ООО «ADC*lab*»

Комплект средств разработки

для создания приложений сбора и обработки данных от устройств EL200/EL100

РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ ФУНКЦИЙ

Москва 2010

Содержание

1 Введение 6
2 Описание структур данных 8
ADCParametersBLOCK 8
ADCParametersSTREAM 13
3 Описание функций 18
Setup 18
Release 20
Init 21
Start 22
Stop 23
GetData 24
PortIO 27
Get 29
4 Использование alf -файлов 31
4.1. Использование alf -файлов 32
4.2. Описание содержимого alf – файла 39
5 Приложение 41
5.1. Пример использования устройств EL200E, EL200SD 41
5.2. Пример использования устройств EL200E, EL200SD с организацией отдельного потока для обработки данных 46

В этом руководстве рассматриваются вопросы создания приложений для среды Microsoft Windows, взаимодействующих с устройствами сбора аналоговых сигналов EL100/EL200. В качестве среды разработки используется Microsoft Visual C++. Описываемые функции реализованы в составе драйвера, поставляемого вместе с устройствами сбора. Описание установки драйвера приведено в руководстве по устройствам EL100/200 и здесь не рассматривается. Для использования описываемых в данном документе функций необходимо, чтобы вышеупомянутый драйвер был корректно установлен. Перед началом разработки программ следует ознакомиться с описанием устройств EL100/EL200, которое находится на прилагаемом к устройству компакт-диске.

Устройство EL100 содержит одноканальный 12-разрядный АЦП, с помощью которого производится последовательная оцифровка 32 напряжений в диапазоне от -10 до +10 вольт. Кроме того EL100, содержит блок разовых команд (8 на вход и 8 на выход). Так же в состав устройства входит 2-канальный блок ЦАП.

Устройство EL200 содержит 2-х канальный 16-разрядный АЦП, с помощью которого производится последовательная оцифровка 16-ти пар напряжений в диапазоне от -10 до +10 вольт, при этом напряжения в каждой паре считываются синхронно. Возможна работа в дифференциальном режиме. Кроме того EL200, содержит блок разовых команд (8 на вход и 8 на выход).

Также оба устройства обеспечивают автономный сбор информации на флэш - карту SD.

ОРГАНИЗАЦИЯ ДОКУМЕНТА

1. ***Введение***
   Изложены фундаментальные основы построения приложений в среде Microsoft Visual C++ с учетом особенностей использования функций драйвера.

2. ***Описание структур данных***
   Приведены форматы структур данных, используемых при вызове функций драйвера, и описаны элементы этих структур.

3. ***Описание функций***
   Показаны правила обращения к функциям драйвера с описанием входных и выходных параметров.

4. ***Использование ALF-файлов***
   Описан специализированный формат ALF, используемый для сохранения на диске данных, собранных устройствами ADClab. Рассмотрены программные функции, облегчающие запись собранных данных в файлы ALF.

5. ***Приложение***
   В приложении приведены законченные примеры программ на C, использующих описанные функции для взаимодействия с устройствами ADClab.

1 Введение

Комплект средств разработки предназначен для создания приложений сбора и обработки данных от устройств EL200/EL100 в среде Microsoft Windows.

В качестве среды разработки предлагается использовать Microsoft Visual C/C++.

Для создания приложения, взаимодействующего с устройством, необходимо сделать следующее:

1. Запустить приложение Microsoft Visual C/C++.

2. Открыть новый или существующий проект, в котором предполагается организовать сбор и обработку данных от устройства.

3. Вставьте в исходные файлы проекта следующие строки:

```
#include   "../include/DllClient.h"
#include   "../include/IADCDevice.h"
#include   "ALF.cpp"
```

Файл DllClient.h содержит описание функций и констант.
Файл IADCDevice.h содержит описание интерфейса драйвера устройства.
Файл ALF.cpp содержит функции для работы с alf-файлами.

Эти файлы изначально находятся на установочном компакт-диске, поставляемом с устройством.

4. Главная функция программы должна начинаться следующим образом:

```
int main()
{
   DllClient DClient;
   IADCDevice* pADC =
(IADCDevice*)DClient.LoadDriver
                  ("EL200E", "IADCDevice", 0);
     .    .    .
// Текст программы с обращениями к функциям

}
```

В качестве первого параметра функции LoadDriver передается название используемого устройства “EL200E” или “EL100E”

В этом фрагменте устанавливается связь с драйвером устройства, после чего можно вызывать описываемые ниже функции как элементы класса pADC, например

```
pADC->Setup(1, 0, 0, 0);
```

Если указатель pADC, возвращаемый функцией DClient.LoadDriver, равен NULL, то это означает, что при установке связи с драйвером произошли ошибки и дальнейшая работа с функциями невозможна.

2 Описание структур данных

Структуры данных используются при вызове функций драйвера.

Структура ADCParametersBLOCK

Описание

Структура содержит настройки режима сбора данных и используется при вызове функции **Init**. Эта структура используется, когда в драйвере не создается отдельного потока для обработки принимаемых данных.

