в процессе энумерации. Соответственно, можно получить номер текущей конфигурации и установить альтернативную конфигурацию. Установка конфигурации выполняется на стадии конфигурирования USB устройства, в процессе энумерации.

Запросы GET_INTERFACE (0Ah) и SET_INTERFACE (0Bh), позволяют работать с альтернативными установками интерфейсов. Альтернативные установки интерфейсов, позволяют менять некоторые параметры интерфейса после стадии конфигурирования, в процессе нормальной работы. Альтернативные настройки интерфейсов здесь не рассматриваются.

Последний, стандартный запрос SYNC_FRAME (0Ch), имеет достаточно узкую специализацию и применяется в работе изохронных каналов, которые здесь не рассматриваются.

На данном этапе нужно понять, что существует набор запросов, которые посылает хост для управления USB устройством и для конфигурирования на стадии энумерации. В библиотеке libusb имеется функция, с помощью которой можно посылать запросы USB устройству.

Канал управления

Ранее упоминались запросы (requests) , которые выполняют роль команд для USB устройств. Сейчас рассмотрим механизм управляющих передач на канале управления, с помощью которого передаются эти запросы.

Вспомним транзакции на шине USB, все транзакции состоят из трёх фаз: фаза маркера, фаза данных и фаза квитирования. На каждой из фаз в ту или иную сторону отправляется соответственный пакет. Управляющие передачи на канале управления – это как бы транзакции внутри транзакций. Каждая передача управления состоит из двух или трёх фаз, а каждая фаза представляет транзакцию, которая сама состоит из трёх фаз.

Вспомним, что управляющий запрос к USB устройству может содержать параметры, стандартный запрос SET_ADDRESS, в качестве параметра передаёт USB устройству его адрес. Стандартный запрос GET_DESCRIPTOR, должен получить дескриптор от USB устройства, а стандартный запрос SET_DESCRIPTOR, должен передать дескриптор USB устройства. Стадии управляющей передачи:

- стадия установки (Setup Stage) , предназначается для передачи кода запроса от хоста к функции;
- стадия данных (Data Stage) , служит для передачи дополнительной управляющей информации от хоста (Write Control) или к хосту (Read Control). Если передача параметров не предусмотрена в запросе, то эта стадия может отсутствовать. В стандартном запросе SET_FEATURE, нет параметров запроса, поэтому стадия данных отсутствует. В запросе GET_DESCRIPTOR, присутствует стадия данных, дескриптор передаётся от USB устройства к хосту, в стадии данных;
- стадия передачи состояния (Status Stage) , служит для уведомления хоста об успешности или не успешности выполнения предыдущих фаз. Это аналог фазы квитирования в транзакции.

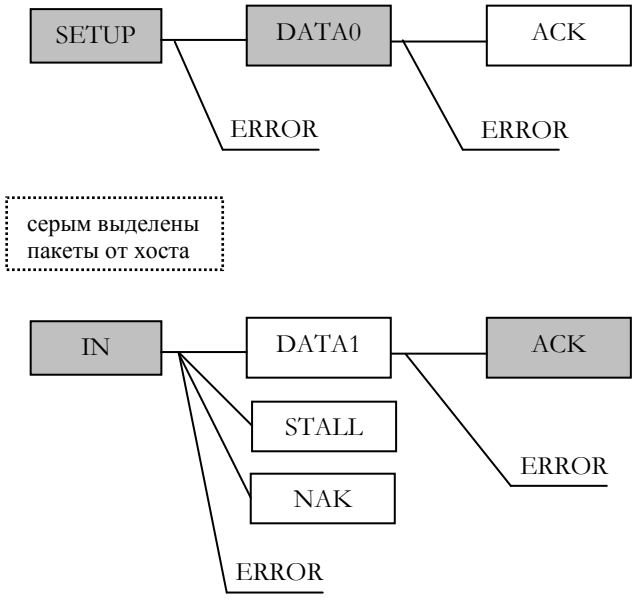


Рисунок 10

На рисунке 10, показана управляющая передача из 2-х стадий, из стадии установки и стадии передачи состояния. Такие управляющие передачи применяются для простейших запросов. Первым передаётся маркерный пакет SETUP, затем пакет данных типа DATA0, всегда размером в 8 байт, иногда этот пакет называют конфигурационным. Хост должен ответить пакетом ACK. Если пакет повреждён, то USB устройство делает тайм-аут, сигнализируя о необходимости повторной передачи пакета. Конфигурационный пакет несёт содержательную информацию.

После обязательной стадии установки, стадия передачи состояния. В этой стадии хост намеревается получить ответ о результате обработки запроса, принятого на предыдущей стадии установки. Первым хост посылает маркерный пакет IN. Если запрос ещё не обработан, в ответ посылается пакет NAK. Если запрос обработан, но запрос не корректен, возвращается STALL. Конечная точка управления не может быть заблокированной. Под ошибкой понимается логичность запроса для текущего состояния USB устройства, а не повреждённость пакета IN. Если запрос правильно обработан, возвращается пакет данных типа DATA1 нулевой длины (Zero Length Packet, ZLP), на что хост отвечает пакетом ACK. На этом управляющий запрос из 2-х стадий завершается.

Конфигурационный пакет имеет строго регламентированную структуру. В таблице приведены поля, на которые разделены 8 байт конфигурационного пакета.

смещение	поле	размер	описание
0	bmRequestType	BYTE	тип запроса
1	bRequest	BYTE	код запроса
2	wValue	WORD	параметр запроса
4	wIndex	WORD	индекс или смещение
6	wLength	WORD	число байт для передачи

Биты поля bmRequestType:

- бит 7 - если сброшен, то данные запроса будут передаваться в USB устройство, если установлен, то в обратном направлении. Когда управляющая передача не имеет стадии данных, то этот бит равен 0;
- биты 6..5 – задают тип запроса, 0 – стандартный, 1 – для класса, 2 – специальный, 3 – зарезервировано;
- биты 4..0 – получатель, 0 – устройство, 1 – интерфейс, 2 – точка, 3 – другой;
- биты 4..31 – зарезервированы.

Поле bRequest, задаёт код запроса. Интерпретация значения этого поля зависит от типа запроса заданного в поле bmRequestType. Если запрос стандартный, то здесь задаётся номер одного из 11 стандартных запросов. Если запрос специальный изготовителя, то значение задаёт и интерпретирует изготовитель. Запросы классов мы не рассматриваем.

Поле wValue, задаёт параметр, который может быть передан USB устройству вместе с запросом. Например, для стандартного запроса установки адреса (SET_ADDRESS 05h), здесь передаётся адрес USB устройству, число от 0-127. Адрес, назначаемый USB устройству в процессе энумерации. Если разработчик использует не стандартный запрос, то в этом поле можно задать свою информацию.

Поле wIndex, значение зависит от типа запроса.

Поле wLength, задаёт количество байт данных передаваемых в стадии данных, если такая предусмотрена запросом.

На рисунке 11 представлена управляющая передача чтения, имеющая стадию данных. Такие передачи используются и для приёма дескрипторов от USB устройства. Транзакций стадии данных может быть несколько, если требуется передать значительный объём информации. После стадии установки (Setup Stage) , хост начинает стадию передачи данных (Data Stage). Маркерный пакет IN от хоста, затем пакет полезных данных типа DATA1 от USB устройства и пакет квитирования от USB устройства. После стадии данных, наступает черёд стадии передачи состояния.

Хост посылает маркерный пакет OUT. Если USB устройство одобряет действия хоста, то возвращает пакет данных типа DATA1 нулевой длины (ZLP). Если USB устройство занято и не

может сию минуту обработать запрос, возвращает NAK. Если запрос противоречит текущему состоянию устройства, возвращает STALL. При принятии повреждённого пакета, стандартная реакция, показано как ERROR .

На рисунке 12 представлена управляющая передача записи, имеющая стадию данных. Передача используется для посылки значительного количества данных в запросе к USB устройству. Из рисунка очевидна логика работы.

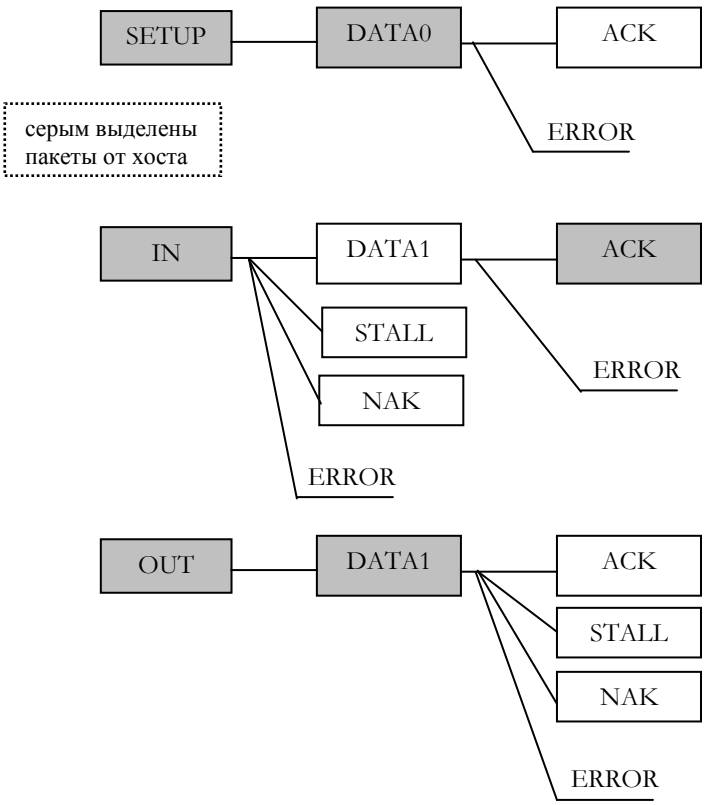


Рисунок. 11

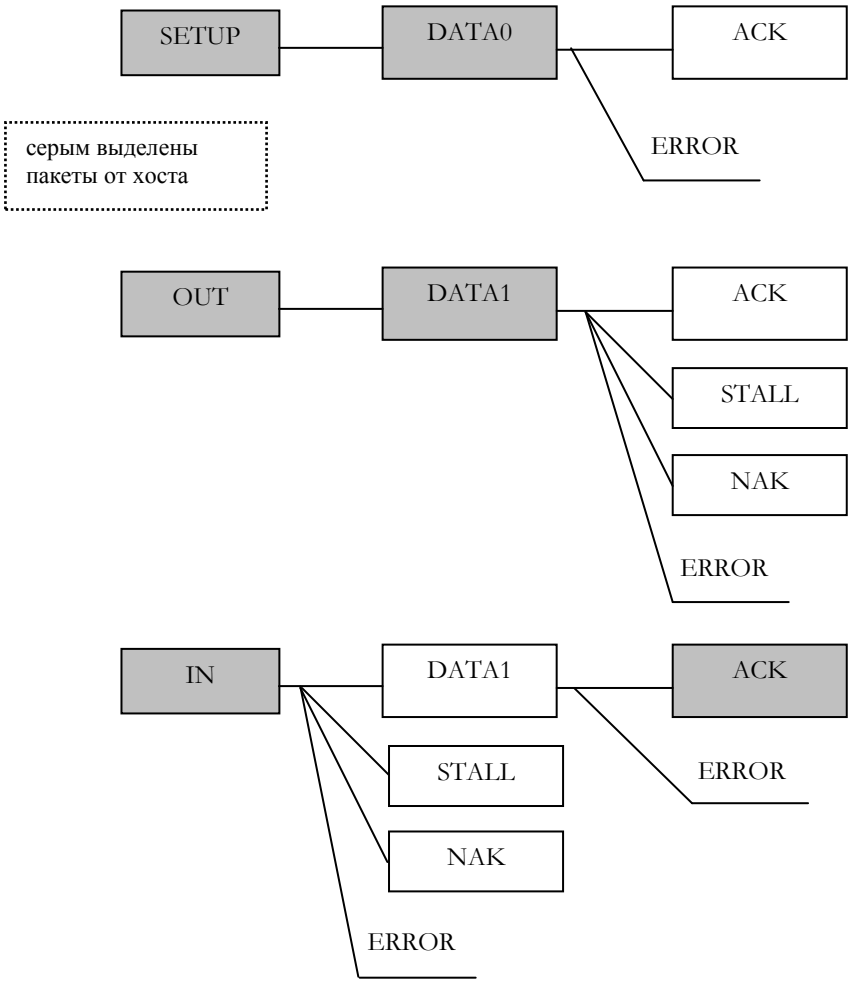

SETUP
DATA0
ACK
ERROR
серым выделены
пакеты от хоста
OUT
DATA1
ACK
STALL
NAK
ERROR
IN
DATA1
ACK
STALL
ERROR
NAK
ERROR

