Практикум программиста USB-устройств Часть 2. Разработка аппаратно-программного ядра USB-устройства

Дмитрий Чекунов (г. Ижевск)

Что необходимо для запуска USB-устройства? Драйвер – уже имеется. Теперь собираем типовую схему, пишем программу управления… Раз, два… включаем!

Типовая схема включения EZ-USB FX2LP

Для любого микроконтроллера (МК) существует типовая схема включения, в которой имеется минимальный набор элементов, обеспечивающий его работоспособность. Подобная схема практически без изменений присутствует во всех устройствах, построенных на базе этого МК, и её можно условно выделить в некое самодостаточное ядро.

Структурная схема такого ядра для нашего USB-устройства представлена на рис. 1. В его состав входят следующие элементы:

- микроконтроллер CY7C68013A;
- DC/DC-преобразователь с выходным напряжением +3,3 В;
- формирователь сигнала RESET с активным уровнем лог. 0;
- микросхема памяти с интерфейсом I²C;
- кварцевый резонатор на 24 МГц;
- разъём USB-B;
- пара переключателей и кнопка.

Подключение устройства к шине USB осуществляется с помощью разъёма USB-B. Напряжение питания от внутреннего источника шины USB (5 В) попадает на устройство при замкнутом переключателе S1, а если переключатель разомкнут, то устройство должно иметь собственный источник питания. В обоих случаях входное напряжение поступает на DC/DC-преобразователь, который формирует рабочее напряжение питания ядра (3,3 В). При появлении рабочего напряжения запускается тактовый генератор МК, а формирователь сигнала RESET генерирует сигнал для сброса МК в начальное состояние. После окончания сигнала RESET внутренняя логика модуля I²C (встроенного в FX2LP) аппаратно осуществляет поиск подключенных микросхем с загрузочной записью. При замкнутом переключателе S3 и наличии такой записи в микросхеме памяти происходит загрузка программы в МК. Далее FX2LP регистрируется на шине USB, и устройство готово к работе.

В некоторых случаях, когда программа, записанная в микросхему памяти, имеет критические ошибки, может потребоваться отключение микросхемы от шины I²C на время запуска устройства. При разомкнутом переключателе S3 память МК остаётся незагруженной, и модуль USB самостоятельно регистрируется на шине, топология устройства в таком случае будет соответствовать представленной [1, рис. 2].

Кнопка S2 позволяет оператору перезагрузить устройство без отключения его от шины USB.

Как видим, элементы, входящие в состав ядра USB-устройства, хорошо знакомы и повсеместно используются разработчиками радиоэлектронной аппаратуры. Поэтому сейчас проектирование принципиальной схемы не вызовет затруднений.

На рис. 2 в качестве примера представлен один из возможных вариантов принципиальной схемы ядра. Микросхема DA1 (MAX6471UT33BD3) совмещает в себе функции DC/DC-преобразователя и формирователя сигнала RESET. Без дополнительных делителей и обратных связей эта ИС вырабатывает необходимое рабочее напряжение 3,3 В. Также у неё имеется вход для внешнего сигнала, при появлении которого формируется сигнал RESET. Микросхема DD1 (AT24C128) предназначена для хранения закодированной программы и способна вместить отформатированный код объёмом до 16 Кб. Адрес микросхемы на шине I²C соответствует единице, что определено требованиями, предъявляемыми к загрузочной микросхеме [2]. Фильтрующие конденсаторы и «подтягивающие» резисторы установлены в соответствии с общеизвестными требованиями. При использовании МК типа CY7C68013 (семейство EZ-USB FX2) конденсаторы C11 и C12 должны иметь ёмкость 22 пФ.