Элементы структуры

```
struct  ADCParametersBLOCK
{
    int             m_nStartOf;
    int             m_nBlockSize;
    int             m_nMode;
    int             m_nControl;
    int             m_nLevel;
    double          m_fFreqStart;
    double          m_fFreqPack;
    int             m_nChannelCount;
    int*            m_pnGains;
    int*            m_pnChannels;
};
```

Описание элементов

m_nStartOf Определяет режим запуска «сбор данных». Возможные значения заданы в таблице.

Значение параметра	Описание
ADCPARAM_START_PROGRAMM	
ADCPARAM_START_TIMER	Сбор данных начинается сразу после вызова функции Start
ADCPARAM_START_EXT	Сбор данных запускается замыканием контактов на разъеме
ADCPARAM_START_COMP	

m_nBlockSize Размер буфера отсчетов (число элементов массива), заполняемого при вызове функции GetData. Этот размер зависит от заданной частоты дискретизации АЦП, количества считываемых каналов и периода обращений к функции GetData. В любом случае, этот размер должен быть не меньше, чем общее количество отсчетов АЦП за интервал между обращениями к функции GetData при заданной частоте дискретизации. Важно отметить, что величина размера должна быть кратна величине:

2 x m_nChannelCount x 32,

где m_nChannelCount – количество считываемых каналов.

m_nMode Режим сбора данных.

Значение параметра	Описание
ADC_BLOCK_MODE_MODECYCLE	
ADC_BLOCK_MODE_MODECYCLE1	Циклический сбор данных в кольцевой буфер длиной 32К отсчетов.

	Программа пользователя должна периодически считывать собранные данные посредством вызова функции GetData. При этом интервал считывания должен быть таким, чтобы кольцевой буфер не переполнился.
ADC_BLOCK_MODE_MODECYCLE2	Циклический сбор данных в кольцевой буфер длиной 32К отсчетов. Сбор осуществляется в отдельном потоке, создаваемом драйвером. Для обработки каждой следующей порции собранных данных, из этого потока вызывается пользовательская CallBack – функция.
ADC_BLOCK_MODE_MODECYCLE3	
ADC_BLOCK_MODE_MODECYCLE_EVENT	

m_nControl Режим управления АЦП.
АЦП в устройствах EL200 является двухканальным, т.е. за каждый цикл одновременно считываются напряжения с двух каналов. Синхронные пары каналов имеют следующие номера: 1 – 17, 2 – 18, . . . , 16 – 32. Устройство EL100 считывает за цикл напряжение с одного канала.

Значение параметра	Описание
ADC_BLOCK_CONTROL_MODEZERO	
ADC_BLOCK_CONTROL_MODEDIFF	Задается работа АЦП в дифференциальном режиме (для

	EL200) . Получаемые функцией GetData данные содержат разницу отсчетов для каждой пары каналов, считываемой синхронно за цикл АЦП.
ADC_BLOCK_CONTROL_ZERO_MUL	
ADC_BLOCK_CONTROL_GAIN_PAIR	Получаемые функцией GetData данные содержат пары отсчетов, считываемые синхронно за цикл АЦП.

m_nLevel Этот параметр всегда должен быть равен 0.

m_fFreqStart Частота сбора данных АЦП. Следует отметить, что за один цикл сбора EL200 синхронно считывает напряжение в двух каналах, а EL100 - напряжение в одном канале.

m_fFreqPack Этот параметр всегда должен быть равен 0.

m_nChannelCount

Для EL200 - количество синхронных пар каналов, может принимать значения от 1 до 16. При этом в выходном массиве функции GetData будет содержаться по два отчета для каждой синхронной пары.

m_pnGains Для EL200 - массив, содержащий коэффициенты усиления для каждой

пары синхронных каналов. Каждый элемент представляет собой 32- разрядное целое число, при этом младшие 16 бит содержат коэффициент для одного канала из пары, а старший для другого. Коэффициенты могут принимать значения 1, 2, 5, 10, либо 1, 10, 100, 200. Размерность массива должна быть равна m_nChannelCount.

m_pnChannels

Для EL200 - массив целых 32-разрядных чисел, содержащий номера, используемых синхронных пар. Номера пар могут принимать значения от 1 до 16. Размерность массива должна быть равна m_nChannelCount.

Структура ADCParametersSTREAM

Описание

Структура содержит настройки режима сбора данных и используется при вызове функции **Init**. Эта структура используется, когда в драйвере создается отдельный поток для обработки принимаемых данных.