Рисунок 12

Распределение энергии

Шина USB создавалась в соответствии с концепцией ACPI (Advanced Configuration and Power Interface) – расширенный интерфейс конфигурирования и питания. ACPI представляет собой довольно сложную комбинацию средств. ACPI – позволяет операционной системе управлять конфигурированием и энергопотреблением устройств и компьютера в целом и описывает и программный и аппаратный интерфейсы. Поэтому режимы энергопотребления USB устройства имеют несколько усложнённый вид.

Спецификация USB позволяет устройству получать энергию прямо от шины, и это имеет свои очевидные преимущества. Но имеются и некоторые сложности, которые нужно учитывать. По стандарту, любое USB устройство с питанием от шины, может потреблять не более 500 мА. Это ограничение касается не усреднённого значения за период, а пикового значения. Потребляемый ток должен быть заявлен в процессе энумерации USB устройства. Но для энумерации на микроконтроллер уже должна поступать энергия, получается замкнутый круг. Поэтому по умолчанию на микроконтроллер, при подключении должно подаваться не менее 100 мА, а что сверху, то после энумерации.

Так как современные операционные системы поддерживают ACPI, то режимы питания шины USB зависят от программного и аппаратного обеспечения хоста. По моему мнению, питание от шины подходит для гаджетов, а не для ответственных приборов. Серьёзные приборы лучше питать от собственных источников, с индикацией состояния на собственную панель.

Физически – шина USB состоит из 4-х проводников:

- VBUS – линия питания;
- GND – нулевой провод;
- D- и D+ - линии данных.

В составе USB устройства условно можно выделить подсистему, выполняющую коммуникационные функции протокола USB и подсистему, выполняющую основные функции, например оцифровки аналогового сигнала. Уже говорилось, что имеется 3 типа USB устройств по типу подачи энергии:

- ток от шины (Low powered),
- повышенный ток от шины (High powered),
- с собственным источником тока (Self powered).

Рассмотрим USB устройство, имеющее собственный источник питания. Схема приведена на рисунке 13. «БО» – это блок обработки, выполняющий основную функцию. Квадрат с «USB» – это подсистема, отвечающая за обмен по USB. Квадраты «Рег» - это стабилизаторы. «ИП» - локальный источник питания USB устройства.

При такой схеме питания, возможны 2 варианта организации подачи тока. Можно подавать ток в подсистему USB от шины VBUS, через регулятор Рег1, а БО питать от внутреннего источника питания.

Второй вариант, питание всех подсистем от внутреннего источника. Подача энергии на подсистему USB отмечена штриховой линией, а регулятор Рег1 может отсутствовать. В первом случае, инициализация процесса энумерации происходит при подаче питания на подсистему USB, через регулятор Рег1. Во втором случае, когда Рег1 отсутствует, для обнаружения подключения и отключения к хосту, необходимы дополнительные цепи.

Требуется постоянное сканирование присутствия шины VBUS. Напряжение с VBUS, через делитель подаётся на один из портов ввода-вывода микроконтроллера и производится чтение состояния порта. Когда на контакте обнаруживается 1, начинается процесс энумерации.

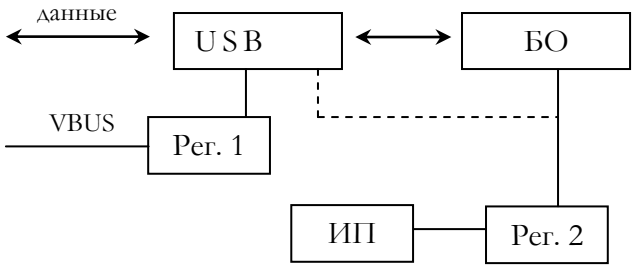


Рисунок 13

На рисунке 14 изображена конфигурация USB устройства с высоким потреблением энергии и питанием от шины. Питание на такую функцию подаётся в 2 ступени. Первоначально подаётся не более 100 мА на USB, а затем после энумерации подаётся требуемый ток, но не больше 500 мА. . Ток подаётся с хоста, от хаба. А команду хабу, на выдачу требуемой энергии, подаёт операционная система.

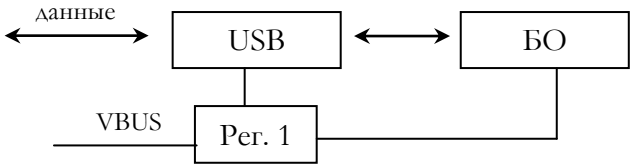


Рисунок 14

Подача тока на USB устройство с малым потреблением энергии, самая простая, в одну ступень. Все цепи питаются от VBUS.

Проще понять архитектуру питания AT90USB162, соотнеся её со стандартами USB. На рисунке 15, приведена схема питания:

- VCC- цепь питания цифровой подсистемы,
- UVCC – вход внутреннего регулятора напряжения USB,
- UCAP – выход внутреннего регулятора USB, должен подключаться конденсатор 1МкФ,
- AVCC – цепь питания аналоговой подсистемы,
- UVSS, VSS – земля.

Видно, что AT90USB162 позволяет реализовать, все режимы питания, описанные в стандарте. Для подсистемы USB требуется 3,3 В. Если проектировщик решит не использовать ток от VBUS, а использовать собственный источник 5В, тогда напряжение подаётся на UVCC и задействуется встроенный регулятор на 3,3В. А на цепи VCC и AVCC подаётся 5В. Дополнительно требуется сканирование линии VBUS.

Если на микроконтроллер подаётся напряжение 3,3В от внешнего источника, тогда внутренний регулятор программно отключается, управляет включением регистр REGCR(0x63), BIT0 – REGDIS. Для отключения регулятора, этот бит нужно установить. После сброса бит

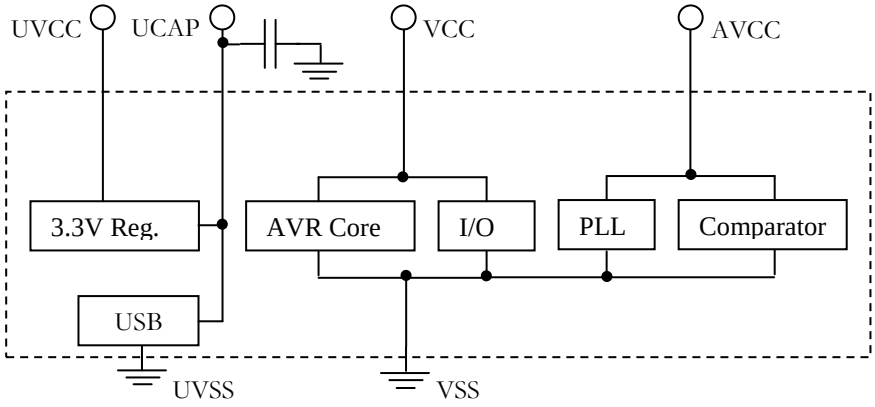


Рисунок 15

обнуляется, регулятор включен. Внутренний стабилизатор можно отключать, для избежания потерь энергии в режимах с низким потреблением.

В случае питания от VBUS, все цепи, UVCC, VCC, AVCC подключаются к этой линии.

Следует заметить, что только хаб имеет право подавать напряжение на VBUS. Подача напряжения от USB устройства запрещена.

Именно хаб управляет сегментом шины с подключенным USB устройством. По спецификации USB, хаб может сигнализировать USB устройству о необходимости перехода в состояние с пониженным потреблением энергии. Это выражается в отсутствии пакетов SOF в течение 3 мс, затем в течение 10 мс ещё поддерживается нормальный ток на шине VBUS. За эти 10мс USB устройство должно перейти в режим с низким энергопотреблением или переключиться на другой источник. Для устройств получающих энергию от шины с повышенным потреблением энергии, максимальный ток ограничивается в 2,5 мА, а для устройств с низким потреблением и того меньше, всего 0,5 мА. Практически хаб может давать до 2-3 мА, не отключаясь совсем. Устройства, имеющие собственный источник энергии, сами решают, как им действовать в случае потери активности шины.

Если устройство способно подавать сигнал удалённого пробуждения (Remote Wakeup), то USB устройство может разбудить хост и проснувшийся хост переведёт порт хаба в активное состояние, тогда USB устройство может нормально общаться с хостом. Сигнал удалённого пробуждения, один из немногих, которые может посылать USB устройство хосту по своей инициативе. Конечно, хост может перевести сегмент шины в активное состояние и по своей инициативе. Каким образом, USB устройство подаёт сигнал пробуждения хабу, рассмотрим в следующих главах.

Ближе к проводам

Как уже сообщалось, шина USB состоит всего из 4 проводников:

- V-BUS шина питания
- D+ и D- сигнальные шины
- GND сигнальное заземление.

На стандартном штекере кабеля, контакты V-BUS и GND длиннее сигнальных, это позволяет подать питание раньше подключения сигнальных линий. Основная масса информации передаётся с использованием дифференциальных приёмопередатчиков. Сигнал передаётся по линиям в противофазе, по витой паре. Потенциал помехи добавляется к сигнальному напряжению на обоих проводах в равной мере, не влияя на разность напряжений. Применение дифференциальных приёмопередатчиков существенно повышает помехозащищённость. Дифференциальный передатчик способен выдавать на линии не только сигналы в противофазе, но и когда обе линии данных под высоким потенциалом или наоборот, когда обе линии под низким. Такие режимы не характерны для классических дифференциальных передатчиков. Кроме дифференциальных приёмников и передатчиков имеются устройства позволяющие измерять напряжения на каждой из сигнальных шин в отдельности, а не только разностный сигнал. Схема приёмопередатчика FS (Full speed) приведена на рисунке 16. Здесь можно увидеть дифференциальные передатчики нисходящего порта хаба и USB устройства, а так же линейные приёмники. Резисторы R5 и R6 по 15 килоом постоянно подключены к нисходящему порту хаба. Резистор R7 используется для целей энумерации, этот резистор спрятан в микроконтроллере AT90USB162 и подключается программно. Резисторы R3 и R4 внешние по 22 ом, для согласования и защиты от перегрузок, номинал задан в спецификации на микроконтроллер. Номинал резисторов R1 и R2, зависит от конструкции хаба и служат для целей согласования и защиты от перегрузок.