Представленное ядро позволяет программисту попрактиковаться в

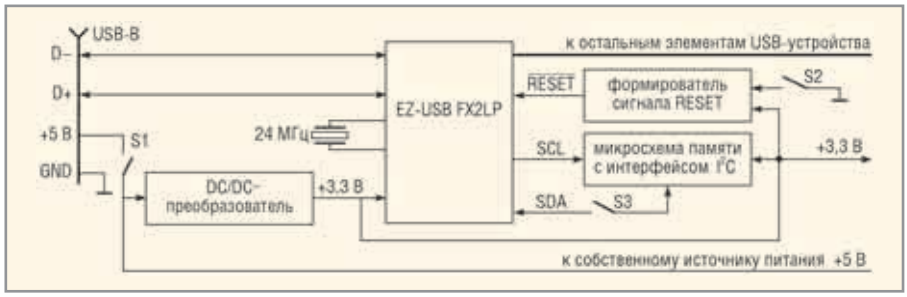

Рис. 1. Структурная схема ядра USB-устройства

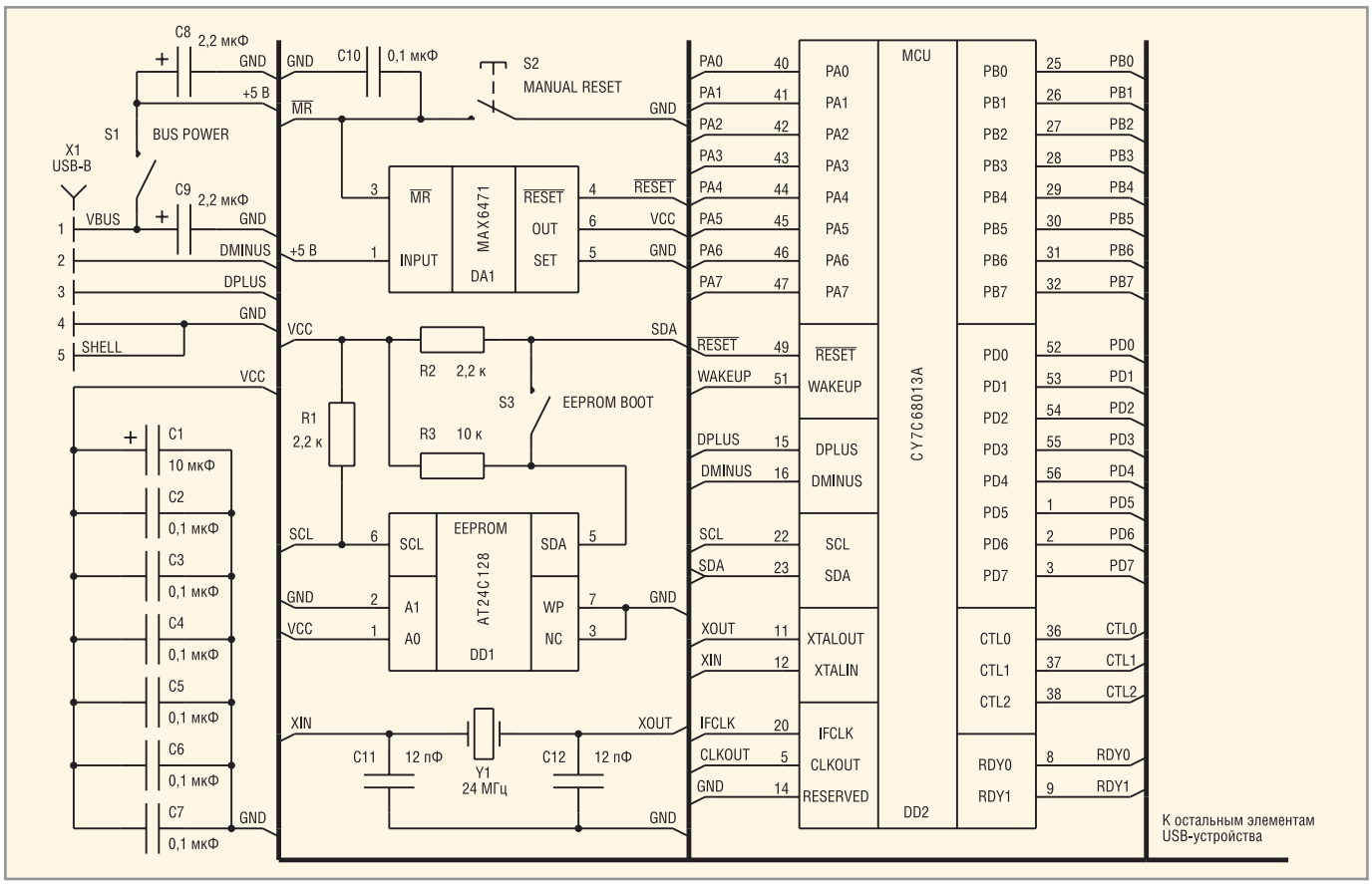

Рис. 2. Принципиальная схема ядра USB-устройства на базе CY7C68013A

обработке USB-требований и в работе «малых» точек МК.

Структура программной части проекта

Программистам хорошо известно, что текст программы в процессе программирования разрастается очень быстро, и уже через некоторое время в длинном тексте программы сам автор начинает ориентироваться с большим трудом. Постороннему человеку разобраться в такой программе тем более сложно.

Для того чтобы наш проект был легко управляемым, а его структура – интуитивно понятной, разделим его на несколько файлов. Имя файла будет соответствовать модулю, для которого он содержит подпрограммы.

Предварительная структура программной части проекта представлена на рис. 3. Рассмотрим поподробнее назначение файлов:

- mydevice.asm – главный файл проекта. Он явно включает в себя три файла и содержит директивы для транслятора, основную таблицу векторов прерываний и короткую универсальную программу, алгоритм которой представлен на рис. 4. Логика основной программы сводится к инициализации программных переменных, отключению от шины USB, деинициализации системы прерываний (необходимо, если программа загружена через USB), инициализации устройства и системы прерываний, подключению к шине USB и зацикливанию («засыпанию») с дальнейшей работой по прерываниям;
- fx2lp.asm – файл с предопределёнными именами регистров для МК семейства EZ-USB FX2LP;
- var.asm – файл, описывающий переменные программной и аппаратной частей;
- config.asm – файл конфигурации, предназначенный для быстрого добавления или удаления файла в составе всего проекта. Он явно включает в себя основную часть файлов;
- macros.asm – файл с макросами, заменяющими наборы наиболее часто используемых команд;
- util.asm – файл с вспомогательными подпрограммами общего назначения;
- table.asm – дополнительная таблица векторов прерываний для USB и GPIF/FIFO;
- system.asm – содержит подпрограммы инициализации и деинициализации системы;
- int4gpif.asm – обработчики прерывания для входа INT4 или для прерываний, формируемых модулем GPIF;
- intusb.asm – обработчики прерываний для модуля USB. Здесь содержатся обработчики для системных прерываний шины USB и для контрольных транзакций. Обработчики для прерываний, формируемых при обращениях к прочим точкам,

