1 Общее описание изделия USB2185

Изделие USB2185 представляет собой одноплатное интерфейсное устройство на шину USB (rev 2.0, rev 1.1). Изделие предназначено для подключения пользовательских (прикладных) устройств, блоков или систем ввода-вывода к компьютерам под управлением ОС Win2k, XP или Linux.

Изделие USB2185 самостоятельно (без участия пользовательских аппаратных и программных решений) обеспечивает скоростной (см. Технические характеристики) дуплексный обмен данными и передачу событий в обоих направлениях.

В то же время изделие обеспечивает гибкость при подключении пользовательской аппаратуры и позволяет вести первичную ЦОС (Цифровую Обработку Сигнала) при вводе и выводе. Помимо этого обеспечивается возможность выполнения алгоритмов управления объектом в жестком реальном времени, без использования ресурсов компьютера.

Данное сочетание свойств определяется структурой изделия (Рис. 1):

Обращённая к пользователю часть состоит из сигнального процессора (ADSP-2185M производства Analog Devices.) и внешнего разъема BH-50, с контактами которого напрямую соединены выводы DSP, позволяющие задействовать 16-ти разрядную шину данных, два универсальных скоростных сериальных порта, несколько входных и выходных логических линий, входы прерываний. Управление указанными ресурсами выполняется программой, загружаемой в DSP (драйвер DSP).

Часть, обращённая к шине USB (интерфейсная часть), осуществляет обмен информацией между ПК и DSP и функционирует под управлением микроконтроллера. Микроконтроллер предварительно запрограммирован управляющей программой (BIOS) производителем изделия («R-technology group»).

Связь между частями изделия при передаче данных между компьютером и внутренней памятью DSP осуществляется при помощи Внутреннего Прямого Доступа к Памяти (IDMA).

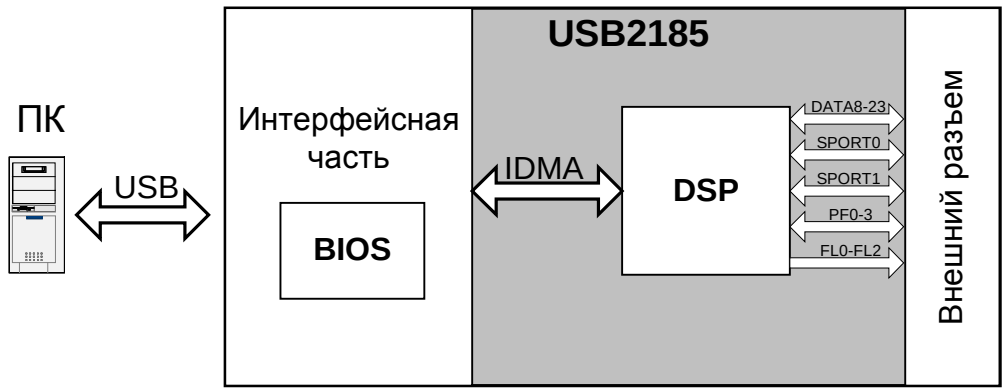


Рис. 1 Структурная схема изделия USB2185.

На практике использование такой структуры имеет следующие преимущества:

- ресурсы DSP не расходуются на управление обменом данными с ПК и полностью высвобождаются для задач пользователя — это обеспечивает уверенное выполнение требований жесткого реального времени в процессе сбора данных или управления объектом;
- со стороны DSP данные, подлежащие отправке в компьютер или принятые из компьютера, выглядят как области памяти DSP — это позволяет программисту DSP создавать программы под любые задачи, не вникая в тонкости архитектуры и протокола USB;

- загрузка программы в DSP осуществляется из компьютера через шину USB, что позволяет отказаться от использования дополнительных аппаратных средств (программатор, JTAG и т.д.). Радикально облегчается процесс отладки.

Кроме того, изделию присущи следующие положительные качества:

- В комплект бесплатного программного обеспечения, помимо универсального драйвера DSP, входят несколько вариантов драйверов, ориентированных на разные пользовательские задачи. Эти варианты поставляются как в виде загрузочных кодов, так и в исходных текстах на ассемблере ADSP21XX, что позволяет пользователю (при необходимости) самостоятельно модифицировать программу. Компиляция и создание загрузочного модуля производятся при помощи бесплатно распространяемого Analog Devices программного обеспечения. Т.к. программы собственно управления процессами сбора данных, как правило, имеют простую структуру и небольшой объём, а процесс передачи данных по протоколу USB реализован независимо, на отдельном микроконтроллере, возможна успешная реализация специализированных драйверов DSP пользователями, не имеющими большого опыта разработки встроенного программного обеспечения.

- В комплект бесплатного программного обеспечения входит специализированный драйвер для ОС Win2k, XP. Управление и обмен данными с устройством осуществляется при помощи входящей в комплект библиотеки функций API. Также в комплект входят примеры использования функций API.

- Пользователь имеет возможность, например, для обновления, легко «перепрошить» BIOS интерфейсной части модуля, обеспечивающей связь ПК с DSP. Данная операция выполняется по интерфейсу USB, при помощи программы, входящей в состав комплекта бесплатного программного обеспечения.

- В состав изделия входит энергонезависимое пользовательское ПЗУ (ППЗУ), которое позволяет пользователю хранить необходимую информацию, например, калибровочные коэффициенты пользовательского АЦП, сериальный номер, название и т.д. Чтение/запись из/в ППЗУ осуществляется программно, через шину USB, и не требует дополнительных аппаратных средств. Подробнее о работе с ППЗУ см. в руководстве программиста.

- На внешний разъем изделия кроме всех выводов DSP (за исключением линий, задействованных для обмена данными с ПК) выведены также питание 5В, питание 3.3В, опорная частота 36МГц - это позволяет пользователю создавать самые разнообразные устройства на базе изделия USB2185, подключая к нему различные модули расширения.

- Питание модуля может осуществляться как непосредственно от шины USB, так и от внешнего источника питания +5В +/- 10%. Низкая потребляемая мощность самого изделия USB2185 позволяет обойтись без внешнего источника. Внешний источник можно использовать при работе USB2185 совместно с различными пользовательскими узлами, если их потребление (совместно с USB2185) превышает допустимые стандартом USB 500 мА, а также при работе с ноутбуком, в целях экономии энергии аккумулятора ноутбука.

- Конструкция изделия позволяет легко размещать устройства на базе USB2185 в стандартных корпусах типа G738 или G738A фирмы Gainta.

1.1 Область применения

Основное назначение изделия USB2185 – служить базовым модулем для пользовательских устройств ввода/вывода и обработки информации.

На базе изделия USB2185 могут быть созданы самые разнообразные устройства, подключаемые к ПК через современную высокоскоростную шину USB, в том числе:

- АЦП
- ЦАП
- логические анализаторы
- системы хранения и обработки информации
- концентраторы данных для АСУТП
- мобильные измерительные комплексы на базе notebook

1.2 Технические характеристики

Тип DSP	ADSP-2185MKST-300
Рабочая частота DSP	72 МГц
Допустимое напряжение на контактах внешнего разъема относительно контактов 47-50 (земли):	0 … 3.3В.
Максимальная скорость передачи массивов данных:	
- из DSP в ПК	12 x 10^6 байт/с[1]
- из ПК в DSP	3.1 x 10^6 байт/с[2]
- дуплексная передача	6 x 10^6 байт/с в ПК и 250 x 10^3 байт/с из ПК
Ток потребления, не более	230 мА
Рабочая температура	10… 50 С
Габариты	117 x 99 мм

[1] Для увеличения скорости передачи данных в ПК до 14.4 x 10^6 байт/с необходимо прошить интерфейсную часть специализированным BIOS (см. Приложение А).

[2] Для увеличения скорости передачи данных из ПК до 14.4 x 10^6 байт/с необходимо прошить интерфейсную часть специализированным BIOS (см. Приложение А).

1.3 Индикаторы и устройства коммутации изделия USB2185

На Рис. 2 представлен внешний вид изделия USB2185.

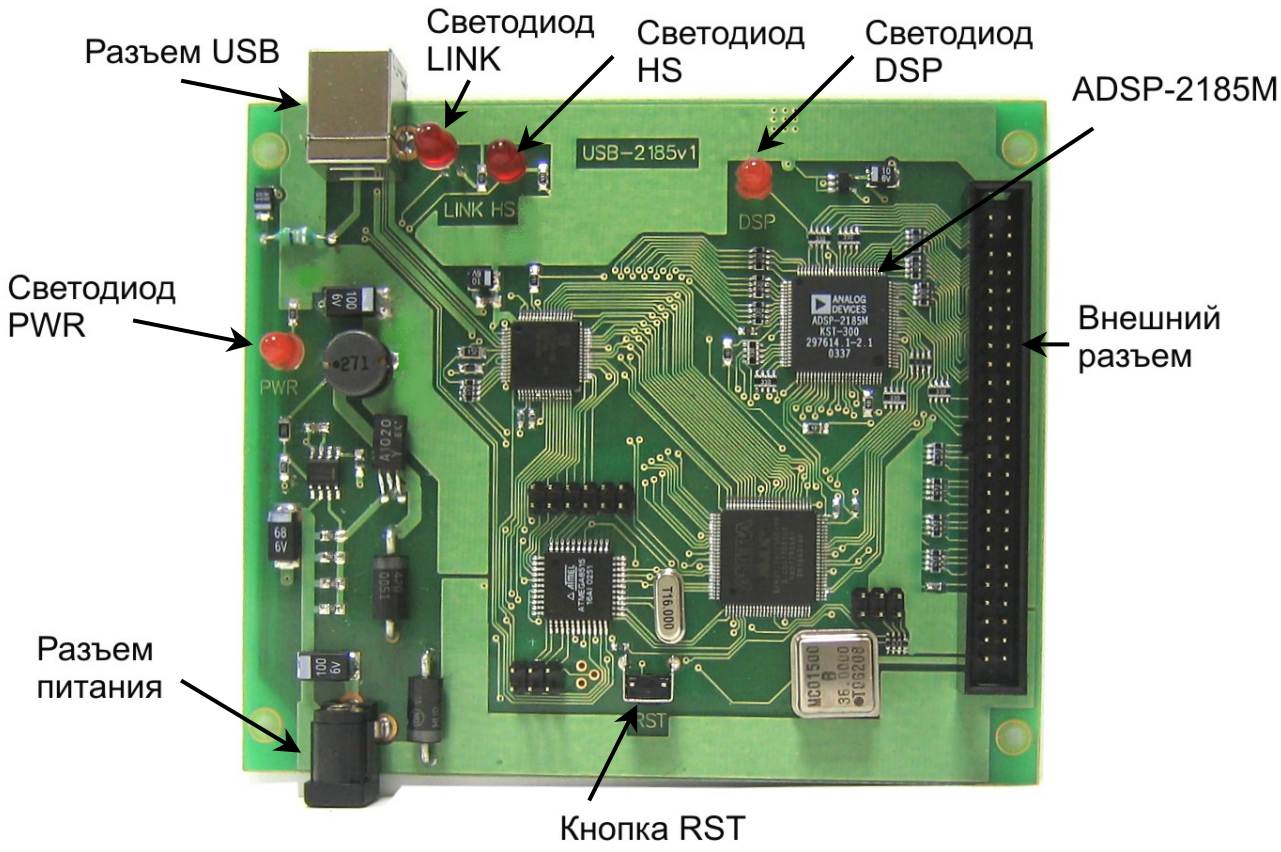


Рис. 2. Внешний вид изделия USB2185.

