L-CARD

Устройства для мобильных систем

E-154

Внешний модуль АЦП/ЦАП/ТТЛ на шину **USB 1.1**

Встроенный 32-битный процессор ARM, частота 48 МГц

Библиотека *Lusbapi 3.2*. Windows 98/2000/XP/Vista

Руководство программиста

Москва. Март 2008 г.
Rev. A1

ЗАО *«Л-КАРД»*,
117105, г. Москва, Варшавское шоссе, д. 5, корп. 4, стр. 2.

тел. (495) 785-95-25
факс (495) 785-95-14

Адреса в Интернет:

WWW: *www.lcard.ru*
FTP: *ftp.lcard.ru*

E-Mail:

Общие вопросы: *lcard@lcard.ru*
Отдел продаж: *sale@lcard.ru*
Техническая поддержка: *support@lcard.ru*
Отдел кадров: *job@lcard.ru*

Представители в регионах:

Украина:	“ХОЛИТ Дэйта Системс, Лтд”	*www.holit.com.ua*	380 (44) 241-67-54
Санкт-Петербург:	ЗАО “АВТЭКС Санкт-Петербург”	*www.autex.spb.ru*	(812) 567-7202
Санкт-Петербург:	Компания "Ниеншанц-Автоматика"	*www.nnz-ipc.ru*	(812) 326-59-24
Новосибирск:	ООО “Сектор Т”	*www.sector-t.ru*	(3832) 22-76-20
Екатеринбург:	Группа Компаний АСК	*www.ask.ru*	(343) 371-44-44
Екатеринбург:	ООО “Авеон”	*www.aveon.ru*	(343) 381-75-75
Казань:	ООО “Шатл”	*shuttle@kai.ru*	(8432) 38-16-00
Самара:	"АСУ-Самара"	*prosoft-s@jiguli.ru*	(846)-998-29-01

Оглавление

1. Введение

Данный раздел описания предназначен для программистов, собирающихся писать свои собственные программы в среде *Windows'98/2000/XP/Vista* для работы с модулями *E-154*. Предварительно очень рекомендуется ознакомиться с *Руководством пользователя*, где можно найти информацию по подключению входных сигналов, распиновке внешних разъёмов, о характерных неисправностях и т.п.

В качестве базового языка при написании штатного программного обеспечения нами был выбран язык C++, а конкретнее, старый надёжный **Borland C++ 5.02**. Для модуля *E-154* фирма **ЗАО "Л-Кард"** поставляет драйвер устройства, готовую динамически подключаемую библиотеку `Lusbapi`, а также целый ряд законченных примеров. Библиотека и примеры поставляются вместе с исходными текстами. В библиотеку мы попытались включить множество разнообразных полезных функций для облегчения пользователю процедуры написания собственных программ по управлению модулем *E-154*. Данная библиотека позволяет использовать практически все возможности модуля, не вдаваясь в тонкости их низкоуровневого программирования.

Поскольку модуль *E-154* разрабатывался с главной целью, которая заключалась в обеспечении ввода в компьютер аналоговой информации, штатная библиотека содержит ряд функций, позволяющих организовывать многоканальный *непрерывный потоковый* сбор данных на частотах вплоть до 120 кГц. Вывод же аналоговой (ЦАП) и ввод-вывод цифровой информации реализован только в однократном, и поэтому достаточно медленном, режиме. Мы надеемся, что описываемая ниже библиотека упростит и ускорит написание Ваших собственных приложений.

Весь пакет штатного программного обеспечения для модуля *E-154* в среде *Windows'98/2000/XP/Vista* находится на прилагаемом к модулю фирменном CD-ROM в директории `\USB\Lusbapi`. **!!!ВНИМАНИЕ!!!** Далее по тексту данного описания все директории, касающиеся непосредственно модуля *E-154*, указаны относительно неё. Также весь пакет штатного программного обеспечения можно скачать с нашего сайта *www.lcard.ru* из раздела *"Модуль E-154"*. Там из подраздела "Программное обеспечение" следует выбрать самораспаковывающийся архив `lusbapi`***XY***`.exe`, где ***X.Y*** обозначает номер версии программного обеспечения. На момент написания данного руководства этот архив имеет имя *lusbapi32.exe*.

2. Общие сведения

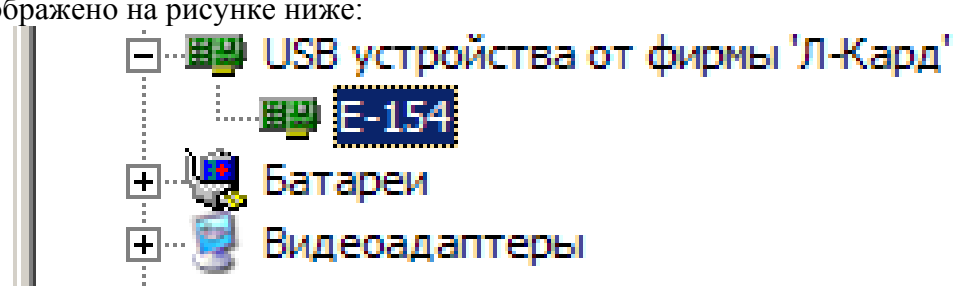
2.1. Подключение модуля к компьютеру

Собственно сама процедура аппаратного подключения модуля *E-154* к компьютеру достаточно тривиальна: необходимо просто соединить **USB** разъём 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 **USB** шины: *Windows'98\2000\XP\Vista*. Причем спецификацией **USB** предусматривается как '*горячее*' подключение или отключение устройств к/от шины **USB**, так и включение компьютера с уже подключенными устройствами **USB**.

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 *E-154* к компьютеру операционная система сама должна запросить файлы драйвера для впервые подключаемого устройства. Тогда ей необходимо указать *inf*-файл с фирменного CD-ROM: `\DRV\Lusbapi.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*'), т.е. *Windows* необходимо проинициализировать подключенное устройство. Такая

процедура *нумерации* устройств, подключенных к шине **USB**, осуществляется динамически по мере их подключения или отключения без какого-либо вмешательства пользователя или клиентского программного обеспечения. Во время выполнения процесса *нумерации* индикатор "*GOOD LINK*" (красный светодиод расположенный рядом с **USB** разъёмом модуля) должен непрерывно мигать, а после успешного окончания нумерации загореться красным цветом. 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 в появившемся разделе *"L-Card USB devices"* (*"Устройства USB от фирмы 'Л-Кард'"*) должно появиться устройство *"E-154 Module"* (*"Модуль E-154"*), как это, например, отображено на рисунке ниже:

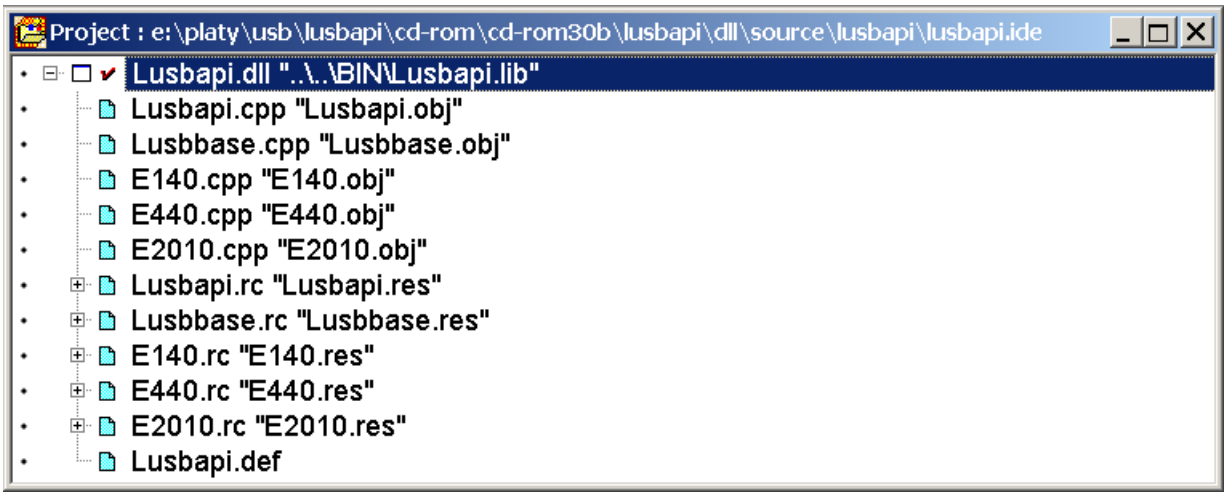


При дальнейшей работе с модулем *E-154* операционная система уже будет знать, где находится драйвер для данного типа устройства, и будет подгружать его автоматически по мере необходимости.

Кроме того, рекомендуется скопировать файл штатной библиотеки `Lusbapi.dll` в директорию `%SystemRoot%\system32`. Это полезно потому, что *Windows'98/2000/XP/Vista* при необходимости автоматически производит поиск файлов в указанной директории. Хотя, в принципе, штатная библиотека `Lusbapi` может находиться в директории Вашего приложения или в одной из директорий, указанных в переменной окружения PATH.

2.2. Библиотека Lusbapi

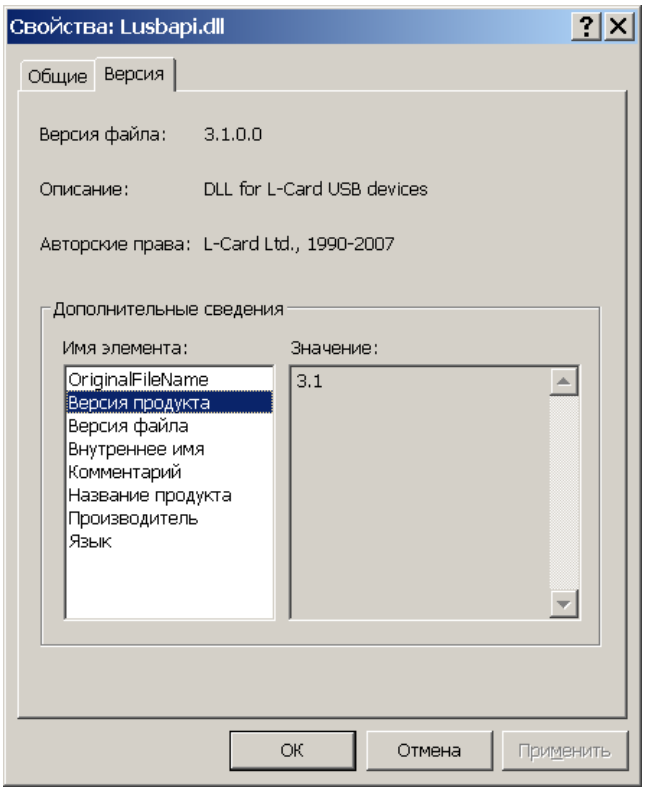
Штатная библиотека `Lusbapi` написана с использованием весьма доступного языка программирования **Borland C++ 5.02**. Кроме модуля *E-154* в библиотеке также осуществлена поддержка модулей типа *E14-440, E14-140* и *E20-10*. Общий вид проекта библиотеки `Lusbapi` в интегрированной среде разработки **Borland C++ 5.02** представлен на рисунке ниже:



Собственно, сама библиотека содержит всего две экспортируемые функции, одна из которых *CreateLInstance()* возвращает указатель на интерфейс модуля *E-154*. В дальнейшем, используя этот указатель, можно осуществлять доступ ко всем интерфейсным функциям DLL библиотеки (см. исходные тексты примеров). **!!!Внимание!!!** Все интерфейсные функции, кроме *ReadData()* строго говоря, не обеспечивают "*потокобезопасную*" работу библиотеки. Поэтому, во избежание недоразумений, в многопоточных приложениях пользователь должен сам организовывать, если

необходимо, корректную синхронизацию вызовов интерфейсных функций в различных потоках (используя, например, критические участки, мьютексы и т.д.).

В сам файл библиотеки `Lusbapi.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: *GetDllVersion()*. Кроме того, оперативно выявить текущую версию библиотеки можно, используя штатные возможности *Windows*. Например, в '*Windows Explorer*' щелкните правой кнопкой мышки над файлом DLL библиотеки `Lusbapi.dll`. Во всплывшем на экране монитора меню следует выбрать опцию '*Properties*' ('*Свойства*'), после чего на появившейся панели выбрать закладку '*Version*' ('*Версия*'). На этой закладке в строчке '*File version*' ('*Версия файла*') можно без труда прочитать текущий номер версии библиотеки. Выглядит это примерно так:



Сам файл штатной библиотеки `Lusbapi.dll` расположен на фирменном CD–ROM в директории `\DLL\BIN`. Её исходные тексты можно найти в директории `\DLL\Source\Lusbapi`. Заголовочные файлы хранятся в директории `\DLL\Include`, а библиотеки импорта и модули объявления для различных сред разработки можно найти в директории `\DLL\Lib`.

