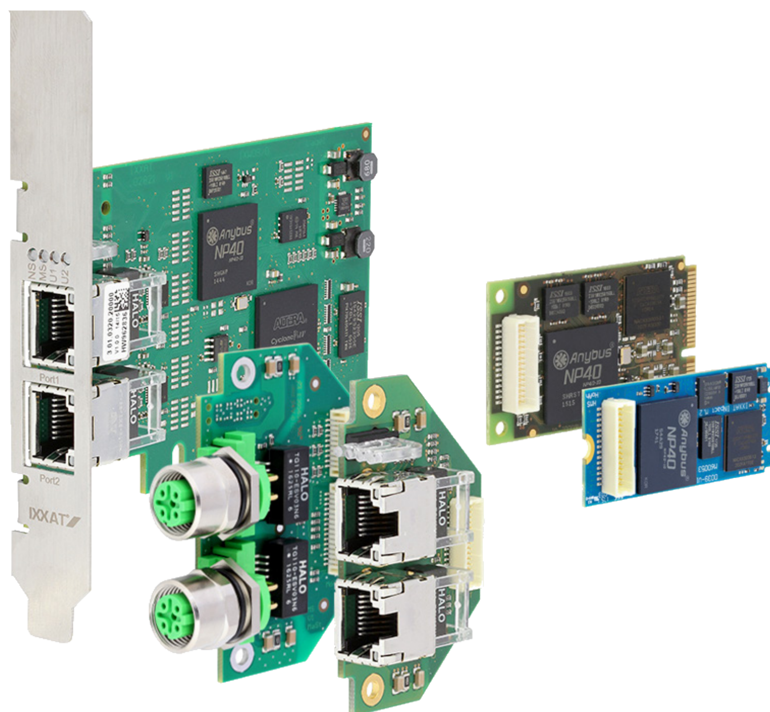
INpact Slave. Getting Started

Руководство по разработке программного обеспечения

HMS Technology Center Ravensburg GmbH

© HMS Technology Center Ravensburg GmbH

INpact Slave. Getting Started



Руководство по разработке программного обеспечения

ДЭЙТАМИКРО

© 2018, Перевод ООО “ДЭЙТАМИКРО“

Важная информация для пользователя

В процессе подготовки данного документа были приняты все меры предосторожности. Пожалуйста, сообщите HMS Industrial Networks AB, если обнаружилась неточность.

Мы, HMS Industrial Networks AB, оставляем за собой право вносить изменения в нашу продукцию в соответствии с нашей политикой постоянного совершенствования продуктов. Информация, содержащаяся в данном документе, может быть изменена без предварительного уведомления и поэтому не следует рассматривать ее в качестве описания для будущих версий. HMS Industrial Networks AB не несет ответственность за любые ошибки, которые могут появиться в этом документе.

Существует множество способов применения описываемого продукта. Тех, кто несет ответственность за использование этого устройства должны убедиться, что все необходимые меры были приняты, убедиться, что приложения отвечают всем требованиями производительности и безопасности, включая любые применимые законы, нормы, коды и стандарты.

HMS Industrial Networks AB ни при каких обстоятельствах не несет ответственности за любые проблемы, которые могут возникнуть в результате неправильного использования или использования в несоответствии с документированными особенностями этого продукта.

Примеры и иллюстрации в данном документе, приведены исключительно в демонстративных целях.

Права на Интеллектуальную Собственность

HMS Industrial Networks AB обладает правами интеллектуальной собственности на технологии, воплощенные в продукте, описанные в этом документе. Эти права включают патенты и патентные заявки в США и других стран.

Номер документа: 4.02.0320.20000

Версия: 1.3

СОДЕРЖАНИЕ

Важная информация для пользователя 2

Права на Интеллектуальную Собственность 2

1. Руководство пользователя 5

1.1. Связанные документы 5

1.2. История документа 5

1.3. Товарные знаки 5

1.4. Условные обозначения 5

1.5. Глоссарий 6

2. Введение 7

2.1. Требования к системе 7

2.2. Обзор 8

3. Инсталляция 10

3.1. Windows 10

3.2. Linux 10

4. Настройка 12

4.1. Настройка системы 12

4.2. Настройка INpact 12

4.2.1. Параметры сообщений и данных процесса 12

4.2.2. Параметры ADI 13

4.2.3. Параметры вывода отладочных событий 13

4.2.4. Время запуска 13

5. Запуск примера приложения 14

5.1. Выбор отображения ADI и данных процесса 14

5.2. Реализация функции main 14

5.2.1. Инициализация оборудования 14

5.2.2. Основной цикл 15

5.2.3. Освобождение оборудования 17

5.3. Компиляция и запуск 17

5.3.1. Windows 17

5.3.2. Linux 18

6. Адаптация и настройка примера приложения 19

6.1. Сетевая идентификация 19

6.1.1. Хост-приложение – Сети 19

6.1.2. Хост-приложение – Другое 20

6.1.3. Объекты хост-приложения – Дополнительно 20

6.2. ADI и отображение данных процесса 20

6.3. Функции обратного вызова обработки данных процесса 24

6.4. Обработка событий 25

© HMS Industrial Networks AB
© 2018, Перевод DATAMICRO LLC

6.5. Обработка сообщений 27
6.5.1. Дополнительные функции в Windows 27
6.5.2. Пример 1: Отправка команды и получение ответа 28
6.5.3. Пример 2: Получение команды и отправка ответа 29
A Обзор ПО 31
A.1 Файлы и папки 31
A.2 Файлы верхнего уровня 31
A.3 Интерфейс драйвера 31
A.4 Внутренние файлы драйвера (только для чтения) 31
A.5 Файлы адаптации системы 32
B Описание API 33
B.1 Функции API 33
B.2 Функции, связанные с API 34
B.3 Функции обратного вызова API 34
B.4 Вспомогательные функции 35
C Автомат состояний хост-приложения 36

1. Руководство пользователя

1.1. Связанные документы

Документ	Автор
Anybus CompactCom 40 Руководство по проектированию программного обеспечения (см. www.anybus.com)	HMS
Anybus CompactCom 40 Сетевые руководства	HMS
Руководство пользователя INpact PCIe	HMS

1.2. История документа

Версия	Дата	Описание
1.0	Март 2016	Первый выпуск
1.1	Февраль 2017	Скорректирована структура каталогов приложения
1.2	Июль 2017	Добавлена информация по Linux, уточнена информация по Windows
1.3	Март 2018	Незначительные исправления

1.3. Товарные знаки

IXXAT® является зарегистрированным товарным знаком HMS Industrial Networks. Все другие торговые марки упомянутые в этом документе – собственность их соответствующих держателей.

1.4. Условные обозначения

Инструкции и результаты показаны следующим образом:

★ Инструкция 1

★ Инструкция 2

☆ Результат 1

☆ Результат 2

Списки представлены следующим образом:

- Пункт 1
- Пункт 2

Жирный шрифт указывает на интерактивные детали, такие как разъемы и переключатели на оборудовании, или меню и кнопки в графическом интерфейсе пользователя.

`Этот шрифт используется для обозначения программного кода и других видов данных ввода/вывода, таких как сценарии конфигурации.`

Это перекрестная ссылка внутри документа: "1.4. Условные обозначения" на стр. 5

Это внешняя ссылка (URL): www.hms-networks.com

Это дополнительная информация, которая может облегчить установку и/или эксплуатацию

Эта информация, которая поможет избежать риск снижения функциональности и/или повреждения оборудования, или риск сетевой безопасности.

1.5. Глоссарий

ADI (Application Data Instance)	Экземпляр данных приложения
API (Application Programming Interface)	Интерфейс прикладного программирования
IDE (Integrated Development Environment)	Интегрированная среда разработки
ABCC (Anybus CompactCom)	

© HMS Technology Center Ravensburg GmbH
© 2018, Перевод ООО "ДЭЙТАМИКРО"

2. Введение

В начале реализации INpact Slave, для ускорения процесса разработки, можно использовать код примера хост-приложения. Код примера хост-приложения включает в себя драйвер Anybus CompactCom (ABCC), который выступает в роли связующего звена между микросхемой NP40 и хост-приложением. Драйвер имеет API (Application Programming Interface), который определяет общий интерфейс к драйверу. Также в код примера включён пример приложения, которое использует API и представляет собой простое приложение, которое можно использовать как основу конечного продукта.

Это руководство описывает пошаговую реализацию драйвера и примера приложения. От программиста требуется базовое знание объектной модели Anybus CompactCom и коммуникационного протокола.

См. список рекомендуемых к прочтению документов в *"1.1. Связанные документы" на стр. 5.*

Руководство описывает настройки по умолчанию и функции, показывает как запустить простое приложение и как настроить код примера на целевой продукт. Затем, если необходимо, приложение может быть расширено.

Драйвер доступен для следующих операционных систем:

- Windows 7/10
- Linux 32/64 бита (архитектура Intel)
- INtime 6.1
- QNX x86

2.1. Требования к системе

Windows

- Visual Studio 2010 или выше

Драйвер IDL для Linux

- ПК с ядром Linux версии 2.6.X или более старшей
- Драйвер HMS IDL для Linux
- GCC версии 4.2 или более новой и рабочая среда сборки ядра
- libc6 версии 2.3.4 или более новой
- libstdc++6 версии 3.4 или более новой
- libgcc1 версии 3.0 или более новой
- Eclipse CDT
- GNU Make

Драйвер тестировался под дистрибутивами Linux Debian/Ubuntu и open SUSE

© HMS Industrial Networks AB
© 2018, Перевод DATAMICRO LLC

2.2. Обзор

Код драйвера специально адаптирован под INpact Slave.

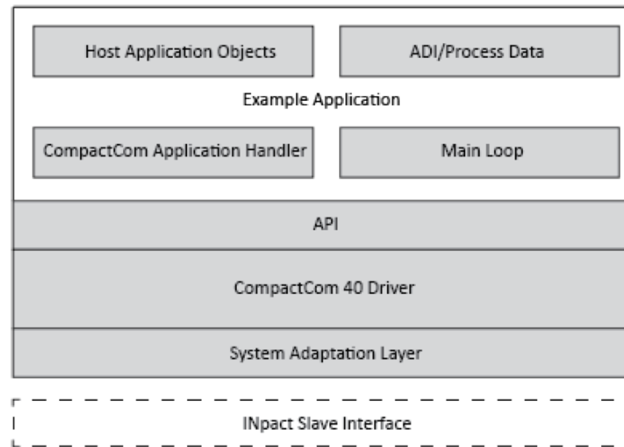


Рис. 2-1. Различные части кода примера хост-приложения

Библиотеки и заголовочные файлы, которые используются в коде примера, включены в драйвер.