Для передачи по линиям шины цифровой сигнал кодируется специальным линейным кодом. Вообще, при передаче данных по линиям связи применяются так называемые линейные коды. Простейший из известных – это униполярный NRZ (non return to zero). Логической 1 соответствует положительный импульс, а логическому 0 отсутствие. Биполярный NRZ - используется в RS-232, логическому 0 соответствует положительный импульс, а логической 1, отрицательный. Из необычных линейных кодов, можно упомянуть Манчестер II, логическая 1 кодируется отрицательным перепадом напряжения, а логический 0 кодируется положительным перепадом. Этот тип линейного кодирования используется в Ethernet.

Шина USB использует код NRZI (non return to zero, inverted). Поток бит в коде униполярный NRZ, кодируется в NRZI и подаётся на дифференциальный передатчик. На приёмной стороне, сигнал с дифференциального приёмника раскодируется обратно в NRZ. У кода NRZI, есть существенный недостаток, если на вход кодера подаётся длинная последовательность единиц, то возможна потеря синхронизации с приёмником. С этим борются техникой вставки дополнительных бит (bit stuffing), если входящий поток бит оказался больше, чем из 6 единиц, то вставляется ноль, а на приёмной стороне этот ноль удаляется. Кодирование в NRZI и обратно происходит чисто аппаратными средствами. Рисунок 17 иллюстрирует принцип кодирования NRZI. При передаче логической 1, напряжение не меняется, а при каждом логическом 0, происходит переход на противоположный уровень. Длительность битового интервала для FS режима составляет 83,3 нсек.

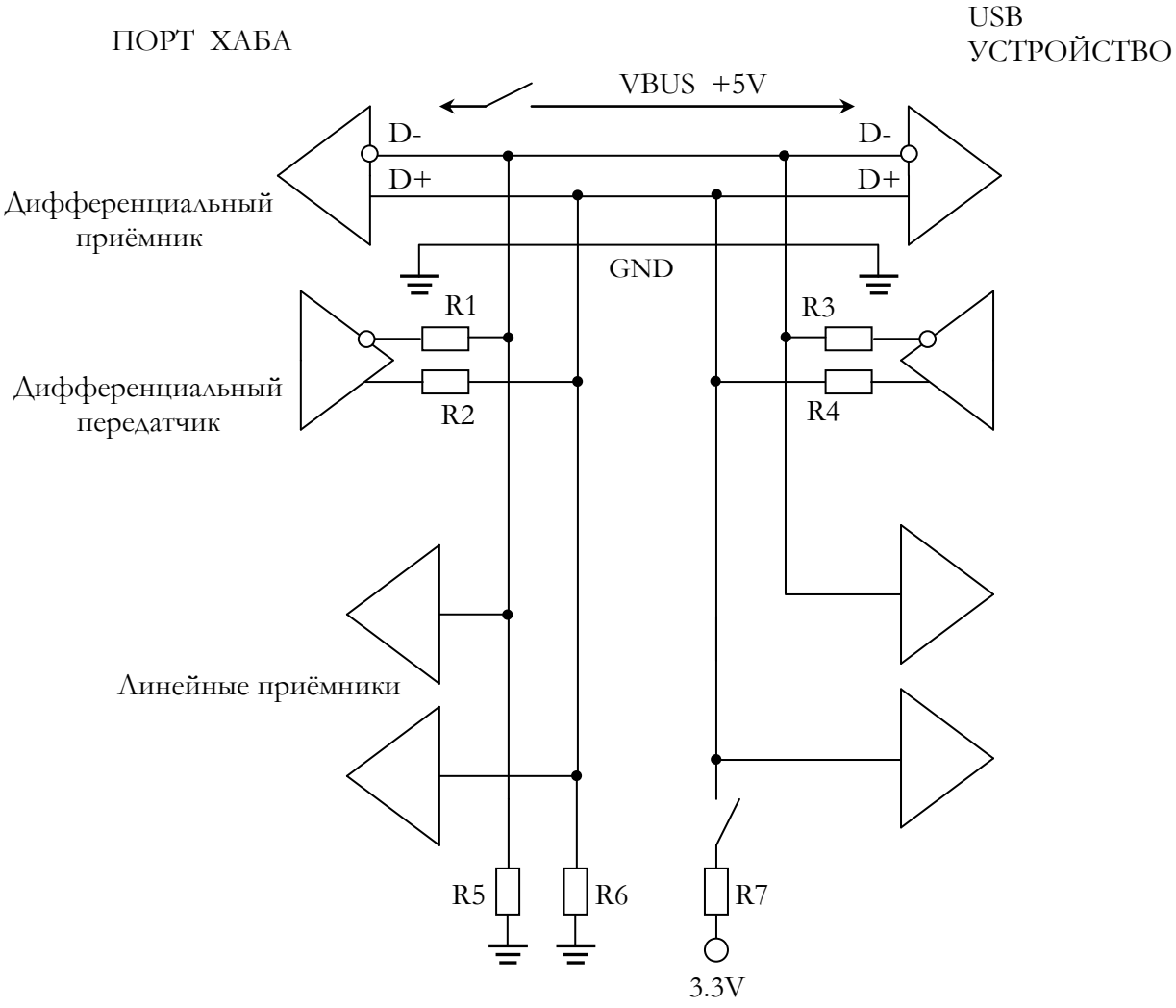


Рисунок 16

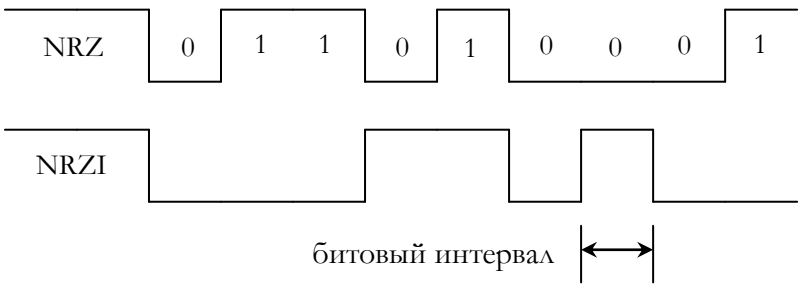


Рисунок 17

Состояние шины, когда потенциал линии приёмника D- низкий, а D+ высокий - это состояние называется дифференциальная единица. А когда противоположная ситуация, то дифференциальный ноль. Проще говоря, когда D+ в плюсе, а D- в минусе, то это дифференциальная единица или состояние J (Data J state) для режима FS. Противоположное состояние называется состояние K (Data K state) для режима FS.

Состояние, когда потенциал обоих линий меньше 0,8 вольта – называется SE0 (Single-Ended Zero). Проще говоря, когда обе линии на нуле – SE0 . Когда описывалась структура пакета, то упоминался EOP (End of Packet) конец пакета, конец пакета отмечается состоянием SE0 в течение 2-х битовых интервалов. Напомним, что для режима FS (Full Speed) длительность одного битового интервала составляет 83,3 нсек.

Состояние, когда потенциал обоих линий приёмника на высоком уровне, больше 2 вольт – называется SE1 (Single-Ended One). Такое состояние не может занимать исправная шина в процессе нормальной работы. Только под действием помехи или неисправности возможно состояние SE1.

Состояние, когда шина длительное время, больше 3 миллисекунд, находится в состоянии J, называется состояние покоя (Idle state). В такое состояние шина переходит, когда хост сигнализирует USB устройству о необходимости перехода в состояние с низким потреблением энергии. Ранее уже описывалось.

Энумерация

Выше говорилось, что USB устройство может находиться в процессе конфигурирования (энумерации) и в рабочем режиме. В рабочем режиме хаб «прозрачен» для USB устройства и выполняет функцию «невидимого» посредника. В процессе энумерации хаб перестаёт быть «невидимым» и активно участвует в энумерации, поэтому кратко затронем работу хаба.

Хост отвечает за обнаружение и конфигурирование всех устройств появляющихся на шине. Посредником между хост-контроллером и USB устройством является хаб или цепочка хабов. Хаб имеет нулевую конечную точку, для приёма управляющих запросов от хост-контроллера и конечную точку прерывания типа IN. Имеется специальная система запросов, предназначенная для хаба. Хост-контроллер управляет хабом, посылая запросы и принимая данные. В хабе имеются нисходящие порты и один порт восходящий. К нисходящим портам хаба подключаются сегменты шины для USB устройств. А восходящий всегда подключается к другому хабу, либо к хост-контроллеру. Нисходящие порты хаба могут быть в нескольких состояниях:

- не запитан (Not powered) - на линии VBUS отсутствует напряжение. Такой порт не способен к взаимодействиям с подключаемым сегментом шины. В это состояние порт может перейти по управляющему запросу хоста (команде с хоста), либо при токовой перегрузке линии VBUS;
- не подключен (Disconnected) - в этом состоянии порт хаба способен обнаружить вновь подключаемое USB устройство или хаб. Напряжение на VBUS присутствует;
- запрещён (Disabled) - трафик через этот порт не транслируется. Переходит в это состояние для изоляции ошибочно работающих устройств на сегменте;
- разрешён (Enabled) – рабочее состояние порта;
- приостановлен (Suspended) – в это состояние порт переводится в случае необходимости перевода устройства на сегменте шины в «спящее» состояние, переводится по специальному запросу хоста.

Для обнаружения вновь подключаемого USB устройства к порту хаба, этот порт должен находиться в состоянии Disconnected. Хаб сканирует свои порты, следя за возможным подключением и отключением USB устройств. Когда к порту хаба не подключено USB устройство, проводники линий данных заземлены через резисторы 15 Ком, а на линии VBUS присутствует 5 вольт. Шина находится в состоянии SE0. В свою очередь, USB устройство обнаруживает хаб, по присутствию напряжения на линии VBUS. Выдержав некоторую паузу после появления напряжения, для стабилизации уровня и устранения дребезга, USB устройство заявляет о своём подключении, как устройстве типа FS(Full speed), повышением напряжения на линии D+ на 2 вольта. Микроконтроллер AT90USB162 имеет встроенный резистор, который подключает линию D+ к источнику 3,3 V через управляемый ключ. Об этом резисторе уже говорилось при обсуждении приёмопередатчиков. Совместно с резисторами, подключенными на стороне хаба, образуется делитель напряжения, ток через который повышает напряжение на линии D+ , переводит шину в состояние покоя (Idle State).

Теперь должен ответить хост на действия USB устройства. Хост обнаруживает, опрашивая хаб, что к порту подключено USB устройство. Хост-контроллер переводит порт из состояния Disconnected в состояние Enabled, теперь через порт можно передавать управляющие запросы к USB устройству.

