ОАО «СКБИС» 195009, Санкт-Петербург, Кондратьевский проспект, дом 2, литер А тел. (812) 334-17-72 доб.175, факс (812) 540-29-33 www.skbis.ru

ОАО «Специальное Конструкторское Бюро Информационно-Измерительных Систем»
Санкт-Петербург

Универсальный протокол взаимодействия с ЛИР

ОПИСАНИЕ БИБЛИОТЕКИ ULP_LIB

ВЕРУ.406920.002 ИС2

01.12.2017

Оглавление

ОАО «СКБИС» 195009, Санкт-Петербург, Кондратьевский проспект, дом 2, литер А тел. (812) 334-17-72 доб.175, факс (812) 540-29-33 www.skbis.ru

Описание

Библиотека ULP_LIB создана для работы с устройствами ЛИР ОАО «СКБ ИС», поддерживающими универсальный протокол взаимодействия. Она позволяет программисту получить удобный программный интерфейс для работы с устройством ЛИР. Более подробную информацию о протоколе взаимодействия читайте в документе «Универсальный протокол взаимодействия. Описание».

Библиотека написана на языке C++ и состоит из следующих файлов:

1. DLL библиотека ulp_lib.dll, собранная для операционной системы Windows.
2. Каталога с заголовочными файлами INC
3. Каталога с исходными текстами библиотеки SOURCE

Заголовочный файл ULP_LIB.h

```
#ifndef ULP_LIB_H
#define ULP_LIB_H


#include <stdint.h>
#include <windows.h>
#include <link_types.h>
#include "module_base_types.h"

#define LIR_ULP_API  extern "C" __declspec(dllexport)
```

ulp_get_num_of_connected_devices

```
/* Функция возвращает количество подключенных устройств при подключении по USB и
колчество COM-портов компьютера при подключении через COM-порт
 _ulp_link - тип подключения (USB или COM)*/
LIR_ULP_API uint32_t
ulp_get_num_of_connected_devices(_ulp_link::__CONNECTION_TYPE_TYPEDEF type);
```

ulp_get_name_of_device

```
 /*Функция возвращает имя устройства при подключении по USB и название COM порта при
подключении через COM.
_ulp_link - тип подключения (USB или COM)
 device_Number - порядковый номер устройства (от 0 до ulp_get_num_of_connected_devices) в
системе.
 device_Name - указатель на строку. Не должен быть NULL.
 Возвращаемое значение - true, если устройство с номером device_Number существует, false
в противном случае*/
LIR_ULP_API bool ulp_get_name_of_device(_ulp_link::__CONNECTION_TYPE_TYPEDEF, uint8_t
device_Number, wchar_t * device_Name);
```

ulp_create

```
/*Функция создает в памяти и возвращает указатель на устройство, подключенное по
_ulp_link
_ulp_link - тип подключения (USB или COM)*/
LIR_ULP_API void* ulp_create(_ulp_link::__CONNECTION_TYPE_TYPEDEF  connection_type);
```

ulp_free

```
/*Функция освобождает память, занятую устройством.
 ulp_device - указатель на устройтсво, полученный с помощью ulp_create*/
LIR_ULP_API void  ulp_free(void *ulp_device);
```

ulp_link_connect

```
/*Функция создает соединение с физическим устройством
 ulp_device - указатель на устройство, полученный с помощью ulp_create
 device_Number - порядковый номер устройства (от 0 до ulp_get_num_of_connected_devices) в
системе.
 Возвращаемое значение - true в результате успеха, false в противном случае*/
LIR_ULP_API bool ulp_link_connect(void* ulp_device, uint8_t device_Number);
```

ulp_link_disconnect

```
/*Функция разрывает соединение с физическим устройством
 ulp_device - указатель на устройство, полученный с помощью ulp_create
 Возвращаемое значение - true в результате успеха, false в противном случае*/
LIR_ULP_API bool ulp_link_disconnect(void* ulp_device);
```

ulp_change_link_settings

```
 /*Функция производит настройки COM порта и номер устройства. Испольлзуется только при
подключении устройства через COM порт.
 Должна вызываться только после ulp_link_connect.
 ulp_device - указатель на устройство, полученный с помощью ulp_create
 settings - настройки порта и номер устройства.
 Возвращаемое значение - true в результате успеха, false в противном случае*/
LIR_ULP_API bool ulp_change_link_settings(const void* ulp_device,
_ulp_link::__COM_SETTINGS_TYPEDEF* settings);
```