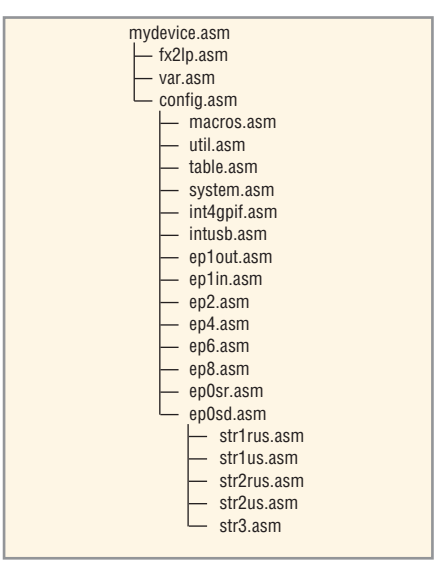

Рис. 3. Структура программной части проекта

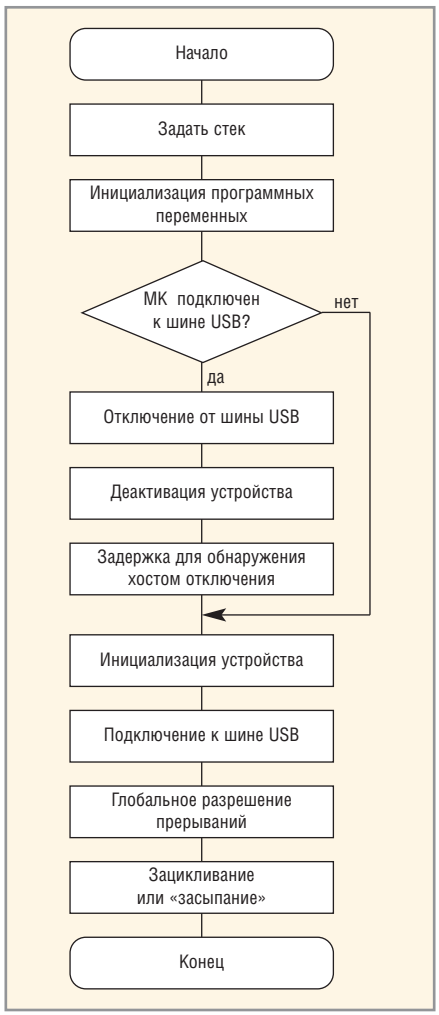

Рис. 4. Алгоритм головной программы

находятся в файлах с соответствующим номером точки;

- ep1out.asm – обработчики прерываний и подпрограммы обслуживания точки EP1OUT;
- ep1in.asm – обработчики прерываний и подпрограммы обслуживания точки EP1IN;
- ep2.asm – обработчики прерываний и подпрограммы обслуживания точки EP2, а также принадлежащего ей FIFO;
- ep4.asm, ep6.asm, ep8.asm – аналогично ep2.asm;
- ep0sr.asm – подпрограммы обслуживания стандартных требований USB (sr – Standard Requests);
- ep0sd.asm – стандартные описания свойств устройства (sd – Standard Descriptions). В него явно включены файлы с закодированными строковыми дескрипторами;
- str1rus.asm, str1us.asm – описание производителя на русском и английском языке соответственно;
- str2rus.asm, str2us.asm – название устройства на русском и английском языке соответственно;
- str3.asm – серийный номер устройства.

На первый взгляд может показаться, что структура проекта очень громоздка и контролировать состояние подпрограмм сложно. Но на начальном этапе большинство подпрограмм мы оформим в виде «заглушек», а в дальнейшем увидим, что однажды написанная подпрограмма больше не нуждается в редактировании. И, таким образом, мы получим некое программное ядро, на основе которого можно оперативно разрабатывать программы для новых устройств.

Приступим к оформлению программной части проекта. Создадим новый каталог и в нём – пустые файлы с именами, показанными на рис. 3, за исключением fx2lp.asm, который можно создать по документации [2] или взять из дополнительных материалов к статье, размещённых на сайте журнала. Скопируем в этот же каталог файл asm.bat, созданный нами ранее [1]. Итак, начнём редактирование файлов.

Файл config.asm. Составим в нём список включаемых файлов в соответствии с рис. 3:

```
$INCLUDE (macros.asm)
$INCLUDE (util.asm)
$INCLUDE (table.asm)
…
$INCLUDE (ep0sr.asm)
$INCLUDE (ep0sd.asm)
```

Файл mydevice.asm. Начинаем файл директивами для управления трансляцией и форматом листинга; набор директив каждый определяет на свой вкус. Потом включаем файлы с предопределёнными именами регистров и программными переменными в соответствии с рис. 3:

```
$INCLUDE (fx2lp.asm)
$INCLUDE (var.asm)
```

Далее объявляем сегмент кода и включаем файл config.asm, поскольку подпрограммы и данные, косвенно содержащиеся в этом файле, должны располагаться в сегменте кода:

```
CSEG AT 0
ljmp main ; переход на начало программы
$INCLUDE (config.asm)
```

Теперь осталось написать код, реализующий алгоритм, который представлен на рис. 4, и в самом конце написать директиву:

```
END ; mydevice.asm конец
```

Эта директива служит для транслятора признаком завершения работы.

Файл var.asm. Зарезервируем программные переменные, которые используются в головной программе. Для переменных общего назначения будем использовать область памяти, начиная с адреса 30h:

```
DSEG AT 30h
stackPoint EQU $ ; значение для указателя стека
```

Файл system.asm. Создадим в нём «заглушки» для функций initSystem и haltSystem.

Файл macros.asm. Напишем несколько макросов для набора часто используемых команд:

- _MOVX_R_ – чтение байта из внешнего ОЗУ;
- _MOVX_W_ – запись байта во внешнее ОЗУ;
- _MOVX_W_A_ – запись аккумулятора во внешнее ОЗУ;
- _SETB_EXT_ – установка бита во внешнем ОЗУ;
- _CLR_EXT_ – сброс бита во внешнем ОЗУ;
- _ALLOC_DB_ – размещение в памяти заданного количества байт;
- _ALLOC_DW_ – размещение в памяти заданного количества слов.

Варианты организации USB-устройства

О наличии встроенной аппаратной поддержки стандартных требований USB в FX2LP уже говорилось ранее [1]. Данное качество может быть весьма привлекательным для начинающих разработчиков, поскольку оно избавит их от необходимости изучать стандартные команды USB. Однако в МК существует возможность программного отключения встроенной поддержки. Казалось бы, что может дать эта возможность – только добавить «лишней» работы по изучению и кодированию обработчиков стандартных требований. Но не будем спешить с выводами и рассмотрим, какие варианты организации устройства получатся в том или ином случае.

