Создание USB-устройств

.

Юрий КОЗЛОВ, инженер КБ "Тц ЖАиС"
Вячеслав ПРОНИН, ведущий инженер КБ "Тц ЖАиС"

Интерфейс USB прочно вошел в мир интеллектуальной техники и продолжает завоевывать все большую популярность. Взамен традиционных коммуникационных интерфейсов COM и LPT производители компьютеров оснащают современные модели множеством USB-разъемов. Поддержка USB для разработчиков устройств из модного направления превратилась едва ли не в обязанность. Однако эта обязанность может стать серьезной проблемой из-за нехватки информации о данном вопросе. Данный цикл статей призван восполнить недостаток освещения USB-интерфейса на русском языке.

Часть 1. Вступление

Интерфейс USB прочно вошел в мир интеллектуальной техники и продолжает завоевывать все большую популярность. Взамен традиционных коммуникационных интерфейсов COM и LPT производители компьютеров оснащают современные модели множеством USB-разъемов. Поддержка USB для разработчиков устройств из модного направления превратилась едва ли не в обязанность. Однако эта обязанность может стать серьезной проблемой из-за нехватки информации о данном вопросе. Данный цикл статей призван восполнить недостаток освещения USB-интерфейса на русском языке.

USB против RS

До появления USB как способ подключения внешних устройств господствовали ставшие теперь классическим COM-интерфейсы по протоколу RS-232/485. Их самое главное достоинство для разработчика, по нашему мнению, это простота реализации и программирования. Но это качество меркнет в тени двух огромных недостатков: низкой скорости передачи данных и невозможности «горячего» подключения устройств. В отсутствии альтернативы с этими недостатками приходилось мириться. Но с появлением USB стала заманчивой возможность повысить потребительские свойства своих продуктов, внедрив в них поддержку этого интерфейса. Ведь USB это облегчение жизни пользователя

- простые разъемы и легкость подключений,
- поддержка архитектуры Plug-&-Play, самоидентификация устройств,
- высокая скорость обмена данными.

Также интерфейс USB обладает еще рядом достоинств для разработчиков:

- поддержка многих устройств на шине, а также составных устройств;
- поддержка разных типов передач данных, гибкая настройка форматов передач;
- гарантированная пропускная способность;
- мощные средства контроля и исправления ошибок.

Кроме этого, интерфейс USB привлекателен именно своей универсальностью. Ведь кроме COM-разъемов большинство компьютеров оснащены разъемами PS/2 мыши и клавиатуры, LPT разъемом и др. Некоторые из них, а иногда большинство просто не используются.

Все сказанное не может не сыграть роль в пользу выбора USB для ваших продуктов.

Обзор источников информации

Главным и непререкаемым источником информации по любым вопросам, связанным с USB, являются документы, распространяемые Форумом[1] разработчиков USB. Эти документы, публикуемые на английском языке, являются первоисточником и доступны для свободного скачивания на сайте форума www.usb.org.

Также на английском языке существуют две очень достойные внимания, по мнению авторов данной статьи, книги: USB Complete (Jan Axelson) и USB Design by Example (John Hyde). Изложение материала в них

ведется доступным языком и в достаточном объеме, а самое главное в них нет лишней информации. Найти ссылки на эту литературу можно с помощью специализированного поисковика электронных книг www.ebdb.ru.

Кроме того, в российском интернете активно рекламируется и продается одна книга (в двух редакциях) на русском языке. Ценность ее весьма сомнительна. Издание перегружено фотографиями usb-мышек, флеш-накопителей и других устройств; бесполезными некомментированными листингами дескрипторов репортов[2] и т.п. Оторванные от контекста куски информации без начала и продолжения буквально выплеснуты на страницы книги и перемешаны без жалости к читателю. Однако, на безрыбье...

AT89C5131: пример одного решения

В отсутствии других русскоязычных источников многим разработчикам, начинающим внедрять поддержку USB в свои продукты, приходится обращаться именно к изданию, упомянутому последним в предыдущем разделе. В нем рассматриваются примеры реализации USB устройств на основе микроконтроллера AT89C5131 фирмы ATMEL. Стоит отметить, что это весьма разумное решение: недорогая микросхема со встроенным USB-контроллером на хорошо зарекомендовавшем себя ядре MCS51 с приятным ассемблером. Однако и здесь не без проблем: встроенный USB-контроллер содержит достаточно ошибок и разработчику придется потрудиться, чтобы заставить его работать. Но не торопитесь отказываться от идеи построить свое устройство с использованием этого контроллера: мы нашли решение многих его проблем и поделимся опытом в будущих статьях цикла.

Закрывая глаза на все другие недостатки книги, мы считаем самым жирным ее минусом построение программных примеров на основе линейной модели. Автор пишет что «всю работу можно перенести в обработчики прерываний, но линейная структура программы более читабельна и понятна». С последним утверждением мы согласны, но вот практическая ценность примеров теряется, когда они отдалены от практических потребностей. Промышленный код для микроконтроллеров в большинстве случаев должен быть построен на основе системы прерываний: ведь кроме обработки USB-запросов контроллер должен выполнять некую полезную работу. Создается впечатление, что автор линейной модели просто не смог заставить контроллер использовать его систему прерываний. Кроме того, приведенный код написан на языке С, хотя наибольшее представление о контроллере, его архитектуре и принципе работы может дать только код на языке ассемблера. Наши программы, которые мы приведем в последующих статьях цикла, построены на основе механизма прерываний и написаны на языке ассемблера. Если статьи найдут свою аудиторию, мы могли бы портировать код под С по заявкам читателей.

Следует отметить, производитель микросхемы – фирма ATMEL знает о проблемах своего продукта, но скрывает это. Приведенные на их сайте www.atmel.com примеры программирования AT89C5131 так же имеют линейную структуру (интересно, что коды на сайте и в книге как родные братья похожи друг на друга, и кто только у кого содрал:-) Однако, сейчас мы точно знаем: реализовывать поддержку USB на AT89C5131 при помощи системы прерываний можно. А в большинстве практических случаев еще и нужно.

Далее наше обсуждение пойдет по двум ветвям. В одной ветви мы на про-тяжении 3-х статей рассмотрим теоретические вопросы организации шины USB. Затем, используя полученные знания, напишем программу для микроконтроллера AT89C5131, реализующую поддержку USB протокола. Во второй ветви, мы будем обсуждать вопросы, не требующие как основу знаний об организации шины. В первой статье рассмотрим демонстрационную плату, опишем тонкости выбора контроллера и его подключения. В последующих статьях обсудим вопросы про-граммирования USB со стороны компьютера.
Заметим, что в наших работах мы будем двигаться путем, описанным в упомянутой выше книге, модернизируя и дополняя материал до того вида, в кото-ром хотели бы видеть его сами в свое время, но не смогли найти.

[1] Понятие «форум» означает здесь не интернет-сервис, позволяющий пользователям общаться посредством текстовых сообщений, а сообщество единомышленников.

[2] Понятия дескрипторов и репортов мы рассмотрим в следующих статьях цикла.

Логическая структура USB-устройств

Стандарт USB предусматривает логическое разбиение внутренней структуры устройств по нескольким уровням иерархии. Так же как и топологию шины USB логическую организацию USB-устройств схематично удобно представить в виде дерева. Пример такой организации приведен на рисунке 2. Узлы ветвей обозначают режимы работы, листья дерева определенные функции. Настройка устройства на некоторый конкретный режим

работы выполняется указанием узлов соответствующей ветви. Установку и смену режимов осуществляет хост. Режим остается активным до следующей команды смены режима. Функции, доступные в активном режиме, называются конечными точками. Выполнение функции осуществляется при обращении хоста к соответствующей конечной точке. В логической структуре устройства возможны режимы, в которых отсутствуют какие-либо функции (на рисунке 2 ветвь cfg1-if0-alt0). Такие режимы обычно используются для перевода устройства в некое бездействующее состояние типа простоя.

