L-CARD

Устройства для мобильных систем

E20-10

Внешний быстродействующий модуль на шину **USB 2.0**

Потоковый ввод с АЦП: 4 канала, 10 МГц, 14 бит
Асинхронный цифровой ввод-вывод: по 16 линий
Асинхронный вывод на ЦАП: 2 канала, 12 бит (опция)

Библиотека *Lusbapi 3.3.*

Windows'98/Me/2000/XP/Vista/7

Руководство программиста

Москва. Март 2011 г.
Ревизия документа А3

ООО *«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*
Санкт-Петербург:	ЗАО “АВТЭКС Санкт-Петербург”	*www.autex.spb.ru*
Санкт-Петербург:	Компания "Ниеншанц-Автоматика"	*www.nnz-ipc.ru*
Новосибирск:	ООО “Сектор Т”	*www.sector-t.ru*
Екатеринбург:	ООО “Авеон”	*www.aveon.ru*
Казань:	ООО “Шатл”	*www.shuttle.kazan.ru*

© Copyright 1989–2011, **ООО “Л Кард”**. Все права защищены.

Оглавление

1. Введение

Данное описание предназначено для пользователей, собирающихся разрабатывать свои собственные приложения в операционной среде *Windows'98/2000/XP/Vista/7* для работы с быстродействующими модулями *E20-10* от фирмы ООО "Л Кард". Предварительно настоятельно рекомендуется ознакомиться с *"E20-10. Руководство пользователя"*, где можно найти достаточно подробную техническую информацию о модуле, включая описание функциональной схемы, подключение входных сигналов, распиновка внешних разъёмов, характерные неисправности и многое другое.

Для быстродействующего модуля *E20-10* фирма ООО "Л Кард" поставляет **USB** драйвер устройства, готовую динамически подключаемую библиотеку `Lusbapi` с целым рядом законченных примеров. В качестве базового языка при написании библиотеки `Lusbapi` был выбран C++, а конкретнее, старый надёжный **Borland C++ 5.02**. Причём сама библиотека и все примеры поставляются вместе с исходными текстами, снабжёнными достаточно подробными комментариями. Штатная библиотека `Lusbapi` включает в себя множество разнообразных функций помогающих пользователю использовать все заложенные в модуль *E20-10* возможности.

Модуль *E20-10* разрабатывался с главной целью – обеспечить надежный высокоскоростной сбор в компьютер аналоговой информации. Для этого штатная библиотека `Lusbapi` содержит целый ряд функций, позволяющих организовывать многоканальный *непрерывный потоковый* сбор аналоговых данных на частотах АЦП вплоть до 10 МГц. При сборе аналоговой информации конечный пользователь может использовать широкий спекрт типов синхронизации ввода данных. Вывод же аналоговой (на ЦАП) и ввод/вывод цифровой информации реализован только в однократном, и поэтому относительно медленном, режиме. Мы надеемся, что описываемая ниже библиотека `Lusbapi` упростит и ускорит написание Ваших собственных *Windows*-приложений.

Весь пакет штатного программного обеспечения для работы с модулем *E20-10* в среде *Windows'98/2000/XP/Vista/7* находится на прилагаемом к модулю фирменном CD-ROM в директории `\USB\Lusbapi`. **!!!ВНИМАНИЕ!!!** Далее по тексту данного описания все директории указаны относительно неё. Также весь штатный софт можно скачать с нашего сайта *www.lcard.ru* из раздела *"Библиотека файлов"*. Там из подраздела *"ПО для внешних модулей"* следует выбрать самораспаковывающийся архив `lusbapi`***XY***`.exe`, где ***X.Y*** обозначает номер версии программного обеспечения. На момент написания данного руководства последняя библиотека *Lusbapi* имеет версию **3.3**, а содержащий её архив называется *lusbapi33.exe*.

2. Общие сведения

2.1. Что нового?

Как правило, в данном параграфе будут приводиться только основные изменения как аппаратного, так и программного характера. За более подробной информацией следует обращать к:

- *"E20-10. Руководство пользователя"*;
- *"E20-10. Библиотека Lusbapi. История дополнений и изменений"*.

2.1.1. Библиотека Lusbapi 3.3

В библиотеке `Lusbapi` версии **3.3** было сделано всего два небольших изменения, а именно:

- Модуль *E20-10* стал доступен в двух вариантах (исполнениях):
  - ✓ с полосой пропускания входного сигнала равной 1.25 МГц (базовое исполнение);
  - ✓ с полосой пропускания входного сигнала равной 5.0 МГц;

  С целью информирования пользователя о текущем исполнении модуля в структуру MODULE_INFO_LUSBAPI было введено новое числовое поле Modification.
- Для модуля *E20-10 (Rev.'A')* в функции ReadData() подправлены нижнее ограничение и кратность величины запроса NumberOfWordsToPass структуры IO_REQUEST_LUSBAPI. Раньше эти величины были равны 128 отсчётам, теперь – 256.

2.1.2. Библиотека Lusbapi 3.2

В начале 2008 г. начато производство новой ревизии модуля *E20-10 (Rev.'B')*. Данная модификация представляет собой продукт основательной аппаратной модернизации *E20-10 (Rev.'A')*. Библиотека `Lusbapi` версии **3.2** призвана осуществлять полную поддержку всех новых функций и свойств, появившихся у модуля *E20-10 (Rev.'B')*. Так как изменений достаточно много отметим лишь следующие основные отличия от предыдущей ревизии модуля:

- Процедура калибровки данных с АЦП теперь можно выполнять на уровне ПЛИС модуля;
- Расширен диапазон межкадровой задержки;
- Введены расширенные возможности синхронизации при работе с АЦП;
- Улучшена проверка состояния процесса сбора данных.

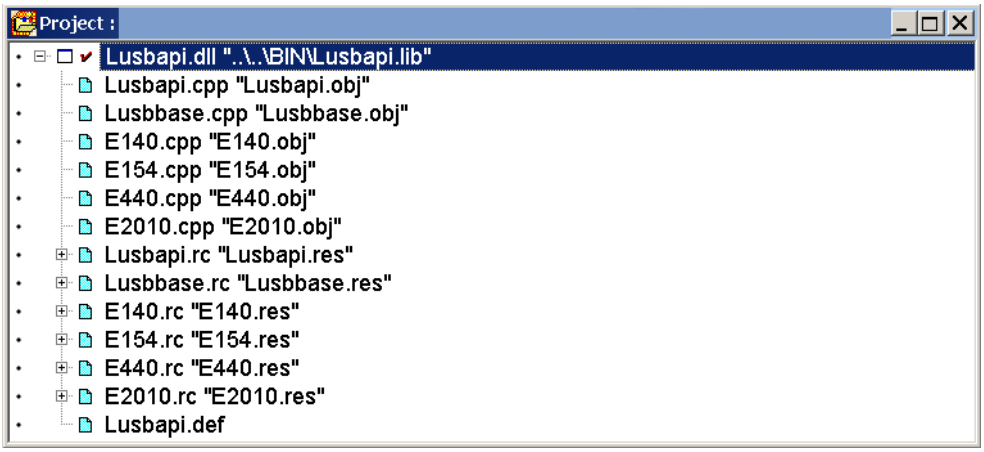
2.2. Подключение модуля E20-10 к компьютеру

Все подробности по процедуре аппаратного подключения модуля *E20-10* к компьютеру конечного пользователя и надлежащей установке **USB** драйверов можно найти в *"E20-10. Руководство пользователя, § 4 "Инсталляция и настройка"*.

Стоит особо подчеркуть, что, начиная с версии **3.2** в библиотеке `Lusbapi` изменился основной файл **USB** драйвера. Теперь он называется **`Ldevusbu.sys`** вместо бывшего ранее **`Ldevusb.sys`**. Т.о. при переходе со старых версий **Lusbapi** на более новую версию **3.2** и выше, конечному пользователю следует через *"Device Manager*" ("*Диспетчер устройств*") переключить модуль *E20-10* на работу с новым **USB** драйвером.

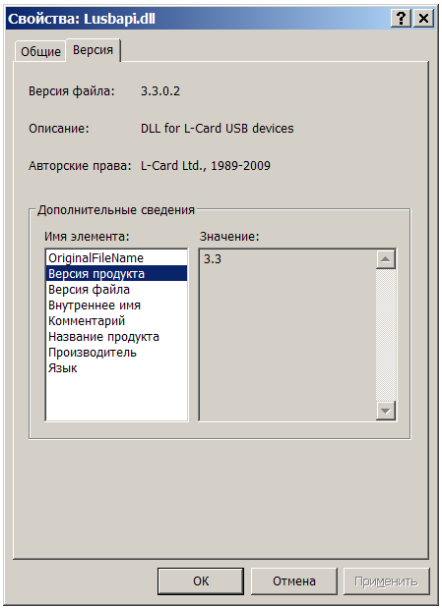
2.3. *Библиотека Lusbapi*

Штатная библиотека `Lusbapi` написана с использованием весьма доступного языка программирования **Borland C++ 5.02**. Кроме модуля *E20-10* в библиотеке также осуществлена поддержка модулей типа *E-154*, *E14-140* и *E14-440*. 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 *E20-10*. В дальнейшем, используя этот указатель, можно осуществлять доступ ко всем интерфейсным функциям штатной 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 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 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:

- *E20-10*\Examples\Borland C++ 5.02;
- *E20-10*\Examples\Borland C++ Builder 5.0;
- *E20-10*\Examples\Borland Delphi 6.0;
- *E20-10*\Examples\MicroSoft Visual C++ 6.0.

Например, для получения возможности вызова интерфейсных функций в пользовательском проекте на Borland C++ необходимо выполнить 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 *E20-10*;
- в принципе, с помощью функции *GetDllVersion()*, желательно сравнить версию используемой DLL библиотеки с версией текущего программного обеспечения;
- 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.

Поклонникам 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 *E20-10*\Examples\MicroSoft Visual C++ 6.0\DynLoad.
2. При статической компоновке штатной DLL в Вашем проекте использовать файл библиотеки импорта LUSBAPI.LIB из директории \DLL\Lib\MicroSoft.

При работе с модулем типа *E20-10* 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.

