Оглавление.

Контакты:
http://www.R-Technology.ru
Info@R-Technology.ru - Общие вопросы
Sales@R-Technology.ru - Отдел продаж
Support@R-Technology.ru - Техническая поддержка

1. Штатное программное обеспечение модуля USB3000.

1.1. Введение

Настоящий документ 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++, поскольку он является одним из самых доступных, а также достаточно широко распространенных и применяемых языков.

Штатное 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 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.

1.2. Используемые термины и форматы данных

1.2.1. Используемые термины

Название	Пояснение
InputRate	Частота ввода данных (частота работы АЦП или частота опроса цифровых линий для режима Логический анализатор) в *кГц*
ChannelRate	Частота работы аналогового канала в *кГц*
InterKadrDelay	Межкадровая задержка в *млс*
OutputRate	Частота вывода данных (частота работы ЦАП) в *кГц*
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
ControlTableLength	Длина управляющей таблицы
Address	Адрес ячейки в памяти программ или данных DSP модуля

1.2.2. Форматы данных

1.2.2.1. 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 1. Соответствие кода АЦП напряжению на аналоговом входе

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

1.2.2.2. 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 2. 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.

Код, выдаваемый модулем на 12[ти] битный ЦАП, связан с выходным напряжением, устанавливаемым на аналоговом разъеме, в соответствии со следующей таблицей:

Таблица 3. Соответствие кода ЦАП напряжению на выходе

Код	Напряжение, В
+4000	+5.0
+2000	0
0	-5.0

1.2.2.3. Логический номер канала АЦП

На модуле USB3000 для управления работой входного аналогового каскада определяется такой параметр, как $4^{х}$ 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 $10^{ти}$ 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.)

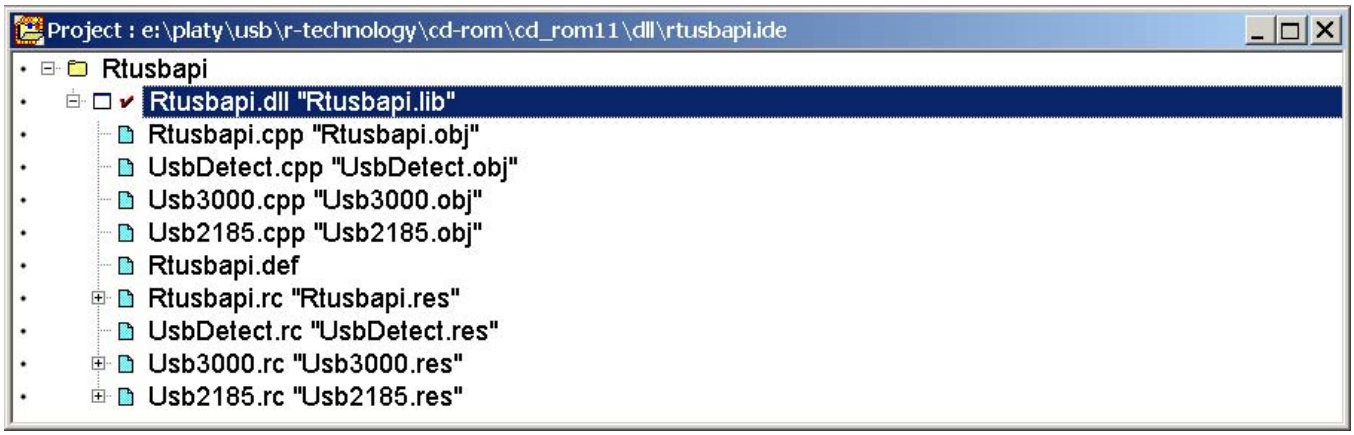
Если библиотека Rtusbapi обнаружит в управляющей таблице хоть один *логический номер канала*, содержащий флаг ввода с цифровых линий, то модуль автоматически переводится в режим «логический анализатор».

1.3. Штатная библиотека

В комплект поставки 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.

