Преобразователь интерфейсов USB–SPI на новом x51-микроконтроллере EFM8UB1
Часть 3

Алексей Кузьминов (Москва)

В первой и второй частях этой статьи (см. предыдущие номера журнала) были рассмотрены принципиальные схемы, варианты разводки плат и фотографии реальных устройств преобразователей интерфейсов USB–SPI на базе новейших высокоскоростных миниатюрных x51-совместимых микроконтроллеров EFM8UB1 компании Silicon Laboratories. Применение этих микроконтроллеров в подобных преобразователях кардинально снижает стоимость и габариты и существенно повышает скорость обмена. В третьей (заключительной) части статьи рассматривается ПО устройств (исходные тексты программ, их оттранслированные варианты в hex-формате – для МК и исполняемые файлы – для ПК), а также результаты их тестирования.

Программные средства, предназначенные для тестирования устройств преобразователей интерфейсов USB–SPI на базе микроконтроллера (МК) EFM8UB1, состоят из четырёх программ.

Первая программа, предназначенная для ПК, четырёхкратно посылает массив данных объёмом 1 КиБ (кибибайт, 1024 байта), что составляет в общей сложности 4 КиБ, по интерфейсу USB, к которому подключён преобразователь интерфейса USB–SPI. Затем по этому же интерфейсу программа принимает массив данных объёмом 4 КиБ и отображает его на экране монитора (для экономии места отображается 1/64 от массива 1 КиБ, т.е. строка длиной 64 символа). При этом отображается 1/64 переданного массива размером 1 КиБ и 1/64 принятого – для сравнения их между собой.

Перед передачей массива включается программный таймер, а после полного приёма всего массива таймер выключается, что позволяет оценить общее время обмена и вычислить его скорость. Эта программа написана автором на языке Кларион для Windows (Clarion V.6.0). Она подробно обсуждалась в статье [1] и здесь используется без изменений, поэтому не обсуждается. Однако сам текст программы, её файл-проект и все необходимые файлы для её трансляции на языке Кларион приведены в дополнительных материалах к данной статье (опубликованы вместе со статьёй на сайте журнала). Там же приведён и готовый к запуску исполняемый файл TestUSB.exe.

Вторая программа, предназначенная для МК EFM8UB1, четырёхкратно принимает массив данных размером в 1 КиБ и записывает этот массив (INARRAY[1024]) в оперативную память МК (xdata). Приём осуществляется пакетами по 64 байта. На такие пакеты массив разбивает ПК, а точнее – драйвер USB, который необходимо предварительно установить на ПК. Этот драйвер входит в бесплатно поставляемый компанией Silicon Laboratories пакет программ под общим названием USBXpress.

Далее МК EFM8UB1 либо посылает принятый массив в МК C8051F061 или EFM8LB12 (в зависимости от того, к какой макетной плате подключено устройство USB–SPI), либо, минуя интерфейс SPI (т.е. пропустив обмен по SPI), посылает этот массив обратно в ПК. Последняя опция используется для оценки скорости обмена исключительно по USB, что, в свою очередь, позволяет при обмене информацией по интерфейсам USB+SPI оценить скорость работы только по SPI, поскольку из времени работы по интерфейсам USB+SPI можно вычесть время работы только по интерфейсу USB (при этом объём информации известен и одинаков).

Логика работы этой программы та же, что и логика работы программы для устройств USB–SPI на базе МК C8051F321, приведённой в статье [1]. Однако в новой библиотеке (USBXpress.lib), поставляемой для EFM8UB1, во-первых, изменены функции обращения к интерфейсу USB, во-вторых, изменена сама логика работы функций (в частности, вместо физического используется виртуальное прерывание и др.). В-третьих, в МК EFM8UB1 изменены названия многих регистров (например, некоторых регистров интерфейса SPI) по сравнению с названиями, принятыми в C8051F321. Ко всему прочему, если для инициализации C8051F321 (портов ввода/вывода, матрицы соединений, генераторов, интерфейсов и т.п.) используется специальная программа Config2.exe, то для подобных функций для EFM8UB1 используется новое ПО под общим названием Simplicity Studio. Это же ПО используется и для трансляции программ с языка Си (C-51 Keil). В связи с этим, некоторые моменты программы для МК EFM8UB1 требуют обсуждения.

На сайте журнала в дополнительных материалах к данной статье приведены все необходимые файлы для трансляции этой программы (текст, различные *.h-файлы и т.п.). Кроме того, там же приведена уже оттранслированная программа в hex-формате, которая может быть сразу запрограммирована в МК EFM8UB1 с помощью USB-Debug-адаптера. Название этой программы – USBXpress_Echo (по аналогии с примером программы, приведённой компанией Silicon Laboratories в Simplicity Studio).

Третья программа, предназначенная для МК C8051F061/67, принимающая массив данных по интерфейсу SPI и отправляющая этот массив по этому же интерфейсу в МК EFM8UB1 (только в обратном порядке следования элементов массива), уже подробно обсуждалась в статье [1]. Тем не менее, на сайте журнала в дополнительных материалах к настоящей статье приведены все необходимые файлы для трансляции этой программы (текст программы на Си (C-51 Keil), bat-файл для её трансляции и т.д.). Там же приведена и сама оттранслированная программа в hex-формате, которую можно запрограммировать в МК C8051F067/061 с помощью USB-Debug-адаптера. Название этой программы – F067SPI4096.

