Преобразователь интерфейсов USB-SPI с гальванической развязкой

(часть 2)

Алексей Кузьминов (Москва)

Во второй части статьи описаны аппаратные и программные средства сопряжения микроконтроллера C8051F321 с компьютером по интерфейсу USB и с микроконтроллером C8051F067 по гальванически изолированному интерфейсу SPI с помощью быстродействующих цифровых изоляторов SI8663BC-B-IS, ADUM7441C и ISO7220C. Программные средства представлены в виде текстов программ для компьютера на языке Clarion v.6.0 и для микроконтроллеров на языке Си (Keil C51 v.6.14, v.9.01) с использованием библиотеки USBXpress фирмы Silicon Labs. Описан программный способ синхронизации микроконтроллеров при обмене по интерфейсу SPI, позволяющий повысить скорость обмена в трёхпроводном режиме.

Микросхемы цифровых изоляторов ADUM3160/ADUM4160 фирмы Analog Devices предназначены для гальванической изоляции самого интерфейса USB на пробивное напряжение 2,5 кВ (ADUM3160) и 5 кВ (ADUM4160). Согласно спецификации USB 1.2, они могут работать как на низкой скорости (low speed) 1,5 Мбод, так и на полной (full speed) – 12 Мбод. По сравнению со всеми цифровыми изоляторами, описанными ранее в статье, цифровые изоляторы ADUM3160/ADUM4160 имеют достаточно сложную внутреннюю структуру, которая учитывает двухполярность и двунаправленность линий данных D+ и D– интерфейса USB. Изоляция в этих микросхемах, как и во всех других цифровых изоляторах фирмы Analog Devices, осуществляется по технологии iCoupler посредством высокочастотного электромагнитного поля (как в ВЧ-трансформаторах). Более подробно с этими микросхемами можно ознакомиться в [7].

Сложное внутреннее устройство микросхем ADUM3160/ADUM4160 упрощает реализацию гальванической развязки интерфейса USB. Всё, что требуется для работы схемы, показанной на рисунке 17, – подать напряжение +5 В на изолированную сторону. Это можно сделать тремя способами:

- если устройство оборудовано собственным источником питания с напряжением +5 В, то напряжение следует подать на контакт 1 разъёма USBA X2 (см. рис. 17);
- если устройство не имеет собственного источника питания, то необходимо использовать внешний источник питания +5 В, подключив его к соответствующим контактам разъёма X3 (см. рис. 17);
- если использование внешнего источника питания невозможно, можно применить изолированный DC/DC-преобразователь 5 В → 5 В. В качестве таких преобразователей автор рекомендует использовать либо промышленный прибор C10205BA, либо микросхему DCR010505P, либо изготовить DC/DC-преобразователь своими силами на основе микросхемы MAX253 [8].

Когда интерфейс USB гальванически изолирован, можно преобразовать его в интерфейс SPI без гальванической изоляции. Схема такого USB-SPI-преобразователя на базе микроконтроллера C8051F321 приведена на рисунке 18.

Выбор портов микроконтроллера в матрице соединений для сигналов интерфейса SPI (см. рис. 18, 19а) не случаен. Если матрицу модифицировать (см. рис. 19б) только с помощью программных средств, то такое устройство можно легко превратить в преобразователь USB-RS-232 с дополнительными выходными push-pull-сигналами (P0.1, P0.2 и P0.3); фрагмент такой схемы показан на рисунке 20.

На первый взгляд кажется, что эффективность такого преобразователя невысока, поскольку скорости обмена информацией в интерфейсах USB и RS232 различаются на два порядка: у USB 1.2 – 12 Мбод, а у RS-232 – 0,12 Мбод. Однако это не совсем так. В интерфейсе USB, в котором обмен информацией осуществляется пакетами, реальная скорость обмена существенно ниже 12 Мбод и зависит от размера пакета: чем меньше размер пакета, тем ниже скорость. При размере пакета в 4 Кб реальная скорость обмена по интерфейсу USB не превышает 5 Мбод. При размере пакета в 1 Кб скорость снижается до 4 Мбод, а при размере пакета в 64 байта скорость составляет 1 Мбод, что всего на порядок больше скорости обмена по интерфейсу RS-232. Если передавать несколько байтов, то скорость обмена по USB сравнима со скоростью обмена по RS-232.

Кроме того, встречаются ситуации, когда требуется обмен информацией исключительно по интерфейсу RS-232. Например, подавляющее большинство микроконтроллеров программируются «в системе» (ISP) только по интерфейсу RS-232, поскольку скорость такого программирования существенно

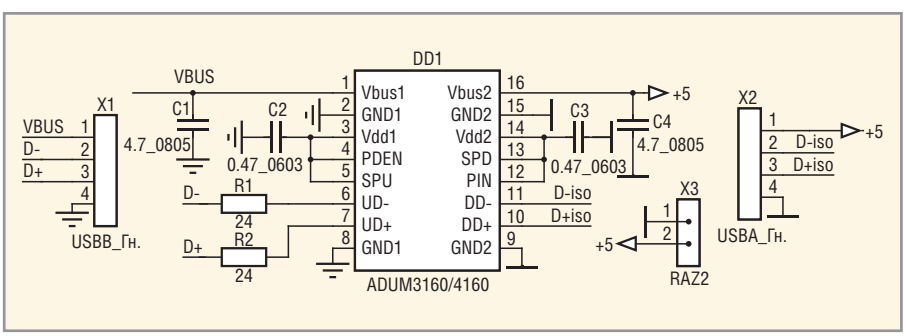

Рис. 17. Схема гальванически изолированного интерфейса USB на базе микросхем ADUM3160/4160, работающего в режиме полной скорости интерфейса USB 1.2

меньше максимальной скорости обмена по RS-232 (115 200 бод).

Скорость обмена информацией в беспроводных устройствах, таких, например, как беспроводные модемы, работающие в «гражданском» диапазоне частот 433 МГц, составляет максимум 9600 бод. Существует множество измерительных приборов, обмен информацией которых с компьютером возможен исключительно по интерфейсу RS-232 (например, различные счётчики воды, газа и тепла, газовые корректоры и т.п.).

Иногда требуется передать информацию на достаточно большое расстояние (до 1 км) по проводной линии связи. В этом случае, преобразовав интерфейс RS-232 в интерфейс RS-422 и обратно, можно решить поставленную задачу [3].

В связи с тем что подавляющее большинство современных компьютеров не оборудовано интерфейсом RS-232, существуют специальные устройства в виде плат и переходных кабелей для портов USB. Поэтому многие фирмы поставляют готовые микросхемы и модули преобразователей USB-RS-232.

В качестве примеров на рисунке 21 представлены схемы подключения преобразователя интерфейсов USB-RS-232 (см. рис. 20) к микроконтроллерам, работающим как в штатном режиме обмена по интерфейсу RS-232, так и в режиме программирования в системе (ISP) [3].

Разводка печатных плат (см. рис. 22) гальванически изолированногоинтерфейса USB на базе микросхем ADUM 3160/4160 (см. рис. 17) и неизолированного преобразователя интерфейса USB-SPI/RS-232 (схемы на рис. 18 и 20) сделана автором с помощью программы Sprint LayOut 5. Устройство изолированного интерфейса USB получилось достаточно компактным (30 × 18 мм), а устройство неизолированного преобразователя интерфейсов USB-SPI/RS-232 – самым миниатюрным из всех устройств, представленных в статье (10 × 25 мм). Фотографии обоих устройств показаны на рисунке 23.