2.4. Микроконтроллер модуля

На модуле E20-10 в качестве ‘*рабочей лошадки*’ используется микроконтроллер (MCU) типа *AVR Atmega 162* от фирмы *Atmel Corporation*. MCU 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 MCU, является его двухкомпонентность. Т.е. он состоит как бы из двух частей: основной программы (Firmware) и загрузчика (BootLoader). Фирменный загрузчик, как впрочем, и основная программа, ‘*заливается*’ в MCU на этапе наладки модуля E20-10 в ООО “Л Кард” и конечный пользователь не имеет возможности его обновления без специального прошивочного кабеля. Но при этом BootLoader предоставляет возможность безболезненной перепрошивки Firmware модуля по **USB** шине, что является крайне удобным при обновлении версий основной программы. Самую последнюю версию Firmware MCU всегда можно скачать с нашего сайта *www.lcard.ru* из раздела *“Библиотека файлов”*. Там из подраздела *“Firmware и BIOS”* следует выбрать архив e2010fw_**WX**a_**YZ**b.zip, где **W.X** означает номер версии основной программы MCU для модуля *E20-10 (Rev.'A')*, а **Y.Z** – для модуля *E20-10 (Rev.'B')*. На момент написания данного руководства этот архив имеет имя *e2010fw_17a_21b.zip.zip*.

2.5. Загрузка модуля

В качестве одного из основных функциональных узлов модуля *E20-10* можно смело назвать *программируемую логическую интегральную схему* (ПЛИС) семейства *ACEX* (для модуля *Rev.'A'*) или *Cyclone* (для модуля *Rev.'B'*) от фирмы *Altera Corporation*. Основное функциональное назначение ПЛИС – осуществлять полное аппаратное управление потоковым вводом аналоговой информации. Применяемая ПЛИС обладает так называемой загружаемой архитектурой. Т.е. её необходимо загружать всякий раз после подачи питания на модуль, а также можно перезагружать уже в процессе работы модуля.

В состав штатного программного обеспечения в директории `\DLL\Source\` можно найти файлы готовых прошивок ПЛИС для различных ревизий модуля *E20-10*, а именно:

- `E2010.pld` — для модуля *E20-10 (Rev.'A')*;
- `E2010m.pld` — для модуля *E20-10 (Rev.'B')*.

Данные файлы также встроены в виде ресурсов в библиотеку `Lusbapi.dll`, которая имеет специальную интерфейсную функцию *LOAD_MODULE()* для корректной загрузки прошивки в ПЛИС модуля. Только после загрузки ПЛИС можно переходить непосредственно к самому управлению модулем, т.е. переводить его в различные режимы работы с АЦП, ЦАП и т. д.

2.6. Возможные проблемы при работе с модулем

1. Перед началом работы со штатным программным обеспечением модуля *E20-10*, 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*), не совсем корректно работают в среде *Windows'98/2000/XP/Vista/7*. Это проявляется при запросах с большим кол-вом 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 АЦП в *кГц*.
InterKadrDelay	Межкадровая задержка в *млс*.
KardRate	Частота кадра отсчётов в *кГц*.
Buffer	Целочисленный массив данных типа *SHORT*.
ControlTable	Управляющая таблица, содержащая целочисленный массив с *логическими* номерами каналов. Используется аппаратурой для организации циклического опроса каналов АЦП при сборе данных.
ControlTableLength	Размер управляющей таблицы.

3.2. Форматы данных

3.2.1. Формат слова данных с АЦП

Данные, поступающие с 14[ти] битного АЦП модуля *E20-10*, представляются в формате знакового целого двухбайтного числа от -8192 до 8191. Эти "*сырые*" отсчёты с АЦП рекомендуется откорректировать, например, с помощью *штатных (заводских)* корректировочных коэффициентов, которые хранятся в самом модуле и доступны с помощью штатной функции *GET_MODULE_DESCRIPTION()*. Процедура корректировки данных АЦП возможна как на верхнем программном уровне, так и на уровне ПЛИС модуля, и достаточно подробно описана в *§ 4.5.1. "Корректировка данных АЦП"*. Взаимосвязь откорректированного кода АЦП с входным напряжением приведена в таблице ниже:

Таблица 1. Соответствие откорректированного кода АЦП входному напряжению

Диапазон, В	Код АЦП	Напряжение, В
±3.0; ±1.0; ±0.3	+8000	+3.0; +1.0; +0.3
	0	0
	-8000	–3.0; –1.0; –0.3

3.2.2. Формат слова данных для ЦАП

На модуле по желанию пользователя может быть установлена микросхема 2^{x} канального $12^{ти}$ битного ЦАП. Для выставления какого-нибудь напряжения на выходе ЦАП в модуль *E20-10* необходимо передать $16^{ти}$ битное слово данных. Формат этого слова данных приведен в следующей таблице:

Таблица 2. Формат слова данных ЦАП

Номер бита	Назначение
<11..0>	$12^{ти}$ битный код ЦАП
12	Номер канала ЦАП: • ‘0’ – первый канал; • ‘1’ – второй канал.
<15..13>	Не используются

Собственно сам код ЦАП перед отправкой его в модуль рекомендуется откорректировать. 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 $12^{ти}$ битный ЦАП, связан с устанавливаемым на внешнем разъёме напряжением в соответствии со следующей таблицей

Таблица 3. Соответствие откорректированного кода ЦАП выходному напряжению

Код ЦАП	Напряжение, В
+2047	+5.0
0	0
–2048	–5.0

3.2.3. Логический номер канала

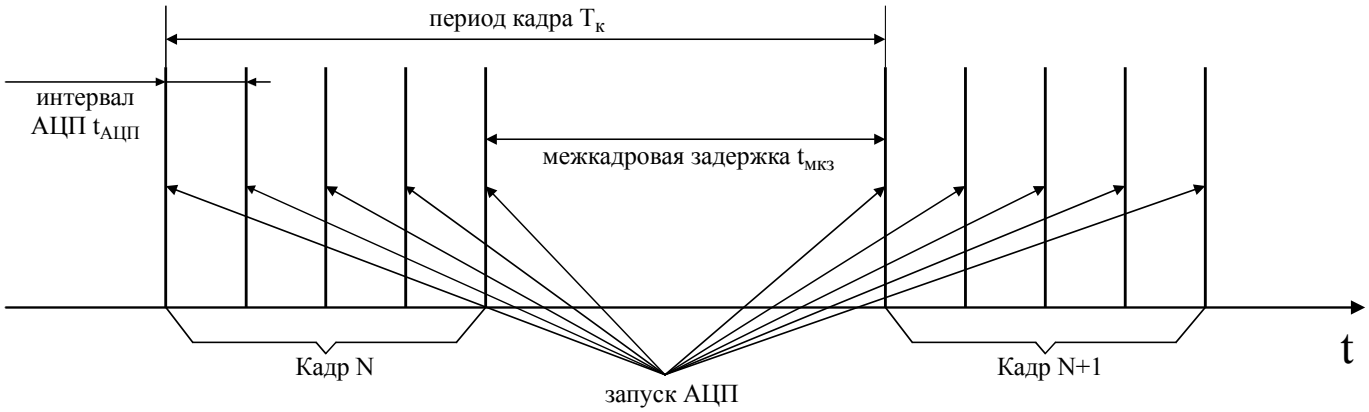
На модуле *E20-10* для управления работой входного аналогового каскада определяется такой параметр, как *$16^{ти}$ битный логический номер канала*. Именно массив логических номеров каналов, образующих управляющую таблицу **ControlTable**, задает циклическую последовательность работы АЦП при сборе данных. Для модуля *E20-10* логический номер канала содержит только собственно сам физический номер аналогового канала АЦП. Побитовый формат логического номер канала приведён в таблице ниже:

Таблица 4. Формат логического номера канала.

Битовые поля	Обозначение	Функциональное назначение
<0..1>	**CH<0..1>**	Номер канала АЦП: • ‘00’ – первый канал; • ‘01’ – второй канал. • ‘10’ – третий канал; • ‘11’ – четвёртый канал.
<2..15>	———	Зарезервировано

3.2.4. Формат кадра отсчётов

Под кадром подразумевается последовательность отсчётов с логических каналов, начиная от **ControlTable[0]** до **ControlTable[ControlTableLength–1]**, где **ControlTable** – управляющая таблица (массив логических каналов), а **ControlTableLength** определяет размер (длину) этой таблицы. Загрузить нужную управляющую таблицу в модуль можно с помощью интерфейсной функции ***SET_ADC_PARS()*** (см. *§ 4.5.4. "Установка параметров работы АЦП"*). Временные параметры работы модуля 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 отсчётом текущего кадра и первым отсчётом последующего, $\mathbf{t_{АЦП}}$ – 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 отсчётов в кадре, равен **ControlTableLength**, то все эти временные параметры можно связать следующей формулой:

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

или

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

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

4. Описание библиотеки Lusbapi

В настоящем разделе приведены достаточно подробные описания констант, структур и интерфейсных функций, входящих в состав штатной DLL библиотеки `Lusbapi` для модуля *E20-10*.

4.1. Общие принципы работы с модулем

Целью штатной DLL библиотеки `Lusbapi`, поставляемой с модулем *E20-10*, 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 *E20-10*.