После обнаружения, что сегмент шины перешёл в состояние Idle state, благодаря подключенному резистору, выждав не мене 100 миллисекунд, хост посылает управляющий запрос хабу, выполнить сброс сегмента шины, к которому подключено USB устройство. Хаб выполняет команду (Bus Reset), от 10 миллисекунд до 20 миллисекунд линии D+ и D- имеют потенциал на

уровне 0 вольт. Другими словами, шина приводится в состояние SE0, на 10-20 миллисекунд. По этой команде, подсистема микроконтроллера, отвечающая за коммуникации на шине USB (SIE), должна выполнить сброс своих регистров и флагов. Нужно понимать, что речь идёт только о подсистеме USB, а не о микроконтроллере. После команды (Bus Reset), USB устройство должно перейти в «дежурное» состояние (Default state). В этом состоянии USB устройство имеет нулевой адрес и инициализированную нулевую конечную точку. Сейчас USB устройство может потреблять не более 100 миллиампер от шины и способно принимать запросы от хоста. На рисуноке 18 , приведены временные диаграммы.

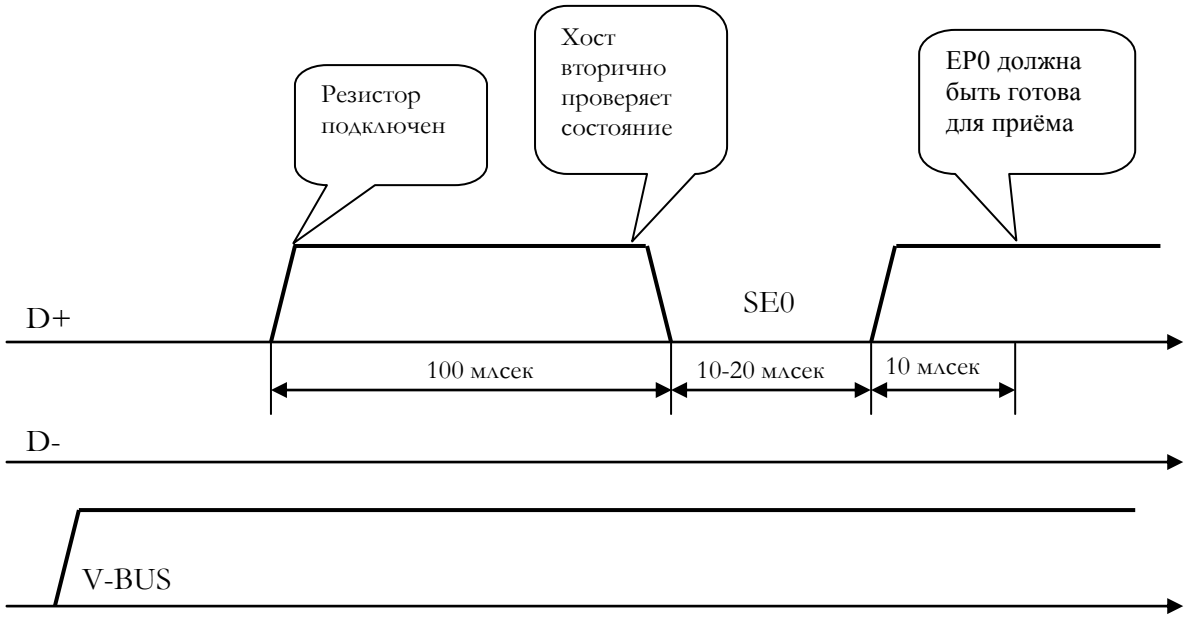


Рисунок 18

Сообщалось, что USB устройству не может быть постоянно назначен нулевой адрес, этот адрес может использоваться в процессе энумерации. Так как USB устройство уже способно принимать управляющие запросы, хост своим стандартным управляющим запросом Get_Device_Descriptor, запрашивает дескриптор USB устройства, точнее его первые 8 байт, для определения максимального размера конечной точки управления. Конечная точка управления может иметь максимальный размер из ряда: 8,16,32,64 байта. Стандартные управляющие запросы уже упоминались ранее. Затем хост посылает стандартный запрос Set_Address, назначает уникальный адрес USB устройству. Теперь USB устройство находится в состоянии «Адресовано» (Adressed state).

Хост вторично выдаёт запрос Get_Device_Descriptor, теперь запрашивается дескриптор устройства целиком, а не 8 байт. Затем запрашивается дескриптор конфигурации и все подчинённые дескрипторы: дескриптор интерфейса и дескрипторы конечных точек, если их несколько. Получив дескрипторы от USB устройства, хост подбирает драйвер, на основе имеющейся информации.

Далее хост посылает управляющий запрос Set_Configuration, ставя точку в процессе энумерации, теперь USB устройство в состоянии «Сконфигурировано» (Configured State). Здесь дана несколько упрощённая схема, что конфигурация одна и интерфейс один. А вообще-то хост может выбирать между несколькими конфигурациями, информацию о которых он получил с дескрипторами и утвердить одну наиболее подходящую для его возможностей. Но как ранее говорилось, USB устройства с несколькими конфигурациями здесь не рассматриваются.

В процессе работы USB устройство переходит из одного состояния в другое, в стандарте USB имеется граф переходов, по которому наглядно видно как переходит устройство из одного состояния в другое в случае возникновения событий. На рисунке 19 представлен этот граф.

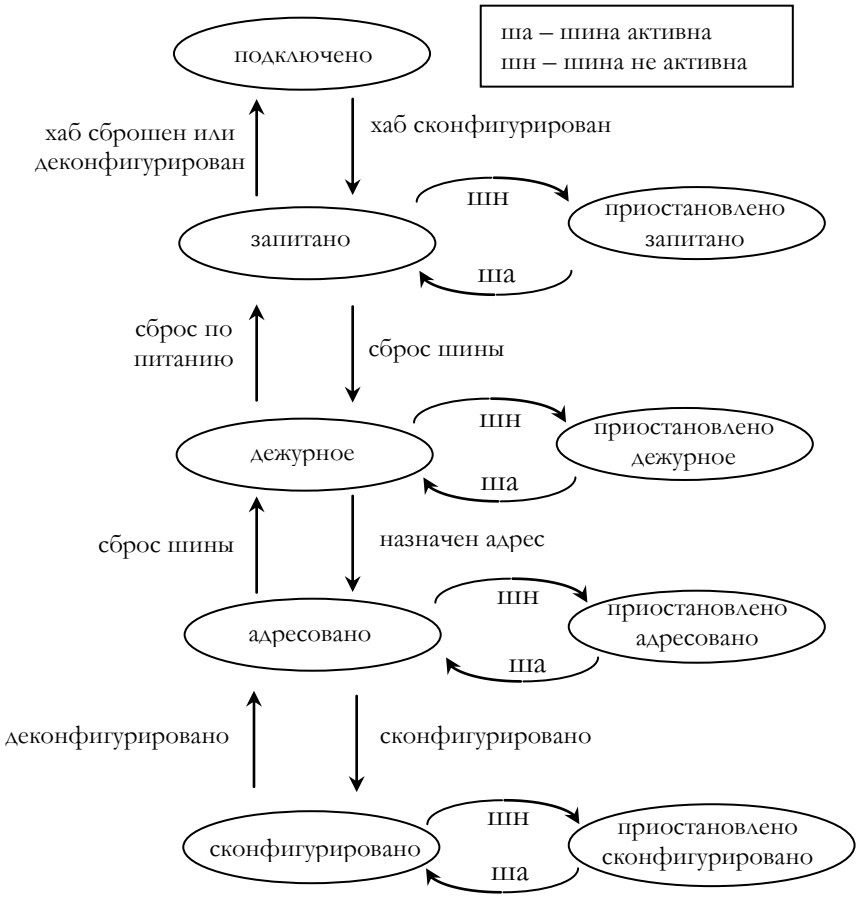


Рисунок 19

Состояние «приостановлено» - уже рассматривалось ранее. Состояние «подключено» - когда USB устройство физически соединено, но напряжение на VBUS не подано. Такая ситуация редко встречается на практике. А вообще-то, при подключении USB устройства, напряжение уже имеется на VBUS, поэтому USB устройство сразу переходит в состояние «запитано» (Powered). Программное обеспечение USB устройства, должно постоянно вести мониторинг возможных событий, а не только в процессе энумерации. Из состояний: «адресовано», «сконфигурировано», «приостановлено», по сигналу «Bus Reset», USB устройство должна переходить в состояние «дежурное».

Переход из состояния «сконфигурировано» в состояние «адресовано», может происходить при изменении текущей конфигурации. К примеру, USB устройство имеет несколько конфигураций, одна конфигурация для устройств с собственным источником питания, а другая с питанием от шины. Тогда для переключения текущей конфигурации, необходимо перейти в состояние «адресовано», а затем по управляющему запросу Set_Configuration, выбрать другую конфигурацию. У запроса Set_Configuration имеется числовой параметр wValue – это номер конфигурации, которую требуется установить. Когда этот код равен нулю, USB устройство переходит в состояние «адресовано», чтобы затем перейти в состояние «сконфигурировано», при получении нового управляющего запроса Set_Configuration с нужным номером конфигурации.

Энумерация – часть вторая

Событийно-ориентированное программирование (event driven programming) – это то, что нужно, что бы лучше понимать работу программ микроконтроллера с аппаратной поддержкой USB. Иначе такие программы называют – программы управляемые событиями. Логика программы реализуется, как максимально изолированные обработчики спонтанных событий. Во многих объектно-ориентированных средах разработки – эта парадигма реализована. При программировании микроконтроллеров, такими обработчиками событий могут быть обработчики прерываний, а основная программа может выродиться в загрузчик пробелов, выполнять пустой цикл. Кончено обработчики прерываний общаются между собой через доступную всем обработчикам область памяти. Особенно эффективно решаются задачи, алгоритм работы которых описывается конечным автоматом. Часто такие задачи возникают при описании коммуникационных протоколов и работы технологических механизмов.

Иногда события, после первичной обработки направляют в единственный обработчик событий, который занимается их разделением и обработкой. Граф переходов между состояниями USB устройства, в процессе функционирования, напрашивается сам собой на анализ, с точки зрения парадигмы событийно-управляемого программирования.

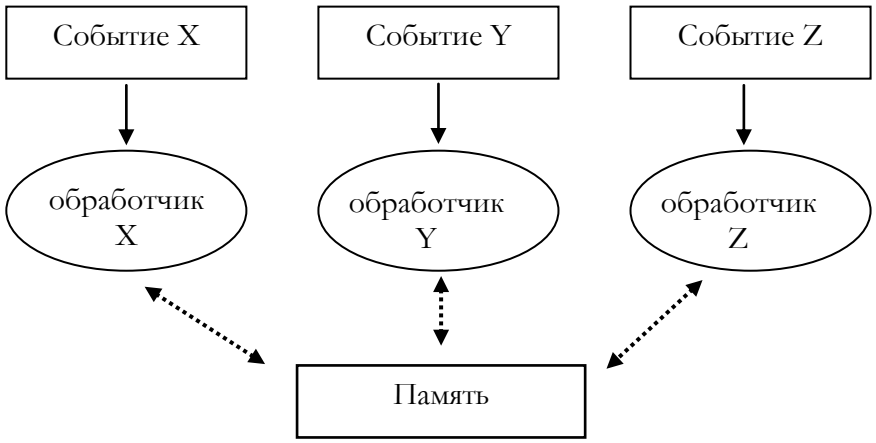


Рисунок 20

Выше говорилось о процессе энумерации, но процесс энумерации только часть жизненного цикла устройства на шине, события могут возникать спонтанно в любое время, и USB устройство должно корректно на них реагировать.