Последняя, четвёртая программа, написанная автором для МК EFM8LB12 с той же логикой работы, что и для МК C8051F061/67, требует пояснения. Дело в том, что для конфигурирования портов ввода/вывода и инициализации иных устройств в МК EFM8LB12 используется уже упомянутое ПО Simplicity Studio. Кроме того, названия многих регистров МК EFM8LB12 (в частности, некоторых регистров интерфейса SPI) отличаются от названий соответствующих регистров C8051F061/67. На сайте журнала можно найти все необходимые файлы для трансляции этой программы и hex-файл myProject_9 для прошивки в МК EFM8LB12.

Далее мы обсудим программу для МК EFM8UB1, используемую в устройствах USB–SPI, после чего будут описаны некоторые нюансы программы для EFM8LB12, применяемого в описанной макетной плате. Следом мы приведём результаты тестирования устройств преобразователей интерфейсов USB–SPI на базе EFM8UB1, работающих с макетными платами МК C8051F061/67 и EFM8LB12.

Программы для микроконтроллеров EFM8UB1 и EFM8LB12

Вначале о программе для EFM8UB1. Для её написания, прежде всего, необходимо сконфигурировать МК или, другими словами, произвести инициализацию всех его устройств, т.е. привести в то состояние, в котором он должен находиться после сброса (Reset) и перед началом штатной работы. Здесь необходимо заметить, что для подобного конфигурирования МК C8051F321, который в корпусе QFN28 по выводам полностью совместим с EFM8UB1 в таком же корпусе (EFM8UB10F16G-C-QFN28), используется специальная программа Config2.exe.

Такое конфигурирование в автоматизированном режиме предусмотрено и в программе Simlicity Studio. Для выбора этой опции в основном меню Simlicity Studio нужно выбрать режим Configurator, задать соответствующий МК и все необходимые для его функционирования устройства (подробное описание этой процедуры приведено в руководстве по Simlicity Studio). Здесь же будут приведены только те моменты, которые касаются конфигурирования портов, связанных с сигналами SPI.

Когда интерфейс SPI с необходимыми опциями выбран и разрешён матрицей соединений, на экран выводится рисунок самого МК с выводами. Если интерфейс SPI выбран четырёхпроводный, то все четыре вывода, связанные с сигналами SPI (SPI0_SCK, SPI0_MISO, SPI0_MOSI и SPI0_NSS), по умолчанию занимают позицию, начиная с самых первых выводов в начале матрицы соединений (P0.0, P0.1 и т.п.). Но такое расположение выводов SPI очень неудобно как с точки зрения самих сигналов SPI (они расположены рядом с интерфейсом USB), так и с точки зрения разводки платы. Поэтому с помощью опции пропуска (skip) эти сигналы необходимо «перетащить» в правую часть МК (см. рис. 26).

Преимущества подобного расположения выводов с сигналами SPI следующие. Во-первых, сигналы USB расположены с левой стороны МК, и рядом с ними удобно расположить сам входной разъём USB. Во-вторых, сигналы интерфейса C2, по которому программируется МК, расположены снизу, и рядом с ними удобно расположить все компоненты (резисто-

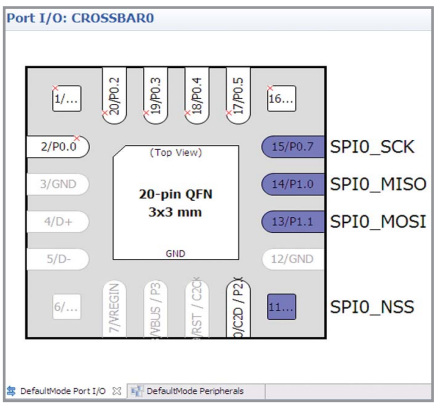

Рис. 26. Необходимое расположение сигналов интерфейса SPI в МК EFM8UB1 после нескольких команд skip

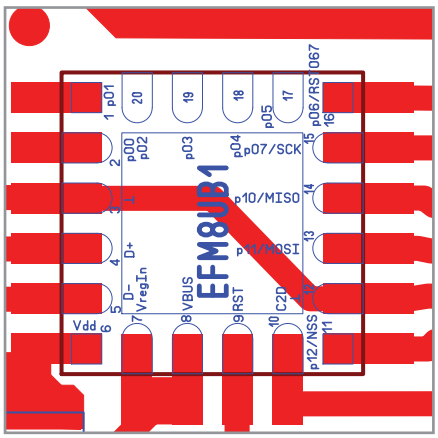

Рис. 27. Фрагмент разводки платы устройства в районе расположения МК EFM8UB1

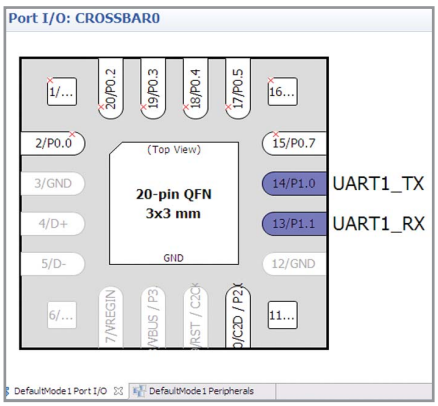

Рис. 28. Пример расположения сигналов интерфейсов в преобразователе USB-RS232

ры и конденсаторы), которые необходимы как для работы устройства в штатном режиме, так и для его функционирования в режиме программирования. В-третьих, рядом с выводами с сигналами SPI удобно расположить выходной разъём, на который и выводятся эти сигналы, а также сигналы интерфейса C2.