Пусть в первом варианте устройство использует встроенную поддержку стандартных требований. Тогда

модуль USB МК самостоятельно выполняет регистрацию на шине с параметрами, которые аппаратно в нём заложены. Таким образом, и топология устройства будет определяться аппаратно (см. [1, рис. 2]). Следовательно, в данном варианте будет присутствовать простая программа, работающая по прерываниям доступных точек при фиксированной аппаратно топологии. Для простых устройств этого вполне достаточно.

Что мы имеем во втором варианте? Если встроенная поддержка стандартных требований будет отключена, то программа должна обслуживать их самостоятельно. В таком случае и все дескрипторы формирует программист. Получается, что топология задаётся программно и может быть сколь угодно сложной или напротив – простой. Следовательно, при организации устройства по второму варианту программист имеет абсолютный контроль над USB-частью МК и может создать сложное устройство или устройство, работающее по протоколу некоторого класса/подкласса.

Какой вариант организации USB-устройства наиболее предпочтителен – каждый может решить для себя сам. Но сразу замечу, что когда возникнет потребность в обслуживании дополнительных (собственных) требований, то всё равно придётся изучить логику работы контрольной точки и методы «общения» с хостом.

Наш проект будет развиваться по второму варианту, поэтому в головной программе необходимо перехватить обслуживание контрольной точки в момент подключения к шине USB. В файле mydevice.asm должна присутствовать строка, где выполняется установка бита RENUM регистра USBCS:

```
mov a,#2
mov dptr,#USBCS
movx @dptr,a
```

Дополнительные возможности FX2LP

В FX2LP, как и во многих современных МК с ядром 8051, имеется два указателя на данные. В области SFR-регистров они занимают следующие адреса: dpl–82h, dph–83h, dpl1–84h, dph1–85h. Для выбора активного указателя используется регистр dps. Состояние его младшего бита (SEL) определяет ячейки, на которые спроецирован указатель dptr. При SEL, равном нулю, команды, использующие dptr, будут обращаться к регистрам dpl и dph, при единице – к регистрам dpl1 и dph1. Независимо от используемых указателем ячеек памяти другая пара регистров доступна и для записи, и для чтения.

Второй dptr весьма полезен для передачи данных во внешней памяти. Но ведущая роль для решения этой задачи принадлежит модулю автоматических указателей (autopointers). Данный модуль введён специалистами фирмы Cypress для обслуживания буферов «больших» точек, где объёмы передаваемых данных могут достигать 1024 байт.

Идея работы модуля заключается в автоматическом приращении содержимого указателей при любом обращении к зарезервированным регистрам во внешнем ОЗУ.

В состав модуля входят 5 регистров специального назначения (SFR) и 2 регистра, расположенных во внешнем ОЗУ. Рассмотрим их назначение и логику взаимодействия.

Регистры autoptrh1, autoptrl1, autoptrh2, autoptrl2 расположены в области SFR и организованы аналогично указателям dptr, то есть они хранят адрес внешней ячейки ОЗУ и загружаются, соответственно, по частям: старший байт адреса в autoptrh1 или autoptrh2, а младший – в autoptrl1 или autoptrl2. Каждому указателю сопоставлен регистр, расположенный во внешнем ОЗУ МК. Для autoptr1 это XAUTODAT1, а для autoptr2 – XAUTODAT2.

Управление режимами работы автоматических указателей и общее разрешение работы этой системы осуществляется с помощью регистра autoptrsetup (SFR – 0AFh). Состояние битов APTR1INC и APTR2INC в этом регистре определяет логику изменения адреса, хранящегося в указателе. Когда бит APTR1INC установлен, при любом обращении команды movx к регистру XAUTODAT1 происходит обмен данными с ячейкой внешнего ОЗУ, адрес которой находится в autoptrh1/autoptrl1, и последующее автоматическое приращение этого адреса. Таким образом, циклическое обращение к регистру XAUTODAT1 позволяет получить доступ к последовательности ячеек памяти. В случае, если бит APTR1INC сброшен, адрес указателя не изменяется. Аналогичные действия происходят для связки: бит APTR2INC, указатель autoptrh2/autoptrl2 и регистр XAUTODAT2.

Бит APTREN в регистре autoptrsetup используется для глобального разрешения (1) или запрещения (0) работы модуля автоматических указателей.

Организация системы прерываний USB и GPIF/FIFO

Раньше мы уже встречали упоминание о том, что запрос на прерывание от модулей USB и GPIF/FIFO формируется множеством возможных событий, а также о том, что для быстрого вызова соответствующего обработчика существует система автоматического перехода на вектор прерывания. Сейчас рассмотрим подробно логику работы этой системы и создадим в нашем проекте условия для её функционирования.

Механизм работы системы автоматического перехода представлен на рис. 5. Как видим, в памяти программ МК имеются основная и дополнительная таблицы векторов прерываний. Векторы для прерываний USB и GPIF/FIFO в основной таблице содержат команду длинного безусловного перехода на дополнительную.

Например, при возникновении запроса HISPEED (переход устройства в высокоскоростной режим работы) активизируется источник прерывания модуля USB с номером 5. Этот номер заносится во временный регистр INT2VEC со сдвигом на 2 бита влево, что в результате даёт число, равное смещению в дополнительной таблице векторов. В нашем случае INT2VEC будет содержать смещение 14h.