Необходима глобальная переменная, в которой будет храниться текущее состояние USB устройства. При возникновении события, обработчик определит текущее состояние, по значению глобальной переменной, сделает необходимые действия и установит новое значение этой переменной в соответствии с новым состоянием USB устройства.

Выполнять мониторинг событий можно различными методами. Организовать как обработчик прерываний, или опрашивать состояние регистров, или по прерыванию таймера вызывать монитор и опрашивать регистры.

Событие	Реакция
USB устройство обнаруживает напряжение на линии VBUS	USB устройство подключает резистор к D+
Хост посылает команду «Сброс шины» (Bus Reset)	USB устройство, настраивается на приём управляющих запросов на точку управления по нулевому адресу
Хост посылает команду, выдать первые 8 байт дескриптора устройства	USB устройство посылает 8 байт дескриптора
Хост назначает адрес	USB устройство адрес устанавливает
Хост запрашивает все дескрипторы	USB устройство эти дескрипторы выдаёт
Хост назначает конфигурацию.	USB устройство активизирует конфигурацию и готово к работе.

Перечислим события шины, которые должны вызывать переход USB устройства из одного состояния в другое.

Переход напряжения из «0» в «1» на шине VBUS. При подключении USB устройства к хосту. Переходит из состояния «подключено» в состояние «запитано».

Переход напряжения из «1» в «0» на шине VBUS. Из любого состояния переходит в состояние «подключено».

Команда «сброс шины» (Bus Reset). Из любого состояния, кроме «подключено», переходит в состояние «дежурное».

Хост перевёл сегмент шины с USB устройством в состояние «приостановлено» (Suspended state), состояние шины Bus Idle более 3 миллисекунд. Переход в состояние «приостановлено» возможен из любого состояния, кроме «подключено».

Хост выводит USB устройство из состояния «приостановлено». Хост посылает специальную команду «пробуждение» (Resume). Суть команды, из состояния J шина переводится в состояние K на 20 миллисекунд, и затем в состояние SE0 на 1,33 микросекунд и опять в состояние J на 0,66 мксек. На рисунке 21 представлена временная диаграмма. После этой команды USB устройство должно быть готово к приёму пакетов SOF. По этой команде происходит возврат в активное состояние, из которого USB устройство перешло в состояние «приостановлено».

Нужно вспомнить, что USB устройство может по своей инициативе сообщить хосту о необходимости пробуждения. Устройство USB выводит хост из спящего состояния командой «удалённое пробуждение» (Upstream Resume), переводит шину на интервал от 1 до 15 миллисекунд в состояние K (K-state), перед этим шина должна быть в состоянии J как минимум 5 миллисекунд.

Команда достигает хоста и он в свою очередь отвечает командой «пробуждение» направленной к USB устройству. Команда «пробуждение» от хоста описана выше.

Стандартные управляющие запросы, должны корректно обрабатываться USB устройством. Некоторые из них выполняют функцию событий, приводящих к изменению состояния USB устройства.

На управляющий запрос Get_Device_Descriptor по нулевому адресу, в состоянии «дежурное», USB устройство остаётся в дежурном состоянии, выполнив посылку первых 8 байт дескриптора устройства хосту.

Управляющий запрос Set_Address назначает адрес, эффективен только в состоянии «дежурное», в остальных состояниях, должен рассматриваться как некорректный. Событие переводит USB устройство в состояние «адресовано» и присваивает адрес.

На управляющий запрос Get_Device_Descriptor , USB устройство посылает дескриптор устройства целиком, оставаясь в состоянии «адресовано». Затем хост по меньшей мере ещё два раза

посылается этот управляющий запрос, для получения дескриптора конфигурации и дескрипторов конечных точек. После приёма дескрипторов USB устройство остаётся в состоянии «адресовано».

На управляющий запрос Set_Configuration, с ненулевым номером конфигурации, USB устройство переходит из состояния «адресовано» в состояние «сконфигурировано».

На управляющий запрос Set_Configuration , с нулевым номером конфигурации, если USB устройство в состоянии «сконфигурировано», переходит в состояние «адресовано».

Программное обеспечение контроллера, должно так же распознавать события по каналам передачи данных. Когда по инициативе хоста, USB устройство обработает транзакцию, то генерируется событие, которое можно условно назвать «транзакция данных обработана». Организовать обработку события можно либо опросом флагов, либо обработкой прерывания. Транзакции по каналам данных обрабатываются аппаратурой SIE. Да и по каналам управления транзакции обрабатываются аппаратурой SIE.

Твёрдое знание протокола рабочего режима и процесса энумерации очень важно для

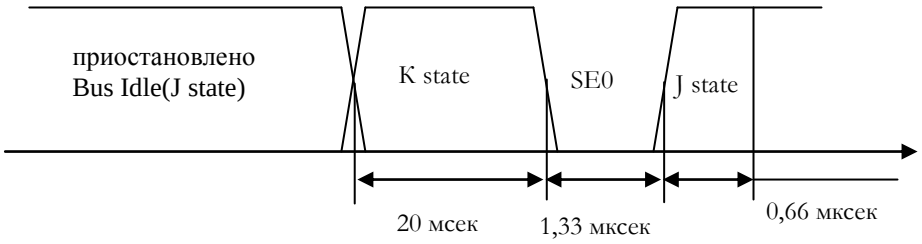


Рисунок. 21

понимания назначения регистров SIE микроконтроллера и построения программы микроконтроллера.

Огромную помощь в изучении шины USB может оказать анализатор протокола USB. Аппаратура такого анализатора подключается к шине, захватывает пакеты и демонстрирует на мониторе. Но такая аппаратура дорого стоит и не всегда доступна.

Другое подспорье – это программный анализатор, по своим возможностям он не сравним с аппаратным анализатором, но может оказаться очень полезным. Существует множество программных анализаторов. Мной использовалась свободная программа Wireshark (www.wireshark.org). Это программный анализатор широкого применения, чаще он применяется для анализа сетевого трафика, но можно использовать и для анализа активности USB устройств. Существуют порты как для Windows так и для Linux. Для Windows XP необходимо устанавливать версию не старше 1.10. Для Windows нужно дополнительно установить программу USBPcap, не путать с WinPcap. С Windows порядок работы следующий. После установки Wireshark, распаковываем архив с USBPcap и запускаем консольную программу USBPCapCMD.exe. Программа выводит список корневых хабов и устройств, подключенных к хабам в системе, и спрашивает, какой номер хаба выбираете, иначе говоря, какой слот будем прослушивать. Введите номер соответственный USB устройству. Затем просит ввести название файла, куда программа сбросит все захваченные данные. По Ctrl+C прерывается процесс прослушивания. Запускаем Wireshark и открываем полученный файл дампа. Можно вести мониторинг с помощью Wireshark и в реальном режиме, на родном сайте http://desowin.org/usbpcap/, приведён образец командной строки.

Для Linux нужно ядро не моложе 2.6.21 и загруженный модуль usbmon. Только начиная с версии 1.1.0 библиотеки libpcap, правильно работает Wireshark для Linux, поэтому проверьте версию библиотеки на вашей системе. Средства перехвата даннх встроены в ядро Linux и никаких дополнительных программ устанавливать не надо.

Чтобы захватить процесс энумерации, запускаем прослушивание нужного слота, а затем в этот слот вставляем штекер исследуемого USB устройства. Экран Wireshark с запросами процесса энумерации показан на рисунке 22.

Для Windows, захватом данных занимается USBPcap, а Wireshark используется для понятного представления данных. В свою очередь USBPcap состоит из 2-х частей. Одна часть – это драйвер, работающий в пространстве ядра операционной системы, и вторая часть – это консольная программа, работающая в пространстве пользователя. В стек USB ядра операционной системы, данные поступают от драйвера в виде пакетов URB (USB request blocks). Это особые структуры, которые в себе содержат данные запросов от хоста к USB устройству. Проще говоря, URB– это некий контейнер, содержащий запрос или транзакцию ввода-вывода. Программа USBPcap перехватывает URB и передаёт программе пользовательского уровня. Важно понимать, что Wireshark показывает URB и содержащиеся в них данные. Без труда можно разобраться, где запросы, а где транзакции ввода-вывода.

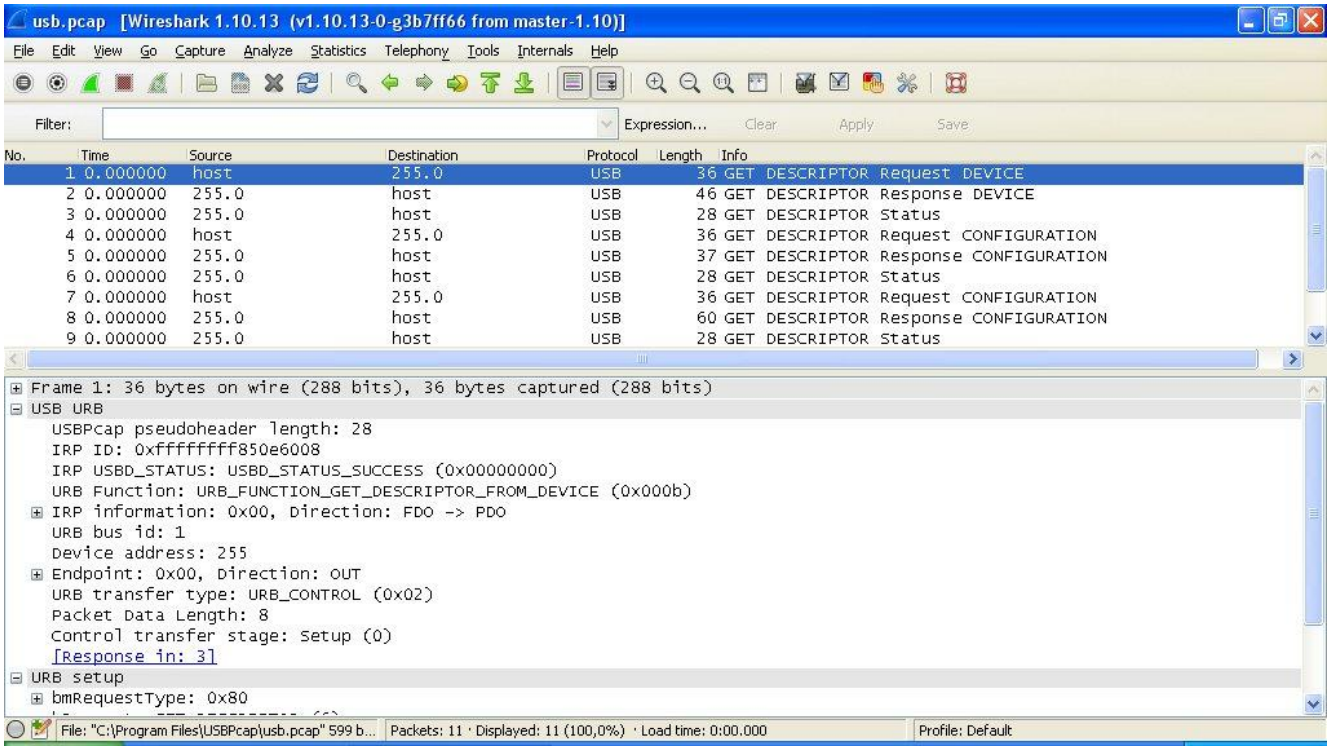
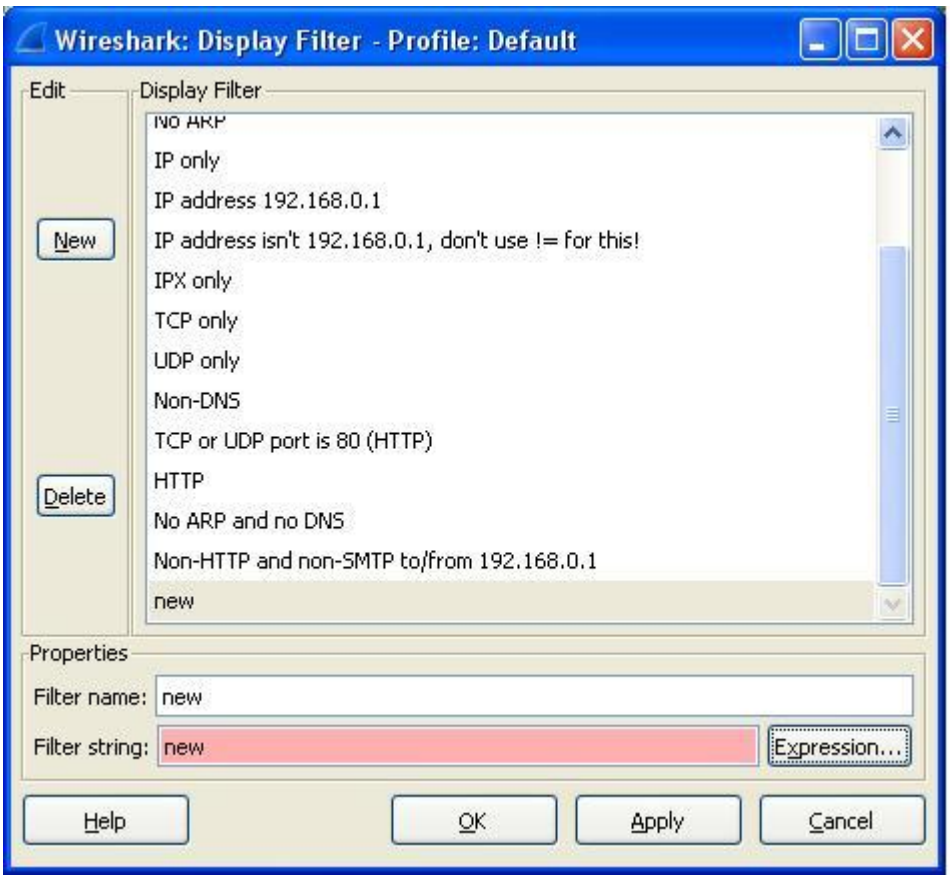


