

ВНИМАНИЕ!!! Перед подключением модуля USB3000 к компьютеру и к источникам сигналов необходимо изучить [главу 2](#) описания!

Оглавление.

Оглавление.	1
1. Общая информация.....	3
1.1. Назначение модуля USB3000.	3
1.2. Возможности модуля USB3000.....	4
1.3. Технические характеристики модуля USB3000.	7
1.4. Комплект поставки модуля USB3000.....	11
2. Подключение модуля USB3000.....	12
2.1. Подготовка к работе	12
2.2. Подключения модуля USB3000 к компьютеру.....	14
2.3. Подключение к модулю USB3000 источников сигнала.....	15
2.3.1. Подключение источников сигнала к аналоговому разъему.	15
2.3.2. Подключение источников сигнала к цифровому разъему.....	17
3. Руководство программиста.....	18
3.1. Введение	18
3.2. Используемые термины и форматы данных	19
3.2.1. Используемые термины.....	19
3.2.2. Форматы данных	20
3.2.2.1. Формат слова данных АЦП	20
3.2.2.2. Формат слова данных ЦАП	20
3.2.2.3. Логический номер канала АЦП.....	21
3.3. Штатная библиотека.....	22
3.3.1. Общий подход к работе со штатной библиотекой	22
3.3.2. Описание структур штатной библиотеки.....	27
3.3.2.1. Структура RTUSB3000::INPUT_PARS.....	27
3.3.2.2. Структура RTUSB3000::OUTPUT_PARS.....	29
3.3.2.3. Структура RTUSB3000::FLASH.....	30
3.3.3. Описание функций штатной библиотеки	31
3.3.3.1. Функции общего характера.....	31
3.3.3.1.1. Получение версии DLL библиотеки	31
3.3.3.1.2. Получение указателя на интерфейс модуля.....	31
3.3.3.1.3. Завершение работы с модулем	31
3.3.3.1.4. Инициализация доступа к модулю.....	32
3.3.3.1.5. Освобождение виртуального слота.....	32
3.3.3.1.6. Получение названия модуля	33
3.3.3.1.7. Получение серийного номера модуля.....	33
3.3.3.1.8. Получение текущей скорости работы USB	33
3.3.3.1.9. Получение версии BIOS модуля.....	34
3.3.3.1.10. Загрузка драйвера DSP	34
3.3.3.1.11. Проверка загрузки модуля	35
3.3.3.1.12. Получение версии загруженного драйвера DSP	35
3.3.3.1.13. Сброс DSP модуля	36
3.3.3.1.14. Передача номера команды в драйвер DSP	36
3.3.3.1.15. Получение описания ошибок выполнения функций.....	37
3.3.3.2. Функции для доступа к памяти DSP модуля.....	38
3.3.3.2.1. Чтение слова из памяти данных DSP	38
3.3.3.2.2. Чтение слова из памяти программ DSP	38
3.3.3.2.3. Запись слова в память данных DSP.....	38
3.3.3.2.4. Запись слова в память программ DSP.....	39

3.3.3.2.5	Чтение массива слов из памяти данных DSP	39
3.3.3.2.6	Чтение массива слов из памяти программ DSP	39
3.3.3.2.7	Запись массива слов в память данных DSP	40
3.3.3.2.8	Запись массива слов в память программ DSP	40
3.3.3.2.9	Чтение переменной штатного драйвера DSP	41
3.3.3.2.10	Запись переменной штатного драйвера DSP	41
3.3.3.3.	Функции ввода данных	42
3.3.3.3.1	Запуск ввода данных	42
3.3.3.3.2	Останов ввода данных	42
3.3.3.3.3	Установка параметров ввода данных	43
3.3.3.3.4	Получение текущих параметров ввода данных	43
3.3.3.3.5	Получение данных из модуля	44
3.3.3.4.	Функции вывода данных	45
3.3.3.4.1	Запуск вывода данных	45
3.3.3.4.2	Останов вывода данных	45
3.3.3.4.3	Установка параметров вывода данных	46
3.3.3.4.4	Получение текущих параметров вывода данных	46
3.3.3.4.5	Передача данных в модуль	47
3.3.3.5.	Функции для работы с внешними цифровыми линиями	48
3.3.3.5.1	Разрешение выходных цифровых линий	48
3.3.3.5.2	Чтение внешних цифровых линий	48
3.3.3.5.3	Вывод на внешние цифровые линии	48
3.3.3.6.	Функции для работы с пользовательским ППЗУ	49
3.3.3.6.1	Разрешение записи в ППЗУ	49
3.3.3.6.2	Запись данных в ППЗУ	49
3.3.3.6.3	Чтение данных из ППЗУ	49
3.4.	Драйвер DSP	50

1. Общая информация.

1.1. Назначение модуля USB3000.

На [Рис. 1.](#) представлен внешний вид модуля USB3000.



Рис. 1.

Модуль USB3000 представляет собой универсальный многоканальный АЦП/ЦАП/логический анализатор. Модуль предназначен для построения систем ввода, вывода и обработки аналоговой и цифровой информации на базе IBM-совместимых компьютеров.

Особенностью модуля является наличие буферных повторителей на входах, что позволяет изделию сохранять требуемую точность на высоких частотах переключения канального мультиплексора.

В состав модуля входит восьмиканальный АЦП, двухканальный ЦАП, а так же логические входные и выходные линии.

Использование для связи с компьютером современной высокоскоростной шины USB 2.0 позволяет вести непрерывный дуплексный ввод/вывод данных при максимальной частоте дискретизации. Кроме того, интерфейс USB позволяет использовать модуль USB3000 в составе мобильных измерительных комплексов на базе notebook.

В состав модуля входит цифровой сигнальный процессор (DSP), что позволяет вести первичную ЦОС (Цифровую Обработку Сигнала) при вводе и выводе и обеспечивает гибкое задание алгоритма переключения каналов, опроса логических линий и синхронизации процессов сбора информации.

1.2. Возможности модуля USB3000.

Модуль USB3000 обладает следующими положительными качествами:

- Использование интерфейса USB значительно упрощает процесс подключения модуля USB3000 к компьютеру и обеспечивает возможность работы с модулем в режиме реального Plug&Play. Скоростные характеристики современной шины USB 2.0 позволяют вести непрерывный дуплексный ввод/вывод данных в режиме жесткого реального времени на максимальной скорости. Кроме того, интерфейс USB позволяет использовать модуль USB3000 в составе мобильных измерительных комплексов на базе ноутбука.
- Входные аналоговые каналы имеют внутреннюю буферизацию. Это полностью решает одну из основных проблем высокочастотных многоканальных систем сбора данных — проблему межканального прохождения при высоких частотах переключения каналов. Кроме того, внутренняя буферизация гарантирует отсутствие проникновения коммутационных шумов в линии связи с объектом.
- Входные аналоговые каналы модуля USB3000 – дифференциальные. Дифференциальное подключение источника сигнала снижает уровень синфазных помех. Помимо этого, дифференциальные входы позволяют подключать источники сигнала таким образом, чтобы токи сигнальных цепей не протекали через один общий провод, что повышает точность измерений.
- Буферизация выходов ЦАП позволяет подключать к модулю приемники сигналов с низким входным сопротивлением.
- В состав модуля входит цифровой сигнальный процессор (DSP), что обеспечивает возможность выполнения алгоритмов управления объектом в жестком реальном времени без использования ресурсов компьютера, а также позволяет вести первичную ЦОС (Цифровую Обработку Сигнала) при вводе и выводе.
- Несколько режимов внутренней и внешней синхронизации ввода и вывода данных позволяют реализовывать самые разнообразные режимы работы модуля USB3000 в составе систем сбора и обработки информации.
- В состав модуля входит энергонезависимое пользовательское ПЗУ (ППЗУ) емкостью 256 байт. Пользователь имеет возможность хранить в ППЗУ любую необходимую информацию, например, калибровочные коэффициенты АЦП и ЦАП, серийный номер, название и т.д. Чтение/запись из/в ППЗУ осуществляется программно, через шину USB, и не требует дополнительных аппаратных средств.
- При вводе данных с АЦП и выводе данных на ЦАП может производиться программная корректировка с использованием калибровочных коэффициентов, записываемых в ППЗУ модуля на этапе наладки производителем (либо с использованием пользовательских коэффициентов). Программная корректировка позволяет обойтись без подстроечных резисторов, что повышает надежность модуля и улучшает его шумовые характеристики.
- Питание модуля может осуществляться как непосредственно от шины USB, так и от внешнего источника питания, входящего в комплект поставки. Низкая потребляемая мощность модуля USB3000 позволяет обойтись без внешнего источника. Внешний источник можно использовать при работе с ноутбуком, в целях экономии энергии аккумулятора ноутбука.

Схемотехническое решение модуля и штатное программное обеспечение позволяют реализовывать следующие режимы работы:

1. Режим «АЦП»

В режиме «АЦП» модуль USB3000 осуществляет многоканальный ввод аналоговых сигналов с частотой преобразования АЦП до 3 МГц и с частотой переключения каналов до 3 МГц.

Данные, считываемые с АЦП модуля, представляют собой целые знаковые двухбайтовые числа от -8000 (соответствует напряжению -5В на входе канала) до +8000 (соответствует напряжению +5В на входе канала). При вводе данных в ПК имеется возможность программной корректировки получаемых с АЦП значений с использованием калибровочных коэффициентов, записываемых в ППЗУ модуля на этапе наладки производителем. Пользователь имеет возможность в случае необходимости использовать для программной корректировки собственные калибровочные коэффициенты.

Модуль имеет 8 дифференциальных буферизированных входов. Пользователь задает частоту работы АЦП и [управляющую таблицу](#) – массив номеров каналов, в соответствии с которой модуль будет осуществлять циклический сбор данных. Например, пользователь задал частоту АЦП 3 МГц и управляющую таблицу, содержащую номера: 0, 1, 2, 0, 8, 2. В этом случае модуль будет оцифровывать каналы 0, 1, 2, 0, 8, 2, 0, 1, 2, 0, 8, 2... и т.д., переключая их со скоростью 3 МГц. Если в таблице задан только один канал, то модуль будет оцифровывать только этот канал с заданной частотой.

Сбор данных может быть запущен как по команде с ПК, так и по внешнему сигналу – цифровому импульсу на линии ["SYN"](#) аналогового разъема модуля.

Полученные с АЦП данные могут непрерывно записываться в ОЗУ ПК в реальном времени. Полученные данные могут быть также записаны на жесткий диск ПК, при этом максимальная скорость непрерывной записи на диск определяется только производительностью ПК.

2. Режим «ЦАП»

В режиме «ЦАП» модуль USB3000 осуществляет двухканальный вывод аналоговых сигналов с частотой преобразования ЦАП до 100 кГц и с частотой переключения каналов до 100 кГц.

При выводе данных из ПК имеется возможность программной корректировки выдаваемых на ЦАП значений с использованием калибровочных коэффициентов, записываемых в ППЗУ модуля на этапе наладки производителем. Пользователь имеет возможность в случае необходимости использовать для программной корректировки собственные калибровочные коэффициенты.

Формат данных, передаваемых на ЦАП, описан в п. [Формат слова данных ЦАП](#). Номер канала ЦАП зашифрован непосредственно в самих данных, поэтому для работы с ЦАП пользователю необходимо задать только скорость работы ЦАП и подготовить в памяти ПК буфер с данными для передачи.

Данные могут непрерывно передаваться на ЦАП из ОЗУ ПК (или с жесткого диска) в реальном времени.

3. Режим «Логический анализатор»

В режиме «Логический анализатор» модуль USB3000 осуществляет ввод в ПК цифровых сигналов с 10 входных цифровых линий с частотой опроса до 6 МГц.

