Примеры использования Lpcie SDK для работы с модулями E16, E-502 и L-502

Таблица 1. Ревизии теккущего документа

1.0.0	20.05.2025	Первая версия данного документа

Оглавление

Глава 1

Введение

Настоящий документ содержит практические примеры работы с библиотекой L-502/E-502/E16 SDK, предназначенной для разработки программного обеспечения для работы с модулями E16, E-502 и L-502.

Цель документа — продемонстрировать базовые сценарии использования SDK на нескольких языках программирования (C/C++, C#, Python) и помочь разработчикам быстро приступить к работе с оборудованием.

Важные уточнения:

1. Документ не заменяет официальную документацию — за полным описанием функций и API обращайтесь к:
   - «Руководству программиста L-502/E-502/E16»
   - «Руководству пользователя модулей E16[1]/E-502/L-502
2. Примеры носят ознакомительный характер и должны быть расширены под конкретные задачи.

Данные примеры будут полезны инженерам, исследователям и разработчикам, впервые работающим с продуктами фирмы L-Card.

[1] Для E16 руководства пользователя еще нет, можно воспользоваться руководством для E-502

Глава 2

Интеграция библиотеки и подключение к модулю

2.1. Добавление SDK в проект

Перед началом работы с модулями нужно добавить необходимые динамические библиотеки (`.dll` для Windows или `.so` для Linux) из SDK в проект. Всего предоставляются три библиотеки:

- **x502api** – содержит общие функции для обоих модулей. Должна включаться в любой проект, работающий с одним из модулей, за исключением проектов, написанных только для L-502 до появления библиотеки x502api
- **l502api** – содержит специфические функции для модуля L-502, а также функции, оставленные для совместимости с проектами, написанными до появления x502api.
- **e502api** – содержит специфические функции для модулей E-502 и E16.

C/C++:

Для добавления библиотек в проект на C/C++ необходимо скопировать нужные динамические библиотеки а так же соответствующие им .h файлы в ваш проект. На Windows вместе с `.dll` файлами, в ту же директорию так же надо скопировать `.lib` файлы, в которых содержится информация об экспортируемых функциях библиотеки.

Рекомендуемая структура проекта:

```
your_project/
├── include/      <- x502api.h, lcard_pstdint.h, ...
├── lib/          <- x502api.lib, x502api.dll, ...
├── src/          <- Ваши исходные файлы
└── CMakeLists.txt
```

После копирования требуемых файлов, нежно добавить библиотеки в систему сборки. Мы будем использовать CMake.

Добавляем в CMakeLists.txt следующие строчки:

```
# Указание директории с библиотеками для линковки
link_directories(${CMAKE_SOURCE_DIR}/lib)

# Список DLL библиотек  проекта только( имена без расширений)
set(PROJECT_DLL_LIBRARIES
    x502api
    e502api
    l502api
)

# Линковка  с DLL библиотеками
target_link_libraries(${PROJECT_NAME} ${PROJECT_DLL_LIBRARIES})
```

Для проверки интеграции в проект выведем версию библиотеки:

```
#include <stdio.h>
#include <x502api.h>
#include <e502api.h>

int main() {
    unsigned ver = X502_GetLibraryVersion();
    printf(
        "Версия библиотеки: %d.%d.%d.%d\n",
    (ver>>24), (ver>>16)&0xFF, (ver>>8)&0xFF, ver&0xFF
    );
    return 0;
}
```

Чтобы скомпилировать CMake проект необходимо создать директорию для сборки, и инициализировать в ней CMake, указав на директорию с основным CMakeLists.txt файлом.

```
$ mkdir build
$ cd build
$ cmake ..
```

Теперь можно собрать проект

```
$ cmake --build .
```

После компиляции запускаем исполняемый файл и получаем следующий вывод:

```
Версия библиотеки: 1.1.35.0
```

Это значит, что библиотеки были успешно подключены, и можно приступать к дальнейшей работе.

Python:

Для работы с библиотекой на языке Python будем использовать модуль `ctypes`, который описывает совместимые с языком `C` типы данных, и позволяет вызывать функции в DLL.

```
import ctypes

# Загрузка библиотеки x502api
x502api = ctypes.CDLL('lib/x502api.dll')  # Для Windows
# x502api = ctypes.CDLL('libx502api.so') \# Для Linux

# Загрузка библиотеки x502api
e502api = ctypes.CDLL('lib/e502api.dll')  # Для Windows
# e502api = ctypes.CDLL('libe502api.so') \# Для Linux
```

Теперь можем работать с функциями из библиотеки

```
version = x502api.X502_GetLibraryVersion()
print(f'{version >> 24}.{version >> 16 & 0xFF}.{version >> 8 & 0
   xFF}.{version & 0xFF}')
```

2.2. Подключение по USB/PCI-Express

Если устройство подключено к компьютеру по интерфейсу USB или PCI-E, то работать с ним можно используя серийный номер. Для этого библиотека предлагает следующие функции:

- `L502_Open(t_x502_hnd hnd, const char *serial)` – получает дескриптор модуля L-502 по серийному номеру устройства.
- `E502_OpenUsb(t_x502_hnd hnd, const char* serial)` – получает дескриптор модуля E-502 по серийному номеру устройства.
- `E16_OpenUsb(t_x502_hnd hnd, const char* serial)` – получает дескриптор модуля E16 по серийному номеру устройства.