Код примера хост-приложения разделён на различные папки.

В Windows папки автоматически устанавливаются в директории `C:\Users\Public\Documents\HMS\IXXAT INpact\Samples\IDL\src`.

С Linux, файлы должны быть извлечены из драйверного пакета IDL (`src/IdlApp`).

/abcc_abp (часть приложения)

- Содержит все объекты Anybus и определения коммуникационного протокола
- Файлы обновляются когда доступны новые версии
- Не изменяйте содержащиеся в этой папке файлы

/abcc_adapt (часть драйвера)

- Содержит файлы адаптации и конфигурации
- Файлы используются для адаптации кода драйвера и примера к системному окружению
- Не изменяйте содержащиеся в этой папке файлы

/abcc_drv (часть драйвера)

- Содержит исходные и заголовочные файлы драйвера
- Файлы обновляются когда доступны новые версии
- Не изменяйте содержащиеся в этой папке файлы

/abcc_obj

- Содержит все реализации объектов хост-приложения
- Содержащиеся в этой папке файлы могут быть модифицированы при необходимости, для оптимизации и/или добавления функций

/example_app

Пример приложения, включая:

- Основной конечный автомат управляющий инициализацией, перезапуском, нормальным состоянием и состоянием ошибки.
- Шаблоны конечного автомата, демонстрирующие как отправлять сообщения.

© HMS Technology Center Ravensburg GmbH
© 2018, Перевод ООО “ДЭЙТАМИКРО“

- Реализация функций обратного вызова, необходимых для драйвера.
- Определение ADI (Application Data Instances) и установка отображения данных процесса по умолчанию.
- Файлы должны быть адаптированны к приложению программистом. Содержащиеся в этой папке файлы могут быть модифицированы при необходимости, для оптимизации и/или добавления функций.

© HMS Industrial Networks AB
© 2018, Перевод DATAMICRO LLC

3. Инсталляция

3.1. Windows

- Загрузите драйвер VCI V4 с сайта www.ixxat.com
- Чтобы проверить готовность к использованию интерфейсной платы и драйвера, запустите файл **IDLc_TestApp.exe** в каталоге **C:\Users\Public\Documents\ HMS\IXXAT INpact\Samples\IDL\bin\x32 resp. .\bin\x64**

3.2. Linux

После установки интерфейсных плат модуль ядра IDL должен быть установлен и скомпилирован, чтобы иметь возможность использовать IDL.

- Загрузите пакет драйверов IDL с сайта www.ixxat.com и распакуйте

Компиляция модуля ядра

- Установите исходный код используемого в системе ядра
- Убедитесь, что файл конфигурации ядра `/usr/src/linux/.config` существует или что пакет linux-header соответствует версии ядра

Следующие шаги показывают, как использовать ассистент модулей (module-assistant - m-a) для создания рабочей среды сборки ядра с Debian/Ubuntu.

- Чтобы установить ассистент модулей, выполните команду **sudo apt-get install module-assistant**
- С помощью команды **sudo module-assistant prepare** ассистент модулей устанавливает необходимые пакеты
- Чтобы собрать модуль ядра, запустите `make` в каталоге исходных файлов модуля ядра `src/KernelModule`
- Чтобы установить модуль ядра, выполните команду **sudo make install**
  - Модуль будет скопирован в каталог `/lib/modules/$(KERNEL_VERSION)/kernel/drivers/misc/`
  - Сопутствующий файл правил udev будет скопирован в `/etc/udev/rules.d`
- Можно редактировать файл правил udev, чтобы предоставить узлам устройства, расположенным в каталоге `/dev` разные разрешения
  - Makefile вызывает `depmod -a` для экспорта файла `modalias` и `modprobe` для загрузки модуля
  - Makefile копирует библиотеки в `/usr/lib`, настраивает все необходимые ссылки и запускает **idconfig**

В зависимости от дистрибутива может потребоваться адаптация `TARGET_PATH` в Makefile. Например, в 64-битной системе openSUSE путь к библиотечным файлам по умолчанию - `/usr/lib64`.

- Используйте команду **cat/proc/idl**, чтобы узнать, какие интерфейсы найдены и какие адреса ввода/вывода и прерывания заняты

Встраиваемые системы без заголовочных файлов ядра

Например, при использовании Raspberry Pi без стандартной версии ядра, нет доступных файлов ядра или заголовочных файлов. Ядро должно быть перекомпилировано и установлено на устройство вручную, чтобы позволить IDL функционировать.

Придерживайтесь инструкций, см. https://www.raspberrypi.org/documentation/linux/kernel/building.md.

Использование приложения IDL

- Извлеките файлы из пакета драйвера IDL на ПК с установленным Linux
- Перейдите в каталог `IdlLinux_[ARCH]/bin/release` (`[ARCH]` может быть `i386` или `amd64`) и введите `./LinuxIdlApp` в консоли, для запуска приложения
  - Приложение открывает первую поддерживаемую плату и переводит её в состояние `WAIT_PROCESS`
  - Теперь мастер шины может выполнять дальнейшие действия

- По умолчанию активируется приложение **APPL_ADI_SETUP_SPEED**
- Если приложение распознает интерфейсную плату INpact, оно выводит данные, аналогичные следующим:
  - **Network type**: EtherCAT
  - **Firmware version**: 1.10 build 1
  - **Network type** 87
  - **Serial number**: 0:E:B1:0

4. Настройка

4.1. Настройка системы

Общие настройки для системного окружения устанавливаются в драйвере и сохраняются в:

- Windows: `abcc_adapt/abcc_td.h`
- Linux: `inc/abcc_td.h`

Поддерживаются следующие настройки:

- Порядок следования байтов – младшими вперед
- Восьмибитовая система кодирования символа

4.2. Настройка INpact

Определения и функции сохраняются в:

- Windows: `abcc_adapt/abcc_drv_cfg.h`
- Linux: `inc/abcc_drv_cfg.h`

4.2.1. Параметры сообщений и данных процесса

Значения, установленные в драйвере		Описание
#define ABCC_CFG_MAX_NUM_APPL_CMDS	32	Количество команд сообщений, которые могут быть отправлены без получения ответа. Необходимо минимум два сообщения
#define ABCC_CFG_MAX_NUM_ABCC_CMDS	32	Количество команд сообщений, которые могут быть получены без отправки ответа. Необходимо минимум два сообщения
#define ABCC_CFG_MAX_MSG_SIZE	1524	Размер самого большого сообщения в байтах, которое будет использоваться
#define ABCC_CFG_MAX_PROCESS_DATA_SIZE	4096	Максимальный размер данных процесса в байтах, которые будут использоваться в любом направлении. Примечание: Актуальный используемый размер зависит от используемого протокола (см. Anybus CompactCom 40 Руководство по проектированию программного обеспечения).
#define ABCC_CFG_REMAP_SUPPORT_ENABLED	TRUE	Разрешает или запрещает поддержку драйвера и объекта данных приложения для команды перена-значения

© HMS Technology Center Ravensburg GmbH
© 2018, Перевод ООО “ДЭЙТАМИКРО“

4.2.2. Параметры ADI

Значения, установленные в драйвере		Описание
#define ABCC_CFG_STRUCT_DATA_TYPE	TRUE	Влияет на AD_AdiEntryType в abcc_drv/inc/abcc_ad_if.h, используется для определения пользовательских ADI. Объём используемой памяти увеличевается
#define ABCC_CFG_ADI_GET_SET_CALLBACK	TRUE	Влияет на AD_AdiEntryType в abcc_drv/inc/abcc_ad_if.h, используется для определения пользовательских ADI. Посылает уведомления через функции обратного вызова каждый раз, когда ADI читается или записывается. Если ADI читается по сети, функция обратного вызова вызывается перед действием. Если ADI записывается по сети, функция обратного вызова вызывается после действия
#define ABCC_CFG_64BIT_ADI_SUPPORT	FALSE	Поддержка 64-битных типов данных в приложении отключена

4.2.3. Параметры вывода отладочных событий

Для целей разработки, разработчику доступны несколько функций отладки. Следующие определения влияют на вывод отладочной информации драйвером. Определения описаны в \abcc_adapt (Linux: inc/abcc_drv_cfg.h).

- #define ABCC_CFG_ERR_REPORTING_ENABLED (TRUE)

  Включение определения ABCC_CFG_ERR_REPORTING_ENABLED активирует функцию обратного вызова сообщения об ошибках ABCC_CbfDriverError(). Функция описана в abcc_drv/inc/abcc.h.

- #define ABCC_CFG_DEBUG_EVENT_ENABLED (FALSE)

  Включение определения ABCC_CFG_DEBUG_EVENT_ENABLED активирует поддержку драйвером вывода внутренних событий отладки. ABCC_PORT_DebugPrint() используется для вывода отладочной информации. Функция описана в abcc_adapt/abcc_sw_port.h

4.2.4. Время запуска

Время запуска передаётся в ABCC_StartDriver(). Если ничего не определено (0) в качестве значения по умолчанию используется 1500 мс.

5. Запуск примера приложения

Доступен уровень API, который определяет общий интерфейс к драйверу для всех сетевых приложений. Предоставляется пример приложения чтобы продемонстрировать, как стандартное приложение реализует драйвер, используя API. Для этого шага, будут использованы параметры по умолчанию (идентификация HMS).

API содержится в:

- Windows: `abcc_drv/inc/abcc.h`
- Linux: `inc/abcc.h`

5.1. Выбор отображения ADI и данных процесса

Данные процесса являются неотъемлемой частью приложения. Данные процесса добавляются в приложение путём создания ADI (Application Data Instances) и их отображения на требуемые области данных процесса (чтения или записи).

★ В Windows откройте **IDLc_TestApp_VS2010.vcxproj** в Visual Studio и убедитесь, что включено отображение `APPL_ADI_SETUP_SPEED` в `appl_adi_config.h`

★ Для Linux по умолчанию `APPL_ADI_SETUP_SPEED` активируется в `/src/IdlApp/example_app/appl_adi_config.h`

★ Для описания отображения см. `appl_speed.c` в разделе "6.2. ADI и отображение данных процесса" на стр. 20

Изменение отображения ADI в Linux

Отображения ADI можно найти в файлах `IdlLinux_[ARCH]/src/IdlApp/example_app/app_adimap_*.c`. Каждый файл является независимым примером. В файлах `app_adimap_*.c` есть массив, который содержит все доступные ADI.