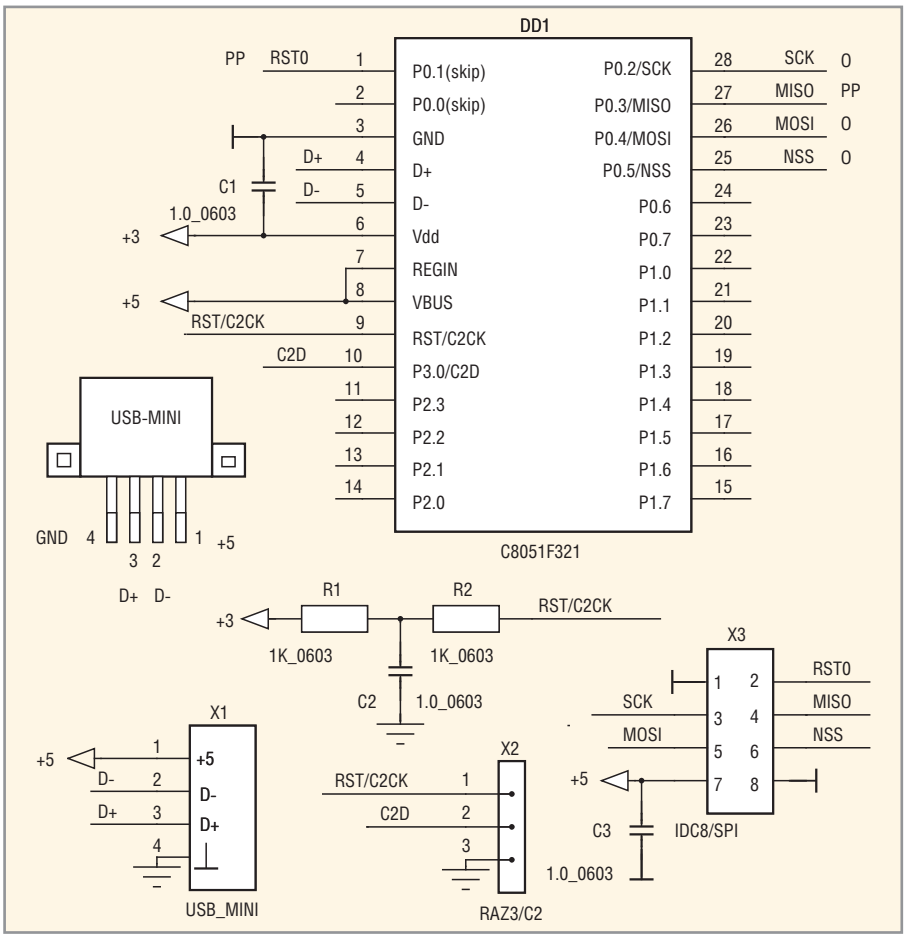

Рис. 18. Схема неизолированного преобразователя

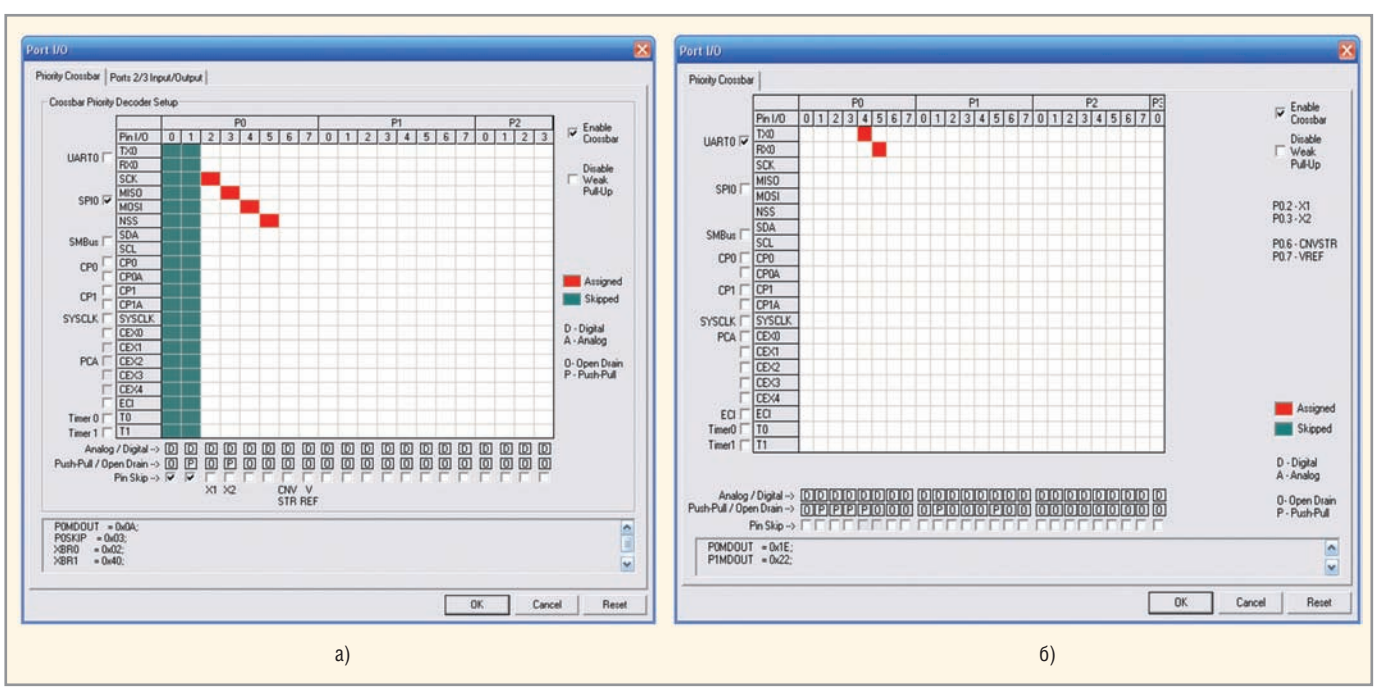

Рис. 19. Конфигурация портов микроконтроллера C8051F321 для преобразователя USB-SPI (а) и USB-RS-232 (б)

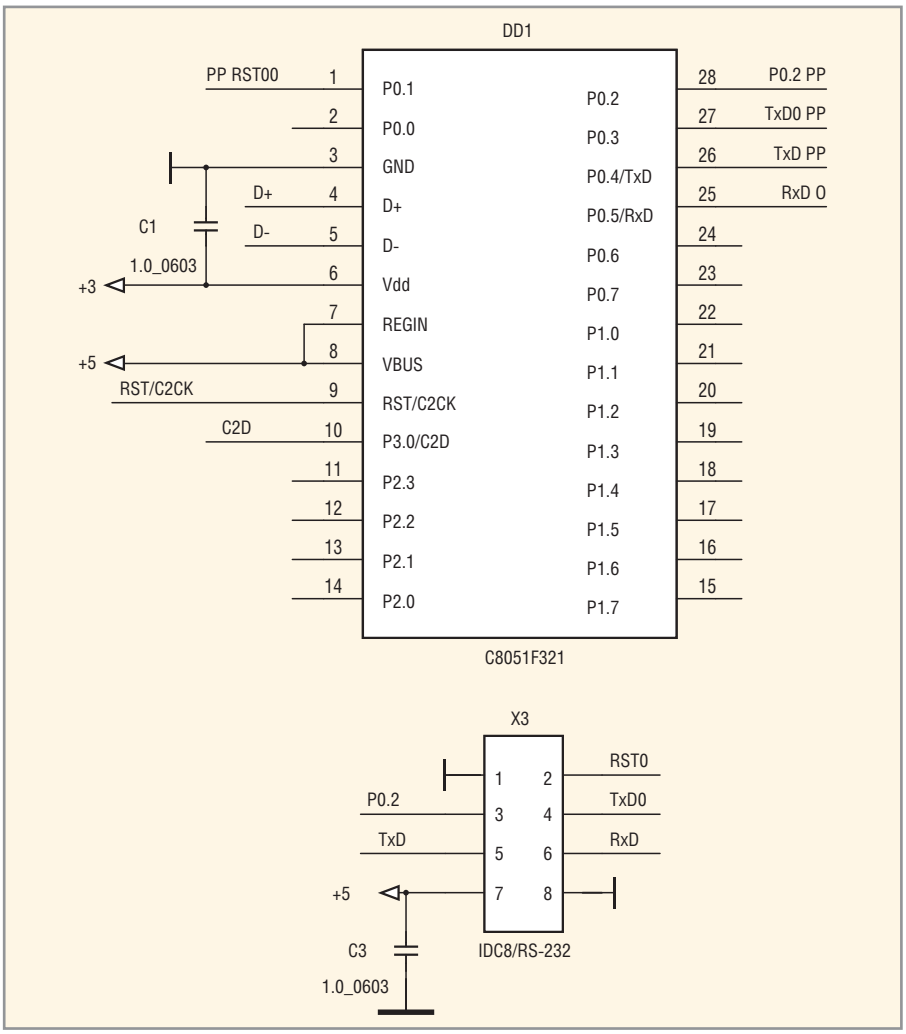

Рис. 20. Фрагмент схемы преобразователя USB-SPI на базе микроконтроллера C8051F321