Такое расположение выводов и было выбрано окончательно, что отразилось на разводке платы устройства. На рисунке 27 приведён фрагмент разводки (вся разводка представлена на рисунке 13 во второй части статьи). Расположение МК здесь идентично его расположению на рисунке 26. Таким образом, задача проектирования устройства USB–SPI решалась «в обратном порядке»: начало – в программном обеспечении, затем разводка, а далее, в соответствии с ней, – схема устройства.

Если в программе Simplicity Studio щёлкнуть правой кнопкой мыши по картинке (см. рис. 26) и выбрать опцию Generation Source (сгенерировать код), то можно получить текст подпрограммы InitDevice.c (инициализация устройства), в котором, в частности, будут присутствовать как текст инициализации интерфейса SPI, так и текст инициализации матрицы соединений. Необходимо отметить, что текст инициализации устройства, а особенно текст подпрограммы инициализации матрицы соединений (если его распечатать), займёт несколько страниц, и в нём придётся достаточно долго разбираться. Поэтому можно поступить несколько иным образом.

Дело в том, что версия МК EFM8UB1 с 28 выводами (QFN28) по своей цоколёвке полностью совместима с выводами МК C8051F321 (тот же корпус QFN28). Для генерации кода подпрограммы инициализации этого МК используется известная программа Config2.exe. С её помощью можно получить аналогичный текст инициализации устройства (в том числе SPI и матрицы соединений) для EFM8UB1. Такой текст более компактен и занимает всего несколько строчек кода. Используя картинку (см. рис. 26) и определив, какие выводы МК используются в интерфейсе SPI, с помощью программы Config2.exe можно получить текст подпрограммы инициализации устройства (в том числе, интерфейса SPI и матрицы соединений). Переименовав сигналы SPI МК C8051F321 в соответствии с их названиями, принятыми в МК EFM8UB1, можно получить короткий код подпрограммы инициализации устройства. Здесь следует добавить, что сигнал, разрешающий матрицу соединений (XBARE), в C8051F321 расположен в регистре XBR1, а в EFM8UB1 – в регистре XBR2. Поэтому строку кода XBR1 = = 0x40 нужно переправить на XBR2 = 0x40.

В любом случае, тексты подпрограмм инициализации уже можно использовать для написания программы работы всего устройства.

И последнее, что, на взгляд автора, следует добавить: запретив интерфейс SPI и выбрав и разрешив интерфейс UART1 (а не UART0, «сдвинуть» который с его позиции, принятой по умолчанию, невозможно), можно легко получить устройство преобразования USB-RS232 (см. рис. 28).

Теперь поговорим об основной программе работы всего устройства.

Логика работы этой программы не отличается от таковой для C8051F321 в [1]. Однако, поскольку библиотека USBXpress.lib для EFM8UB1 отличается от соответствующей библиотеки для C8051F321 (USBX_F320_1.lib), далее приведено описание двух основных отличий новой библиотеки USBXpress.lib от USBX_F320_1.lib.

Первое отличие состоит в том, что в основной программе для C8051F321 используется классическая подпрограмма обработки физического прерывания (Interrupt Service Routine – ISR), имеющая конкретный номер, а точнее – приоритет. Он равен 16 для МК C8051F321 и 17 для МК C8051F340/380. В библиотеке для EFM8UB1 как такового физического прерывания нет. Вместо него используется виртуальное прерывание, для которого, естественно, никакого физического приоритета нет, поэтому классическая подпрограмма обработки прерывания не нужна. Вместо неё, во-первых, необходимо использовать функцию, разрешающую эти виртуальные прерывания:

API_Callback_Enable(myAPICallback),

которую следует применить после инициализации всех устройств.

Во-вторых, поскольку хоть и виртуальные, но всё-таки это прерывания, после их разрешения указанной функцией необходимо разрешить глобальные прерывания, установив бит разрешения глобальных прерываний в 1 и организовав бесконечный цикл обращения оператором while(1); к самому себе:

IE_EA = 1; // Enable global interrupts

while (1); // Spin forever.

Сама же подпрограмма обработки прерывания – это просто функция:

uint32_t Get_Callback_Source(void).

Эта функция возвращает 32-битное значение причины прерывания и сбрасывает флаг этого прерывания. Она должна вызываться в самом начале обращения к функциям USB и определяет события – источники прерываний. Вызов этой функции осуществляется так:

uint32_t INTVAL = Get_Callback_Source();,

где INTVAL – это и есть источник прерываний. Его значение, в частности,

может быть следующим: DEVICE_OPEN, DEVICE_CLOSE, RX_COMPLETE, TX_COMPLETE и др.