Светодиод PWR – загорается при подаче питания от шины USB или от внешнего источника.

Светодиод LINK – загорается при подключении изделия к шине USB после успешной нумерации устройства. Сигнализирует о том, что USB-порт ПК правильно распознал изделие USB2185.

Светодиод HS (High Speed) – горит, если USB-порт ПК, к которому подключено изделие, поддерживает стандарт USB 2.0 (High Speed USB). Если светодиод HS при подключении устройства к ПК на короткое время загорается, а затем гаснет – значит, USB-порт ПК поддерживает стандарт USB 1.1. (Full Speed USB), и скоростные возможности изделия USB2185 при работе с таким портом будут ограничены (например, максимальная скорость передачи данных из DSP в ПК падает до 900 Кбайт/с).

Светодиод DSP – соединен с выводом PF2 DSP. Загорается при PF2 = 0. Служит для облегчения отладки пользователем своего драйвера DSP. Например, пользователь может отслеживать, заходит ли DSP в обработчик прерывания, включая в этом обработчике светодиод.

Кнопка RST – общий сброс. При нажатии на эту кнопку происходит сброс интерфейсной части, сброс DSP и реенумирация изделия (устройство отключается от шины USB на короткое время и снова подключается к ней).

Разъем USB – тип Б. Стандартный разъем для подключения устройства к ПК по шине USB кабелем типа А-Б (кабель входит в комплект поставки изделия USB2185).

Разъем питания – предназначен для подачи внешнего питания на изделие USB2185 от внешнего источника питания +5В +/- 10%. Низкая потребляемая мощность самого изделия USB2185 позволяет обойтись без внешнего источника. Внешний источник можно использовать при работе USB2185 совместно с различными пользовательскими узлами, если их потребление (совместно с USB2185) превышает допустимые стандартом USB 500 мА, а также при работе с ноутбуком, в целях экономии энергии аккумулятора ноутбука.

ВНИМАНИЕ!!! Производитель гарантирует нормальную работу изделия USB2185 только с источником питания, входящим в комплект поставки изделия! Изделие не подлежит гарантийному ремонту, если оно использовалось совместно с источником питания, не входящим в комплект поставки изделия USB2185.

1.4 Внешний разъем изделия USB2185

На внешний разъем изделия USB2185 кроме всех выводов DSP (за исключением линий, задействованных для обмена данными с ПК) выведены также питание 5В, питание 3.3В, опорная частота 36МГц - это позволяет пользователю создавать самые разнообразные устройства на базе изделия USB2185, подключая к нему различные модули расширения.

Тип внешнего разъема – ВН-50.

Все выводы разъема соединены с выводами DSP через последовательно включенные сопротивления 33 Ом. Все выводы DSP, соединенные с разъемом (за исключением выводов PF1, PF3) подтянуты к питанию 3.3 В через сопротивления 4.7 кОм. Выводы PF1, PF3 подтянуты к земле через сопротивления 4.7 кОм.

Ниже представлена таблица соединения контактов внешнего разъема. Подробнее о назначении цепей см. документацию на процессор ADSP-2185M, входящую в комплект документации на изделие USB2185.

Таблица 1. Контакты внешнего разъема.

N вывода внешнего разъема	Название цепи	N вывода DSP	Назначение
1	A0	97	Адресный выход
2	PF0	94	Программируемый вход/выход
3	PF1	93	Программируемый вход/выход
4	PF2	89	Программируемый вход/выход
5	PF3	88	Программируемый вход/выход
6	FL0	87	Выходной флаг
7	FL1	86	Выходной флаг
8	FL2	85	Выходной флаг
9	DATA23	84	Линия данных
10	DATA22	83	Линия данных
11	DATA21	82	Линия данных

N вывода внешнего разъема	Название цепи	N вывода DSP	Назначение
12	DATA20	81	Линия данных
13	DATA19	79	Линия данных
14	DATA18	78	Линия данных
15	DATA17	77	Линия данных
16	DATA16	76	Линия данных
17	DATA15	75	Линия данных
18	DATA14	74	Линия данных
19	DATA13	73	Линия данных
20	DATA12	72	Линия данных
21	DATA11	70	Линия данных
22	DATA10	69	Линия данных
23	DATA9	68	Линия данных
24	DATA8	65	Линия данных
25	SCLK1	42	Клоки сериального порта 1
26	DR1	40	Входные данные сериального порта 1
27	RFS1	39	Входной фрейм сериального порта 1
28	TFS1	38	Выходной фрейм сериального порта 1
29	DT1	37	Выходные данные сериального порта 1
30	SCLK0	35	Клоки сериального порта 0
31	DR0	34	Входные данные сериального порта 0
32	RFS0	33	Входной фрейм сериального порта 0
33	TFS0	32	Выходной фрейм сериального порта 0
34	DT0	31	Выходные данные сериального порта 0
35	-IRQE (PF4)	26	Прерывание (программируемый вход/выход)
36	-CMS	25	Выход выбора Комбинированной Памяти
37	-IOMS	24	Выход выбора пространства IO
38	-DMS	22	Выход выбора Памяти Данных
39	-BMS	21	Выход выбора Байтовой Памяти
40	-RD	20	Строб чтения
41	-WR	19	Строб записи
42	36MHZ		Опорная частота – меандр 36МГц, 3.3 В.
43,44	VCC		Питание 3.3 В.
45,46	5V		Питание 5 В.
47,48,49,50	GND		Земля

ВНИМАНИЕ! При подключении пользовательских модулей расширения к изделию USB2185 следует учитывать, что для корректной работы изделия необходимо, чтобы в начале работы (до загрузки драйвера DSP в DSP) на программируемых входах/выходах PF0 – PF3 были следующие логические уровни: PF0, PF2 = 1; PF1, PF3 = 0. Поэтому на пользовательском модуле расширения эти контакты должны быть (до загрузки DSP) либо в третьем состоянии (входы), либо не задействованы, либо на них должны быть поданы соответствующие логические уровни. После загрузки DSP состояние этих выводов может быть произвольным.

2 Подключение изделия USB2185 к ПК

Сама процедура аппаратного подключения изделию USB2185 к компьютеру достаточно тривиальна: необходимо просто соединить **Разъем USB** изделия с любым свободным USB-портом компьютера при помощи кабеля, входящего в комплект штатной поставки. При этом подразумевается, что на компьютере уже установлена операционная система, способная корректно поддерживать функционирование шины USB: *Windows2000\XP*. Причем спецификацией USB предусматривается как "горячее" подключение или отключение устройств к/от шины USB (при работающем ПК), так и включение компьютера с уже подключенными устройствами USB.

Если Вы собираетесь использовать внешний источник питания, то для корректной работы нужно сначала подать питание с внешнего источника на изделие USB2185, а потом подключать изделие к ПК.

Шина USB предоставляет пользователям реальную возможность работать с периферийными устройствами в режиме Plug & Play. Это означает, что стандартом USB предусмотрено подключение устройства к работающему компьютеру, автоматическое его распознавание немедленно после подключения и последующая загрузка операционной системой соответствующих данному устройству драйверов.

При самом первом подсоединении изделия USB2185 к компьютеру (с помощью прилагаемого стандартного кабеля USB) операционная система должна запросить файлы драйвера для впервые подключаемого устройства. Тогда ей необходимо указать *inf*-файл с CD-ROM: `\DRV\RtecUsb.inf`. При этом операционная система сама скопирует файл драйвера в нужное ей место и сделает необходимые записи в своём реестре. После чего операционная система должна произвести так называемую операцию *нумерации* (enumeration, ‘переписи’), т.е. проинициализировать подключенное устройство. Такая процедура нумерации устройств, подключенных к шине USB, осуществляется динамически по мере их подключения или отключения без какого-либо вмешательства пользователя или клиентского программного обеспечения. По окончании нумерации должен загореться светодиод LINK. Это будет говорить о том, что подключенное устройство корректно опознано операционной системой и полностью готово к дальнейшей работе. Дополнительно проконтролировать правильность распознавания операционной системой подключенного модуля можно в *“Device Manager”* (*“Диспетчер устройств”*). Там должен появиться раздел *“R-Technology Devices”*, а в этом разделе должно появиться устройство *“RT USB-2185 Unit (USB 2.0)”*, как это, например, отображено на рисунке ниже:

При дальнейшей работе с изделием USB2185 операционная система уже будет знать, где находятся драйвера для данного типа устройства, и будет подгружать их автоматически по мере необходимости.

3 Руководство программиста

3.1 Введение

Основное назначение изделия USB2185 – служить базовым модулем для пользовательских устройств ввода/вывода и обработки информации. Структура изделия USB2185 позволяет создавать самые разнообразные устройства путем подключения к Внешнему разъему различных пользовательских модулей расширения.

Программирование устройств, созданных на базе изделия USB2185, состоит из двух этапов:

1) программирование DSP, который управляет пользовательскими модулями расширения - т.е. написание драйвера DSP;
2) программирование процесса передачи и обработки информации в ПК в среде *Windows2000/XP*, другими словами – управления изделием USB2185 со стороны ПК.

Возможности современного мощного цифрового сигнального процессора ADSP-2185M позволяют не только осуществлять управление пользовательскими модулями расширения и внешними устройствами в режиме жесткого реального времени, но и вести первичную обработку информации. Ресурсы DSP не расходуются на управление обменом данными с ПК и полностью высвобождаются для задач пользователя. Тем не менее, для упрощения процесса программирования DSP, необходимо придерживаться общих рекомендаций, изложенных в главе Рекомендуемая структура драйвера DSP.