Далее система автоматического перехода считывает команду вектора прерывания USB из основной таблицы (адреса 43h, 44h, 45h), декодирует её и на место последнего байта в адресе перехода подставляет смещение из INT2VEC. В результате подстановки получается новый адрес перехода (0114h), сопоставленный модулю и событию, инициировавшему прерывание. После перехода на адрес 0114h выполняется команда следующего перехода – на соответствующий обработчик прерывания (isrHiSpeed).

Работа системы автоматического перехода для прерываний GPIF/FIFO осуществляется аналогично, с той лишь разницей, что в качестве вре-

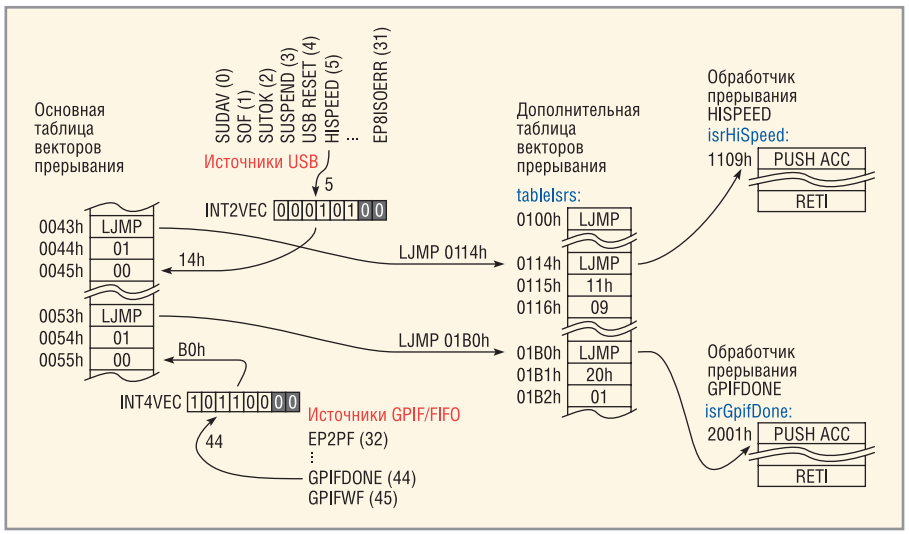

Рис. 5. Механизм работы системы автоматического перехода

менного регистра для получения смещения используется INT4VEC.

Как можно заметить, использование данной системы значительно упрощает работу программиста и освобождает его от «ручного» поиска источника прерывания.

Добавим в наш проект код, необходимый для работы системы автоматического перехода.

Файл mydevice.asm. Основная таблица должна находиться между переходом на начало программы и строкой включения файла config.asm. Добавим команды перехода на дополнительную таблицу:

```
ljmp main ; переход на начало
программы
ORG 43h
ljmp tableIsrs
ORG 53h
ljmp tableIsrs
```

Файл system.asm. В функцию initSystem, используемую для инициализации всего устройства, добавим разрешение на работу системы автоматического перехода. Управление разрешением работы системы осуществляется с помощью регистра INTSETUP:

- INTSETUP.3 – разрешение для модуля USB;
- INTSETUP.0 – разрешение для модуля GPIF/FIFO.

```
mov dptr,#INTSETUP
mov a,#8
movx @dptr,a ; разрешаем
автоматический переход для USB
```

Разрешаем общее прерывание для модуля USB:

```
orl eie,#1 ; разрешение общего
прерывания модуля USB
```

Файл table.asm. Данный файл предназначен для создания дополнительной таблицы векторов прерываний. При её создании необходимо соблюдать ряд условий [3]. Во-первых, адрес начала таблицы должен быть кратен числу 100h, так как последний байт адреса будет автоматически заменён смещением. Во-вторых, для всех событий модулей USB и GPIF/FIFO используется общая таблица, так как нумерация событий сквозная. Смещение от начала таблицы до нужного вектора получается путём сдвига номера события на 2 бита влево. Нумерация событий эквивалентна их приоритетам и представляет собой последовательность: 0 – SUDAV, 1 – SOF, 2 – SUTOK, ..., 31 – EP8ISOERR, 32 – EP2PF, ..., 45 – GPIFWF (см. таблицы 2, 3 [1]). В-третьих, в дополнительной таблице каждый вектор должен содержать команду длинного безусловного перехода на подпрограмму обслуживания.

Начнём описывать таблицу, следуя вышеизложенным условиям. Для того чтобы адрес начала таблицы был кратен 100h, добавим:

```
offsetTable EQU LOW($)
IF offsetTable <> 0
ORG $ + (100h - offsetTable)
ENDIF
tableIsrs: ; обработчики
прерываний по USB
```

Теперь формируем векторы прерываний. Для того чтобы заполнить пустоты между векторами, резервируем 1 байт. В неиспользуемые векторы вставляем команду перехода на пустой обработчик (isrEmpty)

```
ljmp isrSudav ; смещение 0
DB 0
ljmp isrSof ; смещение 4
DB 0
ljmp isrSutok ; смещение 8
DB 0
...
ljmp isrGpifDone ; смещение 0B0h
DB 0
ljmp isrGpifWf ; смещение 0B4h
```

В результате создания дополнительной таблицы получим множество команд перехода на несуществующие функции. Временно создадим «заглушки» для этих обработчиков с единственной командой reti. Размещение обработчиков в файлах будет следующим:

- intusb.asm – isrSudav, isrSof, isrSutok, isrSuspend, isrUsbReset, isrHiSpeed, isrEp0Ack, isrEp0In, isrEp0Out, isrIbn, isrEp0Ping, isrErrLimit;
- ep1out.asm – isrEp1Out, isrEp1Ping;
- ep1in.asm – isrEp1In;
- ep2.asm, ep4.asm, ep6.asm, ep8.asm – isrEpx, isrEpxPing, isrEpxIsoErr, isrEpxPf, isrEpxEf, isrEpxFf (где вместо x необходимо подставить числа 2, 4, 6, 8 соответственно);
- int4gpif.asm – isrGpifDone, isrGpifWf;
- table.asm – isrEmpty.