Прерывания DEVICE_OPEN и DEVICE_CLOSE возникают, соответственно, когда интерфейс USB открыт или закрыт для использования.

Прерывания RX_COMPLETE и TX_COMPLETE возникают, когда функции Block_Read() и Block_Write(), предназначенные, соответственно, для чтения и записи массива данных по интерфейсу USB, полностью отработали своё назначение.

Второе отличие библиотеки USBXpress.lib от аналогичной библиотеки USBX_F320_1.lib состоит в том, что две функции Block_Read() и Block_Write(), использующиеся и там, и там, существенно различаются.

Прототип для функции Block_Read():

uint8_t Block_Read(uint8_t *block, uint16_t numBytes, uint16_t *count_ptr).

Как видно из этого прототипа, функция имеет три параметра (в отличие от двух параметров в библиотеке USBX_F320_1.lib). Первый параметр – 8-битовый указатель (pointer) на адрес массива (*block), в который будет осуществляться запись прочитанных данных, поступивших из интерфейса USB. Звёздочка как раз и показывает, что это именно указатель. Второй параметр – 16-разрядное число (numBytes), которое указывает, сколько байт требуется считать. Третий параметр – 16-разрядный указатель (pointer) на переменную (*count_ptr), которая равна реальному значению поступающих по интерфейсу USB байт. Звёздочка показывает, что это именно указатель, а постфикс (_ptr) ещё раз подтверждает этот факт (ptr – сокращение от pointer (указатель)). Кроме того, сама функция принимает несколько 8-разрядных значений (uint8_t), которые называются флагами состояния (USBXpress Status Flags). Одно из них (USBX_STATUS_OK) при полном выполнении задачи функции – успешное её завершение, когда функция равна 0.

Пользоваться этой функцией очень просто. Если, например, определён массив размером в 64 байта INARRAY[64] и в этот массив требуется записать, например, 64 байта, поступивших из интерфейса USB, то функция

Block_Read(INARRAY, 64, &L);

выполнит это действие.

Как видно из этого примера, в качестве указателя используется название самого массива без скобок (это стандартный способ указателя на массив

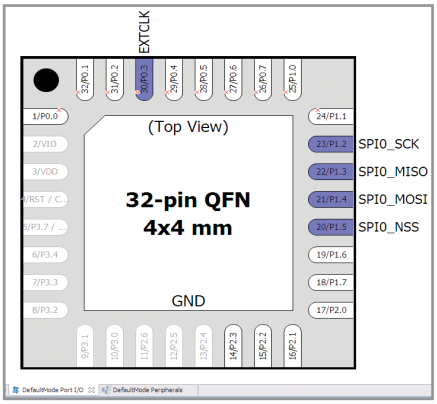

Рис. 29. Необходимое расположение сигналов интерфейса SPI в МК EFM8LB12 после нескольких команд skip

в Си). А вот перед переменной L установлен значок амперсанда (&), показывающий, что это указатель.

После полного и успешного выполнения функции Block_Read() произойдёт три события.

1. Возникнет прерывание RX_COMPLETE, по которому, в частности, можно отследить успешное выполнение функции.
2. Переменная L примет значение, равное 64.
3. Значение функции Block_Read() будет равно нулю (USBX_STATUS_OK).

Как видно из пунктов 2 и 3, успешное выполнение функции Block_Read() можно отследить вообще без использования виртуального прерывания: либо с помощью чтения переменной L, пока она не будет равна 64:

while(L != 64);,

либо с помощью чтения значения самой функции, пока она не будет равна нулю:

while(Block_Read(INARRAY, 64, &L)) != 0);.

Однако эти две возможности в программе автора не используются.

Прототип функции Block_Write() аналогичен прототипу функции Block_Read():

uint8_t Block_Write(uint8_t *block, uint16_t numBytes, uint16_t *count_ptr).

В программе эта функция используется аналогично:

Block_Write(INARRAY, 1024, &L);.

Всё сказанное про функцию Block_Read() справедливо также для функции Block_Write(), за тем исключением, что вместо чтения массива данных, поступающих из интерфейса USB, и записи его в память осуществляется его запись, т.е. массив данных, записанный в памяти, выводится по USB. Поэтому останавливаться на функции Block_Write() и подробно её описывать нет смысла.

В Simplicity Studio приведён пример программы, где как раз используются функции Block_Read() и Block_Write(). Это программа USBXpress_Echo, написанная для МК EFM8UB1. Она принимает массив данных, поступающих из интерфейса USB, по которому компьютер посылает этот массив, и принятый массив отсылает обратно в интерфейс USB, т.е. в компьютер. Вот откуда в названии программы взялось слово Echo (эхо). Если оттранслировать эту программу, запрограммировать её в МК EFM8UB1 и запустить программу автора для компьютера TestUSB.exe, то она будет работать, и на экран выведется точно такая же строка из символов, которая посылается этой программой в МК. Эту возможность можно использовать хотя бы для того, чтобы протестировать устройства в плане их функционирования по интерфейсу USB.