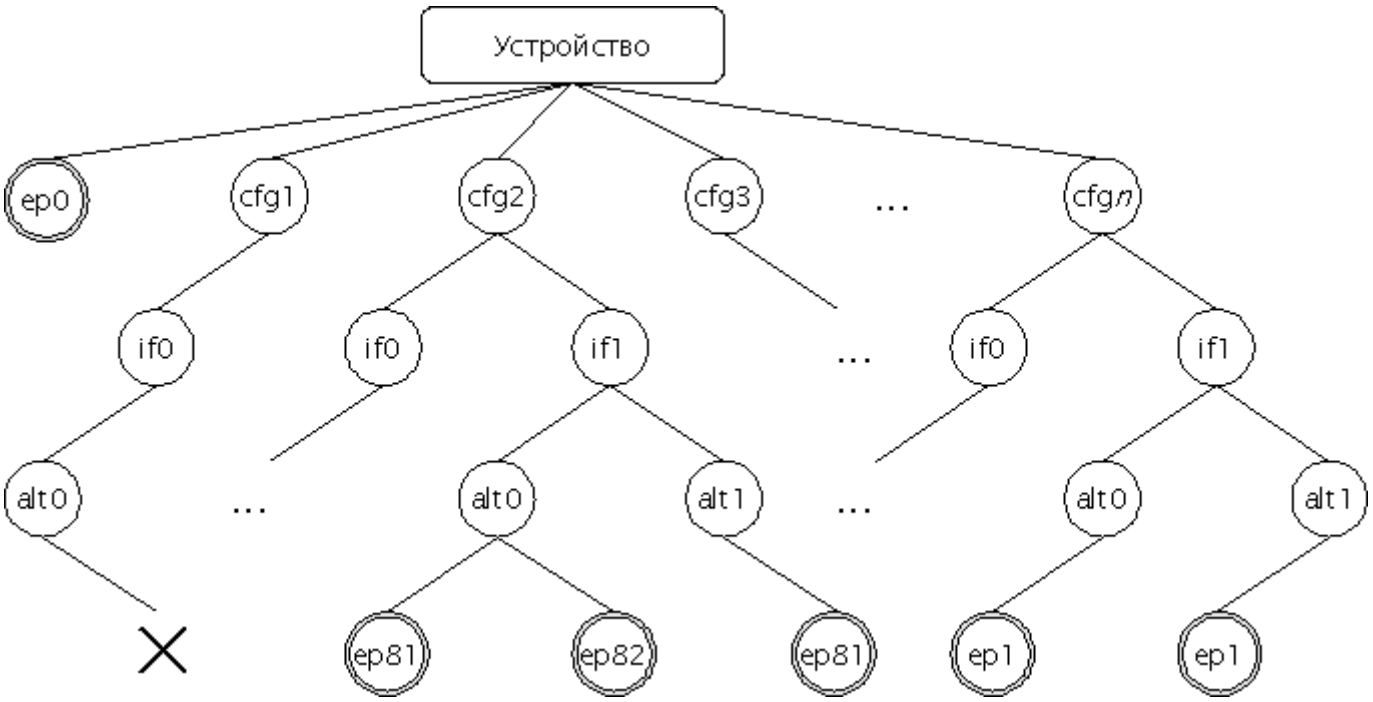


Рис. 2. Общая логическая структкура USB-усройства

Рассмотрим подробнее элементы логической структуры USB-устройств.

Конфигурации (cfg) являются верхним уровнем обобщения и определяют наиболее общие настройки. Устройство может иметь до 255 конфигураций, но хотя бы 1 оно иметь должно. Нумерация конфигураций начинается с 1. Номер 0 используется для обозначения того, что устройство не сконфигурировано.

На следующем уровне иерархии находятся интерфейсы (if). Они позволяют разделить общий режим работы, определяемый конфигурацией, по различным специфическим особенностям. Хотя бы один интерфейс должен присутствовать в любой конфигурации, а максимально их может быть 256. Нумерация интерфейсов начинается с 1.

После интерфейсов еще более конкретно уточняют режим работы альтернативные установки (alt), они описывают частные случаи настройки режима работы устройства. Альтернативные установки определяют количество поддерживаемых в данном режиме конечных точек-функций и способы работы с ними. Нумерация альтернативных установок начинается с нуля, а их количество в каждом интрефейсе может быть от 1 до 256.

Функции, коотрые устройство способно выполнять в текущем режиме, определяются доступными конечными точками (ep). Конечная точка это логический канал передачи данных. При записи данных в канал устройство выполняет некоторую функцию; при чтении из канала устройство возвращает результаты выполнения операций, текущие параметры и т.д. Каждая конечная точка имеет свой уникальный 8-ми разрядный адрес. Конечные точки, входящие в состав альтернативных установок, являются однонаправленными, направление передачи для данной конечной точки определяет старший бит ее адреса: 0 в старшем бите адреса имеют точки направления OUT, 1 - точки, направления IN.Альтернативная установка

может содержать до 15 точек направления OUT с адресами 1..0Fh и до 15 точек направления IN с адресами 81h..8Fh.

Особое значение имеет конечная точка 0 (ep0). Она называется контрольной и прнадлежит не какой-либо альтернативной установке, а всему устройству вцелом. Контрольная точка является двунаправленной и доступна в любом режиме работы устройства, через нее осуществляется идентификация и конфигурирование устройства.

Для примера организации внутренней структкуры USB-устройства рассмотрим некоторый воображаемый мультиметр. Допустим, этот мультиметр может измерять электрическое напряжение, ток, возможно другие электрические параметры; а также имеет функцию генератора электрических импульсов. Безликая структура устройства, представленная на рисунке 2, адаптирована для представления мультиметра и с указанием конкретных величин и обозначений представлена на рисунке 3.

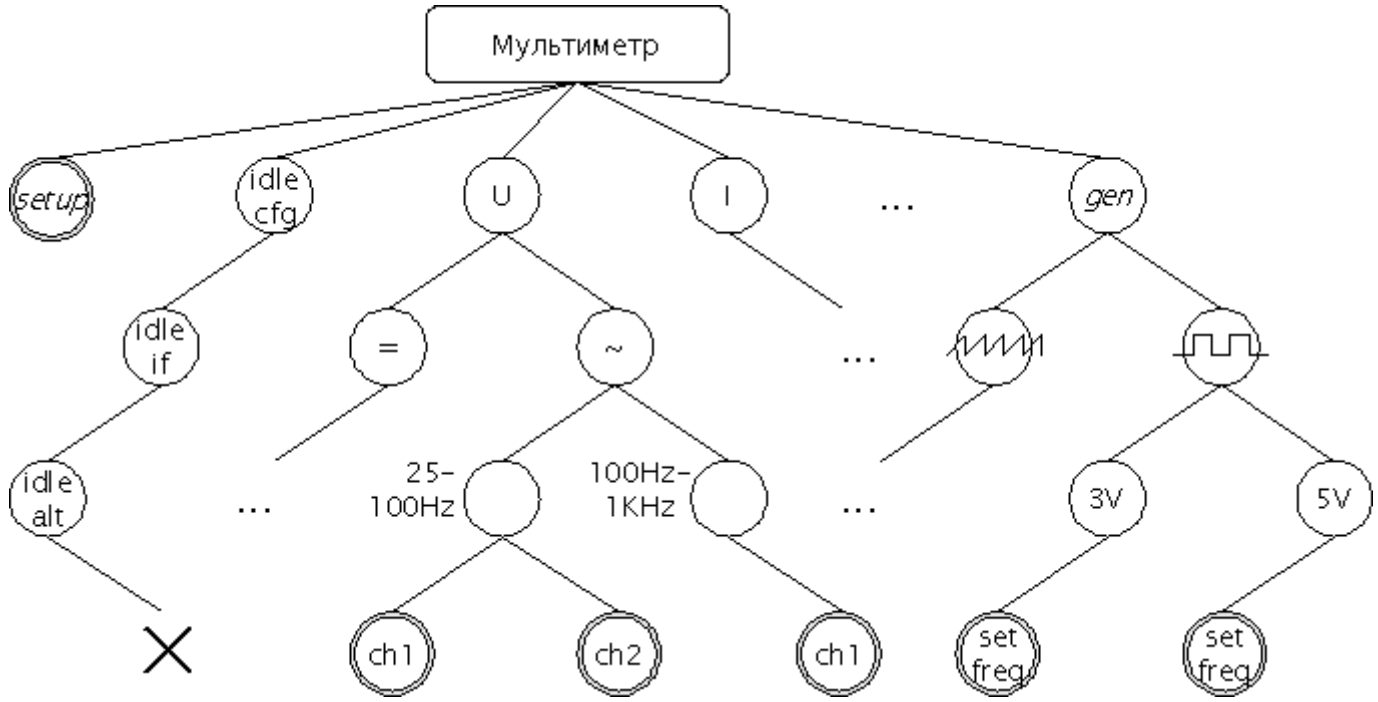


Рис. 3. Пример логической организации USB-мультиметра

Первая ветвь-режим cfg1-if0-alt0 (рис. 2) используется для перевода прибора в режим простоя (idle cfg-idle if-idle alt на рис. 3).

