Практикум программиста USB-устройств Часть 4. Разработка программатора МК P89LPC9xx

Дмитрий Чекунов (г. Ижевск)

В очередной статье цикла, посвящённого проектированию USB-устройств, описана работа с «большими» точками. В качестве примера шаг за шагом рассмотрено создание готового устройства – программатора микроконтроллера P89LPC9xx.

Вступление

В предыдущих статьях мы подробно познакомились с внутренней структурой МК EZ-USB FX2LP, узнали о методах загрузки исполняемого кода и возможных вариантах организации USB-устройства. Мы подробно рассмотрели работу контрольной точки и работу «малых» точек модуля USB, все полученные знания успешно применили на практике и разработали универсальное аппаратно-программное ядро.

На данный момент мы знаем, как настроить драйвер CyUSB для нового устройства и проверить взаимодействие с устройством посредством программы CyConsole.

В составе МК у нас остались неизученными два весьма значимых ресурса – это «большие» точки и параллельный программируемый интерфейс GPIF. Данные ресурсы позволяют организовать высокоскоростной обмен данными и существенно расширяют возможности устройства.

Начнём с изучения «больших» точек, а для того, чтобы теория стала более понятной, применим полученные знания в новой разработке – программаторе микроконтроллеров семейства P89LPC9xx.

Организация «больших» точек МК

Регистры «больших» точек

В составе модуля USB имеются четыре «большие» точки – EP2, EP4, EP6, EP8. Режим работы каждой из них программируется индивидуально. В таблице 1 представлены основные регистры, используемые для работы с точками. Если название регистра начинается с EPx, то это говорит о его принадлежности к конкретной точке (x может принимать значение 2, 4, 6, 8).

Таблица 1. Ресурсы FX2LP для взаимодействия с «большими» точками

Ресурсы	Название
Регистр конфигурации	EPxCFG
Регистр управления и статуса	EPxCS
Буфер	EPxFIFOBUF
Регистры размера пакета – старший и младший	EPxBCH EPxBCL
Регистр запроса прерывания при запросе IN и пустом буфере	IBNIRQ
Регистр разрешения прерывания при запросе IN и пустом буфере	IBNIE
Регистр запроса прерывания при запросе PING и полном буфере	NAKIRQ
Регистр разрешения прерывания при запросе PING и полном буфере	NAKIE
Регистр запроса прерывания при изменении состояния буфера	EPIRQ
Регистр разрешения прерывания при изменении состояния буфера	EPIE
Регистр управления пакетами IN	INPKTEND
Регистр управления пакетами OUT	OUTPKTEND

Начнём знакомство с регистра конфигурации – EPxCFG (см. рис. 1). Работоспособность точки определяется состоянием бита VALID. Если точка x присутствует в описании интерфейса, то в соответствующем ей регистре бит VALID должен иметь значение 1 – точка включена.

Бит DIR задаёт направление передачи данных. Когда DIR сброшен в 0, точка имеет направление OUT, а когда установлен в 1 – направление IN.

«Большие» точки поддерживают три типа передачи данных: изохронный (01b), bulk (10b) и interrupt (11b). Выбор типа осуществляется битами TYPE1, TYPE0 – соответствующие им значения указаны в скобках.

Бит SIZE задаёт размер буфера точки. Когда бит сброшен в 0, размер буфера равен 512 байтам, когда установлен в 1 – 1024 байтам. Для точек EP4 и EP8 бит SIZE всегда сброшен в 0 (буфер этих точек не может быть больше 512 байт).

Глубина буферизации определяется битами BUF1 и BUF0. Точке может принадлежать минимум два буфера. Это значение фиксировано для точек EP4 и EP8. Для точек EP2 и EP6 размер буферизации программируется следующим образом:

- 10b – двукратный;
- 11b – трёхкратный;
- 00b – четырёхкратный.

EPxCS – регистр, при помощи которого осуществляется контроль состояния точки. Назначение битов регистра показано на рисунке 1.

Биты NPAK2, NPAK1, NPAK0 показывают количество заполненных буферов в домене FIFO конкретной точки.

Бит FULL является признаком полного FIFO, его целесообразно исполь-

зовать, когда для точки выбрано направление передачи IN.

Для контроля FIFO точки, работающей в противоположном направлении (OUT), служит бит EMPTY. Он служит признаком пустого FIFO.

Бит STALL используется для передачи одноименного подтверждения на любой запрос хоста.

Регистры EPxBCH, EPxBCL позволяют определить размер пакета, размещённого в буферах точки x. Старший байт размера пакета находится в регистре EPxBCH, младший – в EPxBCL.

Последовательность обращения к регистрам размера имеет значение для точек с направлением IN. Это вызвано тем, что после записи в регистр EPxBCL происходит передача буфера под управление модуля USB. По этой причине размер пакета всегда записывают в два приёма: сначала старший байт – в регистр EPxBCH, затем младший байт – в EPxBCL.

С системой прерываний модуля USB мы уже знакомились ранее [1], поэтому кратко напомню о назначении регистров, связанных с прерываниями.

IBNIRQ, IBNIE – регистры относятся только к точкам с направлением IN. Прерывание в регистре IBNIRQ формируется модулем USB после того, как он ввиду отсутствия данных для передачи отвечает подтверждением NAK на запрос хоста IN. Данное прерывание должно интерпретироваться микроконтроллером как сообщение, что хост уже запрашивает данные и следует поторопиться с их подготовкой.