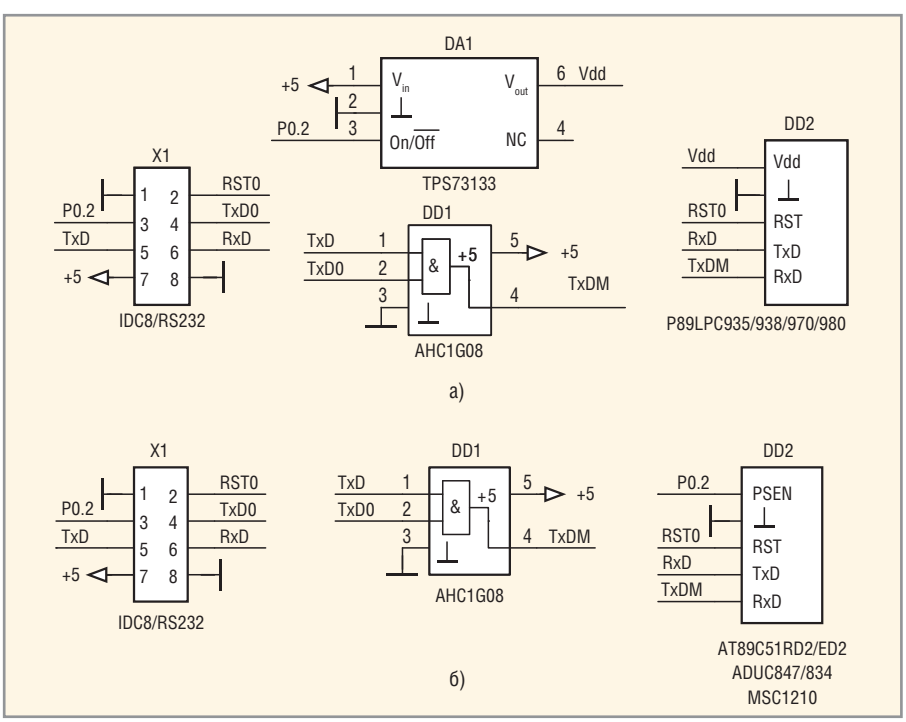

Рис. 21. Схемы подключения преобразователя USB-RS-232 к микроконтроллерам P89LPC935/938/970/980 (а) и AT89C51RD2/ED2, ADUC847/834, MSC1210 (б)

изоляции интерфейса USB при работе в режиме тестирования обмена только по USB приведено на рисунке 24. Изолированная сторона питается от внешнего источника питания напряжением +5 В, которое через разъём X3 подаётся на вывод 16 микросхемы ADUM3160 (DD1), а через разъём X2 – на линию VBUS микроконтроллера C8051F321 (см. схемы на рис. 17 и 18).

Программная поддержка обмена данными по шине USB ничем ни отличается от программного обеспечения (ПО) всех предыдущих устройств. Программное обеспечение преобразования USB-SPI отличается только конфигурацией выводов микроконтроллера C8051F321 и должно соответствовать рис. 19а, а программное обеспечение обмена по интерфейсу RS-232 для схем, показанных на рисунке 21, можно найти в [3].

Микросхемы ADUM3160/4160 широко доступны и могут быть приобретены в единичных экземплярах во многих фирмах, торгующих электронными компонентами, за рублевый эквивалент 20...30 долл. США.

Программные средства

Описанные ниже программы для компьютера и микроконтроллеров C8051F321/C8051F067 демонстрируют работу компьютера и устройств по обмену информацией в тестовом режиме. После текстов программ приведены результаты их работы и некоторые сравнительные характеристики представленных в статье устройств.

Основная идея тестовой программы заключается в следующем. Массив данных (строка) объемом 4 Кб передаётся по интерфейсу USB из компьютера в микроконтроллер C8051F321, принимается этим МК, затем передаётся из микроконтроллера C8051F321 в микроконтроллер C8051F067 уже по интерфейсу SPI, принимается микроконтроллером C8051F067, несколько модифицируется (изменяется порядок следования символов). Затем модифицированный массив передаётся из микроконтроллера C8051F067 обратно в микроконтроллер C8051F321 по интерфейсу SPI, принимается микроконтроллером C8051F321, затем передаётся этим МК в компьютер уже по интерфейсу USB и принимается компьютером.

На экран выводится передаваемая строка и принятая (модифицированная микроконтроллером C8051F067) строка для сравнения их между собой (для экономии места на экране сравниваются 1/64 части строк, т.е. строки, состоящие из 64 символов). Эта процедура повторяется 100 раз. Для измерения времени обмена информацией

(и, соответственно, подсчёта скорости) перед передачей массива запускается организованный на компьютере таймер, который останавливается после полного завершения всего цикла приёма данных. Зная объём данных (4 Кб), количество передач и время работы программы по таймеру в секундах, легко подсчитать скорость обмена.

Программа для микроконтроллера C8051F321 имеет две модификации. Первая кратко описана выше, а вторая отличается тем, что после приёма массива из компьютера по интерфейсу USB, микроконтроллер C8051F321 сразу же передаёт его (массив) обратно в компьютер (т.е. не посылает этот массив в микроконтроллер C8051F067). Вторая модификация позволяет оценить скорость обмена исключительно по интерфейсу USB, а первая – общую скорость работы по интерфейсам USB+SPI.

Фактически должны быть разработаны три программы для разных устройств:

- программа для компьютера, которая передаёт массив данных по интерфейсу USB в микроконтроллер C8051F321 и принимает от него массив такого же объёма;
- программа для микроконтроллера C8051F321, которая принимает массив из компьютера по интерфейсу USB и либо посылает его в микроконтроллер C8051F067 (по SPI), принимает этот массив из него и посылает в компьютер, либо сразу же возвращает массив в компьютер;
- программа для микроконтроллера C8051F067, которая принимает массив из микроконтроллера C8051F321 по интерфейсу SPI, модифицирует этот массив и отправляет его обратно в микроконтроллер C8051F321.

Программы для микроконтроллеров, в свою очередь, должны иметь модификации для работы соответственно в 3- и 4-проводном режимах SPI. Тексты программ приведены на сайте журнала. Предварительно автору хотелось бы остановиться на некоторых моментах, связанных как с интерфейсом USB, так и с интерфейсом SPI.

Особенности программ, связанных с интерфейсом USB

Для обмена информацией по интерфейсу USB между компьютером и микроконтроллером C8051F320/321 фирма Silicon Labs бесплатно поставляет

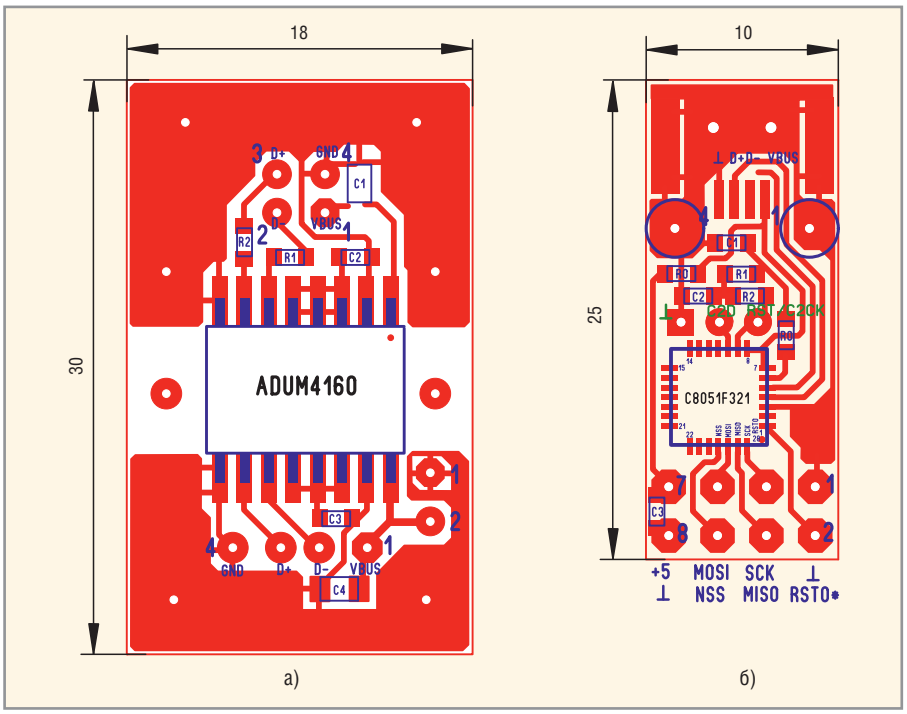

Рис. 22. Рисунки печатных плат гальванически изолированного интерфейса USB (а) и неизолированного преобразователя интерфейсов USB-SPI/USB-RS-232 (б)