Данные поступают в ПК в виде двухбайтовых слов, младшие 10 разрядов которых соответствуют логическим состояниям на 10 входных цифровых линиях модуля («1» - соответствует высокому уровню на входе). Старшие 6 разрядов всегда равны «0».

С точки зрения пользователя этот режим аналогичен режиму «АЦП». Пользователь задает скорость опроса, а в управляющей таблице задает только один канал с битом [TTLIN](#) равном «1».

Сбор данных может быть запущен как по команде с ПК, так и по внешнему сигналу – цифровому импульсу на линии ["SYN"](#) аналогового разъема модуля.

Полученные с цифровых линий данные могут непрерывно записываться в ОЗУ ПК в реальном времени. Полученные данные могут быть также записаны на жесткий диск ПК, при этом максимальная скорость непрерывной записи на диск определяется только производительностью ПК.

Указанные режимы работы могут сочетаться следующим образом:

4. Режим «АЦП+ЦАП»

В этом режиме модуль USB3000 осуществляет полностью дуплексный ввод/вывод информации. Ввод данных с 8 входных каналов АЦП с частотой преобразования до 3 МГц происходит параллельно с выводом данных на 2-х канальный ЦАП с частотой преобразования до 100 кГц.

Дуплексный обмен информацией между ПК и модулем может осуществляться непрерывно в реальном времени.

5. Режим «Логический анализатор + ЦАП»

В этом режиме модуль USB3000 осуществляет полностью дуплексный ввод/вывод информации. Ввод данных с 10 входных цифровых линий с частотой опроса до 3 МГц происходит параллельно с выводом данных на 2-х канальный ЦАП с частотой преобразования до 100 кГц.

Дуплексный обмен информацией между ПК и модулем может осуществляться непрерывно в реальном времени.

Во всех описанных выше пяти режимах работы возможен асинхронный доступ со стороны ПК к 10 входным и 8 выходным цифровым линиям модуля. Например, модуль работает в режиме «АЦП» - собирает данные с 8 входных аналоговых каналов и передает в ПК. Не прерывая сбора данных с АЦП, пользователь имеет возможность с помощью функции [TTL IN](#) считать состояние 10 цифровых входов и с помощью функции [TTL OUT](#) установить 8 выходных цифровых линий в нужное логическое состояние.

1.3. Технические характеристики модуля USB3000.

Аналого-цифровой преобразователь	
Количество каналов	8 дифференциальных буферизированных каналов
Максимальная частота Дискретизации	3 МГц
Максимальная частота переключения каналов	3 МГц
Разрядность АЦП	14 бит
Диапазон входного сигнала	± 5 В
Входное сопротивление	Не менее 10 МОм для любого режима работы
Подавление синфазной составляющей (для входного сигнала 10 кГц)	-85 дБ
Подавление межканального прохождения (для входного сигнала 10 кГц при частоте переключения каналов 2 МГц)	-95 дБ
Динамический диапазон	80 дБ
Тип используемого АЦП	AD9243
Смещение нуля (с использованием калибровочных коэффициентов)	0.03 % (макс)
Смещение полной шкалы (с использованием калибровочных коэффициентов)	0.03 % (макс)
Интегральная нелинейность преобразования	± 2.5 МЗР (тип)
Дифференциальная нелинейность преобразования	± 0.6 МЗР (тип)
Температурный уход нуля	20 ppm/°C (тип)
Температурный уход полной шкалы	20 ppm/°C (тип)
Синхронизация ввода	Внутренняя, внешняя (ТТЛ)
Защита входов от перенапряжения:	
Постоянное напряжение	± 25 В
Импульс 1 мс	± 250 В
Цифро-аналоговый преобразователь	
Тип используемого ЦАП	AD5322
Количество каналов	2 буферизированных канала
Максимальное время установления выходного сигнала	10 мкс
Разрядность ЦАП	12
Выходной диапазон	± 5 В
Буферизация выходов ЦАП	Буфера типа AD8032AR
Логический анализатор (цифровые входы)	
Количество входов	10
Входное напряжение высокого уровня	1.7 ... 5.75 В
Входное напряжение низкого уровня	- 0.5 ... + 0.8 В
Входной ток ($V_I = 3.3$ В)	10 мкА

Максимальная частота опроса	6 МГц
Синхронизация опроса	Внутренняя, внешняя (ТТЛ)
Цифровые выходы	
Количество выходов (ТТЛ)	8
Минимальное напряжение высокого уровня, ($I_O = -8\text{мА}$)	2.4 В
Максимальное напряжение низкого уровня ($I_O = 8\text{ мА}$)	0.4 В
Цифровой сигнальный процессор	
Тип процессора	ADSP-2185M
Тактовая частота	72 МГц
Внутренняя память программ	16 кСлов
Внутренняя память данных	16 кСлов
Общие характеристики	
Интерфейс	USB 2.0, USB 1.1 ¹
Питание	От шины USB либо от внешнего источника ²
Потребляемый ток	До 450 мА
Габариты	140 x 110 x 35 мм
Условия эксплуатации и хранения	
Условия эксплуатации	от +5°C до +55°C при относительной влажности от 5% до 90%
Температура хранения	от -10°C до +70°C

На рисунках ниже представлены типичные характеристики канала АЦП модуля USB3000: нелинейные искажения аналогового тракта, шум в полосе $\frac{1}{2} f_{\text{АЦП}}$, разброс значений, полученных с АЦП при постоянном напряжении на входе модуля, а также график зависимости межканального прохождения от частоты переключения каналов.

¹ При использовании шины USB 1.1. скорость передачи данных между модулем и ПК падает до 1 Мбайт/с.

² Низкая потребляемая мощность модуля USB3000 позволяет обойтись без внешнего источника. Внешний источник можно использовать при работе с ноутбуком, в целях экономии энергии аккумулятора ноутбука.

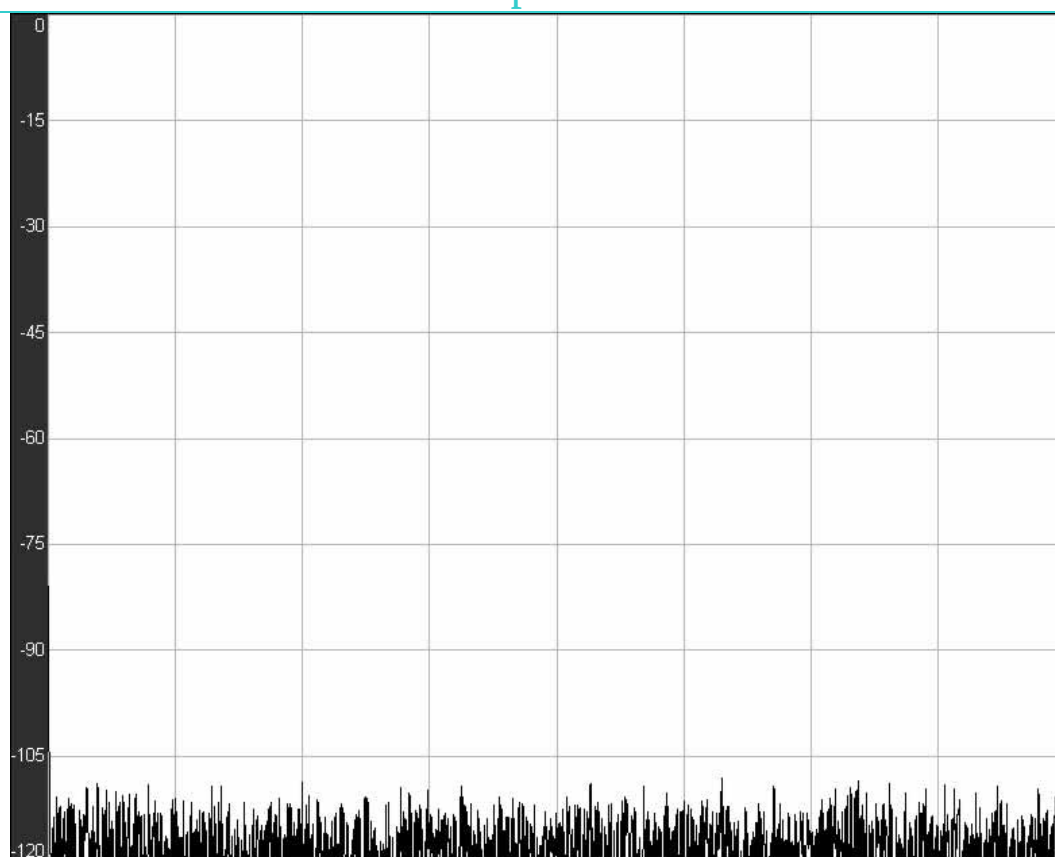


Рис. 2. Шум в полосе $\frac{1}{2} f_{\text{АЦП}}$ при частоте работы АЦП 3 МГц.

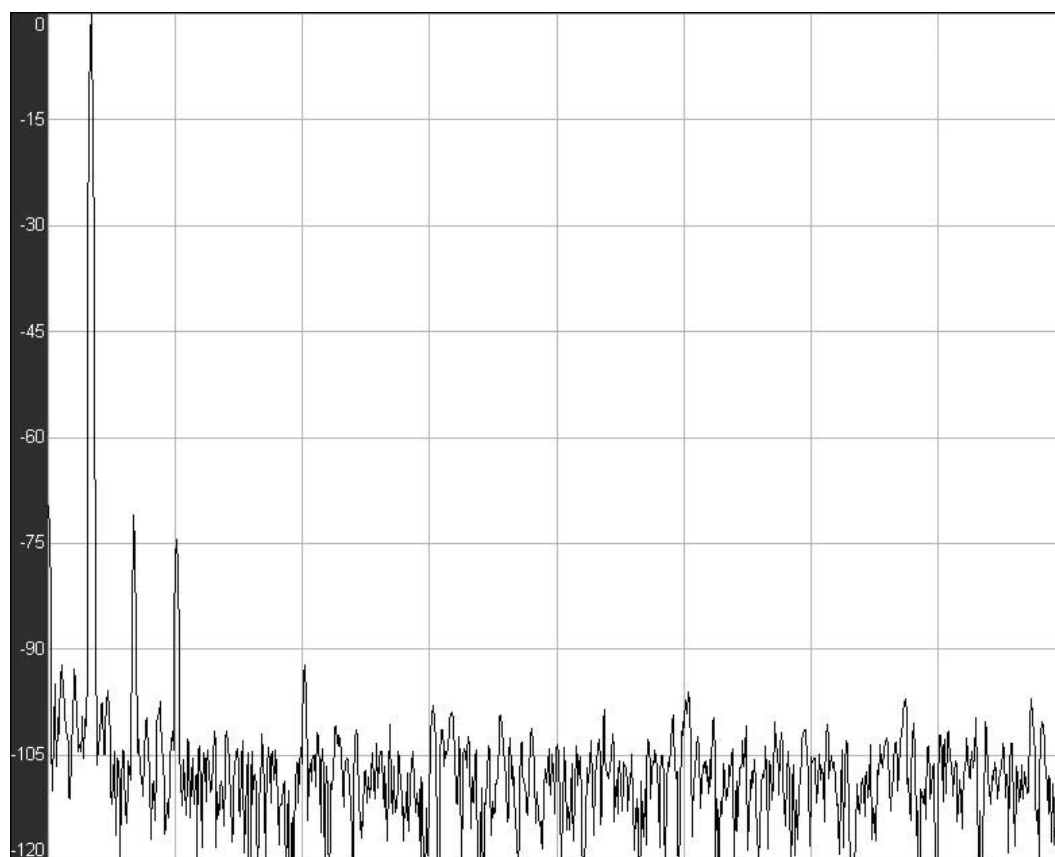


Рис. 3. Нелинейные искажения. Сигнал – синус 10 кГц амплитудой 5В. Частота работы АЦП - 3 МГц.