Регистры NAKIRQ, NAKIE в большей степени относятся к точкам OUT и актуальны только в режиме high-speed. Модуль USB формирует прерывание, когда ответил подтверждением NAK на запрос хоста PING. Такая ситуация возможна, когда все буферы точки заполнены и хост проверяет возможность записи следующего пакета.

В регистрах NAKxxx имеется глобальное управление для всех прерываний IBN. Если его не включить, то прерывания IBN будут маскированы.

Регистры EPIRQ, EPIE связаны с точками обоих направлений. Прерывания для точки IN формируются, когда хост забрал данные из буфера. Для точки OUT прерывания формируются, когда получен пакет данных от хоста.

	D7	D6	D5	D4	D3	D2	D1	D0
EPxCFG	VALID	DIR	TYPE1	TYPE0	SIZE	0	BUF1	BUF0
EPxCS	0	NPAK2	NPAK1	NPAK0	FULL	EMPTY	0	STALL

Рис. 1. Состав регистров конфигурации и управления

Регистры INPKTEND и OUTPKTEND используются для уничтожения или дальнейшей передачи пакетов в буфер точки с соответствующим направлением.

Организация буферов FIFO

Общий объём FIFO, имеющегося в FX2LP, – 4 Кб. Распределение адресного пространства между точками показано на рисунке 2.

Буфер точки EP8 (EP8FIFOBUF) начинается с адреса FC00h и заканчивается адресом FDFFh. Размер буфера составляет 512 байт и не может быть больше, потому что бит SIZE в регистре EP8CFG всегда равен 0. Двойное буферизирование точки EP8 осуществляется за счёт зарезервированной области памяти FE00h – FFFFh.

При работе с буфером точки EP8 все обращения осуществляются к области памяти FC00h – FDFFh. После освобождения первого буфера происходит внутреннее переключение адресов, и второй буфер становится доступен в этом же адресном пространстве.

EP4FIFOBUF, буфер точки EP4, работает аналогичным образом.

Буфер точки EP6 организован несколько иначе. Адресное пространство, занимаемое им, расположено от F800h до FBFFh. Полный размер буфера составляет 1 Кб. Однако используемая и доступная область памяти в конечном счёте определяется битом SIZE. Когда он установлен в 1, то для работы доступно всё адресное пространство. В случае если бит сброшен в 0, доступной остаётся нижняя часть буфера (F800h – F9FFh) размером 512 байт.

Буферизация осуществляется за счёт неиспользуемой части буфера и буфера соседней точки. Так, при размере буфера 512 байт и двойной буферизации используется только область памяти, принадлежащая точке EP6. Если же буферизация – четырёхкратная, то к буферу EP6FIFOBUF подключаются области точки EP8 (FC00h – FDFFh, FE00h – FFFFh). В последнем случае точка EP8 будет неработоспособна.

Для буфера размером 1 Кб возможна только двойная буферизация, которая осуществляется опять же за счёт областей памяти точки EP8.

В любом из рассмотренных случаев обращение возможно только к области памяти F800h – F9FFh (при размере 512 байт) или к области F800h – FBFFh (при размере 1024 байта). После завершения работы с первым буфером происходит переключение адресов, и следующий (второй, третий, четвёртый) буфер становится доступен в этом же адресном пространстве. Программисту не обязательно знать, с каким буфером он работает, поскольку их переключение происходит аппаратно.

Организация буфера точки EP2 в общем похожа на EP6FIFOBUF. Различие заключается лишь в большем приоритете точки EP2 при распределении FIFO перед всеми остальными точками. Буфер EP2FIFOBUF может иметь четырёхкратную буферизацию при размере 1024 байта. Естественно, что в таком случае все остальные точки станут неработоспособными.

С возможными вариантами распределения буферов FIFO мы уже знакомились ранее [2, рис. 5].

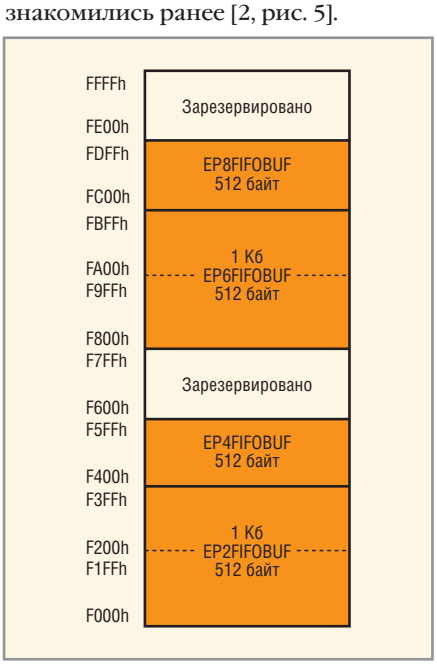

Рис. 2. Распределение адресного пространства FIFO

Рис. 3. Взаимодействие хоста и точки OUT

Методы обслуживания FIFO

В FX2LP имеется три метода обслуживания FIFO: программный, аппаратный и программно-аппаратный.

В первом случае FIFO обслуживает ядро 80C51 под управлением программы. При выборе такого метода программа должна контролировать состояние буферов точек, размеры принимаемых и передаваемых пакетов. Когда принят пакет (точка OUT), необходимо обработать данные и освободить буфер. При работе с точкой IN следует подготовить пакет данных максимально допустимого размера и передать его для отправки. Скорость передачи данных зависит от быстродействия ядра 80C51 и алгоритма программы.