Конфигурация cfg2 (рис. 2) предназначена для установки режима измерения электрического напряжения (узел U на рис. 3). Интерфейс if0 обозначает измерение постоянного напряжения (узел "=" на рис. 3); интерфейс if1 - измерение переменного напряжения (узел "~"). Мультиметр способен измерять переменное напряжение в нескольких диапазонах частот: альтернативные установки alt0 и alt1 интерфейса измерения переменного напряжения определяют соответственно диапазоны от 25Гц до 100Гц и от 100Гц до 1КГц. На низких частотах мультиметр способен измерять переменное напряжение по 2-м входным каналам (листья ch1 и ch2 на рис. 3), что определяет наличие в данной альтернативной установке 2-х конечных точек (ep81 и ep82 на рис. 2) направления IN. На частотах от 100Гц до 1КГц прибор способен воспринимать сигнал с одного канала, что определяет только одну конечную точку для данной альтернативной установки.

Конфигурация cfg3 (рис. 2) используется для измерения электрического тока (узел I на рис. 3).Распределение ее подрежимов может быть аналогично таковому для измерения напряжения.

Конфигурация cfg*n* предназначена для установки режима генератора. Представим что прибор способен генерировать пилообразные и прямоугольные импульсы. Для выбора одного из этих вариантов используются интрефейсы if0 и if1 конфигурации cfg*n* (обозначения рис. 2). Допустим, устройство может генерировать импульсы со значением амплитуды, принадлежащим некоторому дискретному ряду, например, 3В и 5В. Альтернативные установки позволяют выбрать конкретную амплитуду генерируемого сигнала. Конечные точки ep1 направления OUT для каждой из альтернативной установок предназначены для задания частоты импульсов.

Отметим еще раз, альтернативные установки могут иметь одновременно как точки направления IN, так и точки направления OUT произвольных адресов из допустимого диапазона.

Подключение и работа устройства

Подключение устройства к шине производится через порт хаба. Хост, периодически опрашивая состояние хаба, распознает подключение нового устройства и разрешает соответствующий порт. В это время устройство считается не адресованным и не сконфигурированным. Хост обращается к устройству по адресу 0 через контрольную точку ep0, доступную в любом режиме работы устройства. Первой командой хост присваивает устройству уникальный адрес, с которым оно работает до момента отключения от шины. Затем, хост считывает описание устройства и описание всех его конфигураций. Устройство обязано иметь хотя бы одну конфигурацию. Хост устанавливает первую доступную конфигурацию, не анализирую ее назначения. После этого устройство сичтается сконфигурированным и готовым к работе. Полученная информация позволяет операционной системе идентифицировать устройство и загрузить подходящий драйвер. Дальнейшее управление устройством передается драйверу.

Для выбора требуемого режима драйвер через контрольную точку ep0 передает запросы установки соответствующих конфигураций и интерфейсов. Для смены режимов эти запросы выдаются повторно.

Для активизации некоторой функции устройства в данном режиме драйвер обращается к какой-либо конечной точке.

Передача данных по шине USB

На шине USB организована пакетная передача данных. Пакетная передача данных, ведущая роль хоста на шине и контроль целостности данных, заложенный на уровне протокола, в совокупности определяют общий цикл обмена пакетом, состоящий из 3-х тактов: запроса, передачи полезных данных, подтверждения. Запрос определяет тип передачи и адреса получателей: адрес устройства на шине и адрес конечной точки в текущем режиме работы. Подтверждение показывает целостность данных и готовность устройства к продолжению обмена.

Формат пакета запроса представлен на рисунке 4. Он состоит из маркера запроса, определяющего его тип, адреса устройства, адреса конечной точки, контрольной суммы CRC5.

Маркер запроса {SETUP | IN | OUT | PING}	Адрес устройства	Адрес конечной точки	CRC5

Рис. 4. Формат пакета запроса

Маркер запроса может иметь одно из 4-х значений:

- SETUP - означает что хост начинает контрольную передачу для указанной точки устройства;
- IN - хост ожидает данные от устройства;
- OUT - хост начинает передачу данных конечной точке устройства;
- PING - хост на высокоскоростной шине проверяет состояние конечной точки направления OUT.

За пакетом запроса следует пакет полезных данных. Его формат показан на рисунке 5. Пакет содержит в себе маркер данных, непосредственно данные и контрольную сумму CRC16.

Маркер данных {DATA0 | DATA1 | DATA2 | MDATA}	Данные	CRC16

Рис. 5. Формат пакета данных

Маркеры данных призваны обеспечивать контроль целостности данных на уровне потока. Применяются следующие значения:

- DATA0 обозначает четный пакет данных;
- DATA1 - нечетный пакет данных;
- значения DATA2 и MDATA используются при изохронном обмене на высокоскоростной шине.

Пакеты подтверждения состоят только из соответствующего маркера. Они предназначены для сообщения о результатах передачи данных и текущего статуса конечной точки. В протоколе предусмотрены следующие маркеры подтверждения:

- ACK - даные получены без ошибок и будут обработаны;
- NAK: для точки направления OUT означает что данные получены без ошибок, но в данный момент не могут быть обработаны и требуется их повторная передача; для точки направления IN говорит о том, что данные еще не готовы, хост может повторить попытку позднее.
- STALL - запрос не поддерживается.
- NYET - данные текущего пакета получены корректно и будут обработаны, но следующий пакет точка принять сразу не сможет.

Кроме перечисленных в протоколе оговорены еще несколько типов пакетов: SOF, PRE, ERR, SPLIT. Они выполняют служебные функции. Например, пакет SOF используется для синхронизации.

На шине USB предусмотрено 4 режима передачи данных:

- CONTROL,

- BULK,
- INTERRUPT,
- ISOCHRONOUS.

Передача типа CONTROL используется при обращении к контрольной точке устройства. Для такого способа передачи хост гарантированно выделяет 10% полосы пропускания шины. Полная транзакция для данного вида передачи состоит из трех фаз (рис. 6).

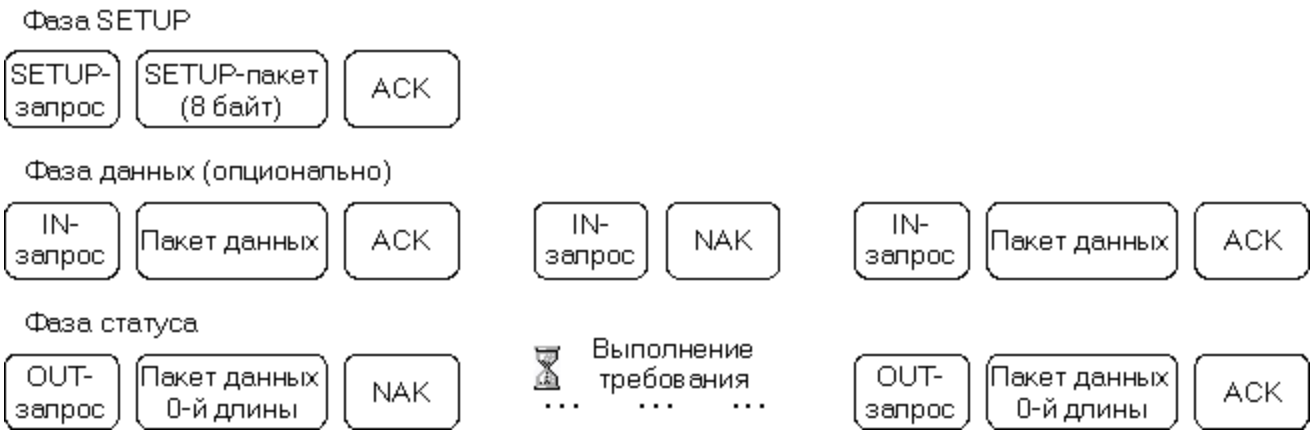


Рис. 6. Контрольная передача

Первая фаза называется фаза SETUP. Во время нее хост в пакете фиксированной длины 8 байт передает требование, которое необходимо выполнить устройству. Заметим, что в этой фазе маркером подтверждения SETUP-пакета является строго маркер ACK. Если устройство получило непподерживаемое или некорректное требование, оно должно подтвердить получение маркером ACK, а в следующей фазе вернуть маркер STALL.