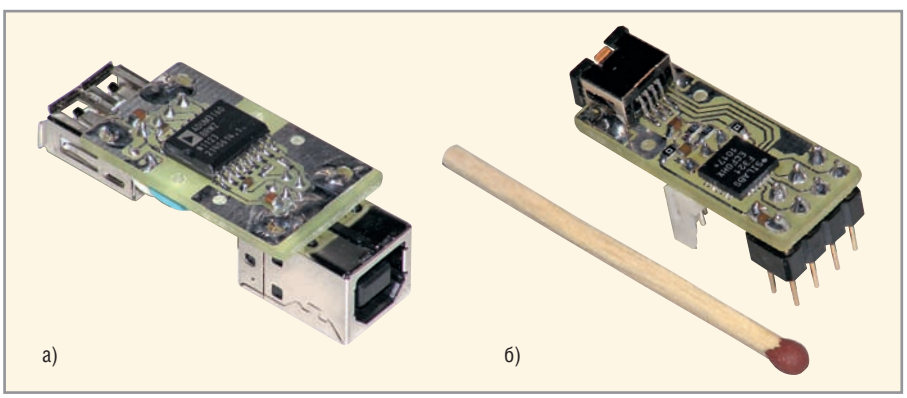

Рис. 23. Фотографии модулей гальванически изолированного интерфейса USB (а) и неизолированного преобразователя интерфейсов USB-SPI/RS-232 (б)

специальный драйвер, который необходимо установить на компьютер, и две библиотеки функций для программирования такого обмена. Первая библиотека предназначена для компьютера, вторая – для микроконтроллера. Общее название этих программных средств – USBExpress, описание всех функций приведено в приложении AN169.pdf [5], а пример работы – в программах для компьютера и МК с общим название FileTransfer [6]. После экспериментов с библиотечными функциями оказалось, что не все функции библиотеки, предназначенной для компьютера, выполняются так, как описано в приложении AN169.pdf, некоторые свойства функций, предназначенных для МК, не описаны в приложении AN169.pdf вообще, хотя используются в примере программы (FileTransfer) обмена информацией по USB для МК и обязательны для выполнения.

Библиотечных функций USBXpress как для компьютера, так и для микроконтроллера достаточно много, однако обязательных – всего несколько. Поэтому вначале будут приведены и описаны свойства функций, использованных для программирования обмена в компьютере, а затем – свойства

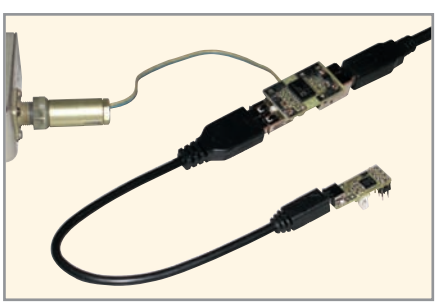

Рис. 24. Подключение платы USB-SPI к плате изолированного интерфейса USB

функций для программирования обмена в микроконтроллере.

Программы, предназначенные для компьютера, были написаны на языке Clarion v.6.0. Операторы этого языка очень похожи на операторы языка Бейсик, поэтому они могут быть понятны даже новичку, а опытные программисты легко смогут перевести эти операторы на тот язык, на котором они привыкли работать.

Для программирования обмена по USB были использованы шесть библиотечных функций:

1) функция открытия коммуникаций по USB и получения числового значения хандлера SI_Open(0,Handler). После её применения функция возвращает своё значение, т.е. число, которому она равна. Кроме того, у этой функции два параметра. Первый параметр необходимо задать (нулём), а второй (переменную Handler) – получить. Числовое значение переменной Handler, возвращаемое функцией SI_Open(0,Handler), используется в других функциях, поэтому этой функцией должна предваряться вся программа. Сама функция SI_Open(0,Handler) может принимать несколько значений, но главным является её успешное выполнение, при котором она принимает нулевое значение (SI_SUCCESS). Таким образом, строка loop while SI_Open(0,Handler) открывает коммуникацию по USB и возвращает значение;
2) функция закрытия коммуникаций по USB SI_Close(Handler). Работа этой функции похожа на первую, за исключением того, что в ней всего один параметр Handler, который требуется задать (равным полученному функцией SI_Open()). После окончания работы с USB этот интерфейс необходимо «закрыть» с помощью строки loop while SI_Close(Handler);
3) функция записи по USB SI_Write(Handler,Buffer,NumBytesToWrite,NumBytesWritten,0). При успешном выполнении значение этой функции нулевое, а параметров у нее пять:
   - Handler – параметр, о котором уже говорилось и который необходимо задать;
   - Buffer – параметр, являющийся названием строки, которую необходимо передать по USB, – этот параметр необходимо задать;
   - NumBytesToWrite – параметр, равный количеству байтов в передаваемой строке, который нужно задать;
   - NumBytesWritten – параметр, равный количеству байтов, реально переданных по USB, который возвращается функцией SI_Write() и который при необходимости можно вывести на экран;
   - 0 – пятый параметр.

   Например,если требуется передать по USB строку с названием STR, имеющую длину в 5 байт и значение STR='АБВГД', то следующая запись выполнит это действие: loop while SI_Write(Handler,STR,5,NumBytesWritten,0). После успешного выполнения этой функции переменная NumBytesWritten будет равна количеству переданных байтов, т.е. 5;
4) функция чтения по USB SI_Read(Handler,Buffer,NumBytesToRead,NumBytesReturned,0). При успешном выполнении значение этой функции также нулевое, и параметров у неё тоже пять:
   - Handler – параметр, который необходимо задать;
   - Buffer – параметр, являющийся названием строки, в которую необходимо записать принятые по USB байты, – этот параметр следует задать;
   - NumBytesToRead – параметр, равный количеству байтов в принимаемой строке, который требуется задать;
   - NumBytesReturned – параметр, равный количеству байтов, реально принятых по USB, который возвращается функцией SI_Read() и который при необходимости можно вывести на экран;
   - 0 – пятый параметр.

   Если требуется принять 64 байта информации по USB и записать эту информацию в строку с названием STR1, имеющую длину 64 байта, то следующая запись выполнит это действие: loop while SI_Read(Handler,STR1,64,NumBytesReturned,0). После успешного выполнения этой функции, как описано в приложении AN169.pdf, переменная NumBytesReturned будет равна количеству принятых байтов, т.е. 64, которые запишутся в строку STR1. Однако это не всегда так, и при успешном выполнении эта функция может возвратить количество байт, не равное 64, а принятые байты не будут соответствовать переданным из микроконтроллера. Дело в том, что применять функцию SI_Read() можно только удостоверившись в том, что эти 64 байта действительно поступили и готовы для чтения. Иначе можно получить непредсказуемый результат. Для проверки, что эти 64 байта действительно поступили из USB и записались в строку STR1, служит другая функция, описание которой приведено ниже;
5) функция проверки очереди приёмника SI_CheckRXQueue(Handler,NumBytesInQueue,QueueStatus). При успешном выполнении значение этой функции также нулевое, а параметров у неё три:
   - Handler – параметр, который необходимо задать;
   - NumBytesInQueue – параметр, равный количеству реально полученных по USB байтов;
   - QueueStatus – параметр, который может принимать несколько значений, одно из которых, SI_RX_READY, означает, что принятые данные готовы для чтения.