Элементы структуры

```
struct  ADCParametersSTREAM
{

    int             m_nStartOf;
    int             m_nBlockSize;
    int             m_nMode;
    int             m_nControl;
    int             m_nLevel;
    double          m_fFreqStart;
    double          m_fFreqPack;
    int             m_nChannelCount;
    int*            m_pnGains;
    int*            m_pnChannels;
    ADCParametersSTREAMCallBack
                    m_pCallBack;
    void*           m_vCallBackObj;
    void*           m_vCallBackData;
    int             m_vCallBackSize;
    int             m_vCallBackMode;

};
```


Описание элементов

Элементы структуры начиная с m_nStartOf и кончая m_pnChannels, полностью идентичны аналогичным элементам рассмотренной выше структуры ADCParametersBLOCK. Их описание здесь не приводится.

m_pCallBack Имя пользовательской CallBack – функции, вызываемой драйвером в специально созданном потоке. Список параметров функции описан ниже.

m_vCallBackObj Первый параметр, передаваемый драйвером в CallBack функцию. Может содержать, например, дескриптор alf – файла, в который записываются собираемые данные.

m_vCallBackData Второй параметр, передаваемый драйвером в CallBack функцию. Должен содержать адрес массива, в который драйвер поместит собранные данные.

m_vCallBackSize Размерность массива m_vCallBackData.

m_vCallBackMode Тип, к которому драйвер преобразует собираемые данные, т.е. тип элементов массива m_vCallBackData. Может принимать значения:

Значение параметра	Описание
ADC_DATA_MODE_ CONVERT2INT16	Собираемые данные преобразуются к типу unsigned short int
ADC_DATA_MODE_ CONVERT2INT16M	
ADC_DATA_MODE_ CONVERT2INT32	Собираемые данные преобразуются к типу unsigned int
ADC_DATA_MODE_ CONVERT2INT32M	
ADC_DATA_MODE_ CONVERT2FLOAT	Собираемые данные преобразуются к типу float
ADC_DATA_MODE_ CONVERT2DOUBLE	Собираемые данные преобразуются к типу double

Описание CallBack – функции

В случае создания драйвером отдельного потока для обработки, программа получает данные через вызов заданной CallBack – функции. Драйвер вызывает эту функцию, когда в его буфере будет накоплено количество отсчетов, достаточное для заполнения выходного массива данных **pData** (размером **size**). При этом время обработки данных в CallBack – функции не должно быть слишком большим, так, чтобы к моменту следующего обращения к этой функции обработка предыдущей порции закончилась. Иначе часть данных может быть потеряна. Важно помнить, что Windows является многозадачной средой и в некоторых случаях время обработки очередного считанного блока данных может увеличиться.

Список параметров **CallBack** - **функции**:

```
nt __stdcall   ADCCallBack
(void* pObj, void*& pData, int size, int mode)
```


Параметры

pObj Значение параметра задается в поле m_vCallBackObj структуры ADCParametersSTREAM при вызове функции Init. Обычно содержит дескриптор alf– файла.

pData Значение параметра задается в поле **m_vCallBackData** структуры ADCParametersSTREAM при вызове функции Init. По этому адресу массива перед вызовом функции драйвер разместит собранные данные. Для устройства EL200 порядок размещения отсчетов в массиве для разных каналов определяется следующей таблицей.

Смещение в массиве	0	1	2	3	4	5	6	7
Номер канала	1	9	2	10	3	11	4	12

Смещение в массиве	8	9	10	11	12	13	14	15
Номер канала	5	13	6	14	7	15	8	16

Смещение в массиве	16	17	18	19	20	21	22	23
Номер канала	17	25	18	26	19	27	20	28

Смещение в массиве	24	25	26	27	28	29	30	31
Номер канала	21	29	22	30	23	31	24	32

size Значение параметра задается в поле **m_vCallBackSize** структуры ADCParametersSTREAM при вызове функции Init. Содержит количество собранных отсчетов.

mode Значение параметра задается в поле **m_vCallBackMode** структуры ADCParametersSTREAM при вызове функции Init. Содержит тип, к которому преобразуются собираемые данные.

3 Описание функций

Обращение к функциям драйвера и описание входных и выходных параметров. Описываемые функции доступны программисту как элементы класса, адрес экземпляра которого `pADC` возвращается функцией DClient.LoadDriver, которую нужно вызвать в начале программы. Например

```
pADC->Setup(1, 0, 0, 0);
```

Функция Setup

Описание

Захват устройства ADClab, подключенного к компьютеру через USB. Одновременно к компьютеру могут быть подключены несколько таких устройств, но одно устройство может быть одновременно захвачено только одной программой. Одна программа может захватывать несколько устройств. Работа с устройством должна начинаться с обращения к этой функцией.

Синтаксис

```
int stdcall Setup
( int baseAdr,int DRQ,int IRQ,HANDLE hEvent );
```

Параметры

baseAdr	Номер устройства, подключенного к компьютеру. Номер определяется предварительно установленным драйвером устройства и может принимать значения 1, 2, …
DRQ	Этот параметр должен быть равен 0.
IRQ	Параметр должен быть равен 0

hEvent	Параметр должен быть равен 0

Код(ы) возврата

> 0	нормальное завершение. Устройство захвачено программой и готово к работе.
≤ 0	при захвате произошли ошибки. Работать с устройством нельзя. Следует освободить устройство вызовом функции Release и выйти из программы.