Перед началом работы с библиотекой `Lusbapi` в пользовательской программе должны быть сделать следующие объявление (как минимум):

```
ILE2010 *pModule;                    // указатель на интерфейс модуля E20-10
MODULE_DESCRIPTION_E2010 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 библиотеки необходимо применять именно этот указатель (см. *пример ниже*).

После этого, используя уже полученный указатель на интерфейс модуля, следует проинициализировать доступ к соответствующему виртуальному слоту, к которому подключён модуль *E20-10*. Для этого предусмотрена интерфейсная функция *OpenLDevice()*. Если нет ошибки при выполнении этой функции, то можно быть уверенным, что устройство типа *E20-10* обнаружено в выбранном виртуальном слоте.

Теперь, в принципе, можно переходить к этапу загрузки обнаруженного модуля, но иногда нелишне бывает определиться с текущей скоростью работы используемого **USB** порта. Для этого предназначена интерфейсная функция *GetUsbSpeed()*. Для работы модуля на эффективных частотах сбора данных свыше 500 кГц необходимо, чтобы модуль совместно с **USB** портом работал в так называемом High-Speed Mode. Это будет соответствовать пропускной способности **USB** шины будет ~60 Мбайт/с. При этом максимальная пропускная способность самого модуля будет составлять ~20 Мбайт/с.

Важной особенностью модуля *E20-10* является то, что на нем установлена загружаемая ПЛИС. Т.е. для того, чтобы "*оживить*" модуль и заставить его работать по требуемому алгоритму, в ПЛИС необходимо предвариительно загрузить фирменную прошивку. Для этого можно воспользоваться интерфейсной функцией *LOAD_MODULE()*. В случае успешного выполнения данной функции, нужно проверить работоспособность загруженного *LBIOS* с помощью интерфейсной функции *MODULE_TEST()*. Если и эта функция выполнена без ошибки, то это означает, что модуль *E20-10* успешно загружен и полностью готов к дальнейшей работе.

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

В общем-то, предварительный этап работы с модулем *E20-10* 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 двуканальный ЦАП;
- однократная, и потому достаточно медленная, работа с входными и выходными цифровыми линиями;
- работа с пользовательским ППЗУ модуля и многое другое.

В качестве примера приведем исходный текст, а вернее сказать ‘*скелет*’, небольшой консольной программы для работы с модулем *E20-10*, предполагая использование Lusbapi версии не ниже 3.0:

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

ILE2010 *pModule;               // указатель на интерфейс модуля
MODULE_DESCRIPTION_E2010 md;    // структура с информацией о модуле
BYTE UsbSpeed;                  // скорость работы шины USB
char ModuleName[7];             // название модуля
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<ILE2010 *>(CreateLInstance("e2010"));
   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
   }
   // теперь отобразим полученную скорость работы шины USB
   printf(" USB is in %s\n", UsbSpeed ? "High-Speed Mode
                       (480 Mbit/s)" : "Full-Speed Mode (12 Mbit/s)");
   // 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
   }
   // на всякий случай проверим: этот модуль - 'E20-10'?
   if(strcmp(ModuleName, "E20-10"))
   {
      printf(" В нулевом виртуальном слоте не 'E20-10'\n");
      return 1;                 //выйдем из программы с ошибкой
   }
```

```
    // теперь можно попробовать загрузить из соответствующего ресурса
    // библиотеки Lusbapi фирменную прошивку в ПЛИС модуля
    if(!pModule->LOAD_MODULE())
    {
       printf("Не выполнена функция LOAD_MODULE()!");
       return 1;                  //выйдем из программы с ошибкой
    }

    // проверим работоспособность загруженного модуля
    if(!pModule->MODULE_TEST())
    {
       printf("Не выполнена функция MODULE_TEST()!");
       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 E20-10 (серийный номер %s) полностью готов к\
                                работе!", md.Module.SerialNumber);

    // далее можно располагать функции для непосредственного
    // управления модулем, например, по сбору данных с АЦП
    . . . . . .

    // 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 *E20-10*. Это весьма повышает *читаемость* и *понимаемость* исходных текстов, а также значительно облегчает сопровождение программ. Рассматриваемые константы расположены в файле `\DLL\Include\Lusbapi.h`.

1. Модулю для запуска процесса сбора данных необходимо наличие аппаратного *сигнала старта*. Данные константы определяют источник формирования этого сигнала. Местом использования этих констант, как правило, является поле *StartSource* структуры *ADC_PARS_E2010*.

Константа	Значение	Назначение
INT_ADC_START_E2010	0	*Сигнал старта* является внутренним и генерируется самим модулем. Этот импульс не транслируется на линию *DI16/START* внешнего разъёма ***DIGITAL I/O***.
INT_ADC_START_ WITH_TRANS_E2010	1	*Сигнал старта* является внутренним и генерируется самим модулем. При этом этот сигнал транслируется на линию *DI16/START* внешнего разъёма ***DIGITAL I/O***.
EXT_ADC_START_ON_ RISING_EDGE_E2010	2	Ожидается использование внешнего *сигнала старта*, который должен быть заведён на линию *DI16/START* внешнего разъёма ***DIGITAL I/O***. При этом сбор данных начинается по первому пришедшему фронту этого сигнала.
EXT_ADC_START_ON_ FALLING_EDGE_E2010	3	Ожидается использование внешнего *сигнала старта*, который должен быть заведён на линию *DI16/START* внешнего разъёма ***DIGITAL I/O***. При этом сбор данных начинается по первому пришедшему спаду этого сигнала.

2. Модулю для работы АЦП требуется наличие аппаратных *тактовых импульсов*. Данные константы определяют источник формирования этих импульсов. Местом использования этих констант, как правило, является поле *SynhroSource* структуры *ADC_PARS_E2010*.

Константа	Значение	Назначение
INT_ADC_CLOCK_E2010	0	*Тактовые импульсы* являются внутренними и генерируются самим модулем. При этом эти импульсы не транслируются на линию *SYNC* внешнего разъёма ***DIGITAL I/O***.
INT_ADC_CLOCK_ WITH_TRANS_E2010	1	*Тактовые импульсы* являются внутренними и генерируются самим модулем. При этом эти импульсы транслируются на линию *SYNC* внешнего разъёма ***DIGITAL I/O***.

EXT_ADC_CLOCK_ON_ RISING_EDGE_E2010	2	Ожидается использование внешних *тактовых импульсов*, которые должны быть заведены на линию *SYNC* внешнего разъёма ***DIGITAL I/O***. При этом АЦП работает по фронту этих импульсов.
EXT_ADC_CLOCK_ON_ FALLING_EDGE_E2010	3	Ожидается использование внешних *тактовых импульсов*, которые должны быть заведены на линию *SYNC* внешнего разъёма ***DIGITAL I/O***. При этом АЦП работает по спаду этих импульсов.

3. Модуль *E20-10 (Rev.'B' и выше)* позволяет дополнительно использовать аналоговую синхронизацию ввода данных. Константы, приведённые в таблице ниже, задают различные режимы этой синхронизации. Местом использования этих констант, как правило, является поле *SynchroAdMode* структуры *SYNCHRO_PARS_E2010*, которая является вложенной по отношению к структуре *ADC_PARS_E2010*.

Константа	Значение	Назначение
NO_ANALOG_SYNCHRO_E2010	0	Отсутствие аналоговой синхронизации.
ANALOG_SYNCHRO_ON_ RISING_CROSSING_E2010	1	Аналоговая синхронизация старта ввода данных по факту перехода сигнала '*снизу-вверх*' через заданный порог на выбранном канале.
ANALOG_SYNCHRO_ON_ FALLING_CROSSING_E2010	2	Аналоговая синхронизация старта ввода данных по факту перехода сигнала '*сверху-вниз*' через заданный порог на выбранном канале.
ANALOG_SYNCHRO_ON_ HIGH_LEVEL_E2010	3	Аналоговая синхронизация ввода данных только при условии нахождения сигнала *выше* заданного порога на выбранном канале.
ANALOG_SYNCHRO_ON_ LOW_LEVEL_E2010	4	Аналоговая синхронизация ввода данных только при условии нахождения сигнала *ниже* заданного порога на выбранном канале.

4. На входные каналы модуля *E20-10* можно подать напряжение, выходящее за пределы установленного диапазона. Это приводит к перегрузке каналов либо в ‘*плюс*’, либо в ‘*минус*’. Аппаратура модуля *E20-10 (Rev.'A')* может по-разному фиксировать факт перегрузки входных каналов при сборе данных с АЦП, что определяется нижеследующими константами. Модуль же *E20-10 (Rev.'B' и выше)* всегда работает в режиме ограничения перегрузки (**CLIPPING_OVERLOAD_E2010**). Местом использования этих констант, как правило, является поле *OverloadMode* структуры *ADC_PARS_E2010*.

Константа	Значение	Назначение
CLIPPING_OVERLOAD_E2010	0	При наличии перегрузки код с АЦП ограничивается значениями -8192 или 8191.
MARKER_OVERLOAD_E2010	1	При наличии перегрузки вместо кода АЦП аппаратура формирует маркеры *ADC_MINUS_OVERLOAD_MARKER* или *ADC_PLUS_OVERLOAD_MARKER*. Только для модулей *Rev. A*.

5. У входных каналов модуля *E20-10* есть три возможных диапазона входных напряжений. Каждый из диапазонов можно задать нижеследующими константами. Местом использования этих констант, как правило, является поле *InputRange* структуры *ADC_PARS_E2010*. Поле *InputRange* является массивом, каждый элемент которого задаёт определённый входной диапазон для соответствующего физического канала АЦП модуля.

Константа	Значение	Назначение
ADC_INPUT_RANGE_3000mV_E2010	0	При использовании в поле *InputRange* задаёт входной диапазон равный ±3000 мВ. Также можно использовать в качестве индекса для доступа к первому элементу константного массива *ADC_INPUT_RANGES_E2010*.
ADC_INPUT_RANGE_1000mV_E2010	1	При использовании в поле *InputRange* задаёт входной диапазон равный ±1000 мВ. Также можно использовать в качестве индекса для доступа ко второму элементу константного массива *ADC_INPUT_RANGES_E2010*.
ADC_INPUT_RANGE_300mV_E2010	2	При использовании в поле *InputRange* задаёт входной диапазон равный ±300 мВ. Также можно использовать в качестве индекса для доступа к третьему элементу константного массива *ADC_INPUT_RANGES_E2010*.

6. У модуля *E20-10* есть два возможных типа подключения входных каналов. Требуемый тип подключения задать нижеследующими константами. Местом использования этих констант, как правило, является поле *InputSwitch* структуры *ADC_PARS_E2010*. Поле *InputSwitch* является массивом, каждый элемент которого задаёт определённый тип подключения для соответствующего физического канала АЦП модуля.

Константа	Значение	Назначение
ADC_INPUT_ZERO_E2010	0	Данная константа соответствует заземлённому каналу АЦП модуля.
ADC_INPUT_SIGNAL_E2010	1	Данная константа задаёт подачу входного сигнала на вход АЦП модуля.

7. На модуле *E20-10* по желанию пользователя может быть установлена микросхема двухканального 12[ти] битного ЦАП. Состояние поля *Dac.Active* структуры служебной информации *MODULE_DESCRIPTION_E2010* отражает факт наличия ЦАП на борту модуля.

Константа	Значение	Назначение
DAC_INACCESSIBLED_E2010	0	На модуле полностью отсутствует микросхема ЦАП.
DAC_ACCESSIBLED_E2010	1	На модуле присутствует микросхема ЦАП.

8. Ревизия модуля *E20-10* отражает определённые конструктивные особенности модуля. Она задаётся одной заглавной латинской буквой и размещается в поле Revision вложенной структуры *MODULE_INFO_LUSBAPI* служебной структуры *MODULE_DESCRIPTION_E2010*. Например, *первая* ревизия модуля обозначается как литера ***'A'***.

Константа	Значение	Назначение
REVISION_A_E2010	0	Данная константа может использоваться в качестве индекса для доступа к первому элементу константного массива *REVISIONS_E2010*.
REVISION_B_E2010	1	Данная константа может использоваться в качестве индекса для доступа ко второму элементу константного массива *REVISIONS_E2010*.

9. Модуль *E20-10* позволяет отслеживать переполнение внутреннего буфера модуля, что приводит к нарушению целостности собираемых с АЦП данных. Эта информация отражается в бите с номером 0 или **BUFFER_OVERRUN_E2010** в поле `BufferOverrun` структуры *DATA_STATE_E2010*. Появление логического состояния '1' в этом бите указывает, что за время сбора данных призошло переполнение внутреннего буфера модуля.

10. Модуль *E20-10 (Rev.'B' и выше)* позволяет отслеживать глобальный (за всё время сбора) и локальные (за время одного запроса) битовые признаки переполнения разрядной сетки. Глобальный битовый признак активируется (переходит в состояние лог."1") при факте переполнения разрядной сетки у любого из 4^{x} физических каналов АЦП на всём промежутке времени от момента *START_ADC()* и вплоть до *STOP_ADC()*. Каждый их локальных битовых признаков активируется (переходит в состояние лог."1") при факте переполнения разрядной сетки у соответствующего физического канала АЦП за время одного запроса *ReadData()*. Каждый из этих признаков занимает соответствующий бит в поле `ChannelsOverFlow` структуры *DATA_STATE_E2010*. Все номера доступных битов приведены в таблице ниже:

Номер бита	Название константы	Назначение
0	**OVERFLOW_OF_CHANNEL_1_E2010**	Локальный признак переполнения разрядной сетки $1^{ого}$ физического канала АЦП.
1	**OVERFLOW_OF_CHANNEL_2_E2010**	Локальный признак переполнения разрядной сетки $2^{ого}$ физического канала АЦП.
2	**OVERFLOW_OF_CHANNEL_3_E2010**	Локальный признак переполнения разрядной сетки $3^{его}$ физического канала АЦП.
3	**OVERFLOW_OF_CHANNEL_4_E2010**	Локальный признак переполнения разрядной сетки $4^{ого}$ физического канала АЦП.
<4..6>	————————	Зарезервировано
7	**OVERFLOW_E2010**	Глобальный признак переполнения разрядной сетки.

11. Различные константы для работы с модулем *E20-10*.

Константа	Значение	Назначение
CURRENT_VERSION_ LUSBAPI	——	Версия используемой библиотеки `Lusbapi`. Как правило, используется совместно с функцией *GetDllVersion()*.
MAX_CONTROL_TABLE_ LENGTH_E2010	256	Максимально возможное кол-во логических каналов в управляющей таблице **ControlTable**.
ADC_CHANNELS_ QUANTITY_E2010	4	Кол-во физических каналов АЦП на модуле.
ADC_CALIBR_COEFS_ QUANTITY_E2010	12	Кол-во корректировочных коэффициентов для данных АЦП. По одному на каждый канал и на каждый входной диапазон. Корректировке подвергаются как смещение, так и масштаб данных АЦП.

DAC_CHANNELS_ QUANTITY_E2010	2	Кол-во физических каналов ЦАП на модуле (при условии наличия микросхемы ЦАП на модуле).
DAC_CALIBR_COEFS_ QUANTITY_E2010	2	Кол-во корректировочных коэффициентов для данных ЦАП. По одному на каждый канал. Корректировке подвергаются как смещение, так и масштаб данных ЦАП.
ADC_INPUT_RANGES_ QUANTITY_E2010	3	Кол-во входных диапазонов.
ADC_INPUT_TYPES_ QUANTITY_E2010	2	Кол-во типов подключений входных каналов.
TTL_LINES_QUANTITY_ E2010	16	Кол-во входных и выходных цифровых линий.
USER_FLASH_SIZE_E2010	512	Размер области пользовательского ППЗУ в байтах
REVISIONS_QUANTITY_ E2010	2	Кол-во ревизий (модификаций) модуля.
ADC_PLUS_OVERLOAD_ MARKER	0x5FFF	Маркер '*плюс*' перегрузки канала АЦП. Подразумевается *маркерный режим* фиксирования перегрузки каналов. Только для модуля *Rev.'A'*.
ADC_MINUS_OVERLOAD_ MARKER	0xA000	Маркер '*минус*' перегрузки канала АЦП. Подразумевается *маркерный режим* фиксирования перегрузки каналов. Только для модуля *Rev.'A'*.

12. Различные *константные массивы* для работы с модулем *E20-10*.

12.1. Массив доступных диапазонов входного напряжения АЦП в Вольтах:

```
const double
    ADC_INPUT_RANGES_E2010[ADC_INPUT_RANGES_QUANTITY_E2010] =
{
  3.0, 1.0, 0.3
};
```

12.2. Диапазон выходного напряжения ЦАП в Вольтах:

```
const double DAC_OUTPUT_RANGE_E2010 = 5.0;
```

12.3. 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_E2010*. Массив доступных ревизий модуля задаётся следующим образом:

```
const BYTE REVISIONS_E2010[REVISIONS_QUANTITY_E2010] =
{
      'A', 'B'
};
```

4.3. Структуры

В данном разделе приведены основные типы структур, которые применяются в библиотеке Lusbapi при работе с модулем *E20-10*.

4.3.1. Структура MODULE_DESCRIPTION_E2010

Структура *MODULE_DESCRIPTION_E2010* описана в файле \DLL\Include\Lusbapi.h и представлена ниже:

```
struct MODULE_DESCRIPTION_E2010
{
    MODULE_INFO_LUSBAPI      Module;        // общая информация о модуле
    INTERFACE_INFO_LUSBAPI   Interface;     // информация об интерфейсе
    MCU_INFO_LUSBAPI<MCU_VERSION_INFO_LUSBAPI> Mcu;  // информация о MCU
    PLD_INFO_LUSBAPI         Pld;           // информация о ПЛИС
    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 *E20-10*. 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_E2010

Структура *ADC_PARS_E2010* представляет собой группу параметров, задающих параметры сбора данных с АЦП. Данная структура описана в файле \DLL\Include\Lusbapi.h и представлена ниже:

```
struct ADC_PARS_E2010
{
    BOOL IsAdcCorrectionEnabled;  // управление автоматической корректировкой
                                  // на уровне ПЛИС модуля получаемых с АЦП
                                  // данных (для модуля Rev.'B' и выше)
    WORD OverloadMode;       //фиксация перегрузки входных каналов (для модуля Rev.'A')
    WORD InputCurrentControl;     // управление входным током смещения
                                  // (для модуля Rev.'B' и выше)
    SYNCHRO_PARS_E2010 SynchroPars;    // параметры синхронизации ввода
                                       // данных с АЦП
    WORD ChannelsQuantity;        // число активных каналов (размер кадра)
    WORD ControlTable[256];       // управляющая таблица с логическими каналами
    WORD InputRange[ADC_CHANNELS_QUANTITY_E2010];   //диапазон вх. напряжения
    WORD InputSwitch[ADC_CHANNELS_QUANTITY_E2010]; // тип подключения канала
    double AdcRate;               // частота работы АЦП в кГц
    double InterKadrDelay;        // межкадровая задержка в мс
    double KadrRate;              // частота кадра в кГц
    double AdcOffsetCoefs[ADC_INPUT_RANGES_QUANTITY_E2010]
                         [ADC_CHANNELS_QUANTITY_E2010];
                  // массив коэффициентов для корректировки смещение отсчётов АЦП:
                  // (3 диапазона)* (4 канала) (для модуля Rev.'B' и выше)
```

```
    double  AdcScaleCoefs[ADC_INPUT_RANGES_QUANTITY_E2010]
                          [ADC_CHANNELS_QUANTITY_E2010];
                    // массив коэффициентов для корректировки масштаба отсчётов АЦП:
                    // (3 диапазона)*(4 канала) (для модуля Rev.'B' и выше)
};
```

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 SYNCHRO_PARS_E2010

Структура `SYNCHRO_PARS_E2010` представляет собой группу параметров, используемых для задания разнообразных режимов синхронизации ввода данных с АЦП. Данная структура описана в файле `\DLL\Include\Lusbapi.h` и представлена ниже:

```
struct SYNCHRO_PARS_E2010
{
    WORD   StartSource;             // источник импульса начала сбора данных с АЦП
    DWORD  StartDelay;              // задержка старта сбора данных в кадрах отсчётов с
                                    // АЦП (для модуля Rev.'B' и выше)
    WORD   SynhroSource;            // источник тактовых импульсов запуска АЦП
    DWORD  StopAfterNKadrs;         // останов сбора данных после задаваемого здесь
                                    // кол-ва собранных кадров отсчётов АЦП (для
                                    // модуля Rev.'B' и выше)
    WORD   SynchroAdMode;           // режим аналоговой сихронизации: по переходу
                                    // или по уровеню (для модуля Rev.'B' и выше)
    WORD   SynchroAdChannel;        // физический канал АЦП для аналоговой
                                    // синхронизации (для модуля Rev.'B' и выше)
    SHORT  SynchroAdPorog;          // порог срабатывания при аналоговой
                                    // синхронизации (для модуля Rev.'B' и выше)
    BYTE   IsBlockDataMarkerEnabled;    // маркирование начала блока данных, что
                                    // весьма удобно, например, при аналоговой
                                    // синхронизации ввода данных по уровню
                                    // (для модуля Rev.'B' и выше)
};
```

Структура `SYNCHRO_PARS_E2010` входит составной частью в структуру *ADC_PARS_E2010*.

4.3.4. 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 назначения полей данной структуры.

4.3.5. Структура USER_FLASH_E2010

Структура *USER_FLASH_E2010* описана в файле `\DLL\Include\Lusbapi.h` и представлена ниже:

```
struct USER_FLASH_E2010
{
    BYTE Buffer[USER_FLASH_SIZE_E2010];        // размер данной структуры в байтах
};
```

Данная структура предназначена для хранения или считывания пользовательской информации. Для работы с ней выделена область размером *USER_FLASH_SIZE_E2010* байт в ППЗУ микроконтроллера. Используется в функциях *READ_FLASH_ARRAY()* и *WRITE_FLASH_ARRAY()*.