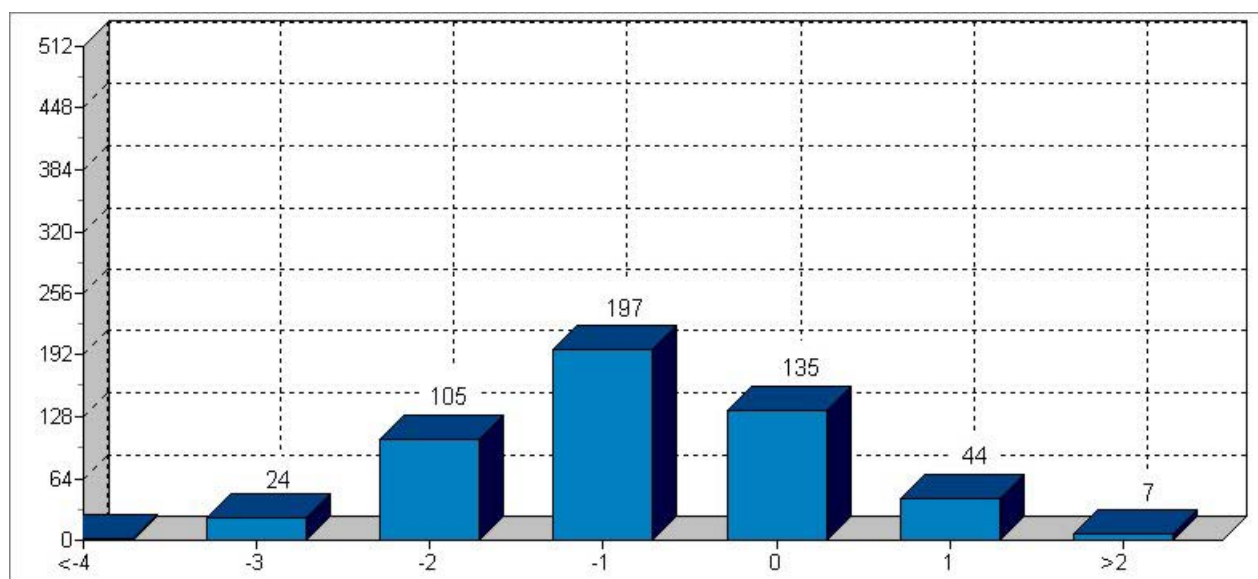


Рис. 4. Разброс значений, полученных с АЦП. Напряжение на входе модуля - 0 В. Частота работы АЦП - 3 МГц. Выборка - 512 значений.

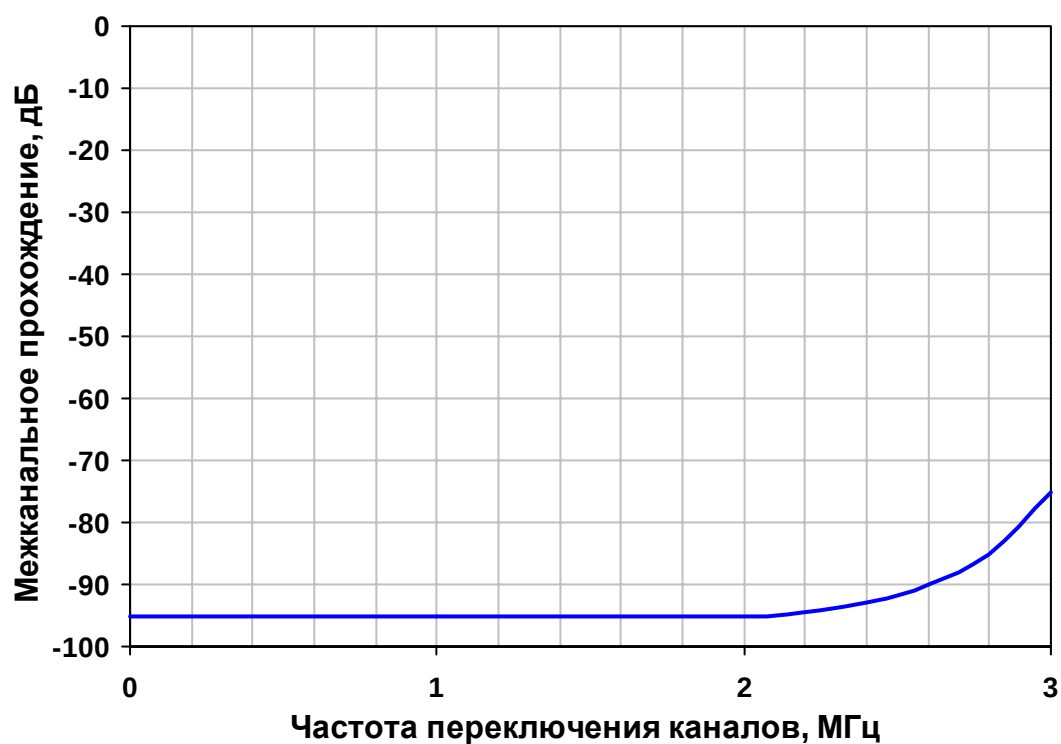


Рис. 5. Зависимость межканального прохождения от частоты переключения каналов.

1.4. Комплект поставки модуля USB3000.

Изделие поставляется в следующей комплектации:

1. Модуль USB3000.
2. Ответные части разъемов для подключения аналоговых и цифровых сигналов.
3. Стандартный кабель USB длиной 1,5 м.
4. Источник питания.
5. CD-ROM, содержащий драйвер модуля USB3000, штатное программное обеспечение и настоящее Техническое описание и инструкцию по эксплуатации.

2. Подключение модуля USB3000.

2.1. Подготовка к работе

На [Рис. 6.](#) представлен вид лицевой панели модуля USB3000:

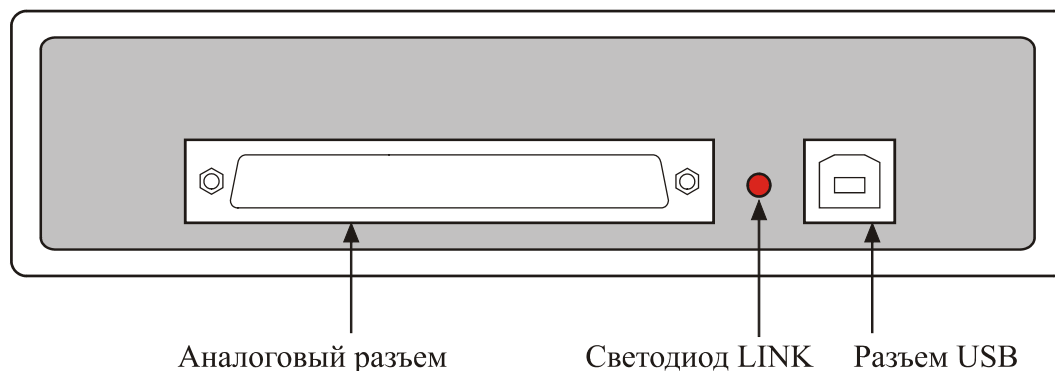


Рис. 6.

Светодиод LINK – загорается при подключении модуля к шине USB после успешной нумерации устройства. Сигнализирует о том, что USB-порт компьютера правильно распознал модуль USB3000.

Разъем USB – тип Б. Стандартный разъем для подключения устройства к ПК по шине USB кабелем типа А-Б (кабель входит в комплект поставки изделия USB3000).

Аналоговый разъем – тип DRB-37M. Служит для подключения аналоговых сигналов к модулю USB3000. В штатный комплект поставки входит ответная часть для этого разъема – разъем типа DB-37F.

На [Рис. 7.](#) представлен вид задней панели модуля USB3000:

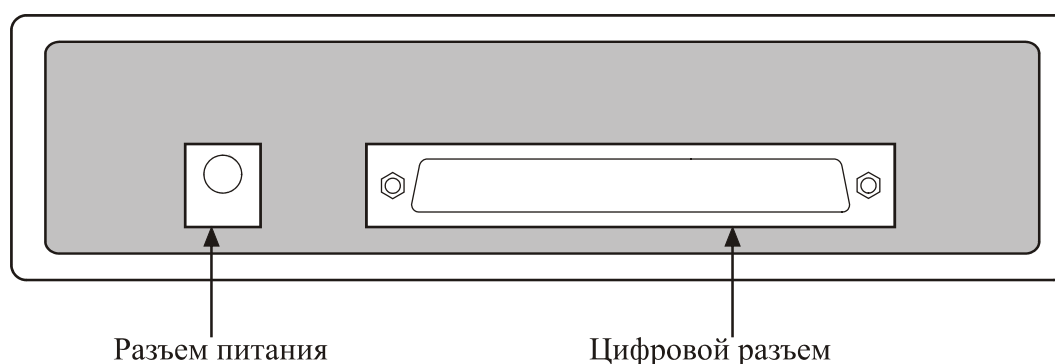


Рис. 7.

Цифровой разъем – тип DRB-37F. Служит для подключения цифровых сигналов к модулю USB3000. В штатный комплект поставки входит ответная часть для этого разъема – разъем типа DB-37M.

Разъем питания – предназначен для подачи внешнего питания на модуль USB2185 от внешнего источника питания. Низкая потребляемая мощность модуля USB3000 позволяет обойтись без внешнего источника. Внешний источник можно использовать при работе с ноутбуком, в целях экономии энергии аккумулятора ноутбука.

ВНИМАНИЕ!!! Производитель гарантирует нормальную работу модуля USB3000 только с источником питания, входящим в комплект поставки! Изделие не подлежит гарантийному ремонту, если оно использовалось совместно с источником питания, не входящим в комплект поставки изделия USB3000.

Порядок включения модуля USB3000 таков:

1. Если Вы собираетесь использовать внешний источник питания, подключите источник питания, входящий в комплект поставки модуля USB3000 к сети переменного тока 220 В и к [Разъему питания](#) модуля USB3000.
2. Включите компьютер, если он был выключен, и загрузите операционную систему Windows 2000/XP.
3. Подключите модуль USB3000 к компьютеру, как описано в главе [Подключения модуля USB3000 к компьютеру](#).
4. Подключите к модулю USB3000 источники сигналов, как описано в главе [Подключение к модулю USB3000 источников сигнала](#).

Порядок отключения модуля USB3000 таков:

1. Отключите от модуля USB3000 источники сигналов.
2. Отключите модуль USB3000 от компьютера.
3. Отсоедините от модуля USB3000 внешний источник питания, если он использовался.

2.2. Подключения модуля USB3000 к компьютеру.

Сама процедура аппаратного подключения модуля USB3000 к компьютеру достаточно тривиальна: необходимо просто соединить [Разъем USB](#) модуля 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: Windows 2000/XP. 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 модуль USB3000, а потом подключать модуль к компьютеру.

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 модуля USB3000 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. 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-3000 ADC/DAC Unit (USB 2.0)", как это 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.

2.3. Подключение к модулю USB3000 источников сигнала.

2.3.1. Подключение источников сигнала к аналоговому разъему.

Аналоговый разъем модуля описан в Табл.1., где X_n — не инвертирующий, а Y_n — инвертирующий входы дифференциального канала n ; NC — вывод разъема зарезервирован.