При аппаратном обслуживании FIFO ядро 80C51 не принимает участия в передаче и обработке данных. Подготовка данных, разбиение их на пакеты необходимого размера и отправка происходят автоматически (направление IN). Приём данных, их обслуживание и освобождение буфера выполняются также автоматически (направление OUT). Для организации работы МК в таком режиме программа всего лишь инициализирует ресурсы FX2LP соответствующим образом. Скорость передачи данных, достижимая при использовании аппаратного метода, максимальна.

Третий метод обслуживания FIFO – программно-аппаратный. Он представляет собой синтез двух предыдущих. В этом случае ядро 80C51 занимается приёмом пакетов (точка OUT), возможно – анализом данных, и передаёт пакет для дальнейшего аппаратного обслуживания. Подготовка пакета данных для точки IN происходит также аппаратно, далее ядро 80C51 под управлением программы принимает решение об отправке пакета.

В данной статье мы рассмотрим программный метод обслуживания FIFO.

Программное управление FIFO

Точка с направлением OUT

Рассмотрим, как происходит программное обслуживание буфера «большой» точки с направлением OUT. Считаем, что точка уже сконфигурирована, заданы направление передачи, размер буфера и глубина буферизации.

В работе точки OUT можно выделить три фазы: инициализация, работа и деактивация (см. рис. 3).

Фаза инициализации. Данная фаза необходима для подготовки конечного устройства к приёму данных. Под конечным устройством следует понимать приёмник, данные к которому поступают от хоста транзитом через FX2LP. Здесь же выполняются прочие сопутствующие операции, такие как обнуление счётчика адреса, коммутация адреса внешнего устройства, включение питания внешнего устройства и т.п.

Сигналом к выполнению инициализации могут являться:

- USB-требование (стандартное или дополнительное);
- отсутствие внутреннего признака МК в фазе работы.

Нам уже известно, что при получении стандартных требований SET_CONFIGURATION и SET_INTERFACE, USB-устройство должно очистить буфер данных каждой работоспособной точки и установить маркер DATA0 для всех точек с типом передачи bulk и interrupt. При обслуживании требования CLEAR_FEATURE те же самые действия выполняются применительно к заданной точке – то есть происходит подготовка точки к работе со стороны USB. Это можно использовать для инициализации конечного устройства точки OUT.

Напомню, что при разработке аппаратно-программного ядра [1] мы включили вызов индивидуальной подпрограммы для каждой работоспособной точки. Организовано это с помощью таблиц векторов подпрограмм инициализации – tableInitFuncEpx.

В рассмотренном случае для инициализации устройства достаточно послать одно из названных стандартных требований.

Тем не менее, в некоторых ситуациях плюсы данного способа могут обратиться в минусы. Во-первых, конечное устройство будет инициализировано всякий раз при смене конфигурации и интерфейса (например, включается питание устройства, а обращения к нему нет). Во-вторых, инициализация выполняется для всех работоспособных точек одновременно. И, наконец, ресурсы, используемые точкой, могут быть задействованы ещё для каких-либо операций. В таком случае задачу инициализации лучше всего возложить на дополнительное требование. Это избавит от возможных конфликтов разделения ресурсов и исключит повторную инициализацию.

Если нет особых требований к быстродействию, то фазу инициализации можно совместить с фазой работы. В этом случае перед каждой обработкой данных необходимо проверить состояние признака инициализации и при его отсутствии – выполнить инициализацию. К плюсам такого способа относятся отсутствие каких-либо USB-требований и немедленный, последовательный переход от инициализации к обработке данных.

Наглядным примером использования рассмотренных способов могут быть:

- посредством стандартного требования – обнуление программного счётчика адреса;
- посредством дополнительного требования – передача команды для подготовки к пакетной записи;
- в составе фазы работы – включение питания устройства: вклю-

чить, если не включено, иначе пропустить.

После завершения фазы инициализации МК готов к приёму данных.

На рисунке 3 команда инициализации показана пакетом с условным названием INIT. При его получении МК освобождает буфер точки и может выполнить дополнительные действия. О готовности буфера к работе свидетельствует бит EMPTY (установлен в 1 – FIFO пуст).

Фаза работы. Контроль приёма данных осуществляется при помощи прерывания EPx (регистры EPIRQ, EPIE) или циклическим анализом бита EMPTY (регистр EPxCS). При получении пакета данных необходимо определить его размер (EPxBCH, EPxBCL) и приступить к обработке. Данные доступны в буфере EPxFIFOBUF.

Пока осуществляется обработка данных первого пакета, хост может успешно передать ещё до трёх пакетов (конечно, это определяется заданной глубиной буферизации). Приём каждого пакета влияет на состояние бит в регистре EPxCS. Так, после получения второго и третьего пакетов число, хранимое битами NPAKx, принимает соответствующее значение. Поступление четвёртого пакета приводит к установке бита FULL – FIFO (заполнено). Теперь на любую попытку хоста передать ещё пакет данных FX2LP автоматически ответит NAK (не готов к приёму данных). Или, если устройство работает в режиме high-speed, на запрос хоста PING МК автоматически ответит NAK и установит запрос в регистре NAKIRQ.

Пока будем считать, что данные полностью переданы за четыре пакета. В этом случае обмен с точки зрения хоста считается завершённым.

FX2LP, закончив обработку первого пакета, освобождает буфер. Любое освобождение буфера корректирует состояние битов статуса в регистре EPxCS (NPAKx, EMPTY, FULL). Из рисунка 3 видно, что после первой такой операции бит FULL принял значение 0 – это сообщает о появлении свободного места в буфере точки.