Функция Release

Описание

Освобождение устройства ADClab, ранее захваченного функцией Setup. Функция должна вызываться перед выходом из программы.

Синтаксис

```
unsigned stdcall  Release;
```

Код(ы) возврата

Нет

Функция Init

Описание

Функция определяет настройки режима сбора данных.

Синтаксис

```
int stdcall  Init
( int mode, ADCParametersBase* prm, float* x )
```

Параметры

mode — Определяет режим работы устройства. Возможно только значение ADC_INIT_MODE_INIT

prm — Структура типа ADCParametersBase. Содержит параметры сбора данных, описание приведено в разделе «Описание структур данных».

x — Параметр должен быть равен 0

Код(ы) возврата

> 0 нормальное завершение.
≤ 0 при определении настроек произошли ошибки.

Функция Start

Описание

Инициализация процесса сбора данных. После вызова этой функции данные из устройства начинают передаваться в буфер драйвера размером 32К. Если используется режим без создания отдельного потока для обработки данных, то программа должна периодически обращаться к функции GetData для считывания накопленных данных. Частота этих обращений должна быть такой, чтобы не возникало переполнения внутреннего буфера драйвера. Если используется режим с созданием отдельного потока, программист должен предусмотреть CallBack – функцию для обработки считанных данных. Частота вызова CallBack – функции определяется драйвером. При этом длительность выполнения CallBack – функции не должна превышать периода обращения к ней. Этот период вычисляется драйвером исходя из заданной частоты АЦП и размера массива для приема данных. Эти параметры рассмотрены в “Описании структур” выше.

Синтаксис

```
int stdcall  Start();
```

Код(ы) возврата

> 0 нормальное завершение. Процесс сбора данных начался.
≤ 0 произошли ошибки

Функция Stop

Описание

Остановка процесса сбора данных, инициированного вызовом функции Start. После выполнения этой функции прекращается передача данных из устройства в компьютер.

Синтаксис

```
void  stdcall  Stop();
```

Код(ы) возврата

Нет

Функция GetData

Описание

Функция предназначена для считывания накопленных данных из буфера драйвера в программный массив. Функция используется в том случае, когда заданный режим работы драйвера (см. “Описание структур”) не предполагает создание отдельного потока для обработки отсчетов. Программист должен периодически вызывать эту функцию для считывания каждой очередной порции данных. При этом интервал обращения не должен быть слишком малым. Иначе внутренний кольцевой буфер драйвера размером 32К может переполниться, и часть данных будет потеряна. В каждом конкретном случае величина интервала рассчитывается исходя из заданной частоты сбора и размера массива для приема данных. При этом следует учитывать, что Windows является многозадачной средой и поэтому в цикле обращения к функции GetData могут быть задержки. То есть интервал между обращениями к функции GetData должен быть несколько меньше рассчитанного.

Синтаксис

```
int  stdcall  GetData
( int mode, void* pAdr, int size, int offset);
```

Параметры

mode Тип, к которому драйвер преобразует собираемые данные, т.е. тип элементов массива **pAdr**. Может принимать значения:

Значение параметра	Описание
ADC_DATA_MODE_ CONVERT2INT16	Собираемые данные преобразуются к типу unsigned short int
ADC_DATA_MODE_ CONVERT2INT16M	
ADC_DATA_MODE_ CONVERT2INT32	Собираемые данные преобразуются к типу unsigned int
ADC_DATA_MODE_ CONVERT2INT32M	
ADC_DATA_MODE_ CONVERT2FLOAT	Собираемые данные преобразуются к типу float
ADC_DATA_MODE_ CONVERT2DOUBLE	Собираемые данные преобразуются к типу double

pAdr Адрес массива для приема считываемых данных. Тип элементов массива определяется параметром mode. Количество элементов массива определяется параметром **size.** Для устройства EL200 порядок размещения отсчетов в массиве для разных каналов определяется следующей таблицей.

Смещение в массиве	0	1	2	3	4	5	6	7
Номер канала	1	9	2	10	3	11	4	12

Смещение в массиве	8	9	10	11	12	13	14	15
Номер канала	5	13	6	14	7	15	8	16

Смещение в массиве	16	17	18	19	20	21	22	23
Номер канала	17	25	18	26	19	27	20	28

Смещение в массиве	24	25	26	27	28	29	30	31
Номер канала	21	29	22	30	23	31	24	32

size Количество элементов в массиве **pAdr**.

offset Смещение от начала массива **pAdr**, начиная с которого размещаются собранные данные (обычно 0).

Код(ы) возврата

> 0	нормальное завершение. Запрашиваемый блок данных считан и размещен по указанному адресу.
≤ 0	запрашиваемый блок данных еще не собран или ошибки при обращении к драйверу.

Функция PortIO

Описание

Функция предназначена для управления 8-разрядным блоком разовых команд, входящим в состав устройства EL200. За 1 обращение выдаются или принимаются сразу все 8 команд.

Синтаксис

```
int stdcall  PortIO
          ( int mode, void* pAdr, int size );
```

Параметры

mode — Параметр определяет режим работы функции. Может принимать следующие значения.