Вторая фаза - фаза данных, которая является необязательной. Она участвует в транзакции в том случае, если для выполнения указанного требования требуются дополнительные данные. Рассмотрим подробнее структуру потока данных на этой фазе. На рисунке 6 приведен пример фазы данных направления IN. В указанном примере в первом цикле этой фазы хост сначала передает запрос на получение данных от устройства, устройство отвечает пакетом данных, далее хост подтверждает его получение маркером ACK. Во втором цикле устройсво не готово передать второй пакет данных и возвращает маркер NAK. При повторном запросе устройство успело сформировать данные для отправки и успешно передает пакет хосту.

Фаза данных может быть как направления IN, так и направления OUT. Пример фазы данных направления OUT отдельно показан на рисунке 7. Здесь хост в течении 2-х циклов успешно передает данные устройству. В 3-м же цикле устройство, занятое обработкой 2-х предыдущих пакетов, маркером NAK сообщает о том, что не может обработать в текущий момент 3-й пакет. Хост впоследствии повторяет передачу того же пакета данных.

Рис. 7. Фаза данных направления OUT

Последней фазой контрольной транзакции является фаза статуса. В этой фазе хост ожидает подтверждение выполнения требования устройством. Он посылает запросы противоположного направления, относительно запросов, которые использовались в фазе данных, а если фаза данных отсутствовала - запросы имеют направление IN. При передаче запроса OUT хост отправляет пакет данных нулевой длины. Во время выполнения запроса устройство отвечает маркером NAK, а по завершении работы с гордостью возвращает маркер ACK.

Еще раз обратим внимание на то, что если устройство получило требование, которое не способно исполнить, то оно должно вернуть маркер STALL в фазе данных или статуса, а получение SETUP-пакета обязано подтвердить маркером ACK.

Размер пакета для контрольных передач на высокоскоростной шине составляет 64 байта, на полноскоростной – 64, 32, 16 или 8 байт.

Передача типа BULK используется в том случае, если требуется гарантированная доставка пакетов, но время доставки не критично. Для пересылки больших объемов данных обычно используется именно BULK-передача. Например, такой способ передачи обычно используется USB-сканерами и принтерами. Протоколом гарантируется целостность доставки данных, которая обеспечивается посредством контрольной суммы CRC16 и различной маркировкой четных и нечетных пакетов. При обнаружении ошибки приемная сторона не подтверждает корректный прием пакета и передающая сторона повторяет посылку.

BULK-трафик занимает всю свободную полосу пропускания шины, но имеет самый низкий приоритет и может приостанавливаться на относительно большие промежутки времени в зависимости от загрузки шины.

Размер пакета данных при этом способе обмена может быть любым, в том числе и нулевым, но не превышать максимального значения. Для высокоскоростной шины эта граница составляет 512 байт, для полноскоростной – 8, 16, 32 или 64 байт.

Структура потока данных на шине повторяет таковую для контрольной передачи в фазе данных. Для примера можно еще раз обратиться к рассотрению контрольной передачи: передача BULK-IN представлена в средней части рисунка 6, передача BULK-OUT - на рисунке 7. Остановимся на последнем чуть подробнее. Он демонстрирует типичный обмен на полноскоростной шине и его главный недостаток. Недостаток заключается в том, что потерянный 3-й пакет необходимо повторять. Это снижает полезную пропускную способность шины. Очевидно, чем больше размер непроизводительного пакета, тем ощутимее снижение. На высокоскоростной шине с этим недостатком борются посредством запроса PING и подтверждения NYET. В подобной ситуации на высокоскоростной шине после приема 2-го пакета устройство ответит маркером NYET. При получении такого ответа, хост отложит передачу 3-го пакета и будет отслеживать готовность устройства с помощью короткого запроса PING. При получении подтверждения ACK хост сможет отправить 3-й пакет.

Следует отметить, что и при использовании механизма опроса PING/NYET потери и повторные передачи пакетов возможны, но в кардинально другом случае: из-за некорректного приема данных. Причиной тому может служить, например, действие внешней электрической помехи. Связка же PING/NYET борится именно с проблемой напрасных расходов пропускной способности шины из-за активности хоста, не скорректированной на возможности устройства.

Тип передачи INTERRUPT используется тогда, когда необходимо осуществлять обмен через заданный временной интервал. Хост обеспечивает опрос с заданным временным интервалом и учитывает это при планировании загрузки шины. Интервал

указывается устройством при конфигурировании и может лежать в переделах 0.125-4мс для высокоскоростной шины и 1-255мс для полноскоростной шины.

Размер пакета для этого вида передачи колеблется в пределах от 1 до 1024 байт для высокоскоростной шины и от 1 до 64 байт для полноскоростной.

Стукрура потока данных на шине подобна рассмотренной выше. Заметим, что отсутствие данных для конечной точки INTERRUPT-IN является штатной ситуацией. Следующий запрос хост пошлет по истечении еще одного заданного интервала.

Передачи типа ISOCHRONOUS в противоположность BULK-передачам используются для трафика, целостностью доставки которого жертвуют в пользу скорости. Контроль целостности в этом режиме передачи производится только при помощи контрольной суммы CRC16. Поврежденный пакет отбрасывается на принимающей стороне, передающая сторона об этом не уведомляется. Такой вид передач подходит для пересылки некомпресированных аудио- и видеопотоков. В таких потоках потеря одного-двух пакетов будет означать лишь легкое, возможно даже незаметное “заикание” для аудиотрафика либо небольшую искаженную (вероятнее всего просто темную) область для видеотрафика. Обращаем внимание на то, что компресированные аудио/видеоданные крайне чувствительны к потерям и не могут передаваться с помощью ISOCHRONOUS-передач.

Размер пакетов данных для этого вида передачи на высокоскоростной шине может быть до 1024 байт, на полноскоростной – до 1023 байт.

Для того чтобы воспользоваться любым типом передачи кроме котрольной устройство должно указать хосту свои возможные режимы работы и доступные в них функции-конечные точки, поддерживаемые ими типы передач. Устройство идентифицирует себя и конфигурируется хостом при помощи стандартных требований протокола USB. Стандартные требования USB мы опишем в следующей статье.

О выборе контроллера

Как мы уже упоминали в первой статье цикла, речь пойдет об ATMEL AT89C5131. Этот контроллер имеет ряд недостатков, однако он весьма дешев и построен на основе хорошо зарекомендовавшего себя ядра MCS-51 с удобными регистровой моделью, системой команд и системой прерываний. Однако, после определения базовой модели микросхемы отдельного рассмотрения заслуживает вопрос выбора конкретной ее модификации. Мы, как и фирма-производитель ATMEL настоятельно рекомендуем выбрать контроллер с суффиксом M в полной маркировке модели, т.е. контроллер с маркировкой AT89C5131A-xxxx**M**, а не AT89C5131A-xxxx**L**. Оба контроллера имеют одинаковую архитектуру и одинаковую цену. Заявленное производителем отличие M-контроллера от его L-собрата в том, что он имеет более широкий диапазон допустимого напряжения питания. Но, кроме того, M-контроллер имеет еще одно, по нашему мнению гораздо более важное, отличие. M-контроллер содержит меньше ошибок. Часть ошибок, присущих L-версии, отсутствуют в M-версии. Другая часть проявляется в меньшей степени. Поясним, что значит «проявление ошибки в меньшей степени» на примере. В некоторых случаях контроллер не успевает отрабатывать свои функции и для обхода этого приходиться искусственно добавлять задержки. Величина минимальной задержки необходимой для M-варианта в 3.5 раза меньше той же задержки L-контроллера.