4.3.6. 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
    DWORD ErrorNumber;             // номер последней ошибки библиотеки Lusbapi
};
```

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

4.3.7. Структура DATA_STATE_E2010

Структура *DATA_STATE_E2010* описана в файле `\DLL\Include\Lusbapi.h` и представлена ниже:

```
struct DATA_STATE_E2010
{
    BYTE ChannelsOverFlow;     // битовые признаки перегрузки входных каналов
                               // для модуля Rev.'B' и выше
    BYTE BufferOverrun;        // битовые признаки переполнения внутреннего
                               // аппаратного буфера модуля
    DWORD CurBufferFilling;    // текущая заполненность внутреннего буфера
                               // модуля Rev.'B' и выше, в отсчётах
    DWORD MaxOfBufferFilling;  // за всё время сбора максимальная заполненность
                               // внутреннего буфера модуля Rev.'B' и выше, в отсчётах
    DWORD BufferSize;      // размер внутреннего буфера модуля Rev.'B' и выше, в отсчётах
    double CurBufferFillingPercent;     // текущая степень заполнения внутреннего
                                        // буфера модуля Rev.'B' и выше, в %
    double MaxOfBufferFillingPercent;  // за всё время сбора максимальная степень
                                // заполнения внутреннего буфера модуля Rev.'B' и выше, в %
};
```

Данная структура используется функцией *GET_DATA_STATE()* при опросе текущего состояния процесса сбора данных. В описании этой функции подробно прокомментированы смысл и назначения полей данной структуры.

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 *E20-10*. 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 – "E2010") .

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 интерфейса модуля *E20-10*, проинициализированного с помощью функции *CreateLInstance()*. Используется для аккуратного завершения сеанса работы с модулем (если предварительно удачно выполнилась функция *CreateLInstance()*). **!!!Внимание!!!** Данная функция должна обязательно вызываться в приложении перед его завершением во избежания утечки ресурсов *Windows*.

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 *E20-10* 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 *E20-10* находится в заданном виртуальном слоте. Если функция ***OpenLDevice()*** успешно выполнилась для заданного виртуального слота, то можно переходить непосредственно к загрузке модуля и его последующему управлению с помощью соответствующих интерфейсных функций 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 *E20-10*.

Возвращаемое значение: *TRUE* – модуль *E20-10* находится в выбранном виртуальном слоте и можно переходить непосредственно к загрузке модуля;
FALSE – устройства типа модуль *E20-10* 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 успешного выполнения данной функции всякий доступ к модулю *E20-10* становится невозможным. Для возобновления нормального доступа к устройству следует 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. Загрузка модуля

Формат:	**BOOL**	***LOAD_MODULE***(*PCHAR const FileName = NULL*)

Назначение:

Данная интерфейсная функция выполняет операцию загрузки прошивки (штатной или пользовательской) в ПЛИС модуля. Бинарный файл *FileName* с кодом прошивки должен находиться в текущей директории приложения. В штатной библиотеке есть возможность загружать ПЛИС фирменной прошивкой, хранящейся в самом теле библиотеки в виде соответствующего ресурса. Для этого достаточно параметр *FileName* задать в виде **NULL**. **NULL** является также значением по умолчанию для параметра *FileName*.

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

Передаваемые параметры:

- *FileName* – строка с названием бинарного файла прошивки ПЛИС модуля. Например, для файлов с фирменной прошивкой это строка `"E2010.pld"`.(для модуля *Rev.'A'*) или `"E2010m.pld"`.(для модуля *Rev.'B'* или выше). Если же данный параметр задан как **NULL** или вообще отсутствует, то загрузка модуля будет осуществляться той *прошивкой*, которая находится в виде ресурса в теле штатной библиотеки.

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

4.4.7. Проверка загрузки модуля

Формат:	**BOOL**	***TEST_MODULE***(*void*)	(версия 3.1 и ниже)
	BOOL	***TEST_MODULE***(*WORD TestModeMask = 0x0*)	(версия 3.2 и выше)

Назначение:

Для модуля *E20-10 (Rev.'A')* эта интерфейсная функция является всего лишь заглушкой и не несёт никакой функциональной нагрузки.

Для модуля *E20-10 (Rev.'B' и выше)* данная интерфейсная функция проверяет функциональное состояние ПЛИС модуля. Кроме того, данная функция позволяет переводить модуль в тестовый режим работы, который используется исключительно в наладочных целях.

Передаваемые параметры:

- *TestModeMask* – требуемый тестовый режим работы модуля. При нулевом параметре *TestModeMask* модуль переходит в штатный режим работы.

Возвращаемое значение: *TRUE* – ПЛИС загружен и функционирует надлежащим образом;
FALSE – произошла ошибка загрузки или функционирования ПЛИС модуля.

4.4.8. 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"*.

Передаваемые параметры:

- *ModuleName* – возвращается строка, не менее 6 символов, с названием модуля (в нашем случае это должна быть строка "E20-10").

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

4.4.9. 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.

4.4.10. Получение дескриптора модуля

Формат: **HANDLE** ***GetModuleHandle***(*void*)
Назначение: Данная функция позволяет получить дескриптор (handle) используемого модуля *E20-10*.
Передаваемые параметры: нет
Возвращаемое значение: В случае успеха – дескриптор модуля *E20-10*; в противном случае – **INVALID_HANDLE_VALUE**.

4.4.11. 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. Для некоторых функций, например *ReadData()*, при выявлении причины ошибки может потребоваться дополнительный вызов стандартной *Windows API* функции ***GetLastError()*** для классификации ошибок самой *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*

Аппаратура модуля *E20-10* и, соответственно, библиотека `Lusbapi` предназначены для организации непрерывного *потокового* сбора данных с АЦП на частотах вплоть до 10 МГц. При этом вполне осуществим даже многомодульный (по принципу '*ведущий-ведомый*') режим сбора данных при различных условиях синхронизации ввода.

Прежде чем запустить сбор данных с АЦП, необходимо передать в модуль требуемые параметры его работы: 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()*. Получаемые с АЦП данные модуль по мере необходимости будет передавать по шине **USB** в компьютер. Для реализации процедеру передачи данных АЦП из модуля в PC следует воспользоваться штатной функцией *ReadData()*. При этом функция *ReadData()* может выполняться как в *синхронном*, так и в *асинхронном* режимах. По окончании ввода последней порции собираемых данных, но не позднее чем через *400 мс*, настоятельно рекомендуется выполнять функцию завершения сбора *STOP_ADC()*. После чего с помощью функции *GET_DATA_STATE()* можно проверить состояние завершённого процесса сбора данных на предмет сбоев или ошибок сбора.

Перед работой модуля на эффективных частотах сбора свыше 500 кГц необходимо убедиться, что модуль *E20-10* взаимодействует с шиной **USB** в так называемом High-Speed Mode. Для этого рекомендуется использовать функцию *GetUsbSpeed()*.

Целый набор примеров по организации с помощью модуля *E20-10* непрерывного сбора данных для различных сред разработок можно найти на нашем CD-ROM'e в директории `\E20-10\Examples\`.

4.5.1. Корректировка данных АЦП

Схемотехника и использованные электронные компоненты обеспечивают линейность передаточной характеристики тракта АЦП модуля *E20-10*. Однако на модуле полностью отсутствуют какие-либо подстроечные резисторы. И хотя это и позволяет улучшить шумовые характеристики модуля и увеличить надёжность модуля, зато с неизбежностью приводит к тому, что входные показания АЦП могут иметь некоторое смешение нуля и неточность в передаче масштаба. Поэтому либо на уровне ПЛИС модуля, либо на уровне приложения необходимо организовывать корректировку получаемых с АЦП данных.

Для модуля *E20-10 (Rev.'A')* была предусмотрена корректировка полученных данных только на уровне приложения. А вот в модуле *E20-10 (Rev.'B' и выше)* заложена дополнительная возможность автоматической корректировке на уровне ПЛИС. Т.е. модуль сам производит корректировку получаемых данных АЦП. При этом в качестве корректировочных коэффициентов вполне можно использовать как *штатные (заводские)*, 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 *E14-140*. При этом вся ответственность за формирование и корректное применение *пользовательских* корректировочных коэффициентов полностью ложится на плечи конечного пользователя.

Штатные (заводские) корректировочные коэффициенты располагаются в полях *Adc.OffsetCalibration[]* и *Adc.ScaleCalibration[]* структуры служебной информации *MODULE_DESCRIPTION_E2010*. Вся служебная информация совместно с корректировочными коэффициентами записывается в модуль на этапе при его наладке в ООО "Л Кард". Поля коэффициентов представляют собой массивы типа *double*. Для модуля *E20-10* в каждом из этих массивов используются только первые *ADC_CALIBR_COEFS_QUANTITY_E2010* элементов. Массив *Adc.OffsetCalibration* содержит коэффициенты для корректировки смещение нуля, а массив *Adc.ScaleCalibration* – для корректировки масштаба. Если обозначить через *i* физический номер канала АЦП модуля, а через *j* – индекс входного диапазона этого канала, то корректировочные коэффициенты канала могут быть получены следующим образом:

- смещение: *Adc.OffsetCalibration[i + j*ADC_CHANNELS_QUANTITY_E2010]*;
- масштаб: *Adc.ScaleCalibration[i + j*ADC_CHANNELS_QUANTITY_E2010]*.

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

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

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

Например, со второго канала АЦП, настроенного на входной диапазон ±1.0 В (индекс диапазон равен 1 или *ADC_INPUT_RANGE_1000mV_E2010*), получены следующие данные: X1 = 1000, X2 = –1000 и X3 = 0. Тогда коэффициенты и скорректированные данные можно представить так:

- A = Adc.OffsetCalibration[1 + 1*ADC_CHANNELS_QUANTITY_E2010];
- B = Adc.ScaleCalibration[1 + 1*ADC_CHANNELS_QUANTITY_E2010];
- Y1 = (A+1000)*B, Y2 = (A-1000)*B, Y3 = A*B.

4.5.2. Запуск сбора данных АЦП

Формат: **BOOL** ***START_ADC***(*void*)
Назначение: Данная функция запускает модуль *E20-10* 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()*. Также функция ***START_ADC()*** сбрасывает в исходное нулевое состояние *признак целостности данных* модуля. Извлечение поступащих с модуля 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. Останов сбора данных АЦП

Формат: **BOOL** ***STOP_ADC***(*void*)
Назначение: Данная функция останавливает в модуле *E20-10* механизм сбора данных с АЦП. Попутно эта функция ‘*приводит в чувство*’ основную программу (*Firmware*) 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()*. Также весьма рекомендуется использование ***STOP_ADC()*** по окончании ввода последней порции собираемых данных, но не позднее определённого периода времени. Например, для частоты сбора 10 МГц этот период не должен быть более чем через *400 мс*. Это дает возможность с помощью функции *GET_DATA_STATE()* получать признак целостности *всех* последних собранных с АЦП данных.
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_E2010 * const AdcPars*)

Назначение:

Данная функция передает в *E20-10* 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_E2010*. Собственно, использование модулем именно этой переданной информации начинается только после выполнения интерфейсной функции *START_ADC()*. Также крайне не рекомендуется вызывать эту функцию собственно в процессе сбора данных. Её следует применять только после выполнения функции *STOP_ADC()*.

Формат структуры *ADC_PARS_E2010* приведен ранее в *§ 4.3.2. "Структура ADC_PARS_E2010"*, а смысл и назначение отдельных её полей достаточно подробно описано ниже.

- Поле AdcPars->IsAdcCorrectionEnabled. Запись. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. Для модуля *E20-10 (Rev.'B' и выше)* с помощью этого поля можно задавать выполнение автоматической корректировки (на уровне ПЛИС модуля) получаемых с АЦП данных. При использовании автоматической корректировки необходимо корректно заполнить соответствующими коэффициентами массивы `AdcOffsetCoefs[]` и `AdcOffsetCoefs[]` данной структуры. Для большинства случаев вполне можно использовать *штатные (заводские)* корректировочные коэффициенты, которые располагаются в полях *Adc.OffsetCalibration[]* и *Adc.ScaleCalibration[]* структуры служебной информации *MODULE_DESCRIPTION_E2010*.

- Поле AdcPars->OverloadMode. Запись. На входные каналы модуля *E20-10* можно подать напряжение, выходящее за пределы установленного диапазона. Это приводит к перегрузке каналов либо в ‘*плюс*’, либо в ‘*минус*’. Аппаратура модуля *E20-10 (Rev.'A')* может по-разному фиксировать факт перегрузки входных каналов при сборе данных с АЦП, что задаётся нижеследующими *константами перегрузки*. Модуль же *E20-10 (Rev.'B' и выше)* всегда работает в режиме ограничения перегрузки (**CLIPPING_OVERLOAD_E2010**). Смысловая нагрузка значений этого поля представлена в таблице ниже:

Значение	Константа	Описание
0	**CLIPPING_ OVERLOAD_E2010**	***Ограничение.*** При наличии перегрузки код отсчёта с АЦП ограничивается значениями -8192 или 8191.
1	**MARKER_ OVERLOAD_E2010**	***Маркеры.*** При наличии перегрузки аппаратура модуля подмешивает в код отсчёта с АЦП *признак* перегрузки. При этом фактически формируются маркеры *ADC_MINUS_OVERLOAD_MARKER* (при ‘*минус*’ перегрузке) или *ADC_PLUS_OVERLOAD_MARKER* (при ‘*плюс*’ перегрузке). Только для модуля *Rev. A*.

- Поле AdcPars->InputCurrentControl. Запись. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. Для модуля *E20-10 (Rev.'B' и выше)* с помощью этого поля можно управлять входным током смещения аналогового трактра модуля. Это поле может принимать значения 0 или 1. Подробнее о входном токе смещения смотри *"E20-10. Руководство пользователя. § 6.5.4. Подключение входа АЦП."*.