Для «больших» точек операция освобождения буфера осуществляется при помощи регистра OUTPKTEND. Необходимо записать в данный регистр результат операции ИЛИ значения 80h и номера точки (2, 4, 6, 8).

После освобождения буфера происходит аппаратное переключение адресов в FIFO, и в пространстве EPxFIFOBUF становится доступен второй пакет данных. Регистры EPxBCL, EPxBCH принимают значение, соответствующее размеру второго пакета.

МК снова выполняет обработку данных и освобождает буфер. О том, что данные в буфере исчерпаны, можно узнать по состоянию бита EMPTY. Его единичное состояние говорит о том, что буфер пуст. Однако это не определяет однозначно завершение фазы работы в целом – возможно, последует дальнейшая передача данных от хоста.

Фаза деактивации используется для исключения подобной неоднозначности. Сигналом для начала фазы деактивации могут служить команды хоста и внутренние или внешние события FX2LP.

По аналогии с фазой инициализации, командами хоста являются SET_CONFIGURATION, SET_INTERFACE и SET_FEATURE или дополнительное требование.

При обработке стандартных требований для каждой работоспособной точки, подлежащей отключению, вызывается подпрограмма из таблицы tableHaltFuncEpx. Эта возможность реализации фазы деактивации была также заложена при разработке аппаратно-программного ядра [1].

При использовании команд хоста (стандартных и дополнительных) возможно возникновение конфликтов, поскольку они асинхронны к действиям МК. Пример показан на рисунке 3. Время передачи пакета данных по USB обычно меньше времени, необходимого для его программной обработки микроконтроллером. Поэтому не исключена ситуация, когда хост, закончив передачу, инициирует фазу деактивации (первый пакет HALT). В результате такого сочетания данные будут потеряны. Поэтому между передачей OUT и командами, влияющими на целостность данных, следует предусмотреть некоторый интервал времени. Или, если используется дополнительное требование, устройство может отвечать STALL до завершения обработки данных. В последнем случае хост повторит команду позднее (второй пакет HALT).

Вариант, когда фаза деактивации начинается по собственным событиям МК, лишён ранее описанного недостатка. Такими событиями могут являться объём обработанных данных, ответ конечного устройства и т.п.

Данные, поступающие после фазы деактивации, следует уничтожать.

Простой пример. В ПЗУ объёмом 16 Кб необходимо записать данные. Хост посылает файл размером 17,5 Кб. В таком случае МК после обработки данных объёмом 16 Кб выполняет фазу деактивации и оставшиеся 1,5 Кб уничтожает (т.е. просто освобождает буфер).

Точка с направлением IN

Логика работы точек с направлением IN нам уже известна [3]. Поэтому сразу познакомимся с регистрами, используемыми для управления передачей через «большую» точку.

Так же, как и для точки OUT, приём данных из точки IN можно разделить на три фазы.

Фаза инициализации. Здесь необходимо выполнить очистку буфера и подготовить ресурсы к приёму данных. Очистка буфера осуществляется с помощью регистра INPKTEND. Для того чтобы уничтожить пакет, помещённый в буфер, необходимо записать результат операции ИЛИ значения 80h и номера точки (82h, 84h, 86h, 88h). Запись следует повторять до тех пор, пока не будет установлен бит EMPTY в регистре EPxCS. Установка данного бита является признаком того, что все буферы, принадлежащие точке, освобождены.

Схема использования стандартных и дополнительных требований для выполнения инициализации аналогична рассмотренной выше для точки OUT. Как правило, инициализация заканчивается разрешением прерывания IBN для соответствующей точки.

Фаза работы. О начале фазы данных можно узнать, осуществляя циклический контроль битов в регистре IBNIRQ или разрешив соответствующее прерывание в регистре IBNIE. При обнаружении запроса хоста следует немедленно начать подготовку данных. В процессе подготовки также возможно присутствие инициализации. Как правило, на этом этапе она больше предназначена для внешнего устройства. Осуществляется инициализация, когда сброшен определённый признак. После этого происходит его установка во избежание повторения инициализации.

Для того чтобы подготовить пакет данных, следует знать его максимально возможный размер. Заполняя буфер EPxFIFOBUF, необходимо учитывать этот размер. После завершения заполнения буфера следует записать сначала старший байт размера в регистр EPxBCH, а потом младший байт размера – в EPxBCL. Сразу после обращения к регистру EPxBCL пакет переходит под управление модуля USB. Бит EMPTY будет сброшен в 0, а значение, индицируемое битами NPAKx, станет равно единице (занят 1 буфер).

Существует два сценария дальнейшей работы.

По первому сценарию разрешаем прерывание EPx в регистре EPIE или программно контролируем состояние битов в EPIRQ. Как только хост заберёт пакет, будет установлен соответствующий флаг. Поскольку буфер освободился, продолжаем подготовку данных.

Во втором сценарии используются преимущества «больших» точек. Мы продолжаем подготовку и запись данных в буфер до тех пор, пока бит FULL сброшен в 0. Сразу после его установки (все буферы точки заполнены) начинаем действовать по первому сценарию.

Фаза деактивации. События, которые служат сигналом к завершению фазы работы, идентичны событиям точки OUT. Хост или устройство самостоятельно могут определить момент начала фазы деактивации.

Реакция устройства на стандартные или дополнительные запросы хоста повторяет действия точки OUT. Хост имеет возможность сообщить о начале фазы деактивации в любой момент, асинхронно к действиям устройства.