Для этой программы в Simplicity Studio уже подготовлен пакет опций, предназначенный для её трансляции с помощью самого языка C-51 и линкера, а поскольку трансляция производится правильно, т.к. программа работает, то этим пакетом опций можно воспользоваться. Дело в том, что в Simplicity Studio существует очень большое число различных опций, в том числе и предназначенных для трансляции программ. Чтобы разобраться в этом огромном объёме информации, требуется немало времени. Поэтому для трансляции собственной программы автор поступил следующим образом.

Файл с текстом оригинальной программы от Simplicity Studio (USBXpress_Echo_main.c) был переименован в файл USBXpress_Echo_main.origc, неиспользуемые автором файлы подпрограмм buffers.c и buffers.h переименованы, соответственно, в файлы buffers.origc и buffers.origh. Такие расширения файлов, естественно, трансляции не подлежат, но позволяют, при необходимости, восстановить оригиналы. Сама же программа автора была названа USBXpress_Echo_main.c, успешно оттранслирована, получен её hex-файл, который впоследствии был благополучно «зашит» в МК EFM8UB1. Тестирование программы показало, что если производить трансляцию подобным образом, то она будет идеально работать.

В дополнительных материалах на сайте журнала приведён текст этой программы, некоторые дополнительные файлы, требующиеся для её работы, и её загрузочный hex-файл.

Теперь поговорим о программе для МК EFM8LB12.

Прежде всего, требуется сконфигурировать устройство, т.е. получить программу, которая инициализирует все устройства микроконтроллера. С помощью конфигуратора, входящего в состав Simplicity Studio, были выбраны все устройства, требующиеся для работы программы (матрица соединений, внешний генератор, и т.п.). Некоторое затруднение вызвал выбор внешнего генератора (EXTOSC0) частотой 50 МГц, т.к. от этого выбора зависят ещё два параметра: CORE (выбор максимальной тактовой частоты при обращении к памяти) и Clock Control (выбор делителя для системной тактовой частоты). При неправильном выборе последовательности, в которой устанавливаются параметры этих трёх опций, возникает ошибка, а правильная последовательность может быть получена только опытным путём.

Далее с помощью команд skip выводы МК для интерфейса SPI были передвинуты в правую часть МК EFM8LB12 (см. рис. 29). На этом же рисунке показано единственное место, куда следует подключить внешний кварцевый генератор (EXTCLK, порт P0.3, четвёртый вывод).

После того как с помощью конфигуратора Simplicity Studio сгенерирована программа инициализации устройства (InitDevice.c), которая состоит из 9 подпрограмм и основной программы (main.c), включающей подпрограмму инициализации и состоящей из единственного оператора бесконечного цикла обращения к себе (while(1);), эту основную программу можно оттранслировать и получить её загрузочный hex-файл, который можно запрограммировать в МК.

Эту программу автор отредактировал, вставив в неё две стандартные подпрограммы ввода и вывода байта по интерфейсу SPI, задав массив размерностью 1024 байта (xdata unsigned char INARRAY[1024];) и вставив в основную программу ввод и вывод массива по интерфейсу SPI.

Основная программа, таким образом, очень проста и включает в себя, помимо прочего, всего два цикла. Первый цикл четырежды принимает массив данных объёмом 1024 байт по интерфейсу SPI (листинг 1).

Второй цикл четырежды выводит этот же массив по интерфейсу SPI, но только в обратном порядке (листинг 2).