- Поле AdcPars->SynchroPars. Запись–Чтение. Это поле является вложенной структурой типа *SYNCHRO_PARS_E2010*, в которой сгруппированы все параметры, относящиеся к синхронизации ввода данных, а именно:
  - Поле AdcPars->SynchroPars.StartSource. Запись. Для запуска процесса сбора данных необходимо наличие аппаратного *сигнала старта*. Данное поле определяет источник формирования этого сигнала. Это поле может принимать одно из четырёх значений от 0 до 3, также можно пользоваться *константами сигнала старта*. Смысловая нагрузка значений этого поля представлена в таблице ниже:

Значение	Описание
0	***Внутренний источник сигнала старта без его трансляции на выход модуля.*** В этом режиме при вызове метода *START_ADC()* происходит автоматический запуск сбора данных с АЦП. При этом цифровая линия *DI16/START* внешнего разъёма ***DIGITAL I/O*** конфигурируется как *входная* по отношению к модулю и *сигнал старта* на нее не транслируется.
1	***Внутренний источник сигнала старта с его трансляцией на выход модуля.*** В этом режиме при вызове метода *START_ADC()* происходит автоматический запуск сбора данных с АЦП. При этом цифровая линия *DI16/START* внешнего разъёма ***DIGITAL I/O*** конфигурируется как *выходная* по отношению к модулю и при старте АЦП на ней будет сформировано событие в виде перехода линии *DI16/START* из состояния лог. ‘0’ в ‘1’ (*передний* фронт). Линия *DI16/START* будет находиться в состоянии лог. ‘1’ до тех пор, пока идёт сбор данных с АЦП и возвратится в исходное состояние лог. ‘0’, когда сбор данных программно остановлен с помощью функции *STOP_ADC()*.
2	***Внешний источник сигнала старта с активностью по переднему фронту.*** В этом режиме при вызове функции *START_ADC()* модуль переходит в режим ожидания события на цифровой линии *DI16/START* внешнего разъёма ***DIGITAL I/O***. При этом цифровая линия *DI16/START* конфигурируется как *входная* по отношению к модулю и при обнаружении на ней события в виде перехода линии *DI16/START* из состояния лог. ‘0’ в ‘1’ (*передний* фронт) происходит запуск сбора данных с АЦП.
3	***Внешний источник сигнала старта с активностью по заднему фронту.*** В этом режиме при вызове функции *START_ADC()* модуль переходит в режим ожидания события на цифровой линии *DI16/START* внешнего разъёма ***DIGITAL I/O***. При этом, цифровая линия *DI16/START* конфигурируется как *входная* по отношению к модулю и при обнаружении события в виде перехода линии *DI16/START* из состояния лог. ‘1’ в ‘0’ (*задний* фронт) происходит запуск сбора данных с АЦП.

  - Поле AdcPars->SynchroPars.StartDelay. Запись–Чтение. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. Для модуля *E20-10 (Rev.'B' и выше)* с помощью этого поля можно задавать задержку момента начала сбора данных, выраженную в *кадрах отсчётов АЦП*. Диапазон допустимых значений от 0 до 16 777 214. Т.о. после прихода аппаратного *сигнала старта* собственно сам сбор данных начнётся только после пропуска заданного кол-ва кадров отсчётов.

- Поле AdcPars->SynchroPars.SynhroSource. Запись. Для работы АЦП требуется наличие аппаратных *тактовых импульсов*. Данное поле определяет источник формирования этих импульсов. Это поле может принимать одно из четырёх значений от 0 до 3, также можно пользоваться *константами тактовых импульсов*. Следует помнить, что при выборе внешнего источника *тактовых импульсов*, их частота должна находиться строго в диапазоне от 1 МГц до 10 МГц. Смысловая нагрузка значений данного поля представлена в таблице ниже:

Значение	Описание
0	***Внутренний источник тактовых импульсов без их трансляции на выход модуля.*** В этом режиме при вызове метода *START_ADC()* происходит внутренняя генерация *тактовых импульсов* для АЦП модуля. При этом цифровая линия *SYNC* внешнего разъёма ***DIGITAL I/O*** конфигурируется как *входная* по отношению к модулю и *тактовые импульсы* на нее не транслируется.
1	***Внутренний источник тактовых импульсов с их трансляцией на выход модуля.*** В этом режиме при вызове метода *START_ADC()* происходит внутренняя генерация *тактовых импульсов* для АЦП модуля. При этом цифровая линия *SYNC* внешнего разъёма ***DIGITAL I/O*** конфигурируется как *выходная* по отношению к модулю и на нее происходит трансляция генерируемых *тактовых импульсов*.
2	***Внешний источник тактовых импульсов с активностью по переднему фронту.*** В этом режиме при вызове функции *START_ADC()* происходит использование внешнего источника *тактовых импульсов*, подключенных к цифровой линии *SYNC* внешнего разъёма ***DIGITAL I/O***. При этом линия *SYNC* конфигурируется как *входная* по отношению к модулю и событие в виде перехода сигнала на этой линии из состояния лог. ‘0’ в ‘1’ (*передний* фронт) трактуется как внешний синхросигнал для работы АЦП.
3	***Внешний источник тактовых импульсов с активностью по заднему фронту.*** В этом режиме при вызове функции *START_ADC()* происходит использование внешнего источника *тактовых импульсов*, подключенных к цифровой линии *SYNC* внешнего разъёма ***DIGITAL I/O***. При этом линия *SYNC* конфигурируется как *входная* по отношению к модулю и событие в виде переход сигнала на этой линии из состояния лог. ‘1’ в ‘0’ (*задний* фронт) трактуется как внешний синхросигнал для работы АЦП.

- Поле AdcPars->SynchroPars.StopAfterNKadrs. Запись–Чтение. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. Для модуля *E20-10 (Rev.'B' и выше)* с помощью этого поля можно организовать останов сбора данных после задаваемого здесь кол-ва *собранных кадров отсчётов*. Диапазон допустимых значений от 0 до 16 777 215. Если поле StopAfterNKadrs приравнять 0, то этот параметр будет полностью проигнорирован модулем при сборе данных. Особо следует упомянуть о продвинутых возможностях работы модуля при StopAfterNKadrs ≠ 0 и определенных условиях синхронизации: по внешнему сигналу *старта* (поле StartSource равно 2 или 3) и/или при аналоговой синхронизации по *переходу* (поле SynchroAdMode равно 1 или 2). При этом модуль будет осуществлять сбор данных блоком по StopAfterNKadrs кадров на каждое выполнение условий синхронизации.

Например, если задать StopAfterNKadrs=1024 и StartSource=0x2, то после выполнения *START_ADC()* модуль перейдёт в режим ожидания активного перепада на линии внешнего синхроимпульса. При обнаружении оного, модуль осуществит сбор блока данных размером 1024 кадров, после чего *автоматически* вернется к ожиданию очередного активного синхроимпульса. Этот процесс будет продолжаться циклически вплоть до выполнения функции *STOP_ADC()*. В принципе модуль может особым образом промаркировать каждый получаемый блок данных, если дополнительно поле IsBlockDataMarkerEnabled установить равным 1.

- Поле AdcPars->SynchroPars.SynchroAdMode. Запись. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. Для модуля *E20-10 (Rev.'B' и выше)* с помощью этого поля можно задавать различные режимы аналоговой сихронизации ввода данных. Это поле может принимать одно из пяти значений от 0 до 4, а также можно пользоваться константами *режимов аналоговой синхронизации*. Смысловая нагрузка значений данного поля представлена в таблице ниже:

Константа	Значение	Назначение
NO_ANALOG_SYNCHRO_E2010	0	Отсутствие аналоговой синхронизации.
ANALOG_SYNCHRO_ON_ RISING_CROSSING_E2010	1	Аналоговая синхронизация старта ввода данных по факту перехода сигнала '*снизу-вверх*' через заданный порог на выбранном канале.
ANALOG_SYNCHRO_ON_ FALLING_CROSSING_E2010	2	Аналоговая синхронизация старта ввода данных по факту перехода сигнала '*сверху-вниз*' через заданный порог на выбранном канале.
ANALOG_SYNCHRO_ON_ HIGH_LEVEL_E2010	3	Аналоговая синхронизация ввода данных только при условии нахождения сигнала *выше* заданного порога на выбранном канале.
ANALOG_SYNCHRO_ON_ LOW_LEVEL_E2010	4	Аналоговая синхронизация ввода данных только при условии нахождения сигнала *ниже* заданного порога на выбранном канале.

- Поле AdcPars->SynchroPars.SynchroAdChannel. Запись. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. Для модуля *E20-10 (Rev.'B' и выше)* с помощью этого поля можно задавать физический канал АЦП для выбранной *аналоговой синхронизации*. Это поле может принимать одно из четырёх значений от 0 до 3.
- Поле AdcPars->SynchroPars.SynchroAdPorog. Запись. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. Для модуля *E20-10 (Rev.'B' и выше)* с помощью этого поля можно задавать порог срабатывания *аналоговой синхронизации*. Порог задаётся в кодах АЦП в диапазоне от -8192 до 8191.

- Поле AdcPars->SynchroPars.IsBlockDataMarkerEnabled. Запись. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. При установке поля IsBlockDataMarkerEnabled равным 1, аппаратура модуля *E20-10 (Rev.'B' и выше)* в первый отсчёт каждого *непрерывного* (по времени) блока данных вставляет искусственный логический признак (маркер), который кодируется установкой значения "01" в поле разрядов <15..14> отсчёта данных. Такой маркер позволяет на верхнем программном уровне отличать начало одного непрерывного участка данных от другого. Наличие такого маркера может быть особенно полезно при, например, вводе данных с использованием аналоговой синхронизации по уровню.

- Поле AdcPars->ChannelsQuantity. Запись–Чтение. Данное поле задаёт количество активных *логических каналов* в управляющей таблице **ControlTable**. Т.е. при сборе данных с АЦП будут использоваться первые AdcPars->ChannelsQuantity элементов массива AdcPars->ControlTable. Предельным значением для данного параметра является величина 256 или *MAX_CONTROL_TABLE_LENGTH_E2010*. Если переданная в функцию величина AdcPars->ChannelsQuantity превышает указанное предельное значение, то функция автоматически выполняет необходимую корректировку. А по завершении функции в поле AdcPars->ChannelsQuantity будет находиться реально установленное количество активных *логических каналов*.

- Поле AdcPars->ControlTable[]. Запись. Данное поле задаёт управляющую таблицу **ControlTable**. Т.е. тот самый массив *логических каналов*, который будет использоваться модулем при работе с АЦП для задания циклической последовательности отсчётов с входных каналов.

- Поле AdcPars->InputRange[]. Запись. Данное поле задаёт входные диапазоны сразу для всех физических каналов АЦП модуля *E20-10*. Поле является массивом типа *WORD*, состоящим из 4^x (*ADC_CHANNELS_QUANTITY_E2010*) элементов. Элемент массива с индексом 0 соответствует входному диапазону первого канала АЦП и т.д. Каждый элемент массива может принимать значение от 0 до 2. Также можно использовать *константы входного диапазона*. Смысловая нагрузка значений этого поля представлена в таблице ниже:

Значение элемента массива	Константа	Описание
0	**ADC_INPUT_RANGE_ 3000mV_E2010**	Входной диапазон равен ±3000 мВ
1	**ADC_INPUT_RANGE_ 1000mV_E2010**	Входной диапазон равен ±1000 мВ
2	**ADC_INPUT_RANGE_ 300mV_E2010**	Входной диапазон равен ±300 мВ

- Поле AdcPars->InputSwitch[]. Запись. Данное поле задаёт тип подключений или режим входной коммутации сразу для всех физических кананов АЦП модуля *E20-10*. Поле является массивом типа *WORD*, состоящим из 4^x (*ADC_CHANNELS_QUANTITY_E2010*) элементов. Элемент массива с индексом 0 соответствует типу подключения первого канала АЦП и т.д. Каждый элемент массива может принимать значение от 0 до 1. Также можно использовать *константы типа подключения*. Смысловая нагрузка значений этого поля представлена в таблице ниже:

Значение элемента массива	Константа	Описание
0	**ADC_INPUT_ZERO_E2010**	Вход канала АЦП скоммутирован на аналоговую землю модуля.
1	**ADC_INPUT_SIGNAL_E2010**	На вход канала АЦП подаётся входной сигнал.

- Поля AdcPars->AdcRate и AdcPars->InterKadrDelay. Запись–Чтение. Данные поля имеют силу только при использовании модулем внутренних *тактовых импульсов*, что определяется полем *AdcPars->SynchroPars.SynhroSource*. При этом 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 возвращаются 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, частота работы АЦП определяется по следующей формуле: **AdcRate** = 30000/N, где **N** – целое число от 3 до 30. Поэтому данная функция просто вычисляет ближайшую к задаваемой дискретную величину **AdcRate**, передает ее в модуль в виде целого числа **N**, а также возвращает ее значение в поле AdcPars->AdcRate. Всё тоже самое верно и для межкадровой задержки, с той лишь разницей, что она задается в единицах 1/**AdcRate**, причем уже предварительно откорректированной **AdcRate**. **InterKadrDelay** может быть в диапазоне от 1/**AdcRate** до 255/**AdcRate** (для модуля *E20-10 (Rev.'A')*) или 65535/**AdcRate** (для модуля *E20-10 (Rev.'B' и выше)*). Например, если задать AdcPars->AdcRate = 0.0, то ***SET_ADC_PARS()*** установит и возвратит минимально возможную величину для данной переменной, т.е. 1000.0 *кГц*. Аналогично: если задать AdcPars->InterKadrDelay = 0.0, то данная 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*. Данное поле имеет силу только при использовании модулем внутренних *тактовых импульсов*, что определяется полем *AdcPars->SynchroPars.SynhroSource*. Эта частота 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"*.
- Поле AdcPars->AdcOffsetCoefs[][]. Запись. Поле является двумерным массивом типа *double*, состоящим из *ADC_INPUT_RANGES_QUANTITY_E2010* **x** *ADC_CHANNELS_QUANTITY_E2010* элементов. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. Для модуля *E20-10 (Rev.'B' и выше)* в этом массиве должны располагаться коэффициенты, используемые ПЛИС модуля для выполнения автоматической корректировки смещения получаемых с АЦП даных. Разрешение использовать автоматическую корректировку данных задаётся с помощью поля AdcPars->CorrectionEnabled. Подробнее о корректировке данных смотри *§ 4.5.1. "Корректировка данных АЦП"*.

- Поле AdcPars->AdcScaleCoefs[][]. Запись. Поле является двумерным массивом типа *double*, состоящим из *ADC_INPUT_RANGES_QUANTITY_E2010* **x** *ADC_CHANNELS_QUANTITY_E2010* элементов. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. Для модуля *E20-10 (Rev.'B' и выше)* в этом массиве должны располагаться коэффициенты, используемые ПЛИС модуля для выполнения автоматической корректировки масштаба получаемых с АЦП данных. Разрешение использовать автоматическую корректировку данных задаётся с помощью поля AdcPars->CorrectionEnabled. Подробнее о корректировке данных смотри *§ 4.5.1. "Корректировка данных АЦП"*.