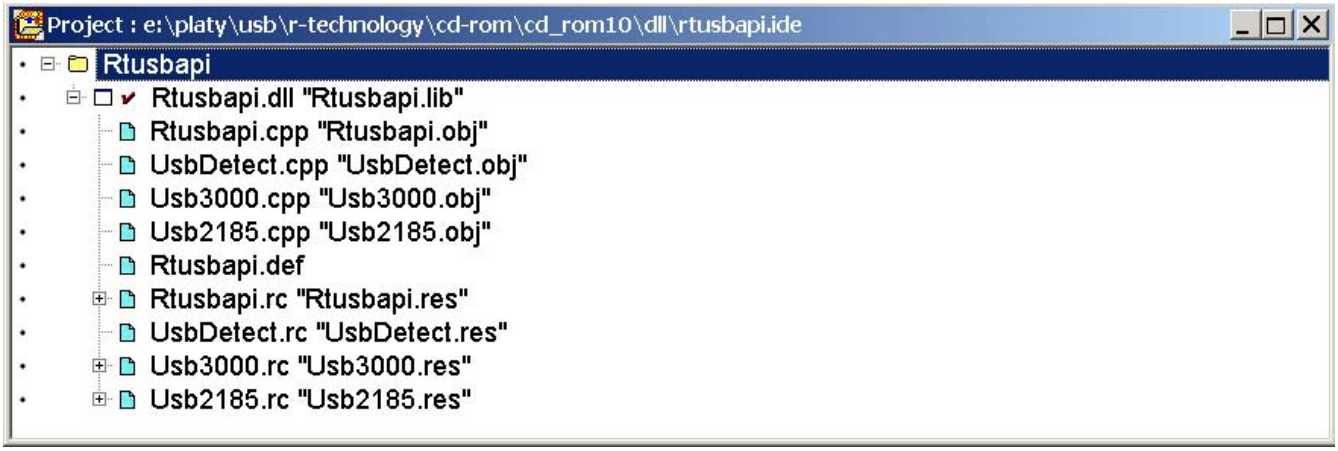
В комплект поставки изделия USB2185 входит штатная библиотека функций, облегчающая пользователю второй этап программирования изделия – управление модулем USB2185 со стороны ПК. Подробнее о работе со штатной библиотекой см главы Общий подход к работе со штатной библиотекой и Описание штатной библиотеки.

3.2 Общий подход к работе со штатной библиотекой

Настоящий раздел описания предназначен для программистов, собирающихся писать свои собственные программы в среде *Windows2000/XP* для работы с изделиями *USB2185*. В качестве базового высокоуровневого языка при написании штатного программного обеспечения нами был выбран язык C++ (а конкретнее, диалект **Borland C++ 5.02**), поскольку он является одним из самых доступных, а также достаточно широко распространенных и применяемых языков. В комплект штатной поставки для изделия *USB2185* входит драйвер устройства, готовая библиотека штатных подпрограмм, в виде динамически подключаемой библиотеки (DLL), и ряд законченных примеров программирования. В библиотеку мы попытались включить множество самых необходимых функций для облегчения пользователю процедуры написания собственных программ по управлению модулем *USB2185*. Данная библиотека позволяет Вам использовать практически все возможности модуля, не вдаваясь в тонкости их низкоуровневого программирования. Для тех, кто сам будет писать софт для данного модуля, наша библиотека может быть использована в качестве законченного и отлаженного примера, на основе которого Вы можете реализовать свои собственные алгоритмы.

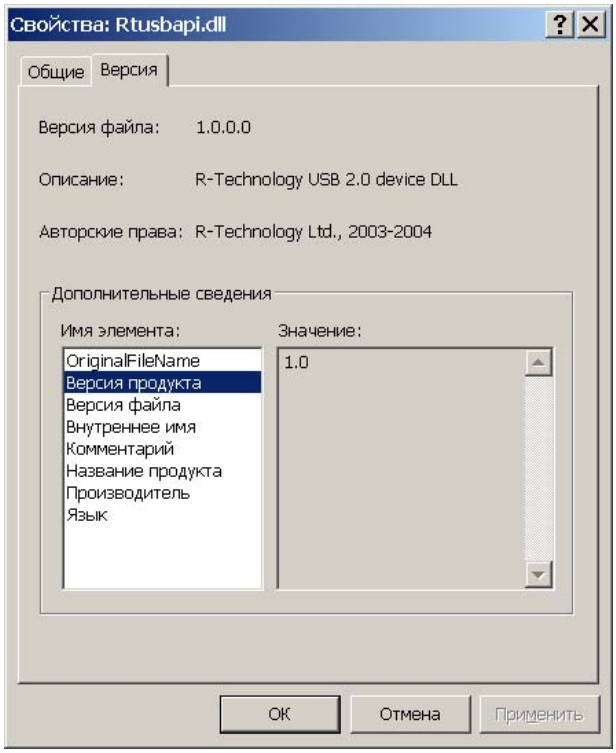
Подробное описание всех функций штатной библиотеки см. в Описание штатной библиотеки.

Общий вид проекта DLL библиотеки в среде разработки **Borland C++ 5.02** представлен на рисунке ниже:



Сама библиотека содержит всего две экспортируемые функции, одна из которых, RtCreateInstance(), возвращает указатель на интерфейс модуля *USB2185*. В дальнейшем, используя этот указатель, можно осуществлять доступ ко всем интерфейсным функциям DLL библиотеки (см. исходные тексты примеров). **!!!Внимание!!!** Все интерфейсные функции (кроме ReadData() и WriteData()), строго говоря, не обеспечивают “потокобезопасную” работу DLL библиотеки. Поэтому, во избежание недоразумений, в многопоточных приложениях пользователь должен сам организовывать, если необходимо, корректную синхронизацию вызовов функций в различных потоках (используя, например, критические участки, мьютексы и т.д.).

В сам файл библиотеки Rtusbapi.dll включена информация о текущей версии DLL. Для получения в Вашем приложении сведений о данной версии можно использовать вторую из экспортируемых функций из штатной библиотеки: RtGetDllVersion(). Кроме того, оперативно выявить текущую версию библиотеки можно, используя штатные возможности *Windows*. Например, в *Windows Explorer* щелкните правой кнопкой мышки над файлом DLL библиотеки Rtusbapi.dll. Во всплывшем меню следует выбрать опцию ‘*Properties*’, после чего на появившейся панели выбрать закладку ‘*Version*’. На этой закладке в строчке ‘*File version*’ можно прочитать номер версии DLL библиотеки (две старшие цифры):



Исходные тексты самой DLL библиотеки Вы можете найти на нашем CD–ROM'е в директории \DLL.

Для получения возможности вызова интерфейсных функций в Вашем проекте на **Borland C++** Вам необходимо следующее:

- создать файл проектов (например, для **Borland C++ 5.02**, `test.ide`);
- добавить в него файл `RTUSBAPI.LIB;`
- создать и добавить в проект Ваш файл с будущей программой (например, `test.cpp`);
- включить в начало вашего файла заголовочный файл `#include "RTUSBAPI.H"`, содержащий описание интерфейса модуля *USB2185*;
- с помощью функции RtGetDllVersion() желательно сравнить версию используемой DLL библиотеки с версией текущего программного обеспечения;
- вызвать функцию RtCreateInstance() для получения указателя на интерфейс модуля;

Теперь Вы можете писать свою программу и в любом месте, используя полученный указатель, вызывать соответствующие интерфейсные функции из штатной DLL библиотеки Rtusbapi.dll.

Приведем пример создания "скелета" типичной программы для управления изделием USB2185.

Итак, в самом начале мы получаем указатель на интерфейс модуля, вызвав функцию RtCreateInstance().

После этого, используя уже полученный указатель на интерфейс модуля, следует проинициализировать доступ к виртуальному слоту, к которому подключён модуль, применяя для этого интерфейсную функцию OpenDevice(). Если ошибки нет, то, в общем случае, какое-то устройство подключено к выбранному виртуальному слоту, и можно переходить к этапу его идентификации.

С помощью интерфейсной функции GetModuleName() нужно проверить название устройства, подключенного к выбранному виртуальному слоту, т.к. к нему, в принципе, может быть присоединено какое-нибудь другое, помимо *USB2185*, изделие, рассчитанное на работу с USB шиной. Если все в порядке и подключенное устройство – это изделие *USB2185*, то можно переходить к следующей стадии работы – загрузке драйвера DSP:

Для этого можно воспользоваться интерфейсной функцией LOAD_DSP().

Если в качестве параметра этой функции задать NULL, то в DSP начнет загружаться штатный демонстрационный драйвер DSP, который хранится в теле штатной библиотеки в виде соответствующего ресурса. Подробнее об этом драйвере см. главу Рекомендуемая структура драйвера DSP . Исходный текст демонстрационного драйвера можно посмотреть в директории \DSP.

В случае успешного выполнения функции LOAD_DSP(), нужно проверить работоспособность загруженного драйвера с помощью интерфейсной функции MODULE_TEST(). Если и эта функция выполнена без ошибки, то это означает, что драйвер DSP успешно загружен и модуль полностью готов к работе.

Далее можно реализовывать любые необходимые алгоритмы работы с изделием USB2185.