★ Чтобы активировать пример, определите идентификатор `APPL_ACTIVE_ADI_SETUP` в `IdlLinux_[ARCH]/src/IdlApp/example_app/appl_adi_config.h`

5.2. Реализация функции main

Выполнение приложения начинается с функции main. В общем примере проекта она находится в **main.c**. Функция main разделена на три части: инициализацию оборудования, основной цикл и освобождение оборудования.

5.2.1. Инициализация оборудования

Linux

С Linux можно использовать несколько интерфейсных плат INpact с одним приложением и одним ПК с установленным Linux. Доступ к интерфейсным платам осуществляется через функции `ABCC_OpenController()` и `ABCC_CloseController()`.

`ABCC_OpenController()`
Эта функция открывает соединение с оборудованием. Если оборудование используется другим приложением, вызов завершается ошибкой.

- `dwHwIndex` открывает оборудование с заданным индексом
- `pstcHwPara` специфические для оборудования параметры
- `hCtrl` дескриптор оборудования
- `TRUE` если оборудование успешно открыто

`ABCC_CloseController()`
Эта функция закрывает соединение с оборудованием и освобождает дескриптор. Если закрытие прошло успешно, дескриптору присваивается значение `IDL_INVALID_HANDLE`.

- `hCtrl` освобождаемый дескриптор

© HMS Technology Center Ravensburg GmbH
© 2018, Перевод ООО “ДЭЙТАМИКРО“

ABCC_HwInit()
Эта функция инициализирует оборудование, необходимое для связи с интерфейсной платой.

- Должна вызываться один раз во время инициализации после включения питания
- Драйвер можно перезапустить без повторного вызова этой функции

Windows

Стартовое приложение не может осуществлять одновременный доступ к двум устройствам INpact, но драйвер INpact имеет такую возможность. При необходимости обратитесь в HMS за дополнительной информацией и помощью.

ABCC_HwInit()
Эта функция инициализирует оборудование, которое требуется для связи с интерфейсной платой.

- Должна вызываться один раз во время инициализации после включения питания
- Драйвер можно перезапустить без повторного вызова этой функции
- Чтобы использовать диалоговое окно для выбора оборудования, установите для параметра `fShowDialog` значение `TRUE`
- `pszHardwareSerial` определяет серийный номер выбранного оборудования, если параметр `fSelectDlg` равен `FALSE`
- Если инициализация прошла успешно, функция возвращает `ABCC_EC_NO_ERROR`

5.2.2. Основной цикл

В основном цикле начинается выполнение приложения. В примере стандартного проекта, он находится в `main.c` и практически одинаков для Windows и Linux.

APPL_HandleAbcc()
Эта функция обрабатывает конечный автомат и осуществляет перезагрузку, выполнение и завершение драйвера.

- Должна вызываться периодически из основного цикла
- Каждый раз при вызове этой функции возвращается состояние драйвера

Состояние	Описание
APPL_MODULE_NO_ERROR	Интерфейс функционирует без ошибок. Это стандартный ответ если всё выполняется нормально
APPL_MODULE_NOT_DETECTED	Интерфейс не обнаружен. Должно быть выполнено информирование пользователя
APPL_MODULE_NOT_SUPPORTED	Обнаружен неподдерживаемый интерфейс. Должно быть выполнено информирование пользователя
APPL_MODULE_NOT_ANSWERING	Возможные причины: Выбрано неправильное API, интерфейс повреждён
APPL_MODULE_RESET	От интерфейса поступил запрос на сброс. Запрос на сброс был получен по сети. Приложение отвечает за перезапуск системы
APPL_MODULE_SHUTDOWN	Поступил запрос на завершение работы
APPL_MODULE_UNEXPECTED_ERROR	Произошла непредвиденная ошибка. Должно быть выполнено информирование пользователя. Если необходимо, перевидите выводы в безопасное состояние

Для получения дополнительной информации см. "C Автомат состояний хост-приложения" на стр. 36.

ABCC_RunUI()
Простой интерфейс пользователя для взаимодействия с примером приложения.

© HMS Industrial Networks AB
© 2018, Перевод DATAMICRO LLC

ABCC_RunTimerSystem()

- Должна вызываться периодически с известным периодом (время в мс, прошедшее со времени последнего вызова):
  - Либо поставив известную задержку в основном цикле и вызывая функцию на каждой итерации
  - или установив прерывание по таймеру
- Ответственна за обработку всех таймеров драйвера
- Рекомендуется вызывать эту функцию на регулярной основе из обработчика прерывания таймера
- Без этой функции функциональность таймаутов и сторожевого таймера не работает

Windows

```
while( LOOP_QUIT != bLoopState )
   {
      bLoopState = LOOP_RUN;

      eAbccHandlerStatus = APPL_HandleAbcc();

      switch( eAbccHandlerStatus )
      {
         case APPL_MODULE_NO_ERROR:
            //
            // если тест действительно запущен, и настройка INpact прошла успешно,
            // завершить приложение !
            if (( fTest )
               && ( IsResetRequestAllowed (ABP_RESET_POWER_ON) ))
            {
               bLoopState = LOOP_QUIT;
            }
            break;
         case APPL_MODULE_RESET:
            APPL_RestartAbcc();
            break;
         default:
            bLoopState = LOOP_QUIT;
            break;
      }

      if ( bLoopState == LOOP_RUN )
      {
         bLoopState = RunUi();
      }

      switch( bLoopState )
      {
         case LOOP_RESET:
            APPL_RestartAbcc();
            bLoopState = LOOP_RUN;
            break;
         case LOOP_RUN:
         case LOOP_QUIT:
         default:
            break;
      }

      if ( bLoopState == LOOP_RUN )
      {
         Sleep( iSleepTimeMS );
         ABCC_RunTimerSystem( iSleepTimeMS );
      }
   }
```

© HMS Technology Center Ravensburg GmbH
© 2018, Перевод ООО “ДЭЙТАМИКРО“

Рекомендуется использовать прерывание таймера для вызова этой функции. Тем не менее, при реализации, для простоты отладки, на начальном этапе можно не использовать прерывание таймера

Linux

```
while( eAbccHandlerStatus == APPL_MODULE_NO_ERROR )
   {
      eAbccHandlerStatus = APPL_HandleAbcc(hCtrl);
      if(APPL_MustQuit())
         APPL_Shutdown(hCtrl);

#if( !USE_TIMER_INTERRUPT )
      ABCC_RunTimerSystem( hCtrl, APPL_TIMER_MS );
      DelayMs( APPL_TIMER_MS );
#endif

      switch( eAbccHandlerStatus )
      {
         case APPL_MODULE_RESET:
            Reset();
            break;
         default:
            break;
      }
   }
```

5.2.3. Освобождение оборудования

Linux

APPL_Shutdown()

- ABCC_HWReset() вызывается для сброса драйвера
- Устанавливает состояние в APPL_HALT

Windows

ABCC_Shutdown()

- Останавливает драйвер и переводит его в состояние SHUTDOWN
- ABCC сбрасывается

ABCC_HwRelease();

Эта функция освобождает оборудование.

5.3. Компиляция и запуск

5.3.1. Windows

- Скомпилируйте проект
- Убедитесь что проект скомпилировался без ошибок
- Запустите проект
- Убедитесь что хост-приложение может взаимодействовать с интерфейсом
- Убедитесь что возможен обмен данными по сети

5.3.2. Linux

Компиляция приложения IDL

- Если вы используете Eclipse CDT IDE, импортируйте существующий проект из `IdlLinux_[ARCH]/src/IdlApp`

или

- Если вы используете GNU make, перейдите в `IdlLinux_[ARCH]/src/IdlApp/LinuxIdlApp` и выполните команду **make all**
- Скомпилируйте и запустите демонстрационный пример в используемой IDE

Компиляция библиотеки драйверов IDL

- Если вы используете Eclipse CDT IDE, импортируйте существующий проект из `IdlLinux_[ARCH]/src/Idl`

или

- Если вы используете GNU make, перейдите в `IdlLinux_[ARCH]/src/IdlApp/LinuxIdlApp` и выполните команду **make all**
- Скомпилируйте библиотеку драйверов в используемой IDE

IDL API

С помощью следующих шагов можно создать проект, определенный пользователем (вместо адаптации и настройки демонстрационного примера, как описано в разделе "6. Адаптация и настройка примера приложения" на стр. 19.

Заголовочные файлы IDL хранятся в папке `IdlLinux_[ARCH]/inc`.

- При создании пользовательского проекта включите папку `IdlLinux_[ARCH]/inc` и свяжите проект с файлами `libidlLinux.so` и `libidl115DriverLinux.so` (115 означает INpact PCIe)
- Откройте устройство функцией `ABCC_OpenController`, которая определена в заголовочном файле `IdlLinux_[ARCH]/inc/IDL.h`
  - Если функция выполнилась успешно, она вернёт дескриптор устройства
  - Дескриптор обычно является первым параметром всех, специфичных для драйвера, функций (например, `ABCC_StartDriver(IDL_CTRL_HDL hCtrl`, `UINT32 lMaxStartupTimeMs`)

6. Адаптация и настройка примера приложения

6.1. Сетевая идентификация

Если все настройки сети останутся отключёнными, продукт будет идентифицироваться как продукт HMS. На этом шаге, настраиваются свойства сетевой идентификации, чтобы соответствовать целевому продукту.

Связанные с идентификацией свойства для каждого включённого сетевого объекта – это параметры, которые должны быть установлены приложением. Они все влияют на то, как интерфейс идентифицируется в сети. Эти настройки хранятся в `abcc_adapt/abcc_identification.h`.