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

- `\E-154\Examples\Borland C++ 5.02`;
- `\E-154\Examples\Borland C++ Builder 5.0`;
- `\E-154\Examples\Borland Delphi 6.0`;
- `\E-154\Examples\MicroSoft Visual C++ 6.0`;
- `\E-154\Examples\Labiew 7.1`;
- `\E-154\Examples\LabWindows CVI 8.0`.

Например, 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 файл библиотеки импорта `\DLL\Lib\Borland\LUSBAPI.LIB`;
- создать и добавить в проект Ваш файл с будущей программой (например, `test.cpp`);

- включить в начало вашего файла заголовочный файл `#include "LUSBAPI.H"`, содержащий описание интерфейса модуля *E-154*;
- в принципе желательно сравнить версию используемой библиотеки `Lusbapi.dll` с версией текущего программного обеспечения, используя для этого функцию *GetDllVersion()*;
- вызвать функцию *CreateLInstance()* для получения указателя на интерфейс модуля;
- в общем-то, ВСЁ! Теперь Вы можете писать свою программу и в любом месте, используя полученный указатель, вызывать соответствующие интерфейсные функции из штатной DLL библиотеки `Lusbapi.dll`.

Сторонникам диалекта **Microsoft Visual C++** можно порекомендовать два способа подключения штатной DLL библиотеки к своему приложению:

1. Динамическая загрузка библиотеки `Lusbapi` на этапе выполнения приложения. Подробности смотри в исходных текстах примера из директории `\E-154\Examples\MicroSoft Visual C++ 6.0\DynLoad`.
2. При статической компоновке библиотеки `Lusbapi` в Вашем проекте следует использовать файл библиотеки импорта `LUSBAPI.LIB` из директории `\DLL\Lib\MicroSoft`.

При работе с модулем типа *E-154* в среде **Borland Delphi** рекомендуется применять модуль объявлений `LUSBAPI.PAS`, расположенный в директории `\DLL\Lib\Delphi`. Также вместо исходного модуля объявлений вполне можно задействовать уже откомпилированную версию `LUSBAPI.DCU`.

2.3. Микроконтроллер модуля

На модуле E-154 в качестве ‘*рабочей лошадки*’ используется 32-разрядный микроконтроллер ARM AT91SAM7S64 от фирмы *Atmel Corporation*. ARM работает на тактовой частоте 48 МГц, имеет встроенную Flash память объемом 64 кБ и встроенное ОЗУ 16 кБ. ARM отвечает за корректное функционирование **USB** интерфейса модуля, а также разбирает все пользовательские команды, поступающие из компьютера и задающие различные режимы работы модуля. Особенностью программного софта, заложенного в основу работы ARM, является возможность его обновления. Т.е. фирменная программа, ‘*заливается*’ в ARM на этапе наладки модуля E-154 в **ЗАО “Л-Кард"**. Но конечный пользователь имеет возможность её перепрошивки чисто программным способом без наличия специального кабеля, что является крайне удобным при обновлении версий программы. Самую последнюю версию фирменной программы MCU всегда можно скачать с нашего сайта *www.lcard.ru* из раздела *“Модуль E-154”*. Там из подраздела “”Программное обеспечение” следует выбрать архив `e154fw`***XY***`.zip`, где ***X.Y*** означает номер версии основной программы MCU модуля. На момент написания данного руководства этот архив имеет имя *e154fw10.zip*. Подробнее об обновлении встроенного ПО см. документ “E154_burner.pdf”.

При необходимости, искушенный пользователь может сам модифицировать встроенное ПО для примененного ARM процессора. Программа написана на языке Ansi C в среде разработки IAR 5.11. Исходник можно легко модифицировать, добавив, например, специфические функции по обработке сигналов или сделать программу, работающей вообще без подключения к компьютеру (на модуль достаточно просто подать внешнее питание +5В, после чего произойдет запуск ARM процессора). Основным преимуществом возможности модификации программы ARM является возможность реализации алгоритмов с обратной связью, требующих работы в реальном времени не зависимо от операционной системы Windows.

2.4. Возможные проблемы при работе с модулем

1. Перед началом работы со штатным программным обеспечением модуля *E-154*, во избежание непредсказуемого его поведения, настоятельно рекомендуется установить драйвера для чипсета используемой материнской платы компьютера. В особенности это касается чипсетов не от ***Intel***: ***VIA***, ***SIS***, ***nVidia***, ***AMD+ATI*** и т.д. Обычно эти драйвера можно найти на фирменном CD-ROM, который поставляется вместе с материнской платой. Также их можно скачать из Интернета с сайта производителя.

2. Компьютеры, у которых материнская плата создана на основе чипсета от фирмы ***SIS*** (*Silicon Integrated System Corporation*), ***AMD+ATI*** (*Advanced Micro Devices, Inc.*) или ***nVidia*** (*NVIDIA Corporation*) могут не совсем корректно работать в среде *Windows'98\2000\XP\Vista*. Это проявляется при запросах с большим количеством данных в интерфейсных функциях *ReadData()*. Например, при вызове этой функции с параметром *NumberOfWordsToRead* = 1024*1024 операционная система *Windows* вполне может, что называется, '*наглухо*' зависнуть вплоть до появления BSOD (Blue Screen Of Death). Решение данной проблемы лежит в русле уменьшения значения *NumberOfWordsToRead*. Причём величина *NumberOfWordsToRead*, при которой всё начинает нормально работать, зависит от конкретного экземпляра компьютера. Так что следует попробовать просто поварьировать величину параметра *NumberOfWordsToRead*.

3. Используемые термины и форматы данных

3.1. Термины

Название	Смысл
AdcRate	Частота работы АЦП в *кГц*
InterKadrDelay	Межкадровая задержка в *млс*
KardRate	Частота кадра отсчётов в *кГц*.
Buffer	Указатель на целочисленный массив для данных
Npoints	Число отсчетов ввода
AdcChannel	Логический номер аналогового канала АЦП
ControlTable	Управляющая таблица, содержащая целочисленный массив с логическими номерами каналов для последовательного циклического ввода данных с АЦП
ControlTableLength	Длина управляющей таблицы

3.2. Форматы данных

3.2.1. Формат слова данных с АЦП

Данные, считанные с $12^{ти}$ битного АЦП модуля *E-154*, представляются в формате знакового целого двухбайтного числа от -2048 до 2047. Эти "*сырые*" отсчёты с АЦП рекомендуется откорректировать с помощью калибровочных коэффициентов, которые хранятся в модуле и доступны с помощью штатной функции *GET_MODULE_DESCRIPTION()*. Процедура корректировки данных АЦП достаточно подробно описана в *§ 4.5.1. "Корректировка данных АЦП"* (в библиотеке имеются функции, предназначенные для преобразования кодов АЦП в напряжение с учетом калибровочных коэффициентов). После этой процедуры взаимосвязь откорректированного кода АЦП с входным напряжением модуля можно отобразить следующим образом:

Таблица 1. Соответствие откорректированного кода АЦП входному напряжению

Диапазон, В	Код АЦП	Входное напряжение, В
±5.0; ±1.6; ±0.5; ±0.16	+2000	+5.0; +1.6; +0.5; +0.16
	0	0
	-2000	-5.0; -1.6; -0.5; -0.16

Таблица 2. Соответствие неоткорректированного кода АЦП входному напряжению

Диапазон, В	Код АЦП	Входное напряжение, В
±5.0	+1975	+5.0
	0	0
	-1975	-5.0
±1.6	+1966	+1.6
	0	0
	-1966	-1.6
±0.5	+1961	+0.5
	0	0
	-1961	-0.5
±0.16	+1987	+0.16
	0	0
	-1987	-0.16

3.2.2. Формат слова данных для ЦАП

На модуле реализован 1-канальный 8[ми] битный ЦАП. Для выставления какого-нибудь напряжения на выходе ЦАП в модуль *E-154* необходимо передать 16[ти] битное слово данных. Формат этого слова данных приведен в следующей таблице:

Таблица 3. Формат слова данных ЦАП

Модуль	Номер бита	Назначение
E-154	<7..0>	8[ми] битный код ЦАП
	<15..8>	Не используются

Собственно сам код ЦАП перед отправкой его в модуль рекомендуется откорректировать или воспользоваться функцией со встроенной коррекцией кода с учетом калибровки. Корректировочные коэффициенты хранятся в модуле и доступны с помощью штатной функции *GET_MODULE_DESCRIPTION()*. Процедура корректировки данных ЦАП достаточно подробно описана в *§ 4.6.1. "Корректировка данных ЦАП"*. После этой процедуры откорректированный код, выдаваемый модулем на 8[ми] битный ЦАП, связан с устанавливаемым на внешнем разъёме напряжением в соответствии со следующей таблицей

Таблица 4. Соответствие откорректированного кода ЦАП выходному напряжению

Модуль	Код ЦАП	Напряжение, В
E-154	+127	+5.0
	0	0
	–128	–5.0

Таблица 5. Соответствие неоткорректированного кода ЦАП выходному напряжению

Модуль	Код ЦАП	Напряжение, В
E-154	+117	+5.0
	0	0
	–117	–5.0

3.2.3. Логический номер канала АЦП

Для управления работой входного аналогового каскада модуля *E-154* был введён такой параметр как 8[ми] битный *логический номер канала АЦП*, фактически управляющее слово для АЦП. Именно массив логических номеров каналов АЦП, образующих управляющую таблицу **ControlTable**, задает циклическую последовательность работы АЦП при вводе данных. В состав логического номера канала входят несколько важных параметров, задающих различные режимы функционирования АЦП модуля:

- физический номер аналогового канала;
- индекс коэффициента усиления, т.е. для каждого логического канала можно установить свой индивидуальный коэффициент усиления (диапазон входного напряжения).

Битовый формат логического номера канала АЦП представлен в таблице ниже:

Таблица 6. Формат логического номера канала.

Номер бита	Обозначение	Функциональное назначение
0	**MA0**	0[ой] бит номера канала
1	**MA1**	1[ый] бит номера канала
2	**MA2**	2[ой] бит номера канала
3	**MA3**	Зарезервирован
4	**MA4**	Зарезервирован
5	**MA5**	Зарезервирован
6	**GS0**	0[ой] бит диапазона измерения (см. *Таблицу 5*)
7	**GS1**	1[ый] бит диапазона измерения (см. *Таблицу 5*)

Например, логический номер канала для модуля *E-154* равный 0x2 будет означать 3-ий канал с диапазоном измерения ±5В, 0x82 – тот же канал с диапазоном ввода ±0.5В.

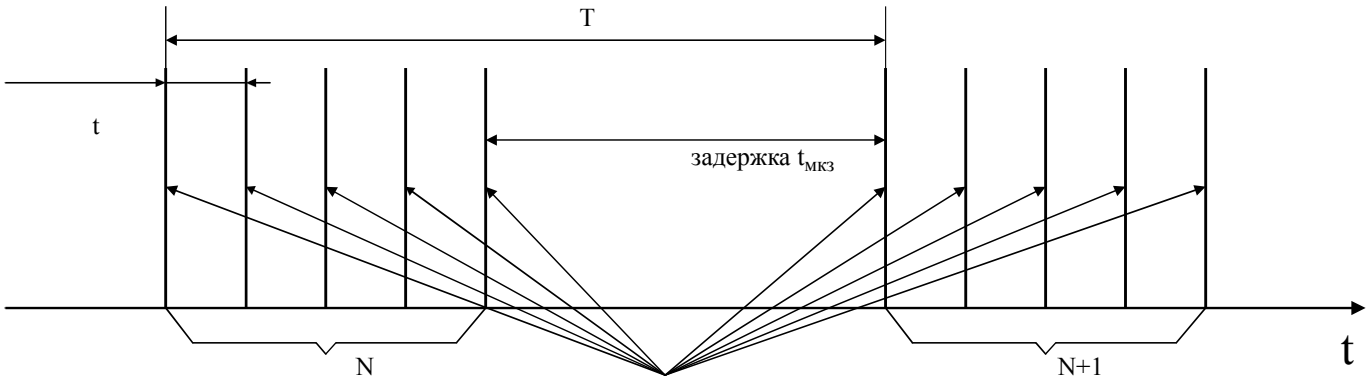
Таблица 7. Диапазоны измерения (биты GS0 и GS1)

Модуль	Бит **GS1**	Бит **GS0**	Диапазон, В
E-154	0	0	±5.0
	0	1	±1.6
	1	0	±0.5
	1	1	±0.16

Например, если в логическом номере канале АЦП установить битовое поле **GS1÷GS0** равным 2, то тем самым будет выбран диапазон входного напряжения ±0.5 В.