В качестве иллюстрации приведем исходный текст очень простой консольной программы для работы с модулем *USB2185*:

```
#include <stdlib.h>
#include <stdio.h>
#include "Rtusbapi.h"       // заголовочный файл штатной библиотеки


//using namespace RTUSB2185;
IRTUSB2185 *pUSB2185;       // указатель на интерфейс модуля
RTUSB2185::FLASH fi;        // структура с информацией из ППЗУ модуля
RTUSB2185::DSP_INFO di;     // структура с информацией о драйвере DSP
char ModuleName[10];        // название модуля
char ModuleSerialNumber[9]; // серийный номер модуля
int main(void)
{
    // проверим версию DLL библиотеки
    if(RtGetDllVersion() != CURRENT_VERSION_RTUSBAPI)
    {
      printf("Неправильная версия Dll!");
      return 1;
    }
    // получим указатель на интерфейс модуля
    pUSB2185 = static_cast<IRTUSB2185 *>(CreateInstance("USB2185"));
    if(pUSB2185 == NULL)
    {
      printf("Не могу получить указатель на интерфейс");
      return 1;                 //выйдем из программы с ошибкой
    }
    // попробуем обнаружить какой-нибудь модуль
    // в нулевом виртуальном слоте
    if(!pUSB2185->OpenDevice(0))
    {
      printf("Не могу получить доступ к модулю!");
      return 1;                 //выйдем из программы с ошибкой
    }
    // прочитаем название модуля в нулевом виртуальном слоте
    if(!pUSB2185->GetModuleName(ModuleName))
    {
      printf("Не могу прочитать название модуля!\n");
      return 1;                 //выйдем из программы с ошибкой
    }
    // проверим: этот модуль - ' USB2185'?
    if(strcmp(ModuleName, "USB2185"))
    {
      printf(" В нулевом виртуальном слоте не 'USB2185'\n");
      return 1;                 //выйдем из программы с ошибкой
    }
```

```
    // прочитаем серийный номер модуля
    if(!pUSB2185->GetModuleSerialNumber(ModuleSerialNumber))
    {
      printf("Не выполнена функция GetModuleSerialNumber()\n");
      return 1;                 //выйдем из программы с ошибкой
      }

    // теперь можно попробовать загрузить из соответствующего ресурса
    // библиотеки Rtusbapi.dll код универсального драйвера
    if(!pUSB2185->LOAD_DSP())
    {
      printf("Не выполнена функция LOAD_DSP()!\n");
      return 1;                 //выйдем из программы с ошибкой
    }

    // проверим работоспособность загруженного BIOS
    if(!pUSB2185->MODULE_TEST())
    {
      printf("Не выполнена функция MODULE_TEST()!\n");
      return 1;                 //выйдем из программы с ошибкой
    }

    // получим версию загруженного BIOSа
    if(!pUSB2185->GET_DSP_INFO(&di))
    {
      printf("Не выполнена функция GET_DSP_INFO ()!\n");
      return 1;                 //выйдем из программы с ошибкой
    }

    // попробуем прочитать информацию, хранящуюся в ППЗУ модуля
    if(!pUSB2185->GET_FLASH(&fi))
    {
      printf("Не выполнена функция GET_FLASH ()!\n");
      return 1;                 //выйдем из программы с ошибкой
    }

    printf("Модуль USB2185 (серийный номер %s) полностью готов к\
                                работе!\n", ModuleSerialNumber);

    // далее можно располагать функции для непосредственного
    // управления модулем!

    . . . . . .
    // завершим работу с модулем
    if(!pUSB2185->ReleaseDevice())
    {
      printf("Не выполнена функция ReleaseDevice()!");
      return 1;                 //выйдем из программы с ошибкой
    }

    // выйдем из программы
    return 0;
}
```

3.3 Рекомендуемая структура драйвера DSP

Пользователь может создавать самые разнообразные устройства на базе изделия USB2185, подключая к нему различные модули расширения. Т.о. пользователю предоставляется полная свобода творчества в процессе разработки и последующего программирования DSP с целью надлежащего управления своими модулями расширения. Однако библиотека Rtusbapi.dll накладывает свои требования на используемый драйвер DSP, а именно:

1. Все переменные драйвера должны располагаться в памяти программ DSP непосредственно после таблицы прерываний, т.е. начиная с адреса определяемого константой VariableBaseAddress (в штатном драйвере это 0x30) и вплоть до адреса задаваемого константой ProgaramBaseAddress (0x400).
2. Собственно код драйвера (т.е. его инструкции) располагается с адреса определяемого константой ProgaramBaseAddress (в штатном драйвере – 0x400).
3. В драйвере DSP необходимо предусмотреть следующие переменные: ModuleReady, TestVar1, TestVar2, TestIntrVar, Command. Библиотека Rtusbapi.dll знает об их существовании и использует в своей работе. Ниже в Таблице 2 приводятся *предопределенные* адреса переменных штатного драйвера DSP, расположенных в памяти **программ** DSP, и их краткие описания. Собственно, сами переменные драйвера являются $16^{ти}$ битными и располагаются в старших $16^{ти}$ битах $24^{х}$ битного слова памяти программ DSP (при этом младшие 8 из этих $24^{х}$ бит не используются). Программист может напрямую обращаться к переменным штатного драйвера, чтобы при необходимости считать и/или изменить их содержимое с помощью интерфейсных функций GET_VAR_WORD() и PUT_VAR_WORD().

Таблица 2. Переменные штатного драйвера DSP.

Название переменной	Адрес Hex	Назначение переменной
D_PROGRAM_BASE_ ADDRESS	0x30	Адрес в памяти программ откуда начинается собственно код инструкций драйвера DSP.
D_TARGET	0x31	Название устройства, для которого предназначен данный драйвер DSP (9 байтов + '\0'). Для модуля USB-2185 должна быть строчка "USB2185"
D_LABEL	0x36	Метка разработчика драйвера DSP (5 байтов + '\0'). Штатный драйвер DSP имеет метку "rtech"
D_VERSION	0x39	Версия драйвера DSP.
D_TEST_VAR1	0x3A	Тестовая переменная. После загрузки драйвера по этому адресу должно читаться число 0x5555.
D_TEST_VAR2	0x3B	Тестовая переменная. После загрузки драйвера по этому адресу должно читаться число 0xAAAA.
D_TEST_INTR_VAR	0x3C	Тестовая переменная для проверки командного прерывания.

D_MODULE_READY	0x3D	После загрузки драйвера сигнализирует о готовность модуля к дальнейшей работе
D_COMMAND	0x3E	При помощи этой переменной драйверу передается номер команды, которую он должен выполнить. Краткое описание номеров команд приведено ниже в Таблице 3.
D_READ_RATE	0x40	Переменная, задающая код частоты ввода данных в PC из модуля.
D_READ_ENABLED	0x41	Переменная, показывающая текущее состояние режима ввода данных (активное или нет).
D_READ_FIFO_BASE_ ADDRESS	0x42	Базовый адрес FIFO буфера ввода данных. Всегда устанавливается драйвером DSP равным **0x0**.
D_READ_FIFO_LENGTH	0x43	Длина FIFO буфера ввода. Всегда устанавливается драйвером DSP равным **0x3000**.
D_WRITE_RATE	0x50	Переменная, задающая код частоты вывода данных из PC в модуль.
D_WRITE_ENABLED	0x51	Переменная, показывающая текущее состояние режима вывода данных (активное или нет).
D_WRITE_FIFO_BASE_ ADDRESS	0x52	Базовый адрес FIFO буфера вывода данных. Всегда устанавливается драйвером DSP равным **0x3000**.
D_WRITE_FIFO_LENGTH	0x53	Длина FIFO буфера ввода. Всегда устанавливается драйвером DSP равным **0x0F80**.

4. Драйвер DSP должен работать по так называемому принципу команд. Т.е. управление работой драйвера должно происходить следующим образом. Сначала из PC в модуль по адресу переменной Command передаётся номер требуемой команды, которую драйвер должен выполнить. Затем инициируется командное прерывание *IRQ2* в DSP модуля. Обработчик этого прерывания считывает номер текущей команды из переменной Command и выполняет надлежащие действия в соответствие с этой командой. Ниже, в Таблице 3, представлены все команды, используемые в штатном драйвере DSP.

Таблица 3. Команды штатного драйвера DSP.

Название команды	Номер команды	Назначение	Используемые переменные
C_TEST	0	Проверка загрузки драйвера DSP и его работоспособности.	**D_TEST_INTR_VAR**
C_START_READ	1	Разрешение ввода данных в ПК из модуля.	———

C_STOP_READ	2	Останов ввода данных в ПК из модуля.	———
C_START_WRITE	3	Разрешение вывода данных из ПК в модуль.	———
C_STOP_WRITE	4	Останов вывода данных из ПК в модуль.	———

5. Двойной FIFO буфер ввода данных в ПК из модуля должен располагаться в памяти данных DSP, начиная с адреса DM(0x0). В штатном драйвере базовый адрес буфера ввода хранится в переменной ReadFifoBaseAddress. Длина этого буфера (в штатном драйвере переменная ReadFifoLength) должна быть строго 0x3000 (12288) слов. Эта информация жестко ‘забита’ в библиотеку Rtusbapi.dll.
6. Двойной FIFO буфер вывода данных из модуля в ПК должен располагаться в памяти данных DSP, начиная с адреса DM(0x3000). В штатном драйвере базовый адрес буфера ввода хранится в переменной WriteFifoBaseAddress. Длина этого буфера (в штатном драйвере переменная WriteFifoLength) должна быть строго 0xF80 (3968) слов. Эти информация жестко ‘забита’ в библиотеку Rtusbapi.dll.

Штатный драйвер DSP для модуля USB2185 удовлетворяет всем вышеуказанным требованиям. Исходные тексты данного драйвера достаточно подробно прокомментированы, и ими можно пользоваться в качестве законченного примера при написании собственных драйверов.

Исходные тексты драйвера DSP содержатся в следующих файлах:

Inttable.dsp - таблица прерываний драйвера DSP;
Usb2185.dsp - инициализация и основные функции драйвера DSP;
const.h - объявления констант;
vars.h - объявления переменных;
read.h - функции для реализации ввода данных из модуля в ПК;
write.h - функции для реализации вывода данных из ПК в модуль.

На Рис.3. приведена карта расположения штатного драйвера DSP в памяти программ и данных DSP.