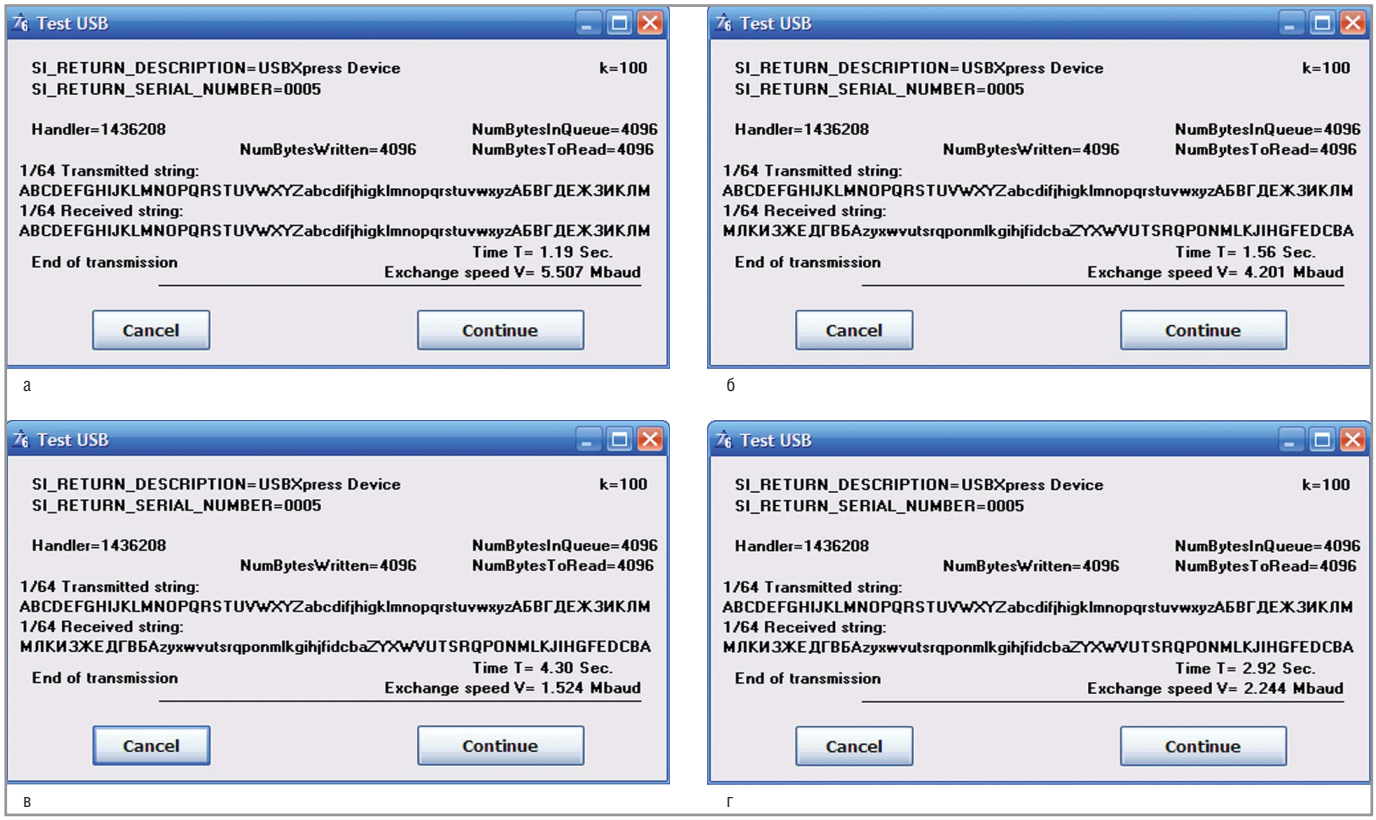

Рис. 30. Скриншоты работы программ: а – оригинальная программа Simplicity Studio (echo); б – программа автора с пропуском обмена по SPI (обмен только по USB всех четырёх устройств); в – обмен USB+SPI МК C8051F061 с устройством с четырёхпроводным изолированным SPI (см. рис. 21) и устройством с неизолированным SPI (см. рис. 19); г – обмен USB+SPI МК EFM8LB12 с устройством с неизолированным SPI (см. рис. 19) и устройством с изолированным USB (см. рис. 22)

Как видно из порядка итерации во втором цикле, она происходит с уменьшением переменной итерации i от 1024 до 1, тогда как в первом цикле итерация происходит с увеличением i от 0 до 1023.

В связи с этим массив INARRAY выводится в обратном порядке, т.е. первым выводится последний элемент массива, вторым – предпоследний и т.п., а последним – первый элемент массива.

Эта программа была оттранслирована, получен её hex-файл, который и был «зашит» в EFM8LB12.

Как ни странно, при первом же запуске программа заработала, поэтому сгенерированные тексты для девяти упомянутых подпрограмм инициализации устройств микроконтроллера автор посчитал правильными и разбираться в них не стал.

Листинг 1

```
for (k = 0; k < 4; k++) {
  for (i = 0; i < 1024; i++) {       // Приняли
      INARRAY[i] = inspi();          // входной массив
                        }            // INARRAY[i] 1024 байт
                    }                // 4 раза.
```

Листинг 2

```
for (k = 0; k < 4; k++)  {           // Вывели
  for (i = 1024; i != 0; i--)   {    // принятый массив
      outspi(INARRAY[i - 1]);        // INARRAY[i]
            }                        //в обратном порядке.
      }                              // 4 раза.
```

Результаты тестирования устройств

Тестирование устройств проводилось в следующей последовательности.

Вначале тестировались все четыре устройства на базе МК EFM8UB1, работающие исключительно по интерфейсу USB, для чего в программах автора обмен по интерфейсу SPI был пропущен. Кроме того, тестировалась оригинальная программа USBXpress_Echo, в которой обмен осуществляется только по USB.

Результаты работы программ показаны на рисунке 30 при различных режимах работы МК EFM8UB1, C8051F061 и EFM8LB12. Рисунок 30а иллюстрирует работу программы USBExpress_Echo, а рисунок 30б – работу программы автора, когда обмен по SPI пропущен. Для этого последовательно в МК EFM8UB1 всех четырёх устройств вначале была загружена программа USBExpress_Echo, а затем программы автора, работающие только с интерфейсом USB (т.е. с пропуском обмена по SPI).

Скорость обмена по USB, как это следует из программы на ПК, составляет:

V=((8*2*4096*100 [бит])/T[c])/1000000, (1)

где T – время в секундах.

Эта формула получена из следующих соображений. Пакет данных объёмом 4096 байт (отсюда коэффициент 4096) передаётся из ПК в МК и обратно (отсюда коэффициент 2) 100 раз (отсюда коэффициент 100). 1 байт содержит

8 бит (отсюда коэффициент 8), и один Мбод содержит 1000000 бод (отсюда коэффициент 1000000).

Например, из рисунка 30б следует, что время обмена составляет 1,56 с, поэтому скорость обмена в соответствии с (1) будет:

V=((8*2*4096,0*100 [бит])/1,56 [с])/ 1000000,0 = 4,201 Мбод,

что и отражает этот снимок экрана.