Несколько иначе выглядит фаза деактивации, выполняемая по самостоятельному решению устройства. В этом случае хост ещё продолжает посылать запросы IN, ожидая данные. Однако устройство, выполнившее фазу деактивации, возвращает пустые пакеты.

Наглядным является пример с ПЗУ. Устройство передаёт хосту 16 Кб данных, а на все последующие запросы возвращает пакеты нулевой длины. Для того чтобы повторить чтение ПЗУ, необходимо заново выполнить фазу инициализации.

Разработка нового USB-устройства

Итак, переходим к разработке программатора микроконтроллеров семейства P89LPC9xx фирмы Philips.

Новое устройство будем разрабатывать на основе аппаратно-программного ядра с использованием «больших» точек.

Знакомство с МК P89LPC9xx

МК P89LPC9xx построен на базе ядра 80C51. Благодаря большому набору функций, МК данного семейства давно пользуются заслуженным вниманием разработчиков.

Всё семейство P89LPC9xx насчитывает около трёх десятков МК. Упакованы они в корпуса различного размера, имеющие от 8 до 48 выводов. Это позволяет легко расширять возможности разрабатываемого устройства, заменяя микроконтроллер более производительным с минимальными доработками ПО.

Немалую роль в популяризации данного семейства сыграла поддержка перепрограммирования МК в системе. Это стало возможным благодаря использованию флэш-памяти и реализации системы самопрограммирования.

Способы программирования

Все МК поддерживают как минимум три способа программирования: параллельный, программный (IAP) и последовательный, через определённые линии порта (ICP).

МК, в составе которых имеется последовательный порт USART, поддерживают программирование и через этот порт (ISP).

Параллельный способ программирования подразумевает, что МК не запаян в плату и может быть установлен в программатор. Данный способ является самым надёжным, поскольку он не зависит от наличия встроенных программ загрузки. С другой стороны, параллельный способ малопривлекателен из-за того, что он неприменим к МК, установленным в устройство.

Программный способ программирования (IAP), а точнее, перепрограммирования, в противоположность параллельному, возможен только для МК, работающего в устройстве. Данный способ используется для обновления встраиваемого ПО и сохранения рабочих констант. Реализовать такой способ может программист при написании программы для разрабатываемого устройства.

Последовательные способы программирования (ICP, ISP) применимы к МК как запаянному в устройство, так и к незапаянному. В данном случае используются встроенные ресурсы МК (загрузочная программа для ISP), которые обеспечивают приём и запись кода во флэш-память.

В разрабатываемом нами программаторе реализуем последовательные способы программирования. Хочу обратить внимание на то, что МК FX2LP, использованный в схеме аппаратно-программного ядра [1], не имеет порта USART. Тем не менее, мы реализуем метод ISP, поскольку его использование возможно при замене FX2LP с 56 выводами на МК с большим числом выводов, где имеется USART.

Спецификация программирования методом ISP описана в руководстве пользователя для любого МК, имеющего в своём составе порт USART. Возьмём, к примеру, МК P89LPC922 и соответствующее ему описание [4].

Метод программирования ICP почему-то не так подробного описан в документации, хотя именно данный метод применим ко всем МК семейства. Тем не менее, существует отдельный документ, в котором описана спецификация программирования методом ICP [5].

Этими документами мы и будем руководствоваться при разработке программатора.

Условия программирования

В каждом методе программирования общим является использование двух линий питания (VCC, GND) и сигнала RESET. Способы начала программирования и линии передачи данных индивидуальны для каждого метода.

Метод ICP. Вход в режим программирования осуществляется при помощи сигнала RESET. После подачи питания необходимо выдать семь единичных импульсов RESET, как показано на рисунке 4. При дальнейшей установке сигнала в состояние «лог. 1» МК переходит в режим программирования.

Для обмена данными используются две линии порта P0:

- P0.4 (PDA) – двунаправленная линия данных;
- P0.5 (PCL) – линия тактового сигнала.

Тактовый сигнал формируется программатором. Данные фиксируются в приёмнике информации по положительному фронту PCL и не могут изменяться, пока PCL находится в состоянии «лог. 1».

Метод ISP. В этом методе обмен данными происходит по линиям TxD и RxD. Передаваемый пакет имеет формат:

- 8 бит данных;
- паритета нет;
- 1 стоп-бит.

Скорость может отличаться от рекомендуемых 9600 бод, поскольку в МК имеется программная автоподстройка скорости обмена.

Вход в режим программирования начинается с включения питания и выдачи трёх импульсов RESET. Далее сигнал остаётся в состоянии «лог. 1», а по линии TxD необходимо посылать символ «U» (55h), для того чтобы программируемый МК подстроил скорость обмена. Вход в режим программирования считается законченным, когда P89LPC9xx пошлёт в ответ символ «U».

Программирование

Когда МК находится в режиме программирования, он готов к приёму команд стирания, записи и чтения. Форматы этих команд для методов ICP и ISP сильно различаются. Более унифицированный формат команд разработан для метода ISP. Поэтому в данном разделе обзор начнем с него.