   При объёме ОЗУ микроконтроллера менее 4 Кб можно вообще не использовать переменную QueueStatus, а использовать переменную NumBytesInQueue, равную количеству реально пришедших по USB байтов:

```
loop until NumBytesInQueue=4096
loop while SI_CheckRXQueue(Hand-
ler,NumBytesInQueue,QueueStatus).
```

Таким образом, чтобы передать в микроконтроллер по USB массив (строку) объёмом в 4 Кб и принять из МК по USB массив такого же объёма (и записать его в строку с именем, предположим, SL1), требуется выполнить следующие три действия в виде трёх записей, разделённых между собой строками комментария (пунктир):

```
!-------------
loop while
SI_Write(Handler,SL,4096,Num-
BytesWritten,0).
!-------------
loop until NumBytesInQueue=4096
 loop while SI_CheckRXQueue(Han-
dler,NumBytesInQueue,QueueSta-
tus).
!-------------
 loop while
SI_Read(Handler,S1L,4096,Num-
BytesReturned,0).
!-------------
```

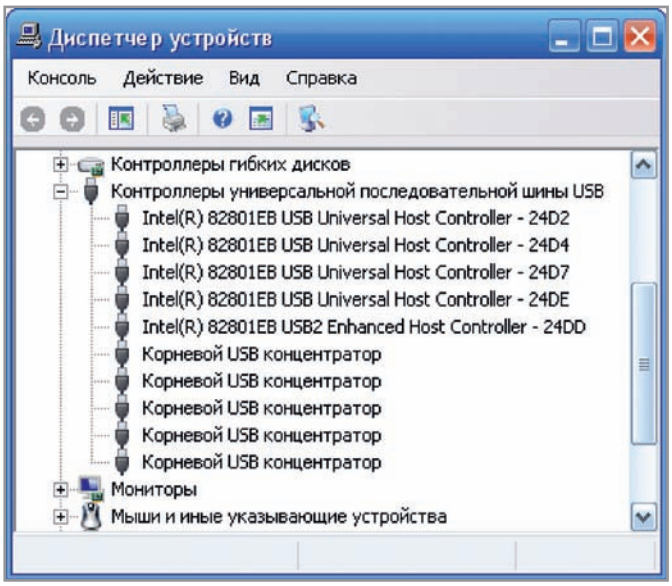

Рис. 25. Окно диспетчера устройств перед подключением устройства к компьютеру

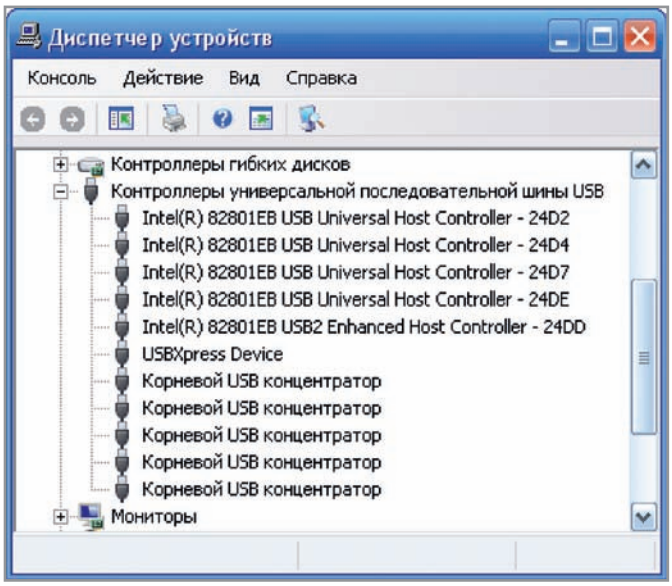

Рис. 26. Окно диспетчера устройств после подключения устройства к компьютеру

Эти три операции являются ядром программы, полный текст которой представлен на сайте журнала. Остальной код относится к определению значений необходимых переменных и прототипов функций, открытию окна, отображающего результат работы программы, распечатки значений переменных в этом окне и т.п. Приведённые прототипы можно переписать в Delphi, если используется базовый язык Паскаль. Если же программировать на языке Си, то следует использовать «родные» прототипы, приведённые в программе FileTransfer для компьютера;

6) следует использовать ещё одну функцию USBXpress, которая очищает буферы приёма и передачи массивов данных. Для её применения необходима запись loop while SI_FlushBuffers (Handler,0,0). Эту операцию следует применить после открытия USB, иначе программа работать не будет.

Существует функция получения описания устройства USB SI_GetProduct-String(NumDevices,DevString,Options), которая не является обязательной, но заслуживает внимания. У функции три параметра:

- NumDevices – этот параметр должен быть всегда равен нулю (NumDevices=0);
- DevString – этот параметр является названием строки, в которую записывается возвращаемое функцией SI_GetProductString() описание устройства;
- Options – этот параметр должен быть задан. Если options SI_Return_SeriaL_Number, то строка DevString примет значение '4321', что означает устройство на базе микроконтроллера C8051F320/321.

Если параметр Options задать равным SI_Return_Description:

```
NumDevices=0
loop while
SI_GetProductString(NumDevices,DevString,SI_Return_Description),
```

то строка DevString примет значение 'USBXpress Device'. Это – описание устройства, которое отражается в панели управления и появляется после подключения устройства с микроконтроллером C8051F321 к компьютеру.

Если до подключения устройства к компьютеру зайти по адресу Пуск/Настройка/Панель управления/Система/Оборудование/Диспетчер устройств/ Контроллер универсальной последовательной шины USB/, то на экран выведется окно, показанное на рисунке 25. После подключения устройства к компьютеру картинка изменится и будет иметь вид, показанный на рисунке 26. Заметим, что в окне появилось устройство 'USBXpress Device'. Строку DevString можно для контроля вывести на экран, хотя делать это не обязательно (так же, как и использовать функцию SI_GetProductString()).

Как видно из представленных выше функций, программирование обмена по USB со стороны компьютера (при использовании библиотеки USBXpress) не требует знаний спецификации USB. Программирование обмена по USB со стороны МК ещё проще, однако там тоже есть особенности. Но вначале о функциях, которые обязательны для использования.

- USB_Clock_Start(); – это функция без параметров, которая инициализирует внутренний генератор для работы по интерфейсу USB и которую необходимо использовать в самом начале программы;
- USB_Init(USB_VID,USB_PID,USB_MfrStr,USB_ProductStr,USB_SerialStr,USB_MaxPower,USB_PwAttributes,USB_bcdDevice); – это функция инициализации USB, параметры которой жёстко определены и которая является также обязательной для исполнения. Подробное описание всех параметров приведено в документе AN169.pdf;
- USB_Int_Enable(); – это функция без параметров, которая разрешает прерывания по различным событиям в интерфейсе USB и является обязательной для выполнения;
- Block_Read (BYTE *Buffer, NumBytes) – это функция с параметрами, предназначенная для чтения информации, поступившей по USB. Первый параметр Buffer – название массива, в который будут записываться поступившие байты; символ * означает, что это адрес массива. Второй параметр NumBytes – количество байт для считывания. Его максимальное значение рано 64, т.е. передача информации по USB в микроконтроллер осуществляется пакетами с максимальным объёмом в 64 байта каж-

дый. Таким образом, если определён массив, например, INARRAY[64] размером в 64 байта, то для того чтобы в этот массив записалась информация, поступившая по USB, требуется следующая запись: Block_Read((BYTE*)&INARRAY, 64);

- Block_Write (BYTE *Buffer, NumBytes) – это функция с параметрами, предназначенная для передачи информации из микроконтроллера в интерфейс USB. Первый параметр Buffer – название массива информации, предназначенной для передачи; символ * означает адрес массива. Второй параметр NumBytes – количество байтов, которые требуется передать. Его максимальное значение равно 4 Кб (4096 байтов), т.е. передача информации из МК по USB осуществляется пакетами с максимальным объёмом в 4096 байтов каждый.

Таким образом, если определён массив, например, INARRAY[1024] размером в 1024 байта, то для его передачи по USB требуется запись:

```
Block_Write((BYTE*)&INARRAY,1024)
```

Символ '&' перед названием массива INARRAY означает, что обращение к массиву производится по адресу.

В отличие от программ для компьютера, приём и передача информации в микроконтроллере организованы с помощью прерываний, источники которых описаны в приложении AN169.pdf. Обязательными для использования, по мнению автора, являются только три:

- RX_Complete – это прерывание возникает, когда в буфер приёмника USB поступит информация и данные можно считать функцией Block_Write();
- TX_Complete – это прерывание возникает, когда данные, записанные с помощью функции Block_Write(), полностью переданы по USB в компьютер;
- Device_Close – это прерывание возникает, когда интерфейс USB закрывается со стороны компьютера функцией SI_Close(Handler), о которой рассказывалось выше.

На сайте журнала приведён фрагмент программы, которая предназначена для приёма микроконтроллером C8051F321 массива размером в 4 Кб, переданным компьютером, передачи и приёма этого массива по интерфейсу SPI в/из микроконтроллера C8051F067 и передачи массива обратно в компьютер. Этот фрагмент является основным в программе, полный текст которой также приведён на сайте журнала.

Фрагмент фактически состоит из двух подпрограмм. Первая подпрограмма State_Machine() определяет, что необходимо сделать с информацией в зависимости от состояния МК, а вторая подпрограмма USB_API_Test_ISR() обрабатывает прерывания, возникающие в интерфейсе USB.