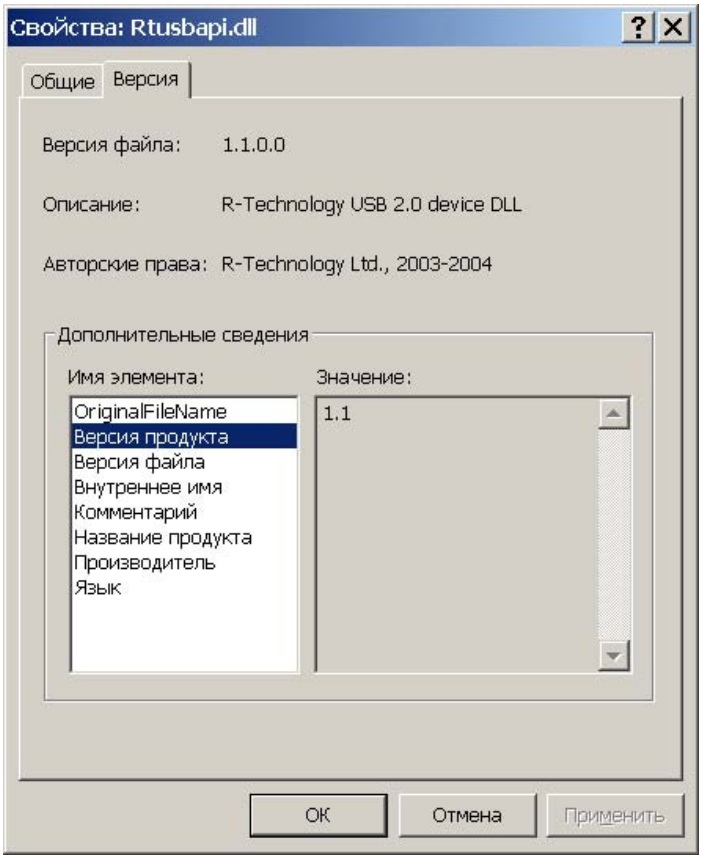
1.3.1. Общий подход к работе со штатной библиотекой

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 прочитать номер версии DLL библиотеки (две старшие цифры):



Исходные тексты самой DLL библиотеки Вы можете найти на нашем CD–ROM'е 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\BCB5\` - для среды **Borland C++ Builder v.5.0**

`\Examples\D6` – для среды **Delphi 6.0**;

`\Examples\LabVIEW71` – для среды **LabView 7.1**.

`\Examples\LabWindows71` – для среды **LabWindows 7.1**

`\Examples\MSVB6` – для среды **MS Visual Basic 6.0**.

`\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`.

1.3.1.1. Пример использования функций штатной библиотеки.

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 функции 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.** Задачей DSP является управление всей устанавливаемой на модуле периферией (АЦП, ЦАП, цифровые линии и т.д.), а также, при необходимости, первичная обработка информации. Для того чтобы “оживить” DSP, т.е. заставить работать по требуемому Вам алгоритму, во внутреннюю память DSP надо записать (загрузить) управляющую программу – драйвер DSP (файл `USB3000.rtd`, входящий в штатное ПО). 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
    }
    // 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->ReleaseInstance())
    {
      printf("Не выполнена функция ReleaseInstance()!\n");
      return 1;                 //выйдем из программы с ошибкой
    }

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

1.3.2. Описание структур штатной библиотеки

1.3.2.1. 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
    WORD InputClockSource;          // источник тактовых импульсов для ввода данных
    BOOL InputType;                 // тип вводимых с модуля даных (АЦП или ТТЛ)
    WORD SynchroType;               // тип синхронизации вводимых с модуля даных
    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
};
```

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()`.

1.3.2.2. 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
};
```

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()`.

1.3.2.3. Структура RTUSB3000::FLASH

Пользовательское ППЗУ состоит из двух областей:

1. служебная область;
2. область пользователя (поле `ReservedByte`).