Из рисунка 30а видно, что обмен по USB в программе USBXpress_echo занимает время 1,19 с, а скорость обмена составляет 5,507 Мбод, т.е. немного выше, чем в программе автора (4,201 Мбод на рисунке 30б). Это, вероятно, можно объяснить тем, что некоторую часть времени обмен по USB идёт в дуплексном режиме, т.е. осуществляется одновременный приём и передача информации. Этому способствует программа USBXpress_Echo, а конкретно – подпрограмма Buffers, входящая в состав USBXpress_Echo.

В программе автора обмен по USB осуществляется только в полудуплексном режиме, т.е. вначале весь массив 4 КиБ передаётся в МК, принимается им, а затем этот же массив передаётся обратно в ПК (т.е. в один и тот же момент времени осуществляется либо приём, либо передача). Такая последовательность обмена специально предназначена для того, чтобы можно было осуществить ещё и обмен по интерфейсу SPI. Соображения, по которым написана формула (1), отражают именно такую последовательность обмена. В связи с этим скорость обмена на рисунке 30а вычислена некорректно, а потому не совсем соответствует действительности.

Далее устройства тестировались в режиме работы по интерфейсам USB и SPI.

Для этого в устройства, работающие в четырёхпроводном режиме (с неизолированным и изолированным интерфейсом SPI), загружалась программа, в которой интерфейс SPI был инициализирован для работы в четырёхпроводном режиме. А в устройство с изолированным интерфейсом SPI, работающее в трёхпроводном режиме, загружалась программа, в которой интерфейс SPI был инициализирован для работы в трёхпроводном режиме.

Далее к макетной плате с МК C8051F061, частота кварцевого резонатора которого составляла 20,97152 МГц, были подключены три устройства с неизолированным и изолированным интерфейсом SPI. Устройство с изолированным интерфейсом USB к макетной плате с относительно медленным микроконтроллером C8051F061 не подключалось, т.к. это высокоскоростное устройство подключать нет смысла, поскольку его применение является избыточным.

Вначале была установлена частота сигнала SCK (в SPI) F_{sck} = 3,33 МГц (SPI0CKR=2), которая являлась максимальной для устройств преобразователей USB–SPI на базе МК C8051F321 [1]. При такой частоте F_{sck} все три устройства работали и показали идеальный результат.

Далее частота F_{sck} была увеличена до 5,24 МГц (SPI0CKR=1). При этой частоте устройство изолированного интерфейса SPI с трёхпроводным режимом отказалось работать, а остальные два устройства показали идеальную работу.

И, наконец, частота F_{sck} была ещё увеличена до 10,5 МГц (SPI0CKR=0). При этой частоте два устройства с четырёхпроводным режимом SPI (с изолированным и неизолированным SPI) также показали идеальный результат, который отражён на рисунке 30в. Как видно из этого скриншота, время обмена по USB+SPI составило 4,3 с. Если из этого времени вычесть время работы только по интерфейсу USB, равное 1,56 с (см. рис. 30б), то получим время работы по интерфейсу SPI – это 2,74 с. Подставив его в формулу (1), можно получить скорость работы интерфейса SPI, равную 2,39 Мбод.

Отсюда вывод: применение МК EFM8UB1 вместо C8051F321 в преобразователе USB–SPI увеличивает скорость обмена более чем в три раза (10,5 МГц / 3,33 МГц = 3,15) при работе с МК C8051F061.

Здесь следует сделать некоторое пояснение относительно оптимального выбора частоты кварцевого резонатора, который определяет скорость обмена по SPI. Напомним, что максимальная частота работы МК C8051F061/67 составляет 25 МГц.

Пусть имеется два кварцевых резонатора с частотами, равными, например, 20,97152 и 23,59296 МГц. Эти частоты выбраны не случайно. Число 20971520 = 65536 × 320, а 23592960 = 65536 × × 360. Это позволяет с помощью таких частот получить очень точные интервалы времени, равные 0,1 с. Если, например, системную тактовую частоту завести на таймер/счётчик 0 (C/T0), выход которого подключить к счётчику PCA (PCA counter/timer, PCA – Programmable Counter Array – программируемый массив счётчиков), то при переполнении C/T0 будет давать частоту, равную системной (SYSCLK), делённой на 65536. Если SYSCLK = 20,97152 МГц, то на счётчик PCA будет подаваться частота 320 Гц. При SYSCLK = 23,59296 МГц на счётчик будет подаваться частота 360 Гц. Разделив, соответственно, 320 на 32 и 360 на 36, счётчик будет переполняться с частотой 10 Гц, или, другими словами, выдавать точные интервалы времени по 0,1 с. Более подробно о том, как с ними работать, можно ознакомиться в статье [2].