3.2.4. Формат кадра отсчетов

Под кадром подразумевается последовательность отсчетов с логических каналов, начиная от **ControlTable[0]** до **ControlTable[ControlTableLength–1]**, где **ControlTable** – управляющая таблица (массив логических каналов), хранящаяся в ОЗУ встроенного ARM, а **ControlTableLength** определяет размер (длину) этой таблицы. Загрузить нужную управляющую таблицу в сигнальный процессор модуля можно с помощью интерфейсной функции ***SET_ADC_PARS()*** (см. *§ 4.5.4. "Установка параметров работы АЦП"*). Временные параметры кадра для **ControlTableLength = 5** приведены на следующем рисунке:



где $\mathbf{T_k}$ – временной интервал между соседними кадрами (фактически частота опроса фиксированного логического номера канала **KardRate**), $\mathbf{t_{мкз}}$= **InterKadrDelay** – временной интервал между последним отсчетом текущего кадра и первым отсчетом следующего, $\mathbf{t_{АЦП}}$ – интервал запуска АЦП или межканальная задержка. Тогда 1/$\mathbf{t_{АЦП}}$= **AdcRate** – частота работы АЦП или оцифровки данных, а величина $\mathbf{t_{мкз}}$ не может принимать значения меньшие, чем $\mathbf{t_{АЦП}}$. Если размер кадра, т.е. число отсчетов с АЦП в кадре, равен **ControlTableLength**, то все эти временные параметры можно связать следующей формулой:

$$\mathbf{T_k} = 1/\mathbf{KadrRate} = (\mathbf{ControlTableLength}–1) * \mathbf{t_{АЦП}} + \mathbf{t_{мкз}},$$

или

$$\mathbf{T_k} = 1/\mathbf{KadrRate} = (\mathbf{ControlTableLength}–1)/\mathbf{AdcRate} + \mathbf{InterKadrDelay}.$$

Временные параметры **AdcRate** и **InterKadrDelay** используются в интерфейсной функции *SET_ADC_PARS()* при задании необходимого режима работы АЦП.

4. Описание библиотеки Lusbapi

В настоящем разделе приведены достаточно подробные описания констант, структур и интерфейсных функций, входящих в состав штатной библиотеки Lusbapi для модуля *E-154*.

4.1. Общие принципы работы с модулем

Целью штатной библиотеки Lusbapi, поставляемой с модулем *E-154*, является предоставление достаточно наглядного и удобного программного интерфейса при работе с данным устройством. Библиотека содержит в себе определенный набор функций, с помощью которых Вы можете реализовывать многие стандартные алгоритмы ввода/вывода данных в/из модуля.

Перед началом работы с библиотекой в пользовательской программе необходимо сделать следующие объявление (как минимум):

```
ILE154 *pModule;                    // указатель на интерфейс модуля E-154
MODULE_DESCRIPTION_E154 md;         // структура служебной информации о модуле
```

Первым делом с помощью функции *GetDllVersion()* следует проверить версии используемой библиотеки Lusbapi и текущего программного обеспечения.

Если версии совпадают, то в Вашем приложении необходимо получить указатель на интерфейс модуля, вызвав функцию *CreateLInstance()*. В дальнейшем для доступа ко всем интерфейсным функциям модуля необходимо применять именно этот указатель (см. *пример ниже*).

После этого, используя уже полученный указатель на интерфейс модуля, следует проинициализировать доступ к виртуальному слоту, к которому подключён модуль *E-154*. Для этого предусмотрена интерфейсная функция *OpenLDevice()*. Если нет ошибки при выполнении этой функции, то можно быть уверенным, что устройство типа *E-154* обнаружено в выбранном виртуальном слоте.

Теперь, в принципе, можно переходить к этапу чтения служебной информации о модуле, но иногда нелишне бывает определиться с текущей скоростью работы используемого **USB** порта. Для этого предназначена интерфейсная функция *GetUsbSpeed()*. Модуль совместно с **USB** портом должен работать в так называемом Full-Speed Mode. Это будет соответствовать пропускной способности модуля по **USB** шины порядка 1 Мбайт/с.

На следующем этапе следует прочитать служебную информацию о модуле. Она требуется при работе с некоторыми интерфейсными функциями библиотеки Lusbapi. Интерфейсная функция *GET_MODULE_DESCRIPTION()* как раз и предназначена для этой цели. Если функция не вернула ошибку, то это означает, что вся служебная информация о модуле успешно получена и можно продолжать работу.

В общем-то, предварительный этап работы с модулем *E-154* можно считать успешно завершённым. Теперь можно спокойно управлять всей доступной периферией на модуле с помощью соответствующих интерфейсных функций библиотеки Lusbapi и организовывать различные режимы работы модуля. Например, такие режимы как:

- непрерывный *потоковый* сбор с АЦП с синхронизацией ввода данных;
- однократный, и потому достаточно медленный, вывод данных на ЦАП;
- однократная работа с входными и выходными цифровыми линиями;
- работа с пользовательским ППЗУ модуля и т.д.

В качестве примера приведем исходный текст, а вернее сказать ‘*скелет*’, консольной программы для работы с *E-154*, предполагая использование Lusbapi версии не ниже 3.2:

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

ILE154 *pModule;                  // указатель на интерфейс модуля
MODULE_DESCRIPTION_E154 md;       // структура с информацией о модуле
char ModuleName[7];               // название модуля
BYTE UsbSpeed;                    // скорость работы шины USB
```

```
int main(void)
{
    // проверим версию DLL библиотеки
    if(GetDllVersion() != CURRENT_VERSION_LUSBAPI)
     {
     printf("Неправильная версия Dll!");
      return 1;                  //выйдем из программы с ошибкой
     }
    // получим указатель на интерфейс модуля
     pModule = static_cast<ILE154 *>(CreateLInstance("e154"));
     if(!pModule)
    {
      printf("Не могу получить указатель на интерфейс");
      return 1;                  //выйдем из программы с ошибкой
    }
    // попробуем обнаружить какой-нибудь модуль
    // в нулевом виртуальном слоте
    if(!pModule->OpenLDevice(0))
    {
      printf("Не могу получить доступ к модулю!");
      return 1;                  //выйдем из программы с ошибкой
    }
   // попробуем получить скорость работы шины USB
   if(!pModule->GetUsbSpeed(&UsbSpeed))
   {
      printf("Не могу узнать скорость работы USB!\n");
      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 - 'E-154'?
    if(strcmp(ModuleName, "E154"))
    {
      printf(" В нулевом виртуальном слоте не 'E-154'\n");
      return 1;                  //выйдем из программы с ошибкой
    }
    // попробуем прочитать информацию о модуле
    if(!pModule->GET_MODULE_DESCRIPTION(&md))
    {
    printf("Не выполнена функция GET_MODULE_DESCRIPTION ()!");
    return 1;                    //выйдем из программы с ошибкой
     }

    printf("Модуль E-154 (серийный номер %s) полностью готов к\
                                  работе!", md.Module.SerialNumber);
    // далее можно располагать функции для непосредственного
    // управления модулем
    . . . . . .
```

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

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

4.2. Константы

Приведённые ниже основные константы настоятельно рекомендуется использовать в исходных текстах приложения при работе с модулем *E-154*. Это весьма повышает *читаемость* и *понимаемость* исходных текстов, а также значительно облегчает сопровождение программ. Рассматриваемые константы расположены в файле `\DLL\Include\Lusbapi.h`.

1. У модуля *E-154* есть четыре возможных диапазона входных напряжений, каждый из которых можно задать данными константами. Местом использования этих констант, как правило, являются битовые поля **GS0÷GS1** *логического номера канала АЦП*. Массив этих логических каналов образуют управляющую таблицу поля *ControlTable* структуры *ADC_PARS_E154*.

Константа	Значение	Назначение
ADC_INPUT_RANGE_ 5000mV_E154	0	Данная константа задаёт входной диапазон, равный ±5000 мВ. Также можно эту константу можно использовать в качестве индекса для доступа к первому элементу константного массива *ADC_INPUT_RANGES_E154*.
ADC_INPUT_RANGE_ 1600mV_E154	1	Данная константа задаёт входной диапазон, равный ±1600 мВ. Также можно эту константу можно использовать в качестве индекса для доступа ко второму элементу константного массива *ADC_INPUT_RANGES_E154*.
ADC_INPUT_RANGE_ 500mV_E154	2	Данная константа задаёт входной диапазон, равный ±500 мВ. Также можно эту константу можно использовать в качестве индекса для доступа к третьему элементу константного массива *ADC_INPUT_RANGES_E154*.
ADC_INPUT_RANGE_ 160mV_E154	3	Данная константа задаёт входной диапазон, равный ±160 мВ. Также можно эту константу можно использовать в качестве индекса для доступа к четвёртому элементу константного массива *ADC_INPUT_RANGES_E154*.

2. Модуль *E-154* в состоянии осуществлять различную синхронизацию ввода данных с АЦП. Данные константы определяют тип требуемой синхронизации. Местом использования этих констант, как правило, является поле *InputMode* структуры *ADC_PARS_E154*.

Константа	Значение	Назначение
NO_SYNC_E154	0	***Отсутствие синхронизации ввода.*** Сбор данных с АЦП модуль начинает непосредственно после выполнения функции *START_ADC()*.
TTL_START_SYNC_E154	1	***Цифровая синхронизация начала ввода.*** Непосредственно после выполнения функции *START_ADC()* модуль переходит в состояние ожидания прихода заданного перепада у ТТЛ-совместимого одиночного импульса на одной из 8 входных цифровых линий (конкретный номер линии, по которой ожидается перепад устанавливает пользователь). Длительность этого синхроимпульса должна быть не менее 1000 нс. Только после этого модуль приступает к сбору данных.
RESERVED_SYNC_154	2	**Зарезервирован.**
ANALOG_SYNC_E154	3	***Аналоговая синхронизация начала ввода***. Непосредственно после выполнения функции *START_ADC()* модуль переходит в состояние ожидания выполнения условий аналоговой синхронизации. Модуль начинает собирать данные с АЦП только после выполнения заданных соотношений между полученным значением с заданного аналогового синхроканала и заданным пороговым значением.
INVALID_SYNC_E154	4	Неправильный вид синхронизации ввода данных.

3. Ревизия модуля *E-154* отражает определённые конструктивные особенности модуля. Она задаётся одной заглавной латинской буквой. Так, *первая* ревизия модуля обозначается ***'A'***.

Константа	Значение	Назначение
REVISION_A_E154	0	Данная константа может использоваться в качестве индекса для доступа к первому элементу константного массива *REVISIONS_E154*.

4. Различные константы для работы с модулем *E-154*.

Константа	Значение	Назначение
CURRENT_VERSION_LUSBAPI	——	Версия используемой библиотеки `Lusbapi`. Как правило, используется совместно с функцией *GetDllVersion()*.
MAX_CONTROL_TABLE_LENGTH_E154	16	Максимально возможное кол-во логических каналов в управляющей таблице **ControlTable**.
ADC_CALIBR_COEFS_QUANTITY_E154	4	Кол-во корректировочных коэффициентов для данных АЦП. По одному на каждый входной

		диапазон. Корректировке подвергаются как смещение, так и масштаб данных АЦП.
ADC_INPUT_RANGES_QUANTITY_E154	4	Кол-во входных диапазонов.
DAC_CHANNELS_QUANTITY_E154	1	Кол-во физических каналов ЦАП на модуле (всегда равно 1).
DAC_CALIBR_COEFS_QUANTITY_E154	1	Кол-во корректировочных коэффициентов для данных ЦАП (всегда равно 1).
TTL_LINES_QUANTITY_E154	8	Кол-во входных и выходных цифровых линий.
USER_FLASH_SIZE_E154	128	Размер пользовательского ППЗУ в байтах
REVISIONS_QUANTITY_E154	1	Кол-во ревизий (модификаций) модуля.

5. Различные *константные массивы* для работы с модулем *E-154*.
   a. Массив доступных диапазонов напряжения АЦП в Вольтах:
   ```
   const double
           ADC_INPUT_RANGES_E154[ADC_INPUT_RANGES_QUANTITY_E154] =
   {
   5.0, 1.6, 0.5, 0.16
   };
   ```
   b. Диапазон выходного напряжения ЦАП в Вольтах:
   ```
   const double DAC_OUTPUT_RANGE_E154 = 5.0;
   ```
   c. Ревизия модуля отражает определённые конструктивные особенности модуля. Она задаётся одной заглавной латинской буквой. Например, *первая* ревизия модуля обозначается через букву ***'A'***. Текущая ревизия модуля содержится в поле *Module.Revision* структуры служебной информации *MODULE_DESCRIPTION_E154*. Массив доступных ревизий модуля задаётся следующим образом.
   ```
   const BYTE REVISIONS_E154[REVISIONS_QUANTITY_E154] = { 'A'};
   ```

4.3. Структуры

В данном разделе приведены основные типы структур, которые применяются в библиотеке `Lusbapi` при работе с модулем *E-154*.

4.3.1. Структура MODULE_DESCRIPTION_E154

Структура *MODULE_DESCRIPTION_E154* описана в файле `\DLL\Include\Lusbapi.h` и представлена ниже:

```
struct MODULE_DESCRIPTION_E154
{
    MODULE_INFO_LUSBAPI      Module;       // общая информация о модуле
    INTERFACE_INFO_LUSBAPI   Interface;    // информация об интерфейсе
    MCU_INFO_LUSBAPI<VERSION_INFO_LUSBAPI> Mcu;     // информация о MCU
    ADC_INFO_LUSBAPI         Adc;          // информация о АЦП
    DAC_INFO_LUSBAPI         Dac;          // информация о ЦАП
    DIGITAL_IO_INFO_LUSBAPI  DigitalIo;    // информация о цифровом вводе-выводе
};
```

В данной структуре представлена самая общая служебная информация об используемом экземпляре модуля *E-154*. Эта структура используется при работе с интерфейсными функциями *SAVE_MODULE_DESCRIPTION()* и *GET_MODULE_DESCRIPTION()*. В определении этой структуры применяются вспомогательные константы и типы данных, описанные в *Приложении А*.

4.3.2. Структура ADC_PARS_E154

Структура *ADC_PARS_E154* описана в файле `\DLL\Include\Lusbapi.h` и представлена ниже:

```
struct ADC_PARS_E154
{
    WORD ClkSource;                 // зарезервирован (для совместимости с другими модулями)
    WORD EnableClkOutput;           // зарезервирован (для совместимости с другими модулями)
    WORD InputMode;                 // режим синхронизации ввода данных с АЦП
    WORD SynchroAdType;             // тип аналоговой синхронизации
    WORD SynchroAdMode;             // режим аналоговой синхронизации
    WORD SynchroAdChannel;          // канал АЦП при аналоговой синхронизации
    SHORT SynchroAdPorog;           // порог срабатывания аналоговой синхронизации
    WORD ChannelsQuantity;          // число активных каналов (размер кадра)
    WORD ControlTable[16];          // управляющая таблица с активными лог. каналами
    double AdcRate;                 // частота работы АЦП в кГц
    double InterKadrDelay;          // межкадровая задержка в мс
    double ChannelRate;             // частота кадра в кГц
};
```

Перед началом работы с АЦП необходимо заполнить поля данной структуры и передать ее в модуль с помощью интерфейсной функции *SET_ADC_PARS()*. В описании этой функции подробно прокомментированы смысл и назначение всех полей данной структуры. Также при необходимости можно считать из модуля текущие параметры функционирования АЦП, используя интерфейсную функцию *GET_ADC_PARS()*.

4.3.3. Структура IO_REQUEST_LUSBAPI

Структура *IO_REQUEST_LUSBAPI* описана в файле `\DLL\Include\LusbapiTypes.h` и представлена ниже:

```
struct IO_REQUEST_LUSBAPI
{
    SHORT * Buffer;                     // буфер для передаваемых данных
    DWORD   NumberOfWordsToPass;        // кол-во отсчётов, которые требуется передать
    DWORD   NumberOfWordsPassed;        // кол-во реально переданных отсчётов
    OVERLAPPED * Overlapped;  // для синхронного запроса – NULL, а асинхроннного
                              // запроса – указатель на структуру типа OVERLAPPED
    DWORD   TimeOut;          // для синхронного запроса – таймаут в мс, а для
                              // асинхронного запроса не используется
};
```

Данная структура используется функцией *ReadData()* при организации передачи получаемых с АЦП данных из модуля в компьютер. В описании этой функции подробно прокомментированы смысл и назначение полей данной структуры.

4.3.4. Структура LAST_ERROR_INFO_LUSBAPI

Структура *LAST_ERROR_INFO_LUSBAPI* описана в файле `\DLL\Include\LusbapiTypes.h` и представлена ниже:

```
struct LAST_ERROR_INFO_LUSBAPI
```

```
{
    BYTE  ErrorString[256];        // строка с кратким описанием последней ошибки
                                   // библиотеки Lusbapi
    DWORD ErrorNumber;             // номер последней ошибки библиотеки Lusbapi
};
```

Данная структура используется функцией *GetLastErrorInfo()* при выявлении ошибок выполнения интерфейсных функций библиотеки Lusbapi.

4.4. Функции общего характера

4.4.1. Получение версии библиотеки

Формат: **DWORD** ***GetDllVersion****(void)*
Назначение: Данная функция является одной из *двух* экспортируемых из штатной библиотеки Lusbapi функцией. Она возвращает текущую версию используемой библиотеки. Формат номера версии следующий: Битовое поле Назначение <31..16> Старшее слово версии библиотеки <15..0> Младшее слово версии библиотеки Рекомендованную последовательность вызовов интерфейсных функций см. *§ 4.1. "Общие принципы работы с модулем"*.
Передаваемые параметры: нет
Возвращаемое значение: номер версии библиотеки Lusbapi.

4.4.2. Получение указателя на интерфейс модуля

Формат: **LPVOID** ***CreateLInstance****(PCHAR const DeviceName)*
Назначение: Данная функция должна обязательно вызываться в начале каждой пользовательской программы, работающей с модулями *E-154*. Она является одной из *двух* экспортируемых из штатной библиотеки Lusbapi функцией и возвращает указатель на интерфейс для устройства с названием *DeviceName*. Все последующие интерфейсные функции штатной библиотеки вызываются именно через этот возвращаемый указатель. Рекомендованную последовательность вызовов интерфейсных функций см. *§ 4.1. "Общие принципы работы с модулем"*.

Передаваемые параметры:
- *DeviceName* – строка с названием устройства (для данного модуля это – “E154”) .

Возвращаемое значение: В случае успеха — указатель на интерфейс, иначе — **NULL**.

4.4.3. Завершение работы с интерфейсом модуля

Формат: **BOOL** ***ReleaseLInstance****(void)*

Назначение:

Данная интерфейсная функция реализует корректное высвобождение интерфейсного указателя, проинициализированного с помощью интерфейсной функции *CreateLInstance()*. Используется для аккуратного завершения сеанса работы с модулем, если предварительно удачно выполнилась функция *CreateLInstance()*. **!!!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. *§ 4.1. “Общие принципы работы с модулем”*.

Передаваемые параметры: нет.

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

4.4.4. Инициализация доступа к модулю

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

Назначение:

С программной точки зрения, не вдаваясь в излишние тонкости, подсоединенный к компьютеру модуль *E-154* можно рассматривать как устройство, подключённое к некоему виртуальному слоту с сугубо индивидуальным номером. Основное назначение данной интерфейсной функции как раз в том и состоит, чтобы определить, что именно модуль *E-154* находится в заданном виртуальном слоте. Если функция ***OpenLDevice()*** успешно выполнилась для заданного виртуального слота, то можно переходить к чтению служебной информации и управлению модулем с помощью соответствующих интерфейсных функций штатной библиотеки `Lusbapi`.

Рекомендованную последовательность вызовов интерфейсных функций см. *§ 4.1. “Общие принципы работы с модулем”*.

Передаваемые параметры:
- *VirtualSlot* – номер виртуального слота, к которому, как предполагается, подключен модуль *E-154*.

Возвращаемое значение: *TRUE* – модуль *E-154* находится в выбранном виртуальном слоте и можно переходить к работе с ним;
FALSE – устройства типа модуль *E-154* в выбранном виртуальном слоте нет. Следует попробовать другой номер виртуального слота.

4.4.5. Завершение доступа к модулю

Формат: **BOOL** ***CloseLDevice*** *(void)*

Назначение:

Данная интерфейсная функция прерывает всякое взаимодействие с *текущим* виртуальным слотом, к которому подключён модуль. При этом данный виртуальный слот аккуратно закрывается и выполняется освобождение связанных с ним ресурсов *Windows*. После её применения всякий доступ к модулю *E-154* становится невозможным. Для возобновления нормального доступа к устройству необходимо вновь воспользоваться интерфейсной функцией *OpenLDevice()*. Таким образом, эта функция, по своей сути, противоположна интерфейсной функции *OpenLDevice()*. Фактически данная функция используется в таких интерфейсных функциях, как *OpenLDevice()* и *ReleaseLInstance()*.

Передаваемые параметры: нет.

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

4.4.6. Получение названия модуля

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

Назначение:

Данная вспомогательная интерфейсная функция позволяет получить название подключенного к слоту модуля. Массив под название модуля *ModuleName* (не менее 6 символов плюс признак конца строки ‘\0’, т.е. нулевой байт) должен быть заранее определен.

Рекомендованную последовательность вызовов интерфейсных функций см. *§ 4.1. "Общие принципы работы с модулем"*. Данную функцию не рекомендуется вызывать во время сбора данных с АЦП.

Передаваемые параметры:

- *ModuleName* – возвращается строка, не менее 6 символов, с названием модуля (в нашем случае это должна быть строка "E154").

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

4.4.7. Получение скорости работы модуля

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

Назначение:

Данная функция позволяет определить, на какой скорости шина **USB** работает с модулем.

Передаваемые параметры:

- *UsbSpeed* – возвращаемое значение этой переменной может принимать следующее:
  - ✓ 0 – модуль работает с **USB** шиной в режиме *Full-Speed Mode* (12 Мбит/с)
  - ✓ 1 – модуль работает с **USB** шиной в режиме *High-Speed Mode* (480 Мбит/с).

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

4.4.8. Получение дескриптора устройства

Формат: **HANDLE** ***GetModuleHandle****(void)*
Назначение: Данная функция позволяет получить дескриптор (*handle*) используемого модуля *E-154*.
Передаваемые параметры: нет
Возвращаемое значение: В случае успеха – дескриптор модуля *E-154*; в противном случае – **INVALID_HANDLE_VALUE**.

4.4.9. Получение описания ошибок выполнения функций

Формат: **BOOL** ***GetLastErrorInfo***(*LAST_ERROR_INFO_LUSBAPI * const LastErrorInfo*)
Назначение: Если в процессе работы с библиотекой `Lusbapi` 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 интерфейсной функции *ReadData()*. Т.к. эта функция фактически является слепком со стандартной *Windows API* функции ***ReadFile()***, то для выявления ошибок следует пользоваться классификацией ошибок, присущей системе *Windows*.
Передаваемые параметры: • *LastErrorInfo* – указатель на структуру типа *LAST_ERROR_INFO_LUSBAPI*, в которой возвращается краткое описание и номер последней ошибки.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE* – функция выполнена с ошибкой.

4.5. Функции для работы с АЦП

Интерфейсные функции штатной библиотеки `Lusbapi` позволяют реализовывать разнообразные алгоритмы работы модуля *E-154* с АЦП. Но основным режимом работы модуля является, конечно же, непрерывный *потоковый* сбор данных с АЦП. А вообще модуль, с точки зрения состояния АЦП, может находиться как бы в двух режимах:

1. режим "*покоя*";
2. потоковый, т.е. *перманентный*, сбор данных с АЦП.

Функция *START_ADC()* позволяет переводить модуль во второе из этих состояний, а *STOP_ADC()* — в первое. Прежде чем запустить модуль на сбор данных с АЦП, необходимо в него передать все требуемые параметры функционирования АЦП: тип синхронизации, частота работы АЦП, управляющую таблицу и т.д. Эту операцию можно осуществить с помощью интерфейсной функции *SET_ADC_PARS()*. После этого, в принципе, можно запускать модуль на сбор данных, выполнив функцию *START_ADC()*. Для извлечения из модуля уже полученных с АЦП данных следует пользоваться функцией *ReadData()*. При этом функция *ReadData()* может выполняться как в *синхронном*, так и в *асинхронном* режимах.

Для повышения надёжности сбора данных с АЦП, эти отсчёты помещаются во внутренний циклический *FIFO* буфер модуля, размер которого составляет 11 кБ. Это соответствует промежутку времени сбора данных, равному ~46 мс при частоте дискретизации АЦП 120 кГц. Приложение может получать отсчёты АЦП из *FIFO* буфера модуля при помощи функции *ReadData()*.

Простой пример применения интерфейсных функций для работы с АЦП можно найти в директории `\E-154\Example\`.

4.5.1. Корректировка данных АЦП

Схемотехника и использованные компоненты обеспечивают линейность передаточной характеристики тракта АЦП модуля *E-154*. Однако на *сегодняшний* день модуль не умеет производить автоматическую корректировку собираемых с АЦП данных. Это приводит к тому, что входные показания АЦП могут иметь некоторое смещение нуля и неточность в передаче масштаба. Поэтому на уровне приложения необходима реализация всей этой нудной задачи по корректировке данных АЦП. Зато благодаря программной коррекции данных на модуле полностью отсутствуют какие-либо подстроечные резисторы, что позволяет улучшить шумовые характеристики модуля и увеличить их надёжность.

В качестве корректировочных коэффициентов вполне можно использовать как *штатные*, так и свои собственные, т.е. *пользовательские*.

Пользовательские корректировочные коэффициенты могут быть использованы, например, для целей компенсации погрешностей целого измерительного тракта какого-нибудь стенда, составной частью которого вполне может служить модуль *E-154*. При этом вся ответственность за формирование и корректное применение *пользовательских* корректировочных коэффициентов полностью ложится на плечи конечного пользователя.

Штатные корректировочные коэффициенты располагаются в полях *Adc.OffsetCalibration[]* и *Adc.ScaleCalibration[]* структуры служебной информации *MODULE_DESCRIPTION_E154*. Вся служебная информация совместно с корректировочными коэффициентами записывается в модуль на этапе при его наладке в **ЗАО "Л-Кард"**. Поля коэффициентов представляют собой массивы типа *double*. Для модуля *E-154* в каждом из этих массивов используются только первые *ADC_CALIBR_COEFS_QUANTITY_E154* элементов. Массив *Adc.OffsetCalibration* содержит коэффициенты для корректировки смещение нуля, а массив *Adc.ScaleCalibration* – для корректировки масштаба. Если в *логическом номере канала АЦП* индекс коэффициента усиления равен *i*, то корректировочные коэффициенты этого логического канала могут быть получены следующим образом:

- смещение: *Adc.OffsetCalibration[i]*;
- масштаб: *Adc.ScaleCalibration[i]*.

В общем виде процедура корректировки отсчётов АЦП выполняется по следующей формуле:

$$Y = (X+A)*B,$$

где: X – некорректированные данные АЦП [в отсчётах АЦП],
Y – скорректированные данные АЦП [в отсчётах АЦП],
A – коэффициент смещения нуля [в отсчётах АЦП],
B – коэффициент масштаба [безразмерный].

Например, пусть со второго канала АЦП, настроенного на входной диапазон ±1.6 , получены следующие данные: X1 = 1000, X2 = –1000 и X3 = 0. Тогда коэффициенты и скорректированные данные можно представить так:

- A = Adc.OffsetCalibration[1];
- B = Adc.ScaleCalibration[1];
- Y1 = (A+1000)*B, Y2 = (A-1000)*B, Y3 = A*B.

4.5.2. Запуск АЦП.

Формат: **BOOL** ***START_ADC****(void)*
Назначение: Данная функция запускает модуль *E-154* на непрерывный *потоковый* сбор данных с АЦП. Перед любым запуском сбора данных настоятельно рекомендуется выполнять функцию *STOP_ADC()*. Перед началом сбора можно установить требуемые параметры работы АЦП, которые передаются в модуль с помощью интерфейсной функции *SET_ADC_PARS()*. Извлечение из модуля уже собранных с АЦП данных можно осуществлять с помощью интерфейсной функции *ReadData()*.
Передаваемые параметры: нет
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE* – функция выполнена с ошибкой.

4.5.3. Остановка АЦП

Формат: **BOOL** ***STOP_ADC****(void)*
Назначение: Данная функция останавливает в модуле *E-154* механизм сбора данных с АЦП. Попутно эта функция ‘*приводит в чувство*’ основную программу микроконтроллера модуля, а также сбрасывает используемый канал передачи данных по **USB** шине. Поэтому настоятельно рекомендуется применять эту функцию перед каждым запуском сбора данных функцией *START_ADC()*.
Передаваемые параметры: нет
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE* – функция выполнена с ошибкой.

4.5.4. Установка параметров работы АЦП

Формат: **BOOL** ***SET_ADC_PARS****(ADC_PARS_E154 * const AdcPars)*

Назначение:

Данная функция передает в *E-154* всю необходимую информацию, которая используется модулем для организации заданного режима сбора данных с АЦП. Всю нужную информацию описываемая интерфейсная функция извлекает из полей передаваемой структуры типа *ADC_PARS_E154*. Собственно, использование модулем именно этой переданной информации начинается только после выполнения интерфейсной функции *START_ADC()*. Нельзя вызывать эту функцию в процессе сбора данных. Её следует применять только после выполнения функции *STOP_ADC()*.

Описание структуры *ADC_PARS_E154* приведено ранее в *§ 4.3.2. "Структура ADC_PARS_E154"*, а назначение отдельных ее полей описано ниже.

- Поле AdcPars->*ClkSource*. Зарезервирован для совместимости с другими модулями.
- Поле AdcPars->*EnableClkOutput*: Зарезервирован для совместимости с другими модулями.
- Поле AdcPars->InputMode. Запись. Значение данного поля может задавать различные виды синхронизации ввода данных с АЦП. Это поле может принимать одно из четырёх значений от 0 до 3, также можно пользоваться *константами синхронизации ввода*. Смысловая нагрузка значений этого поля представлена в таблице ниже:

Значение	Константа	Назначение
0	**NO_SYNC_E154**	***Отсутствие синхронизации ввода.*** Сбор данных с АЦП модуль начинает непосредственно после выполнения функции *START_ADC()*.
1	**TTL_START_SYNC_E154**	***Цифровая синхронизация начала ввода.*** Непосредственно после выполнения функции *START_ADC()* модуль переходит в состояние ожидания прихода заданного перепада у ТТЛ-совместимого одиночного импульса на одной из 8 входных цифровых линий (конкретный номер линии, по которой ожидается перепад устанавливает пользователь). Длительность этого синхроимпульса должна быть не менее 1000 нс. Только после этого модуль приступает к сбору данных. При этом условия цифровой синхронизации ввода данных задаются через посредство полей AdcPars->SynchroAdChannel и AdcPars->SynchroAdPorog. Необходимо иметь в виду, что в силу специфики программно-аппаратной реализации синхронизации непосредственно сбор данных начнется через 4 такта преобразования АЦП после прихода син-

хроимпульса.

2

RESERVED_SYNC_154

Зарезервированное значение для совместимости с другими модулями.

3

ANALOG_SYNC_E154

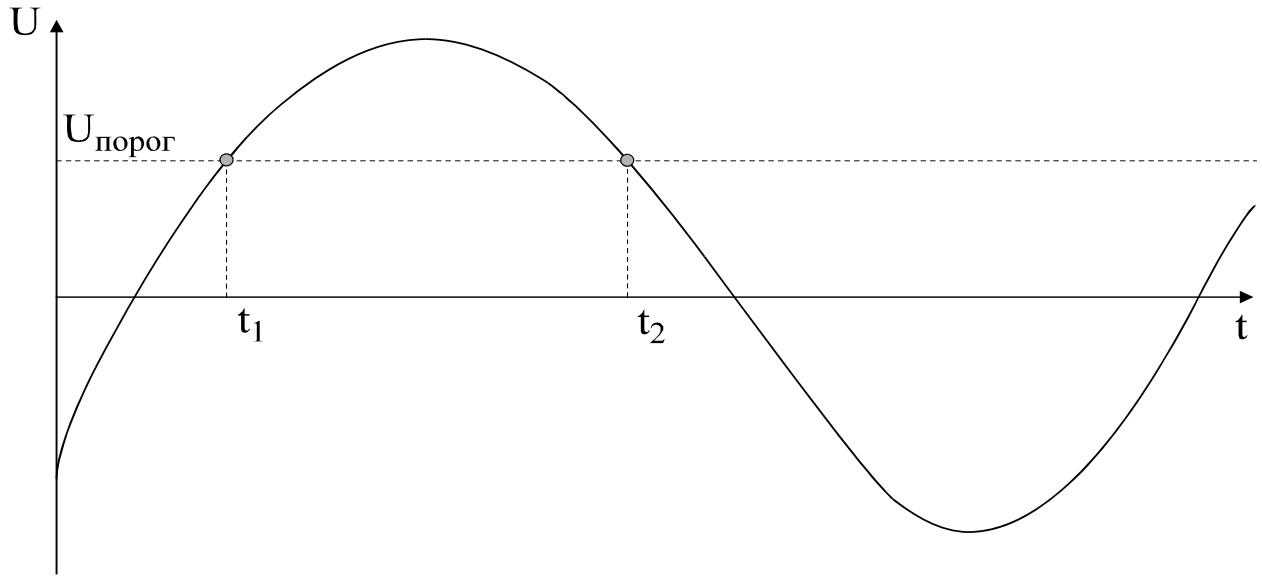
Аналоговая синхронизация начала ввода. Непосредственно после выполнения функции *START_ADC()* модуль переходит в состояние ожидания выполнения условий аналоговой синхронизации. Модуль начинает собирать данные с АЦП только после выполнения заданных соотношений между получаемым значением с заданного аналогового синхроканала и заданным пороговым значением. При этом условия аналоговой синхронизации ввода данных задаются через посредство полей AdcPars->SynchroAdType, AdcPars->SynchroAdMode, AdcPars->SynchroAdChannel и AdcPars->SynchroAdPorog. Необходимо иметь в виду, что в силу специфики программно-аппаратной реализации синхронизации непосредственно сбор данных начнется через 4 такта преобразования АЦП после прихода синхроимпульса.

4

INVALID_SYNC_E154

Неправильная синхронизация ввода данных.

- Установка параметров аналоговой синхронизации. Поля AdcPars->SynchroAdType, AdcPars->SynchroAdMode, AdcPars->SynchroAdChannel, *AdcPars->SynchroAdPorog*. Запись. Различные моменты старта аналоговой синхронизации приведены на следующем рисунке:



Если, например, задана *аналоговая* синхронизация по **переходу**

(AdcPars->SynchroAdType ≠ 0) ‘*снизу-вверх*’ (AdcPars->SynchroAdMode = 0) при пороговом значении равном AdcPars->SynchroAdPorog = $U_{порог}$(в кодах АЦП), то модуль начнет собирать данные только после наступления момента времени $\mathbf{t_1}$. При этом уровень входного сигнала на логическом синхроканале AdcPars->SynchroAdChannel пересечет пороговую линию в направлении ‘*снизу-вверх*’. Аналогично, если задан **переход** ‘*сверху-вниз*’ (AdcPars->SynchroAdMode ≠ 0), то старт начало сбора наступит в момент времени $\mathbf{t_2}$, когда уровень входного сигнала на синхроканале пересечет пороговую линию в направлении сверху вниз. Если же *аналоговая* синхронизация задается по **уровню** (AdcPars->SynchroAdType = 0), то сбор модулем данных начнется в тот момент, когда уровень входного сигнала на синхроканале окажется либо выше (AdcPars->SynchroAdMode = 0), либо ниже (AdcPars->SynchroAdMode ≠ 0) пороговой линии $U_{порог}$.

Все вышесказанное относительно аналоговой синхронизации ввода данных можно кратко свести к следующему:

- поле AdcPars->SynchroAdType может принимать следующие значения:
  - ✓ AdcPars->SynchroAdType = 0 — аналоговая синхронизация по **уровню**,
  - ✓ AdcPars->SynchroAdType ≠ 0 — аналоговая синхронизация по **переходу**;
- поле AdcPars->SynchroAdMode может принимать следующие значения:
  - ✓ при AdcPars->SynchroAdMode = 0**:**
    - аналоговая синхронизация по **уровню** — ‘*выше*’,
    - аналоговая синхронизация по **переходу** — ‘*снизу-вверх*’,
  - ✓ при AdcPars->SynchroAdMode ≠ 0**:**
    - аналоговая синхронизация по **уровню** — ‘*ниже*’,