Обе версии контроллера выглядят одинаково. Их внешний вид в исполнении для использования с подставкой PLCC52 приведен на рисунке 1, а назначение выводов для этого исполнения показано на рисунке 2. Для тех, кто видит его впервые, обратим внимание: одна из граней корпуса имеет скос. Ключ, отмечающий первую ножку микросхемы, находится в центре скошенной плоскости сверху, а также продублирован снизу микросхемы.

Рис. 1. Внешний вид микросхемы AT89C5131 в PLCC52-исполнении

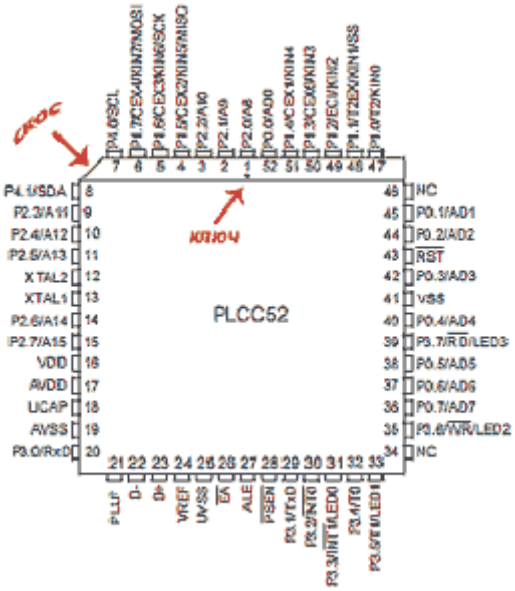
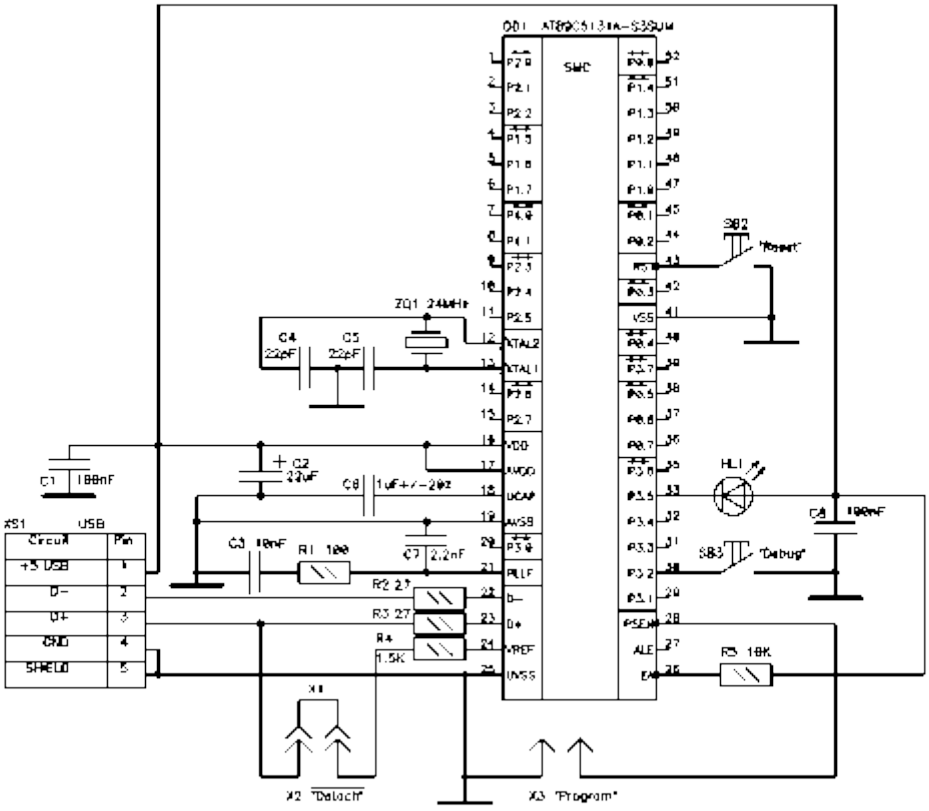


Рис. 2. Назначение выводов микросхемы AT89C513 в PLCC52-исполнении

Типовая схема включения

Для демонстрации программирования USB на AT89C5131 мы дополнили типовую схему его включения некоторыми элементами, оснастив ее небольшим набором сервисных возможностей. Они включают в себя кнопку сброса *“Reset”*, перемычки программирования *“Program”* и подключения к шине *“~Detach”*; а также светодиод *HL1* и вспомогательную кнопку *“Debug”*, которые возможно будет использовать как индикатор и некое условие в наших экспериментах.Схема приведена на рисунке 3.



.Рис. 3. Дополненная типовая схема включения микросхемы AT89C5131.

Обратите внимание, что конденсатор *C2* мы взяли номиналом 22µF в отличие от значения 4.7µF, приведенного в типовой схеме, представленной в справочных данных на микросхему. Здесь возможно использовать и конденсатор 4.7µF, однако для обеспечения помехоустойчивости схемы это должен быть танталовый конденсатор. При использовании алюминиевых конденсаторов их емкость должна быть увеличена по сравнению с номинальной в 5-10 раз.

Номинал же конденсатора *C7* наоборот не должен отстоять от своего заявленного значения 1µF более чем на 20%.

Собирая подобную схему полезно помнить о рекомендации располагать элементы по возможности ближе к микросхеме. Особое внимание стоит уделить элементам *C1-C7*, *R1-R4*, *ZQ1*, а также разъему USB-B *XS1*.

Наш вариант экспериментальной платы вы можете увидеть на рисунке 4.

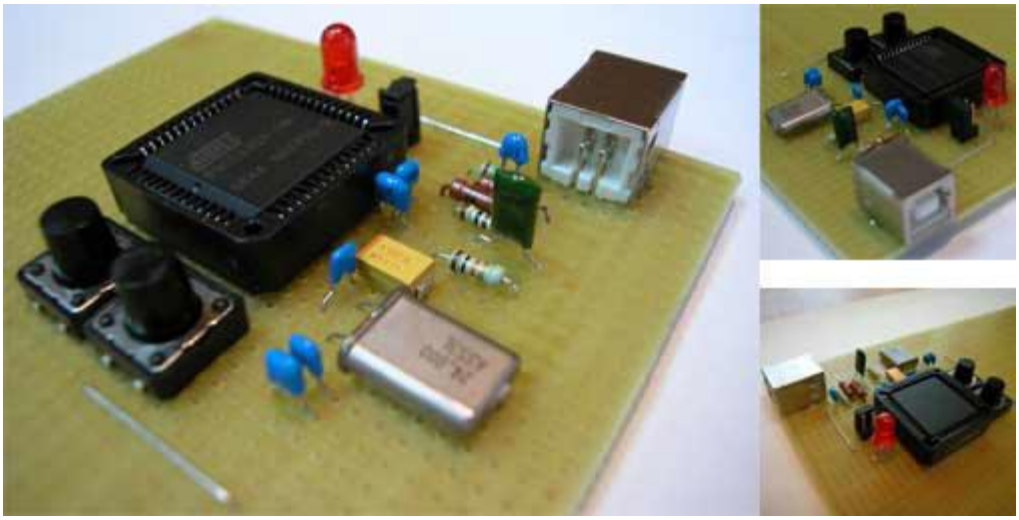


Рис. 4. Экспериментальная плата в сборе

Программирование микроконтроллера

Самый удобный путь программирования AT89C5131 это использование непосредственно USB-интерфейса при помощи программатора FLIP. Он доступен для бесплатного скачивания на сайте производителя www.atmel.com. Также, FLIP версии 3.0.5, которую используем мы в своей работе, можно скачать в формате zip-архива с нашего сайта. Отметим, что FLIP написан на языке Java и для его запуска необходимо установленной в системе среды исполнения Java. Программные инструменты Java доступны на сайте www.sun.com. Дистрибутив среды исполнения Java версии 1.5.0.9 для Windows® можно скачать в формате zip-архива у нас.