После создания всех «заглушек» можно выполнить трансляцию программы и убедиться, посмотрев файл листинга mydevice.lst, что дополнительная таблица векторов начинается с адреса, кратного 100h.

Работа контрольной точки

С форматом контрольной транзакции подробно мы уже знакомились ранее [4]. На первый взгляд, этот тип передачи может показаться сложным. Но встроенный модуль USB МК выполняет аппаратно всю рутинную работу по распознаванию, подтверждению и передаче, выделяя лишь ключевые события, по которым можно чётко судить о текущем состоянии транзакции.

Формат контрольной транзакции с точки зрения МК показан на рис. 6. Рассмотрим действия, которые выполняются аппаратно, и действия, которые должен выполнить программист, работая с контрольной точкой.

В момент начала контрольной транзакции модуль USB обнаруживает

маркер запроса SETUP и формирует прерывание SUTOK. Одновременно происходит аппаратная установка бита HSNAK (регистр EP0CS), сопоставляемого маркеру подтверждения, который передаётся в фазе статуса. В этот же момент происходит сброс бита STALL (регистр EP0CS), так как фаза SETUP обязательно должна завершиться маркером ACK. Таким образом, даже если FX2LP не успевает обработать запросы, а в контрольной транзакции уже началась последняя фаза (статуса), модуль USB будет аппаратно выдавать маркер подтверждения NAK, растягивая фазу статуса до тех пор, пока FX2LP не снимет бит HSNAK.

Прерывание SUTOK используется только для отладочных целей, его появление предупреждает о возможной потере данных в буфере SETUPDAT, обусловленной предстоящим приходом пакета SETUP.

Модуль USB после приёма пакета данных SETUP и проверки его целостности помещает данные (8 байт) в буфер SETUPDAT, формирует прерывание SUDAV и подтверждает получение данных маркером ACK. В случае повреждения принятых данных соответственно не формируется прерывание SUDAV и отсутствует подтверждение ACK, что трактуется хостом как потеря пакета и вынуждает его повторить передачу.

Прерывание SUDAV является ключевым в контрольной транзакции. Начиная обрабатывать его, желательно скопировать пакет из буфера SETUPDAT в локальный буфер. Далее проверяем корректность полученного требования, определяем необходимость дополнительных данных (наличие фазы данных) и приступаем к его выполнению. Если требование некорректно, то независимо от текущей фазы (данных или статуса) устанавливаем бит STALL (регистр EP0CS). В этом случае модуль USB при первой же передаче маркера подтверждения выдаст на шину STALL, что послужит для хоста сигналом о попытке передачи недопустимого для устройства требования.

Итак, выполняем требование. Допустим, хост запросил описание устройства, что подразумевает обязательное присутствие фазы данных. Хост посылает пакеты запроса IN, пытаясь получить данные, но мы ещё не успели их подготовить. В этом случае модуль USB самостоятельно выдаёт

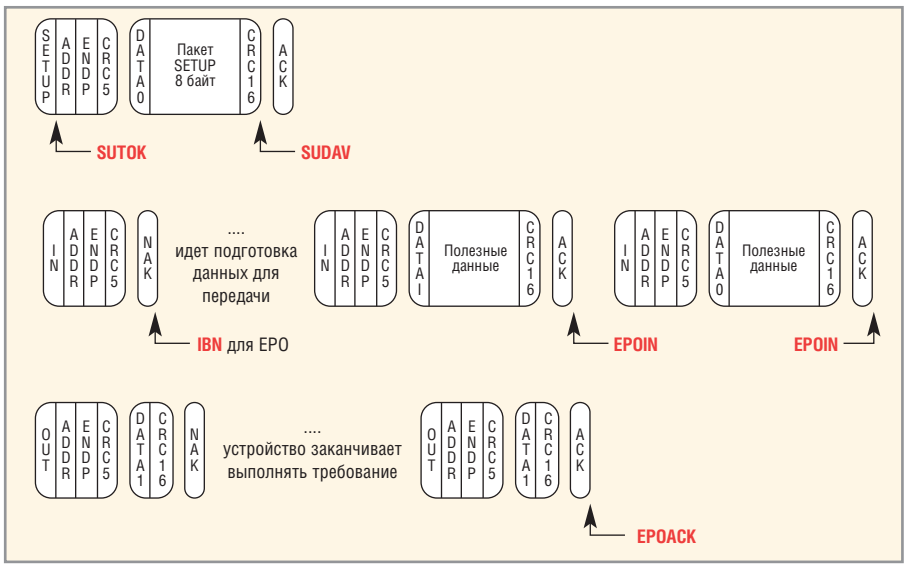

Рис. 6. Контрольная транзакция с позиции микроконтроллера

маркер подтверждения NAK и одновременно формирует прерывание IBN. Появление данного прерывания сообщает о начале фазы данных, а если сказать точнее, то об отсутствии данных в буфере точки, к которой поступает запрос IN.

Подготовленные данные можно передать двумя способами: «ручным» и автоматическим. При передаче «ручным» способом мы должны поместить максимально возможную порцию данных в буфер EP0BUF (размер 64 байта) и записать количество байт в этой порции сначала в регистр EP0BCH (старшая часть количества), а потом в EP0BCL (младшая часть). Сразу после записи в регистр EP0BCL буфер данных EP0BUF поступает под управление модуля USB, который при получении следующего запроса IN передаст пакет данных и, получив подтверждение ACK, выставит прерывание EP0IN. Буфер EP0BUF возвращается под управление МК. Прерывание EP0IN показывает, что буфер EP0BUF пуст и готов для загрузки новой порции данных.