Таблица 1

№ линии	Назначение	№ линии	Назначение
1	вход X1	20	вход Y1
2	вход X2	21	вход Y2
3	вход X3	22	вход Y3
4	вход X4	23	вход Y4
5	вход X5	24	вход Y5
6	вход X6	25	вход Y6
7	вход X7	26	вход Y7
8	вход X8	27	вход Y8
9	AGND – аналоговая земля	28	AGND – аналоговая земля
10	SYN – вход внешней синхронизации ¹	29	NC
11	DAC1 – выход первого канала ЦАП	30	DAC2 – выход второго канала ЦАП
12	выход - 6.3 В (аналоговое питание)	31	NC
13	выход + 6.3 В (аналоговое питание)	32	NC
14	выход + 3.3 В (цифровое питание)	33	выход + 3.3 В (цифровое питание)
15	выход + 5 В (цифровое питание)	34	выход + 5 В (цифровое питание)
16	NC	35	NC
17	NC	36	NC
18	NC	37	NC
19	NC		

¹ Допустимое напряжение на входе SYN – 0... 3,5 В относительно земли модуля (контакты 9, 28)

Входные аналоговые каналы модуля USB3000 – дифференциальные. Дифференциальное подключение источника сигнала снижает уровень синфазных помех. Помимо этого, дифференциальные входы позволяют подключать источники сигнала таким образом, чтобы токи сигнальных цепей не протекали через один общий провод, что повышает точность измерений.

Правильное подключение источников аналогового сигнала — наиболее важное условие корректной работы системы сбора данных, которое позволяет избежать множества проблем при эксплуатации системы.

При подключении источников аналогового сигнала к модулю USB3000 необходимо придерживаться следующих рекомендаций:

1. При дифференциальном подключении измеряется именно разность напряжений между инвертирующим и неинвертирующим входами канала, т.е. дифференциальное напряжение. Тем не менее, необходимо помнить, что **напряжение относительно аналоговой земли модуля на обоих входах (синфазное напряжение) не должно превышать допустимого диапазона входного сигнала** (см. [Технические характеристики модуля USB3000.](#)).
2. Правильное подключение сигнала к дифференциальному входу — это всегда **трехпроводное соединение**. Необходимо разделять сигнальные провода, подключенные к высокоимпедансному входу, и общий провод заземления. Таким образом, исключается протекание большого тока по сигнальным проводам, снижающее точность измерений.
3. При подключении нескольких источников сигнала к модулю желательно, чтобы их общие провода соединялись **только в одной точке** — на контакте **AGND** аналогового разъема модуля. Это исключит образование «земляных петель», являющихся источником дополнительных помех.
4. **Неиспользуемые входы** необходимо заземлить — т.е. просто соединить с контактом **AGND** аналогового разъема модуля.

На [Рис. 8.](#) приведены примеры корректного подключения однофазных и двухфазных (дифференциальных) источников сигнала к модулю USB3000. Обратите внимание, что подключение к дифференциальному входу даже однофазных источников сигнала должно осуществляться тремя проводами!

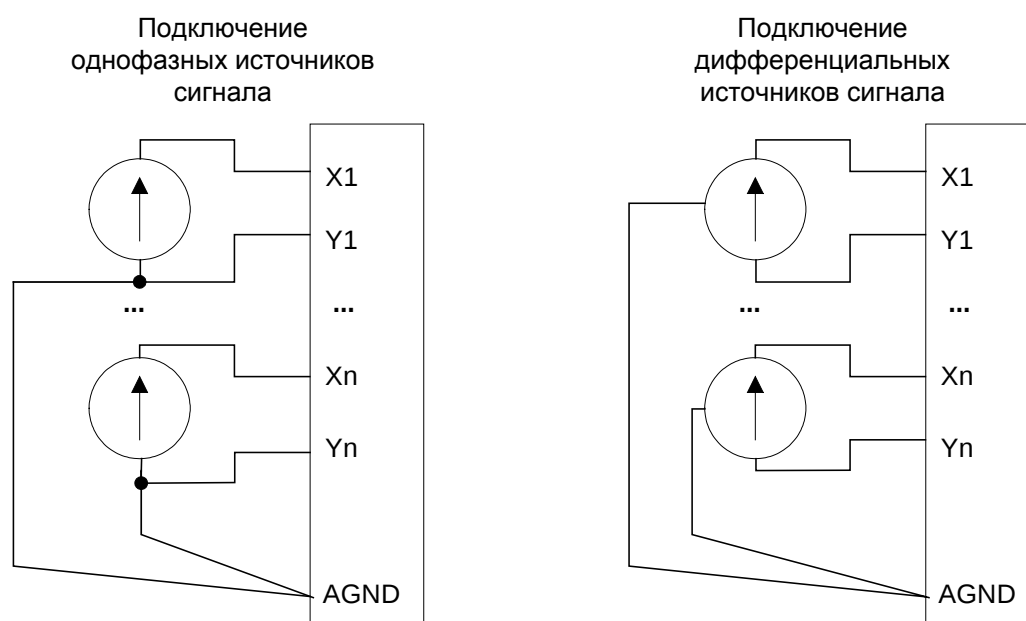


Рис. 8.

2.3.2. Подключение источников сигнала к цифровому разъему.

Цифровой разъем модуля описан в Табл. 2.

NC – контакт разъема зарезервирован.

Выход ADC_CONV – это сигнал запуска АЦП модуля, выведенный на разъем. Может использоваться для синхронизации работы нескольких модулей USB3000.

Вход DSYN (дополнительное прерывание DSP) штатным ПО не используется и предназначен для нужд пользователей, пишущих свои программы для DSP. Может оставаться неподключенным.

Таблица 2

N линии	Назначение	N линии	Назначение
1	вход DIN1	20	выход DOUT1
2	вход DIN2	21	выход DOUT2
3	вход DIN3	22	выход DOUT3
4	вход DIN4	23	выход DOUT4
5	вход DIN5	24	выход DOUT5
6	вход DIN6	25	выход DOUT6
7	вход DIN7	26	выход DOUT7
8	вход DIN8	27	выход DOUT8
9	вход DIN9	28	NC
10	вход DIN10	29	DSYN – вход, прерывание DSP (IRQ0) ¹
11	ADC_CONV - выход, запуск АЦП	30	GND – цифровая земля
12	GND – цифровая земля	31	NC
13	NC	32	выход + 5 В (цифровое питание)
14	выход + 5 В (цифровое питание)	33	NC
15	NC	34	выход + 3.3 В (цифровое питание)
16	выход + 3.3 В (цифровое питание)	35	NC
17	NC	36	NC
18	NC	37	NC
19	NC		

¹ Допустимое напряжение на входе DSYN – 0... 3,5 В относительно земли модуля (контакты 12, 30)

3. Руководство программиста.

3.1. Введение

Программное обеспечение модуля USB3000 включает в себя следующие компоненты:

- драйвер модуля USB3000 для работы в среде Windows'2000/XP;
- драйвер (управляющая программа) DSP модуля USB3000;
- штатная библиотека подпрограмм (DLL) для работы с модулем USB3000;
- ряд законченных примеров программирования модуля.

Опишем кратко назначение этих компонентов:

Драйвер модуля USB3000 предназначен для корректной работы изделия под управлением ОС Windows'2000/XP. При самом первом подключении модуля к ПК необходимо указать ОС месторасположение драйвера модуля USB3000. При дальнейшей работе с изделием USB3000 операционная система уже будет знать, где находятся драйвера для данного типа устройства, и будет подгружать их автоматически по мере необходимости.

На модуле USB3000 установлен современный DSP (цифровой сигнальный процессор) производства фирмы [Analog Devices](#) — ADSP-2185MKST-300. Именно DSP осуществляет низкоуровневое управление модулем: управляет работой АЦП, ЦАП, логических линий, входных коммутаторов, осуществляет калибровку данных, их буферизацию и т.д. DSP функционирует под управлением драйвера DSP, входящего в комплект штатного ПО.

Штатная библиотека подпрограмм для работы с модулем написана на языке C++. В библиотеку мы попытались включить множество разнообразных функций для облегчения пользователю процедуры написания собственных программ по управлению модулем *USB3000*. Данная библиотека позволяет Вам использовать практически все возможности модуля, не вдаваясь в тонкости их низкоуровневого программирования. Если же Вы все-таки собираетесь сами программировать модуль на низком уровне, то наша библиотека может быть использована Вами в качестве законченного и отлаженного примера, на основе которого Вы можете реализовать свои собственные алгоритмы.

Примеры программирования представляют собой небольшие законченные программы, написанные с использованием функций штатной библиотеки. Эти программы поясняют основные приемы работы с модулем USB3000, а также демонстрируют возможности модуля.

3.2. Используемые термины и форматы данных

3.2.1. Используемые термины

Название	Пояснение
AdcRate	Частота работы АЦП (частота опроса цифровых линий для режима Логический анализатор) в $\kappa\Gamma\text{ц}$
ChannelRate	Частота работы аналогового канала в $\kappa\Gamma\text{ц}$
DacRate	Частота работы ЦАП в $\kappa\Gamma\text{ц}$
Buffer	Указатель на целочисленный массив для данных
Npoints	Число отсчетов ввода
AdcChannel	Логический номер канала АЦП
ControlTable	Управляющая таблица, содержащая целочисленный массив с логическими номерами каналов для последовательного циклического ввода данных с АЦП
ControlTableLength	Длина управляющей таблицы
Address	Адрес ячейки в памяти программ или данных DSP модуля

3.2.2. Форматы данных

3.2.2.1. Формат слова данных АЦП

Данные, считанные с 14^{ти} битного АЦП модуля USB3000, представляются в формате знакового целого двухбайтного числа от -8192 до 8191. Соответствие кода АЦП напряжению на аналоговом входе приведено в следующей таблице:

Таблица 3. Соответствие кода АЦП напряжению на аналоговом входе

Код	Напряжение, В
+8000	+5.0
0	0
-8000	-5.0

3.2.2.2. Формат слова данных ЦАП

Формат 16^{ти} битного слова данных, передаваемого из ПК в модуль для последующей выдачи на ЦАП, приведен в следующей таблице:

Таблица 4. Формат слова данных ЦАП

Номер бита	Назначение
0÷11	12 ^{ти} битный код ЦАП
12	0
13	0
14	1
15	Выбор номера канала ЦАП: ✓ '0' – первый канал; ✓ '1' – второй канал.

Собственно код, выдаваемый модулем на 12^{ти} битный ЦАП, связан с выходным напряжением, устанавливаемым на аналоговом разъеме, в соответствии со следующей таблицей:

Таблица 5. Соответствие кода ЦАП напряжению на выходе

Код	Напряжение, В
+2047	+5.0
0	0
-2048	-5.0

3.2.2.3. Логический номер канала АЦП

На модуле USB3000 для управления работой входного аналогового каскада определяется такой параметр, как 4^x битный *логический номер канала* (фактически, управляющее слово). Именно массив логических номеров каналов, образующих управляющую таблицу **ControlTable**, задает циклическую последовательность работы при вводе данных. В состав логического номера канала входят два параметра, задающих различные режимы функционирования модуля:

- номер аналогового канала АЦП;
- флаг ввода информации с 10^{th} цифровых линий.

При этом сам формат логического номера канала имеет вид:

Номер бита	Обозначение	Функциональное назначение
0	ADC0	0 ^{ой} бит номера канала АЦП
1	ADC1	1 ^{ый} бит номера канала АЦП
2	ADC2	2 ^{ой} бит номера канала АЦП
3	TTLIN	флаг ввода с цифровых линий («1» - активн.)