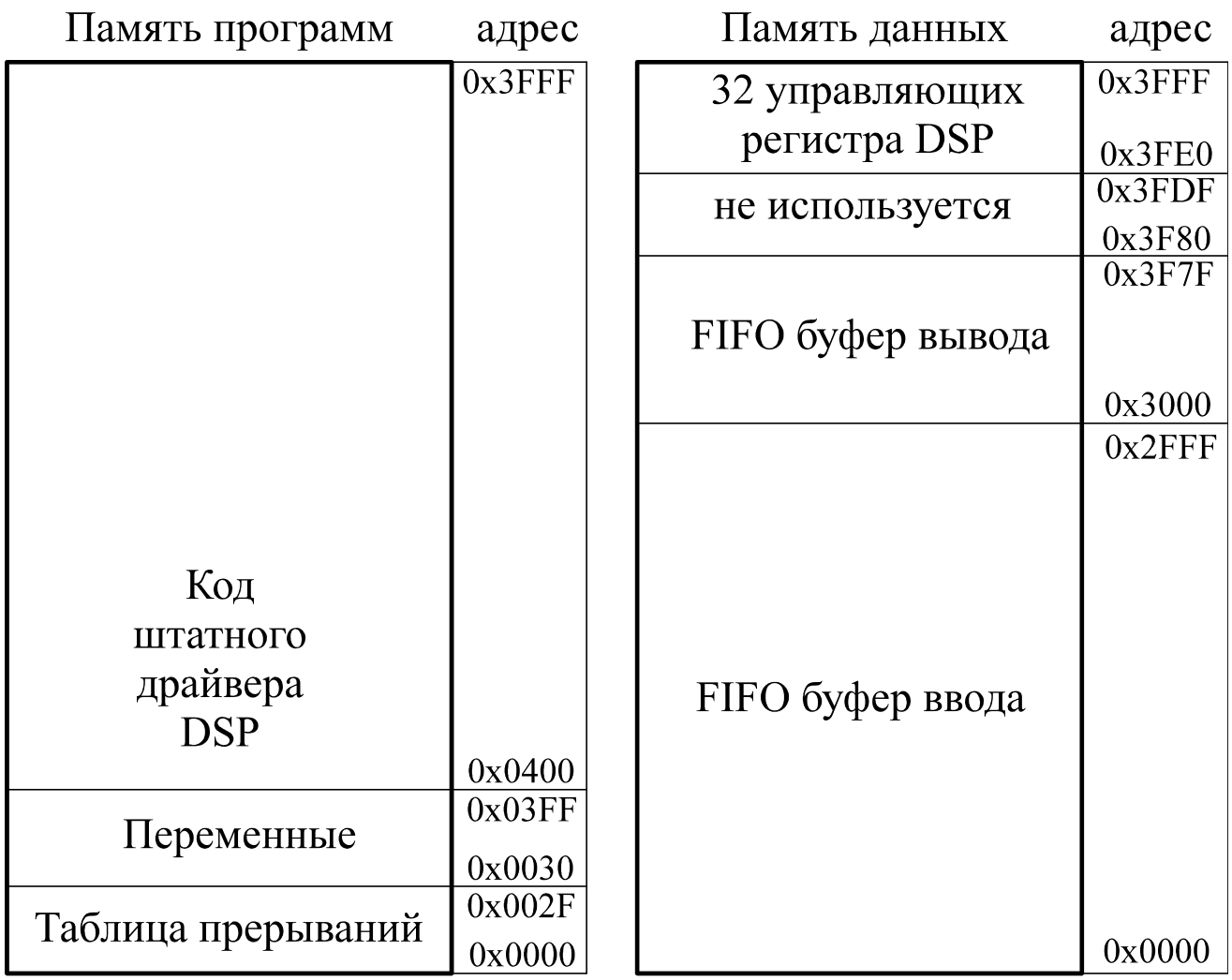


Рис. 3 Карта расположения штатного драйвера в памяти DSP

Как уже говорилось, в штатном драйвере DSP программно организовано два циклических FIFO буфера: для ввода данных в ПК из модуля и для вывода данных из модуля в ПК. Передача данных из FIFO буфера ввода в ПК производится порциями по ReadFifoLength/2 отсчетов по мере их поступления в буфер. Т.е. при поступлении из ПК команды на запуск ввода, драйвер DSP ожидает накопления данных в первой половине FIFO буфера ввода. После того как первая половина буфера полностью заполнится готовыми данными, инициируется соответствующее прерывание в интерфейсную часть модуля, которое говорит о том, что пора отправлять первую половину буфера в ПК (в тоже время не прекращается сбор данных во вторую половину FIFO буфера). После накопления данных во второй половине FIFO буфера опять дается прерывание на их передачу в ПК и продолжается сбор данных уже в первую половину. И так до бесконечности по циклу, пока не придет команда из ПК на останов работы ввода данных.

Все вышесказанное применимо и для алгоритма работы с FIFO буфером вывода данных:

Передача данных из ПК в циклический FIFO буфер вывода DSP производится порциями по WriteFifoLength/2 отсчетов по мере освобождения буфера. Т.е. при поступлении из ПК команды на запуск вывода, драйвер DSP начинает вычитывать данные из первой половины FIFO буфера вывода (освобождать буфер). После того как первая половина буфера полностью освободится, инициируется соответствующее прерывание в интерфейсную часть модуля, которое говорит о том, что пора заполнять первую половину буфера новыми данными из ПК (в тоже время не прекращается вычитывание данных из второй половины FIFO буфера). После освобождения второй половины FIFO буфера опять дается прерывание на передачу из ПК очередной порции

данных, и продолжается вычитывание данных уже из первой половины. И так до бесконечности по циклу, пока не придет команда из ПК на останов работы вывода данных.

Штатный драйвер DSP написан таким образом, что можно совершенно независимо управлять работой ввода и/или вывода информации в/из ПК.

Конечно же, пользователь может по тем или иным причинам проигнорировать вышеуказанные требования при создании своего драйвера DSP и полностью переписать весь софт, включая библиотеку и драйвер DSP. Тогда штатное программное обеспечение для изделия *USB2185* может послужить законченным, хорошо прокомментированным пособием для такого рода занятия.

3.4 Описание штатной библиотеки

3.4.1 Функции общего характера

3.4.1.1 Получение версии DLL библиотеки

Формат: **LPVOID** ***RtGetDllVersion****(void)*
Назначение: Данная функция является одной из двух экспортируемых из штатной DLL функцией и возвращает версию используемой DLL библиотеки. Рекомендованную последовательность вызовов интерфейсных функций см. в Общий подход к работе со штатной библиотекой
Передаваемые параметры: нет.
Возвращаемое значение: номер версии DLL библиотеки.

3.4.1.2 Получение указателя на интерфейс модуля

Формат: **LPVOID** ***RtCreateInstance****(const PCHAR const DeviceName)*
Назначение: Данная функция должна обязательно вызываться в начале каждой пользовательской программы, работающей с изделиями *USB2185*. Она является одной из двух экспортируемых из штатной DLL функцией и возвращает указатель на интерфейс для устройства с названием *DeviceName*. Все интерфейсные функции штатной DLL библиотеки вызываются именно через этот возвращаемый указатель. Рекомендованную последовательность вызовов интерфейсных функций см. в Общий подход к работе со штатной библиотекой
Передаваемые параметры: • *DeviceName* – строка с названием устройства (для данного изделия это – “USB2185”).
Возвращаемое значение: В случае успеха — указатель на интерфейс, иначе — **NULL**.

3.4.1.3 Функция завершения работы с модулем

Формат: **BOOL** ***ReleaseDevice****(void)*
Назначение: Данная интерфейсная функция реализует корректное высвобождение интерфейсного указателя, проинициализированного с помощью интерфейсной функции RtCreateInstance(). Используется для аккуратного завершения сеанса работы с модулем (если предварительно удачно выполнилась функция RtCreateInstance()). **!!!Внимание!!!** Данная функция **должна обязательно** вызываться в Вашем приложении перед непосредственным выходом из него во избежания утечки ресурсов компьютера. Рекомендованную последовательность вызовов интерфейсных функций см. в Общий подход к работе со штатной библиотекой
Передаваемые параметры: нет.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

3.4.1.4 Инициализация доступа к модулю

Формат: **BOOL** ***OpenDevice****(const WORD VirtualSlot)*

Назначение:

С программной точки зрения, подсоединенное к компьютеру изделие *USB2185* можно рассматривать как устройство, подключённое к некоему виртуальному слоту с сугубо индивидуальным номером. Основное назначение данной интерфейсной функции – определить, находится ли хоть какое-нибудь устройство в заданном виртуальном слоте. Если функция ***OpenDevice()*** успешно выполнилась для заданного виртуального слота, то далее следует убедиться, что к этому слоту подключен именно модуль *USB2185* (функция GetModuleName()). И если это так – можно переходить непосредственно к загрузке модуля и его последующему управлению с помощью соответствующих интерфейсных функций библиотеки Rtusbapi.dll. Рекомендованную последовательность вызовов интерфейсных функций см. в Общий подход к работе со штатной библиотекой

Передаваемые параметры:

- *VirtualSlot* – номер виртуального слота, к которому, как предполагается, подключен модуль *USB2185*.

Возвращаемое значение: *TRUE* – устройство находится в выбранном виртуальном слоте и можно попробовать определить его название с помощью интерфейсной функции GetModuleName() (см. ниже);
FALSE – никакого устройства в выбранном виртуальном слоте нет (может, стоит попробовать другой номер виртуального слота).

3.4.1.5 Освобождение виртуального слота

Формат: **BOOL** ***CloseDevice*** *(void)*

Назначение:

Данная интерфейсная функция прерывает, если необходимо, всякое взаимодействие с текущим виртуальным слотом, т.е. выполняет его освобождение (и связанных с ним ресурсов компьютера). После её применения всякий доступ к модулю *USB2185* становится невозможным. Для возобновления нормального доступа к устройству необходимо вновь воспользоваться интерфейсной функцией OpenDevice(). Таким образом, эта функция, по своей сути, противоположна интерфейсной функции OpenDevice(). Фактически данная функция используется в таких интерфейсных функциях как OpenDevice() и ReleaseDevice().

Передаваемые параметры: нет.

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE– функция не выполнена.

3.4.1.6 Получение названия модуля

Формат: **BOOL** ***GetModuleName*** (*PCHAR const ModuleName*)

Назначение:

Так как интерфейсная функция OpenDevice() определяет только наличие какого-нибудь устройства USB в выбранном виртуальном слоте, то, очевидно, необходимо каким-то образом это устройство идентифицировать. В принципе, к данному виртуальному слоту может быть подключено какое-нибудь другое (помимо модуля *USB2185*) устройство, рассчитанное на работу с шиной USB. Данная интерфейсная функция позволяет получить название подключенного к слоту модуля. Массив под название модуля *ModuleName* (не менее 6 символов плюс признак конца строки '\0', т.е. нулевой байт) должен быть заранее определен. Рекомендованную последовательность вызовов интерфейсных функций см. в Общий подход к работе со штатной библиотекой

Передаваемые параметры:

- *ModuleName* – возвращается строка, не менее 11 символов, с названием модуля (в нашем случае это должна быть строка "USB2185").

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE– функция не выполнена.

3.4.1.7 Получение серийного номера модуля

Формат: **BOOL** ***GetModuleSerialNumber*** *(PCHAR const SerialNumber)*

Назначение:

Данная интерфейсная функция в переменной *SerialNumber* возвращает строку с серийным номером модуля. Строка (9 байтов включая '\0') должна быть предварительно определена

Передаваемые параметры:

- *SerialNumber* – строка с серийным номером модуля.

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE– функция не выполнена.

3.4.1.8 Получение текущей скорости работы USB

Формат: **BOOL** ***GetUsbSpeed****(BYTE * const UsbSpeed)*

Назначение:

Данная интерфейсная функция в переменной *UsbSpeed* возвращает текущую скорость работы шины USB.

Передаваемые параметры:

- *UsbSpeed* – эта переменная принимать следующие значения:
  - ✓ 0 – модуль работает в режиме *Full Speed* (12 Мбит/с, USB 1.1)
  - ✓ 1 – модуль работает в режиме *High Speed* (480 Мбит/с, USB 2.0).

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE– функция не выполнена.