Значение параметра	Описание
ADC_PORTIO_DISABLE	Сброс блока разовых команд. Сброс нужно делать в начале и в конце программы. При этом параметры pAdr и size должны быть равны 0.
ADC_PORTIO_OUTB	Выдача 8-ми разовых команд.
ADC_PORTIO_INB	Прием 8-ми разовых команд.

pAdr — Параметр содержит адрес байта, в котором находятся значения разовых команд для выдачи, или в который помещаются принятые разовые команды.

size — Длина блока передаваемой информации в байтах, в данном случае должна быть 1.

Код(ы) возврата

> 0	нормальное завершение.
≤ 0	при работе функции обнаружены ошибки

Функция Get

Описание

Получение служебной информации из устройства EL100/200.

Синтаксис

```
int stdcall  Get ( int mode, void* value );
```

Параметры

mode — Параметр определяет вид запрашиваемой информации. Может принимать следующие значения.

Значение параметра	Описание
ADC_GET_RANGE_BIPOLAR	Запрос входного диапазона устройства для коэффициента усиления 1.0. В качестве параметра value должен быть указан адрес переменной типа float.
ADC_GET_SERIALNUMBER_LONG	Запрос серийного номера устройства EL100/200. В качестве параметра value должен быть указан адрес переменной типа long.
ADC_GET_SERIALNUMBER_STRING	Запрос серийного номера устройства EL100/200 в символьном виде. В качестве параметра value должна быть указана переменная типа char[].

value — Этот параметр содержит адрес блока памяти для размещения считываемой информации. Длина этого блока зависит от значения

параметра mode.

Код(ы) возврата

> 0	нормальное завершение.
≤ 0	при работе функции обнаружены ошибки

4 Использование ALF-файлов

Для сохранения собранных данных в файлы используется специально разработанный формат ADCLABFF. Файлы этого формата имеют расширение*.alf. ADCLABFF - бинарный формат файлов, используемый для хранения больших объемов данных. Данный формат обеспечивает возможность быстрой последовательной записи данных и быстрый доступ к данным при чтении. Далее для краткости, говоря о файлах, содержащих данные в формате ADCLABFF, будем называть их «alf-файлами».

Для работы с alf-файлами предусмотрено несколько структур данных, которые описаны ниже.

4.1. Описание структур данных

Структуры данных используются при работе с alf-файлами. Важно отметить, что элементы структур должны быть выровнены по границе одного байта, поэтому перед их описанием в программе должна быть директива:

```
#pragma pack(1)
```

*Структура ADCLABFF_*

Описание

Структура содержит название блока данных и его длину. Данная структура размещается в начале каждой из последующих описанных структур.

Элементы структуры

```
struct ADCLABFF_
{
    char name [ 24 ];
    __int64 len;
};
```

Описание элементов

name Содержит название блока данных, который расположен сразу после этой структуры.

len Содержит длину блока данных.

*Структура SAMPLESFMT_*

Описание

Структура содержит информацию о сохраняемой в файле выборке данных.

Элементы структуры

```
struct SAMPLESFMT_
{
     ADCLABFF_ el;
     unsigned          m_mask;
     int               m_channelsCount;
     double            m_frequencyPerChannel;
     char              m_samplesType;
     char              m_lowerSignificantBit;
     char              m_significantBitsCount;
     char              m_mode;
};
```

Описание элементов

el Структура типа ADCLABFF_, в которой элемент **name** должен содержать значение “SAMPLES_FORMAT ”, а элемент **len** должен быть равен 20.

m_mask Поле должно содержать значение SFMT_CHANNELS_COUNT | SFMT_FREQUENCY | SFMT_TYPE или 7.

m_channelsCount

Количество каналов, информация из которых записывается в файл. Может принимать значения от 1 до 32.

m_frequencyPerChannel

Частота выборки отсчетов для каждого из записываемых каналов. Определяется как частота сбора данных АЦП, деленная на количество каналов **m_channelsCount**.

m_samplesType

Должен быть равен 1 для записи отсчетов в формате float.

m_lowerSignificantBit

Параметр должен быть равен 0.

m_significantBitsCount

Параметр должен быть равен 0.

m_mode Параметр должен быть равен 0.

*Структура CHANNELS_INFO_HEADER_*

Описание

Содержит информацию о диапазонах данных для всех каналов.

Элементы структуры

```
struct CHANNELS_INFO_HEADER_
{
     ADCLABFF_ el;
     UINT       m_mask;
     double     m_rangemin;
     double     m_rangemax;
};
```

Описание элементов

el Структура типа ADCLABFF_, в которой элемент **name** должен содержать значение “ CHANNELS_INFO_HEADER”, а элемент **len** должен быть равен 20.

m_mask Поле определяет тип информации, находящейся в этой структуре должно содержать значение 3.

m_rangemin
Минимальное значение общего диапазона данных для всех каналов.

m_rangemax
Максимальное значение общего диапазона данных для всех каналов.