2.2.1. Подключение по серийному номеру

Самый примитивный способ открытия по серийному номеру - вписать серийник в код напрямую или получить его из ввода пользователя. Например открыть модуль E16 с серийным номером `2T782137` можно с помощью функции `E16_OpenUsb()` можно следующим образом:

C/C++:

```
    t_x502_hnd hnd = X502_Create();

    int err = E16_OpenUsb(hnd, "2T782137");
```

```
    if(err != X502_ERR_OK){
        printf("Ошибка при открытии устройства: %s \n",
   X502_GetErrorString(err));
        goto free;
    }

free:
    X502_Free(hnd);
```

После получение дескриптора устройства, можем его открыть и получить дополнительную информацию, используя функцию `X502_GetDevInfo()`, которая запишет данные устройства в структуру с типом `t_x502_info`.

```
    t_x502_info info;

    err = X502_GetDevInfo(hnd, &info);
    if(err != X502_ERR_OK) {
        printf("Ошибка при получении данных устройства: %s \n",
   X502_GetErrorString(err));
        goto close;
    }

    printf("Открыто устройство\n Название: %s\n Сер. номер: %s \n
   ", info.name, info.serial);

close:
    X502_Close(hnd);
free:
    X502_Free(hnd);
```

Не забываем закрыть устройство с помощью `X502_Close()` и освободить память, выделенную для описателя с помощью `X502_Free()`. Это необходимо делать, даже если используется язык с автоматическим управлением памятью, такой как Python.

2.2.2. Автоматическое получение серийных номеров

API позволяет получить серийные номера подключенных устройств с помощью следующих функций:

- `L502_GetSerialList()`
- `E502_UsbGetSerialList()`
- `E16_UsbGetSerialList()`

```
    #define MAX_DEVCNT 20

    char serials[MAX_DEVCNT][X502_SERIAL_SIZE];
    size_t count = 0;
```

```
 int err = E16_UsbGetSerialList(serials, MAX_DEVCNT, 0, &count
);
 if (err < X502_ERR_OK) {
     printf("Ошибка: %s", X502_GetErrorString(err));
 }
 printf("Обнаружено %d устройств:\n", count);
```

2.3. Подключение по Ethernet

2.3.1. Подключение по IP адресу

Установить соединение с модулем, подключенным по интерфейсу Ethernet можно используя нижеприведенные функции, указав IP адрес устройства.

- E502_OpenByIpAddr()
- E16_OpenByIpAddr()

Примечание. IPv4 адреса модулей записываются в виде 32-битного слова. Например, адрес "a.b.c.d" должен быть записан следующим образом:

```
uint32_t ip_addr = (a<<24)|(b<<16)|(c<<8)|d
```

C/C++

```
// Пусть адрес нашего устройства будет
// 192.168.12.120
ip_addr = (192<<24)|(168<<16)|(12<<8)|120;
t_x502_hnd hnd = X502_Create();
err = E16_OpenByIpAddr(hnd, ip_addr, 0, 200);
```

2.3.2. Поиск устройств в локальной сети

Для поиска устройств в сети используются служба Bonjour (Windows) или демон Avahi (Linux).

Поиск устройств в локальной сети производится с помощью контекста поиска устройств t_e502_eth_svc_browse_hnd. Управление контекстом поиска устройств в локальной сети в библиотеке e502api осуществляется с помощью следующих функций:

- E502_EthSvcBrowseStart() – Создает контекст.
- E502_EthSvcBrowseGetEvent() – получает события.
- E502_EthSvcBrowseStop() – Закрывает и освобождает ресурсы.

Функция E502_EthSvcBrowseGetEvent() создает описатели сетевого сервиса t_e502_eth_svc_record_hnd. Для работы с описателями библиотека предоставляет набор функций. В данном примере мы рассмотрим только следующие функции:

- E502_EthSvcRecordResolveIPv4Addr() – позволяет получить IP адрес сетевого сервиса.
- E502_EthSvcRecordFree() – удаляет и освобождает ресурсы сетевого описателя.

C/C++:

```
int run = 1;
uint32_t count = 0;
int32_t err = X502_ERR_OK;
// Открываем контекст поиска устройств
t_e502_eth_svc_browse_hnd browse_hnd;
err = E502_EthSvcBrowseStart(&browse_hnd, 0);

while (run) {
    t_e502_eth_svc_record_hnd svc_hnd;
    uint32_t event;
    // получаем события из контекста
    err = E502_EthSvcBrowseGetEvent(browse_hnd, &svc_hnd, &event,
    0, 200);
    if(err){
        break;
    }
    switch (event) {
    // если новые устройства не обнаружены, прекращаем поиск
    case E502_ETH_SVC_EVENT_NONE:
        run = 0;
        break;
    // получаем IP адрес нового устройства
    case E502_ETH_SVC_EVENT_ADD: {
        uint32_t ip_addr;
        E502_EthSvcRecordResolveIPv4Addr(svc_hnd, &ip_addr, 200);
        E502_EthSvcRecordFree(svc_hnd);
        count++;
    }
}
if(count == 0) {
    printf("Не найдено ни одного устройства!\n");
}
err = E502_EthSvcBrowseStop(browse_hnd);
```

2.4. Поиск и подключение с помощью записей устройств

Для упрощенного поиска и подключения к модулям в библиотеке реализованы функции работы с записями устройств t_x502_devrec.