3.4.1.9 Получение версии BIOS интерфейсной части

Формат: **BOOL** ***GetAvrVersion****(PCHAR const AvrVersion)*
Назначение: Данная интерфейсная функция в строке *AvrVersion* возвращает номер версии BIOS, зашитого в микроконтроллер AVR, управляющий интерфейсной частью.
Передаваемые параметры: • *AvrVersion* – номер версии BIOS.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

3.4.1.10 Загрузка драйвера DSP

Формат: **BOOL** ***LOAD_DSP****(const PCHAR FileName = NULL)*
Назначение: Данная интерфейсная функция выполняет операцию загрузки драйвера DSP модуля. Файл *FileName* с кодом драйвера должен находиться в текущей директории Вашего приложения. В DLL библиотеке есть дополнительная возможность загружать штатный демонстрационный драйвер, содержимое которого хранится в самом теле библиотеки в виде соответствующего ресурса. Для этого достаточно параметр *FileName* задать в виде **NULL**. **NULL** является также значением по умолчанию для параметра *FileName*. Рекомендованную последовательность вызовов интерфейсных функций см. в Общий подход к работе со штатной библиотекой.
Передаваемые параметры: • *FileName* – строка с названием файла, содержащим код загружаемой управляющей программы. Например, для штатного драйвера DSP это строка `“Usb2185.rtd”.` Если данный параметр задан как **NULL**, то загрузка модуля будет осуществляться тем драйвером, который находится в виде ресурса в теле штатной DLL библиотеки.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

3.4.1.11 Проверка загрузки модуля

Формат: **BOOL** ***MODULE_TEST****(void)*

Назначение:

Данная интерфейсная функция проверяет правильность загрузки DSP и его работоспособность. **!!!Внимание!!!** Данная функция работает надлежащим образом **только** после выполнения интерфейсной функции LOAD_DSP().Рекомендованную последовательность вызовов интерфейсных функций см. в Общий подход к работе со штатной библиотекой.

Передаваемые параметры: нет

Возвращаемое значение: *TRUE* – драйвер DSP успешно загружен и функционирует надлежащим образом,
FALSE – произошла ошибка загрузки или функционирования драйвера DSP.

3.4.1.12 Получение версии загруженного драйвера DSP

Формат: **BOOL** ***GET_DSP_INFO****(RTUSB2185::DSP_INFO * const DspInfo)*

Назначение:

Данная интерфейсная функция позволяет считать краткую информацию о загруженном драйвере DSP. **!!!Внимание!!!** Данная функция работает надлежащим образом **только** после выполнения интерфейсных функций LOAD_DSP() и MODULE_TEST(). Рекомендованную последовательность вызовов интерфейсных функций см. в Общий подход к работе со штатной библиотекой.

Передаваемые параметры:

Структура *RTUSB2185::DSP_INFO* содержит краткую информацию о драйвере DSP. Она описана в файле `Rtusbapi.h` и представлена ниже:

```
struct DSP_INFO
{
    BYTE Target[10];        // название устройства “USB2185”
    BYTE Label[6];          // разработчик драйвера DSP "rtech"
    BYTE DspMajor;          // старший байт версии драйвера DSP
    BYTE DspMinor;          // младший байт версии драйвера DSP
};
```

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE– функция не выполнена.

3.4.1.13 Сброс DSP на модуле

Формат: **BOOL** ***RESET_DSP****(void)*
Назначение: Данная интерфейсная функция производит сброс (RESET) DSP модуля. Используется при перезагрузке драйвера DSP или для полной остановки работы DSP. Необходимо помнить, что после выполнения данной функции работа DSP устройства полностью останавливается, и для приведения его снова в рабочее состояние требуется перезагрузить драйвер DSP (например, с помощью функции LOAD_DSP()).
Передаваемые параметры: нет
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

3.4.1.14 Передача номера команды в драйвер DSP

Формат: **BOOL** ***SEND_COMMAND****(const WORD Command)*
Назначение: Данная интерфейсная функция передаёт в рекомендованную переменную D_COMMAND номер требуемой команды и вызывает командное прерывание *IRQ2* в DSP модуля. В ответ на это прерывание драйвер DSP выполняет действия, соответствующие номеру переданной команды. **!!!Внимание!!!** Данная функция работает надлежащим образом **только** после выполнения интерфейсных функции LOAD_DSP() и MODULE_TEST(). Рекомендованную последовательность вызовов интерфейсных функций см. в Общий подход к работе со штатной библиотекой.
Передаваемые параметры: • *Command* – номер команды, передаваемый в драйвер DSP.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

3.4.1.15 Получение описания ошибок выполнения функций

Формат: **int** ***GetLastErrorString*** (*LPTSTR const lpBuffer, const DWORD nSize*)

Назначение:

Если в процессе работы с DLL библиотекой Rtusbapi.dll какая-нибудь интерфейсная функция штатной библиотеки вернула ошибку, то **только** непосредственно после этого с помощью вызова данной интерфейсной функции можно получить краткое толкование произошедшего сбоя. **!!!Внимание!!!** Данная интерфейсная функция не выполняет классификацию ошибок для интерфейсных функций ReadData() и WriteData(). Т.к. эти функции фактически являются слепком со стандартных *Windows API* функций *ReadFile ()* и *WriteFile()*. Для выявления ошибок их выполнения следует пользоваться классификацией ошибок, присущей системе *Windows*.

Передаваемые параметры:

- *lpBuffer* – указатель на строку, в которой функция вернет описание ошибки;
- *nSize* – длина строки (рекомендуется 128 символов).

Возвращаемое значение: В случае успеха – кол-во скопированных в буфер символов; в противном случае – ноль.

3.4.2 Функции для доступа к памяти DSP модуля

Интерфейсные функции данного раздела обеспечивают доступ как к отдельным ячейкам, так и к целым массивам памяти DSP. Эта возможность позволяет программисту работать с модулем напрямую, непосредственно обращаясь к соответствующим ячейкам памяти. При этом для работы с этими функциями, в принципе, совсем не требуется загруженного в модуль драйвера DSP.

3.4.2.1 Чтение слова из памяти данных DSP

Формат: **BOOL** ***GET_DM_WORD****(WORD Address, SHORT * const Data)*

Назначение:
Данная функция считывает значение слова, находящееся по адресу *Address* в памяти данных DSP модуля.

Передаваемые параметры:
- *Address* – адрес ячейки в памяти данных DSP, значение которой необходимо считать;
- *Data* – указатель на переменную, куда функция положит считанное $16^{ти}$ битное слово.

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE– функция не выполнена.

3.4.2.2 Чтение слова из памяти программ DSP

Формат: **BOOL** ***GET_PM_WORD****(WORD Address, LONG * const Data)*

Назначение:
Данная функция считывает значение слова, находящееся по адресу *Address* в памяти программ DSP модуля.

Передаваемые параметры:
- *Address* – адрес ячейки в памяти программ DSP, значение которой необходимо считать;
- *Data* – указатель на переменную, куда функция положит считанное $24^{х}$ битное слово.

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE– функция не выполнена.

3.4.2.3 Запись слова в память данных DSP

Формат: **BOOL** ***PUT_DM_WORD****(WORD Address, SHORT Data)*

Назначение:
Данная функция записывает значение *Data* в ячейку с адресом *Address* в памяти данных DSP модуля.

Передаваемые параметры:
- *Address* – адрес ячейки в памяти данных DSP, куда необходимо записать значение *Data*;
- *Data* – значение записываемого $16^{ти}$ битного слова.

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE– функция не выполнена.

3.4.2.4 Запись слова в память программ DSP

Формат: **BOOL** ***PUT_PM_WORD****(WORD Address, LONG Data)*

Назначение:
Данная функция записывает значение *Data* в ячейку с адресом *Address* в памяти программ DSP модуля.

Передаваемые параметры:
- *Address* – адрес ячейки в памяти программ DSP, куда записывается значение *Data*;
- *Data* – значение записываемого 24^{x} битного слова.

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE– функция не выполнена.

3.4.2.5 Чтение массива слов из памяти данных DSP

Формат: **BOOL** ***GET_DM_ARRAY****(WORD BaseAddress, WORD NPoints, SHORT * const Buffer)*

Назначение:
Данная функция считывает массив слов длиной *NPoints* в буфер *Buffer*, начиная с адреса ячейки *BaseAddress* в памяти данных DSP модуля. Буфер *Buffer* надлежащей длины необходимо заранее определить.

Передаваемые параметры:
- *BaseAddress* – стартовый адрес в памяти данных DSP, начиная с которого производится чтение массива;
- *NPoints* – длина считываемого массива;
- *Buffer* – указатель на буфер, в который передаются считываемые значения.

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE– функция не выполнена.

3.4.2.6 Чтение массива слов из памяти программ DSP

Формат: **BOOL** ***GET_PM_ARRAY****(WORD BaseAddress, WORD NPoints, LONG * const Buffer)*

Назначение:
Данная функция считывает массив слов длиной *NPoints* в буфер *Buffer*, начиная с адреса ячейки *BaseAddress* в памяти программ DSP модуля. Буфер *Buffer* надлежащей длины необходимо заранее определить. При использовании этой функции следует помнить, что одно слово памяти программ DSP является 24^{x} битным.

Передаваемые параметры:
- *BaseAddress* – стартовый адрес в памяти программ DSP, начиная с которого производится чтение массива;
- *NPoints* – число считываемых 24^{x} битных слов;
- *Buffer* – указатель на буфер, в который передаются считываемые значения.

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE– функция не выполнена.

3.4.2.7 Запись массива слов в память данных DSP

Формат: **BOOL** ***PUT_DM_ARRAY****(WORD BaseAddress, WORD Npoints, SHORT * const Buffer)*

Назначение:

Данная функция записывает массив слов длиной *NPoints* из буфера *Buffer* в память данных DSP модуля, начиная с адреса *BaseAddress.*

Передаваемые параметры:

- *BaseAddress* – стартовый адрес в памяти данных DSP, начиная с которого производится запись массива;
- *NPoints* – длина записываемого массива;
- *Buffer* – указатель на буфер, из которого идет запись.

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE– функция не выполнена.

3.4.2.8 Запись массива слов в память программ DSP

Формат: **BOOL** ***PUT_PM_ARRAY****(WORD BaseAddress, WORD NPoints, LONG *const Buffer)*

Назначение:

Данная функция записывает массив слов длиной *NPoints* из буфера *Buffer* в память программ DSP модуля, начиная с адреса *BaseAddress.* При использовании этой функции следует помнить, что одно слово памяти программ DSP является 24^{x} битным.

Передаваемые параметры:

- *BaseAddress* – стартовый адрес в памяти программ DSP, начиная с которого производится запись массива;
- *NPoints* – число записываемых 24^{x} битных слов;
- *Buffer* – указатель на буфер, из которого идет запись.

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE– функция не выполнена.