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
    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 для целей пользователей
};
```

Данная структура используется в интерфейсных функциях, которые работают с пользовательским ППЗУ: `PUT FLASH()` и `GET FLASH()`. **!!!ВНИМАНИЕ!!!** В целях предотвращения порчи или полной потере системной области пользовательского ППЗУ необходимо быть **КРАЙНЕ** аккуратным при работе с функцией `PUT FLASH()`.

1.3.2.4. Структура RTUSB3000::DSP_INFO

Структура *RTUSB3000::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 “USB3000”
    BYTE Label[6];            // разработчик драйвера DSP "rtech"
    BYTE DspMajor;            // старший байт версии драйвера DSP
    BYTE DspMinor;            // младший байт версии драйвера DSP
};
```

Данная структура используется в интерфейсной функции `GET DSP INFO`.

1.3.3. Описание функций штатной библиотеки

1.3.3.1. Функции общего характера

1.3.3.1.1. 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 Примере использования функций.
Передаваемые параметры: нет.
Возвращаемое значение: номер версии DLL библиотеки.

1.3.3.1.2. Получение указателя на интерфейс модуля

Формат: **LPVOID** ***RtCreateInstance****(PCHAR const DeviceName)*
Назначение: Данная функция должна обязательно вызываться в начале каждой пользовательской программы, работающей с модулями *USB3000*. 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 Примере использования функций.
Передаваемые параметры: *DeviceName* – строка с названием устройства (для данного модуля это – "USB3000").
Возвращаемое значение: В случае успеха — указатель на интерфейс, иначе — **NULL**.

1.3.3.1.3. Завершение работы с модулем

Формат: **BOOL** ***ReleaseInstance****(void)*
Назначение: Данная интерфейсная функция реализует корректное высвобождение интерфейсного указателя, проинициализированного с помощью интерфейсной функции RtCreateInstance(). 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 Примере использования функций. В библиотеках версии 1.4. и ниже данная функция называлась ***ReleaseDevice()***. Для совместимости при использовании новых библиотек функцию можно вызывать как под именем ***ReleaseDevice()***, так и под именем ***ReleaseInstance()***.
Передаваемые параметры: нет.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

1.3.3.1.4. Инициализация доступа к модулю

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

Назначение:

С программной точки зрения, подсоединенный к компьютеру модуль *USB3000* можно рассматривать как устройство, подключённое к некоему виртуальному слоту с индивидуальным номером. Основное же назначение данной интерфейсной функции – определить, находится ли устройство в заданном виртуальном слоте. Если функция ***OpenDevice()*** успешно выполнилась для заданного виртуального слота, то можно быть уверенным, что к этому слоту подключён именно модуль *USB3000*. Далее 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 Примере использования функций.

Передаваемые параметры:

VirtualSlot – номер виртуального слота, к которому, как предполагается, подключен модуль *USB3000*.

Возвращаемое значение:

TRUE – устройство *USB3000* находится в выбранном виртуальном слоте;
FALSE –устройства *USB3000* в выбранном виртуальном слоте нет (может, стоит попробовать другой номер виртуального слота).

1.3.3.1.5. 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 ReleaseInstance().

Передаваемые параметры: нет.

Возвращаемое значение:

TRUE – функция успешно выполнена;
FALSE– функция не выполнена.

1.3.3.1.6. Получение названия модуля

Формат: **BOOL** ***GetModuleName***(*PCHAR const ModuleName*)
Назначение: Данная интерфейсная функция позволяет получить название модуля, подключенного к выбранному виртуальному слоту. 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 Примере использования функций.
Передаваемые параметры: *ModuleName* – возвращается строка, не менее 11 символов, с названием модуля (в нашем случае это должна быть строка "USB3000").
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

1.3.3.1.7. 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.

1.3.3.1.8. 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.

1.3.3.1.9. Получение версии BIOS модуля

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

1.3.3.1.10. Загрузка драйвера DSP

Формат: **BOOL** ***LOAD_DSP****(PCHAR const 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 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 Примере использования функций.
Передаваемые параметры: *FileName* – 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: *TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

1.3.3.1.11. 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 Примере использования функций.
Передаваемые параметры: нет
Возвращаемое значение: *TRUE* – драйвер DSP успешно загружен и функционирует надлежащим образом, *FALSE* – произошла ошибка загрузки или функционирования драйвера DSP.

1.3.3.1.12. 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 Примере использования функций.
Передаваемые параметры: *DspInfo* – адрес структуры типа DSP INFO.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

1.3.3.1.13. Сброс DSP модуля

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.