ulp_send_message

```
/* Функция создает сообщение для устройства и записывает его в посылку для устройства.
Размер посылки для устройства ограничен.
 Для подключения по USB - 254 байта, для COM- 251 байт. Следует отметить, что сообщение
только помещается в ближайшую посылку. Отправка
 сообщения физическому устройству не происходит.
 ulp_device - указатель на устройство, полученный с помощью ulp_create
 _module - номер модуля, которому предназначается сообщение.
 _command - номер команды для модуля _module
 _data - указатель на данные для передачи в устройство
 _data_size - размер данных в байтах
 Возвращаемое значение - true, если сообщение помещается в ближайшую посылку,  false в
противном случае*/
LIR_ULP_API bool ulp_send_message(void* ulp_device, uint8_t _module, uint8_t _command,
uint8_t* _data, uint32_t _data_size);
```

ulp_exchange

```
 /*Функция запускает обмен посылками между устройством и компьютером.
 ulp_device - указатель на устройство, полученный с помощью ulp_create
 SleepTime - время в миллисекундах между отправкой послыки устройству и чтением посылки,
полученной от устройства.
 ReadBeforeWrite - устанавливает порядок, в котором будет производится чтение и запись
посылки.
 Если ReadBeforeWrite = true, то сначала производится получение посылки, полученной от
устройства, затем выполняется задержка ReadBeforeWrite, затем выполняется запись посылки
в устройство
 Если ReadBeforeWrite = false, то сначала производится запись посылки в устройство, затем
выполняется задержка ReadBeforeWrite, затем выполняется получение посылки от устройства*/
LIR_ULP_API void ulp_exchange(void* ulp_device, uint32_t SleepTime = 0, bool
ReadBeforeWrite = true);
```

ОАО «СКБИС» 195009, Санкт-Петербург, Кондратьевский проспект, дом 2, литер А тел. (812) 334-17-72 доб.175, факс (812) 540-29-33 www.skbis.ru

ulp_set_message_callback

```
/* Процедура задает функцию обратного вызыва для устройства. Данная функция будет вызвана
для каждого сообщения из посылки, полученной от устройства
в процессе работы функции ulp_exchange.
ulp_device - указатель на устройство, полученный с помощью ulp_create
_callback - указатель на функцию, которая будет вызвана для каждого сообщения из посылки,
полученной от устройства
в процессе работы функции ulp_exchange.
_module – номер модуля
_cmd – номер команды
_ptr – указатель на данные
ndata_bytes – число байтов данных*/
LIR_ULP_API void ulp_set_message_callback(const void* ulp_device,
void(*_callback)(__ulp_module::__TYPE::__TYPEDEF& _module, uint8_t _cmd, uint8_t* _ptr,
uint8_t ndata_bytes));
#endif
```

Порядок работы с библиотекой.

1. Вызвать функцию ulp_get_num_of_connected_devices, которая вернет количество подключенных устройств через USB или количество COM портов компьютера.
2. С помощью функции ulp_get_name_of_device выбрать устройство, к которому необходимо подключиться.
3. Создать экземпляр устройства в памяти, вызвав функцию ulp_create.
4. Подключиться к устройству, вызыв функцию ulp_link_connect. Передать в нее номер устройства (от 0 до ulp_get_num_of_connected_devices-1). Следует отметить, что при работе через последовательный порт, необходимо передавать не номер COM порта, а номер устройства, имя которого соответствует возвращаемому значению функции ulp_get_name_of_device.
5. Если подключение происходит через COM порт, вызываем функцию настройки параметров COM порта ulp_change_link_settings.
6. Задаем указатель на функцию обратного вызова в ulp_set_message_callback.
7. С помощью ulp_send_message добавляем сообщения с командами необходимым модулям в посылку для устройства.
8. Вызываем функцию ulp_exchange. Если устройство отвечает, то на каждое сообщение от устройства вызывается функция обратного вызова, которая была зарегистрирована в ulp_set_message_callback.
9. После завершения работы устройством отключаемся от устройства с помощью функции ulp_link_disconnect.
10. Удаляем выделенную под устройство память с помощью ulp_free.