    - аналоговая синхронизация по **переходу** — ‘*сверху-вниз*’;
- поле AdcPars->SynchroAdChannel – логический номер синхроканала АЦП;
- поле AdcPars->SynchroAdPorog – пороговое значение в кодах АЦП;

- Установка параметров цифровой синхронизации. Если выбран режим цифровой синхронизации старта, то параметры цифровой синхронизации старта задаются полями AdcPars->SynchroAdChannel, *AdcPars->SynchroAdPorog*. Запись.
  - Поле AdcPars->SynchroAdChannel может принимать следующие значения:
    - AdcPars->SynchroAdChannel = 1: сбор данных начнется после того, как на заданном цифровом входе произойдет переход из нуля в единицу.
    - AdcPars->SynchroAdChannel = 0: сбор данных начнется после того, как на заданном цифровом входе произойдет переход из единицы в ноль.
  - Поле *AdcPars->SynchroAdPorog* устанавливает, какая конкретно из восьми входных цифровых линий будет использована как линия синхронизации. Данное поле используется как маска, т.е. для использования конкретной линии как входа синхронизации, необходимо соответствующий бит параметра *AdcPars->SynchroAdPorog* установить в единицу (например, для использования цифрового входа 1 необходимо присвоить *AdcPars->SynchroAdPorog=0x01,* для использования входа 8 необходимо присвоить *AdcPars->SynchroAdPorog=0x80*.
- Поле AdcPars->ChannelsQuantity. Запись–Чтение. Данное поле задаёт количество активных *логических каналов* в управляющей таблице **ControlTable**. Т.е. модуль при сборе данных с АЦП будет использовать первые AdcPars->ChannelsQuantity элементов массива AdcPars->ControlTable. Предельным значением для данного поля является величина *MAX_CONTROL_TABLE_LENGTH_E154*.
- Поле AdcPars->ControlTable[]. Запись. Данное поле является массивом типа *WORD*. Оно задаёт управляющую таблицу **ControlTable**. Т.е. тот массив *логических каналов*, который будет использоваться модулем при работе с АЦП для задания циклической последовательности отсчётов с входных аналоговых каналов.
- Поля AdcPars->AdcRate и AdcPars->InterKadrDelay. Запись–Чтение. Данные поля име-

ют силу только при использовании модулем внутренних *тактовых импульсов*, что определяется полем *AdcPars->ClkSource*. При этом поле AdcPars->InterKadrDelay не принимается в рассмотрение при использовании *покадровой синхронизации ввода*, которая может задаваться полем *AdcPars->InputMode*. При входе в функцию в данных полях должны содержаться требуемые временны**е** параметры сбора данных: частота работы АЦП **AdcRate** (обратная величина *межканальной* задержки) и межкадровая задержка **InterKadrDelay**. При этом **AdcRate** задаётся в *кГц*, а **InterKadrDelay** – в *мс*. После выполнения функции ***SET_ADC_PARS()*** 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 образуют некую сетку частот. Так, в общем виде, частота работы АЦП определяется по следующей формуле: **AdcRate** = $F_{clock}/2/N$, где F_{clock} – тактовая частота установленного на модуле микроконтроллера, равная 48000 кГц, а N – целое число в диапазоне от 10 до 65530. Кроме этого, при вычислении дискретного значения частоты задается масштабный делитель частоты микроконтроллера, который позволяет разделить 48000 кГц на одну из следующих величин: 1, 4, 16,64, 512. Данная функция просто вычисляет ближайшую к задаваемой дискретную величину **AdcRate**, передает её в модуль в виде целого числа N, а также возвращает её значение в поле AdcPars->AdcRate. Все то же самое верно и для межкадровой задержки **InterKadrDelay**, с той лишь разницей, что она задается по формуле **InterKadrDelay** = K/**AdcRate**, где K – целое число в диапазоне от 1 до 65530. Причём здесь используется уже откорректированная величина **AdcRate**. В итоге минимальное значение **AdcRate** может составлять 5 Гц, максимальное – 120 кГц. Например, если задать AdcPars->AdcRate = 0, то ***SET_ADC_PARS()*** установит и возвратит минимально возможную величину для данного поля, т.е. 0.005 кГц. Аналогично, если задать AdcPars->InterKadrDelay = 0, то функция установит и возвратит минимально возможную межкадровую задержку, т.е. 1/AdcPars->AdcRate.

- Поле AdcPars->KadrRate. Чтение. В данном поле возвращается частота кадра **KardRate** в *кГц*. **KardRate** рассчитывается исходя из величины AdcPars->ChannelsQuantity, а также уже скорректированных AdcPars->AdcRate и AdcPars->InterKadrDelay. Дополнительно про соотношения между упомянутыми выше величинами AdcPars->ChannelsQuantity, AdcPars->AdcRate, AdcPars->InterKadrDelay и AdcPars->KadrRate смотри *§ 3.2.4. "Формат кадра отсчетов"*.

Передаваемые параметры:

- *AdcPars* – адрес структуры типа *ADC_PARS_E154* с требуемыми параметрами функционирования АЦП.

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

4.5.5. Получение текущих параметров работы АЦП

Формат: **BOOL** ***GET_ADC_PARS***(*ADC_PARS_E154 * const AdcPars)*

Назначение:

Данная функция считывает из модуля *E-154* всю текущую информацию, которая используется при сборе данных с АЦП. Нельзя вызывать эту функцию в процессе сбора данных. Её следует применять только после выполнения функции *STOP_ADC()*.

Передаваемые параметры:

- *AdcPars* – адрес структуры типа *ADC_PARS_E154* с полученными из модуля текущими параметрами функционирования АЦП.

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

4.5.6. Получение массива данных с АЦП

Формат: **BOOL** ***ReadData***(*IO_REQUEST_LUSBAPI * const ReadRequest*)

Назначение:

Данная функция предназначена для извлечения из модуля *E-154* очередной порции собранных данных АЦП. Эта функция должна использоваться совместно с функциями *START_ADC()* и *STOP_ADC()*.

Поля передаваемой структуры типа *IO_REQUEST_LUSBAPI* определяют параметры и требуемый режим получения данных с модуля *E-154*. Назначения полей этой структуры приведены в таблице ниже:

Название поля	Описание
Buffer	***Буфер данных.*** Чтение. **Buffer** предназначен для хранения получаемых с модуля данных АЦП. Перед его использованием в функции приложение само должно позаботиться о выделении достаточного кол-ва памяти под этот буфер. Полученные данные в буфере будут располагаться по-кадрово: 1ый кадр, 2ой кадр и т.д. Причём положение отсчётов в кадрах будет совпадать с порядком размещения соответствующих *логических каналов* в управляющей таблице **ControlTable**.
NumberOf WordsToPass	***Кол-во передаваемых данных.*** Запись–Чтение. Данный параметр задаёт то кол-во отсчётов АЦП, которое данная функция просто обязана стребовать с модуля. Величина параметра NumberOfWordsToPass должна находиться в диапазоне от 32 до (1024*1024), а также быть кратной 32. В противном случае данная функция сама подкорректирует величину этого поля, и по возвращении из функции в нём будет находиться *реально* использованное значение кол-ва затребованных данных.

NumberOfWordsPassed

Кол-во переданных данных. Чтение. В данном параметре возвращается то кол-во отсчётов АЦП, которое данная функция реально получила из модуля. Для *асинхронного* режима работы данной функции (см. ниже поле Overlapped) в этом параметре вполне может вернуться число 0, что *не является* ошибкой, учитывая специфику данного режима.

Overlapped

Структура Overlapped. Данное поле определяет, в каком именно режиме будет выполняться данная функция: *синхронном* или *асинхронном*:

- Overlapped = NULL. В этом случае от функции требуется *синхронный* режим её выполнения. При этом функция честно пытается получить из модуля все затребованные данные, причём в течение всего этого времени функция не возвращает управление вызвавшему её приложению. Если в течение времени TimeOut *мс* (см. ниже) все требуемые данные из модуля не получены, то функция завершается и возвращает ошибку.
- Overlapped != NULL. В этом случае от функции требуется *асинхронный* режим её выполнения. Подразумевается, что этому полю приложение уже присвоило указатель на заранее подготовленную структуру типа *OVERLAPPED*. В этом режиме данная функция выставляет системе, т.е. Windows, *асинхронный* запрос на получение требуемого кол-ва данных из модуля и сразу возвращает управление приложению. Т.е. происходит как бы полное перекладывание задачи по сбору данных на *ядро* системы. Так как *асинхронный* запрос выполняется уже на уровне *ядра*, то пока оно его обрабатывает, приложение вполне может заниматься своими собственными задачами. Окончание же текущего *асинхронного* запроса приложение можно отслеживать с помощью стандартных *Windows API* функций, таких как: *WaitForSingleObject()*, *GetOverlappedResult()* или *HasOverlappedIoCompleted()*. Данные функции используют событие *Event*, которые предварительно уже должно было быть определено приложением в соответствующем поле структуры Overlapped. Событие *Event* активируется системой по окончании сбора *всех* затребованных данных, завершая тем самым текущий *асинхронный* запрос. В ряде случаев бывает просто необходимо прервать выполняющийся *асинхронный* запрос. Для этой цели и существует штатная *Windows API* функция *CancelIo()*. К сожалению, существует эта функция только в *Windows NT* подобных системах.

TimeOut

Время ожидания сбора данных. Это поле предназначено для использования только в *синхронном* режиме. Задаёт максимальное время ожидания выполнения *синхронного* запроса на сбор данных в *мс*. Если по истечении этого времени *все* затребованные данные недополучены, функция завершается и возвращает ошибку.

В модуле *E-154* существует внутренний *FIFO* буфер АЦП, размер которого составляет 11 кБ. Такой буфер необходим для уверенного и надёжного сбора данных на предельных частотах работы АЦП. Так при частотах сбора порядка 120 кГц переполнение буфера произойдёт только через ~48 мс, что является вполне достаточным периодом времени даже для такой ‘*задумчивой*’ системы как *Windows*.

Теперь следует упомянуть о некоторой специфике, присущей режимам данной функции:

1. *Синхронный* режим. Данный режим рекомендуется применять при организации однократного сбора данных, в котором количество отсчётов не превышает 1024*1024 = 1 МСлова. В этом режиме функция ***ReadData()*** должна вызываться

ТОЛЬКО после успешного исполнения *START_ADC()*, которой, в принципе, должна предшествовать функция *STOP_ADC()*. Следует с большой осторожностью использовать данный режим при достаточно медленных частотах сбора и большом количестве запрашиваемых данных. Иначе данная функция может надолго '*уйти*' в сбор данных и, следовательно, весьма длительное время не возвращать управление приложению. Пример корректного использования функций библиотеки `Lusbapi` в *синхронном* режиме в виде консольного приложения можно найти на нашем CD-ROM в директории `\E-154\Examples\Borland C++ 5.02\ReadDataSynchro`.

2. *Асинхронный* режим. Этот режим функционально намного гибче, чем *синхронный* режим и его рекомендуется использовать при организации непрерывного *потокового* сбора данных, когда количество вводимых отсчётов превышает 1024*1024 = 1 МСлов. Данный режим позволяет организовывать в системе *Windows* очередь *асинхронных* запросов. Так можно сформировать очередь предварительных запросов даже непосредственно перед запуском сбора данных, но после функции *STOP_ADC()*. Такое использование очереди запросов позволяет резко повысить надёжность сбора данных. Операционная система *Windows* не является, что называется, средой реального времени. Поэтому, работая в ней, как это обычно бывает, всего лишь на *пользовательском* уровне, а не на уровне *ядра*, никогда нельзя быть полностью уверенным в том, что система в самый нужный момент не отвлечётся на свои собственные нужды на более или менее продолжительный промежуток времени. Например, если для частоты сбора 100 кГц после *START_ADC()*, но перед началом выполнения функции получения данных ***ReadData()*** система '*задумалась*' более чем на *327 мс* (что бывает очень редко, но вполне возможно), то сбой в принимаемых данных практически обеспечен. Однако если непосредственно перед *START_ADC()* выставить с помощью ***ReadData()*** несколько (можно и один) предварительных запросов, которые будут отрабатываться уже на уровне *ядра* системы, то сбоев не будет. Это происходит потому, что время отклика на отработку какого-нибудь события (в нашем случае это запрос) на уровне *ядра* существенно меньше, чем на *пользовательском* уровне. Т.о. получается, что после выполнения функции *START_ADC()* у нас уже есть готовые к обслуживанию запросы на уровне *ядра* системы. Практически никаких задержек. А теперь, пока система выполняет наши предварительные запросы, можно не торопясь, по мере надобности, выставлять один или несколько следующих запросов. Важно понимать, что для каждого выставленного в очередь или уже выполняющегося запроса у приложения должен существовать свой экземпляр структуры типа *IO_REQUEST_LUSBAPI* со своим индивидуальным событием *Event*.

Передаваемые параметры:

- *ReadRequest* – структура типа *IO_REQUEST_LUSBAPI* с параметрами извлечения готовых данных АЦП из модуля *E-154*.

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

4.5.7. Получение текущего состояния аппаратного FIFO буфера модуля E-154

Формат: **BOOL** **FIFO_STATUS**(DWORD *FifoOverflowFlag, double *FifoMaxPercentLoad, DWORD *FifoSize, DWORD *MaxFifoBytesUsed*)*

Назначение:

Во время ввода данных с АЦП в модуле E-154 данные проходят через FIFO буфер размером 11 кБ для того, чтобы не произошло потери данных, если операционная система задержит выполнение приложения. При помощи функции FIFO_STATUS() пользователь может узнать, не произошло ли переполнение FIFO буфера модуля, а также может отслеживать интенсивность использования FIFO буфера. Естественно, данную функцию можно вызывать в процессе сбора данных.

Передаваемые параметры:

- *FifoOverflowFlag* – указатель на переменную, в которую будет записан флаг переполнения FIFO буфера. Данный флаг сбрасывается в ноль при вызове функции *START_ADC()* и устанавливается в единицу, если в процессе сбора модуль обнаружит переполнение FIFO буфера;
- *FifoMaxPercentLoad* - указатель на переменную, в которую будет записана интенсивность использования FIFO буфера. Эта переменная сбрасывается в ноль при вызове функции *START_ADC()*, в процессе сбора переменная отражает максимальное использование FIFO буфера в процентах. В норме, в отсутствие серьезных задержек операционной системы, данный параметр должен быть в диапазоне 2-8 процентов;
- *FifoSize* - указатель на переменную, в которую будет записан размер FIFO буфера модуля E-154 в байтах. На момент составления данного описания размер FIFO буфера модуля E-154 составлял 11 кБ, в будущих версиях встроенного ПО для процессора ARM возможно изменение этого параметра, поэтому для пользователя оставлена возможность выяснить размер FIFO буфера.
- *MaxFifoBytesUsed* - указатель на переменную, в которую будет записан максимальный размер в байтах использованного FIFO буфера. Эта переменная сбрасывается в ноль при вызове функции *START_ADC()*, в процессе сбора переменная отражает максимальное использование FIFO буфера в байтах. По сути, параметры данной функции связаны соотношением:
  FifoMaxPercentLoad=100. MaxFifoBytesUsed/ FifoSize.*

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

4.5.8. Ввод кадра отсчетов с АЦП

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

Назначение:

Данная интерфейсная функция позволяет осуществлять ввод кадра отсчётов с АЦП модуля. Эта функция удобна для осуществления достаточно *медленного* (порядка нескольких десятков *Гц*) асинхронного ввода целого кадра данных с требуемых входных аналоговых каналов. Такие параметры сбора кадра, как количество опрашиваемых каналов (не более 16^x), частота работы АЦП и т.д., должны предварительно быть заданы с помощью штатной интерфейсной функции *SET_ADC_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 `\E-154\Example\Borland C++ 5.02\AdcKadr`. Данную функцию нельзя вызывать во время сбора данных с АЦП.

Передаваемые параметры:

- *Data* – указатель на массив, в который складываются полученный кадр отсчётов с АЦП.

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

4.5.9. Однократный ввод с АЦП

Формат:	**BOOL**	***ADC_SAMPLE***(*SHORT * const AdcData, WORD AdcChannel*)

Назначение:

Данная функция устанавливает заданный логический канал и осуществляет его однократное аналого-цифровое преобразование. Эта функция удобна для осуществления достаточно *медленного*, порядка нескольких десятков *Гц*, асинхронного ввода данных с задаваемового логического канала АЦП (см. *§ 3.2.3. "Логический номер канала АЦП"*). Более подробно смотри исходные тексты примера из директории `\E-154\Example\Borland C++ 5.02\AdcSample`. Данную функцию нельзя вызывать во время сбора данных с АЦП.

Передаваемые параметры:

- *AdcSample* – результат преобразования по заданному логическому каналу АЦП *AdcChannel*;
- *AdcChannel* – требуемый логический номер канала АЦП.

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

4.5.10. Преобразование одного отсчета АЦП в физическую величину

Формат: **BOOL** ***ProcessOnePoint***(*SHORT AdcData, double *Destination, DWORD AdcChannel, BOOL calibr, BOOL voltage*)

Назначение:

Данная функция позволяет осуществить преобразование кода АЦП в физическую величину (в вольты) с применением встроенных калибровочных коэффициентов. Для корректного преобразования функции необходимо передать в функцию сам код АЦП и логический номер канала АЦП, с которого был получен преобразуемый код АЦП (см. *§ 3.2.3. "Логический номер канала АЦП"*). Более подробно смотри исходные тексты примера из директории `\E-154\Example\Borland C++ 5.02\AdcSample`. Если функция используется во время сбора данных с АЦП, до начала вызова функции *START_ADC()* необходимо вызвать функцию ***GET_MODULE_DESCRIPTION().*** Дело в том, что функция ***GET_MODULE_DESCRIPTION()*** считывает калибровочные коэффициенты из модуля, которые понадобятся для функции ***ProcessOnePoint()***. Если этого не сделать, функция ***ProcessOnePoint()*** обнаружив, что калибровочные коэффициенты не были считаны, сама вызовет функцию ***GET_MODULE_DESCRIPTION()***, что нельзя делать во время ввода данных с АЦП.

Передаваемые параметры:

- *AdcData* – код АЦП, который необходимо преобразовать;
- *AdcChannel* –логический номер канала АЦП, с которого был получен код АЦП;
- Destination - адрес переменной, в которую будет помещен результат преобразования;
- calibr - включает режим использования встроенных калибровочных коэффициентов
  - calibr=TRUE - при преобразовании кода АЦП будут использоваться встроенные калибровочные коэффициенты;
  - calibr=FALSE - при преобразовании кода АЦП не будут использоваться встроенные калибровочные коэффициенты
- voltage - включает режим преобразования в вольты
  - voltage =TRUE - код АЦП будет преобразован в вольты;
  - voltage =FALSE - преобразование кода АЦП в вольты осуществляться не будет

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

4.5.11. Обработка массива с кодами АЦП (преобразование кодов в вольты)

Формат: **BOOL** ***ProcessArray***(*SHORT *AdcData, double *Destination, DWORD Size, BOOL calibr, BOOL voltage*)

Назначение:

Данная функция позволяет осуществить преобразование введенного массива АЦП в физические величины (в вольты) с применением встроенных калибровочных коэффициентов. Для корректного преобразования необходимо передать в функцию массив с кодами АЦП. Также для корректной работы этой функции необходимо, чтобы параметры ввода (число каналов и массив с логическими номерами каналов), устанавливаемые с помощью функции ***SET_ADC_PARS***(), соответствовали параметрам, при которых был осуществлен ввод преобразуемого массива АЦП. Более подробно смотри исходные тексты примера из директории `\E-154\Example\Borland C++ 5.02\AdcKadr`. Если функция используется во время сбора данных с АЦП, до начала вызова функции *START_ADC()* необходимо вызвать функцию ***GET_MODULE_DESCRIPTION().*** Дело в том, что функция ***GET_MODULE_DESCRIPTION()*** считывает калибровочные коэффициенты из модуля, которые понадобятся для функции ***ProcessArray ()***. Если этого не сделать, функция ***ProcessArray ()*** обнаружив, что калибровочные коэффициенты не были считаны, сама вызовет функцию ***GET_MODULE_DESCRIPTION()***, что нельзя делать во время ввода данных с АЦП.

Передаваемые параметры:

- *AdcData* – указатель на массив с кодами АЦП, который необходимо преобразовать;
- Destination - указатель на массив, в который будет помещен результат преобразования;
- Size - размер массива (число преобразуемых отсчетов АЦП)
- calibr - включает режим использования встроенных калибровочных коэффициентов
  - calibr=TRUE - при преобразовании кода АЦП будут использоваться встроенные калибровочные коэффициенты;
  - calibr=FALSE - при преобразовании кода АЦП не будут использоваться встроенные калибровочные коэффициенты
- voltage - включает режим преобразования в вольты
  - voltage =TRUE - код АЦП будет преобразован в вольты;
  - voltage =FALSE - преобразование кода АЦП в вольты осуществляться не будет

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

4.6. Функции для работы с ЦАП

Аппаратура модуля *E-154* и, соответственно, библиотека `Lusbapi` позволяет управлять выводом на ЦАП только асинхронным (однократным) образом. Т.о. вывод на ЦАП получается сравнительно медленной операцией, т.к. на модуле не реализована аппаратная поддержка *потоковой* работы с ЦАП. Примеры корректного применения интерфейсной функции для работы с ЦАП можно найти в директории `\E-154\Examples\Borland C++ 5.02\DacSample`.

4.6.1. Корректировка данных ЦАП

Схемотехника и использованные компоненты обеспечивают линейность передаточной характеристики ЦАП тракта модуля *E-154*. Однако на *сегодняшний* день модуль не умеет производить автоматическую корректировку выводимых на ЦАП данных. Это приводит к тому, что выходные показания ЦАП могут иметь некоторое смещение нуля и неточность в передаче масштаба. Поэтому на уровне приложения необходима реализация всей этой нудной задачи по корректировке данных ЦАП. Для этих целей предназначены соответствующие калибровочные коэффициенты, хранящиеся в служебной информации модуля. Служебная информация совместно с нужными коэффициентами записывается в модуль на этапе при его наладке в **ЗАО "Л-Кард"**. Благодаря этому на модуле отсутствуют подстроечные резисторы, что улучшает шумовые характеристики модуля и увеличивает их надежность.

Сами коэффициенты располагаются в полях *Dac.OffsetCalibration[0]* и *Dac.ScaleCalibration[0]* структуры служебной информации *MODULE_DESCRIPTION_E154*. Эти поля представляют из себя массивы типа *double*. Для модуля *E-154* в каждом из этих массивов используются только первые *DAC_CALIBR_COEFS_QUANTITY_E154* элементов. Массив *Dac.OffsetCalibration* содержит коэффициенты для корректировки смещение нуля каналов ЦАП, а массив *Dac.ScaleCalibration* – для корректировки масштаба.

Корректировка данных ЦАП производится следующим образом: Y = (X+A)*B, где: X – некорректированные данные ЦАП [в кодах ЦАП], Y – скорректированные данные ЦАП [в кодах ЦАП], A – коэффициент смещения нуля [в кодах ЦАП], B – коэффициент масштаба [безразмерный].

Например, на ЦАП необходимо выставить напряжения, соответствующие следующим кодам ЦАП: X1 = 1000, X2 = -1000, X3 = 0. Тогда коэффициенты коррекции и данные для ЦАП можно представить так: A = Dac.OffsetCalibration[0], B = Dac.ScaleCalibration[0], Y1=(A+1000)*B, Y2=(A-1000)*B, Y3=A*B.

4.6.2. Однократный вывод на ЦАП

Формат: **BOOL** ***DAC_SAMPLE***(*SHORT * const DacData, WORD DacChannel*)

Назначение:

Функция *DAC_SAMPLE()* сохранена для совместимости с другими модулями. Для новых разработок рекомендуется использовать функцию ***DAC_SAMPLE_VOLT***(). Данная функция позволяет однократно устанавливать на ЦАП напряжение в соответствии со значением *DacData* (в кодах ЦАП). Ожидается, что величина *DacData* должна находиться в диапазоне от –128 до 127, иначе функция автоматически ограничит значение *DacData* указанными пределами. Функция ***DAC_SAMPLE()*** выполняется достаточно *медленно* и, используя её, можно достичь частоты вывода данных на ЦАП порядка нескольких десятков *Гц*. О соответствии кода ЦАП величине устанавливаемого на выходе модуля аналогового напряжения см. *§ 3.2.2. "Формат слова данных для ЦАП"*. Данную функцию можно вызывать во время сбора данных с АЦП.

Передаваемые параметры:
- *DacData* – устанавливаемое значение напряжения в кодах ЦАП (от –128 до 127).
- *DacChannel* – зарезервирован (параметр оставлен для обеспечения совместимости с другими устройствами).

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

4.6.3. Однократный вывод на ЦАП с учетом калибровочных коэффициентов

Формат: **BOOL** ***DAC_SAMPLE_VOLT*(double** *const DacDataVolt, BOOL calibr*)

Назначение:

Данная функция позволяет однократно устанавливать на ЦАП напряжение в соответствии со значением *DacDataVolt* (значение задается непосредственно в вольтах). Ожидается, что величина *DacDataVolt* должна находиться в диапазоне от -5.0 до +5.0, иначе функция автоматически ограничит значение *DacDataVolt* указанными пределами. Функция ***DAC_SAMPLE_VOLT()*** выполняется достаточно *медленно* и, используя её, можно достичь частоты вывода данных на ЦАП порядка нескольких десятков *Гц*. О соответствии кода ЦАП величине устанавливаемого на выходе модуля аналогового напряжения см. *§ 3.2.2. "Формат слова данных для ЦАП"*. Данную функцию можно вызывать во время сбора данных с АЦП. Если функция используется во время сбора данных с АЦП, до начала вызова функции *START_ADC()* необходимо вызвать функцию ***GET_MODULE_DESCRIPTION().*** Дело в том, что функция ***GET_MODULE_DESCRIPTION()*** считывает калибровочные коэффициенты из модуля, которые понадобятся для функции ***DAC_SAMPLE_VOLT ()***. Если этого не сделать, функция ***DAC_SAMPLE_VOLT ()*** обнаружив, что калибровочные коэффициенты не были считаны, сама вызовет функцию ***GET_MODULE_DESCRIPTION()***, что нельзя делать во время ввода данных с АЦП.

Передаваемые параметры:
- *DacDataVolt* – устанавливаемое значение напряжения в вольтах (от –5.0 до +5.0).
- *calibr* – значение TRUE включает использование калибровочных коэффициентов при выводе на ЦАП.

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

4.7. Функции для работы с цифровыми линиями

Все входные и выходные цифровые линии модуля *E-154* располагаются на внешнем разъёме ***DIGITAL I/O***. По умолчанию, непосредственно после подачи на модуль внешнего напряжения питания все выходные цифровые линии находятся в *высокоимпедансном* состоянии.

Аппаратура модуля *E-154* и, соответственно, библиотека `Lusbapi` позволяет работать с цифровыми линиями только асинхронным (однократным) образом. Т.о. работа с цифровыми линиями получается сравнительно медленной операцией, т.к. на модуле не предусмотрена аппаратная поддержка *потоковой* работы с ними.

4.7.1. Разрешение выходных цифровых линий.

Формат:	**BOOL**	***ENABLE_TTL_OUT***(*BOOL EnableTtlOut*)

Назначение:

Данная интерфейсная функция позволяет осуществлять управление разрешением *всех* выходных линий внешнего цифрового разъёма ***DIGITAL I/O***. Т.о. существует возможность перевода их в третье, *высокоимпедансное*, состояние и обратно. Данную функцию можно вызывать во время сбора данных с АЦП.

Передаваемые параметры:

- *EnableTtlOut* – флажок, управляющий состоянием разрешения всех цифровых выходных линий.

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

4.7.2. Чтение внешних цифровых линий

Формат:	**BOOL**	***TTL_IN***(*WORD * const TtlIn*)

Назначение:

Данная интерфейсная функция осуществляет однократное асинхронное чтение состояний *всех* $8^{\text{ми}}$ входных цифровых линий на внешнем разъёме ***DIGITAL I/O***. Функция ***TTL_IN()*** выполняется достаточно *медленно* и, используя её, можно достичь частоты ввода данных с цифровых линий порядка нескольких сотен *Гц*. Данную функцию можно вызывать во время сбора данных с АЦП.

Передаваемые параметры:

- *TtlIn* – переменная, содержащая побитовое состояние входных цифровых линий.

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

4.7.3. Вывод на внешние цифровые линии

Формат:	**BOOL**	***TTL_OUT***(*WORD TtlOut*)

Назначение:

Данная интерфейсная функция осуществляет установку *всех* 8[ми] выходных цифровых линий на внешнем цифровом разъёме ***DIGITAL I/O*** модуля *E-154* в соответствии с битами передаваемого параметра *TtlOut*. Если есть необходимость, то работа с цифровыми выходами предварительно должна быть разрешена с помощью интерфейсной функции *ENABLE_TTL_OUT()*. Функция *TTL_OUT()* выполняется достаточно *медленно* и, используя её, можно достичь частоты вывода данных на цифровые линии порядка нескольких сотен *Гц*. Данную функцию можно вызывать во время сбора данных с АЦП.

Передаваемые параметры:

- *TtlOut* – переменная, содержащая побитовое состояние устанавливаемых выходных цифровых линий.

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

4.8. Функции для работы с пользовательским ППЗУ

На модуле *E-154* часть памяти микроконтроллера отведена под пользовательское ППЗУ. Размер этой области составляет *USER_FLASH_SIZE_E154* байт и представляет собой 128 байт. Всю эту область пользователь может смело использовать в своих сугубо частнособственнических интересах.

Для предотвращения случайной порчи содержимого пользовательского ППЗУ модуля, предусмотрена специальная последовательность записи. Для осуществления операции записи, предварительно следует разблокировать ППЗУ для записи при помощи интерфейсной функции *ENABLE_FLASH_WRITE()*. А по окончании всей процедуры записи заблокировать ППЗУ при помощи той же интерфейсной функции *ENABLE_FLASH_WRITE()*.

4.8.1. Разрешение записи в ППЗУ

Формат: **BOOL** ***ENABLE_FLASH_WRITE***(*BOOL IsUserFlashWriteEnabled*)

Назначение:

Данная интерфейсная функция разрешает (*TRUE*) либо запрещает (*FALSE*) режим записи в пользовательское ППЗУ модуля с помощью штатной интерфейсной функции *WRITE_FLASH_ARRAY()*. Следует помнить, что после завершения всех требуемых операций записи информации в пользовательское ППЗУ, необходимо с помощью данной интерфейсной функции запретить режим записи. Данную функцию нельзя вызывать во время сбора данных с АЦП.

Передаваемые параметры:

- *EnableFlashWrite* – переменная может принимать следующие значения:
  - ✓ если *TRUE*, то режим записи в пользовательское ППЗУ разрешен,
  - ✓ если *FALSE*, то режим записи в пользовательское ППЗУ запрещен.

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

4.8.2. Запись массива в ППЗУ

Формат: **BOOL** ***WRITE_FLASH_ARRAY***(*unsigned char * const UserFlash*)

Назначение:

Данная интерфейсная функция выполняет запись массива из 128 байт в пользовательское. Перед началом операции записи в ППЗУ необходимо разрешить её с помощью интерфейсной функции *ENABLE_FLASH_WRITE()*. После окончания цикла записи настоятельно рекомендуется запретить режим записи в пользовательское ППЗУ с помощью той же функции *ENABLE_FLASH_WRITE()*. Данную функцию нельзя вызывать во время сбора данных с АЦП.

Передаваемые параметры:

- *UserFlash* – массив из 128 байт.

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

4.8.3. Чтение массива из ППЗУ

Формат:	**BOOL**	***READ_FLASH_ARRAY***(unsigned char * *const UserFlash*)
Назначение: Данная интерфейсная функция считывает 128 байт из пользовательского ППЗУ. Данную функцию нельзя вызывать во время сбора данных с АЦП.		
Передаваемые параметры: • *UserFlash* – указатель на массив, в который будут считаны 128 байт.		
Возвращаемое значение:	*TRUE* – функция успешно выполнена; *FALSE* – функция выполнена с ошибкой.	

4.9. Функции для работы со служебной информацией

Служебная информация содержит самые общие данные об используемом модуле *E-154*: название модуля, его серийный номер и ревизию, корректировочные коэффициенты для АЦП и ЦАП, версию текущей прошивки MCU, тактовую частоту работы исполнительного устройства MCU и многое другое. Некоторые данные из этой служебной информации необходимы функциям штатной библиотеки `Lusbapi` для своей корректной работы.

4.9.1. Чтение служебной информации

Формат: **BOOL** ***GET_MODULE_DESCRIPTION***(
*MODULE_DESCRIPTION_E154 * const ModuleDescription*)

Назначение:

Данная интерфейсная функция осуществляет чтение всей служебной информации в структуру типа *MODULE_DESCRIPTION_E154*. Эта информация требуется при работе с некоторыми интерфейсными функциями штатной библиотеки `Lusbapi`. Поэтому данную функцию, во избежания непредсказуемого поведения приложений, следует обязательно вызывать непосредственно после открытия интерфейса с модулем (см. *§ 4.1. "Общие принципы работы с модулем"*). Данную функцию нельзя вызывать во время сбора данных с АЦП.

Передаваемые параметры:

- *ModuleDescription* – указатель на структуру типа *MODULE_DESCRIPTION_E154*, в которую считывается вся служебная информация модуля.

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

4.9.2. Запись служебной информации

Формат: **BOOL** ***SAVE_MODULE_DESCRIPTION***(
*MODULE_DESCRIPTION_E154 * const ModuleDescription*)

Назначение:

Данная интерфейсная функция позволяет сохранять в модуле всю служебную информацию из структуры типа *MODULE_DESCRIPTION_E154*. **!!!Внимание!!!** Применять данную функцию нужно только в случае крайней необходимости. Например, когда по тем или иным обстоятельствам испортилось содержимое служебной информации. Особое внимание надо обратить на то, что при выполнении данной функции будут изменены калибровочные коэффициенты. Порча калибровочных коэффициентов может привести к возникновению большой погрешности при преобразовании кодов АЦП или ЦАПа в вольты. Данную функцию нельзя вызывать во время сбора данных с АЦП.

Передаваемые параметры:

- *ModuleDescription* – указатель на структуру типа *MODULE_DESCRIPTION_E154*, из которой служебная информация переносится в модуль.

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

Приложение А.
Вспомогательные константы и типы

Вспомогательные константы и типы данных описаны в заголовочном файле `\DLL\Include\LusbapiTypes.h` и рассмотрены в нижеследующих разделах.

А.1. Константы

Вспомогательные константы, определённые в библиотеке `Lusbapi`, приведены в следующей таблице:

Название	Значение	Смысл
NAME_LINE_LENGTH_LUSBAPI	25	Длина строки с названием чего-либо. Например, название производителя или изделия, имя автора и т.д.
COMMENT_LINE_LENGTH_LUSBAPI	256	Длина строки с комментарием в какой-либо вспомогательной структуре.
ADC_CALIBR_COEFS_QUANTITY_LUSBAPI	128	Максимально возможное число корректировочных коэффициентов АЦП.
DAC_CALIBR_COEFS_QUANTITY_LUSBAPI	128	Максимально возможное число корректировочных коэффициентов ЦАП.

А.2. Структура VERSION_INFO_LUSBAPI

Вспомогательная структура *VERSION_INFO_LUSBAPI* содержит более или менее подробную информацию о программном обеспечении, работающем в каком-либо исполнительном устройстве: MCU, DSP, PLD и т.д. Данная структура описывается следующим образом:

```
struct VERSION_INFO_LUSBAPI
{
    BYTE Version[10];                   // версия ПО для исполнительного устройства
    BYTE Date[14];                      // дата сборки ПО
    BYTE Manufacturer[NAME_LINE_LENGTH_LUSBAPI];    // производитель ПО
    BYTE Author[NAME_LINE_LENGTH_LUSBAPI];          // автор ПО
    BYTE Comment[COMMENT_LINE_LENGTH_LUSBAPI];      // строка комментария
};
```

А.3. Структура MODULE_INFO_LUSBAPI

Данная вспомогательная структура *MODULE_INFO_LUSBAPI* содержит самую общую информацию о модуле: название фирмы-изготовителя изделия, название изделия, серийный номер изделия, ревизия изделия и строка комментария. Эта структура описывается следующим образом:

```
struct MODULE_INFO_LUSBAPI
{
    BYTE CompanyName[NAME_LINE_LENGTH_LUSBAPI];  // название фирмы-изготовите
                                                 // ля изделия
    BYTE DeviceName[NAME_LINE_LENGTH_LUSBAPI];   // название изделия
    BYTE SerialNumber[16];                       // серийный номер изделия
    BYTE Revision;                               // ревизия изделия
    BYTE Comment[COMMENT_LINE_LENGTH_LUSBAPI];   // строка комментария
};
```

A.4. Структура INTERFACE_INFO_LUSBAPI

Вспомогательная структура *INTERFACE_INFO_LUSBAPI* содержит самую общую информацию об используемом интерфейсе для доступа к модулю. Данная структура описывается следующим образом:

```
struct INTERFACE_INFO_LUSBAPI
{
    BOOL Active;                    // флаг достоверности остальных полей структуры
    BYTE Name[NAME_LINE_LENGTH_LUSBAPI];          // название интерфейса
    BYTE Comment[COMMENT_LINE_LENGTH_LUSBAPI];    // строка комментария
};
```

A.5. Структура MCU_INFO_LUSBAPI

Вспомогательная структура *MCU_INFO_LUSBAPI* содержит самую общую информацию об используемом исполнительном устройстве типа микроконтроллер (MCU). Данная структура описывается следующим образом:

```
struct MCU_INFO_LUSBAPI
{
    BOOL   Active;                  // флаг достоверности остальных полей структуры
    BYTE   Name[NAME_LINE_LENGTH_LUSBAPI]; // название MCU
    double ClockRate;                      // тактовая частота работы MCU в кГц
    VersionType Version;                   // информация о программе MCU
    BYTE   Comment[COMMENT_LINE_LENGTH_LUSBAPI];    // строка комментария
};
```

A.6. Структура ADC_INFO_LUSBAPI

Вспомогательная структура *ADC_INFO_LUSBAPI* содержит самую общую информацию об используемом устройстве типа АЦП. Данная структура описывается следующим образом:

```
struct ADC_INFO_LUSBAPI
{
    BOOL   Active;          // флаг достоверности остальных полей структуры
    BYTE   Name[NAME_LINE_LENGTH_LUSBAPI];       // название АЦП
    double OffsetCalibration[ADC_CALIBR_COEFS_QUANTITY_LUSBAPI];
    // корректировочные коэффициенты смещения нуля АЦП
    double ScaleCalibration[ADC_CALIBR_COEFS_QUANTITY_LUSBAPI];
    // корректировочные коэффициенты масштаба АЦП
    BYTE   Comment[COMMENT_LINE_LENGTH_LUSBAPI];    // строка комментария
};
```

A.7. Структура DAC_INFO_LUSBAPI

Вспомогательная структура *DAC_INFO_LUSBAPI* содержит самую общую информацию об используемом устройстве типа ЦАП. Данная структура описывается следующим образом:

```
struct DAC_INFO_LUSBAPI
{
    BOOL   Active;                  // флаг достоверности остальных полей структуры
    BYTE   Name[NAME_LINE_LENGTH_LUSBAPI];       // название ЦАП
    double OffsetCalibration[DAC_CALIBR_COEFS_QUANTITY_LUSBAPI];
    // корректировочные коэффициенты смещения нуля ЦАП
    double ScaleCalibration[DAC_CALIBR_COEFS_QUANTITY_LUSBAPI];
    // корректировочные коэффициенты масштаба ЦАП
```

```
    BYTE    Comment[COMMENT_LINE_LENGTH_LUSBAPI];      // строка комментария
};
```

A.8. Структура DIGITAL_IO_INFO_LUSBAPI

Вспомогательная структура *DIGITAL_IO_INFO_LUSBAPI* содержит самую общую информацию об используемых устройствах цифрового ввода-вывода. Данная структура описывается следующим образом:

```
struct DIGITAL_IO_INFO_LUSBAPI
{
    BOOL Active;                        // флаг достоверности остальных полей структуры
    BYTE Name[NAME_LINE_LENGTH_LUSBAPI];       // название цифровой микросхемы
    WORD InLinesQuantity;                      // кол-во входных линий
    WORD OutLinesQuantity;                     // кол-во выходных линий
    BYTE Comment[COMMENT_LINE_LENGTH_LUSBAPI];      // строка комментария
};
```