Если рассмотреть подпрограмму обработки прерываний USB_API_TEST_ISR(), то можно заметить, что в ней используются три прерывания: TX_Complete, RX_Complete и Device_Close, два из которых (TX_Complete и RX_Complete) обрабатываются программой State_Machine(), а третье (Device_Close) этой программой не обрабатывается. Можно сделать вывод, что это прерывание не требуется (т.к. не определено, что требуется сделать, если оно возникает), и состояние M_State = ST_Wait_DEV вообще не рассматривать. Однако это приведёт к тому, что программа обмена по USB в компьютере запустится только один раз, а при повторном запуске «зависнет». В связи с этим непонятно, что происходит с микроконтроллером, когда M_State = ST_Wait_DEV, и как библиотека USBXpress использует это состояние.

Во фрагменте программы, приведённом выше, обмен по SPI описан между двумя строками, состоящими из звездочек (*), т.е. от закомментированного оператора // goto AAA; до метки //AAA:. Если убрать комментарии перед этим оператором (и перед тремя предыдущими) и меткой, то обмен по SPI будет пропущен и принятые из компьютера данные, минуя интерфейс SPI, будут сразу передаваться в компьютер. Этот режим работы USB удобно использовать при оценке скорости обмена по USB или при других обстоятельствах, когда обмен по SPI не требуется. В таком режиме, например, работает устройство, подключенное к компьютеру, как показано на рисунке 16а.

Особенности программ, связанных с интерфейсом SPI

В обмене по SPI в вышеприведённом фрагменте присутствует подпрограмма HSSPI(), текст которой приведён ниже.

```
void HSSPI (void) {
while(!MOSI); //только для режима
3-wire. В режиме 4-wire
while(MOSI); //синхронизация не
обязательна,
//но хуже не будет.
//-------------------------------
-------------
SPIEN=1;
outspi(0x40);
}
```

Эта подпрограмма предназначена для синхронизации начала обмена по SPI ведомого микроконтроллера с ведущим в трёхпроводном режиме. Дело в том, что в трёхпроводном режиме момент начала обмена по SPI между ведомым МК и ведущим МК не определён, т.к. отсутствует определяющий сигнал NSS. В результате может сложиться ситуация, когда ведущий МК пытается ввести байт, посылаемый ведомым МК, но из-за того, что ведомому неизвестен момент начала чтения байта ведущим, посылка байта ведомым МК может начаться не с начала чтения байта ведущим МК, а, например, после четвёртого стробирующего импульса SCK. Это приведёт к тому, что, во-первых, ведущий МК прочитает неверный результат и, во-вторых, ведомый МК, выведя 4 бита, будет ожидать ещё четыре импульса SCK, и, пока они не поступят, вывод остальных 4 бит будет приостановлен. При повторной попытке чтения байта ведущим МК, первые четыре импульса SCK, посылаемые ведущим МК, продолжат вывод остальных 4 бит ведомым МК, в результате чего ведущий МК опять получит неверный результат, и ситуация повторится. В результате либо обмен сорвётся, либо всё «зависнет». Чтобы такая ситуация не возникла, ведомый МК должен посылать байт именно в тот момент, когда ведущий МК начинает чтение этого байта. Но как ведомый МК может определить этот момент без сигнала NSS?

Предположим, для начала обмена информацией ведущий микроконтроллер должен получить подтверждение от ведомого МК, – какой-нибудь известный обоим микроконтроллерам байт. Если такой байт получен ведущим МК, то для него это будет означать, что ведомый МК готов передавать информацию. Пусть, для примера, значение такого байта будет равно 0×40 (символ '@').

Как известно, для ввода или вывода байта по SPI требуется записать информацию в регистр SPI0DAT. Если ве-

дущему микроконтроллеру требуется вывести байт, то записанный в SPI0DAT байт выводится по линии MOSI. Если же ведущему МК требуется ввести байт, то в регистр SPI0DAT записывается «ненужный» нулевой байт (0×00), который также выводится по линии MOSI, а вот по линии MISO в ведущий МК вводится информационный байт, посылаемый ведомым МК. Таким образом, линия MOSI при вводе байта ведущим работает просто на холостой вывод байта, который ни ведущему, ни ведомому МК не нужен, да и сама линия MOSI в этот момент (в информационном смысле) не используется. Так почему бы не использовать эту линию для синхронизации вывода байта ведомым МК с вводом байта ведущим МК?

Пусть значение неинформативного байта, выводимого ведущим МК по линии MOSI, равно 0×0F. Тогда на линии MOSI при нескольких попытках ввода байта ведущим МК сигнал будет иметь форму меандра (см. рис. 27). Анализируя состояние линии MOSI, ведомый МК может приурочить вывод своего байта 0×40, или символа '@', как раз к тому моменту, когда ведущий будет его читать. Этот момент происходит каждый раз, когда на линии MOSI будет очередной срез импульса (t_0 на рис. 27). Этот момент t_0 ведомый микроконтроллер может легко определить с помощью всего двух команд:

```
while(!MOSI);
while(MOSI);
```

Таким образом, независимо от того, в какой момент времени начнут выполняться две приведённые команды, их выполнение закончится строго в момент t_0. Если теперь ведомый МК выведет символ '@', то ведущий МК правильно его считает. Чтобы вывести этот символ, в программе ведомого МК должны быть записаны следующие две команды:

```
SPIEN=1;
outspi(0x40);
```

Эти две команды, естественно, выполнятся с некоторой задержкой t_3 (см. рис. 27), однако если это время истечёт до первого фронта импульса SCK, обмен произойдёт без ошибок. Если же время задержки превысит время от момента t_0 до фронта первого импульса SCK, то ведомый МК опять считает неверный результат. Время задержки

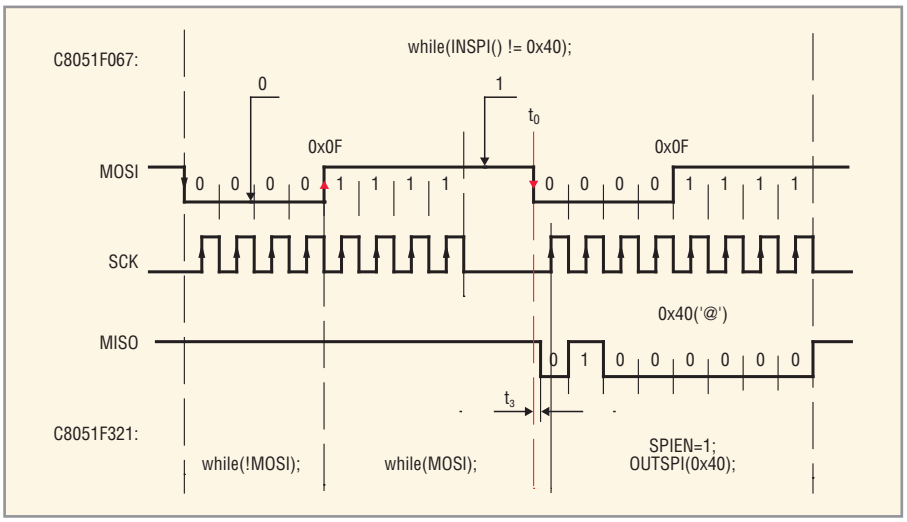

Рис. 27. Сигналы интерфейса SPI при синхронизации ведомого МК с ведущим МК

скорости обмена по SPI зависит от оперативности, с которой ведомый микроконтроллер выполнит вышеприведённые четыре команды. Это время также зависит от того, какую задержку вносят ИС цифровых изоляторов. Например, микросхемы ISO7220C и ADUM7441C, работающие на максимальных скоростях 25 Мбод, вносят задержку в 40...50 нс, а микросхема SI8663BC, работающая на скоростях до 150 Мбод, вносит задержку в несколько наносекунд. Естественно ожидать от неё работу по интерфейсу SPI с большей скоростью (в реальности так и произошло).

Отметим, что эти четыре команды составляют программу синхронизации, приведённую выше и названную HSSPI(). Символы HS – от первых букв Hand Shake, т.е. рукопожатие (ведомого и ведущего микроконтроллера).

Следует также упомянуть о необходимости «сдвинуть» сигналы SPI на один разряд вправо в матрице соединений для микроконтроллера C8051F321. На рисунке 28а показана матрица соединений, на которой сигналы SPI занимают выводы P0.0 – P0.3 (стандартный вариант), а на рисунке 28б – после того, как вывод P0.0 пропущен командой skip. В таком состоянии вывод P0.0 – вывод 2 микросхемы микроконтроллера C8051F321 (DD1, см. рис. 1) можно использовать для сигнала сброса микроконтроллера C8051F067 (RST067). Тогда печатные платы устройств 3-wireSPI и 4-wireSPI разводятся намного проще (пропуск сделан исключительно для удобства).