Передаваемые параметры:
- *AdcPars* – адрес структуры типа *ADC_PARS_E2010* с требуемыми параметрами функционирования сбора данных с 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_E2010 * const AdcPars*)

Назначение:
Данная функция считывает из модуля *E20-10* всю текущую информацию, которая используется при сборе данных с АЦП.

Передаваемые параметры:
- *AdcPars* – адрес структуры типа *ADC_PARS_E2010* с полученными из модуля текущими параметрами функционирования АЦП.

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

4.5.6. Получение данных АЦП

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

Назначение:
Данная функция предназначена для получения очередной порции данных с АЦП модуля. 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 *E20-10*. 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 покадрово: 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. В зависимости от ревизии модуля этот параметр имеет следующие ограничения: • для модуля *E20-10 (Rev.'A')* значение NumberOfWords-ToPass должно находиться в диапазоне от 256 до (1024*1024), а также быть кратным 256; • для модуля *E20-10 (Rev.'B' и выше)* значение NumberOfWordsToPass должно находиться в диапазоне от 1 до (1024*1024). 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.
NumberOf-WordsPassed	*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. Оно задаёт максимальное время в *мс* на ожидание завершения *синхронного* запроса на сбор требуемого количества данных. Если по истечении этого времени *все* затребованные запросом данные недополучены, функция завершается и возвращает ошибку.

На модуле *E20-10* устанавливается внутренний аппаратный *FIFO* буфер данных размером 8 Мбайт. Такой большой буфер необходим уверенного сбора данных на больших частотах ввода. Так при частотах сбора порядка 10 МГц переполнение буфера произойдёт только через 400 *мс*, что является достаточно большим 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 выполнения функции *START_ADC()*, которой, в принципе, должен предшествовать вызов 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 ожидание завершения сбора данных и, следовательно, весьма длительное время не возвращать управление приложению. Пример корректного использования штатных функций библиотеки `Lusbapi` в *синхронном* режиме в виде обычного консольного приложения можно найти на нашем фирменном CD-ROM в директории `\E20-10\Examples\Borland C++ 5.02\ReadDataSynchro`.
2. *Асинхронный* режим. Этот режим функционально намного гибче, чем *синхронный* режим и его рекомендуется использовать при организации различных алгоритмов непрерывного *потокового* сбора данных, когда количество вводимых отсчётов превышает 1 МСлов. Данный режим, например, 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()*. 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 10 МГц после *START_ADC()*, но перед началом выполнения функции получения данных ***ReadData()*** система '*задумалась*' более чем на *400 мс* (что бывает очень редко, но вполне возможно), то сбой в принимаемых данных практически обеспечен. Этот сбой в данных проявляется в виде нарушения целостности данных и его можно отследить с помощью функции *GET_DATA_STATE()*. 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 существо-

вать 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 *E20-10*.

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

4.5.7. Проверка состояния процесса сбора данных

Формат: **BOOL** ***CHECK_DATA_INTERGRITY***(*BYTE * const DataIntegrity*) (версия 3.1 и ниже)
BOOL ***GET_DATA_STATE***(*DATA_STATE_E2010 * const DataState*) (версия 3.2 и выше)

Назначение:

Данная функция позволяет получать в структуре типа *DATA_STATE_E2010* текущее состояние процесса сбора данных. Формат структуры *DATA_STATE_E2010* приведен ранее в *§ 4.3.7. "Структура DATA_STATE_E2010"*, а назначение отдельных ее полей достаточно подробно описано ниже.

- Поле DataState->ChannelsOverFlow. Чтение. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. Для модуля *E20-10 (Rev.'B' и выше)* с помощью этого поля можно получить глобальный (за всё время сбора) и локальные (за время одного запроса) битовые признаки переполнения разрядной сетки. Глобальный битовый признак активируется (переходит в состояние лог."1") при факте переполнения разрядной сетки у любого из 4^{x} физических каналов АЦП на всём промежутке времени от момента *START_ADC()* и вплоть до *STOP_ADC()*. Каждый их локальных битовых признаков активируется (переходит в состояние лог."1") при факте переполнения разрядной сетки у соответствующего физического канала АЦП за время одного запроса *ReadData()*. В качестве номеров используемых битов можно использовать *константами номеров битов перегрузки каналов*. Смысловая нагрузка битов этого поля представлена в таблице ниже:

Номер бита	Название константы	Назначение
0	**OVERFLOW_OF_CHANNEL_1_E2010**	Локальный признак переполнения разрядной сетки $1^{ого}$ физического канала АЦП.
1	**OVERFLOW_OF_CHANNEL_2_E2010**	Локальный признак переполнения разрядной сетки $2^{ого}$ физического канала АЦП.
2	**OVERFLOW_OF_CHANNEL_3_E2010**	Локальный признак переполнения разрядной сетки $3^{его}$ физического канала АЦП.

3	**OVERFLOW_OF_CHANNEL_4_E2010**	Локальный признак переполнения разрядной сетки 4ого физического канала АЦП.
<4..6>	————————	Зарезервировано
7	**OVERFLOW_E2010**	Глобальный признак переполнения разрядной сетки.

- Поле DataState->BufferOverrun. Чтение. *E20-10* позволяет отслеживать глобальный признак переполнение внутреннего аппаратного буфера модуля. Причём модуль отслеживает этот признак на всём промежутке времени от момента *START_ADC()* и вплоть до *STOP_ADC()*. Эта информация отражается в бите с номером 0 или **BUFFER_OVERRUN_E2010** данного поля структуры. Появление логического состояния '1' в этом бите указывает, что за время сбора данных призошло переполнение внутренного буфера модуля. При этом версия *основной программы MCU* модуля должна быть 1.7 или выше. Кроме того, если в процессе сбора с АЦП модуль обнаруживает переполнение буфера, то он визуально информирует об этой ситуации путём мигания своего светодиодного индикатора красным (для модуля *E20-10 (Rev.'A')*) или красно-зелёным (для модуля *E20-10 (Rev.'B' и выше)*) цветом.
- Поле DataState->CurBufferFilling. Чтение. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. Для модуля *E20-10 (Rev.'B' и выше)* данное поле содержит текущую заполненность внутреннего аппаратного буфера модуля. Выражается в отсчётах.
- Поле DataState->MaxOfBufferFilling. Чтение. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. Для модуля *E20-10 (Rev.'B' и выше)* данное поле показывает, какая максимальная заполненность внутреннего аппаратного буфера модуля была достигнута на всём промежутке сбора данных от момента *START_ADC()* и вплоть до *STOP_ADC()*. Выражается в отсчётах.
- Поле DataState->BufferSize. Чтение. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. Для модуля *E20-10 (Rev.'B' и выше)* данное поле содержит полный размер внутреннего аппаратного буфера модуля. Выражается в отсчётах.
- Поле DataState->CurBufferFillingPercent. Чтение. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. Для модуля *E20-10 (Rev.'B' и выше)* данное поле показывает процентный уровень текущей заполненности внутреннего аппаратного буфера. Выражается в %.
- Поле DataState->MaxOfBufferFillingPercent. Чтение. Для модуля *E20-10 (Rev.'A')* это поле не несёт никакой функциональной нагрузки. Для модуля *E20-10 (Rev.'B' и выше)* данное поле показывает какой максимальный процентный уровень заполненности внутреннего аппаратного буфера был достигнут на всём промежутке сбора данных от момента *START_ADC()* и вплоть до *STOP_ADC()*. Выражается в %.

Передаваемые параметры:

- *DataState* – возвращаемая структура, с текущим состоянием процесса сбора данных.

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

4.6. *Функции для работы с ЦАП*

Аппаратура модуля *E20-10* и, соответственно, библиотека `Lusbapi` позволяет управлять выводом данных 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 `\E20-10\Examples\Borland C++ 5.02\DacSample`.

4.6.1. Корректировка данных ЦАП

Схемотехника и использованные компоненты обеспечивают линейность передаточной характеристики ЦАП тракта модуля *E20-10*. 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 ООО “Λ Кард”. Благодаря этому на модуле отсутствуют подстроечные резисторы, что улучшает шумовые характеристики модуля и увеличивает их надежность.

Сами коэффициенты располагаются в полях *Dac.OffsetCalibration[]* и *Dac.ScaleCalibration[]* структуры служебной информации *MODULE_DESCRIPTION_E2010*. Эти поля представляют из себя массивы типа *double*. Для модуля *E20-10* в каждом из этих массивов используются только первые *DAC_CALIBR_COEFS_QUANTITY_E2010* элементов. Массив *Dac.OffsetCalibration* содержит коэффициенты для корректировки смещение нуля первого и второго 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 втором канале ЦАП необходимо выставить напряжения, соответствующие следующим кодам ЦАП: X1 = 1000, X2 = -1000, X3 = 0. Тогда коэффициенты коррекции и данные для второго канала ЦАП можно представить так: A = Dac.OffsetCalibration[1], B = Dac.ScaleCalibration[1], 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: Данная функция позволяет однократно устанавливать на указанном канале ЦАП *DacChannel* напряжение в соответствии со значением *DacData* (в кодах ЦАП). Функция *DAC_SAMPLE()* выполняется достаточно *медленно* и, используя её, можно достичь частоты вывода данных на ЦАП порядка нескольких сотен *Гц*. О соответствии кода ЦАП величине устанавливаемого на выходе модуля аналогового напряжения см. *§ 3.2.2. "Формат слова данных для ЦАП"*
Передаваемые параметры: • *DacData* – устанавливаемое значение напряжения в кодах ЦАП (от –2048 до 2047). • *DacChannel* – требуемый номер канала ЦАП (0 или 1).
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 *E20-10* располагаются на внешнем разъёме ***DIGITAL I/O***. Если на этом разъёме специальная линия *EN_OE* никак не задействована (см. *Руководство пользователя*), то, по умолчанию, непосредственно после подачи на модуль внешнего напряжения питания выходные цифровые линии находятся в *высокоимпедансном* состоянии. Если же эта линия *EN_OE* надлежащим образом задействована, то после подачи питания *все* выходные линии становятся активными.

Аппаратура модуля *E20-10* 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. Если же надлежащим образом задействована специальная линия *EN_OE* разъёма ***DIGITAL I/O*** (см. *Руководство пользователя*), то данная функция не оказывает никакого влияния на работу модуля.
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* $16^{ти}$ 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*.
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* $16^{ти}$ выходных цифровых линий на внешнем цифровом разъёме ***DIGITAL I/O*** модуля *E20-10* 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*.
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 *E20-10* часть памяти микроконтроллера отведена под пользовательское ППЗУ. Размер этой области составляет *USER_FLASH_SIZE_E2010* байт. Всю эту область пользователь может смело использовать в своих сугубо частнособственнических интересах.

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.
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 данных в ППЗУ

Формат: **BOOL** ***WRITE_FLASH_ARRAY***(*USER_FLASH_E2010 * const UserFlash*)
Назначение: Данная интерфейсная функция выполняет запись массива байт размером *USER_FLASH_SIZE_E2010* в ППЗУ. Т.о. перезаписывается сразу *вся* доступная область пользовательского ППЗУ. Перед началом процедуры записи в пользовательское ППЗУ необходимо разрешить эту операцию с помощью интерфейсной функции *ENABLE_FLASH_WRITE()*. После окончания процедуры записи всей требуемой информации необходимо запретить режим записи с помощью той же функции *ENABLE_FLASH_WRITE()*.
Передаваемые параметры: • *UserFlash* – фактически это байтовый массив, который должен быть записан в пользовательское ППЗУ.
Возвращаемое значение: *TRUE* – функция успешно выполнена; *FALSE* – функция выполнена с ошибкой.

4.8.3. Чтение данных из ППЗУ

Формат: **BOOL** ***READ_FLASH_ARRAY***(*USER_FLASH_E2010 * const UserFlash*)
Назначение: Данная интерфейсная функция вычитывает содержимое сразу всей области пользовательского ППЗУ.
Передаваемые параметры: • *UserFlash* – в этом байтовом массиве возвращается образ всего пользовательского ППЗУ.
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 *E20-10*: название модуля, его серийный номер и ревизию, корректировочные коэффициенты для АЦП и ЦАП, версии используемых прошивок ПЛИС и MCU, тактовые частоты работы исполнительных устройств (ПЛИС, 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_E2010 * const ModuleDescription*)

Назначение:

Данная интерфейсная функция осуществляет чтение всей служебной информации в структуру типа *MODULE_DESCRIPTION_E2010*. 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 загрузки ПЛИС модуля и проверки его работоспособности (см. *§ 4.1. "Общие принципы работы с модулем"*)

Передаваемые параметры:

- *ModuleDescription* – указатель на структуру типа *MODULE_DESCRIPTION_E2010*, 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_E2010 * const ModuleDescription*)

Назначение:

Данная интерфейсная функция позволяет сохранять в модуле всю служебную информацию из структуры типа *MODULE_DESCRIPTION_E2010*. **!!!Внимание!!!** Применять данную функцию нужно только в случае крайней необходимости. Например, когда по тем или иным обстоятельствам испортилось содержимое служебной информации.

Передаваемые параметры:

- *ModuleDescription* – указатель на структуру типа *MODULE_DESCRIPTION_E2010*, 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 MCU_VERSION_INFO_LUSBAPI

Вспомогательная структура *MCU_VERSION_INFO_LUSBAPI* состоит из двух структур *VERSION_INFO_LUSBAPI* и содержит информацию о программном обеспечении исполнительного устройства, которая включает в себя информацию о прошивках как основной программы (*Firmware*), так и загрузчика (*Bootloader*). Данная структура описывается следующим образом:

```
struct MCU_VERSION_INFO_LUSBAPI
{
    VERSION_INFO_LUSBAPI FwVersion;     // версия основной программы (Firmware)
    VERSION_INFO_LUSBAPI BlVersion;     // версия загрузчика (BootLoader)
};
```

A.4. 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 Modification;                           // исполнение (вариант) модуля;
    BYTE Comment[COMMENT_LINE_LENGTH_LUSBAPI];    // строка комментария
};
```

A.5. 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.6. 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:

```
template <class VersionType>
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 Firmware и BootLoader
    BYTE   Comment[COMMENT_LINE_LENGTH_LUSBAPI];    // строка комментария
};
```

A.7. Структура PLD_INFO_LUSBAPI

Вспомогательная структура *PLD_INFO_LUSBAPI* содержит самую общую информацию об используемом исполнительном устройстве типа программируемая логическая интегральная схема (ПЛИС). Данная структура описывается следующим образом:

```
struct PLD_INFO_LUSBAPI
{
    BOOL   Active;                  // флаг достоверности остальных полей структуры
    BYTE   Name[NAME_LINE_LENGTH_LUSBAPI];       // название ПЛИС
    double ClockRate;                      // тактовая частота работы ПЛИС в кГц
```

```
    VERSION_INFO_LUSBAPI Version;          // информация о версии прошивке ПЛИС
    BYTE   Comment[COMMENT_LINE_LENGTH_LUSBAPI];     // строка комментария
};
```

A.8. 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.9. 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.10. 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
};
```