6.1.1. Хост-приложение – Сети

Аббревиатуры сетей

Сеть	Аббревиатура
DeviceNet	DEV
EtherCAT	ECT
Ethernet POWERLINK	EPL
EtherNet/IP	EIP
Profibus DP-V1	DPV1
PROFINET IO	PRT
Modbus TCP	EIT

- Чтобы активировать поддерживаемую сеть, установите соответствующий объект хост-приложения в файле `abcc_adapt/abcc_obj_cfg.h` в `TRUE`
- Чтобы активировать сеть установите `vendor_id` и `product_code` (см. следующий пример) в соответствии с используемым протоколом
- Делайте дальнейшую реализацию хост-приложения в папке `abcc_obj`, в которой каждый объект имеет свой собственный c- и h-файлы

Пример установки Vendor ID и Product Code

```
/*-----------------------------------------------------------------
** Ethernet Powerlink (0xE9)
**-----------------------------------------------------------------
*/
#if EPL_OBJ_ENABLE
/*
** Атрибут 1: Идентификатор производителя (UINT32 - 0x00000000-0xFFFFFFFF)
*/
#define EPL_IA_VENDOR_ID_ENABLE                     TRUE
#define EPL_IA_VENDOR_ID_VALUE                      0xFFFFFFFF
/*
** Атрибут 2: Тип кода продукта (UINT32 - 0x00000000-0xFFFFFFFF)
*/
#define EPL_IA_PRODUCT_CODE_ENABLE                  TRUE
#define EPL_IA_PRODUCT_CODE_VALUE                   0xFFFFFFFF
```

Также можно определить функцию вместо константы чтобы получить значение. Серийный номер является хорошим примером, для которого использование функции будет уместным. Ниже показан пример, в котором серийный номер при производстве продукта записывается в специальную область памяти, и в примере этот же серийный номер извлекается:

```
extern char* GetSerialNumberFromProductionArea(void);

#define PRT_IA_IM_SERIAL_NBR_ENABLE TRUE

#define PRT_IA_IM_SERIAL_NBR_VALUE GetSerialNumberFromProductionArea()
```

© HMS Industrial Networks AB
© 2018, Перевод DATAMICRO LLC

6.1.2. Хост-приложение – Другое

★ В abcc_adapt/abcc_obj_cfg.h определите все остальные объекты хост-приложения которые будут поддерживаться реализацией.

6.1.3. Объекты хост-приложения – Дополнительно

Файлы abcc_adapt/abcc_obj_cfg.h и abcc_adapt/abcc_identification.h содержат все свойства для всех поддерживаемых хост-объектов. Все специфичные для сети сервисы отключены по умолчанию, и, если они необходимы, их нужно реализовать в приложении.

Файл abcc_adapt/abcc_platform_cfg.h может быть использован для переопределения определений для объектов и свойств в файлах abcc_adapt/abcc_obj_cfg.h и abcc_adapt/abcc_identification.h.

Для переопределения добавьте необходимые определения в файл abcc_adapt/abcc_platform_cfg.h

6.2. ADI и отображение данных процесса

Следующие примеры отображения ADI, которые демонстрируют различные типы ADI, включены в пример приложения.

example_app/appl_speed.c
Мастер устанавливает эталонное значение скорости, и ведомый обновляет собственное соответствующее значение скорости равным эталонному значению скорости.

- ADI1 (SINT16, отображается как вход)
- ADI2 (SINT16, отображается как выход)
- ADI3 (BOOL, отображается как выход)
- ADI4 (SINT16, отображается как вход)
- ADI5 (UINT8, отображается как выход)

example_app/appl_adimap_simple16.c
Зацикленные друг на друга массивы из 32-х 16-битных слов.

- ADI1 (массив из 32 элементов типа UINT16)
- ADI2 (массив из 32 элементов типа UINT16)
- ADI отображаются в обоих направлениях.
- ADI1 и ADI2 зациклены друг на друга, так как обои ADI ссылаются на одну и ту же область памяти.
- Структуры и функции обратного вызова не используются.

example_app/appl_adimap_separate16.c
Пример использования функций обратного вызова get/set.

- ADI10 (массив из 32 элементов типа UINT16, отображается как выходные данные)
- ADI11 (массив из 32 элементов типа UINT16, отображается как входные данные)
- ADI12 (UINT16, не отображается на данные процесса)
- ADI10 и ADI11 отображаются на данные процесса в каждом направлении.
- Функция обратного вызова используется, когда сеть читает ADI11. Эта функция обратного вызова увеличит значение ADI12 на единицу.
- Функция обратного вызова используется, когда сеть записывает ADI10. Эта функция обратного вызова копирует ADI10 в ADI11.

© HMS Technology Center Ravensburg GmbH
© 2018, Перевод ООО "ДЭЙТАМИКРО"

Определение `ABCC_CFG_ADI_GET_SET_CALLBACK` должно быть активировано в файле `abcc_adapt/abcc_drv_cfg.h` (Linux: `inc/abcc_drv_cfg.h`), так как используются функции обратного вызова. *См. "4.2.2. Параметры ADI" на стр. 13*

example_app/appl_adimap_alltypes.c

Пример использования структурированных и битовых типов данных.

ADI	Описание
ADI20	UINT32 (отображается как выходные данные)
ADI21	UINT32 (отображается как входные данные)
ADI22	SINT32 (отображается как выходные данные)
ADI23	SINT32 (отображается как входные данные)
ADI24	UINT16 (отображается как выходные данные)
ADI25	UINT16 (отображается как входные данные)
ADI26	SINT16 (отображается как выходные данные)
ADI27	SINT16 (отображается как входные данные)
ADI28	BITS16 (отображается как выходные данные)
ADI29	BITS16 (отображается как входные данные)
ADI30	UINT8 (отображается как выходные данные)
ADI31	UINT8 (отображается как входные данные)
ADI32	SINT8 (отображается как выходные данные)
ADI33	SINT8 (отображается как входные данные)
ADI34	PAD8 (отображается как выходные данные, зарезервированная область, не содержит данных)
ADI35	PAD8 (отображается как входные данные, зарезервированная область, не содержит данных)
ADI36	BIT7 (отображается как выходные данные)
ADI37	BIT7 (отображается как входные данные)
ADI38	Struct (отображается как выходные данные)
ADI39	Struct (отображается как входные данные)

Определение `ABCC_CFG_STRUCT_DATA_TYPE` должно быть активировано в файле `abcc_adapt/abcc_drv_cfg.h` (Linux: `inc/abcc_drv_cfg.h`), так как используются структуры, *см. "4.2.2. Параметры ADI" на стр. 13*

Настройка отображения ADI

Только одно отображение может быть использовано в любой момент времени.

- Для того чтобы определить ADI создайте `AD_AdiEntryType` в списке элементов ADI (то есть экземпляры данных, которые будут использованы в реализации)
- Смотрите спецификацию всех параметров ADI в Списке элементов ADI, см. "Список элементов ADI" на стр. 22
- Отобразите ADI которые должны использоваться как данные процесса в списке `AD_DefaultMapType`, см. "Запись и чтение отображения данных процесса" на стр. 23

© HMS Industrial Networks AB
© 2018, Перевод DATAMICRO LLC

Список элементов ADI

Элемент ADI	Описание	
iInstance	Номер экземпляра ADI (1-65535). Значение 0 зарезервировано для Class	
pabName	Название ADI (строка символов, атрибут #1 экземпляра ADI). Если NULL, будет возвращено название нулевой длины	
bDataType	ABP_BOOL:	Булево
	ABP_SINT8:	Знаковое 8-битное целое
	ABP_SINT16:	Знаковое 16-битное целое
	ABP_SINT32:	Знаковое 32-битное целое
	ABP_UINT8:	Беззнаковое 8-битное целое
	ABP_UNIT16:	Беззнаковое 16-битное целое
	ABP_UINT32:	Беззнаковое 32-битное целое
	ABP_CHAR:	Символ
	ABP_ENUM:	Перечисление
	ABP_SINT64:	Знаковое 64-битное целое
	ABP_UINT64:	Беззнаковое 64-битное целое
	ABP_FLOAT:	Значение с плавающей точкой (32-битное)
	ABP_OCTET	Неопределённые 8 бит данных (Только в серии 40)
	ABP_BITS8	8-битное битовое поле (Только в серии 40)
	ABP_BITS16	16-битное битовое поле (Только в серии 40)
	ABP_BITS32	32-битное битовое поле (Только в серии 40)
	ABP_BIT1	1-битное битовое поле (Только в серии 40)
	ABP_BIT2	2-битное битовое поле (Только в серии 40)
	...	...
	ABP_BIT7	7-битное битовое поле (Только в серии 40)
	ABP_PAD0	0-битный заполнитель (Только в серии 40)
	ABP_PAD1	1-битный заполнитель (Только в серии 40)
	...	...
	ABP_PAD16	16-битный заполнитель (Только в серии 40)
bNumOfElements	Для массивов: количество элементов, имеющих тип данных, указанный в bDataType. Для структурированных типов данных: количество элементов в структуре	
bDesc	Описатель записи. Битовые значения, в соответствии со следующими конфигурациями: ★ `ABP_APPD_DESCR_GET_ACCESS`: Включён сервис получения значения атрибута ★ `ABP_APPD_DESCR_SET_ACCESS`: Включён сервис установки значения атрибута ★ `ABP_APPD_DESCR_MAPPABLE_WRITE_PD`: Включён сервис отображения значения атрибута ★ `ABP_APPD_DESCR_MAPPABLE_READ_PD`: Включён сервис отображения значения атрибута Дескрипторы могут объединяться через побитовое ИЛИ. В примере, `ALL_ACCESS` равен побитовому ИЛИ всех перечисленных выше значений. Примечание: Игнорируется для структурированных типов данных	
pxValuePtr	Указатель на переменную локального значения. Тип зависит от bDataType. Примечание: Игнорируется для структурированных типов данных	

Элемент ADI	Описание
pxValuePropPtr	Указатель на структуру свойств локального значения, если NULL, никакие свойства не применяются (макс./мин./по умолчанию). Тип зависит от bDataType. Примечание: Игнорируется для структурированных типов данных
psStruct	Указатель на AD_StructDataType. Установите в NULL для неструктурированных типов данных. Поле активируется путём определения ABCC_CFG_STRUCT_DATA_TYPE. (Опционально, Только в серии 40)
pnGetAdiValue	Указатель на функцию типа ABCC_GetAdiValueFuncType, вызываемую для получения значения ADI. (Опционально)
pnSetAdivalue	Указатель на функцию типа ABCC_SetAdiValueFuncType, вызываемую для установки значения ADI. (Опционально)

См. пример использования в abcc_drv/inc/abcc_ad_if.h.

Запись и чтение отображения данных процесса

Элемент отображения данных	Описание
iInstance	Номер ADI, который отображается (см. "Список элементов ADI" на стр. 22)
eDir	Направление отображения. Установите в PD_END_MAP для индикации окончания списка отображения по умолчанию
bNumElem	Количество отображаемых элементов. Может быть > 1 только для массивов или структур. AD_DEFAULT_MAP_ALL_ELEM означает что должны быть отображены все элементы. Если экземпляр == AD_MAP_PAD_ADI, bNumElem содержит количество бит для заполнения
bElemStartIndex	Индекс первого элемента в массиве или структуре. Если ADI не является массивом или структурой, установите в 0

Отображения выполняются в том порядке, в котором они появляются в сети.