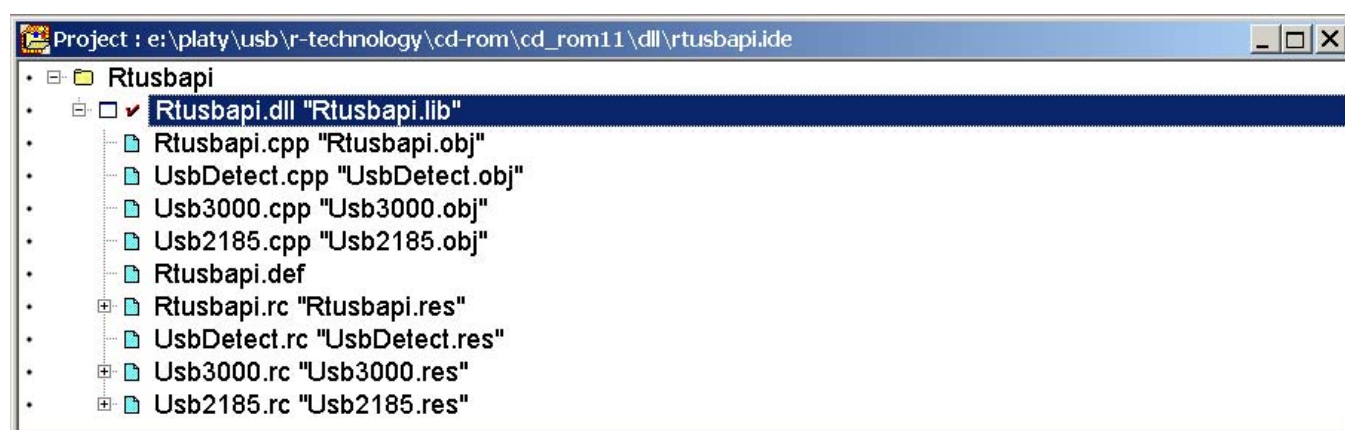
Если библиотека Rtusbapi обнаружит в управляющей таблице хоть один *логический номер канала*, содержащий флаг ввода с цифровых линий, то модуль автоматически переводится в режим «логический анализатор».

3.3. Штатная библиотека

Настоящий раздел описания предназначен для программистов, собирающихся писать свои собственные программы в среде *Windows2000/XP* для работы с изделиями USB3000. 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 USB3000 входит готовая библиотека штатных подпрограмм, в виде динамически подключаемой библиотеки (DLL). В библиотеку мы попытались включить множество самых необходимых функций для облегчения пользователю процедуры написания собственных программ по управлению модулем USB3000. 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.

3.3.1. Общий подход к работе со штатной библиотекой

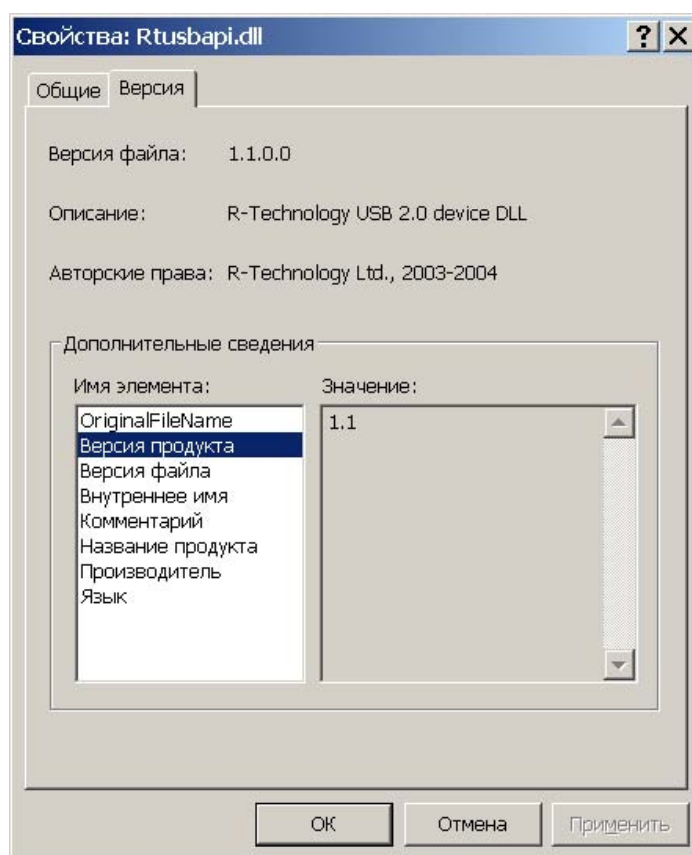
Общий вид проекта DLL библиотеки в среде разработки **Borland C++ 5.02** представлен на рисунке ниже:



Сама библиотека содержит всего две экспортируемые функции, одна из которых, [RtCreateInstance\(\)](#), возвращает указатель на интерфейс модуля USB3000. 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 прочесть номер версии DLL библиотеки (две старшие цифры):



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

Тексты законченных примеров применения интерфейсных функций из штатной DLL библиотеки для различных сред разработки приложений можно найти в следующих директориях:

- \Examples\BC5 – для среды **Borland C++ 5.02**;
- \Examples\MSVC6 – для среды **MS Visual C++ 6.0**.

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 USB3000;
- 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.

Поклонники диалекта **Microsoft Visual C++** могут использовать соответствующую библиотеку импорта **RTUSBAPI.LIB** из директории **\DLL\MSVC**.

Приведем пример создания "скелета" типичной программы для управления изделием USB3000.

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 модуль USB3000 подключен к выбранному виртуальному слоту.

С помощью дополнительной интерфейсной функции [GetModuleName\(\)](#) можно прочитать название устройства, подключенного к выбранному виртуальному слоту.

Теперь можно узнать серийный номер модуля. Для этого следует воспользоваться интерфейсной функцией [GetModuleSerialNumber\(\)](#).

Важной особенностью модуля USB3000 является то, что на нем установлен цифровой сигнальный процессор (DSP – Digital Signal Processor) с фиксированной точкой [ADSP-2185M](#) фирмы **Analog Devices**. Для того чтобы его “оживить”, т.е. заставить работать по требуемому Вам алгоритму, во внутреннюю память DSP надо записать (загрузить) управляющую программу (файл **USB3000.rtd**, входящий в штатное ПО). Задачей DSP является управление всей устанавливаемой на модуле периферией (АЦП, ЦАП, цифровые линии и т.д.), а также, при необходимости, первичная обработка информации.

Теперь необходимо загрузить в сигнальный процессор модуля управляющую программу (драйвер DSP). Для этого можно воспользоваться интерфейсной функцией [LOAD_DSP\(\)](#). В случае успешного выполнения данной функции, 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 рекомендуется получить информацию о загруженном в модуль драйвере DSP. Сделать это можно с помощью дополнительно введенной интерфейсной функции [GET_DSP_INFO\(\)](#).

На следующем этапе Вам следует прочитать служебную информацию, хранящуюся в ППЗУ модуля. Она требуется при работе с некоторыми интерфейсными функциями штатной DLL библиотеки. Для этой цели предназначена интерфейсная функция [GET_FLASH\(\)](#). Если функция не вернула ошибку, то это означает, что информация из ППЗУ модуля успешно считана, и можно продолжать работу.

Далее можно реализовывать любые необходимые алгоритмы работы с модулем USB3000.

В качестве иллюстрации приведем исходный текст очень простой консольной программы для работы с модулем USB3000:

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

IRTUSB3000 *pModule;          // указатель на интерфейс модуля
RTUSB3000::FLASH fi;           // структура с информацией из ППЗУ модуля
RTUSB3000::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
    pModule = static_cast<IRTUSB3000 *>(CreateInstance("usb3000"));
    if(pModule == NULL)
    {
        printf("Не могу получить указатель на интерфейс");
        return 1;                //выйдем из программы с ошибкой
    }

    // попробуем обнаружить модуль
    // в нулевом виртуальном слоте
    if(!pModule->OpenDevice(0))
    {
        printf("Не могу получить доступ к модулю!");
        return 1;                //выйдем из программы с ошибкой
    }

    // прочитаем название модуля в нулевом виртуальном слоте
    if(!pModule->GetModuleName(ModuleName))
    {
        printf("Не могу прочитать название модуля!\n");
        return 1;                //выйдем из программы с ошибкой
    }

    // проверим: этот модуль - ' USB3000'?
    if(strcmp(ModuleName, "USB3000"))
    {
        printf(" В нулевом виртуальном слоте не 'USB3000'\n");
        return 1;                //выйдем из программы с ошибкой
    }

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

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

// получим версию загруженного BIOSa
if(!pModule->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(!pModule->GET_FLASH(&fi))
{
    printf("Не выполнена функция GET_FLASH ()!\n");
    return 1;                //выйдем из программы с ошибкой
}

printf("Модуль USB3000 (серийный номер %s) полностью готов к\
      работе!\n", ModuleSerialNumber);

// далее можно располагать функции для непосредственного
// управления вводом/выводом данных в/из модуля!
. . . . .

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

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

3.3.2. Описание структур штатной библиотеки

3.3.2.1. Структура RTUSB3000::INPUT_PARS

Структура *INPUT_PARS* описана в файле *Rtusbapi.h* и представлена ниже:

```
struct INPUT_PARS
{
    WORD size; // размер данной структуры в байтах
    BOOL InputEnabled // флажок состояние ввода данных (при чтении)
    BOOL CorrectionEnabled; // управление корректировкой данных
    BOOL InputType; // тип вводимых с модуля данных (АЦП или ТТЛ)
    WORD SynchroType; // тип синхронизации вводимых с модуля данных
    WORD SynchroAdType // тип аналоговой синхронизации
    WORD SynchroAdMode; // режим аналоговой синхронизации
    WORD SynchroAdChannel; // канал АЦП при аналоговой синхронизации
    WORD SynchroAdPorog; // порог срабатывания АЦП при аналоговой
    // синхронизации
    WORD ChannelsQuantity; // число активных каналов (размер кадра)
    WORD ControlTable[128]; // управляющая таблица с активными каналами
    WORD InputFifoBaseAddress; // базовый адрес FIFO буфера ввода данных
    WORD InputFifoLength; // длина FIFO буфера ввода данных
    double InputRate; // тактовая частота ввода данных в кГц
    double InterKadrDelay; // межкадровая задержка в мс
    double ChannelRate; // частота одного канала кГц (период кадра)
    double AdcOffsetCoef[8]; // корректировочные коэф. смещение нуля для АЦП
    double AdcScaleCoef[8]; // корректировочные коэф. масштаба для АЦП
};
```

Поле *ap->CorrectionEnabled* позволяет модулю по желанию пользователя осуществлять корректировку получаемых с АЦП данных. В этом случае из модуля в РС будут поступать уже откорректированные данные с АЦП. Если *ap->CorrectionEnabled* равно *TRUE*, то корректировка разрешена, если *FALSE* – нет. При использовании корректировки сами корректировочные коэффициенты должны находиться в полях *ap->AdcOffsetCoef* и *ap->AdcScaleCoef* (см. ниже).

Поле *ap->SynchroType* определяет следующие режимы ввода данных:

ap->SynchroType = 0 – отсутствие какой-либо синхронизации ввода;

ap->SynchroType = 1 – цифровая синхронизация начала (старта) сбора – по импульсу [на контакте SYN](#);

ap->SynchroType = 2 – покадровая цифровая синхронизация (пока не реализована);

ap->SynchroType = 3 – аналоговая синхронизация по логическому каналу АЦП (пока не реализована).

При цифровой синхронизации начала (старта) сбора модуль начинает осуществлять ввод данных только **после** прихода отрицательного перепада () ТТЛ-совместимого одиночного импульса на вход [SYN](#) аналогового разъема. Длительность этого синхроимпульса должна быть не менее 50 нс.

Поле *ap->ChannelsQuantity* определяет количество активных логических каналов для сбора данных (не более 128).