Первый раз программировать AT89C5131 можно без премудростей. При программировании, для того чтобы при перезапуске контроллера он начал исполнять пользовательскую программу, а не стандартный загрузчик, необходимо сбросить бит BLJB (Bootloader Jump Bit). Как выглядит FLIP и где найти бит BLJB показано на рисунке 5.

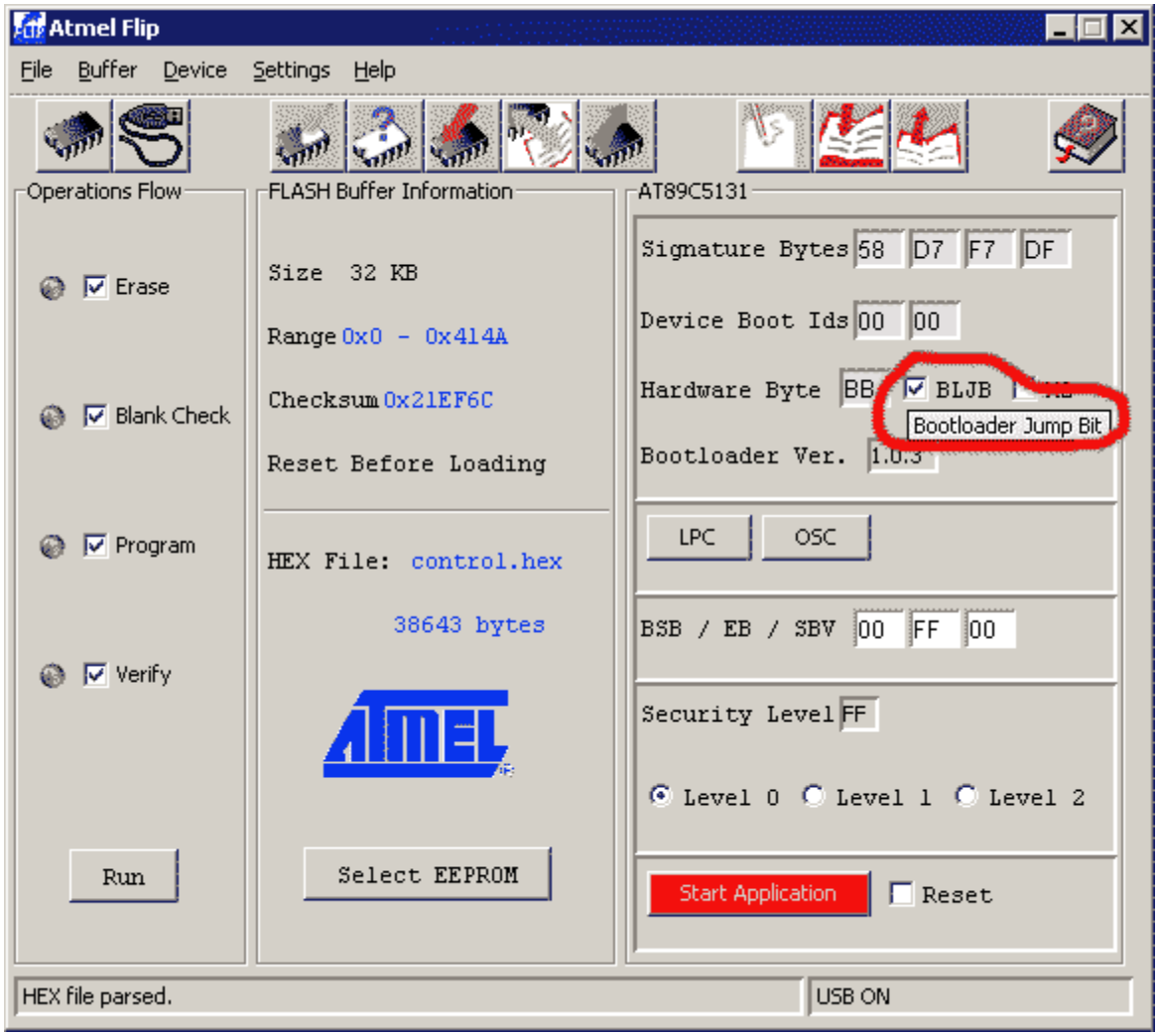


Рис. 5. Внешний вид главного окна программатора FLIP

При сброшенном BLJB для перепрограммирования AT89C5131 необходимо выполнить дополнительные действия. Нужно сначала переставить перемычку в положение *“Program”,* нажать кнопку *“Reset”* и после возвращения перемычки в положение *“~Detach”* контроллер готов к перепрошивке.

Представленный материал дает заинтересовавшимся основу для экспериментов, а мы продолжим обсуждение в следующих статьях цикла

Кодирование требований USB

Доставка требований осуществляется с помощью специально предназначенной для этого контрольной передачи. Схема обмена данными при контрольной передаче описана в предыдущей статье этой ветви обсуждения (часть 2.1). Напомним что полная транзакция такого вида передачи состоит из 3-х фаз: фазы SETUP, фазы даных и фазы статуса. Данные, идентифицирующие требование и его параметры, передаются в фазе SETUP в одноименном пакете фиксированной длины 8 байт. В формате пакета SETUP различают 5 полей (табл. 1).

Таблица 1. Формат пакета SETUP

№	Поле	Размер	Описание
1	bmRequestType	1	Битовая маска, определяющая тип требования
2	bRequest	1	Номер требования
3	wValue	2	Значение поля варьируется в зависимости от

			требования. Обычно содержит некое значение
4	wIndex	2	Значение варьируется. Обычно передает некоторый индекс или смещение
5	wLength	2	Количество байт, для передачи в фазе данных

Поле bmRequestType несет в себе информацию о типе требования. Оно содержит 3 субполя. Его формат представлен в таблице 2.

Таблица 2. Формат поля bmRequestType

Биты	7	6	5	4	3	2	1	0
Суб-поле	Направление передачи в фазе данных 0 - OUT, 1 - IN	Тип требования: 0 - стандартное, 1 - класса, 2 - производителя, 3 - резерв		Получатель требования: 0 - устройство, 1 - интерфейс, 2 - точка, 3 - другие получатели, 4..31 - резерв				

Номера (значения поля bRequest) стандартных требований и некоторых требований, специфичных для USB-устройств класса HID приведены в таблице 3. В этой статье мы будем использовать некоторые термины, применяемые к устройствам HID-класса, без пояснений. Подробно они рассматриваются в отдельной статье, посвященной этой теме.

Таблица 3. Коды стандартных требований и некоторых требований класса HID

Требование	Номер	Описание
Стандартные требования		
GET_STATUS	0	Получить состояние указанного получателя
CLEAR_FEATURE	1	Получить некоторое свойство получателя
SET_FEATURE	3	Установить некоторое свойство получателя
SET_ADDRESS	5	Установить адрес
GET_DESCRIPTOR	6	Получить описание получателя/свойства
SET_DESCRIPTOR	7	Установить описание получателя/свойства
GET_CONFIGURATION	8	Получить номер установленной конфигурации
SET_CONFIGURATION	9	Установить конфигурацию
GET_INTERFACE	10	Получить номер текущей альтернативной установки для заданного интерфейса
SET_INTERFACE	11	Установить номер текущей альтернативной установки для заданного интерфейса
SYNCH_FRAME	12	Получить номер фрейма
Некоторые требования класса HID		
GET_REPORT	1	Получить репорт устройства
SET_REPORT	9	Установить репорт

Назначение полей wValue и wIndex варьируется в зависимости от требования.