*Структура CHANNELS_INFO_*

Описание

Содержит информацию о диапазонах данных в каждом канале.

Элементы структуры

```
struct CHANNELS_INFO_
{
     int        m_ID;
     double     m_rangemin;
     double     m_rangemax;
};
```

Описание элементов

m_ID Поле содержит номер канала.

m_rangemin
Минимальное значение диапазона данных для канала, определяемого номером **m_ID**.

m_rangemax
Максимальное значение диапазона данных для канала, определяемого номером **m_ID**.

Структура SAMPLES_RECORD_INFO

Описание

Данная структура должна находиться в файле alf непосредственно перед структурой SAMPLES_RECORD_, содержащей выборку данных.

Элементы структуры

```
struct SAMPLES_RECORD_INFO_
{
    ADCLABFF_ el;
     __int64 off;
};
```

Описание элементов

el Структура типа ADCLABFF_, в которой элемент **name** должен содержать значение “ SAMPLES_RECORD_INFO”, а элемент **len** должен быть равен 8.

off Параметр должен быть равен 0.

*Структура SAMPLES_RECORD_*

Описание

Содержит заголовок для последующего блока, в который записываются отсчеты.

Элементы структуры

```
struct SAMPLES_RECORD_
{
    ADCLABFF_ el;
};
```

el Структура типа ADCLABFF_, в которой элемент **name** должен содержать значение “ SAMPLES_RECORD”, а все байты элемента **len** должны быть равны 0xff.

4.2. Описание содержимого alf – файла

Alf – файл состоит из заголовка и следующего за ним блока отсчетов.

Заголовок содержит следующие данные:

1. Структура типа ADCLABFF_, в которой элемент **name** должен содержать значение “ADCLABFFS”, а элемент **len** должен быть равен 0.

2. Структура типа SAMPLESFMT_, заполненная в соответствии с записываемыми далее данными.

3. Структура типа CHANNELS_INFO_HEADER_, заполненная в соответствии с записываемыми далее данными.

4. Структура типа ADCLABFF_, в которой элемент **name** должен содержать значение “CHANNELS_INFO”, а элемент **len** должен быть равен 20 **x** количество каналов, определенное в структуре SAMPLESFMT_.

5. Заполненные структуры типа CHANNELS_INFO_ для каждого канала. Их количество определяется числом каналов, заданных в структуре SAMPLESFMT_.

6. Структура типа SAMPLES_RECORD_INFO_.

7. Структура типа SAMPLES_RECORD_.

Далее следует блок отсчетов, состоящий из элементов длиной 4 **x** на количество каналов. Каждый элемент содержит последовательно расположенные отсчеты (по одному отсчету для каждого канала). Длина блока отсчетов не определена, но должна быть кратна величине 4 **x** на количество каналов. Таким образом, отсчеты располагаются до конца файла.

5 Приложение

5.1. *Пример использования устройств EL200E, EL200SD*

Вам будет достаточно копировать куски кода в свои проекты, для правильного функционирования устройств. Для упрощения понимания пример не содержит ничего, кроме управления устройством и элементарной печати хода выполнения.

Для устройств EL200xx с USB интерфейсом предусмотрен один режим работы, непрерывный сбор аналоговой информации во внутреннюю память с использованием FIFO буферизации. Каждый элемент FIFO представляет собой буфер размером 64 байта или 32 слова АЦП. Передача в PC производится порциями по 32 отсчета. Пока эти данные не собрались, устройство не передает их в компьютер. В данном примере драйвер не создает отдельного потока для обработки принимаемых данных.

```
/*#####################################################*/
#include    "Windows.h"
#include    "stdio.h"
#include    "conio.h"

#ifndef     DIM
#define     DIM(a)      (sizeof(a) / sizeof(a[0]))
#endif

#include    "../include/DllClient.h"
#include    "../include/IADCDevice.h"
#include    "ALF.cpp"
/*#####################################################*/
#define     NUMBER_OF_PACKETS   200
#define     FREQ_STREAM         60000.0
/*#####################################################*/
```


Собираем три синхронные пары каналов с единичным коэффициентом усиления. Можно поменять количество и значения.

```
#define     GAIN    1   // вместо 1 можно поставить 1,
10, 100, 200
int         Channels[3] = { 0, 1, 2 };
int         Gains[3]    = { GAIN + (GAIN<<16), GAIN +
                        (GAIN<<16), GAIN + (GAIN<<16) };

float       volt[DIM(Gains) * 10240];
/*-----------------------------------------------------------------------*/
```

Все указанные выше устройства работают одинаково, поэтому нужно лишь менять имя устройства.

```
int     main()
{
    int r = 0;          //  Переменная для DEBUG
```

Создаем фабрику для DLL потокового сбора ADC данных

```
    DllClient DClient;
```

Именно то место, где нужно подставить правильное название устройства. Получаем интерфейс

```
    IADCDevice* pADC =
           (IADCDevice*)DClient.LoadDriver("EL200E",
                                           "IADCDevice", 0);
    if(pADC == 0)
    {
        printf("\n ERROR - Object not created! \n");
        return 0;
    }
```