Рисунок. 22

Выводимые на экран данные можно фильтровать по различным признакам. Например фильтровать по адресу USB устройства. Из меню «Analyze -> Display filters». Создать новый фильтр «new», затем жмём кнопку «Expression». Появляется окно конструктора фильтров, выбираем секцию «USB» и необходимый атрибут, и в конце «Apply». На рисунке 23 показаны снимки экрана. Программа анализатора даёт наглядное представление о процессе энумерации. Важно на начальных этапах изучения шины USB, брать простые устройства. В качестве простейшего устройства для изучения шины, можно взять загрузчик DFU микроконтроллера AT90USB162. Об этом загрузчике сказано ниже.


Wireshark: Display Filter - Profile: Default
Edit
Display Filter
New
Delete
IP only
IP address 192.168.0.1
IP address isn't 192.168.0.1, don't use != for this!
IPX only
TCP only
UDP only
Non-DNS
TCP or UDP port is 80 (HTTP)
HTTP
No ARP and no DNS
Non-HTTP and non-SMTP to/from 192.168.0.1
new
Properties
Filter name: new
Filter string: new
Expression...
Help
OK
Apply
Cancel

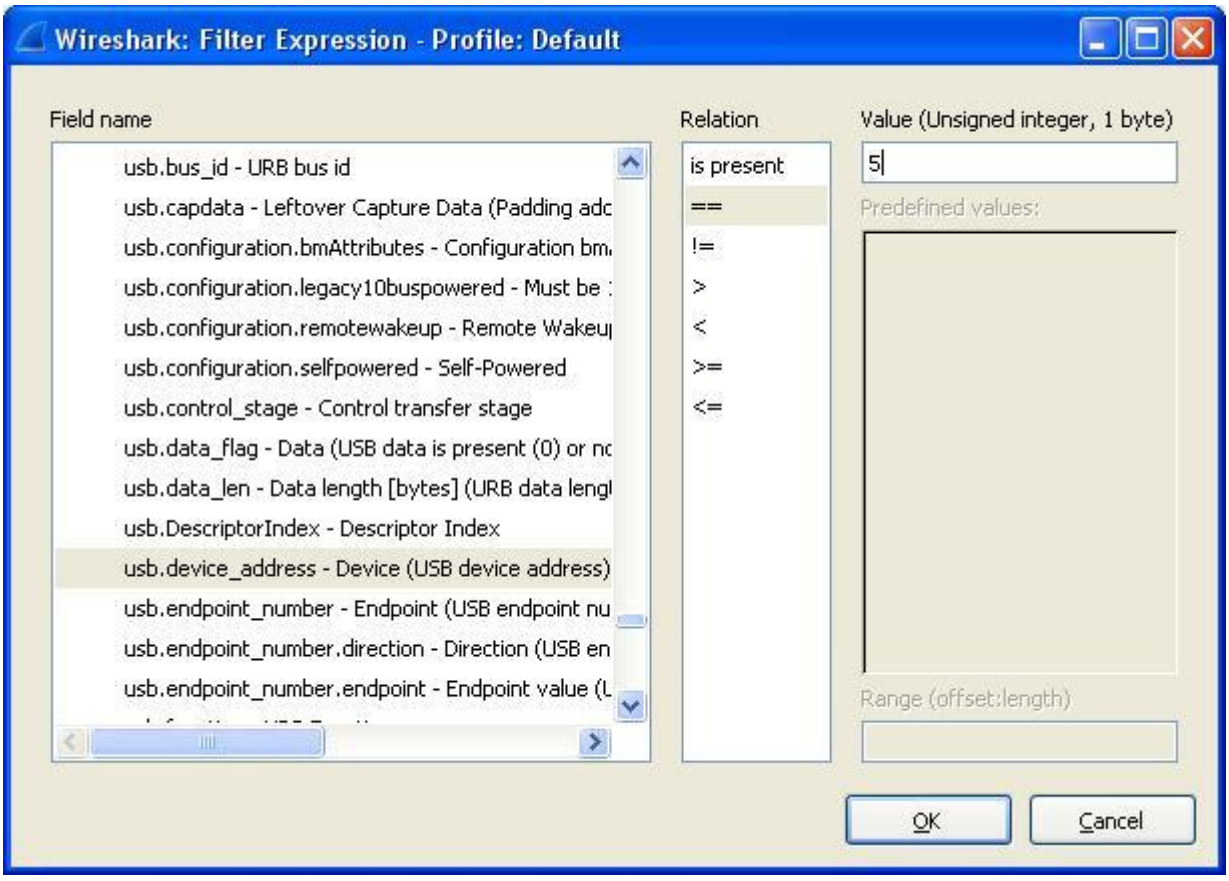

Wireshark: Filter Expression - Profile: Default
Field name
Relation
Value (Unsigned integer, 1 byte)
usb.bus_id - URB bus id
usb.capdata - Leftover Capture Data (Padding adc
usb.configuration.bmAttributes - Configuration bm
usb.configuration.legacy10buspowered - Must be
usb.configuration.remotewakeup - Remote Wakeu
usb.configuration.selfpowered - Self-Powered
usb.control_stage - Control transfer stage
usb.data_flag - Data (USB data is present (0) or nc
usb.data_len - Data length [bytes] (URB data lengt
usb.DescriptorIndex - Descriptor Index
usb.device_address - Device (USB device address)
usb.endpoint_number - Endpoint (USB endpoint nu
usb.endpoint_number.direction - Direction (USB en
usb.endpoint_number.endpoint - Endpoint value (U
is present
==
!=
>
<
>=
<=
5
Predefined values:
Range (offset:length)
OK
Cancel

Рисунок 23

Микроконтроллер AT90USB162

Предполагается, что микроконтроллер питается от собственного источника и сигнал с линии VBUS подаётся на один из портов через делитель напряжения. Делитель нужен, чтобы уменьшить потребление тока от линии VBUS. В состоянии «приостановлено», максимальный ток от линии VBUS сильно ограничен.

Напомню, что подсистему AT90USB162, отвечающую за коммуникации USB в дальнейшем будем называть SIE.

Микроконтроллер имеет развитые средства конфигурирования режимов с низким энергопотреблением, для упрощения изучения, подробно их рассматривать не будем.

Прерывания связанные с SIE делятся на 2 большие категории:

- прерывание вызванное функционированием всего SIE (вектор 12)
- прерывание вызванное функционированием конечных точек (вектор 13).

Используется всего 2 вектора прерывания для поддержки функционирования USB. Будем их называть, прерывание устройства и прерывание конечных точек. Для прерывания устройства, источник прерывания определяется опросом флагов в подпрограмме обработчика прерывания по 12-му вектору. Для конечных точек, источник прерывания определяется так же опросом в подпрограмме, но по 13-му вектору. Для уточнения, с какой конечной точкой произошло событие, нужно опросить регистр UEINT . В регистре UEINT, каждой конечной точке соответствует бит, который показывает, в какой конечной точке произошло прерывание. Вообще для манипулирования с регистрами конечной точки, эту точку необходимо предварительно выбрать. Выбор заключается в загрузке нужного значения в регистр UENUM биты 2:0 EPNUM.

Действия программы микроконтроллера на внешние события. Когда обнаружится напряжение на VBUS, необходимо:

- разрешить работу ФАПЧ,
- проверка срабатывания захвата ФАПЧ ,
- перевести SIE в состояние включено ,
- сконфигурировать конечную точку управления EP0 ,
- подключить резистор к линии D+ для режима FS.

Подключается резистор встроенный в микроконтроллер, сбросом бита 0 DETACH регистра UDCON. По умолчанию бит установлен в 1.

Отключать и включать ФАПЧ не обязательно, если контроллер питается от собственного источника и не требуется переводить микроконтроллер в режим с низким потреблением.

На захват частоты требуется значительное время во временном масштабе контроллера, поэтому проверка готовности ФАПЧ обязательна.

SIE может быть в 2-х состояниях, включено (enable) и выключено (disable) . Управляет состоянием бит 7 USBE регистра USBCON. В состоянии выключено, сигнальные передатчики работающие на шину отключены и не потребляют тока, тактирование SIE отключено и не потребляет тока. В состоянии выключено, SIE физически отключен от шины и не тактируется, поэтому не способен реагировать ни на какие события на шине. После отключения SIE состояние его регистров и флагов сбрасывается в состояние по умолчанию. Возникновение напряжения на VBUS является первым событием, в цепочке событий энумерации.

Когда напряжение на VBUS пропадёт, необходимо:

- отключить резистор от линии D+ ,
- перевести SIE в состояние отключено ,
- запретить работу ФАПЧ, для снижения потребления энергии.