1. L502_GetDevRecordsList()
2. E502_UsbGetDevRecordsList()
3. E16_UsbGetDevRecordsList()

Данные функции работают аналогично друг другу и принимают одинаковый набор аргументов:

```
int32_t  E502_UsbGetDevRecordsList  (
    t_x502_devrec *list,
    uint32_t size,
    uint32_t flags,
    uint32_t *devcnt
)
```

Для поиска и получения записей устройств, подключенных по Ethernet можно воспользоваться функцией E502_SearchEthForDevicesIPv4Addr().

```
int32_t E502_SearchEthForDevicesIPv4Addr(
    t_x502_devrec* rec_list,
    uint32_t flags,
    uint32_t size,
    uint32_t *devcount,
    uint32_t event_tout,
    uint32_t tcp_tout
)
```

Данная функция является оберткой для функций E502_EthSvcBrowseXXX() и E502_EthSvcRecordResolveIPv4Addr(). Реализована аналогично примеру в подразделе 2.3.2 и создает записи t_x502_devrec.

Если уже известен IP адрес требуемого модуля, то получить запись устройства можно получить с помощью функций

- E502_MakeDevRecordByIpAddr()
- E502_MakeDevRecordByIpAddr2()

Ниже приведен пример, который создает записи для всех найденных модулей, подключенных по интерфейсам USB, Ethernet и PCI-Express.

Определение TCP_CONNECTION_TOUT задает таймаут в мс на подключение по сетевому интерфейсу, а EVENT_CONNECTION_TOUT задает таймаут в мс на поиск нового события в контексте t_e502_eth_svc_browse_hnd.

Эти функции можно так же использовать, чтобы узнать только количество устройств подключенных по определенному интерфейсу.

```
 t_x502_devrec **pdevrec_list;

 int32_t fnd_devcnt = 0;
 uint32_t pci_devcnt = 0;
 uint32_t e502_usb_devcnt = 0;
 uint32_t e16_usb_devcnt = 0;
 uint32_t eth_devcnt = 0;

 t_x502_devrec *rec_list = NULL;

 /* получаем количество подключенных устройств по интерфейсам
PCI и USB */
 L502_GetDevRecordsList(NULL, 0, 0, &pci_devcnt);
 E502_UsbGetDevRecordsList(NULL, 0, 0, &e502_usb_devcnt);
 E16_UsbGetDevRecordsList(NULL, 0, 0, &e16_usb_devcnt);
 E502_SearchEthForDevicesIPv4Addr(NULL, 0, 0, &eth_devcnt,
EVENT_CONNECTION_TOUT, TCP_CONNECTION_TOUT);
```

Получив количество подключенных устройств, выделим память для массива записей устройств и заполним его используя соответствующие функции.

```
 if ((pci_devcnt + e502_usb_devcnt + e16_usb_devcnt +
eth_devcnt) != 0) {
     /* выделяем память для массива для сохранения найденного
количества записей */
     rec_list = malloc((pci_devcnt + e502_usb_devcnt +
e16_usb_devcnt + ip_cnt + eth_devcnt) * sizeof(t_x502_devrec))
;

     if (rec_list != NULL) {
         unsigned i;
         /* получаем записи о модулях L502, но не больше
pci_devcnt */
         if (pci_devcnt!=0) {
             int32_t res = L502_GetDevRecordsList(&rec_list[
fnd_devcnt], pci_devcnt, 0, NULL);
             if (res >= 0) {
                 fnd_devcnt += res;
             }
         }
         /* добавляем записи о модулях E502, подключенных по
USB, в конец массива */
         if (usb_devcnt!=0) {
             int32_t res = E502_UsbGetDevRecordsList(&rec_list
[fnd_devcnt], e502_usb_devcnt, 0, NULL);
             if (res >= 0) {
                 fnd_devcnt += res;
             }
```

```
            }

            /* добавляем записи о модулях E16, подключенных по
    USB, в конец массива */
            if (e16_usb_devcnt!=0) {
                int32_t res = E16_UsbGetDevRecordsList(&rec_list[
    fnd_devcnt], e16_usb_devcnt, 0, NULL);
                if (res >= 0) {
                    fnd_devcnt += res;
                }
            }

            /* Добавляем записи о модулях E16 и E-502,
    подключенных по Ethernet, в конец массива */
            if(eth_devcnt != 0) {
                int32_t res = E502_SearchEthForDevicesIPv4Addr(&
    rec_list[fnd_devcnt], 0, eth_devcnt, NULL,
    EVENT_CONNECTION_TOUT, TCP_CONNECTION_TOUT);
                if (res >= 0) {
                    fnd_devcnt += res;
                }
            }
        }
     }
```