Поле wLength показывает какое количество информации должно быть передано в фазе данных. Направление передачи определяет старший бит поля bmRequestType (см. табл. 2).

При разработке USB-устройств стандарт определяет возможность встроить поддержку новых требований. При этом обязательным является строгое соблюдение структуры поля bmRequestType и правильное использование поля wLength. Остальные поля разработчик может использовать по своему усмотрению, перед отправкой хост их не анализирует. При добавлении требования, которое будет поддерживать только один тип устройств , следует биты 6 и 5 поля bmRequestType устанавливать в значение 10b, что является признаком требования производителя (продавца). При добавлении требования, которое будет поддерживать целый класс устройств, будет логично описать это требование как принадлежащее классу, т.е. биты 6 и 5 выставить в значение 01b.

СТАНДАРТНЫЕ ТРЕБОВАНИЯ

Требование GET_STATUS (получить состояние)

Требование GET_STATUS используется хостом для выясения состояния устройства, интерфейса или конечной точки. Еще раз отметим, что получатель требования указывается в младших 5-ти битах поля bmRequestType (см. табл. 2). В ответ на это требование соответствующий получатель возвращает 16-ти битное слово состояния.

При обращении требования к устройству:

- поле bmRequestType имеет значение 10000000b, что кодирует IN-передачу в фазе данных (бит 7 равен 1) стандартного требования (биты 6, 5 содержат 0), обращенного к устройству (биты 4-0 содержат 0);
- поле wValue содержит 0 (как и для всех требований GET_STATUS - направленных любому получателю);
- поле wIndex равно 0;
- поле wLength равно 2 - закрепленный размер ответа на это требование в байтах для всех требований GET_STATUS, адресованного любому получателю.

Устройство отвечает на это требование 16-ти битной посылкой, в которой старшие 14 бит зарезервированы для будущего использования и должны быть сброшены в ноль. Бит 1 называется RemoteWakeup и показывает возможность устройства самостоятельно сообщить хосту о выходе из приостановленного состояния. Как мы уже указывали при обсуждении общей организации шины USB, пробуждение ото «сна» от внешнего воздействия это единственный случай, когда устройство потенциально может начать передачу данных без приглашения хоста. Наличием такой возможности управляет флаг RemoteWakeup. Хост может сбрасывать и устанавливать этот флаг командами CLEAR_FEATURE и SET_FEATURE соотвественно. Бит 0 слова состояния устройства носит имя SelfPowered и показывает источник питания устройства: если бит установлен - устройство имеет независимый источник питания, сброшен - устройство питается от шины USB. Хост на способ питания устройства влияния не имеет.

При обращении хоста с требованием GET_STATUS к интерфейсу:

- поле bmRequestType имеет значение 10000001b (субполе получателя содержит код интерфейса);

- поле wValue содержит 0;
- поле wIndex содержит номер интерфейса. Номер интрефейса должен быть допустимым в рамках текущей установленной конфигурации.
- поле wLength имеет значение 2.

Интеррфейс передает 2-х байтную посылку, все биты которой сброшены в ноль и зарезервированы для будущего использования.

Поля пакета требования GET_STATUS, адресованного точке имеют значения:

- bmRequestType – 10000010b (получатель – точка);
- поля wValue и wLength содержат типичные для этого требования значения – 0 и 2 соответственно;
- wIndex содержит номер допустимой в текущем режиме работы устройства точки. Еще раз отметитм: контрольная точка 0 доступна в любом режиме работы устройства.

В слове состояния точки зарезервированы и имеют нулевое значение 15 старших бит, а младший называется Halt и показывает возможность обмена bulk- и interrupt-точек. Если этот бит имеет установлен, значит точка находится в нетрудоспособном состоянии и на любые попытки хоста завязать с ней диалог отвечает отказом, высылая маркер STALL. Хост может управлять этой чертой с помощью команд CLEAR_FEATURE/SET_FEATURE.

Требование CLEAR_FEATURE (очистить свойство)

Посредством этого требования хост имеет возможность сбросить некоторые свойства устройства, интерфейса или конечной точки.

Фаза данных в транзакции обработки этого требования отсутствует, поле wLength для него всегда нулевое. Поле bmRequestType аналогично такому для предыдущего требования с соответствующими получателями за исключением признака направления передачи в фазе данных. Для требования CLEAR_FEATURE и любого другого требования с отсутствующей фазой данных признак направления передачи в поле bmRequestType сброшен в 0 и кодирует несуществующую OUT-передачу в фазе данных. Напомним, что, во-первых, направление запросов в фазе статуса противоположно направлению запросов в фазе данных, а во-вторых, в отсутствии фазы данных запросы в фазе статуса имеют направление IN.

Поле wIndex уточняет получатея требования. Для требования, обращенного к устройству это поле содержит 0. Для требования, направленного к интерфейсу, поле wIndex содержит номер интерфейса. Интерфейс с таким номером должен существовать в текущей активной конфигурации. Для требования CLEAR_FEATURE, адресованного конечной точке, wIndex задает адрес точки. Точка с таким адресом должна поддерживаться в текущей активной конфигурации.

Поле wValue определяет свойство, которое необходимо очистить. Для устройства значение 1 в поле wValue означает, что нужно сбросить флаг RemoteWakeup. Это означает, что устройство теряет возможность уведомлять хост о выходе из приостановленного режима, т.е. теряет единственную возможность самостоятельно начать передачу не дожидаясь запроса хоста. Для конечной точки значение 0 говорит о том, что для нее необходимо сбросить флаг состояния Halt. Это означает перевод точки в рабочее состояние. При отработке этого требования для bulk-IN и interrupt-IN точек необходимо установить марекр DATA0 для следующего пакета данных.

Требование SET_FEATURE (установить свойство)

Требование SET_FEATURE противоположно требованию CLEAR_FEATURE по назначению и аналогично ему по структуре. Требованием SET_FEATURE хост заставляет получателя установить свойство, которое может быть сброшено аналогичным требованием CLEAR_FEATURE.

Фаза данных при обработке этого требования также отсутствует, поле wLength для него всегда нулевое. Поле bmRequestType для требований SET_FEATURE, обращенных к устройству, интерфейсу и точке имеют соответственно значения 00000000b, 00000001b и 00000010b.

Для требования, адресованного устройству, значение 1 в поле wValue указывает, что устройство должно установить флаг RemoteWakeup, поле wIndex при этом содержит 0. Установка флага RemoteWakeup означает, что устройство получает возможность самостоятельно оповещать хост о выходе из приостановленного режима вследствии внешнего воздействия. Во всех остальных случаях устройство имеет право начать передачу только с запроса хоста.

Для устройств, работающих в высокоскоростном режиме, в поле wValue допустимо значение 2, обязывающее устройство перейти в тестовый режим и провести тест, номер которого указан в старшем байте поля wIndex.

Для требования, адресованного интерфейсу, его номер указывается в поле wIndex. Интерфейс с таким номером должен существовать в активной конфигурации.

Для требования, адресованного точке, ее адрес указывается в поле wIndex. Точка с таким адресом должна существовать в текущем режиме работы устройства. Значение 0 в поле wValue определяет, что для точки должен быть установлен флаг Halt. Установка этого флага означает для bulk- или interrupt-точки приостановку нормальной работы: на все запросы хоста она должна отвечать маркером STALL.

Требование SET_ADDRESS (установить адрес)

Требование SET_ADDRESS применяется для установки адреса устройству на шине. Корректными адресами на шине USB являются числа от 1 до 127, а также значение 0, указывающее что устройство является неадресованным.

Новый адрес устройства оно получает в поле wValue данного требования. Фаза данных при обработке тебования SET_ADDRESS отсутствует - значение поля wLength равно нулю; поле wIndex не используется и содержит ноль. Получателем данного требования разумеется является только устройство, поле bmRequestType также содержит 0.

Требование GET_DESCRIPTOR (получить описание)

Требование GET_DESCRIPTOR используется для получения описаний определеного типа.

Прежде всего стоит следует обратить внимание на следующее. Все описания, используемые в стандарте USB, можно условно разделить на два вида:

- описания общего назначения, применяемые для всех USB-устройств;
- описания, специфичные для USB-устройств некоторого класса, например, класса HID.