При автоматическом способе передачи данных используется система Setup Data Pointer (SDP). Она предназначена для отправки хосту стандартных дескрипторов, и для её корректной работы необходимо, чтобы в дескрипторе устройства поле bMaxPacketSize0 имело значение 64 (максимальный размер пакета данных в одноименной фазе). Для разрешения работы системы необходимо установить бит SDPAUTO (регистр SUDPTRCTL), а для её запуска записать в регистры SUDPTRH и SUDPTRL адрес из внешнего ОЗУ МК, где располагается соответствующий дескриптор. Система автоматически вычисляет полный объём передаваемых данных, анализируя запрошенное хостом количество байт из буфера SETUPDAT и размер дескриптора, заданный в нём самом (берётся наименьшее число). Далее система SDP подготавливает пакеты данных, помещает их в EP0BUF и передаёт для отправки модулю USB. При освобождении буфера EP0BUF в него незамедлительно помещается следующий пакет, и операция передачи повторяется. Прерывания EP0IN при использовании системы SDP не генерируются, и о завершении работы этой системы узнать невозможно, но в этом и нет необходимости, поскольку, передав дескриптор, устройство фактически выполнило требование хоста.

Программист имеет возможность узнать о полном завершении контрольной транзакции по прерыванию EP0ACK. Всегда, закончив выполнять требование, необходимо снять бит HSNAK (регистр EP0CS), и тогда при следующем запросе модуль USB автоматически закончит фазу статуса маркером подтверждения ACK и сгенерирует прерывание EP0ACK, что и является признаком конца контрольной транзакции.

После знакомства с механизмом работы модуля USB в части обслуживания контрольной точки можно заметить, что обработка требования сводится к анализу пакета в буфере SETUPDAT, выполнению требования (иногда присутствует передача дополнительных данных) и снятию бита

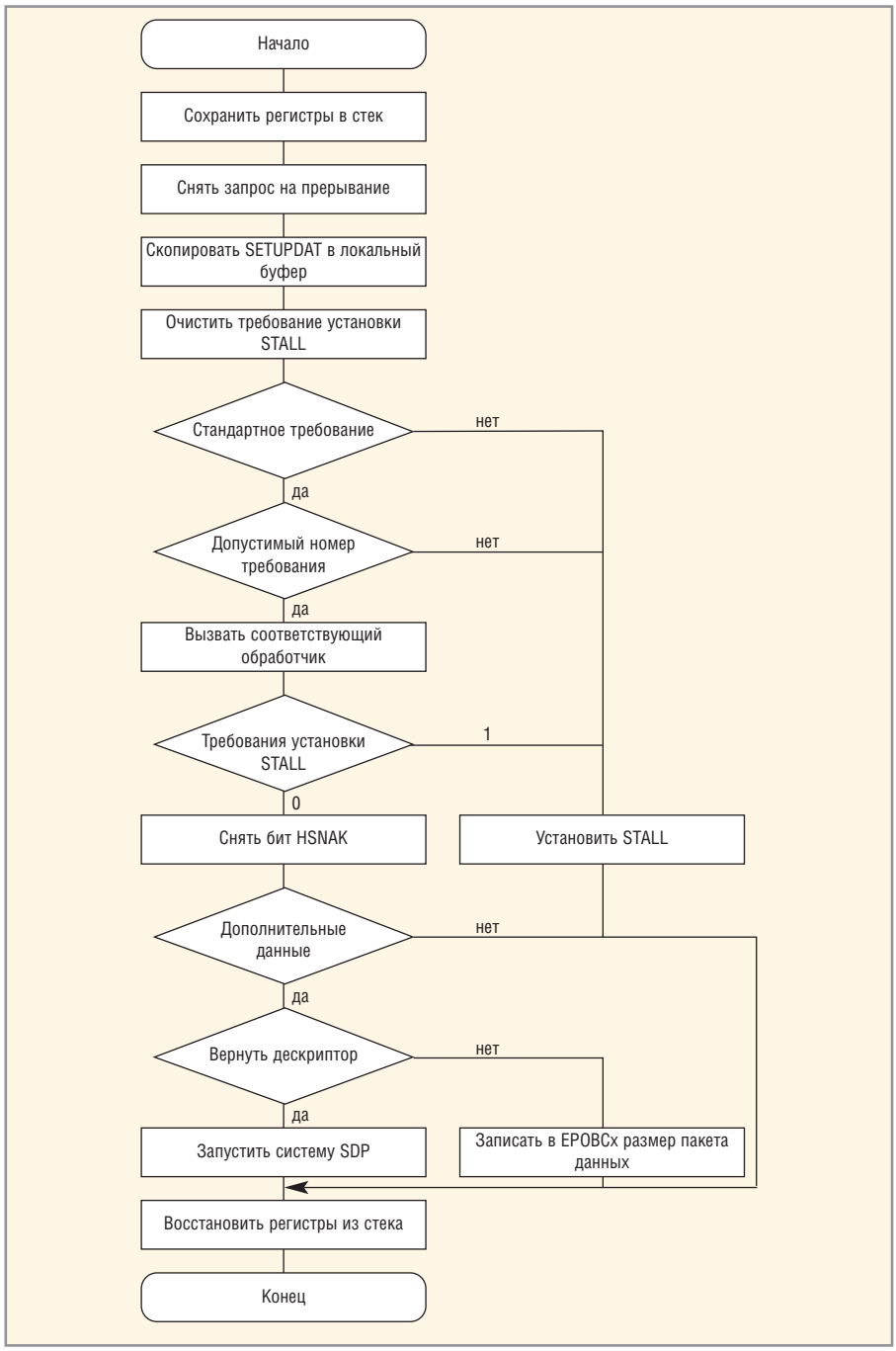

Рис. 7. Алгоритм обработки прерывания SUDAV

HSNAK для завершения контрольной транзакции.