Проверяем, если получилось создать записи об устройствах. Если устройств не найдено, освобождаем массив.

```
     if (fnd_devcnt != 0) {
         /* если создана хотя бы одна запись, то сохраняем
    указатель на выделенный массив */
         *pdevrec_list = rec_list;
     } else {
         *pdevrec_list = NULL;
         free(rec_list);
     }
}
```

Имея массив подключенных устройств, можем выбрать из него один или несколько устройств, с которыми будем продолжать работу. После открытия нужных устройств с помощью X502_OpenByDevRecord(), необходимо освободить память выделенную для записей устройств. Для этого имеется функция X502_FreeDevRecordList().

```
/* освобождение ресурсов действительных записей из списка */
X502_FreeDevRecordList(devrec_list, fnd_devcnt);
/* очистка памяти самого массива */
free(devrec_list);
```

Глава 3

Настройка параметров модуля

Перед началом работы с модулем необходимо выполнить настройку его параметров. Сначала все настройки записываются в поля структуры описателя модуля с помощью семейства функцийX502_SetXXX. После чего установленные параметры передаются в модуль с помощью X502_Configure().

3.1. Логические каналы

Модули L-502, E-502 и E16 представляют собой АЦП с последовательной коммутацией каналов. То есть измерение нескольких каналов происходит последовательно, путем переключения входного коммутатора АЦП.

Последовательность опроса каналов задается с помощью управляющей таблицы логических каналов АЦП. Всего таблица может содержать от одного до

- X502_LTABLE_MAX_CH_CNT = 256 для модулей L-502 и E-502.
- E16_LTABLE_MAX_CH_CNT = 128 для модулей E16.

логических каналов.

Каждый логический канал задает следующие параметры:

- Номер физического канала, с которого производится измерение.
- Режим измерения АЦП из t_x502_lch_mode.
- Используемый диапазон измерения (из t_x502_adc_range или t_e16_adc_range).
- Коэффициент усреднения (не реализовано в E16).

Для настройки логических каналов используются следующие функции:

- X502_SetLChannelCount() – Задает количество логических каналов.
- X502_SetLChannel() – Задает параметры логического канала.

Для примера настроим 3 логических канала следующим образом:

1. Измеряет напряжение входа X1 относительно общей земли для диапазона +/-10В.
2. Измеряет напряжение между входами X16 и Y16 (16 канале в дифференциальном режиме) с диапазоном +/-1В.
3. Измеряет напряжение между Y1 и общей землей (17 канал в режиме с общей землей) с диапазоном +/-0.2В

В этом случае настройка логической таблицы будет выглядеть следующим образом:

C/C++:

```
/* устанавливаем 3 логических канала */
int32_t err = X502_SetLChannelCount(hnd, 3);
if (err == X502_ERR_OK) {
    /* первый логический канал соответствует измерению 1 канала
   относительно общей земли */
    err = X502_SetLChannel(hnd,0,0,X502_LCH_MODE_COMM,
   X502_ADC_RANGE_10,0);
}
if (err == X502_ERR_OK) {
    /* второй логический канал соответствует измерению 16 канала
    в диф. режиме */
    err = X502_SetLChannel(hnd,1,15,X502_LCH_MODE_DIFF,
    X502_ADC_RANGE_1, 0);
}
if (err == X502_ERR_OK) {
    /* третий логический канал - измерение 17 канала
    относительно общей земли */
    err = X502_SetLChannel(hnd,2,16,X502_LCH_MODE_COMM,
    X502_ADC_RANGE_02, 0);
}
if (err == X502_ERR_OK) {
    /* установка других настроек */
}
if (err == X502_ERR_OK) {
    /* передаем настройки в модуль */
    err = X502_Configure(hnd,0);
}
if (err != X502_ERR_OK) {
    /* произошла ошибка при настройке параметров... */
}
```

3.2. Установка частот и синхронных потоков

При подаче питания на модуль, все потоки выключены, а вводы и выводы находятся в асинхронном режиме. Однако при закрытии модуля одной программой и по-

вторном открытии другой, без отключения питания модуля, конфигурация потоков не сбрасывается. Для избежание непредвиденного поведения при запуске программы, настоятельно рекомендуется остановить и запретить все запущенные потоки.

```
// останавливаем все запущенные потоки
X502_StreamsStop(hnd);
// запрещаем все потоки
X502_StreamsDisable(hnd, X502_STREAM_ALL_IN | X502_STREAM_ALL_OUT
    );
```

После чего можно проводить конфигурацию с ”чистого листа” и разрешать только требуемые в программе потоки, без вероятности коллизий с предыдущей конфигурацией устройства.

```
uint32_t streams_en = X502_STREAM_ADC | X502_STREAM_DOUT |
    X502_STREAM_DIN;
/* разрешаем синхронные потоки */
if (X502_ERR_OK == err) {
    err = X502_StreamsEnable(hnd, streams_en) ;
}
```

Все частоты потокового сбора и выдачи данных основываются на опорной частоте синхронизации. В качестве опорной частоты может использоваться внутренний источник частоты или внешний. Для выбора источника опорной частоты используется функция `X502_SetSyncMode()`. Для примера будем использовать внутреннюю опорную частоту.

```
// используем внутреннюю частоту синхронизации
X502_SetSyncMode(hnd, X502_SYNC_INTERNAL);
```