Если тщательно подойти к выбору портов матрицей соединений, то для каждого из устройств 3-wireSPI и 4-wireSPI можно найти решение, оптимальное с точки зрения разводки платы. Например, если выбрать матрицу соединений в соответствии с конфигурацией, показанной на рисунке 29, и для сигнала RST067 использовать порт P0.1, то плату 4-wireSPI_AD можно развести оптимально. Однако если для каждого из устройств выбрать свою конфигурацию матрицы соединений, потребуются отдельные программы.

В листингах 1 – 3 (см. дополнительные материалы на сайте журнала) приведены тексты всех трёх программ, представляющие собой программное обеспечение всех устройств, описанных в статье. Это программа для компьютера, написанная на языке Clarion v.6.0, со своим файл-проектом, программа для микроконтроллера C8051F321 со своим запускающим*.bat-файлом, предназначенным для её трансляции, и программа для микроконтроллера C8051F067 также со своим *.bat-файлом.

В связи с тем что в программах использованы примеры, взятые из программ FileTransfer [6], а для конфигурирования портов ввода/вывода и периферийных устройств микроконтроллеров – бесплатно поставляемая фирмой Silicon Labs программа Config2.exe (v. 2.5), комментарии от Silicon Labs в текстах программ сохранены в оригинальном виде.

Обсуждение результатов

Результаты работы программ показаны на рисунке 30 при различных режимах работы микроконтроллеров C8051F321 и C8051F067. На рисунке 30а приведён снимок экрана, когда микроконтроллер C8051F321 работает только с интерфейсом USB, а обмен по интерфейсу SPI пропущен. Можно за-

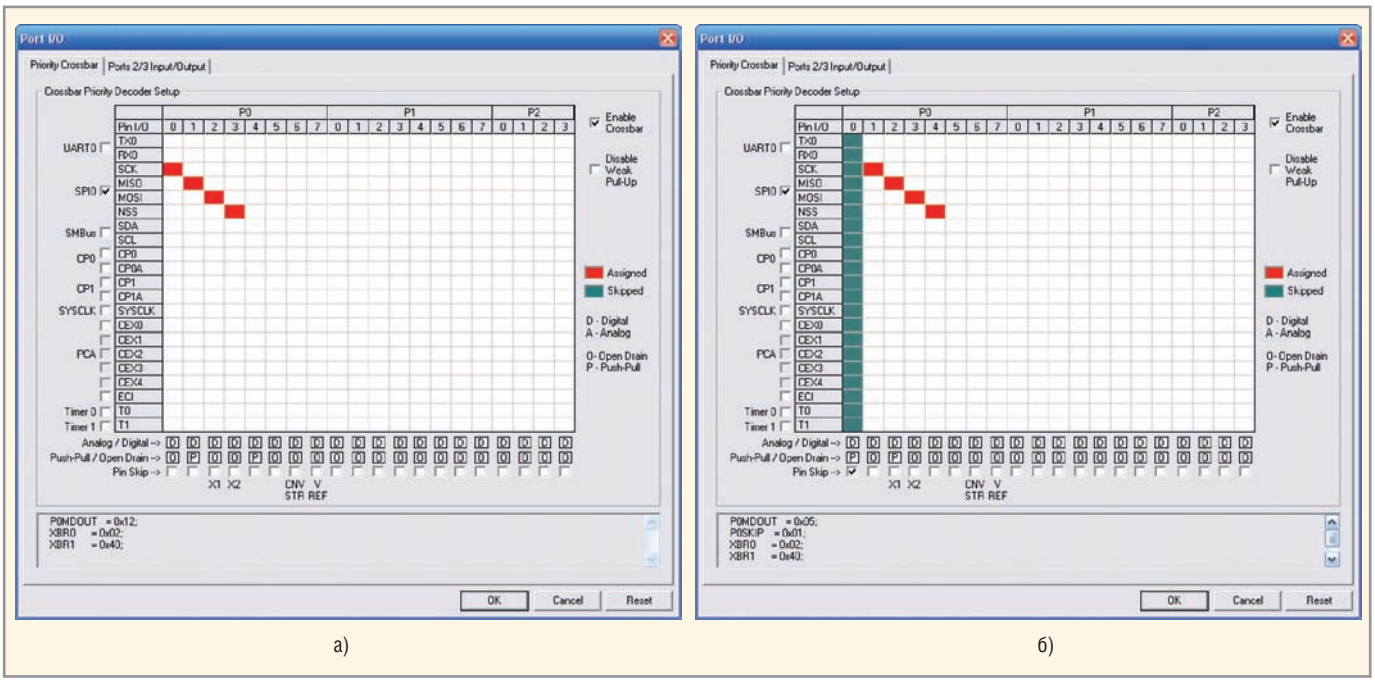

Рис. 28. Вид матрицы соединений до пропуска (skip) вывода P0.0 (а) и после него (б)

метить, что скорость обмена по USB составляет 5 Мбод. Теоретическое значение скорости можно вычислить по формуле:

$$V = ((8*2*4096{,}0*100\ [\text{бит}])/1{,}29\ [\text{с}])/ /1000000{,}0 = 5{,}08\ \text{Мбод}. \quad (1)$$

На рисунке 30б показан снимок экрана, когда микроконтроллер C8051F321 работает по интерфейсу USB с компьютером, а по интерфейсу SPI – с микроконтроллером C8051067 в четырёх проводном режиме на скорости SPI в 4 Мбод (точнее, частота сигнала SCK, F_{sck} = 4 МГц). Можно заметить, что общая скорость обмена (USB+SPI) составляет 2,7 Мбод. Скорость в этом режиме рассчитана по формуле (1), в которую добавлен ещё один множитель 2 из-за добавления передачи пакета по SPI из микроконтроллера C8051F321 в микроконтроллер C8051F067 и обратно:

$$V = ((8*2*2*4096{,}0*100\ [\text{бит}])/4{,}89\ [\text{с}])/ /1000000{,}0 = 2{,}68\ \text{Мбод}.$$

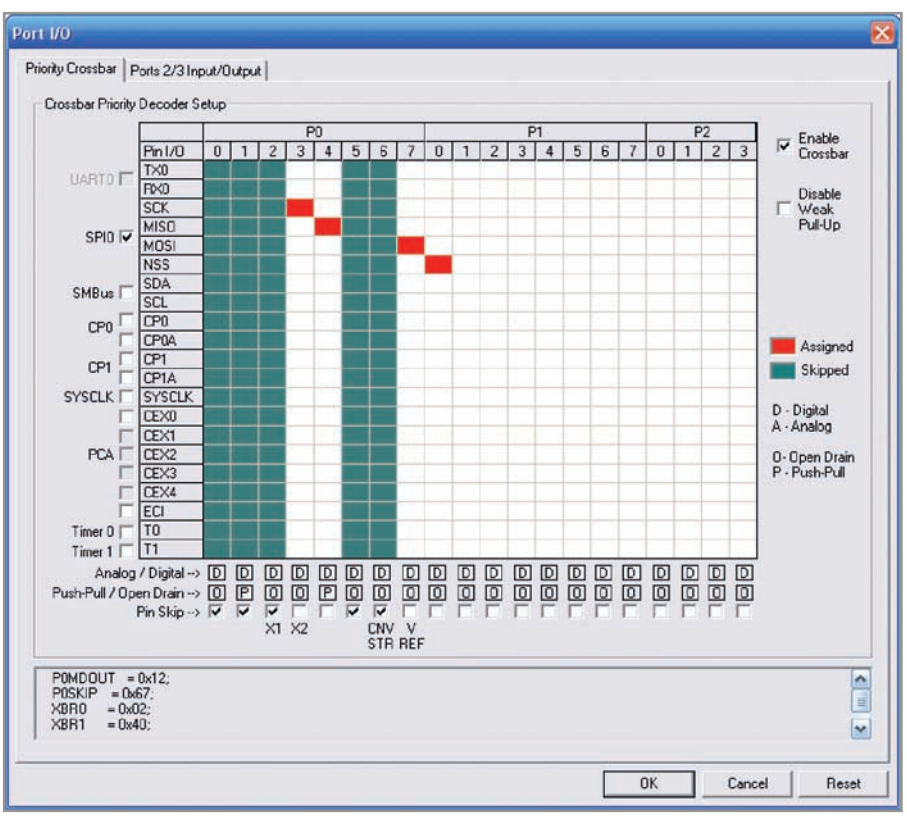

Рис. 29. Пример выбора матрицы соединений для оптимальной разводки платы устройства 4-wireSPI_AD

На самом деле средняя скорость обмена в этом режиме подсчитана неточно. Если исходить из того, что обмен только по USB идёт со скоростью 5 Мбод и занимает 1,29 с (см. рис. 30а), то обмен по SPI (рис. 30б) длится 4,89 – – 1,29 = 3,6 с. Тогда можно подсчитать реальную скорость работы по SPI по формуле (1), подставив в неё время 3,6 с. Результат такого расчёта даёт скорость SPI в 1,8 Мбод. А средняя скорость обмена по USB + SPI будет равна (5 Мбод + 1,8 Мбод)/2 = 3,44 Мбод.