1.3.3.1.14. Передача номера команды в драйвер DSP

Формат: **BOOL** ***SEND_COMMAND****(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 Примере использования функций.
Передаваемые параметры: *Command* – номер команды, передаваемый в драйвер DSP.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

1.3.3.1.15. Получение описания ошибок выполнения функций

Формат: **int** ***GetLastErrorString*** (*LPTSTR const lpBuffer, 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.

1.3.3.2. 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 (например, к переменным драйвера DSP). При этом для работы с этими функциями, в принципе, совсем не требуется загруженного в модуль драйвера DSP.

1.3.3.2.1. 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 $16^{ти}$ битное слово.	
Возвращаемое значение:	*TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

1.3.3.2.2. 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.

1.3.3.2.3. 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 $16^{ти}$ битного слова.	
Возвращаемое значение:	*TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

1.3.3.2.4. 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.

1.3.3.2.5. 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.

1.3.3.2.6. 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.

1.3.3.2.7. 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.

1.3.3.2.8. 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.

1.3.3.2.9. 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 ***программ*** DSP модуля (см. Драйвер 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.

1.3.3.2.10. 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 ***программ*** DSP модуля (см. Драйвер 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.

1.3.3.3. 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`.

1.3.3.3.1. 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: *TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

1.3.3.3.2. Останов ввода данных

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

1.3.3.3.3. 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). Непосредственная запись в модуль этой информации осуществляется **только** при выполнении интерфейсной функции START READ(), поэтому до выполнения функции START READ() нельзя использовать функцию GET INPUT PARS.
Передаваемые параметры: *ap* – адрес структуры типа INPUT_PARS с параметрами ввода данных.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE*– функция не выполнена.

Поле *ap->CorrectionEnabled* позволяет модулю по желанию пользователя осуществлять корректировку получаемых с АЦП данных. В этом случае из модуля в PC 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->InputClockSource* определяет, под управлением какого источника синхроимпульсов будет осуществляться ввод данных (запуск АЦП или чтение цифровых линий):

ap->InputClockSource=0 – сбор данных тактируется внутренним тактовым генератором модуля;

ap->InputClockSource=1 – сбор данных тактируется внешним тактовым генератором.

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->InputRate* и *ap->InterKadrDelay* должны находиться частота сбора данных InputRate (частота работы АЦП или частота опроса цифровых линий в кГц) и межкадровая задержка InterKadrDelay (зарезервирована, должна ровняться 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 InputRate и InterKadrDelay не являются непрерывными величинами, а принадлежат некоему дискретному ряду. Так, в общем виде, частота работы АЦП определяется по следующей формуле: InputRate=$\mathbf{F}_{clockout}$/(2*(**N**+1)), где $\mathbf{F}_{clockout}$ – тактовая частота, установленного на модуле DSP, равная 72000 кГц, **N** – целое число. Поэтому данная функция просто вычисляет ближайшую к задаваемой величину InputRate, передает ее в модуль (в виде целого числа **N**) и возвращает ее значение в поле *ap->InputRate*. В случае, когда для тактирования сбора данных используется внешний источник импульсов (*ap->InputClockSource=1),* значение *ap->InputRate* уже не влияет на частоту сбора данных.

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. **Внимание!!!** Для данного модуля эта величина может находиться в диапазоне от 0x800 (2048) до 0x3000 (12288), а также быть обязательно кратной 0x800 (2048). Единственное разрешенное значение вне этого ряда - 0x400 (1024). Передача данных из FIFO буфера ввода модуля в PC производится порциями по *ap->InputFifoLength/2* отсчетов (по мере их готовности).

Поля *ap->AdcOffsetCoef[]* и *ap->AdcScaleCoef[]*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.

1.3.3.3.4. 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: *ap* – адрес структуры типа INPUT_PARS с полученными из модуля текущими параметрами ввода данных.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE* – функция не выполнена.

1.3.3.3.5. 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).