Модули могут опрашивать цифровые входы и входы с АЦП с разной частотой и задаются соответственно разными функциями

- `X502_SetAdcFreq()` – Устанавливает частоту ввода АЦП.
- `X502_SetDinFreq()` – Устанавливает частоту ввода цифровых входов.

```
    /* устанавливаем частоты ввода для АЦП и цифровых входов */
double adc_freq = X502_REF_FREQ_2000KHZ / 20;
double digin_freq = X502_REF_FREQ_2000KHZ / 20;

if (X502_ERR_OK == err) {
    err = X502_SetAdcFreq(hnd, &adc_freq, NULL);
    printf("Частота сбора АЦП: %.1f, статус: %s\n", out_freq,
    X502_GetErrorString(err));
}

if (X502_ERR_OK == err) {
    err = X502_SetDinFreq(hnd, &digin_freq);
    printf("Частота цифрового ввода: %.1f, статус: %s\n",
    out_freq, X502_GetErrorString(err));
}
```

Однако синхронный вывод на ЦАП и цифровые выходы может происходить только с одинаковой частотой, которая задается функцией X502_SetOutFreq()

```
double out_freq = X502_REF_FREQ_2000KHZ / 20;
if (X502_ERR_OK == err) {
    err = X502_SetOutFreq(hnd, &out_freq);
    printf("Частота синхронного вывода: %.1f, статус: %s\n",
   out_freq, X502_GetErrorString(err));
}
```

Осталось только записать конфигурацию в устройство.

```
/* записываем настройки в модуль */
if (X502_ERR_OK == err) {
    err = X502_Configure(hnd, 0);
}
```

Глава 4

Вывод данных: ЦАП и цифровые выходы

4.1. Асинхронный вывод

По умолчанию, при включении питания устройства, все каналы находятся в асинхронном режиме. При этом задержка от вызова функции до непосредственно момента измерения данных для ввода или выставления указанного значения на выходе для вывода точно не определена Также точно не может быть определена задержка между двумя последовательными операциями ввода/вывода.

4.1.1. Цифро-аналоговый преобразователь

Для асинхронного вывода на ЦАП настройка модуля не требуется, достаточно только вызвать функцию: `X502_AsyncOutDAC()`.

C/C++:

Например:

```
#define VOLTAGE 3.5
X502_AsyncOutDac(hnd, X502_DAC_CH1, VOLTAGE, X502_DAC_FLAGS_VOLT)
   ;
```

Выведет на первый канал ЦАП 3.5 вольта.

4.1.2. Цифровые выходы

Аналогично для цифрового вывода, настройка не требуется, достаточно вызвать функцию `X502_AsyncOutDig()`.

C/C++

```
    X502_AsyncOutDig(hnd, 0x5555, 0);
```

Включает все нечетные выводы, при этом маска не указана, это значит, что предыдущее значение цифрового вывода будет перезаписано.

```
X502_AsyncOutDig(hnd, 0xAAAA, 0x5555);
```

Включает все четные выводы, при этом указана маска на все нечетные выводы, это значит, что значение нечетных выводов не изменится при вызове.

После последовательного выполнения вышеуказанных вызовов функций все цифровые выходы будут включены.

4.2. Синхронный вывод

В синхронном режиме ввод или вывод данных осуществляется с заданной частотой. Запуск синхронного вывода для всех каналов осуществляется одновременно.

4.2.1. Поток

В потоковом режиме данные постоянно создаются управляющем компьютером и передаются на модуль, после чего модуль выводит их с заданной частотой.

Для запуска синхронного режима, необходимо сперва с помощью функции `X502_StreamsEnable()` разрешить синхронный режим по требуемым каналам, а затем запустить синхронный вывод по всем разрешенным каналам с помощью `X502_StreamsStart()`. В enum `t_x502_streams` описаны флаги для обозначения синхронных потоков.

В драйвере (L-502) или библиотеке (E16, E-502), реализован буфер на вывод, из которого модуль запрашивает данные. Чтобы записать значения в этот буфер необходимо подготовить данные с помощью функции `X502_PrepareData()`, после чего данные записывается буфер на вывод с помощью функции `X502_Send()`.

C/C++:

```
/* Выбираем 1 и 2 канал ЦАП */
uint32_t streams = X502_STREAM_DAC1 | X502_STREAM_DAC2;

/* Разрешаем выбранные потоки */
int32_t err = X502_StreamsEnable(hnd, streams);

if (err == X502_ERR_OK) {
    /* Установка других настроек */
}
if (err == X502_ERR_OK) {
    /* Запускаем разрешенные потоки */
    err = X502_StreamsStart(hnd);
}
if (err != X502_ERR_OK) {
    printf("Ошибка: %s!", X502_GetErrorString(err));
}
```

Пример: ШИМ сигналы на цифровых выводах модуля E16 в потоковом режиме.

Целью данного примера – генерация и вывод ШИМ сигналов на цифровых выводах модуля E16.