Последовательность отображений заканчивается значением AD_DEFAULT_MAP_END_ENTRY, которое ДОЛЖНО присутствовать в конце списка. Во время установки, драйвер запросит эту информацию через вызов функции ABCC_CbfAdiMappingReq()

Пример

```
/*------------------------------------------------------------------------------------
** Отображение всех adi в обоих направлениях
**-----------------------------------------------------------------------------------
** 1. Экземпляр AD | 2. Направление | 3. Количество элементов | 4. Начальный индекс |
**-----------------------------------------------------------------------------------
*/
const AD_DefaultMapType APPL_asAdObjDefaultMap[] =
{
   { 1, PD_WRITE, AD_DEFAULT_MAP_ALL_ELEM, 0 },
   { 2, PD_READ, AD_DEFAULT_MAP_ALL_ELEM, 0 },
   { AD_DEFAULT_MAP_END_ENTRY }
};
```

Пример может быть найден в \example_app (Linux: src/IdlApp/example_app/appladimap.simple16.c).

Другие примеры могут быть найдены в abcc_drv/inc/abcc_ad_if.h (Linux: inc/abcc_ad_if.h).

© HMS Industrial Networks AB
© 2018, Перевод DATAMICRO LLC

6.3. Функции обратного вызова обработки данных процесса

Необходимо реализовать две функции обратного вызова, связанные с обновлением данных процесса. Пример доступен в `example_app/appl_abcc_handler.c`.

★ `ABCC_CbfUpdateWriteProcessData()`: Обновляет текущие данные процесса для записи.

Скопируйте данные в буфер до того, как вернуть управление из функции.

★ `ABCC_CbfNewReadPd()`: Вызывается при получении из сети новых данных процесса.

Скопируете данные процесса в ADI приложения до того, как вернуть управление из функции.

Как будет показано ниже в коде примера, они обе вызывают сервис объекта данных приложения для обновления информации. Эти сервисы работают, в общем случае, для отображения любых данных процесса, но они работают медленно, так как для общего случая необходимо предусмотреть все возможности. Для лучшей производительности, пожалуйста рассмотрите вариант написания функций обновления, специфичных для приложения.

Windows

```
void ABCC_CbfNewReadPd( void* pxReadPd )
{
   /*
   ** AD_UpdatePdReadData функция для общего случая, обновляющая все ADI в соответствии
   ** с текущим отображением.
   ** Если отображение ADI фиксировано, есть возможность сделать это более оптимальным путём,
   ** например используя memcpy.
   */
   AD_UpdatePdReadData( pxReadPd );
}

BOOL ABCC_CbfUpdateWriteProcessData( void* pxWritePd )
{
   /*
   ** AD_UpdatePdWriteData функция для общего случая, обновляющая все ADI в соответствии
   ** с текущим отображением.
   ** Если отображение ADI фиксировано, есть возможность сделать это более оптимальным путём,
   ** например используя memcpy.
   */
   return( AD_UpdatePdWriteData( pxWritePd ) );
}
```

Linux

```
BOOL ABCC_CbfUpdateWriteProcessData( IDL_CTRL_HDL hCtrl,
void* pxWritePd )
{
/*
   ** AD_UpdatePdWriteData функция для общего случая, обновляющая все ADI в соответствии
   ** с текущим отображением.
   ** Если отображение ADI фиксировано, есть возможность сделать это более оптимальным путём,
   ** например используя memcpy.
*/
return( AD_UpdatePdWriteData( pxWritePd ) );
}
...
void ABCC_CbfNewReadPd( IDL_CTRL_HDL hCtrl,void* pxReadPd )
{
/*
   ** AD_UpdatePdReadData функция для общего случая, обновляющая все ADI в соответствии
   ** с текущим отображением.
   ** Если отображение ADI фиксировано, есть возможность сделать это более оптимальным путём,
   ** например используя memcpy.
*/
AD_UpdatePdReadData( pxReadPd );
```

© HMS Technology Center Ravensburg GmbH
© 2018, Перевод ООО "ДЭЙТАМИКРО"

6.4. Обработка событий

Приведённые ниже настройки активируют прерывания при чтении сообщений и чтении данных процесса. Но только в Linux функции обратного вызова при чтении данных процесса будут вызваны в контексте прерывания напрямую драйвером. Событие чтения сообщения будет перенаправлено приложению вызовом функции ABCC_CbfEvent().

```
#define ABCC_CFG_INT_ENABLE_MASK_PAR (ABP_INTMASK_RDPDIEN | ABP_INTMASK_RDMSGIEN)
#define ABCC_CFG_HANDLE_INT_IN_ISR_MASK (ABP_INTMASK_RDPDIEN)
```

Ниже показаны примеры того, как обработчик события для обработки события может запустить задачу в Windows и Linux.

Windows

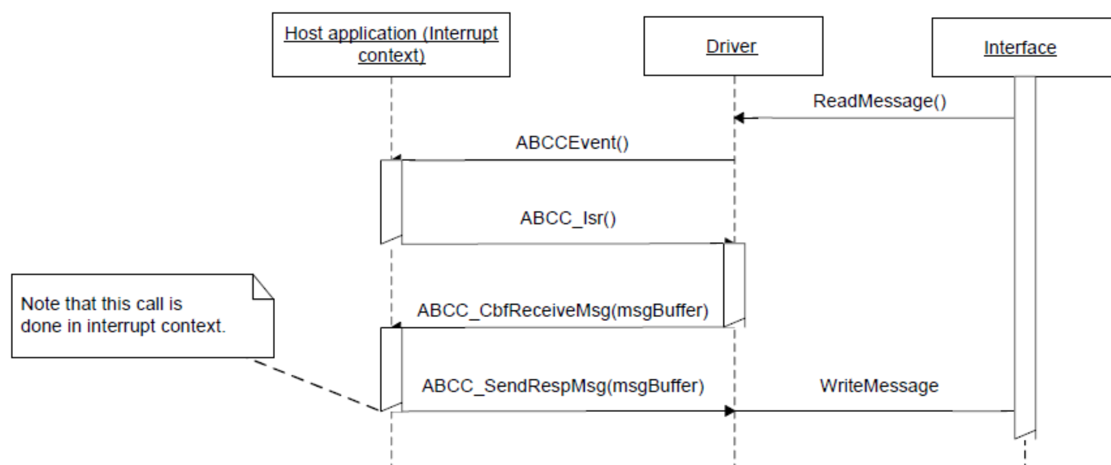
```
void ABCC_CbfEvent( UINT16 iEvents )
{
   if( iEvents & ABCC_EVENT_RDMSGI )
   {
      ABCC_fRdMsgEvent = TRUE;
   }
}
```

Приведённый выше код демонстрирует как задача (показано ниже) может быть запущена обработчиком события драйвера.

Этот код показывает задачу, обрабатывающую события получения сообщений.

```
volatile BOOL ABCC_fRdMsgEvent = FALSE;
void HandleEvents( void )
{
   ABCC_fRdMsgEvent = FALSE;
   while( 1 )
   {
      if( ABCC_fRdMsgEvent )
      {
         ABCC_fRdMsgEvent = FALSE;
         ABCC_TriggerReceiveMessage();
      }
   }
}
```

Обработка событий

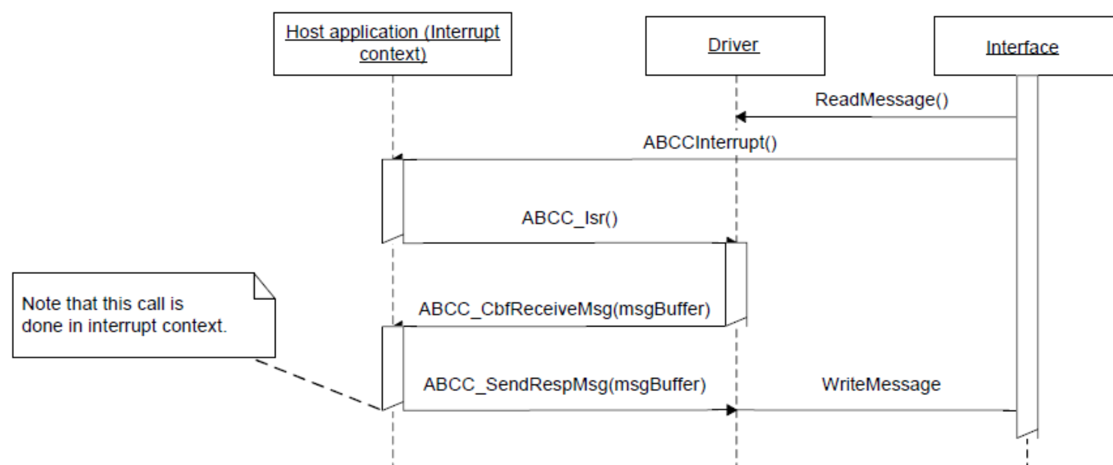


Linux

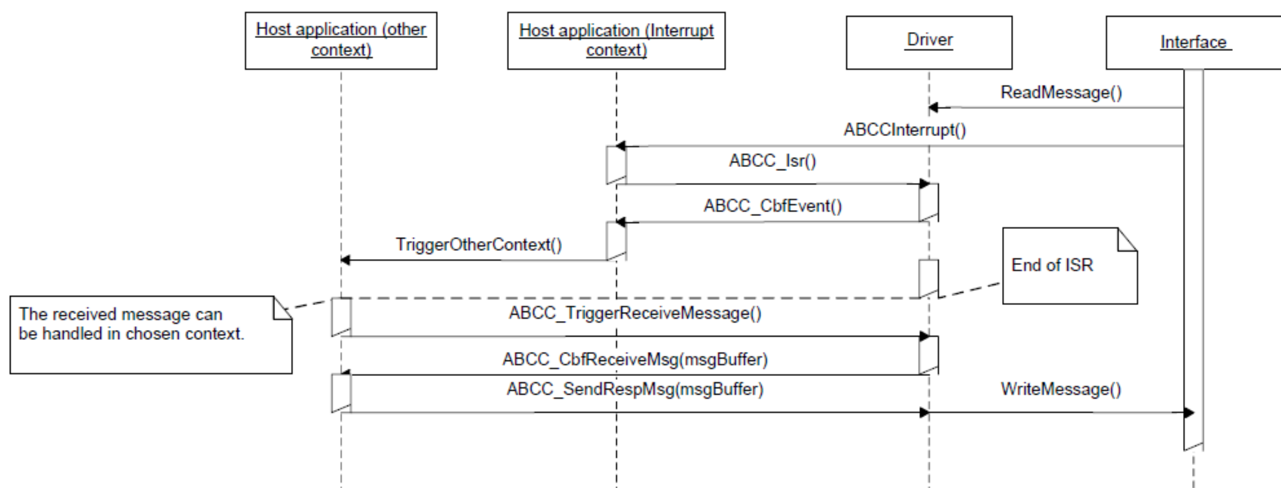
```
void ABCC_CbfEvent( IDL_CTRL_HDL hCtrl, UINT16 iEvents )
{
   if( iEvents & ABCC_ISR_EVENT_RDMSG )
   {
      ABCC_fRdMsgEvent = TRUE;
   }
}

volatile BOOL ABCC_fRdMsgEvent = FALSE;
void Task( IDL_CTRL_HDL hCtrl )
{
   ABCC_fRdMsgEvent = FALSE;
   while( 1 )
   {
      if( ABCC_fRdMsgEvent )
      {
         ABCC_fRdMsgEvent = FALSE;
         ABCC_TriggerReceiveMessage( hCtrl );
      }
   }
}
```