1.3.3.3.6. Однократный ввод данных

Формат: **BOOL** ***READ_SAMPLE****(WORD Channel, SHORT * const Sample)*

Назначение:

Данная функция устанавливает заданный логический номер канала и осуществляет однократный ввод данных с этого канала (см. Логический номер канала АЦП). Эта функция удобна для осуществления асинхронного ввода данных с требуемого канала. Предварительно, с помощью штатной интерфейсной функции SET INPUT PARS(), можно разрешить корректировку данных с заданного канала АЦП. Более подробно смотри исходные тексты примера из директории \Example\BC5\AdcSample.

Передаваемые параметры:

Channel – требуемый логический номер канала;
Sample – отсчёт с заданного логического канала *Channel*.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

1.3.3.3.7. Однократный ввод кадра данных

Формат: **BOOL** ***READ_KADR****(SHORT * const Data)*

Назначение:

Эта функция удобна для осуществления асинхронного ввода целого кадра данных с требуемых логических каналов (см. Логический номер канала АЦП). Такие параметры сбора кадра отсчетов, как разрешение корректировки данных с АЦП, количество опрашиваемых логических каналов, частота ввода данных и т.д., должны предварительно быть заданы с помощью штатной интерфейсной функции SET INPUT PARS() (при этом информация о синхронизации ввода данных никак не используется, т.е. просто игнорируется). Массив *Data* необходимой длины для получаемых с модуля данных следует заранее определить. Более подробно смотри исходные тексты примера из директории \Example\BC5\AdcKadr.

Передаваемые параметры:

Data – указатель на массив, в который складываются получаемые с модуля данные.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

1.3.3.4. 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().

1.3.3.4.1. 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.

1.3.3.4.2. 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.

1.3.3.4.3. 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. Непосредственная запись в модуль этой информации осуществляется **только** при выполнении интерфейсной функции `START_WRITE()`, поэтому до выполнения функции `START_WRITE()` нельзя использовать функцию `GET_OUTPUT_PARS`.
Передаваемые параметры: *dm* – адрес структуры типа `OUTPUT_PARS`.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE* – функция не выполнена.

Поле *dp->OutputRate* задает частоту вывода `OutputRate` в *кГц*. **После** выполнения данной интерфейсной функции в данном поле находится **реально установленное** значение частоты вывода данных. Это происходит вследствие того, что реальные значения `OutputRate` не являются непрерывной величиной, а принадлежат некоему дискретному ряду. Так, в общем виде, частота вывода определяется по следующей формуле: `OutputRate`=$\mathbf{F}_{clockout}$/(9*(**N**+1)), где $\mathbf{F}_{clockout}$ – тактовая частота установленного на модуле DSP равная 72000 кГц, **N** – целое число. Поэтому данная функция просто вычисляет ближайшую к задаваемой дискретную величину `OutputRate`, 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. **Внимание!!!** Для данного модуля эта величина может находиться в диапазоне от 0x80 (128) до 0xF80 (3968), а также также быть обязательно кратной 0x80(128). Передача (‘*подкачка*’) новых данных из PC в FIFO буфер ЦАП модуля производится порциями по *dp->OutputFifoLength/2* отсчетов (по мере их необходимости).

1.3.3.4.4. 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: *dp* – адрес структуры `OUTPUT_PARS` с полученными из модуля текущими параметрами вывода данных.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE* – функция не выполнена.

1.3.3.4.5. Передача данных в модуль

Формат: **BOOL** ***WriteData****(SHORT * const Buffer,*
*DWORD * consNumberOfWordsToWrite,*
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 – 128, 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).

1.3.3.4.6. Однократный вывод данных

Формат: **BOOL** ***WRITE_SAMPLE****(WORD Channel, SHORT * const Sample)*
Назначение: Данная функция позволяет осуществлять однократный вывод на указанный канал ЦАП *Channel* напряжения в соответствии со значением *Sample* (код –2048 соответствует –5В на выходе ЦАП, код 2047 соответствует +5В на выходе ЦАП). Более подробно смотри исходные тексты примера из директории `\Example\BC5\DacSample`.
Передаваемые параметры: *Channel* – требуемый номер канала ЦАП модуля (**0** или **1**); *Sample* – устанавливаемое значение напряжения в кодах ЦАП (от –2048 до 2047).
Возвращаемое значение: *true* – функция успешно выполнена; *false*– функция не выполнена.