Сперва необходимо произвести конфигурацию модуля, для работы в потоковом режиме.

```
// останавливаем все запущенные потоки
X502_StreamsStop(hnd);
// запрещаем все потоки
X502_StreamsDisable(hnd, X502_STREAM_ALL_IN | X502_STREAM_ALL_OUT
    );

// используем внутреннюю частоту синхронизации
X502_SetSyncMode(hnd, X502_SYNC_INTERNAL);

double out_freq = X502_REF_FREQ_2000KHZ / 20;
if (X502_ERR_OK == err) {
     err = X502_SetOutFreq(hnd, &out_freq);
     printf("Частота синхронного вывода: %.1f, статус: %s\n",
    out_freq, X502_GetErrorString(err));
}

uint32_t streams_en = X502_STREAM_DOUT;
/* разрешаем синхронные потоки */
if (X502_ERR_OK == err) {
     err = X502_StreamsEnable(hnd, streams_en) ;
}
/* записываем настройки в модуль */
if (X502_ERR_OK == err) {
     err = X502_Configure(hnd, 0);
}
/* запускаем разрешенные потоки */
if (X502_ERR_OK == err) {
     err = X502_StreamsStart(hnd);
}

if (err) {
     printf("Ошибка %d: %s\n", err, X502_GetErrorString(err));
}
```

Примечание. *Запускать потоки с помощью функции `X502_StreamsStart()` не обязательно перед вызовом `X502_Send()`, можно сначала заполнить буфер данными с помощью `X502_Send()`, и уже после этого запускать поток.*

После конфигурации модуля можем начинать работу с потоковым выводом.

Так как значения на всех цифровых выводах меняется одновременно, данные передаются в виде 32 битного слова. Младшие 16 бит выводятся на цифровые выводы, а старшие используются для передачи флагов.

При генерации данных для цифрового вывода удобно использовать поразрядные

операции:

- `&` поразрядное И
- `|` поразрядное ИЛИ
- `<<` сдвиг влево
- `~` поразрядное НЕ

Для включения/выключения определенного выхода:

```
uint32_t word = 0;
word |= (1 << k);
```

Включает k-тый пин цифрового вывода.

```
word &= ~(1 << k);
```

Выключает k-тый пин цифрового вывода.

При работе в потоковом режиме основной цикл программы состоит из двух циклов:

1. Заполнение буфера данными
2. Отправка данных на модуль

Фактическая частота вывода является количеством отсчетов в одной секунде, это можно использовать, чтобы менять состояние выводов, с определенной частотой, кратной частоте вывода.

```
    if(i >= out_freq / n){
        i = 0;
        cycle++;
    }
```

В данном примере переменная `cycle` будет инкрементироваться каждые $\frac{1}{n}$ секунды, что позволяет создавать временные метки для управления выводами.

Следующий код формирует буфер данных, при воспроизведении которого происходит последовательное включение каждого из 16 цифровых выводов с частотой 24 переключения в секунду:

```
for (uint32_t i = 0, j = 0; (err == X502_ERR_OK); i++, j++){
    if(i >= out_freq / 24){
        i = 0;
        cycle++;
    }

    if(j < DIG_BUFFER_SIZE){
        dig_buffer[j] = 1 << (cycle % 16);
    } else {
        /* Отправка данных */
    }
}
```

Чтобы вывести ШИМ сигнал будем использовать стандартный метод пересечения функции с пилообразным сигналом. Мы будем использовать синус, но можно использовать любую другую функцию, в т.ч. константу.

```
#define MAX_DUTY_CYCLE 100
/* Пилообразная функция */
int f_gen_saw(int x) {
    return abs(x % MAX_DUTY_CYCLE);
}
/* Генерация синуса, с возможностью сдвига фазы */
static double f_gen_sin(uint32_t x, uint32_t phase) {
    double half = (float)MAX_DUTY_CYCLE / 2;
    return half * sin(((float)x / 20) - (float)phase / 5) + half;
}
```

У синуса и пилы амплитуда 100, это позволяет использовать возвращаемые значения функции `f_gen_sin()` как коэффициент заполнения напрямую. Остальные коэффициенты синуса были подобраны произвольно.

Чтобы получить значение ШИМ сигнала для каждого отсчета нужна функция компаратор.

```
uint32_t f_compare (uint32_t saw, float duty_cycle) {
    float threshold = (MAX_DUTY_CYCLE / 100 * duty_cycle);
    if(saw < threshold){
        return 1;
    }
    return 0;
}
```

Имея данные функции можно добавить их в цикл

```
    #define DIG_BUFFER_SIZE 0xFFFF

    uint32_t dig_buffer[DIG_BUFFER_SIZE];
    uint32_t send_buffer[DIG_BUFFER_SIZE];
    uint32_t cycle = 0;

    for (uint32_t i = 0, j = 0; (err == X502_ERR_OK); i++, j++){
        if(i >= out_freq / 24){
            i = 0;
            cycle++;
        }
        /* Заполняем буфер */
        if(j < DIG_BUFFER_SIZE){
            uint32_t saw = f_gen_saw(i);
            uint32_t temp = 0;
            /* Собираем слово на вывод */
            for (int k = 0; k < 16; k++) {
                /* Для каждого светодиода сдвигаем фазу на k
   отсчетов */
                float duty_cycle = f_gen_sin(cycle, k);
```

```
                temp |= f_compare(saw, duty_cycle) << k;
            }
            dig_buffer[j] = temp;
        } else {
            /* Отправка данных */
        }
    }
```

После того, как сгенерировали данные, их нужно отправить в буфер на отправку, из которого модуль будет принимать и выводить данные.