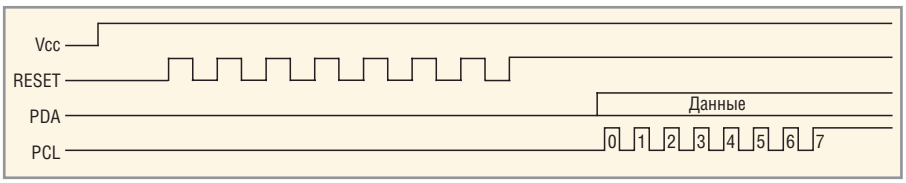

Рис. 4. Диаграмма входа в режим программирования ICP

Метод ISP. Формат всех команд в данном методе соответствует формату HEX-записи, созданному более 20 лет назад фирмой Intel. Запись представляет собой следующую строку:

:NNAAAARRDDDD.....DDCC<crlf>

Это обычная текстовая строка, начинающаяся с символа «:» и заканчивающаяся переводом строки. Внутри записи могут встречаться только ASCII-коды шестнадцатеричных чисел. Два символа NN характеризуют количество байт данных в записи. Следующие четыре символа AAAA задают адрес, с которого необходимо записать данные. Два символа RR являются признаком типа записи (обычные данные – 00, конец файла – 01 и т.д.). Далее следуют символы данных DD. Последние два символа CC являются контрольной суммой и позволяют проверять целостность записи.

Поскольку формат команды соответствует HEX-записи, её передача программируемому МК осуществляется ASCII-символами. При получении каждого символа P89LPC9xx возвращает его по линии RxD («эхо»). Когда HEX-запись передана полностью, МК проверяет контрольную сумму. В случае если обнаружено несоответствие, по линии RxD будет возвращён символ «X», иначе P89LPC9xx выполняет команду и возвращает символ «.».

Как видим, программирование методом ISP довольно просто выполнить: необходимо войти в режим программирования и передать по линии TxD файл формата HEX, символ за символом. Список команд протокола ISP показан в таблице 2.

Метод ICP. К сожалению, в данном методе нет такого унифицированного формата команд, как в ISP. В методе ICP необходимо оперировать содержимым регистров, обслуживающих внутреннюю флэш-память P89LPC9xx:

- FMCON – регистр управления;
- FMDATA – регистр данных;
- FMADRH, FMADRL – регистры старшего и младшего байта адреса.

При помощи команд, представленных в таблице 3, мы получаем доступ к названым регистрам и можем инициировать выполнение микроконтроллером любой операции из таблицы 4. Для контроля окончания выполнения операции служит байт статуса.

Передача данных в протоколе ICP осуществляется байтами. На линию PDA байт выдаётся младшим битом вперед. Изменения на линии данных

Таблица 2. Система команд метода программирования ISP

Назначение	Формат команды	Параметры	Возвращаемый результат
Запись кода во флэш-память	:NN AAAA 00 DD ... DD CC	NN – количество байт данных в записи; AAAA – адрес в области памяти; DD – данные; CC – контрольная сумма	«X» – несоответствие контрольной суммы; «.» – операция выполнена успешно*
Чтение версии ПО	:00 0000 01 FF	–	«X» – несоответствие контрольной суммы; CC CC CC CC «.» – операция выполнена успешно
Запись служебной информации	:02 0000 02 AA DD CC	AA – адрес в служебной области; DD – данные; CC – контрольная сумма	«X» – несоответствие контрольной суммы; «.» – операция выполнена успешно
Чтение служебной информации	:01 0000 03 AA CC	AA – адрес в служебной области; CC – контрольная сумма	«X» – несоответствие контрольной суммы; CC CC «.» – операция выполнена успешно
Стирание сектора или страницы	:03 0000 04 TT AAAA CC	TT – приёмник команды (0 – страница, 1 – сектор); AAAA – адрес страницы или сектора; CC – контрольная сумма	«X» – несоответствие контрольной суммы; «.» – операция выполнена успешно
Чтение контрольной суммы сектора	:01 0000 05 AA CC	AA – адрес сектора; CC – контрольная сумма	«X» – несоответствие контрольной суммы; CC CC CC CC CC CC CC CC «.» – операция выполнена успешно
Чтение глобальной контрольной суммы	:00 0000 06 FA	–	«X» – несоответствие контрольной суммы; CC CC CC CC CC CC CC CC «.» – операция выполнена успешно
Загрузка генератора USART	:02 0000 07 HH LL CC	HH – старший загружаемый байт; LL – младший загружаемый байт; CC – контрольная сумма	«X» – несоответствие контрольной суммы; «.» – операция выполнена успешно
Перезагрузка МК	:00 0000 08 F8	–	«X» – несоответствие контрольной суммы; «.» – операция выполнена успешно