Что касается требований GET_DESCRIPTOR на получение общих описаний, то для них получателем может быть только устройство, что закреплено в спецификации USB. При запросе специфичных описаний получателем требований может являться не только устройство. Например, при запросе HID-специфичных описаний, получателем требований является интерфейс, что закреплено в спецификации HID.

Рассмотрим подробнее формат требования.

Поле bmRequestType для требований GET_DESCRIPTOR содержит значения 100xxxxxb, где xxxxx – двоичный код получателя, которому адресовано требование. Получатели устройство и интерфейс кодируются значениями 00000 и 00001. Список всех кодов получателей для стандартных требований USB см. в табл. 2. Значения указанного вида в поле bmRequestType означают IN-передачу в фазе данных обработки стандартного требования, обращенного соответствующему получателю.

Старший байт поля wValue определяет тип описания, которое требуется получить. Применяемые значения, их смысл, а также смысл младшего байта, уточняющего тип описания, приведены в табл. 4.

Таблица 4. Коды типов описаний USB-устройств

Значение	Тип описания	Назначение младшего байта
0x01	Устройство	Не используется, всегда 0
0x02	Конфигурация	Индекс конфигурации
0x03	Строковое описание	Индекс строкового описания
0x06	Устройство при работе в другом скоростном режиме	Не используется, всегда 0
0x07	Конфигурация при работе в другом скоростном режиме	Не используется, всегда 0
Описания, специфичные для HID-устройств		
0x21	Описание HID	Не используется, всегда 0
0x22	Описание HID-репорта	Не используется, всегда 0

Значения 0x06 и 0x07 применимы только для устройств, работающих в высокоскоростном режиме. Значения 0x21 и 0x22 применимы только для устройств HID-класса.

В устройстве может быть несколько описаний типа конфигурация и строка, поэтому в младшем байте поле wValue задается индекс описания этого типа. Для других типов описаний младший байт не используется и должен быть нулевым.

Поле wIndex нулевое для всех типов общих описаний кроме строки. Для строковых описаний в поле wIndex задается номер кодовой страницы. Если в качестве номера страницы будет задан 0, то устройство должно вернуть первое доступное описание строки с заданным индексом. Для HID-описаний поле wIndex содержит номер интрефейса, которому обращено требование.

Длина описания заранее не известна, поэтому хост обычно считывает его за два этапа. На первом этапе он задает в поле wLength некоторое ориентировочное значение. Устройство в ответ на требование с таким прогнозируемым значением размера передает начальные wLength байт описания. Первым байтом описания (полный формат описаний мы рассмотрим в следующей статье) передается его размер. Далее хост, повторяет требование с уточненным значением поля wLength.

Требование SET_DESCRIPTOR (установить описание)

Применяется для установки описаний заданного типа. Это требование аналогично требованию GET_DESCRIPTOR по структуре его SETUP-пакета и противоположно ему по действию.

Требование GET_CONFIGURATION (получить конфигурацию)

Требование дает возможность получить номер текущей конфигурации устройства.

Поле bmRequestType для этого требования равно 10000000b, а поле wLength содержит значение 1. Вместе эти поля передают, что хост расчитывает получить от устройства номер его активной конфигурации в фазе данных в посылке размером 1 байт. Возвращаемое значение 0 является признаком того, что устройство не сконфигурировано.

Поля wValue и wIndex для этого требования не несут полезной информации и содержат нулевые значения.

Требование SET_CONFIGURATION (установить конфигурацию)

Это требование предназначено для установки конфигурации устройства.

Поле bmRequestType содержит значение 00000000b, фаза данных при обработке этого требования отсутствует и поле wLength содержит нулевое значение. Поле wIndex не используется и также содержит нулевое значение. Номер устанавливаемой конфигурацтт передается в поле wValue.

Установка конфигурации номер 0 означает деконфигурирование устройства. Описания доступных конфигураций хост получает в ответ на требование GET_DESCRIPTOR. При выборе доступной конфигурации отличной от 0, необходимо кроме того выполнить установку интерфейса 0 в данной конфигурации и альтернативной установки 0. Напомним, что любая конфигурация содержит хотя бы один интерфейс, а любой интерфейс содержит хотя бы одну альтернативную установку. Установка требуемых интерфейса и альтернативной установки осуществляется посредством требования SET_INTERFACE. Для всех bulk- и interrupt-точек направления IN в альтернативной требуется установить маркер DATA0 для следующего пакета данных. Флаг Halt сбрасывается для всех конечных точек.

Требование GET_INTERFACE (получить интерфейс)

С помощью этого требования хост может получить номер текущей альтернативной установки указанного интерфейса активной конфигурации устройства.

Это требование хост адресует интрефейсу, от которого в фазе даных ожидает получить номер альтернативной установки в пакете данных размером 1 байт: поле bmRequestType имеет значение 10000001b, а поле wLength значение 1. Поле wValue не исполоьзуется, а поле wIndex содержит номер интерфейса, к которому обращено требоваание.

Требование SET_INTERFACE (установить интерфейс)

Требование SET_INTERFACE используется хостом для выбора альтернативной установки в заданном интерфейсе активной конфигурации устройства.

Хост этим требованием обращается к интерфейсу, номер которого задан в поле wIndex, указывая ему установить альтернативную установку с номером, опреляемым полем wValue. Фаза данных при обработке этого требования отсутствует . Поле bmRequestType содержит значение 00000001b, а поле wLength – 0.

Номера доступных интерфейсов и их альтернативных установок хост узнает с помощью требования GET_DESCRIPTOR. При смене альтернативной установки для всех bulk- и interrupt-точек IN-направления нужно установить маркер DATA0 для следующего пакета данных. Флаг Halt для всех точек сбрасывается.

Требование SYNCH_FRAME (синхронизировать кадр)

С помощью этого требования хост контролирует номер кадра синхронизации для конечной точки в изохронном режиме передачи с неявной синхронизацией.

ТРЕБОВАНИЯ КЛАССА HID

Свойства USB-устройств класса HID и основные понятия, применяемые к этому классу рассматриваются в отдельной статье, посвященной этой теме. В данной статье рассматриваются лишь некоторые основные требования для устройств HID класса.

Требование GET_REPORT (получить репорт)

Требование GET_REPORT позволяет хосту получить от устройства репорт – данные в специальном формате – через контрольную точку.

Как это ясно из назначения требования, в транзакции его обработки используется фаза данных направления IN. Поле wLength содержит длину репорта. Поле bmRequestType имеет значение 10100001b, что кодирует требование класса, обращенное к интерфейсу, при обработке которого присутствует IN-фаза данных. Поле wIndex содержит номер интерфейса, к которому обращено требование. Поле wValue кодирует тип репорта – младший байт – и его идентификатор – старший байт. Идентификатор репорта может не использоваться: в этом случае старший байт репорта должен содержать 0. Для определения типа репорта используются следующие значения:

- 1 – INPUT-репорт,
- 2 – OUTPUT-репорт,
- 3 – FEATURE-репорт,
- значения 4-255 зарезервированы для будущего использования.

Требование SET_REPORT (установить репорт)

Это требование аналогично предыдущему по назначению и кодированию полей и противоположно ему по назначению: с помощью требования SET_REPORT хост может передать данные устройству через контрольную точку.

Поле bmRequestType содержит значение 00100001b. В фазе данных направления OUT хост передает устройству репорт размером, указанном в поле wLength. В младшем байте поля wValue указывается тип репорта, в старшем – его идентификатор. Поле wIndex содержит номер интерфейса, к которому обращено требование.

Литература

1. Universal Serial Bus Specification Revision 2.0. www.usb.org
2. Device Class Definition for Human Interface Devices (HID) v.1.11. www.usb.org
3. Чекунов Д. Стандартные требования USB. Современная электроника. 2004, № 12

3. Axelson J. USB Complete. 3-d ed.
4. Hyde J. USB Design by Example.

5. Агуров П. В. Интерфейс USB. Практика использования и программиррования. – СПб.: БХВ-Петербург, 2005.