Здесь же эти два резонатора приведены совсем для другой цели. А именно, чтобы ответить на вопрос: какой из них выбрать? На первый взгляд кажется, что чем больше частота резонатора и чем она ближе к максимальной (25 МГц), тем лучше, поэтому лучший выбор – резонатор с частотой 23,59296 МГц. Однако это не так. И вот почему. В микроконтроллере частота следования импульсов синхронизации SCK (F_{sck}) в интерфейсе SPI определяется значением регистра SPI0CKR по следующей формуле:

$$F_{sck} = SYSCLK/(2\times(SPI0CKR+1)),$$

где SYSCLK – значение системной тактовой частоты работы МК. Если, например, SYSCLK = 20,97152 МГц, а SPI0CKR = 1, то F_{sck} = 20,97152 МГц / (2 × (1 + 1)) = = 5,24288 МГц ≈ 5,24 МГц. Если SPI0CKR = = 0, то F_{sck} = 10,48576 МГц ≈ 10,5 МГц. Если SYSCLK = 23,59296 МГц, то при SPI0CKR = = 1 F_{sck} = 5,89824 МГц ≈ 5,9 МГц, а при SPI0CKR = 0 F_{sck} = 11,79648 МГц ≈ 11,8 МГц.

У автора были в наличии оба резонатора. Их использование показало следующее.

При F_{sck} = 11,8 МГц ни одно из устройств не работало, а при F_{sck} = =10,5 МГц, как уже было упомянуто ранее, работали устройства с четырёхпроводным интерфейсом SPI (изолированным и неизолированным). Поэтому, выбрав резонатор с частотой 23,59296 МГц, для работы устройств придётся установить SPI0CKR = 1, который даёт F_{sck} = = 5,9 МГц, что почти в два раза меньше, чем 10,5 МГц. Отсюда вывод: в данном случае подходит только резонатор частотой 20,97152 МГц. И не всегда чем выше тактовая частота резонатора, тем лучше.

Далее все четыре устройства были последовательно подключены к макетной плате с МК EFM8LB12 (с частотой внешнего кварцевого генератора 50 МГц).

Самую низкую скорость обмена по интерфейсу SPI показало устройство изолированного интерфейса SPI, работающее в трёхпроводном режиме. Оно работоспособно только при F_{sck} = 4,16 МГц (SPI0CKR = 5). Время обмена составило 4,4 с, а скорость обмена – 1,524 Мбод.

Следующее по быстродействию – устройство с изолированным интерфейсом SPI, работающее в четырёхпроводном режиме. Для него частота F_{sck} = 12,5 МГц (SPICKR = 1), время обмена равно 3,14 с и скорость – 2,087 Мбод. Здесь необходимо заметить, что такая частота F_{sck} обмена по интерфейсу SPI принята для EFM8LB12 по умолчанию (SPICKR = 1). Она в четыре раза меньше системной частоты (SYSCLK), составляющей в данном случае 50 МГц.

И, наконец, самое скоростное – устройство с неизолированным интерфейсом SPI и устройство с изолированным USB, которые работают в четырёхпроводном режиме SPI. Для них частота F_{sck} = 25 МГц (SPICKR = 0), время обмена составляет 2,92 с, а скорость – 2,244 Мбод. Эти данные взяты из рисунка 30г. Зная время обмена устройств с ПК только по интерфейсу USB, равное 1,56 с (см рис. 30б), можно определить время обмена только по интерфейсу SPI – 1,36 с. Подставив его в формулу (1), получим скорость обмена только по интерфейсу SPI, равную 4,819 Мбод, что выше скорости обмена только по USB (4,201 Мбод на рисунке 30б). Это означает, что интерфейс SPI по скорости превосходит интерфейс USB (конечно, при соответствующем выборе МК). Здесь следует добавить, что при F_{sck} = 25 МГц идеальная скорость обмена по SPI составляет 25 Мбод, что в два раза превышает скорость обмена по USB в режиме Full Speed (12 Мбод).

Автор решил проверить, а может ли устройство с неизолированным SPI работать ещё быстрее? Дело в том, что МК EFM8LB12 оборудован внутренним генератором (вторым) с тактовой частотой 75 МГц. Для этого EFM8LB12 был переконфигурирован, и в качестве системной была установлена тактовая частота второго внутреннего генератора, которая по умолчанию установилась равной 72 МГц. При этом частота тактирования интерфейса SPI (F_{sck}) установилась равной 36 МГц (при SPICKR = 0). Однако устройство с неизолированным интерфейсом при такой частоте отказалось работать (на экран вывелась строка, состоящая из 64 чёрных прямоугольников, имеющих, как известно, ASCII-код 255). Автор из-за этого ничуть не огорчился, т.к. этим экспериментом был поставлен предел тактовой частоты, равный 25 МГц, при обмене этого устройства по интерфейсу SPI.

Заключение

Применение микроконтроллеров EFM8UB1 в преобразователях интерфейсов USB–SPI кардинально снижает их стоимость и габариты, существенно увеличивает скорость обмена, что делает конструкции подобных преобразователей почти безальтернативным вариантом. Использование таких преобразователей в системах сбора и обработки информации значительно повысит скоростные и надёжностные характеристики обмена данными компьютера с микроконтроллером по интерфейсу USB.

Литература

1. *Кузьминов А.* Преобразователь интерфейсов USB–SPI с гальванической развязкой. Современная электроника. 2012. № 1, 2.
2. *Кузьминов А.* Как заставить встроенный в микроконтроллер АЦП поразрядного уравновешивания работать с разрешением дельта-сигма-АЦП. Современная электроника. 2012. № 3.