Номера устройств начинаются с 1. Захват устройства с номером 1.

```
    if(pADC->Setup(1, 0, 0, 0) <= 0)
    {
        printf("\n ERROR - Setup not ready! \n");
        pADC->Release();
        return 0;
    }
/*-------------------------------------------------------------------------*/
```

Считываем входной диапазон платы для коэффициента усиления 1.0.

```
    float fRange = 0.0f;
    r = pADC->Get(ADC_GET_RANGE_BIPOLAR, &fRange);
    printf("\n Bipolar Range = %5.2f", double(fRange));
```

Получение серийного номера устройства

```
    unsigned serN = 0;
    r = pADC->Get(ADC_GET_SERIALNUMBER_LONG, &serN);
    printf("\n Serial Number = 0x%08X", serN);
    char serNc [ 100 ]; serNc [ 0 ] = 0;
    r = pADC->Get(ADC_GET_SERIALNUMBER_STRING, serNc);
    printf("\n Serial Number = %s", serNc);
```

Читаем калибровки. Вообще они применяются автоматически и пользователю не нужны. Выводим дату калибровки устройства. То же, можно читать только когда не идет сбор данных. Иначе возможны сбои при высокой частоте сбора.

```
    ADCParametersCALIBRATION pc;
    pc.m_nADC   = 5;
    pc.m_nDAC   = 0;
    pc.m_nExtra = 0;
    r = pADC->Init(ADC_INIT_MODE_CHECK, &pc, 0);
    if(r <= 0)
        printf("\n Calibration Read ERROR!");
    else
```



```
      printf("\n Calibration Read OK. Date: %02d:%02d:%4d",
                  pc.m_nExtra & 0xFF, (pc.m_nExtra >> 8)
                                & 0xFF, pc.m_nExtra >> 16);
/*-------------------------------------------------------------------------*/
```

Сбор потока данных с режимом сканирования каналов. Структура параметров измерения.

```
    ADCParametersBLOCK pb;
    pb.m_nStartOf      = ADCPARAM_START_TIMER;
                                 // источник запусков;
    pb.m_nBlockSize    = DIM(volt);
                               // размер блока данных;
    pb.m_nMode         = ADC_BLOCK_MODE_MODECYCLE1;
     // режим сбора без отдельного потока, циклический;

    pb.m_nControl      = ADC_BLOCK_CONTROL_GAIN_PAIR;
                          // синхронные пары каналов;
    pb.m_nLevel        = 0;            // не применимо;
    pb.m_fFreqStart    = FREQ_STREAM;
                               // частота сбора данных;
    pb.m_fFreqPack     = 0.0;          // не применимо;

    pb.m_nChannelCount = DIM(Gains);
    pb.m_pnGains       = Gains;
    pb.m_pnChannels    = Channels;
```

Если возвращаемое значение больше 0, то все прошло успешно.

```
    r = pADC->Init(ADC_INIT_MODE_INIT, &pb, 0);

    HANDLE hFile = SaveALFHeader("Dat1.alf", 0, pb);
```

Запуск измерения. Сразу после этой строчки начинается процесс сбора данных. Данные складываются в кольцевой буфер.

```
    pADC->Start();
    printf("\n\n Start ADC stream...\n");

    for(int n = 0; n < NUMBER_OF_PACKETS;)
    {
```

Количество данных должно быть таким же, как и было заказано в pb.m_nBlockSize, кратно 32. Отдельного потока не создается, размер надо делать > 0.5 сек.

```
     int s = pADC->GetData(ADC_DATA_MODE_CONVERT2FLOAT,
                           volt, pb.m_nBlockSize, 0);
     if(s > 0)
     {
        SaveBlock(hFile, volt, pb.m_nBlockSize *
                                        sizeof(float));
        printf("\r Read Buffer %3d. Size = %d.", n, s);
        ++n;
        }
    }
```

Остановим сбор.

```
    pADC->Stop();

    CloseALF(hFile);
    printf("\n Stop ADC.");
/*-------------------------------------------------------------------------*/
```

Все!

```
    printf("\n Press any key for EXIT.");
    getch();
```

Освобождение устройства.

```
    pADC->Release();
    return 0;
}

/*#####################################################*/
```


5.2. Пример использования устройств EL200E, EL200SD с организацией отдельного потока для обработки данных

Показан пример управления устройством в режиме с созданием отдельного потока для обработки принимаемых данных. Все рекомендации, приведенные в примере 5.1 верны.

```
/*#####################################################*/
#include    "Windows.h"
#include    "stdio.h"
#include    "conio.h"

#ifndef     DIM
#define     DIM(a)      (sizeof(a) / sizeof(a[0]))
#endif

#include    "../include/DllClient.h"
#include    "../include/IADCDevice.h"
#include    "ALF.cpp"
/*#####################################################*/
#define     FREQ_STREAM         60000.0
/*#####################################################*/
int __stdcall   ADCCallBack(void* pObj, void*& pData,
int size, int mode)
{
    if(pObj == 0 || pData == 0 || size <= 0 || mode !=
ADC_DATA_MODE_CONVERT2FLOAT)
        return 0;

    float* pf = (float*)pData;
    HANDLE hFile = *(HANDLE*)pObj;
    if(hFile != INVALID_HANDLE_VALUE)
    {
        SaveBlock(hFile, pf, size * sizeof(float));
        putch('.');
```

```
        return 1;
    }
    return 0;
}
/*---------------------------------------------------------------------------*/
```