Обработка событий в контексте прерывания



Обработка событий с помощью функции обратного вызова ABCC_CbfEvent()



6.5. Обработка сообщений

Интерфейсные функции обработки сообщений описаны в abcc.h (Windows и Linux).

При отправке командного сообщения, пользователь должен использовать функцию ABCC_GetCmdMsgBuffer() для получения буфера памяти сообщения. При получении ответа, пользователь должен обработать или скопировать необходимые данные из буфера ответа в контекст функции-обработчика ответа.

Функция ABCC_GetCmdMsgBuffer() может вернуть указатель равный NULL, если буфер памяти более недоступен. Это обязанность пользователя отправить сообщение позже, либо интерпретировать эту ситуацию как критическую ошибку.

6.5.1. Дополнительные функции в Windows

Стартовое приложение Windows реализует некоторые дополнительные функции, упрощающие обработку сообщений. Реализация находится в \example_app\communication\appl_com.c.

Эти вспомогательные функции уже использовались при инициализации модуля Anybus в соответствии с описанием в главе 11 Руководства по разработке программного обеспечения Anybus CompactCom 40. См. файл abcc_setup.c для изучения использования этих функций.

SetAttribute

Функция для установки содержимого атрибута Anybus.

Параметр	Описание
bObject	Объект(01 == Anybus, 02 == Диагностический, 03 == сетевой)
wInstance	Экземпляр объекта
bAttribute	Атрибут экземпляра
hEvent	Событие для оповещения
wWrite	Размер данных для записи
pbData	Указатель в памяти на данные для записи
pwWritten	Указатель на переменную типа WORD, в которую будет сохранено количество записанных данных
dwTimeout	Время ожидания ответа

Возможные ответы	Описание
TRUE	Функция выполнена успешно
FALSE	Произошла ошибка

GetAttribute

Функция получения содержимого атрибута Anybus.

Параметр	Описание
bObject	Объект(01 == Anybus, 02 == Диагностический, 03 == сетевой)
wInstance	Экземпляр объекта
bAttribute	Атрибут экземпляра
hEvent	Событие для оповещения
wSize	Размер данных для чтения
pbData	Указатель в памяти для хранения прочитанных данных
pwReaden	Указатель на переменную типа WORD, в которую будет сохранено количество прочитанных данных
dwTimeout	Время ожидания ответа

© HMS Industrial Networks AB
© 2018, Перевод DATAMICRO LLC

Возможные ответы	Описание
TRUE	Функция выполнена успешно
FALSE	Произошла ошибка

6.5.2. Пример 1: Отправка команды и получение ответа

При отправке команды драйвер соединит идентификатор источника с функцией обработчика ответа, в данном случае с `appl_HandleResp()`.

Функция `appl_HandleResp()` будет вызвана драйвером при получении ответа с соответствующим идентификатором источника.

Обратите внимание, что нет необходимости освобождать буфер принятого сообщения, это делается автоматически в драйвере, после возврата из `appl_HandleResp()`.

Windows

```
void appl_HandleResp( ABP_MsgType* psMsg )
{
   HandleResponse(psMsg);
}
```

Linux

```
void appl_HandleResp( IDL_CTRL_HDL hCtrl, ABP_MsgType* psMsg )
{
   HandleResponse( hCtrl, psMsg );
}
```

Комментарий 1

```
psMsg = ABCC_GetCmdMsgBuffer();

if( psMsg != NULL )
{

   ABCC_GetAttribute( psMsg, ABP_OBJ_NUM_ANB, 1, ABP_ANB_IA_EXCEPTION, ABCC_GetNewSourceId() );
   if( ABCC_SendCmdMsg( psMsg, msgRespHandler ) != ABCC_EC_NO_ERROR )
   {
      APPL_UnexpectedError();
   }
}
```

Комментарий 2

```
static void msgRespHandler( ABP_MsgType* psMsg )
{
   if( ABCC_VerifyMessage( psMsg ) != ABCC_EC_NO_ERROR )
   {
      APPL_UnexpectedError();
      return;
   }

   /*
   ** Обработка данных ответа
   */
}
```

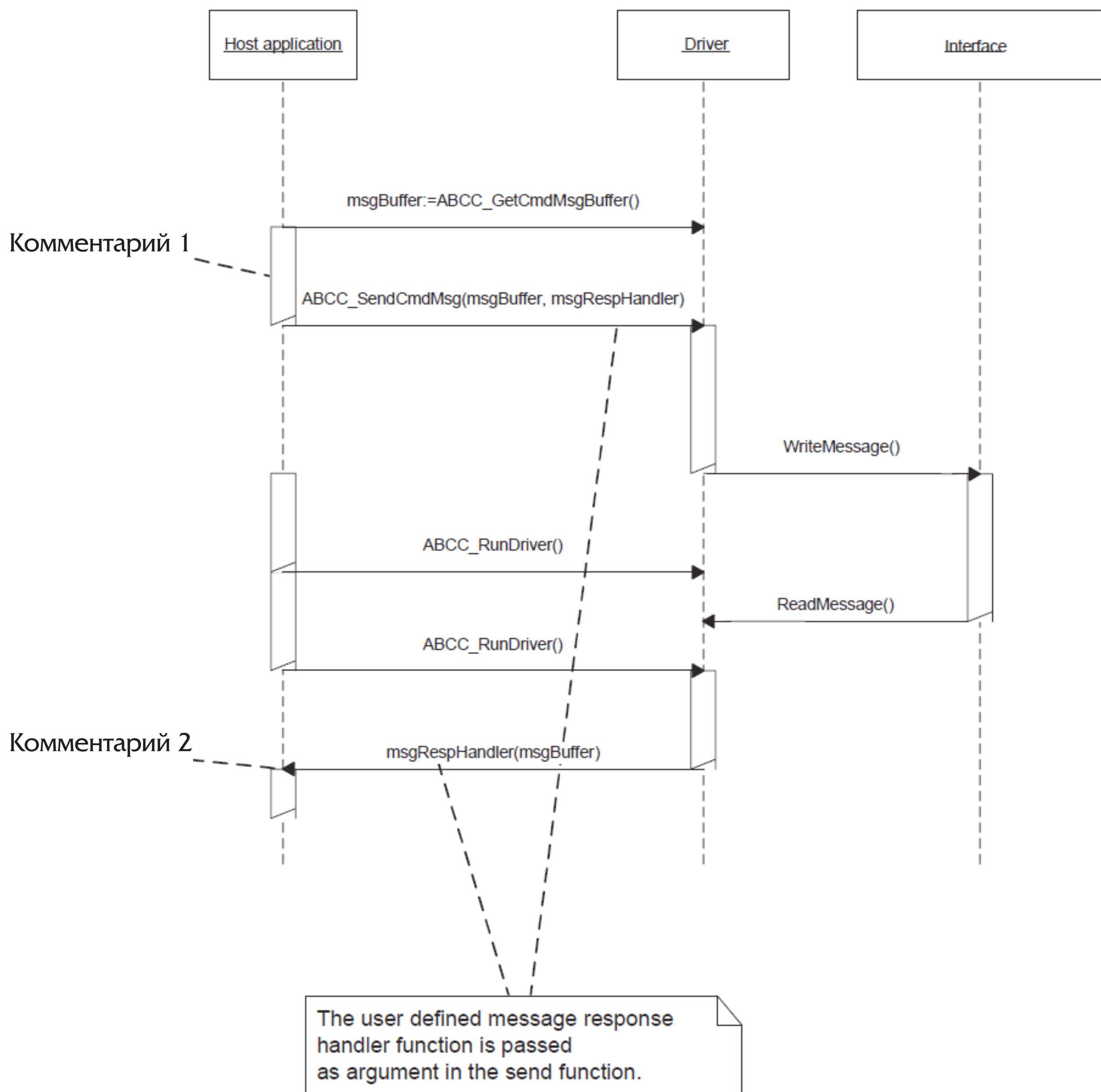


Рис. 6-1. Отправка команды на интерфейс

6.5.3. Пример 2: Получение команды и отправка ответа

Windows

Буфер принятой команды повторно используется для ответа.

```
void appl_ProcessCmdMsg( ABP_MsgType* psNewMessage )
{
   /* Повторно использовать буфер команды для ответа */
   ABP_SetMsgResponse( psNewMessage, ABP_UINT8_SIZEOF );
   eErr = ABCC_SendRespMsg( psNewMessage );
}
```

Linux

```
void appl_ProcessCmdMsg( IDL_CTRL_HDL hCtrl, ABP_MsgType* psNewMessage )
{
   /* Повторно использовать буфер команды для ответа */
   ABP_SetMsgResponse( psNewMessage, ABP_UINT8_SIZEOF )
   eErr = ABCC_SendRespMsg( hCtrl, psNewMessage );
}
```