Поле *ap->ControlTable[]* задает [управляющую таблицу](#) (массив) логических каналов, количество которых равно *ChannelsQuantity*. Эта таблица используется модулем при сборе данных для задания циклической последовательности ввода.

При вызове данной интерфейсной функции в полях *ap->AdcRate* и *ap->InterKadrDelay* должны находиться частота оцифровки данных [AdcRate](#) (частота работы АЦП или частота опроса цифровых линий в кГц) и межкадровая задержка *InterKadrDelay* (зарезервирована, должна равняться 0). После выполнения этой функции в этих полях находятся **реально установленные** значения величин межканальной и межкадровой задержек, **максимально близкие** к изначально задаваемым. Это происходит вследствие того, что реальные значения *AdcRate* и *InterKadrDelay* не являются непрерывными величинами, а принадлежат некоему дискретному ряду. Так, в общем виде, частота работы АЦП определяется по следующей формуле: $AdcRate = F_{clockout} / (2 * (N + 1))$, где $F_{clockout}$ – тактовая частота, установленного на модуле DSP, равная 72000 кГц, N – целое число. Поэтому данная функция просто вычисляет ближайшую к задаваемой величину *AdcRate*, передает ее в модуль (в виде целого числа N) и возвращает ее значение в поле *ap->AdcRate*.

Поле *ap->ChannelRate* является выходным для данной функции и в нем возвращается частота опроса [ChannelRate](#) (в кГц) фиксированного канала из управляющей таблицы (фактически это период кадра).

Поле *ap->InputFifoBaseAddress* задает базовый адрес FIFO буфера ввода в DSP модуля. Для данного модуля он всегда равен 0x0.

Поле *ap->InputFifoLength* задает длину FIFO буфера ввода в DSP модуля. Для данного модуля эта величина может находиться в диапазоне от 0x800 (2048) до 0x3000 (12288), а также быть обязательно кратной 0x800 (2048). Если переданное в функцию значение не удовлетворяет указанным требованиям, то выполняется необходимая корректировка, и по завершении данной функции в поле *ap->InputFifoLength* будет находиться **реально установленное** значение длины FIFO буфера ввода. Передача данных из FIFO буфера ввода модуля в PC производится порциями по *ap->InputFifoLength/2* отсчетов (по мере их готовности).

Поля *ap->AdcOffsetCoeff[]* и *ap->AdcScaleCoeff[]* содержит корректировочные коэффициенты для получаемых с АЦП данных. Управление корректировкой данных осуществляется с помощью поля *ap->CorrectionEnabled*. В качестве этих коэффициентов можно использовать либо Ваши собственные, либо штатные, которые хранятся в ППЗУ модуля.

Перед началом ввода данных в ПК необходимо заполнить поля данной структуры и передать ее в модуль с помощью интерфейсной функции [SET_INPUT_PARS\(\)](#). При этом в качестве корректировочных коэффициентов для получаемых с АЦП отсчетов можно использовать соответствующую информацию из ППЗУ модуля. Также при необходимости можно считать из модуля текущие параметры функционирования АЦП, используя [GET_INPUT_PARS\(\)](#).

3.3.2.2. Структура RTUSB3000::OUTPUT_PARS

Структура *OUTPUT_PARS* описана в файле *Rtusbapi.h* и представлена ниже:

```
struct OUTPUT_PARS
{
    WORD size; // размер данной структуры в байтах
    BOOL OutputEnabled; // флажок состояние вывода данных (при чтении)
    double OutputRate; // частота вывода данных в кГц
    WORD OutputFifoBaseAddress; // базовый адрес FIFO буфера вывода
    WORD OutputFifoLength; // длина FIFO буфера вывода
};
```

Поле *dp->OutputRate* задает частоту вывода [DacRate](#) в кГц. После выполнения данной интерфейсной функции в данном поле находится **реально установленное** значение частоты вывода данных. Это происходит вследствие того, что реальные значения *DacRate* не являются непрерывной величиной, а принадлежат некоему дискретному ряду. Так, в общем виде, частота вывода определяется по следующей формуле: $DacRate = F_{clockout} / (9 * (N + 1))$, где $F_{clockout}$ – тактовая частота установленного на модуле DSP равная 72000 кГц, *N* – целое число. Поэтому данная функция просто вычисляет ближайшую к задаваемой дискретную величину *DacRate*, передает ее в модуль (в виде целого числа *N*) и возвращает ее значение в поле *dp->OutputRate*.

Поле *dp->OutputFifoBaseAddress* задает базовый адрес FIFO буфера вывода. Для данного модуля он всегда равен 0x3000.

Поле *dp->OutputFifoLength* задает длину FIFO буфера вывода. Для данного модуля эта величина может находиться в диапазоне от 0x80 (128) до 0xF80 (3968), а также также быть обязательно кратной 0x80(128). Если переданное в функцию значение не удовлетворяет указанным требованиям, то выполняется необходимая корректировка и по завершении данной функции в поле *dp->OutputFifoLength* будет находиться **реально установленное** значение длины FIFO буфера вывода. Передача ('подкачка') новых данных из PC в FIFO буфер ЦАП модуля производится порциями по *dp->OutputFifoLength/2* отсчетов (по мере их необходимости).

Перед началом вывода данных необходимо заполнить поля этой структуры и передать ее в модуль с помощью интерфейсной функции [SET_OUTPUT_PARS\(\)](#). Также при необходимости можно считать из модуля текущие параметры вывода данных, используя [GET_OUTPUT_PAR\(\)](#).

3.3.2.3. Структура RTUSB3000::FLASH

Структура *FLASH* описана в файле `Rtusbapi.h` и представлена ниже:

```
struct FLASH
{
    WORD CRC16;                // контрольная сумма
    WORD size;                 // размер данной структуры в байтах
    BYTE SerialNumber[9];      // серийный номер модуля (8 символов)
    BYTE Name[11];             // название модуля – “USB3000”
    BYTE Revision;             // ревизия модуля: ‘А’
    BYTE DspType[17];          // тип установленного на модуле DSP:
    BYTE IsDacPresented;       // флажок наличия ЦАП на модуле:
                                // 0 – ЦАП отсутствует,
                                // 1 - ЦАП присутствует,
    DWORD DspClockout;         // тактовая частота DSP, равная 72 000 000 Гц.
    float AdcOffsetCoef[8];    // корректировочные коэф. смещения нуля для АЦП
    float AdcScaleCoef[8];     // корректировочные коэф. масштаба для АЦП
    float DacOffsetCoef[2];    // корректировочные коэф. смещения нуля для ЦАП
    float DacScaleCoef[2];     // корректировочные коэф. масштаба для ЦАП
    BYTE ReservedWord[129];    // зарезервировано
};
```

Данная структура используется в интерфейсных функциях, которые работают со служебной областью пользовательского ППЗУ: [PUT_FLASH\(\)](#) и [GET_FLASH\(\)](#).

3.3.3 Описание функций штатной библиотеки

3.3.3.1. Функции общего характера

3.3.3.1.1 Получение версии DLL библиотеки

Формат:	LPVOID	<i>RtGetDllVersion(void)</i>
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.		

3.3.3.1.2 Получение указателя на интерфейс модуля

Формат:	LPVOID	<i>RtCreateInstance(const PCHAR DeviceName)</i>
Назначение: Данная функция должна обязательно вызываться в начале каждой пользовательской программы, работающей с модулями <i>USB3000</i> . Она является одной из двух экспортируемых из штатной DLL функцией и возвращает указатель на интерфейс для устройства с названием <i>DeviceName</i> . Все интерфейсные функции штатной DLL библиотеки вызываются именно через этот возвращаемый указатель. Рекомендованную последовательность вызовов интерфейсных функций см. Общий подход к работе со штатной библиотекой .		
Передаваемые параметры: <i>DeviceName</i> – строка с названием устройства (для данного модуля это – “USB3000”).		
Возвращаемое значение: В случае успеха — указатель на интерфейс, иначе — NULL .		

3.3.3.1.3 Завершение работы с модулем

Формат:	BOOL	<i>ReleaseDevice(void)</i>
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 выполнялась функция <i>RtCreateInstance()</i>). !!!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: <i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.		

3.3.3.1.4 Инициализация доступа к модулю

Формат:	BOOL	<i>OpenDevice(const WORD VirtualSlot)</i>
Назначение:	С программной точки зрения, подсоединенный к компьютеру модуль <i>USB3000</i> можно рассматривать как устройство, подключённое к некоему виртуальному слоту с сугубо индивидуальным номером. Основное же назначение данной интерфейсной функции – определить, находится ли устройство в заданном виртуальном слоте. Если функция <i>OpenDevice()</i> успешно выполнялась для заданного виртуального слота, то можно быть уверенным, что к этому слоту подключён именно модуль <i>USB3000</i>. Далее можно переходить непосредственно к загрузке модуля и его последующему управлению с помощью соответствующих интерфейсных функций библиотеки <i>Rtusbapi.dll</i>. Рекомендованную последовательность вызовов интерфейсных функций см. Общий подход к работе со штатной библиотекой	
Передаваемые параметры:	• <i>VirtualSlot</i> – номер виртуального слота, к которому, как предполагается, подключен модуль <i>USB3000</i>.	
Возвращаемое значение:	<i>TRUE</i> – устройство <i>USB3000</i> находится в выбранном виртуальном слоте; <i>FALSE</i> – никакого устройства в выбранном виртуальном слоте нет (может, стоит попробовать другой номер виртуального слота).	

3.3.3.1.5 Освобождение виртуального слота

Формат:	BOOL	<i>CloseDevice (void)</i>
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 <i>USB2185</i> становится невозможным. Для возобновления нормального доступа к устройству необходимо вновь воспользоваться интерфейсной функцией OpenDevice(). Таким образом, эта функция, по своей сути, противоположна интерфейсной функции <i>OpenDevice()</i>. Фактически данная функция используется в таких интерфейсных функциях как <i>OpenDevice()</i> и ReleaseDevice().	
Передаваемые параметры:	нет.	
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.	

3.3.3.1.6 Получение названия модуля

Формат:	BOOL	<i>GetModuleName(PCHAR const ModuleName)</i>
Назначение: Данная интерфейсная функция позволяет получить название модуля, подключенного к выбранному виртуальному слоту. Массив под название модуля <i>ModuleName</i> (не менее 6 символов плюс признак конца строки '\0', т.е. нулевой байт) должен быть заранее определен. Рекомендованную последовательность вызовов интерфейсных функций см. Общий подход к работе со штатной библиотекой .		
Передаваемые параметры: <i>ModuleName</i> – возвращается строка, не менее 11 символов, с названием модуля (в нашем случае это должна быть строка "USB3000").		
Возвращаемое значение: <i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.		

3.3.3.1.7 Получение серийного номера модуля

Формат:	BOOL	<i>GetModuleSerialNumber (PCHAR const SerialNumber)</i>
Назначение: Данная интерфейсная функция в переменной <i>SerialNumber</i> возвращает строку с серийным номером модуля. Строка (9 байтов включая '\0') должна быть предварительно определена		
Передаваемые параметры: <i>SerialNumber</i> – строка с серийным номером модуля.		
Возвращаемое значение: <i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.		

3.3.3.1.8 Получение текущей скорости работы USB

Формат:	BOOL	<i>GetUsbSpeed(BYTE * const UsbSpeed)</i>
Назначение: Данная интерфейсная функция в переменной <i>UsbSpeed</i> возвращает текущую скорость работы шины USB.		
Передаваемые параметры: <i>UsbSpeed</i> – эта переменная принимать следующие значения: ✓ 0 – модуль работает в режиме <i>Full Speed</i> (12 Мбит/с, USB 1.1) ✓ 1 – модуль работает в режиме <i>High Speed</i> (480 Мбит/с, USB 2.0).		
Возвращаемое значение: <i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.		