Собираем три синхронные пары каналов с единичным коэффициентом усиления. Можно поменять количество и значения.

```
#define     GAIN    1   // вместо 1 можно поставить 1,
                                                10, 100, 200
int         Channels[3] = { 0, 1, 2 };
int         Gains[3]    = { GAIN + (GAIN<<16), GAIN +
                        (GAIN<<16), GAIN + (GAIN<<16) };

float       mvolt[1024*100];
/*---------------------------------------------------------------------------*/
```

Все указанные выше устройства работают одинаково, поэтому нужно лишь менять имя устройства.

```
int     main()
{
    int r = 0;              //  Переменная для DEBUG
```

Создаем фабрику для DLL потокового сбора ADC данных

```
    DllClient DClient;
```

Именно то место, где нужно подставить правильное название устройства. Получаем интерфейс

```
    IADCDevice* pADC =
              (IADCDevice*)DClient.LoadDriver("EL200E",
                                        "IADCDevice",0);
    if(pADC == 0)
    {
        printf("\n ERROR - Object not created! \n");
        return 0;
```



```
    }
```

Номера устройств начинаются с 1. Захват устройства с номером 1.

```
    if(pADC->Setup(1, 0, 0, 0) <= 0)
    {
        printf("\n ERROR - Setup not ready! \n");
        pADC->Release();
        return 0;
    }
/*---------------------------------------------------------------------------*/
```

Считываем входной диапазон платы для коэффициента усиления 1.0.

```
    float fRange = 0.0f;
    r = pADC->Get(ADC_GET_RANGE_BIPOLAR, &fRange);
    printf("\n Bipolar Range = %5.2f", double(fRange));
```

Получение серийного номера устройства

```
    unsigned serN = 0;
    r = pADC->Get(ADC_GET_SERIALNUMBER_LONG, &serN);
    printf("\n Serial Number = 0x%08X", serN);

/*---------------------------------------------------------------------------*/
    static HANDLE hFile = INVALID_HANDLE_VALUE;
```

Сбор потока данных с режимом сканирования каналов. Структура параметров измерения.

```
 ADCParametersSTREAM ps;

    ps.m_Block.m_nStartOf      = ADCPARAM_START_TIMER;
                                  // источник запусков;
    ps.m_Block.m_nBlockSize    = 1024*100;
                                 // размер блока данных;
    ps.m_Block.m_nMode     = ADC_BLOCK_MODE_MODECYCLE2;
        // режим сбора циклический, в отдельном потоке.
    ps.m_Block.m_nControl =ADC_BLOCK_CONTROL_GAIN_PAIR;
                               // синхронные пары каналов;
    ps.m_Block.m_nLevel        = 0;    // не применимо;
    ps.m_Block.m_fFreqStart    = FREQ_STREAM;
                               // частота сбора данных;
```

```
ps.m_Block.m_fFreqPack      = 0.0;  // не применимо;
ps.m_Block.m_nChannelCount  = DIM(Gains);
ps.m_Block.m_pnGains        = Gains;
ps.m_Block.m_pnChannels     = Channels;

ps.m_pCallBack              = ADCCallBack;
// Вызывается из отдельного потока внутри драйвера;
ps.m_vCallBackObj           = &hFile;
                // указатель на данные пользователя;
ps.m_vCallBackData          = mvolt;
                                    // буфер данных;
ps.m_vCallBackMode    = ADC_DATA_MODE_CONVERT2FLOAT;
ps.m_vCallBackSize          = DIM(mvolt);
                            // Размер буфера данных;
```

Если возвращаемое значение больше 0, то все прошло успешно.

```
r = pADC->Init(ADC_INIT_MODE_INIT, &ps, 0);

hFile = SaveALFHeader("Dat1.alf", 0, ps.m_Block);
```

Запуск измерения. Сразу после этой строчки начинается процесс сбора данных. Данные складываются в кольцевой буфер.

```
pADC->Start();
printf("\n\n Start ADC stream CallBack...");

printf("\n Press any key to Stop.\n");
for(;;)
{
    if(_kbhit())
        break;
    Sleep(50);
}
```

Остановим сбор.


```
    pADC->Stop();

    CloseALF(hFile);
    printf("\n Stop ADC.");
    getch();
/*-------------------------------------------------------------------------*/
```

Все!

```
    printf("\n Press any key for EXIT.");
    getch();
```

Освобождение устройства.

```
    pADC->Release();
    return 0;
}

/*#########################################################*/
```