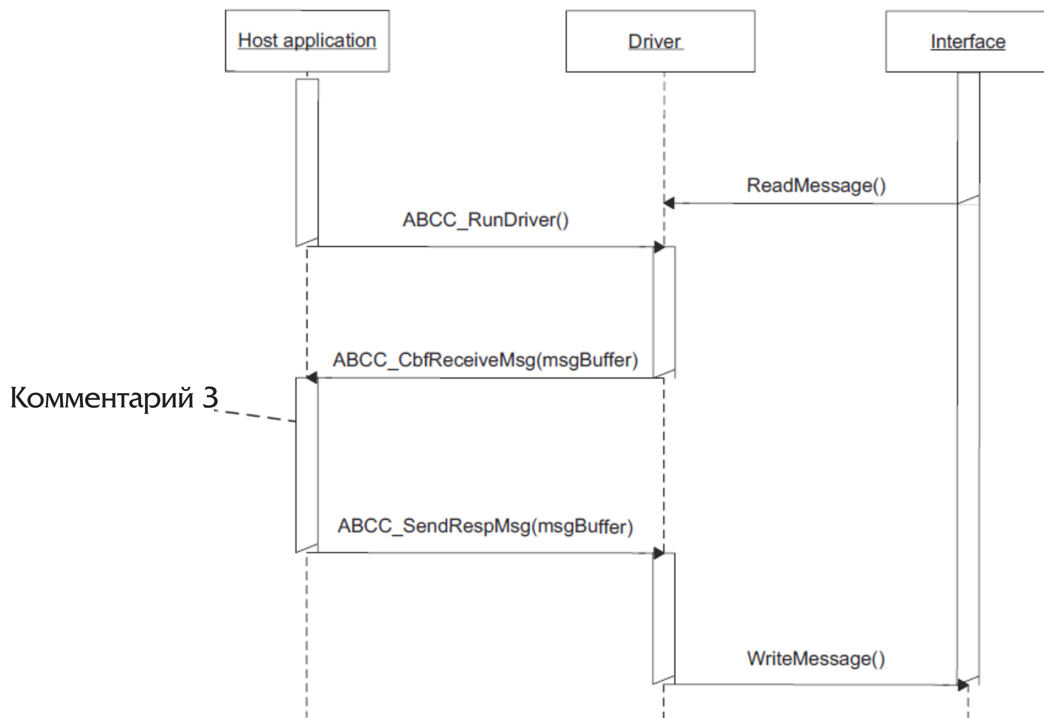


Рис. 6-2. Получение команды от интерфейса

Комментарий 3

```
void ABCC_CbfReceiveMsg( ABP_MsgType* msgBuffer )
{

   /*
   ** Обработать сообщение команды
   */

   /*
   ** Повторно использовать буфер команды для ответа
   */
   ABP_SetMsgResponse( msgBuffer , ABP_UINT8_SIZEOF );
   eErr = ABCC_SendRespMsg( msgBuffer );
}
```

Драйвер использует неблокирующую обработку сообщений. Это означает, что необходимо использовать конечный автомат для отслеживания команд и ответов.

Два примера конечного автомата, которые могут использоваться как шаблоны, можно найти в файле `example_app/appl_abcc_handler.c`.

Пример 1: Когда вызывается функция `ABCC_CbfUserInitReq()`, IP адрес или адрес узла устанавливается перед вызовом функции `ABCC_UserInitComplete()`.

Пример 2: Когда интерфейс показывает состояние исключения, считываются коды исключения.

A Обзор ПО

A.1 Файлы и папки

Папки	Описание
`$\abcc_abp`	Содержит все определения объекта и коммуникационного протокола. Файлы обновляются, когда доступны новые версии ПО
`$\abcc_adapt`	Содержит всю адаптацию и конфигурацию. Файлы используются для адаптации драйвера и кода примеров к системному окружению
`$\abcc_drv\inc`	.h файлы, предназначенные для использования в приложении. Содержит конфигурационные файлы драйвера для приложения и для зависимых от системы частей драйвера
`$\abcc_drv\src`	Реализация драйвера
`$\abcc_obj`	Содержит всю реализацию хост-объекта. Файлы могут быть модифицированы для оптимизации и/или добавления новой функциональности
`$\example_app`	Пример приложения. При необходимости файлы могут быть модифицированы для оптимизации и/или добавления новой функциональности

A.2 Файлы верхнего уровня

Папки	Описание
`$\main.c`	Главный файл примера приложения
`$\abcc_versions.h`	Содержит определения версии для кода примера, драйвера и ABP

A.3 Интерфейс драйвера

Имя файла	Описание
`\abcc_drv\inc\abcc.h`	Открытый интерфейс для драйвера
`\abcc_drv\inc\abcc_ad_if.h`	Определения типов для отображения ADI
`\abcc_drv\inc\abcc_cfg.h`	Конфигурационные параметры драйвера
`\abcc_drv\inc\abcc_sys_adapt.h`	Интерфейс для зависящих от целевого устройства функций, общих для всех режимов функционирования

A.4 Внутренние файлы драйвера (только для чтения)

Содержимое файлов в папке `/abcc_drv/src` не должно меняться.

Имя файла	Описание
`\abcc_drv\src\abcc_handler.h` `\abcc_drv\src\abcc_handler.c`	Реализация обработчиков, включая независящие от режима функционирования части обработчиков
Windows: `abcc_driver.c`	Реализация доступа драйвера

© HMS Industrial Networks AB
© 2018, Перевод DATAMICRO LLC

A.5 Файлы адаптации системы

Файл	Описание
\abcc_adapt\abcc_drv_cfg.h	Пользовательская конфигурация драйвера. Параметры конфигурации задокументированы в открытом интерфейсе драйвера abcc_cfg.h
\abcc_adapt\abcc_identification.h	Пользовательская конфигурация, предназначенная для установки идентификационных параметров интерфейса
\abcc_adapt\abcc_obj_cfg.h	Пользовательская конфигурация реализации объекта
\abcc_adapt\abcc_sw_port.c	Макросы и функции, необходимые драйверу и реализации объекта
\abcc_adapt\abcc_sw_port.h	Макросы и функции, необходимые драйверу и реализации объекта. Описание макросов находится в \abcc_drv\inc\abcc_port.h
\abcc_adapt\abcc_td.h	Определение типов данных

B Описание API

Слой API определяет интерфейс к драйверу, общий для всех сетевых приложений. Интерфейс находится в abcc.h.

B.1 Функции API

Функция	Описание
Windows: ABCC_AccessDriver()	Открывает драйвер и устанавливает соединение с INpact
Linux: ABCC_OPEN(), ABCC_CLOSE()	Открывает и закрывает драйвер
ABCC_StartDriver()	Инициализирует драйвер, включает прерывания и устанавливает режим работы. В результате вызова этой функции может быть запущена система отсчёта времени. **Внимание!** Эта функция НЕ освободит сброс интерфейса
ABCC_IsReadyforCommunication()	Эта функция должна опрашиваться после вызова ABCC_StartDriver(), пока она не вернёт значение TRUE. Это показывает что интерфейс готов к коммуникации и что запускается последовательность установки
ABCC_ShutdownDriver()	Останавливает драйвер и переводит его в состояние SHUTDOWN
ABCC_HWReset()	Аппаратный сброс интерфейса. ABCC_ShutdownDriver() вызывается из этой функции. **Внимание!** Эта функция только лишь установит на контакте сброса низкий уровень. Тот кто её вызывает, должен обеспечить достаточную длительность времени сброса (время между вызовами функций ABCC_HWReset() и ABCC_HWReleaseReset())
ABCC_HWReleaseReset()	Освобождает сброс интерфейса
ABCC_RunTimerSystem()	Обрабатывает все таймера для драйвера. Рекомендуется вызывать эту функцию регулярно из обработчика прерывания таймера. Без этой функции не будут работать тайм-ауты и сторожевой таймер
ABCC_RunDriver()	Управляет механизмом отправки и приёма в драйвере. Эта основная процедура должна вызываться циклически во время опроса. TRUE: Драйвер запущен и готов к коммуникации. FALSE: Драйвер остановлен или не запущен
ABCC_UserInitComplete()	Эта функция должна вызываться приложением при получении последнего ответа от специфичной для пользователя установки. Это завершит последовательность установки и будет отправлено ABCC_SETUP_COMPLETE
ABCC_SendCmdMsg()	Посылает командное сообщение в интерфейс
ABCC_SendRespMsg()	Посылает сообщение ответа в интерфейс
ABCC_SendRemapRespMsg()	Посылает ответ на переотображение в интерфейс
ABCC_SetAppStatus()	Устанавливает текущее состояние приложения, в соответствии с ABP_AppStatusType в abp.h
ABCC_GetCmdMsgBuffer()	Выделяет буфер командного сообщения
ABCC_ReturnMsgBuffer()	Освобождает буфер сообщения
ABCC_TakeMsgBufferOwnership()	Отдаёт владение буфером сообщения
ABCC_ModCap()	Вычитывает возможности интерфейса
ABCC_LedStatus()	Вычитывает состояние светодиодной индикации
ABCC_AnbState()	Вычитывает текущее состояние Anybus
ABCC_HwRelease()	Освобождает оборудование

B.2 Функции, связанные с API

Функция	Описание
ABCC_ISR()	Эта функция должна вызываться из обработчика прерывания для подтверждения и обработки полученных событий (запускается выводом IRQ на прикладном интерфейсе)
ABCC_TriggerRdPdUpdate()	Запускает чтение RdPd
ABCC_TriggerReceiveMessage()	Запускает чтение полученных сообщений
ABCC_TriggerWrPdUpdate()	Показывает что новые данные процесса получены от приложения и будут отправлены интерфейсу
ABCC_TriggerAnbStatusUpdate()	Проверяет изменение состояния Anybus
ABCC_TriggerTransmitMessage()	Проверяет очередь отправки

B.3 Функции обратного вызова API

Все эти функции должны быть реализованы в приложении.