3.4.2.9 Чтение переменной штатного драйвера DSP

Формат: **BOOL** ***GET_VAR_WORD****(WORD Address, SHORT * const Data)*
Назначение: Данная функция осуществляет считывание $16^{ти}$ битной переменной штатного драйвера, расположенной по адресу *Address* в $24^{х}$ битной памяти ***программ*** DSP модуля (см. Рекомендуемая структура драйвера DSP).
Передаваемые параметры: • *Address* – адрес ячейки переменной драйвера в памяти ***программ*** DSP, значение которой необходимо считать; • *Data* – указатель на переменную, куда функция положит считанное $16^{ти}$ битное значение переменной штатного драйвера.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

3.4.2.10 Запись переменной штатного драйвера DSP

Формат: **BOOL** ***PUT_VAR_WORD****(WORD Address, SHORT Data)*
Назначение: Данная функция осуществляет запись $16^{ти}$ битного значения *Data* в переменную штатного драйвера, расположенную по адресу *Address* в $24^{х}$ битной памяти ***программ*** DSP модуля (см. Рекомендуемая структура драйвера DSP).
Передаваемые параметры: • *Address* – адрес ячейки переменной драйвера в памяти ***программ*** DSP, куда необходимо записать значение *Data*; • *Data* – значение записываемого $16^{ти}$ битного значения переменной штатного драйвера.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

3.4.3 Функции ввода данных

Интерфейсные функции штатной DLL библиотеки позволяют реализовывать разнообразные алгоритмы ввода информации (*независимо* от состояния вывода). Изделие USB2185 с точки зрения состояния ввода данных, может находиться в двух режимах:

1. режим “покоя”;
2. потоковый (*перманентный*) ввод данных.

Функция START_READ() позволяет переводить модуль во второе из этих состояний, а STOP_READ() — в первое. При этом подразумевается, что драйвер DSP будет строго следовать рекомендованной процедуре ввода информации (см. Рекомендуемая структура драйвера DSP). При этом сами вводимые в PC данные модуль должен хранить в двойной циклический FIFO буфере ввода, который расположен в памяти данных DSP модуля. Для извлечения из модуля этих данных следует пользоваться функцией ReadData(). Пример корректного применения интерфейсных функций для ввода данных можно найти в директории `\Examples\ReadTest`.

3.4.3.1 Запуск ввода данных

Формат: **BOOL** ***START_READ****(void)*	
Назначение: Данная функция запускает DSP на перманентный ввод данных в ПК. Извлечение из модуля требуемой информации можно осуществлять с помощью интерфейсной функции ReadData.	
Передаваемые параметры: нет	
Возвращаемое значение:	*TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

3.4.3.2 Останов ввода данных

Формат: **BOOL** ***STOP_READ****(void)*	
Назначение: Данная функция останавливает процедуру ввода данных из модуля.	
Передаваемые параметры: нет	
Возвращаемое значение:	*TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

3.4.3.3 Получение данных из модуля

Формат: **BOOL** ***ReadData****(SHORT * const Buffer, DWORD * const NumberOfWordsToRead, LPDWORD *NumberOfBytesRead, LPOVERLAPPED Overlapped)*

Назначение:

Данная функция обеспечивает ***асинхронный режим*** получения очередных *NumberOfWordsToRead* данных из FIFO буфера ввода, расположенного в памяти данных DSP модуля. Величина *NumberOfWordsToRead* должна быть в диапазоне от 32 до (1024*1024) и должна быть кратной 32. В противном случае интерфейсная функция сама подкорректирует величину переменной *NumberOfWordsToRead* и по возвращении из ***ReadData()*** в ней будет находиться истинное значение количества полученных с модуля данных.

Поскольку данная функция осуществляет именно ***асинхронный режим*** приёма информации, ***ReadData()*** может вполне законно завершиться перед тем, как прекратится собственно сама операция чтения всего заказанного массива данных. При этом функция ***ReadData()*** должна вернуть *FALSE*, а *NumberOfBytesRead* может быть равно нулю. В этом случае следующим шагом необходимо вызвать *Windows API* функцию ***GetLastError()*** и убедиться, что она вернула ERROR_IO_PENDING. Это будет означать, что все в порядке и собственно операция чтения данных из модуля продолжает успешно выполняться. Окончание операции необходимо впоследствии отслеживать с помощью соответствующих *Windows API* функций (***WaitForSingleObject()*** или ***GetOverlappedResult()***), использующих обнаружение события, предварительно указанного в структуре *Overlapped*. Подробнее см. хелп на *Windows API* функцию ***ReadFile()***, а также исходные тексты прилагаемой к модулю законченной программы в директории `\Examples\ReadTest`.

!!!ВНИМАНИЕ!!! Для того чтобы эта функция корректно функционировала**, строго необходимо**, чтобы предварительно ввод данных был разрешён с помощью интерфейсной функции START_READ().

Передаваемые параметры:

- *Buffer* – указатель на массив, в который складываются принимаемые из модуля данные;
- *NumberOfWordsToRead* – количество данных (минимум – 32, максимум –1024*1024), которые необходимо получить из модуля и положить в *Buffer;*
- *NumberOfBytesRead* – количество реально полученных байтов;
- *Overlapped* – указатель на *OVERLAPPED* структуру (см. исходники примеров).

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE – функция не выполнена (см. замечания в ‘Назначении’ к этой функции).

3.4.4 Функции вывода данных

Интерфейсные функции штатной DLL библиотеки позволяют реализовывать разнообразные алгоритмы вывода информации (*независимо* от состояния ввода). Изделие USB2185 с точки зрения вывода данных, может находиться в двух состояниях:

1. режим “покоя”;
2. потоковая (*перманентная*) выдача данных.

Функция START_WRITE() позволяет переводить модуль во второе из этих состояний, а STOP_WRITE() — в первое. При этом подразумевается, что драйвер DSP будет строго следовать рекомендованной процедуре вывода информации (см. Рекомендуемая структура драйвера DSP). Для передачи данных в модуль уже в процессе работы (т.е. после запуска вывода) следует пользоваться функцией WriteData(). Пример корректного применения интерфейсных функций для вывода данных можно найти в директории `\Examples\WriteTest`.

3.4.4.1 Запуск вывода данных

Формат: **BOOL** ***START_WRITE****(void)*
Назначение: Данная функция запускает устройство на перманентный вывод данных из ПК. Саму передачу данных в модуль (уже после запуска) можно осуществлять с помощью интерфейсной функции `WriteData()`.
Передаваемые параметры: нет
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

3.4.4.2 Останов вывода данных

Формат: **BOOL** ***STOP_WRITE****(void)*
Назначение: Данная функция останавливает передачу данных в модуль.
Передаваемые параметры: нет
Возвращаемое значение: *TRUE* – функция выполнена; *FALSE*– функция не выполнена.

3.4.4.3 Передача данных на модуль

Формат: **BOOL** ***WriteData****(WORD *Buffer, DWORD *NumberOfWordsToWrite, LPDWORD *NumberOfBytesWritten, LPOVERLAPPED Overlapped)*

Назначение:

Данная функция обеспечивает ***асинхронный режим*** передачи *NumberOfWordsToWrite* данных из массива *Buffer* в модуль. Величина *NumberOfWordsToWrite* должна быть в диапазоне от 32 до (1024*1024) и должна быть кратна 32. В противном случае интерфейсная функция сама подкорректирует величину переменной *NumberOfWordsToWrite*, и по возвращении из ***WriteData()*** в ней будет находиться истинное значение количества передаваемых в модуль данных.

Поскольку данная функция осуществляет именно ***асинхронный режим*** передачи информации, ***WriteData()*** может вполне законно завершиться перед тем, как прекратится собственно сама операция записи в модуль всего заданного массива данных. При этом функция ***WriteData()*** может вернуть *FALSE*, а *NumberOfBytesWritten* может быть равно нулю. В этом случае следующим шагом необходимо вызвать *Windows API* функцию ***GetLastError()*** и убедиться, что она вернула ERROR_IO_PENDING. Это будет означать, что все в порядке и собственно операция записи данных в модуль продолжает успешно выполняться. Окончание операции необходимо впоследствии отслеживать с помощью соответствующих *Windows API* функций (***WaitForSingleObject()*** или ***GetOverlappedResult()***), использующих обнаружение события, предварительно указанного в структуре *Overlapped*. Подробнее см. хелп на *Windows API* функцию ***WriteFile()***, а также исходные тексты прилагаемых к модулю законченных программ в директории `\Example\WriteTest` и `\Example\ReadWrite`.

!!!ВНИМАНИЕ!!! Для того чтобы эта функция корректно функционировала, **строго необходимо**, чтобы предварительно работа ЦАП была разрешена с помощью интерфейсной функции START_WRITE().

Передаваемые параметры:

- *Buffer* – указатель на массив, из которого берутся передаваемые в модуль данные для FIFO буфера;
- *NumberOfWordsToWrite* – количество отсчетов (минимум – 32, максимум – 1024*1024), которые необходимо передать в модуль;
- *NumberOfBytesWritten* – количество реально переданных байтов;
- *Overlapped* – указатель на *OVERLAPPED* структуру (см. исходники примеров).

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE – функция не выполнена (см. замечания в ‘Назначение’ к этой функции).

3.4.5 Функции для работы с пользовательским ПЗУ (ППЗУ)

В состав изделия входит энергонезависимое пользовательское ПЗУ (ППЗУ) емкостью 256 байт, которое позволяет пользователю хранить необходимую информацию, например, калибровочные коэффициенты пользовательского АЦП, сериальный номер, название и т.д. Чтение/запись из/в ППЗУ осуществляется программно, через шину USB, и не требует дополнительных аппаратных средств.

3.4.5.1 Запись массива в ППЗУ

Формат: **BOOL** ***PUT_FLASH****(RTUSB2185::FLASH * const fi)*

Назначение:
Данная функция переписывает первые 256 байт из структуры *fi* в ППЗУ.

Передаваемые параметры:
- *fi* – указатель на структуру типа FLASH:

```
struct FLASH
{
    BYTE FlashBytes[256];        // массив байтов
};
```

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE – функция не выполнена

3.4.5.2 Чтение массива из ППЗУ

Формат: **BOOL** ***GET_FLASH****(RTUSB2185::FLASH * const fi)*

Назначение:
Функция считывает содержимое ППЗУ (256 байт) и записывает считанную информацию в структуру *fi*.

Передаваемые параметры:
- *fi* – указатель на структуру типа FLASH:

```
struct FLASH
{
    BYTE FlashBytes[256];        // массив байтов
};
```

Возвращаемое значение: *TRUE* – функция успешно выполнена;
FALSE – функция не выполнена

4 Примеры работы с изделием USB2185

В состав штатного бесплатного программного обеспечения входят примеры работы с изделием USB2185, демонстрирующие возможности устройства и основные приемы работы с ним.