В таком случае добавим в наш проект обработчик прерывания SUDAV, который будет выполнять предварительный контроль полученного требования, запускать соответствующую подпрограмму его обслуживания, при необходимости посылать данные (фаза данных) и завершать фазу статуса. Алгоритм такого обработчика представлен на рис. 7. Рассмотрим поподробнее выполняемые действия. Сохранение регистров в стек и снятие запроса на прерывание – это стандартные шаги, исполняемые в начале любого обработчика прерывания. Далее копируем данные из буфера SETUPDAT в локальный буфер, и все дальнейшие манипуляции проводим только над ним. Перед началом проверок очищаем программную переменную, отвечающую за необходимость установки маркера STALL. Проверяем полученное требование на принадлежность к стандартным, и если оно является таковым, то проверяем заданный номер требования. В обоих случаях при ошибке устанавливаем STALL, сообщая о некорректном требовании.

Если же проверки пройдены успешно, вызываем обработчик для соответствующего требования и после окончания его работы анализируем признак установки маркера STALL. Если он не установлен, то требование выполнено успешно. Далее снимаем бит HSNAK, то есть фактически готовимся завершить фазу статуса. Здесь может возникнуть естественный вопрос – а как же фаза данных? В действительности сброс бита HSNAK никак не может повлиять на фазу данных. Если сейчас выполняется фаза данных, то хост «нетерпеливо» посылает пакеты запроса IN, а модуль USB отвечает маркером NAK (см. рис. 6). Состояние бита HSNAK будет иметь значение только тогда, когда начнётся фаза статуса.

А вот следующим действием проверяем, ждёт ли хост от нас данные (наличие фазы данных). Если нет, то восстанавливаем регистры из стека и выходим из обработчика прерывания.

В случае присутствия фазы данных анализируем характер передаваемых данных и выбираем способ их передачи. Если предстоит отправить дескриптор, то воспользуемся системой SDP, иначе записываем в регистры EP0BCH, EP0BCL размер отправляемого пакета (данные в буфер EP0BUF должен поместить обработчик требования). Независимо от выбранного способа передачи данных, после выполнения описанных действий выходим из обработчика прерываний. Все дальнейшие действия по передаче данных и завершению контрольной транзакции модуль USB выполнит аппаратно.

Теперь можем добавить необходимый код в проект.

Файл var.asm. Согласно таблице 1 зададим константы распределения полей пакета SETUP в буфере SETUPDAT:

```
bRT  EQU   0   ;  смещение  для
bmRequestType
bR   EQU 1 ; смещение для bRequest
wVL  EQU 2 ; смещение для wValueL
wVH  EQU 3 ; смещение для wValueH
wIL  EQU 4 ; смещение для wIndexL
wIH  EQU 5 ; смещение для wIndexH
wLL  EQU 6 ; смещение для wLengthL
wLH  EQU 7 ; смещение для wLengthH
```

Регистры r0 и r1 (из банка 0) оставим для общего использования, а начиная с адреса 2, будем объявлять системные переменные:

```
_ptrDest       EQU r0
; указатель на приёмник данных
```

Таблица 1. Распределение полей пакета SETUP в буфере SETUPDAT

Смещение	Поле	Назначение
0	bmRequestType	Тип требования
1	bRequest	Номер требования
2	wValueL	Младший байт поля «значение»
3	wValueH	Старший байт поля «значение»
4	wIndexL	Младший байт поля «индекс»
5	wIndexH	Старший байт поля «индекс»
6	wLengthL	Младший байт поля «длина»
7	wLengthH	Старший байт поля «длина»

```
_ptrSrc EQU r1 ; указатель на
источник данных
DSEG AT 2
usbBufSetup:   DS 8 ; локальный
буфер для SETUP пакета
```

В области переменных общего назначения (с адреса 30h) объявим счётчик:

```
cntTmp: DS 1 ; счетчик общего
назначения
```

В битовой области объявляем флаги:

```
BSEG AT 0
flagStallEp0:  DBIT 1 ; требо-
вание установки STALL для EP0
flagGetDesc:   DBIT 1
; признак передачи дескриптора
```

Файл system.asm. В функцию initSystem добавляем разрешение прерывания SUDAV:

```
mov a,#1
mov dptr,#USBIE
movx @dptr,a
; разрешаем прерывание SUDAV
```

Включаем систему передачи дескрипторов SDP:

```
mov a,#1
mov dptr,#SUDPTRCTL
movx @dptr,a ; разрешаем автома-
тическую передачу дескрипторов
```

Для быстрого обращения к массивам данных во внешнем ОЗУ включаем дополнительную функцию МК – автоинкремент указателей на данные:

```
mov autoptrsetup,#7 ; разрешаем
систему авт. инкремента
```

Файл util.asm. Для копирования данных из буфера SETUPDAT (внешнее ОЗУ) в локальный буфер usbBufSetup (внутреннее ОЗУ) напишем простую подпрограмму movExt2Into.

Файл intusb.asm. В обработчик прерывания SUDAV добавляем код, реализующий алгоритм, представленный на рис. 7. Подпрограммы обслуживания требований, вызываемые из обработчика, создадим в виде «заглушек» в файле ep0sr.asm.

(Продолжение следует)

Литература

1. *Чекунов Д.* EZ-USB FX2LP – универсальное USB-решение. Современная электроника. 2005. № 4.
2. CY7C68013A/CY7C68015A EZ-USB FX2LP USB Microcontroller High-Speed USB Peripheral Controller. www.cypress.com.
3. EZ-USB FX2 Technical Reference Manual. www.cypress.com.
4. *Чекунов Д.* Стандартные требования USB. Современная электроника. 2004. № 2.