Функция	Описание
ABCC_CbfAdiMappingReq()	Функция вызывается, когда драйвер собирается начать автоматическое отображение данных процесса. Она возвращает информацию об отображении для PD чтения и записи
ABCC_CbfUserInitReq()	Функция вызывается для запуска специфичной для пользователя установки во время установки состояния интерфейса
ABCC_CbfUpdateWriteProcessData()	Обновляет текущие данные процесса записи. Данные должны быть скопированы в буфер до возврата из функции
ABCC_CbfNewReadPd()	Вызывается при получении новых данных процесса. Данные процесса необходимо скопировать в ADI до возврата из функции
ABCC_CbfReceiveMsg()	От интерфейса было получено сообщение
ABCC_CbfWdTimeout()	Функция вызывается при потере коммуникации с интерфейсом
ABCC_CbfRemapDone()	Эта функция обратного вызова вызывается при успешной отправке интерфейсу ответа REMAP
ABCC_CbfAnbStatusChanged()	Эта функция обратного вызова вызывается когда интерфейс меняет состояние, например когда изменяется состояние Anybus или состояние наблюдения
ABCC_CbfEvent()	Вызывается для необработанных событий. Необработанные события – это события активированные в ABCC_USER_INT_ENABLE_MASK, но отсутствующие в ABCC_USER_HANDLE_IN_ABCC_ISR_MASK
ABCC_CbfSync_Isr()	Если поддерживается события синхронизации, эта функция будет вызываться при получении события синхронизации

B.4 Вспомогательные функции

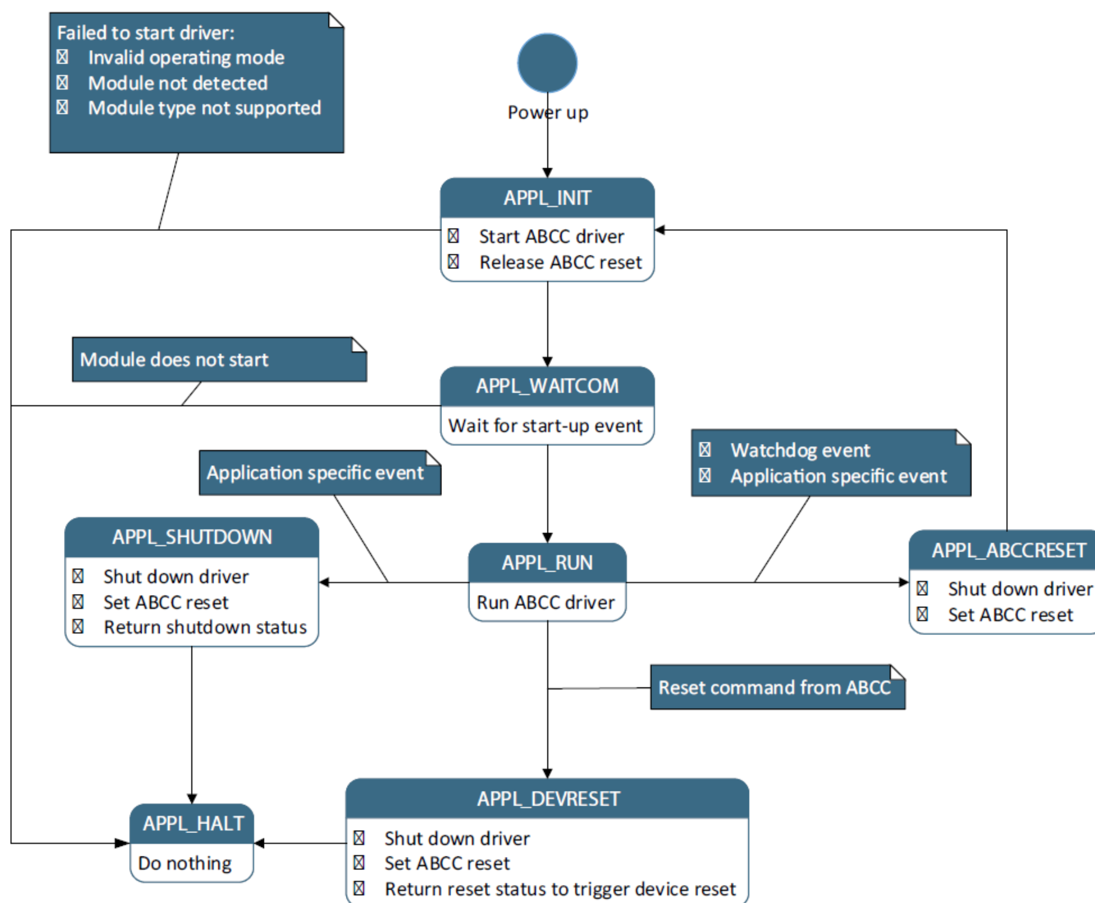
Функция	Описание
ABCC_NetworkType()	Возвращает тип сети
ABCC_ModuleType()	Возвращает тип интерфейса
ABCC_DataFormatType()	Возвращает порядок байтов сети
ABCC_ParameterSupport()	Проверяет, поддерживается ли заданный параметр
ABCC_GetAttribute()	Заполняет сообщение параметрами, для получения атрибута
ABCC_SetByteAttribute()	Заполняет сообщение параметрами, для установки атрибута
ABCC_VerifyMessage()	Проверяет сообщение ответа
ABCC_GetDataTypeSize()	Возвращает размер типа данных ABP

C Автомат состояний хост-приложения

Поток управления в коде примера приложения управляется автоматом состояний, показанным на блок-схеме ниже.

Функция `APPL_HandleAbcc()`, вызываемая циклически из основного цикла, реализует автомат состояний и отвечает за выполнение различных задач при каждом состоянии.

При первом вызове `APPL_HandleAbcc()` происходит вход в состояние `APPL_INIT`.



APPL_INIT

- Проверяет, определён ли интерфейс
- Инициализируется объект данных приложения при помощи необходимого отображения ADI
- Вызывается функция `ABCC_StartDriver()` для инициализации драйвера
- Вызывается функция `ABCC_HwReleaseReset()` для освобождения сброса интерфейса
- Устанавливает состояние `APPL_WAITCOM`

APPL_WAITCOM

- Ожидает от интерфейса сигнала о том что интерфейс готов к коммуникации
- Устанавливает состояние `APPL_RUN`

APPL_RUN

- `ABCC_RunDriver()`
  - Вызывается для запуска драйвера
  - Функции обратного вызова вызываются для отдельных событий

- Все функции обратного вызова, используемые драйвером, именуются следующим образом: `ABCC_Cbf<x>()`
- Необходимые функции обратного вызова реализованы в `appl_abcc_handler.c`

Во время запуска следующие события генерируются драйвером (в приведённой последовательности):

- `ABCC_CbfStateChanged()`
  - Вызывается, когда интерфейс входит в состояние `ABP_ANB_STATE_SETUP`
  - При необходимости, установите точку останова или используйте функцию отладки для индикации изменения состояния
- `ABCC_CbfAdiMappingReq()`
  - Вызывается когда интерфейс готов отправить команду отображения по умолчанию
  - Код общего примера запрашивает у объекта данных приложения настроенное отображение по умолчанию
- `ABCC_CbfUserInitReq()`
  - Вызывается когда у приложения есть возможность отправлять команды настройки в интерфейс или чтения информации из интерфейса
  - В коде примера, функция запускает автомат состояний пользовательской инициализации для начала отправки последовательности команд интерфейсу
  - При получении ответа на последнее сообщение вызывается функция `ABCC_UserInitComplete()` для оповещения драйвера об окончании пользовательской последовательности инициализации. В результате этого драйвер посылает команду `SETUP_COMPLETE` интерфейсу
  - Если пользовательская инициализация не требуется, функция `ABCC_UserInitComplete()` может быть вызвана напрямую из `ABCC_CbfUserInitReq()`
- `ABP_ANB_STATE_NW_INIT`
  - Когда установка закончится, интерфейс переходит в это состояние
  - Вызывается `ABCC_CbfStateChanged()`
  - Отправляются команды из интерфейса в объекты хост-приложения
  - Все полученные команды обрабатываются в `ABCC_CbfReceiveMsg()`
  - Ответы на команды зависят от того, какие хост-объекты реализованы, и от настроек, сделанных в файлах `abcc_identification.h` и `abcc_obj_cfg.h`
  - При необходимости, установите точку останова в `ABCC_CfgReceiveMsg()` для отображения отсылаемых команд, и того как они обрабатываются
- `ABP_ANB_STATE_WAIT_PROCESS`
  - При завершении инициализации сети интерфейс переходит в это состояние
  - `ABCC_CbgStateChanged()` вызывается драйвером
  - Можно установить соединение для ввода-вывода данных по сети
- `ABP_ANB_STATE_PROCESS_ACTIVE`
  - При установке соединения для ввода-вывода данных, интерфейс переходит в это состояние (или, в некоторых сетях, в состояние `ABP_ANB_STATE_IDLE`)
  - При получении данных процесса от интерфейса, вызывается функция `ABCC_CbfNewReadPd()`
  - Код примера перенаправляет данные объекту данных приложения вызовом `AD_UpdatePdReadData()`, для обновления данных приложения
  - Вызывается `ABCC_TriggerWrPdUpdate()` для обновления записываемых данных процесса, так как код примера только зацикливает данные
  - Функция `ABCC_TriggerWrPdUpdate()` вызывает `ABCC_CbfUpdateWriteProcessData()`, которая вызывается каждый раз, когда драйвер готов к отправке новых данных процесса. `ABCC_TriggerWrPdUpdate()`

должна вызываться всегда при доступности обновлённых данных процесса

- ABP_ANB_STATE_EXCEPTION
  - Причиной исключения может быть чтение из интерфейса путём активации автомата состояний чтения исключений
  - RunExceptionSM() вызывается из состояния APPL_RUN когда интерфейс находится в этом состоянии
- APPL_Reset()
  - Вызывается для инициализации перезапуска интерфейса
  - Возникает если хост-объект приложения получает запрос на сброс от интерфейса. Автомат состояний обработчика переходит в состояние APPL_ABCCRESET
- APPL_RestartAbcc()
  - Используется для инициализации перезапуска интерфейса, по аналогии с APPL_Reset()
  - Если вызывается, то автомат состояний обработчика переходит в состояние APPL_ABCCRESET. (Сейчас эта функция не используется в коде примера. Она может использоваться вместо APPL_Reset(), так как не производит включение-выключение питания)
- APPL_Shutdown()
  - Вызывается для инициализации завершения работы драйвера

APPL_SHUTDOWN

- ABCC_HWReset() вызывается для сброса интерфейса
- Изменяет состояние на APPL_HALT

APPL_ABCCRESET

- ABCC_HWReset() вызывается для сброса интерфейса
- Изменяет состояние на APPL_INIT

APPL_DEVRESET

Значение, возвращённое в основной цикл, (через вызов функции из APPL_AbccHandler()) показывает, что интерфейс должен быть сброшен.

- ABCC_HWReset() вызывается для сброса интерфейса
- Изменяет состояние на APPL_HALT

APPL_HALT

- Не выполняет никаких действий