Так как функция `X502_Send()` не всегда может отправить все данные: внутренний буфер библиотеки или драйвера может быть уже заполнен, функция возвращает количество записанных данных. Если не обрабатывать значение, возвращаемое функцией `X502_Send()`, это может привести к потере данных.

```
for (uint32_t i = 0, j = 0; (err == X502_ERR_OK); i++, j++){
        /* ... */
        if(j < DIG_BUFFER_SIZE){
            /* Заполняем буфер */
        } else {
        j = 0;
        err = X502_PrepareData(hnd, NULL, NULL, dig_buffer,
   DIG_BUFFER_SIZE, 0, send_buffer);

        if(err == X502_ERR_OK){
            int32_t res = 0;
            uint32_t to_send = DIG_BUFFER_SIZE;
            uint32_t *to_send_ptr = send_buffer;
            /* Отправка данных в буфер */
            while(to_send){
                res = X502_Send(hnd, to_send_ptr, to_send, 500);
                if (res < 0) {
                    err = res;
                    break;
                }
                if (res > 0) {
                    to_send -= res; //
                    to_send_ptr += res;
                }
            }
        }
    }
}
```

При запуске данной программы на каждый цифровой вывод будет подаваться синусоидальный ШИМ сигнал, при этом, каждый вывод будет иметь сдвиг по фазе.

Для большей наглядности при работе с цифровыми выходами можно собрать для них ’заглушку’, которая соединяет каждый цифровой вывод с землей (DGND/ Digital GND) с помощью светодиода (и резистора), таким образом можно визуально определить состояние цифровых выходов.

Данный пример на такой заглушке будет выглядеть как ”волна” из меняющих яркость светодиодов.

4.2.2. Циклический буфер

Данные для загрузки в циклический буфер подготавливаются также как и для потокового вывода с помощью `X502_PrepareData()` и записываются с помощью `X502_Send()` и могут содержать комбинацию данных на оба канала ЦАП и на цифровые выводы.

Циклический вывод является вариантом синхронного вывода и для его работы нужно разрешить нужные потоки на вывод через `X502_StreamsEnable()` и должен быть запущен синхронный ввод-вывод через `X502_StreamsStart()`.

Также как и с обычным потоковым выводом, часть каналов может использоваться для вывода циклического сигнала, а часть — асинхронно.

Однако нельзя часть каналов вывода использовать в циклическом режиме, а часть в потоковом режиме с подкачкой (естественно, циклический режим на вывод можно использовать с потоковым на ввод).

C/C++:

```
#define OUT_BLOCK_SIZE 256

double dac1_data[OUT_BLOCK_SIZE] = { ... };
double dac2_data[OUT_BLOCK_SIZE] = { ... };
uint32_t dout_data[OUT_BLOCK_SIZE] = { ... };

err = X502_PrepareData(hnd, dac1_data, dac2_data, dout_data,
size, X502_DAC_FLAGS_VOLT | X502_DAC_FLAGS_CALIBR, sbuf);
if (err != X502_ERR_OK) {
    fprintf(stderr, ”Ошибка подготовки данных на передачу: %s
\n”,
            X502_GetErrorString(err));
} else {
    /* посылаем данные */
    int32_t snd_cnt = X502_Send(hnd, sbuf, size*ch_cnt,
SEND_TOUT);
    if (snd_cnt < 0) {
        err = snd_cnt;
        fprintf(stderr, ”Ошибка передачи данных: %s\n”,
X502_GetErrorString(err));
    } else if ((uint32_t)snd_cnt != size*ch_cnt) {
        /* так как мы шлем всегда не больше чем готово, то
должны
           всегда передать все */
        fprintf(stderr, ”Передано недостаточно данных:
передавали = %d, передано = %d\n”,
                size*ch_cnt, snd_cnt);
        err = X502_ERR_SEND_INSUFFICIENT_WORDS;
```

```
        }
    }
```

Глава 5

Ввод данных: АЦП и цифровые входы

5.1. Асинхронный ввод

5.1.1. Аналого-цифровой преобразователь

Для асинхронного ввода данных с АЦП используется функция X502_AsyncGetAdcFrame(), которая выполняет ввод одного кадра данных АЦП.

В отличии от других функций асинхронного ввода-вывода, перед вызовом данной функции необходимо выполнить настройку модуля: необходимо задать управляющую таблицу АЦП, и частоту ввода АЦП. Измерение логических каналов внутри кадра происходит синхронно с заданной частотой сбора АЦП.

Асинхронным является ввод самих кадров, то есть задержка между измерением кадров при последовательном вызове X502_AsyncGetAdcFrame() не определена.