3.3.3.1.9 Получение версии BIOS модуля

Формат:	BOOL	<i>GetAvrVersion(PCHAR const AvrVersion)</i>
Назначение:	Данная интерфейсная функция в строке <i>AvrVersion</i> возвращает номер версии BIOS, зашитого в модуль.	
Передаваемые параметры:	• <i>AvrVersion</i> – номер версии BIOS.	
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.	

3.3.3.1.10 Загрузка драйвера DSP

Формат:	BOOL	<i>LOAD_DSP(const PCHAR FileName = NULL)</i>
Назначение:	Данная интерфейсная функция выполняет операцию загрузки драйвера DSP модуля. Файл <i>FileName</i> с кодом драйвера должен находиться в текущей директории Вашего приложения. В DLL библиотеке есть дополнительная возможность загружать драйвер, содержимое которого хранится в самом теле библиотеки в виде соответствующего ресурса. Для этого достаточно параметр <i>FileName</i> задать в виде NULL . NULL является также значением по умолчанию для параметра <i>FileName</i> . Рекомендованную последовательность вызовов интерфейсных функций см. Общий подход к работе со штатной библиотекой .	
Передаваемые параметры:	• <i>FileName</i> – строка с названием файла, содержащим код загружаемой управляющей программы. Например, для штатного драйвера DSP это строка "Usb3000.rtd". Если данный параметр задан как NULL, то загрузка модуля будет осуществляться тем драйвером DSP, который находится в виде ресурса в теле штатной DLL библиотеки.	
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.	

3.3.3.1.11 Проверка загрузки модуля

Формат:	BOOL	MODULE_TEST(void)
Назначение: Данная интерфейсная функция проверяет правильность загрузки драйвера 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.		

3.3.3.1.12 Получение версии загруженного драйвера DSP

Формат:	BOOL	GET_DSP_INFO(RTUSB3000::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 RTUSB3000::DSP_INFO содержит краткую информацию о драйвере DSP. Она описана в файле Rtusbapi.h и представлена ниже: <pre> struct DSP_INFO { BYTE Target[10]; // название устройства "USB3000" BYTE Label[6]; // разработчик драйвера DSP "rtech" BYTE DspMajor; // старший байт версии драйвера DSP BYTE DspMinor; // младший байт версии драйвера DSP }; </pre>		
Возвращаемое значение: TRUE – функция успешно выполнена; FALSE – функция не выполнена.		

3.3.3.1.13 Сброс DSP модуля

Формат:	BOOL	<i>RESET_DSP(void)</i>
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:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.	

3.3.3.1.14 Передача номера команды в драйвер DSP

Формат:	BOOL	<i>SEND_COMMAND(const WORD Command)</i>
Назначение:	Данная интерфейсная функция передаёт в рекомендованную переменную D_COMMAND номер требуемой команды и вызывает командное прерывание <i>IRQ2</i> 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:	<i>Command</i> – номер команды, передаваемый в драйвер DSP.	
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.	

3.3.3.1.15 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.

3.3.3.2. 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.

3.3.3.2.1 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 16TH битное слово.	
Возвращаемое значение:	TRUE – функция успешно выполнена; FALSE – функция не выполнена.	

3.3.3.2.2 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^x битное слово.	
Возвращаемое значение:	TRUE – функция успешно выполнена; FALSE – функция не выполнена.	

3.3.3.2.3 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 16TH битного слова.	
Возвращаемое значение:	TRUE – функция успешно выполнена; FALSE – функция не выполнена.	

3.3.3.2.4 Запись слова в память программ DSP

Формат:	BOOL <i>PUT_PM_WORD(WORD Address, LONG Data)</i>
Назначение:	Данная функция записывает значение <i>Data</i> в ячейку с адресом <i>Address</i> в памяти программ DSP модуля.
Передаваемые параметры:	• <i>Address</i> – адрес ячейки в памяти программ DSP, куда записывается значение <i>Data</i>; • <i>Data</i> – значение записываемого 24^х битного слова.
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.

3.3.3.2.5 Чтение массива слов из памяти данных DSP

Формат:	BOOL <i>GET_DM_ARRAY(WORD BaseAddress, WORD NPoints, SHORT * const Buffer)</i>
Назначение:	Данная функция считывает массив слов длиной <i>NPoints</i> в буфер <i>Buffer</i> , начиная с адреса ячейки <i>BaseAddress</i> в памяти данных DSP модуля. Буфер <i>Buffer</i> надлежащей длины необходимо заранее определить.
Передаваемые параметры:	• <i>BaseAddress</i> – стартовый адрес в памяти данных DSP, начиная с которого производится чтение массива; • <i>NPoints</i> – длина считываемого массива; • <i>Buffer</i> – указатель на буфер, в который передаются считываемые значения.
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.

3.3.3.2.6 Чтение массива слов из памяти программ DSP

Формат:	BOOL <i>GET_PM_ARRAY(WORD BaseAddress, WORD NPoints, LONG * const Buffer)</i>
Назначение:	Данная функция считывает массив слов длиной <i>NPoints</i> в буфер <i>Buffer</i> , начиная с адреса ячейки <i>BaseAddress</i> в памяти программ DSP модуля. Буфер <i>Buffer</i> надлежащей длины необходимо заранее определить. При использовании этой функции следует помнить, что одно слово памяти программ DSP является 24 ^х битным.
Передаваемые параметры:	• <i>BaseAddress</i> – стартовый адрес в памяти программ DSP, начиная с которого производится чтение массива; • <i>NPoints</i> – число считываемых 24^х битных слов; • <i>Buffer</i> – указатель на буфер, в который передаются считываемые значения.
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.

3.3.3.2.7 Запись массива слов в память данных DSP

Формат:	BOOL <i>PUT_DM_ARRAY</i> (WORD <i>BaseAddress</i> , WORD <i>Npoints</i> , <i>SHORT * const Buffer</i>)
Назначение:	Данная функция записывает массив слов длиной <i>NPoints</i> из буфера <i>Buffer</i> в память данных DSP модуля, начиная с адреса <i>BaseAddress</i> .
Передаваемые параметры:	• <i>BaseAddress</i> – стартовый адрес в памяти данных DSP, начиная с которого производится запись массива; • <i>NPoints</i> – длина записываемого массива; • <i>Buffer</i> – указатель на буфер, из которого идет запись.
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.

3.3.3.2.8 Запись массива слов в память программ DSP

Формат:	BOOL <i>PUT_PM_ARRAY</i> (WORD <i>BaseAddress</i> , WORD <i>NPoints</i> , <i>LONG *const Buffer</i>)
Назначение:	Данная функция записывает массив слов длиной <i>NPoints</i> из буфера <i>Buffer</i> в память программ DSP модуля, начиная с адреса <i>BaseAddress</i> . При использовании этой функции следует помнить, что одно слово памяти программ DSP является 24 ^x битным.
Передаваемые параметры:	• <i>BaseAddress</i> – стартовый адрес в памяти программ DSP, начиная с которого производится запись массива; • <i>NPoints</i> – число записываемых 24^x битных слов; • <i>Buffer</i> – указатель на буфер, из которого идет запись.
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.

3.3.3.2.9 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 <i>программ</i> DSP модуля (см. Драйвер DSP).	
Передаваемые параметры:	• Address – адрес ячейки переменной драйвера в памяти <i>программ</i> 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.	

3.3.3.2.10 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 <i>программ</i> DSP модуля (см. Драйвер DSP).	
Передаваемые параметры:	• Address – адрес ячейки переменной драйвера в памяти <i>программ</i> 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.	

3.3.3.3. Функции ввода данных

Интерфейсные функции штатной DLL библиотеки позволяют реализовывать разнообразные алгоритмы ввода информации (*независимо* от состояния вывода). Модуль, 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. Прежде чем запустить ввод данных, необходимо передать в модуль требуемые параметры сбора: тип синхронизации, частота ввода, длина и базовый адрес FIFO буфера ввода, управляющую таблицу и т.д. Эту операцию можно выполнить с помощью интерфейсной функции [SET_INPUT_PARS\(\)](#). Вводимые данные драйвер DSP модуля складывает в двойной циклический FIFO буфер ввода, который расположен в памяти данных DSP модуля (см. [Драйвер DSP](#)). Для извлечения из модуля этих данных следует пользоваться функцией [ReadData\(\)](#). Примеры корректного применения интерфейсных функций для целей ввода данных можно найти в директории \Example\BC5\ReadData.

3.3.3.3.1 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:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.	

3.3.3.3.2 Останов ввода данных

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

3.3.3.3.3 Установка параметров ввода данных

Формат:	BOOL	SET_INPUT_PARS (RTUSB3000::INPUT_PARS * const ap)
Назначение:	Данная функция передает в модуль в виде структуры INPUT_PARS всю необходимую информацию, которая используется при сборе данных (АЦП или Логический анализатор). Использование модулем этой переданной информацией начинается ТОЛЬКО после выполнения интерфейсной функции START_ADC().	
Передаваемые параметры:	• <i>ap</i> – адрес структуры типа INPUT_PARS с параметрами ввода данных.	
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.	

3.3.3.3.4 Получение текущих параметров ввода данных

Формат:	BOOL	GET_INPUT_PARS (RTUSB3000::INPUT_PARS * const ap)
Назначение:	Данная функция считывает из модуля всю текущую информацию, которая используется при вводе данных из модуля.	
Передаваемые параметры:	• <i>ap</i> – адрес структуры типа INPUT_PARS с полученными из модуля текущими параметрами ввода данных.	
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.	

3.3.3.3.5 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 512 до (1024*1024), а также быть кратной 512. 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 же сей асинхронной 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, например, исходные тексты прилагаемой к модулю законченной программы в директории *\Examples\BC5\ReadData*.

!!!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 –512, 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](#)).

3.3.3.4. Функции вывода данных

Интерфейсные функции штатной DLL библиотеки позволяют реализовывать разнообразные алгоритмы вывода информации (*независимо* от состояния ввода). Модуль, 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. Перед запуском вывода данных предварительно необходимо передать в модуль требуемые параметры: частота вывода, длина и базовый адрес FIFO буфера вывода. Эту операцию можно выполнить с помощью интерфейсной функции [SET_OUTPUT_PARS\(\)](#). Данные для выдачи на ЦАП берутся из FIFO буфера вывода, который расположен в памяти данных DSP модуля (см. [Драйвер DSP](#)). Поэтому **очень важно**, прежде чем запустить вывод, проинициализировать этот буфер требуемыми данными (например, с помощью интерфейсной функции [PUT_DM_ARRAY\(\)](#)). О формате данных передаваемых в FIFO буфер вывода смотри [Формат слова данных ЦАП](#)). Для передачи данных в модуль уже в процессе работы (т.е. после запуска вывода) следует пользоваться функцией [WriteData\(\)](#).

3.3.3.4.1 Запуск вывода данных

Формат:	BOOL START_WRITE(void)
Назначение:	Данная функция запускает модуль на перманентную (непрерывную) выдачу данных. При этом данные из FIFO буфера вывода начинают последовательно, с заданной частотой, передаваться в ЦАП. Параметры вывода данных предварительно передаются в модуль с помощью интерфейсной функции SET_OUTPUT_PAR(). Также предварительно перед запуском вывода следует проинициализировать FIFO буфер вывода необходимыми начальными значениями с помощью, например, интерфейсной функции PUT_DM_ARRAY(). Подробности о начальной инициализации FIFO буфера вывода и формате данных для ЦАП см. в прилагаемых к модулю примерах, а также см. Формат слова данных ЦАП). Саму же 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.