1.3.3.5. Функции для работы с внешними цифровыми линиями

1.3.3.5.1. Разрешение выходных цифровых линий

Формат: **BOOL** ***ENABLE_TTL_OUT****(BOOL EnableTtlOut)*
Назначение: Данная интерфейсная функция позволяет разрешать/запрещать *выходные* линии внешнего цифрового разъёма, 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: *EnableTtlOut* – флажок, позволяющий (*TRUE*) либо запрещающий (*FALSE*) использование цифровых выходных линий.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE* – функция не выполнена.

1.3.3.5.2. Чтение внешних цифровых линий

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

1.3.3.5.3. Вывод на внешние цифровые линии

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

1.3.3.6. 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.

1.3.3.6.1. Разрешение записи в ППЗУ

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

1.3.3.6.2. Запись данных в ППЗУ

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

1.3.3.6.3. Чтение данных из ППЗУ

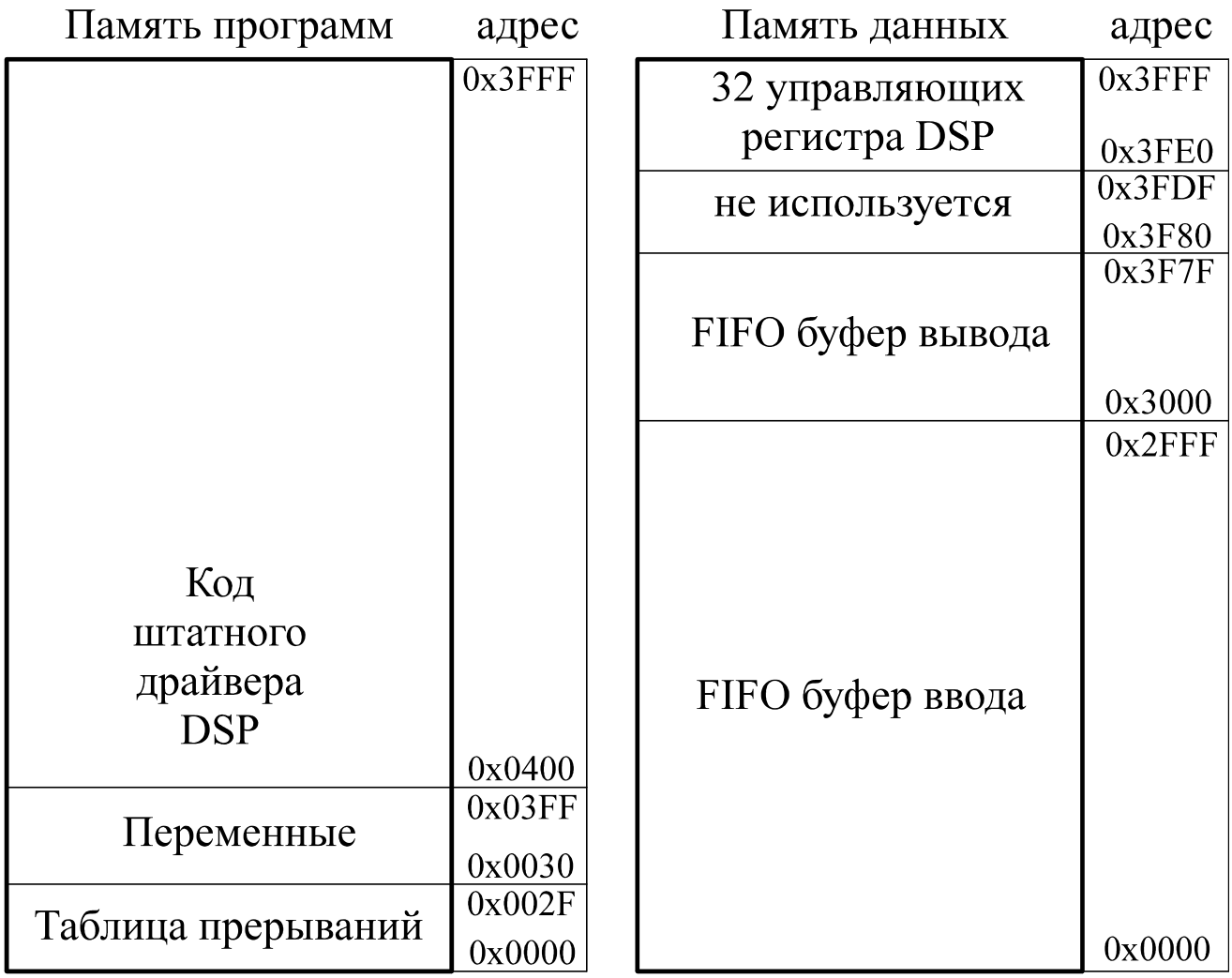
Формат: **BOOL** ***GET_FLASH****(RTUSB3000::FLASH * const fi)*
Назначение: Данная функция осуществляет чтение содержимого всего пользовательского ППЗУ модуля в структуру типа *FLASH*.
Передаваемые параметры: *fi* – в поля данной структуры передаётся содержимое пользовательского ППЗУ.
Возвращаемое значение: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