Следом за символами «X» или «.» следуют два незначащих символа – 0Dh, 0Ah

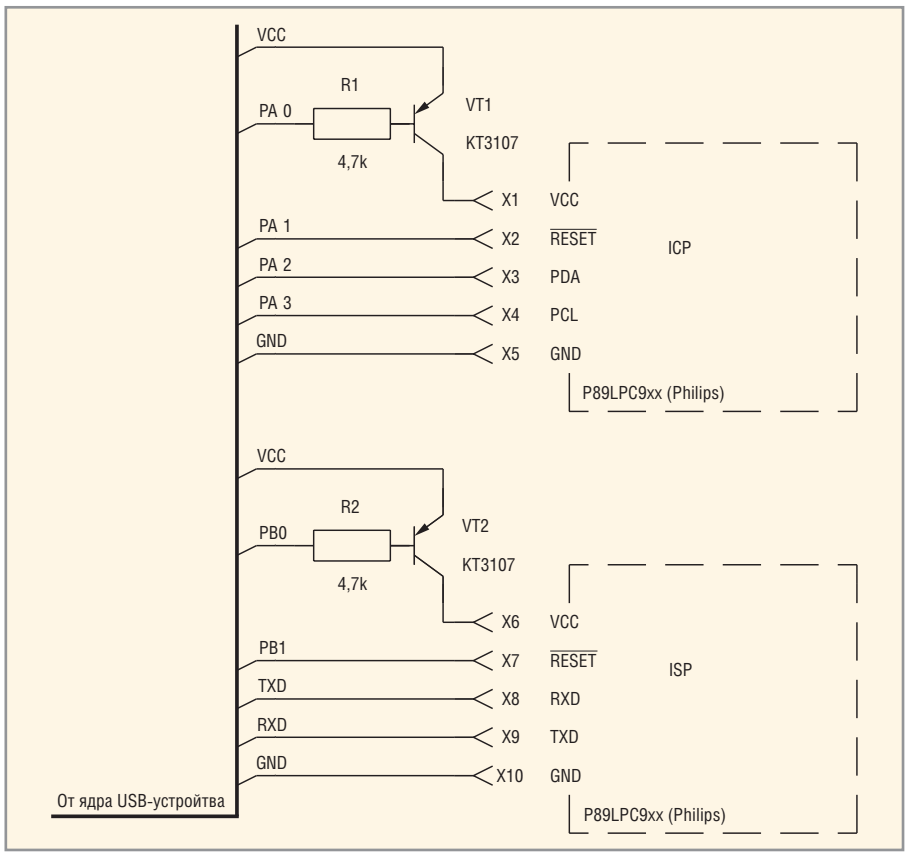

Рис. 5. Электрическая схема программатора МК P89LPC9xx

допускаются только тогда, когда PCL находится в состоянии «лог. 0».

Мы убедились в том, что форматы команд сильно отличаются. Отличие есть и в типе передаваемых данных. Так, в методе ICP байт передаётся в двоичном виде (09 – это девять), а в методе ISP каждый байт передаётся двумя ASCII-кодами (30h 39h – это девять: 30h → «0», 39h → «9»). Все рассмотренные особенности нам необходимо будет учесть при разработке системы команд программатора.

Таблица 3. Коды команд метода программирования ICP

Назначение	Символическое имя	Код
Нет операции	NOP	00h
Запись младшего байта адреса в FMADRL	WR_FMADRL	08h
Чтение младшего байта адреса из FMADRL	RD_FMADRL	09h
Запись старшего байта адреса в FMADRH	WR_FMADRH	0Ah
Чтение старшего байта адреса из FMADRH	RD_FMADRH	0Bh
Запись команды в FMCON	WR_FMCON	0Eh
Чтение статуса из FMCON	RD_FMCON	0Fh
Запись данных в FMDATA	WR_FMDATA	0Ch
Чтение данных из FMDATA	RD_FMDATA	0Dh
Запись данных в FMDATA и увеличение счётчика адреса	WR_FMDATA_I	04h
Чтение данных из FMDATA и увеличение счётчика адреса	RD_FMDATA_I	05h

Таблица 4. Коды внутренних команд P89LPC9xx для работы с флэш-памятью

Назначение	Символическое имя	Код
Загрузить регистр страницы	LOAD	00h
Запрограммировать страницу данными из регистра	PROG	48h
Глобальное стирание	ERS_G	72h
Стереть сектор	ERS_S	71h
Стереть страницу	ERS_P	70h
Доступ к служебной информации	CONF	6Ch
Вычислить глобальную контрольную сумму	CRC_G	1Ah
Вычислить контрольную сумму для сектора	CRC_S	19h
Очистить защиту конфигурации	CCP	67h

Разработка аппаратной части

Для того чтобы осуществить программирование методами ICP и ISP, потребуются минимальные доработки схемы аппаратно-программного ядра ([1], рис. 2).

Для ICP-метода задействуем четыре линии порта PA, которые будут управлять питанием, сигналами RESET, PDA и PCL. Напряжение питания +3,3 В будем коммутировать от внутреннего источника питания ядра к программируемому МК через транзистор VT1 (см. рис. 5).

Для ISP-метода (FX2LP должен быть в корпусе со 100 или более выводами) задействуем две линии порта PB и линии TxD, RxD. Напряжение питания будем коммутировать через транзистор VT2.

На этом разработка аппаратной части завершена. Все остальные действия реализуем программно.

Разработка программной части

Скопируем каталог mydevice в новый каталог и назовем его proglpc. В новом каталоге аналогично переименуем главный файл проекта mydevice.asm и изменим переменную NAME_PROJECT в файле asm.bat. Основа программы для нового устройства готова.

В первую очередь начнём редактировать файл переменных.

Файл var.asm. Поскольку у нас уже разработана схема, то мы зададим символические имена используемым линиям портов:

```
; Порт A
portIcp EQU     ioa ; порт ICP
программирования
initPortIcp     EQU     11111111b
; начальное состояние выводов -
все выключено
confIcp EQU     oea ; регистр
конфигурации порта ISP програм-
мирования
initConfIcp     EQU     00010111b
; начальное направление выводов,
PDA - ввод
initIcpRead     EQU
initConfIcp
initIcpWrite    EQU     00011111b ;
направление выводов, PDA - вывод
```

```
pinPda  BIT     portIcp.3 ; <1>
вход/выход данных
pinPcl  BIT     portIcp.2 ; [1]
выход тактового сигнала
pinResetIcp     BIT     portIcp.1
; [1] выход сигнала "Сброс уст-
ройства"
pinPowerIcp     BIT     portIcp.0
; [1] выход сигнала "Включить
питание"

; Порт B
portIsp EQU     iob ; порт ISP
программирования
initPortIsp     EQU     11111111b
; начальное состояние выводов -
все выключено
confIsp EQU     oeb ; регистр
конфигурации порта ISP програм-
мирования
initConfIsp     EQU     00000011b
; начальное направление выводов

pinResetIsp     BIT     portIsp.1
; [1] выход сигнала "Сброс уст-
ройства"
pinPowerIsp     BIT     portIsp.0
; [1] выход сигнала "Включить
питание"
```

Проектирование топологии устройства

Из спецификации по программированию [4, 5] мы знаем, что передача больших объёмов информации требуется только во время записи кода во флэш-память. Для прочих команд, в том числе и чтения, объёмы передаваемых данных малы.