Когда шина перейдёт в состояние «приостановлено» (Bus Idle), SIE микроконтроллера аппаратно распознает это состояние. Напомним, что в течение 3 миллисекунд перестают появляться пакеты SOF, когда это происходит, SIE аппаратно установит бит 0 SUSPI , регистра UDINT. Если разрешено прерывание, то может быть запущен обработчик. В случае обнаружения состояния «приостановлено», в документации на AT90USB162 рекомендовано выполнить ряд действий по снижению энергопотребления, но это не обязательно для функционирования устройства.

Когда хост пошлёт команду «пробуждение» (resume) , SIE аппаратно обнаружит эту команду от хоста на шине и установит бит 5 EORSMI , регистра UDINT . Если разрешено прерывание , то может быть запущен обработчик. Эта команда от хоста должна выводить контроллер из состояния «приостановлено».

Когда центральный процессор микроконтроллера находится в энергосберегающем режиме, то вывести его из этого режима может команда «пробуждение» от хоста, конечно если разрешено прерывание по этой команде, специально для процессора в спящем режиме. Управляет разрешением прерывания бит 4 WAKEUPE, регистра UDIEN, а устанавливается флаг WAKEUPI. Иначе говоря, этот источник прерывания относится к энергосберегающим режимам, которые рассматривать не будем. Здесь эти сведения приводятся, чтобы избежать путаницы. Если энергосберегающие режимы микроконтроллера не используются, то разрешать эти источники прерывания не надо и флаги не анализировать.

Устройство USB может послать по своей инициативе, по шине команду удалённого пробуждения, об этих командах уже говорилось. Предварительно SIE должно обнаружить, что шина находится в состоянии «приостановлено» и установлен бит SUSPI. Для посылки команды, нужно установить бит RMWKUP регистра UDCON, команда будет сгенерирована аппаратно и послана хосту. Как только начнётся выполнение этой команды, будет установлен бит 6 UPRSMI регистра UDINT, сигнализирующий о выполнении команды. Дополнительно, может быть сгенерировано прерывание, если установлен бит UPRSME регистра UDIEN. Одновременно, с началом выполнения команды, будет сброшен бит SUSPI . А по окончании выполнения команды, аппаратно будет сброшен бит RMWKUP.

При получении этой команды на пробуждение от USB устройства, хост в свою очередь пошлёт «нормальную» команду «пробуждение» (resume) , реакция на которую уже описывалась. В программе для микроконтроллера, способность USB устройства выдавать сигнал «удалённое пробуждение» не реализована. Информацию нужно принять к сведению и только.

Когда на шине устанавливается команда «сброс шины» (Bus Reset), то по окончании этой команды устанавливается бит 3 EORSTI регистра UDINT , а разрешается прерывание по этому событию, битом 3 EORSTE регистра UDIEN. Эта команда вызывает переход всех конечных точек, кроме точки управления в состояние «запрещено». У AT90USB162 имеется особенность, команда сброс шины может вызвать сброс ЦПУ микроконтроллера. Эта функция включается установкой бита 2 RSTCPU, регистра UDCON. Команда «сброс шины», играет ключевую роль в процессе энумерации.

Перейдём к рассмотрению запросов управления. На данном этапе повествования, запросы управления рассматриваются как события, вызывающие переход USB устройства из одного состояния в другое. Образно, эти запросы можно назвать макро событиями. Нужно вспомнить, что для передачи запроса управления хостом, используются управляющие передачи, которые уже описывались. Управляющие передачи обрабатываются аппаратно SIE и попутно могут генерировать прерывания и устанавливать флаги на различных стадиях выполнения. На некотором уровне абстракции, можно рассматривать запрос управления как событие, а если спуститься ниже, то нужно рассматривать как процесс. Чтобы двигаться, дальше, необходимо изучить обработку управляющих запросов и обработку транзакций записи и чтения к конечным точкам.

Транзакция записи. Исходное состояние, флаги RXOUTI и FIFOCON сброшены, в буфере конечной точки нет данных от хоста. От хоста поступает маркерный пакет OUT и пакет данных DATAx , транзакция успешно завершена, хосту послан пакет ACK. После успешного выполнения транзакции, устанавливаются флаги RXOUTI и FIFOCON. Далее, в первую очередь, нужно сбросить флаг RXOUTI для подтверждения прерывания. Затем скопировать данные из буфера

конечной точки и сбросить флаг FIFOCON. С флагом RXOUTI, может быть связано прерывание, и чтение данных из буфера может быть организовано в обработчике прерывания. Только когда сброшены флаги, SIE будет способно обрабатывать дальнейшие транзакции к конечной точке.

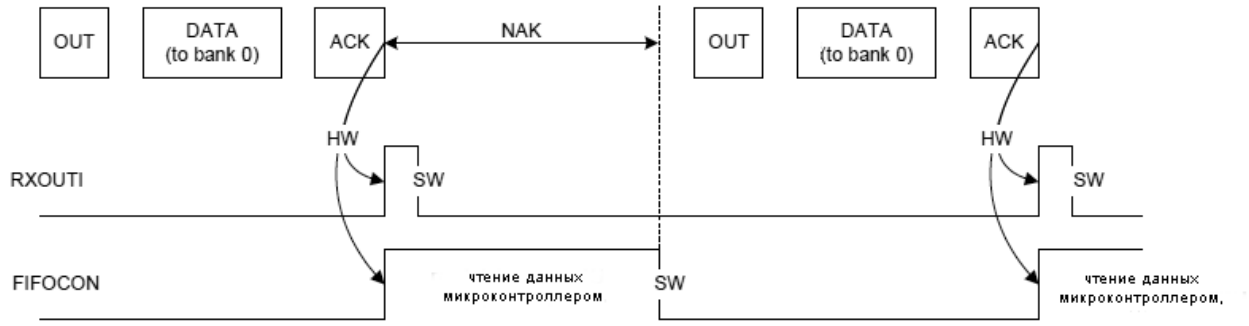


Рисунок 23

Если FIFOCON не сброшен, на все попытки хоста записать данные, SIE будет отправляться пакет квитирования NAK. На рисунке 23 показана транзакция записи.

Транзакция чтения. Исходное состояние, флаги TXINI и FIFOCON установлены, в буфере конечной точки нет данных для отправки хосту. Загружаем данные в буфер, затем сбрасываем флаги TXINI и FIFOCON, TXINI сбрасывается первым. Устройство в состоянии ожидания пакета IN от хоста. После успешного выполнения транзакции чтения, флаги TXINI и FIFOCON аппаратно устанавливаются. С флагом TXINI, может быть связано прерывание, и загрузку данных можно выполнять в обработчике прерывания. Если FIFOCON не сброшен, то на все попытки чтения данных, хосту будут отправляться пакеты квитирования NAK. На рисунке 24 изображена транзакция чтения.

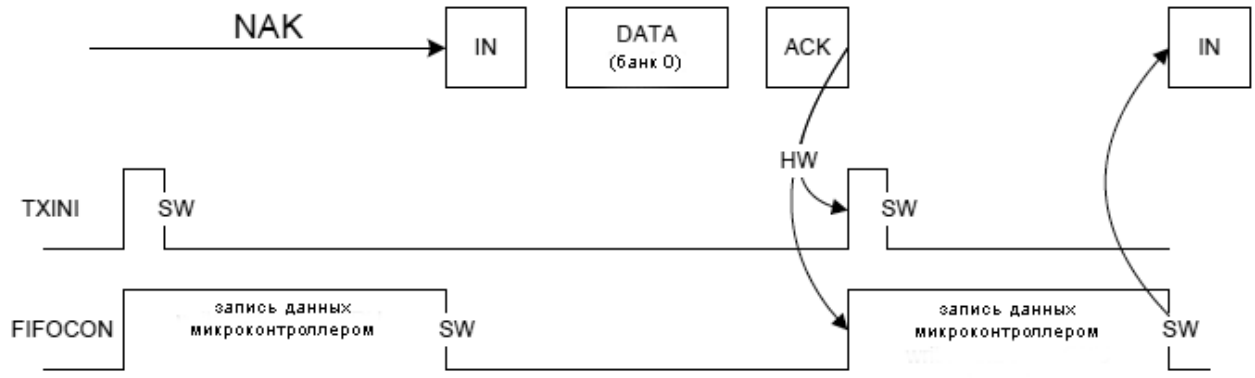


Рисунок 24

Счётчик байт в буфере конечной точки, пара регистров, UEBCHX и UEBCLX. В регистре UEBCLX хранится младшая значащая часть, а в регистре UEBCHX значимы только младшие 3 бита. При каждом чтении или записи из буфера FIFO конечной точки, значение этого счётчика соответственно меняется.

После выбора конечной точки, с помощью регистра UENUM, запись в буфер FIFO выполняется через регистр UEDATX.

Флаг RWAL отражает состояние буфера текущей конечной точки. Для конечной точки записи, этот флаг установлен, когда данные могут читаться микроконтроллером, и сбрасывается аппаратно, когда все данные прочитаны. Для конечной точки чтения, флаг RWAL установлен, когда микроконтроллер может записывать в буфер данные, подлежащие отправке хосту.

У конечной точки управления в процессе работы, возникают события. Когда управляющая конечная точка примет в свой буфер маркерный пакет SETUP и пакет данных DATA0 управляющей передачи стадии установки, то аппаратно установится бит 3 RXSTPI регистра UEINTX. Установка этого флага сигнализирует о необходимости обработки конфигурационного пакета запроса управления. В пакете DATA0 содержится важная информация об управляющем запросе, уже говорилось, что этот пакет называется конфигурационным. Программное обеспечение микроконтроллера, должно проанализировать конфигурационный пакет, выполнить необходимые действия, и сбросить флаг RXSTPI. С флагом может быть связано прерывание. Сброс флага нужен для подтверждения прерывания и сброс очищает буфер EP0. Конечно, буфер нужно очищать, после того как из этого буфера будут прочитаны данные. Прерывание разрешается установкой бита 3 RXSTPE регистра UEIENX. Если в запросе только 2 стадии, то после стадии установки, SIE должно принять маркерный пакет IN стадии статуса и отправить ZLP(Zero Length Packet). Для отправки ZLP, нужно сбросить флаг TXINI. При сбросе флага TXINI, содержимое текущей конечной точки готово для отправки хосту. При поступлении пакета IN от хоста, ZLP будет отправлен, стадия статуса завершена, а флаг TXINI будет аппаратно установлен.

Если, проанализировав конфигурационный пакет, требуется ответить пакетом STALL, то для этого устанавливают флаг STALLRQ, регистра UECONX. На маркерный пакет IN от хоста, будет отправлен STALL. А в начале нового запроса управления от хоста, при получении пакета SETUP, флаг STALLRQ будет аппаратно сброшен и все флаги обработки запроса управления, будут действовать в штатном режиме. Конечная точка управления не может находиться в блокированном состоянии и постоянно отвечать пакетами STALL на пакеты от хоста. Рисунок 25 поясняет сказаное.