2. Низкоуровневое программирование модуля USB3000.

2.1. 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:



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 4. Переменные штатного драйвера DSP

Название переменной	Адрес Hex	Назначение переменной
D_PROGRAM_BASE_ ADDRESS	0x30	Адрес в памяти программ, с которого 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 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 5.
D_CONTROL_TABLE_ LENGHT	0x50	Размер управляющей таблицы (максимум 128 логических каналов). По умолчанию – **0x8**.
D_INPUT_SAMPLE	0x51	Данная переменная используется при однократном вводе данных, храня считанное значение.
D_INPUT_CHANNEL	0x52	Данная переменная используется при однократном вводе данных, задавая логический номер канала.
D_INPUT_RATE	0x53	Переменная, задающая код частоты ввода данных в PC 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**.

Название переменной	Адрес Hex	Назначение переменной
D_CUR_INPUT_FIFO_ LENGTH	0x59	Текущая длина FIFO буфера ввода. По умолчанию равна **0x3000.**
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	Задает 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_SAMPLE	0x70	Данная переменная используется при однократном выводе данных, храня выводимое значение.
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 $8^{\text{мь}}$ бит), в котором побитово хранятся состояния $8^{\text{ми}}$ выходных цифровых линий для их выставления по команде C_TTL_OUT
D_TTL_IN	0x7F	Слово (младшие $10^{\text{ть}}$ бит), в котором после выполнения команды C_TTL_IN побитово хранятся значения $10^{\text{ти}}$ входных цифровых линий.
D_ADC_SCALE	0x80	Массив с $8^{\text{мью}}$ коэффициентами, используемыми для корректировки масштаба вводимых с АЦП данных. По умолчанию все равны **0x8000.**

Название переменной	Адрес Hex	Назначение переменной
D_ADC_ZERO	0x88	Массив с 8мью коэффициентами, используемыми для корректировки смещения нуля вводимых с АЦП данных. По умолчанию все равны **0x0**.
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 **D_CONTROL_TABLE_LENGHT** . 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 соответствие с этой командой. Ниже представлены все команды, используемые в штатном драйвере DSP.

Таблица 5. 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	Ввод кадра отсчётов	———
C_READ_SAMPLE	4	Ввод одного отсчёта	**D_INPUT_SAMPLE** **D_INPUT_CHANNEL**
C_START_WRITE	5	Разрешение вывода данных из PC в модуль.	———
C_STOP_WRITE	6	Останов вывода данных из PC в модуль.	———

Название команды	Номер команды	Назначение	Используемые переменные
C_WRITE_SAMPLE	7	Вывод одного отсчёта	**D_OUTPUT_SAMPLE**
C_ENABLE_ TTL_OUT	8	Управление работоспособностью выходных линий.	**L_ENABLE_TTL_OUT**
C_TTL_IN	9	Считывание состояния 10ти внешних цифровых входных линий.	**D_TTL_IN**
C_TTL_OUT	10	Управление 8мью 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.