На рисунке 30в приведён снимок экрана, когда микроконтроллер C8051F321 работает по интерфейсу USB с компьютером, а по интерфейсу SPI – с микроконтроллером C8051067 в четырёхпроводном режиме при F_{sck} = 3 МГц, т.е. на идеальной скорости SPI в 3 Мбод, которая, как было отмечено выше, отличается от реальной. Можно заметить, что общая скорость обмена (USB + SPI) составляет 2,4 Мбод. Реальная скорость SPI равна 1,57 Мбод, а средняя скорость обмена по USB + SPI составляет 3,29 Мбод.

Эксперименты со всеми четырьмя устройствами, проведённые автором, показали следующее.

Устройства 4-wireSPI_SI и DIP22 работают в 3- и 4-проводном режиме SPI при максимальной частоте F_{sck} = 4 МГц,

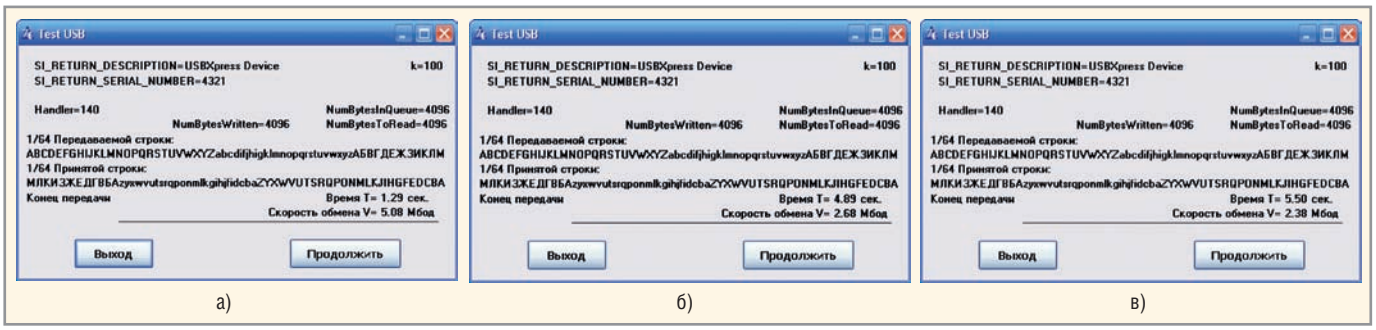

Рис. 30. Результаты работы программ обмена по USB

а) Только по USB, б) USB+SPI, скорость SPI 4 Мбод, в) USB+SPI, скорость SPI 3 Мбод

устройство 4-wireSPI_AD работает в 3- и 4-проводном режиме SPI при максимальной частоте F_{sck} = 3 МГц, а устройство 3-wireSPI работает в трёхпроводном режиме SPI при максимальной частоте F_{sck} = 3 МГц. Эти скоростные характеристики были получены при условии программной синхронизации ведомого микроконтроллера с ведущим в трёхпроводном режиме (см. рис. 27), когда ведущий МК работал от внешнего генератора на кварцевом резонаторе с частотой в 24 МГц. Если не использовать программную синхронизацию в трёхпроводном режиме, то частота импульсов SCK должна быть снижена до значения F_{sck} = 1 МГц для всех устройств (иначе обмен невозможен). Если при этом не использовать кварцевый резонатор и заставить работать ведущий МК от внутреннего генератора с частотой 24,5 МГц, то частоту F_{sck} необходимо снизить до 200...300 кГц. В противном случае либо обмен вообще невозможен, либо имеют место частые сбои (из пяти запусков два не срабатывают).

По результатам экспериментов можно сделать следующие выводы о надёжности и скорости обмена по интерфейсу SPI:

- программная синхронизация и кварцевый резонатор, подключенный к ведущему микроконтроллеру, значительно улучшают надёжность и увеличивают скорость обмена по интерфейсу SPI;
- устройство 4-wireSPI_SI работает с частотой F_{sck} = 4 МГц, максимально возможной для двух микроконтроллеров C8051F321 и C8051F067, если вообще не использовать гальванические развязки, т.е. соединять микроконтроллеры по SPI напрямую, как это сделано в устройстве DIP22. Такой результат – следствие высокой скорости работы гальванической развязки SI8663BC (до 150 Мбод) и малого времени задержки прохождения сигнала (6...10 нс);
- гальванические развязки ISO7220C и ADUM7441C, имеющие максимальные скорости работы до 25 Мбод и время задержки до 40...50 нс, позволяют производить обмен по SPI только при максимальной частоте F_{sck} = = 3 МГц. Это не означает, что данные микросхемы непригодны для использования. Во-первых, как было подсчитано выше, реальная скорость обмена по SPI при F_{sck} = 3 МГц составляет 1,57 Мбод против 1,8 Мбод при F_{sck} = 4 МГц. Во-вторых, существует микросхема более дорогая, ISO7220M, которая работает так же, как и SI8663BC, на скорости в 150 Мбод и имеет такое же время задержки (6...10 нс);
- максимально возможная скорость обмена по SPI, указанная в п. 2 (4 МГц), может быть увеличена в два раза, если вместо микроконтроллера C8051F321, работающего на максимальной тактовой частоте в 24 МГц, использовать микроконтроллеры C8051F342/3/A/B с удвоенной тактовой частотой 48 МГц. Однако, во-первых, эти микроконтроллеры в несколько раз (!) дороже микроконтроллеров C8051F321, а во-вторых, в миниатюрных корпусах QFN28 (т.е. с индексом GM) они недоступны.

Заключение

Использование микроконтроллеров C8051F321 со встроенным интерфейсом USB совместно с цифровыми изоляторами ISO7220C, ADUM7441C и особенно SI8663BC(SI8463BC) позволяет конструировать на их основе уникальные по своей простоте, миниатюрности и дешевизне устройства – преобразователи интерфейса USB в интерфейс SPI с гальванической развязкой.

Удобство и простота программного обеспечения связи по интерфейсу USB, поставляемого с микроконтроллерами C8051F321 (драйвер и библиотеки USBXpress), позволяют быстро программировать обмен информацией по USB без изучения спецификаций этого интерфейса, как со стороны компьютера, так и со стороны микроконтроллера.

Применение программной синхронизации, предложенной автором, при обмене информацией двух микроконтроллеров по интерфейсу SPI позволяет значительно повысить надёжность и скорость такого обмена, особенно в трёхпроводном режиме работы.

Устройства – преобразователи интерфейса USB в интерфейс SPI с гальванической изоляцией, описанные в статье, могут найти широкое применение в системах сбора и обработки информации, построенных на базе микроконтроллеров и компьютеров.

Литература

1. *Кузьминов А.* Метод фоторепродуцирования для изготовления фотошаблона печатных плат в домашних условиях. Технологии в электронной промышленности. 2010. № 5–7.
2. *Кузьминов А.* Изготовление устройств на печатных платах с высоким разрешением в домашних условиях. Технологии в электронной промышленности. 2010. № 8–10.
3. *Кузьминов А.Ю.* Интерфейс RS-232. Связь между компьютером и микроконтроллером. От DOS к Windows 98/XP. ДМК-пресс, 2006.
4. www.microcompsys.narod.ru.
5. www.silabs.com, AN169.pdf, USBXpress® Programmer's Guide.
6. www.silabs.com, FileTransfer Example.
7. www.analog.com.
8. *Кузьминов А.* Применение ИС цифровых изоляторов ADUM7441 и ADUM3473 для гальванической развязки интерфейс RS-232. Современная электроника. 2011. № 9.