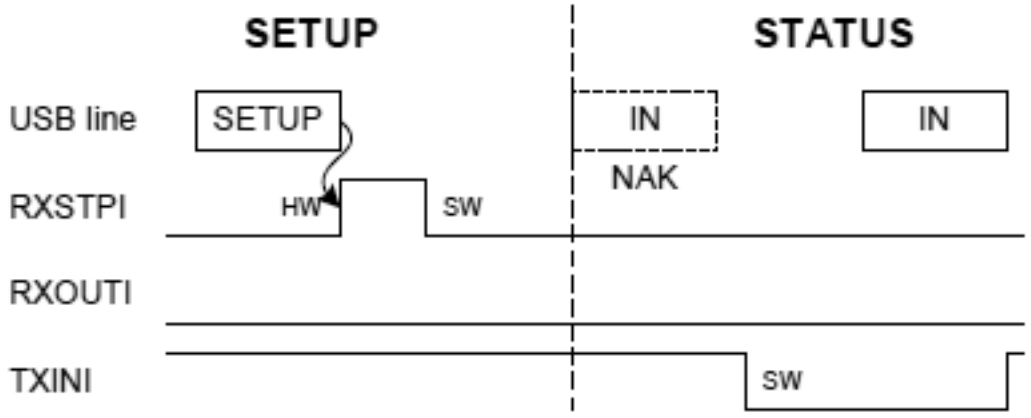


Рисунок 25

Для управляющего запроса имеющего стадию записи данных, стадия установки аналогична описанной выше. Но после стадии установки следует стадия записи данных в SIE. В стадии данных от хоста поступают пакеты OUT и DATAx. Факт присутствия данных в буфере EP0 , отмечается установкой флага RXOUTI регистра UEINTX. Сброс этого флага, очищает буфер EP0 и делает возможным приём следующего маркерного пакета и пакета данных. Если транзакций стадии

данных должно быть несколько, что определяется из анализа конфигурационного пакета, то приход маркерного пакета IN, стадии статуса, определяется по флагу NAKINI. Установка флага NAKINI указывает, что в ответ на пакет IN от хоста, был послан NAK и пришёл черёд стадии статуса. Ответ хосту аналогичен ответу для 2-х стадийных запросов управления, программно сбрасывается флаг TXINI.

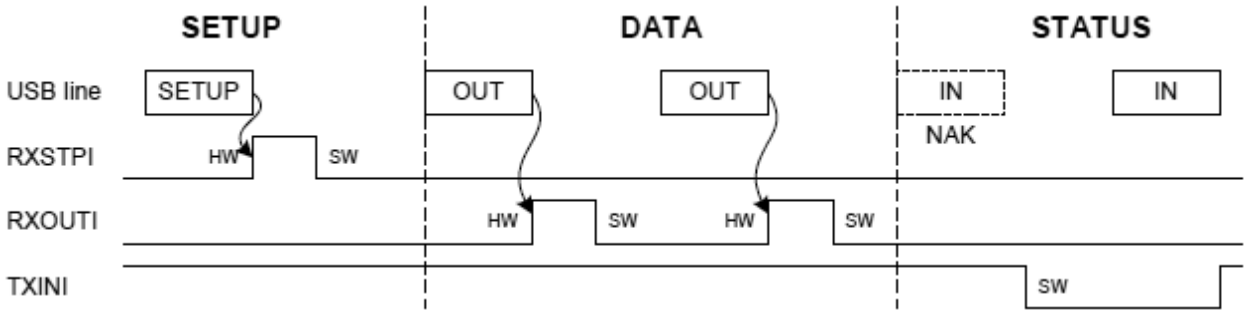


Рисунок 26

Другой вариант, определения момента наступления стадии статуса – это просуммировать количество пакетов DATAx полученных от хоста в стадии данных и сравнивать с тем, что следует из информации извлечённой из конфигурационного пакета. Для надёжности, можно комбинировать оба метода.

Если требуется ответить пакетом STALL, в ответ на пакеты OUT DATAx, в стадии данных, то устанавливается флаг STALLRQ.

Для управляющего запроса имеющего стадию чтения данных, стадия установки (Setup Stage) аналогична описанной выше стадии запроса для запросов управления записи. На стадии данных (Data Stage), SIE в ответ на входящие маркерные пакеты IN от хоста, должен посылать заранее подготовленные пакеты данных. Зная работу транзакций чтения, легко разобраться. Подготовив данные в буфере EP0,программно сбрасывается флаг TXINI и ожидается когда подготовленные данные будут отправлены хосту. Когда всё содержимое конечной точки управления будет передано, аппаратно устанавливается флаг TXINI. Когда флаг TXINI установился, можно посылать следующую порцию данных или ожидать начало стадии статуса, в зависимости от запроса управления.

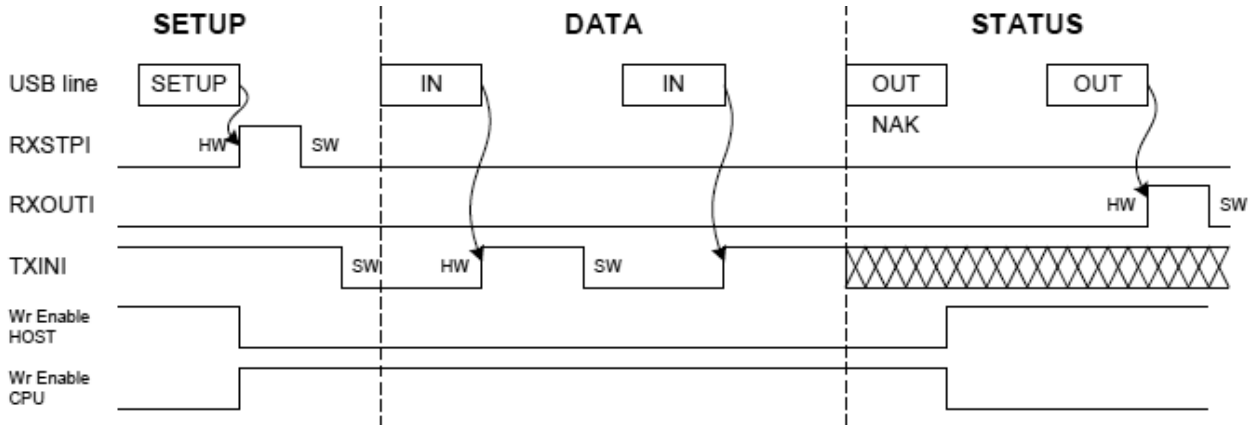


Рисунок 27

После стадии данных, стадия статуса. На стадии статуса, поступает пакет OUT и пакет ZLP от хоста, это вызывает установку флага RXOUTI. Установка флага сигнализирует о завершении транзакции статуса, далее программно сбрасывается этот флаг.

Программа

Понять работу подсистемы USB AT90USB162, будет значительно легче, если ознакомиться с готовыми примерами программ.

Программа для микроконтроллера сознательно сделана максимально просто, так как является учебной. В бесконечном цикле идёт опрос флагов микроконтроллера, отражающих события на шине и события связанные с поступлением данных в конечные точки. Очевидный недостаток такого решения, нельзя на длительное время занимать процессор прикладной задачей. Сканирование событий должно вестись непрерывно. Прикладная задача должна выполняться в порядке общей очереди. Но есть и преимущество, понятная логика и простота, что важно для учебных программ. Поняв логику работы шины, создать программу с обработчиками событий на прерываниях не составит труда.

В непрерывном цикле идёт опрос флагов: связанных с линией VBUS, с командой «сброс» на шине, с переходом в состояние «приостановлено», а так же с командой пробуждения от хоста. После обнаружения события выполняется ряд необходимых действий. Так же ведётся сканирование состояние флага RXSTPI. Установка флага сигнализирует, что от хоста поступил запрос управления, и стадия установки уже выполнена. Требуется считывание из буфера конечной точки управления конфигурационного пакета и его анализ. В зависимости от типа запроса, нужно выполнить необходимые действия. После чего возврат в цикл сканирования.

Программная обработка запросов управления имеет определённые достоинства. Можно создавать свои запросы управления и с их помощью передавать некоторые данные и команды, не привлекая конечные точки чтения и записи, а используя единственную точку управления. Запросы классов, о которых упоминалось (например HID), кроме стандартных запросов, имеют и некоторые специфические запросы управления. Обработка этих запросов выполняется программой микроконтроллера.

Каждый принятый пакет SOF вызывает прерывание. Индикация потока пакетов SOF с шины приводит к миганию светодиода. Удобно для анализа состояния устройства при отладке.

Конечная точка записи массовой передачи микроконтроллера, принимает данные от программы на стороне хоста, модифицирует их и возвращает хосту, используя конечную точку чтения массовой передачи.

Поступление данных в буфер конечной точки записи определяется по состоянию флага RXOUTI. Затем определяется количество данных подлежащих чтению, из регистра UEBCLX и копирование принятых данных. Эти данные копируются в буфер конечной точки чтения и благополучно отправляются хосту.

Для целей отладки задействуется USART, при обработке запроса управления, через USART сбрасываются некоторые символы, показывающие состояние устройства USB. Для отладки состояния «приостановлено», компьютер к которому подключено USB устройство, переводится в состояние S3, это состояние с сохранением содержимого ОЗУ. Символы отладки поступают на другой компьютер с запущенной терминальной программой. Прикладная задача имитируется простым возвращением переданных символов обратно терминальной программе. В Linux для перевода хоста в S3, использовалась утилита pm-suspend.

Для загрузки программ на микроконтроллер AT90USB162 рекомендую использовать dfu-programmer. Микроконтроллеры, имеющие аппаратную поддержку шины USB, укомплектованы программой встроенного загрузчика DFU (Device Firmware Upgrade). Используя программу dfu-programmer, можно записывать пользовательскую программу в память контроллера и запускать, без применения аппаратного программатора. Первоначально программа создавалась для Linux , но имеются порты и для Windows.

Активируется загрузчик следующей процедурой. Нажимается кнопка «RESET», затем не отпуская кнопку «RESET», нажимаем кнопку «HWBE» (Hardware Boot Enable), отпускаем «RESET», отпускаем «HWBE». Утилитами можно посмотреть дескрипторы загрузчика. Интересно, что DFU использует только конечную точку управления для передачи и команд и данных.

В Windows, для работоспособности dfu-programmer, нужно дополнительно установить библиотеку libusb-win32. Библиотека и сопутствующие файлы поставляются в комплекте с программой. После активирования загрузчика микроконтроллера и подключения, Windows сообщит об обнаружении нового устройства и необходимости установки драйвера. На запрос Windows , укажите расположение каталога, где находится файл с расширением inf, который поставляется с dfu-programmer. После успешной установки, программа будет видеть микроконтроллер с загрузчиком на шине. Простой командный файл запишет программу в микроконтроллер:

```
dfu-programmer at90usb162 erase
dfu-programmer at90usb162 flash  my_file.hex
dfu-programmer at90usb162 start .
```

Готовые тексты учебных программ вы можете получить, посетив авторскую страницу vk.com/protocols. Программы компилируются для хоста и микроконтроллера на платформе Linux и Windows. В планах автора дальше развивать тему USB для микроконтроллеров. Дальнейшая работа будет посвящена протоколам устройств класса, таких как HID и CDC и созданием программ для 32-х разрядных микроконтроллеров.

Список литературы

Гук М. Ю. Шины PCI, USB и FireWire. Энциклопедия.- СПб.: Питер, 2005. — 540 с
Агуров П.В. Практика программирования USB – СПб: БХВ-Перербург, 2007 — 624 с
John Hyde. USB Design by Examples. A Practical Guide to Building I/O Drvices. – Intel Press 2001
Axelson Jan. USB Complete. – Lakeview Research, 2001.