В связи с этим наиболее правильным будет оформить все команды, кроме записи, в виде дополнительных требований, поскольку их необходимо подавать редко и выборочно, а вот запись осуществлять через точку OUT, потому что файл с программой для МК имеет большое количество HEX-записей. В этом случае нам необходимо выполнять выборку и обработку каждой записи в пакете. Задача несложная, так как признаки начала и конца записи известны.

Поскольку наше устройство поддерживает два метода программирования, создадим в интерфейсе две альтернативные установки. Первую – для работы с каналом ICP, вторую – с каналом ISP.

На рисунке 6 представлена топология разрабатываемого программатора. На «малые» точки в обоих случаях возложена задача обновления встраиваемого ПО программатора. Передача данных в программируемый МК осуществляется через точки EP2OUT. Использование одной точки в разных альтернативных установках упростит написание программы, поскольку подпрограммы инициализации, деактивации, приёма данных и выборки записи будут общими. Разделение на интерфейсы будет происходить уже на этапе непосредственной записи данных в P89LPC9xx.

Продолжим редактирование файлов проекта.

Файл ep0sd.asm. Установим программатору идентификатор PID, равный 1974h. Количество альтернативных установок зададим равным 2. При изменении количества альтернативных установок следует соответствующим образом скорректировать таблицу значений конфигурационных регистров tableCfg1If0.

Теперь редактируем описание топологии. В дескрипторе интерфейса исправим количество точек на 3 и добавим описание точки EP2OUT:

```
dscrCfg1If0Alt0Ep2:
    DB 7 ; длина дескриптора
    DB 5 ; тип свойства - точка
    DB 2 ; адрес точки
    DB 2 ; тип передачи bulk
sizeAlt0Ep2:
    DB 64,0 ; максимальная длина
пакета
    DB 0 ; интервал для EP_ISO
```

Сразу следом за этим описанием создадим ещё один дескриптор интерфейса с номером альтернативной установки 1. В нём будут описаны те же самые три точки.

В каждом дескрипторе интерфейса добавим строковое описание его назначения. При изменении строковых описаний следует скорректировать переменную DEAL_STRING и указатели на соответствующие строки в таблицах tableStringRu и tableStringUs.

Создадим файлы со строковыми описаниями и назовём их str4xxx.asm (для ICP) и str5xxx.asm (для ISP).

Поскольку в нашем проекте используются «большие» точки, то при подключении устройства к USB 2.0 желательно корректировать в дескрипторах этих точек размер пакета, обслуживаемого ими. Это позволит передавать данные пакетами большего размера, чем по шине USB 1.1.

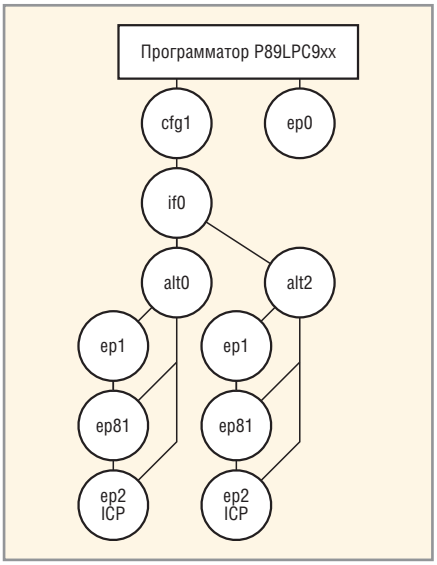

Рис. 6. Топология программатора

Для выполнения этого действия добавим подпрограмму grantHiSpeed, которая будет переписывать соответствующие места в дескрипторах точек:

```
grantHiSpeed:
  mov a,#LOW(SIZE_PAGE_EP2_HIGH)
  mov dptr,#sizeAlt0Ep2
  movx @dptr,a
  mov dptr,#sizeAlt1Ep2
  movx @dptr,a
  mov dptr,#pageEp2 + 1
  movx @dptr,a
  mov a,#HIGH(SIZE_PAGE_EP2_HIGH)
  mov dptr,#sizeAlt0Ep2+1
  movx @dptr,a
  mov dptr,#sizeAlt1Ep2+1
  movx @dptr,a
  mov dptr,#pageEp2
  movx @dptr,a
  ret
```

Файл intusb.asm. В обработчик прерывания isrHiSpeed добавим вызов подпрограммы grantHiSpeed.

Продолжение следует

Литература

1. *Чекунов Д.* Разработка аппаратно-программного ядра USB-устройства. Современная электроника. 2005. № 5, 6.
2. *Чекунов Д.* EZ-USB FX2LP – универсальное USB-решение. Современная электроника. 2005. № 4
3. *Чекунов Д.* Расширение функций ядра USB-устройства. Современная электроника. 2006. № 1.
4. User's Manual P89LPC920/921/922. www.semiconductors.philips.com.
5. P89LPC900 In-Circuit Programming (ICP) Specifications. www.8052.com.