3.3.3.4.2 Останов вывода данных

Формат:	BOOL STOP_WRITE(void)
Назначение:	Данная функция запрещает модулю выводить данные на ЦАП.
Передаваемые параметры:	нет
Возвращаемое значение:	TRUE – функция выполнена; FALSE – функция не выполнена.

3.3.3.4.3 Установка параметров вывода данных

Формат:	BOOL	SET_OUTPUT_PARS (RTUSB3000::OUTPUT_PARS * const dp)
Назначение:	Данная функция передает в модуль в виде структуры OUTPUT_PARS все необходимые для вывода данных параметры. Использование модулем этой информации при выводе начинается только после выполнения интерфейсной функции START_WRITE() .	
Передаваемые параметры:	• <i>dm</i> – адрес структуры типа OUTPUT_PARS.	
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.	

3.3.3.4.4 Получение текущих параметров вывода данных

Формат:	BOOL	GET_OUTPUT_PARS (RTUSB3000::OUTPUT_PARS * const dp)
Назначение:	Данная функция считывает из модуля всю текущую информацию, которая используется в процессе потоковой выдачи данных.	
Передаваемые параметры:	• <i>dp</i> – адрес структуры OUTPUT_PARS с полученными из модуля текущими параметрами вывода данных.	
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.	

3.3.3.4.5 Передача данных в 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 128 до (1024*1024), а также быть кратна 128. 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 же этой асинхронной 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()*.

!!!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 ЦАП;
- *NumberOfWordsToWrite* – количество отсчетов (минимум – ??, 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](#)).

3.3.3.5. Функции для работы с внешними цифровыми линиями

3.3.3.5.1 Разрешение выходных цифровых линий

Формат:	BOOL <i>ENABLE_TTL_OUT(BOOL EnableTtlOut)</i>
Назначение:	Данная интерфейсная функция позволяет разрешать/запрещать <i>выходные</i> линии внешнего <u>цифрового разъёма</u> , т.е. переводит их в третье (высокоимпедансное) состояние и обратно. Непосредственно после подачи на модуль питания выходные цифровые линии находятся в третьем состоянии.
Передаваемые параметры:	• <i>EnableTtlOut</i> – флажок, позволяющий (<i>TRUE</i>) либо запрещающий (<i>FALSE</i>) использование цифровых выходных линий.
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.

3.3.3.5.2 Чтение внешних цифровых линий

Формат:	BOOL <i>TTL_IN(WORD * const TtlIn)</i>
Назначение:	Данная интерфейсная функция осуществляет чтение состояния 10 ^{ти} входных цифровых линий модуля.
Передаваемые параметры:	• <i>TtlIn</i> – переменная, в младших 10^{ти} битах содержащая побитовое состояние входных цифровых линий модуля.
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.

3.3.3.5.3 Вывод на внешние цифровые линии

Формат:	bool <i>TTL_OUT(WORD TtlOut)</i>
Назначение:	Данная интерфейсная функция осуществляет установку 8 ^{ми} выходных цифровых линий модуля в соответствии с 8 ^{мью} младшими битами передаваемого параметра <i>TtlOut</i> . Работа с цифровыми выходами предварительно должна быть разрешена с помощью интерфейсной функции <u>ENABLE_TTL_OUT()</u> .
Передаваемые параметры:	• <i>TtlOut</i> – переменная, содержащая побитовое состояние устанавливаемых выходных цифровых линий модуля.
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.

3.3.3.6. Функции для работы с пользовательским ППЗУ

На модуле *USB3000* реализовано пользовательское ППЗУ емкостью 256 ячеек×8 бит. 129 ячеек данного ППЗУ зарезервировано для целей пользователей.

3.3.3.6.1 Разрешение записи в ППЗУ

Формат:	BOOL	<i>ENABLE_FLASH_WRITE</i> (<i>BOOL EnableFlashWrite</i>)
Назначение:	Данная функция разрешает выполнить с помощью интерфейсной функции PUT_FLASH() процедуру записи данных в пользовательское ППЗУ модуля.	
Передаваемые параметры:	<i>EnableFlashWrite</i> – управление разрешением записью в ППЗУ.	
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.	

3.3.3.6.2 Запись данных в ППЗУ

Формат:	BOOL	<i>PUT_FLASH</i> (<i>RTUSB3000::FLASH * const fi</i>)
Назначение:	Данная функция осуществляет запись содержимого структуры типа FLASH в пользовательское ППЗУ модуля.	
Передаваемые параметры:	<i>fi</i> – содержимое полей данной структуры передается в пользовательское ППЗУ.	
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.	

3.3.3.6.3 Чтение данных из ППЗУ

Формат:	BOOL	<i>GET_FLASH</i> (<i>RTUSB3000::FLASH * const fi</i>)
Назначение:	Данная функция осуществляет чтение содержимого всего пользовательского ППЗУ модуля в структуру типа FLASH .	
Передаваемые параметры:	<i>fi</i> – в поля данной структуры передается содержимое пользовательского ППЗУ.	
Возвращаемое значение:	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция не выполнена.	

3.4. Драйвер DSP

Данная глава предназначена для программистов, владеющих языком программирования DSP и собирающихся корректировать штатный драйвер DSP под свои специфические задачи. Штатный драйвер DSP, поставляемый вместе с модулем USB3000, подходит для большинства стандартных задач по сбору данных. Потребность в написании собственного драйвера DSP может возникнуть, например, если необходимо реализовать в DSP специфическую обработку данных, разгрузив при этом процессор ПК.

Программист при желании может напрямую работать с памятью DSP модуля (и программ, и данных), используя соответствующие интерфейсные [функции](#), которые обеспечивают доступ как к отдельным ячейкам памяти, так и к целым массивам. Эта возможность позволяет программисту работать с модулем, непосредственно обращаясь к соответствующим ячейкам либо памяти программ, либо памяти данных DSP. Карта распределения памяти DSP для штатного драйвера приведена на рисунке ниже:

Память программ	адрес	Память данных	адрес
Код штатного драйвера DSP	0x3FFF	32 управляющих регистра DSP	0x3FFF
			0x3FE0
		не используется	0x3FDF 0x3F80
		FIFO буфер вывода	0x3F7F 0x3000
	0x0400	FIFO буфер ввода	0x2FFF
Переменные	0x03FF		
	0x0030		
Таблица прерываний	0x002F 0x0000		0x0000

Ниже 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.

Таблица 6. 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 USB3000 должна быть строчка "USB3000"
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 готовности 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 7 .
D_CONTROL_TABLE_LENGTH	0x50	Размер управляющей таблицы (максимум 128 логических каналов). По умолчанию – 0x8 .
D_INPUT_RATE	0x53	Переменная, задающая код частоты ввода данных в РС из модуля.
D_INPUT_ENABLED	0x56	Переменная, показывающая текущее состояние режима ввода данных (активное или нет).
D_INPUT_FIFO_BASE_ADDRESS	0x57	Базовый адрес FIFO буфера ввода данных. Всегда устанавливается драйвером DSP равным 0x0 .
D_INPUT_FIFO_LENGTH	0x58	Требуемая длина FIFO буфера ввода. По умолчанию равна 0x3000 .
D_CUR_INPUT_FIFO_LENGTH	0x59	Текущая длина FIFO буфера ввода. По умолчанию равна 0x3000 .

Название переменной	Адрес Hex	Назначение переменной
D_CORRECTION_ENABLED	0x5B	Переменная запрещающая (0) либо разрешающая (1) корректировку данных с аналоговых каналов при помощи корректировочных коэффициентов. По умолчанию равна 0x0 .
D_INPUT_TYPE	0x5C	Задаёт тип ввода информации: 0x0 – с АЦП, 0x1 – с цифровых линий.
D_SYNCHRO_TYPE	0x5D	Переменная, задающая тип синхронизации при вводе данных: цифровая или ее отсутствие.
D_OUTPUT_RATE	0x72	Переменная, задающая код частоты вывода данных из модуля.
D_OUTPUT_ENABLED	0x73	Переменная, показывающая текущее состояние режима вывода данных (активное или нет).
D_OUTPUT_FIFO_BASE_ADDRESS	0x74	Базовый адрес FIFO буфера вывода данных. Всегда устанавливается драйвером DSP равным 0x3000 .
D_OUTPUT_FIFO_LENGTH	0x75	Требуемая длина FIFO буфера ввода. По умолчанию равна 0x0F80 .
D_CUR_OUTPUT_FIFO_LENGTH	0x76	Текущая длина FIFO буфера вывода. По умолчанию равна 0x0F80 .
D_ENABLE_TTL_OUT	0x7D	Данная переменная запрещает (0x0) либо разрешает (0x1) использование выходных цифровых линий.
D_TTL_OUT	0x7E	Слово (младшие 8 ^{Мб} бит), в котором побитово хранятся состояния 8 ^{Мн} выходных цифровых линий для их выставления по команде C TTL OUT
D_TTL_IN	0x7F	Слово (младшие 10 ^{Тб} бит), в котором после выполнения команды C TTL IN побитово хранятся значения 10 ^{Тн} входных цифровых линий.
D_ADC_SCALE	0x80	Массив с 8 ^{МбЮ} коэффициентами, используемыми для корректировки масштаба вводимых с АЦП данных. По умолчанию все равны 0x8000 .
D_ADC_ZERO	0x88	Массив с 8 ^{МбЮ} коэффициентами, используемыми для корректировки смещения нуля вводимых с АЦП данных. По умолчанию все равны 0x0 .

Название переменной	Адрес Hex	Назначение переменной
D_CONTROL_TABLE	0x100	Управляющая таблица, содержащая последовательность логических номеров каналов (максимум 128 элементов). В соответствии с ней драйвер DSP производит последовательный циклический ввод данных. Размер этой таблицы задается переменной <u>D_CONTROL_TABLE LENGHT</u> . По умолчанию – { 0, 1, 2, 3, 4, 5, 6, 7}

Драйвер DSP работает по так называемому принципу команд. Т.е. управление работой драйвера происходит следующим образом. Сначала из ПК в модуль по адресу переменной [D_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 [D_COMMAND](#) и выполняет надлежащие действия в соответствии с этой командой. Ниже представлены все команды, используемые в штатном драйвере DSP.

Таблица 7. Команды штатного драйвера DSP

Название команды	Номер команды	Назначение	Используемые переменные
C_TEST	0	Проверка загрузки модуля и его работоспособности.	D_TEST_INTR_VAR
C_START_READ	1	Разрешение ввода данных в PC из модуля.	_____
C_STOP_READ	2	Останов ввода данных в PC из модуля.	_____
C_READ_KADR	3	Зарезервировано	_____
C_READ_SAMPLE	4	Зарезервировано	_____
C_START_WRITE	5	Разрешение вывода данных из PC в модуль.	_____
C_STOP_WRITE	6	Останов вывода данных из PC в модуль.	_____
C_WRITE_SAMPLE	7	Зарезервировано	_____

Название команды	Номер команды	Назначение	Используемые переменные
C_ENABLE_TTL_OUT	8	Управление работоспособностью выходных линий.	L_ENABLE_TTL_OUT
C_TTL_IN	9	Считывание состояния 10 ^{ти} внешних цифровых входных линий.	D_TTL_IN
C_TTL_OUT	10	Управление 8 ^{мью} внешними цифровыми выходными линиями.	D_TTL_OUT

В штатном драйвере DSP программно организовано два циклических FIFO буфера: для **ввода** данных в ПК из модуля (АЦП, логический анализатор) и для **вывода** данных из ПК в модуль (ЦАП). Передача данных из FIFO буфера **ввода** в ПК производится порциями по [D_INPUT_FIFO_LENGTH/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 [D_OUTPUT_FIFO_LENGTH/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.