В этих примерах работа с изделием осуществляется только посредством функций штатной библиотеки Rtusbapi.dll., входящей в комплект бесплатного программного обеспечения.

Примеры написаны в среде Borland C++ 5.02 и находятся в директории \Examples.

В тестах используется штатный демонстрационный драйвер DSP, который хранится в теле штатной библиотеки в виде соответствующего ресурса. Подробнее об этом драйвере см. главу Рекомендуемая структура драйвера DSP . Исходный текст демонстрационного драйвера можно посмотреть в директории \DSP.

Внимание! Перед запуском тестов необходимо вставить во Внешний разъем USB2185 тестовую заглушку, входящую в комплект поставки изделия.

4.1 Пример ReadTest

Данный пример демонстрирует скоростную передачу данных из изделия USB2185 в ПК.

Принцип работы теста таков: по команде с ПК драйвер DSP с заданной пользователем частотой начинает складывать данные во внутренний циклический FIFO буфер ввода DSP (базовый адрес этого буфера DM(0x0000), длина 0x3000). По мере заполнения буфера данные передаются в ПК. Программа в ПК, принимая данные, следит, есть ли разрывы в последовательности полученных данных, в случае сбоя выдавая сообщение об ошибке.

Итак, при запуске теста (ReadTest.exe) программа просит пользователя ввести требуемую скорость передачи данных из USB2185 в ПК (в кГц) – т.е. частоту, с которой драйвер DSP будет складывать данные во внутренний FIFO буфер ввода DSP.

Штатный универсальный BIOS интерфейсной части изделия позволяет реализовывать непрерывный ввод данных в ПК со скоростью до 12 x 10^6 байт/с (т.е. с частотой до 6 МГц). При этом в качестве задатчика частоты ввода используются тактовые импульсы (SCLK1) сериального порта 1 (SPORT1) DSP, частоту следования которых можно изменять программным образом в достаточно широких пределах. За величину частоты SCLK1 отвечает системный регистр DSP под названием SPORT1 SCLKDIV (расположен в памяти данных по адресу 0x3FF1).

Тактовые импульсы SCLK1 разрешаются по команде с ПК (метка Start_Read_cmd файла read.h). Эти импульсы через тестовую заглушку заведены на линию прерывания DSP, которое именуется IRQE. Прерывание инициируется по спадам импульсов SCLK1. Обработчик данного прерывания (метка MakeReadData в файле read.h) просто инкрементирует содержимое регитра AF и последовательно кладёт его в циклический FIFO буфер ввода. Таким образом, в циклический буфер ввода складывается натуральный ряд (0x0000, 0x0001, 0x0003, … …, 0xFFFF, 0x0000, 0x0001,…).

В теле основного цикла драйвера (метка MainLoop файла Usb2185.dsp) есть функция, которая отлеживает степень заполненности FIFO буфера ввода (метка CheckReadData файла Usb2185.dsp). Как только обнаружится, что первая половина FIFO буфера ввода (6144 слов) полностью заполнена, драйвер DSP сгенерит соответствующее прерывание в интерфейсную часть изделия USB2185, сигнализирующее о том, что пора передавать в ПК первую половину FIFO буфера. При этом драйвер будет продолжать по прерываниям от SCLK1 складывать новые числа во второую половину FIFO буфера ввода. Когда заполнится вторая половина буфера, драйвер DSP снова сгенерит соответствующее прерывание в интерфейсную часть и продолжит складывать данные в первую половину циклического буфера, в то время как данные из второй половины будут передаваться в ПК. И так до тех пор, пока из ПК не придет команда остановить сбор данных.

Программа в ПК после запуска сбора данных в DSP принимает данные из модуля посредством функции ReadData порциями по 1 МСлову (2 МБайта). При этом счетчик на экране монитора ПК отсчитывает количество полученных порций данных, т.е. количество успешно выполненных функций ReadData.

После получения очередной порции из 2 Мбайт данных программа в фоновом режиме проверяет полученные данные на наличие сбоев, т.е. следит, чтобы в полученном бесконечном натуральном ряде (0x0000, 0x0001, 0x0003, … …, 0xFFFF, 0x0000, 0x0001,…) не было разрывов. Штатный универсальный BIOS интерфейсной части изделия позволяет реализовывать непрерывный ввод данных в ПК со скоростью до 12 x 10^6 байт/с без сбоев. Для увеличения скорости до 14.4 x 10^6 байт/с необходимо прошить интерфейсную часть специализированным BIOS (см. Приложение А.).

4.2 Пример WriteTest

Данный пример демонстрирует скоростную передачу данных из ПК в изделие USB2185.

Принцип работы теста таков: ПК передает данные во внутренний циклический FIFO буфер вывода DSP (базовый адрес этого буфера DM(0x3000), длина 0x0F80). По команде с ПК драйвер DSP с заданной пользователем частотой начинает последовательно вычитывать данные из этого буфера (освобождать буфер). По мере освобождения буфера ПК заполняет его новыми порциями данных. Вычитывая данные, драйвер DSP отслеживает, есть ли разрывы в последовательности получаемых данных и в случае сбоя прекращает вычитывать данные, сигнализируя об ошибке выключением Светодиода DSP.

Итак, при запуске теста (WriteTest.exe) программа просит пользователя ввести требуемую скорость передачи данных из ПК в USB2185 (в кГц) – т.е. частоту, с которой драйвер DSP будет вычитывать данные из внутреннего FIFO буфера вывода DSP.

Штатный BIOS интерфейсной части изделия позволяет реализовывать непрерывный, без разрывов, вывод данных из ПК в модуль со скоростью до 3.1 x 10^6 байт/с (1.55 МГц). При этом в качестве задатчика частоты ввода используются тактовые импульсы (SCLK0) сериального порта 0 (SPORT0) DSP, частоту следования которых можно изменять программным образом в достаточно широких пределах. За величину частоты SCLK0 отвечает системный регистр DSP под названием SPORT0 SCLKDIV.

Тактовые импульсы SCLK0 разрешаются по команде с ПК (метка Start_Write_cmd файла write.h). Эти импульсы через тестовую заглушку заведены на линию прерывания DSP, которое именуется IRQ0. Прерывание инициируется по спадам импульсов SCLK0. Обработчик данного прерывания вызывает процедуру (метка CheckDacBuffer файла Usb2185.dsp), которая вычитывает очередное значение из FIFO буфера вывода и отслеживает наличие разрывов в полученных данных. В случае сбоя драйвер DSP прекращает вычитывать данные и сигнализирует об ошибке выключением Светодиода DSP.

Данные, полученные из ПК, представляют собой синусообразно меняющееся значение амплитудой 2000 кодов.

В теле основного цикла драйвера DSP есть функция (метка CheckWriteData файла Usb2185.dsp), которая отслеживает степень заполненности FIFO буфера вывода. Как только обнаружится, что первая половина буфера вычитана драйвером, DSP сгенерит соответствующее прерывание в интерфейсную часть, сигнализируя о том, что пора обновлять данные в первой половине буфера данными из ПК (т.е. пора передавать очередную порцию данных из ПК). При этом драйвер ДСП будет продолжать по прерываниям от SCLK0 вычитывать данные из второй половины FIFO буфера и проверять их на непрерывность. Когда освободится вторая половина буфера, драйвер DSP снова сгенерит соответствующее прерывание в интерфейсную часть и продолжит вычитывать данные из первой половины буфера в то время, как вторая половина будет заполняться данными из ПК. И так до тех пор, пока из ПК не придет команда остановить прием данных.

Программа в ПК после запуска приема данных в DSP передает данные в модуль посредством функции WriteData порциями по 1 МСлову (2 МБайта). При этом счетчик на экране монитора ПК отсчитывает количество переданных порций данных, т.е. количество успешно выполненных функций WriteData .

5 Приложения

Приложение А. Утилита прошивки BIOS изделия USB2185

В комплект бесплатного штатного программного обеспечения, поставляемого с изделием USB2185, входит утилита прошивки BIOS интерфейсной части. С помощью этой утилиты пользователь имеет возможность (в случае необходимости) самостоятельно заменить ("перепрошить") BIOS, например, в целях модернизации при появлении новой версии штатного BIOS.

Кроме того, в процессе эксплуатации изделия USB2185 у пользователя может возникнуть потребность заменить BIOS в случае, если штатный (универсальный) BIOS по каким либо причинам не подходит для выполнения пользовательской задачи. Например, пользователю необходимо осуществлять скоростную передачу данных в ПК со скоростью 14 Мбайт/с, и при этом у пользователя нет необходимости осуществлять скоростную передачу данных в противоположную сторону, т.е. из ПК. Универсальный BIOS (UniBIOS), зашитый в изделие на этапе производства, позволяет передавать данные в ПК со скоростью до 12 x 10^6 байт/с и из ПК со скоростью до 3.1 x 10^6 байт/с (см. Технические характеристики). В этом случае пользователь может прошить интерфейсную часть изделия альтернативным BIOS - ReadBIOS. Изделие USB2185, прошитое ReadBIOS, может передавать данные в ПК со скоростью до 14.4 x 10^6 байт/с, но не может осуществлять скоростную передачу данных в противоположном направлении. ReadBIOS также входит в комплект бесплатного программного обеспечения.

Утилита прошивки BIOS изделия USB2185 представляет собой один файл вида **RF_XvY_Y.EXE**, где:

X – буква, обозначающая тип BIOS: **"U"** – универсальный штатный BIOS, **"R"** – альтернативный BIOS для скоростного чтения данных с модуля (ReadBIOS).

Y_Y – версия BIOS.

Например, файл **RF_Uv1_2.EXE** – это файл с прошивкой универсального BIOS версии 1.2.

Эти файлы находятся на CD-ROM в директории Utils\RewriteFlash\.

Для того, чтобы перепрошить BIOS достаточно просто запустить соответствующий exe-файл. После того, как программа выведет на экран "REWRITE_FLASH() → OK", должен погаснуть светодиод LINK. Это говорит о том, что перезапись BIOS прошла успешно. Теперь необходимо отсоединить кабель USB от изделия USB2185. Окно программы закроется. На этом процедура замены BIOS завершена.

ВНИМАНИЕ!!! Не следует без необходимости использовать утилиту прошивки BIOS, т.к. при каком-либо сбое в процессе перезаписи BIOS (отсоединился кабель USB, повис компьютер и т.д.) работоспособность изделия может быть нарушена, и для восстановления BIOS потребуется доставить изделие производителю.

6 Оглавление