C/C++:

Например, код для выполнения однократного измерения с 7-го физического канала в дифференциальным режиме с диапазоном +/-0.5В может выглядеть следующим образом:

```
/* устанавливаем 1 логический канал в управляющей таблице */
int32_t err = X502_SetLChannelCount(hnd, 1);
if (err == X502_ERR_OK) {
    /* логический канал соответствует измерению 7 канала
    в диф. режиме */
    err = X502_SetLChannel(hnd,0,6,X502_LCH_MODE_DIFF,
   L502_ADC_RANGE_05,0);
}
if (err == X502_ERR_OK) {
    /* передаем настройки в модуль */
    err = X502_Configure(hnd,0);
}
```

```
if (err == X502_ERR_OK) {
    /* Считываем кадр данных АЦП из одного отсчета */
    double val;
    err = X502_AsyncGetAdcFrame(hnd,
    X502_PROC_FLAGS_VOLT, 1000, &val);
    if (err == X502_ERR_OK) {
        /* верно считали значение val */
    }
}
```

5.1.2. Цифровые входы

Для использования цифровых входов в асинхронном режиме не требуется настройка модуля, достаточно вызвать функцию `X502_AsyncInDig()`

C/C++

```
    uint32_t data;
    X502_AsyncInDig(hnd, &data);
```

5.2. Синхронный ввод

В синхронном режиме ввод или вывод данных осуществляется с заданной частотой. Частоты сбора для каждого канала задаются относительно общей опорной частоты синхронизации. Запуск синхронного вывода для всех каналов осуществляется одновременно.

5.2.1. Поток

Для запуска синхронного режима, необходимо сперва с помощью функции `X502_StreamsEnable()` разрешить синхронный режим по требуемым каналам, а затем запустить синхронный ввод/вывод по всем разрешенным каналам с помощью `X502_StreamsStart()`. В enum `t_x502_streams` описаны флаги для обозначения синхронных потоков.

C/C++:

Для получения данных с потока, необходимо поток включить

```
    /* Выбираем 1 и 2 канал ЦАП */
    uint32_t streams = X502_STREAM_ADC;

    /* Разрешаем выбранные потоки */
    int32_t err = X502_StreamsEnable(hnd, streams);
```

```
    if (err == X502_ERR_OK) {
        /* Установка других настроек */
    }
    if (err == X502_ERR_OK) {
        /* Запускаем разрешенные потоки */
        err = X502_StreamsStart(hnd);
    }
    if (err != X502_ERR_OK) {
        printf("Ошибка: %s!", X502_GetErrorString(err));
    }
```

После настройки потоков, можно начать получать данные с АЦП. Получаем данные с помощью функции `X502_Recv()`, которая собирает данные со всех входов устройства Цифровых и Аналоговых, и записывает их в буфер.

Основными элементами цикла потокового ввода является получение данных, и их разбор:

```
#define READ_BLOCK_SIZE 0xC6000
#define READ_TIMEOUT 200

static uint32_t rcv_buf[READ_BLOCK_SIZE];

/* принимаем данные по( таймауту) */
rcv_size = X502_Recv(hnd, rcv_buf, READ_BLOCK_SIZE, READ_TIMEOUT)
   ;
/* результат меньше нуля означает ошибку */
if (rcv_size < 0) {
    err = rcv_size;
    fprintf(stderr, "Ошибка приема данных: %s\n",
   X502_GetErrorString(err));
}
```

В буфер данные записываются с дополнительными метаданными, которые нужны для их дальнейшего разбора и упорядочивания. Чтобы разобрать данные, полученные с устройства используется функция `X502_ProcessData()`.

```
    int32_t X502_ProcessData (
        t_x502_hnd hnd,
        const uint32_t *src,
        uint32_t size,
        uint32_t flags,
        double *adc_data,
        uint32_t *adc_data_size,
        uint32_t *din_data,
        uint32_t *din_data_size
    )
```

Данная функция Обрабатывает данные полученные с модуля, проверяет их, и записывает в два массива – данные от АЦП, и данные от синхронного цифрового ввода. Данные с АЦП записываются в буфер по порядку, согласно своему логическому каналу: `{lch_1, lch_2, lch_3, lch_1, lch_2, ...}`

```
static double   adc_data[READ_BLOCK_SIZE];
static uint32_t din_data[READ_BLOCK_SIZE];

if (rcv_size > 0) {

    adc_size = sizeof(adc_data)/sizeof(adc_data[0]);
    din_size = sizeof(din_data)/sizeof(din_data[0]);

    /* обрабатываем принятые данные, распределяя их на данные
   АЦП и цифровых входов */
    err = X502_ProcessData(hnd, rcv_buf, rcv_size,
   X502_PROC_FLAGS_VOLT,
                           adc_data, &adc_size, din_data, &
   din_size);
    if (err != X502_ERR_OK) {
        fprintf(stderr, "Ошибка обработки данных: %s\n",
   X502_GetErrorString(err));
    }
}
```

Так как первый отсчет в блоке данных полученных с АЦП может не соответствовать началу нового кадра. Можем получить номер логического канала, из которого получен первый отсчет в полученном блоке данных АЦП.

```
    uint32_t first_lch;
    /* получаем номер логического канала, которому соответствует
   первый отсчет АЦП в массиве */
    X502_GetNextExpectedLchNum(hnd, &first_lch);
```