Примеры работы с устройством

Для облегчения понимания примеров работы с устройством был разработан заголовочный файл ulp_tools.h, в котором содержаться вспомогательные функции и другие объявления, упрощающие работу с устройством.

ulp_tools.h

```
#ifndef ULP_TOOLS_H
#define ULP_TOOLS_H

#include "ulp_lib.h"
#include "module_system_types.h"
#include "link_types.h"
#include <vector>



#if  _WIN64
 #if _DEBUG
 #pragma comment (lib, "../../x64/Debug/ulp_lib.lib")
```

```
 #else
 #pragma comment (lib, "../../x64/Release/ulp_lib.lib")
 #endif
#else
 #if _DEBUG
 #pragma comment (lib, "../../Win32/Debug/ulp_lib.lib")
 #else
 #pragma comment (lib, "../../Win32/Release/ulp_lib.lib")
 #endif
#endif

using namespace std;
#define ATIME  10
#define ACK 0xF0
#define NACK 0x0F

static const char* module_type_strings[]  {

              "__SYSTEM ",
              "__SENSOR",
              "__RS485",
              "__HARD_IO",
              "__VIRTUAL_IO",
              "__FUNC_REMAP",
              "__POSITIONER",
              "__GMACHINE",
              "__TELEMETRY",
              "__ZONE_DETECTOR",
              "__LAST"

};

struct Command
{
       uint8_t module;
       uint8_t cmd;
       uint8_t* data_ptr;
       uint8_t ndata_bytes;
};

/*Функция возвращает vector с именами устройств (при подключении по USB) или именами COM
портов, при подключении по последовательному порту
_ctype - тип подключения (USB или COM)
deviceList - массив с именами устройств
Возвращаемое значение - true в результате успеха, false в противном случае*/
bool  ulp_get_device_list(_ulp_link::__CONNECTION_TYPE_TYPEDEF _ctype, vector<wstring>&
deviceList)
{
       deviceList.clear();
       uint32_t deviceNumber = ulp_get_num_of_connected_devices(_ctype);
       for (int i = 0; i < deviceNumber; i++)
       {
              wchar_t* deviceName = new wchar_t[200];
              if (ulp_get_name_of_device(_ctype, i, deviceName))
              {
                     wstring wsname = wstring(deviceName);
                     deviceList.push_back(wsname);
              }
              delete[] deviceName;
       }
       return !deviceList.empty();
}
```

```
/*Функция выделяет память на структуру устройства в памяти, возвращает указатель и
производит подключение к устройству.
Является объединением функций ulp_create и ulp_link_connect.
_type - тип подключения (USB или COM)
_number - номер устройства USB или номер COM-порта в массиве, возвращаемом функцией
ulp_get_device_list, при поключении через COM-порт
handle - указатель на область памяти, где расположен указатель устройства. Возвращаемое
значение.
_comSettings - настройки порта и номер устройства. Только для подключения через COM порт.
Возвращаемое значение - true в результате успеха, false в противном случае*/
bool ulp_create_and_connect(_ulp_link::__CONNECTION_TYPE_TYPEDEF _type, uint8_t _number,
void** handle,  _ulp_link::__COM_SETTINGS_TYPEDEF* _comSettings= NULL)
{
	*handle = ulp_create(_type);
	if (!*handle) return false;

	if (!ulp_link_connect(*handle, _number)) return false;

	if ((_type == _ulp_link::__TYPE_COM) && _comSettings)
	{
		if (ulp_change_link_settings(*handle, _comSettings)) return false;
	}

	return true;

}
/*Функция разрывает соединение с устройством и освобождает память, занятую в памяти под
него.
Является объединением функций ulp_link_disconnect и ulp_free  */
bool ulp_disconnect_and_free(void* handle)
{
	if (!ulp_link_disconnect(handle)) return false;
	ulp_free(handle);
	return true;
}

//Структура, которая хранит данные о последней команде, полученной от устройства.
Command lastCommand;

//Количество данных в буфере
int inBuff_num = 0;

// Контейнер для хранения данных из буфера
vector<__ulp_m_sensor::__SENS_DATA_TYPEDEF> buff_data;

//Функция обратного вызова. Заполняет глобальную структуру lastCommand данными,
полученными из сообщения от устройства
void callback_handler(__ulp_module::__TYPE::__TYPEDEF& _module, uint8_t _cmd, uint8_t*
_ptr, uint8_t ndata_bytes)
{
	// Заполняем структуру
	lastCommand.cmd = _cmd;
	lastCommand.module = _module;
	lastCommand.data_ptr = _ptr;
	lastCommand.ndata_bytes = ndata_bytes;

	// Если данные из буфера, то помещаем их в vector buff_data
	if ((lastCommand.module == __ulp_module::__TYPE::__SYSTEM) && (lastCommand.cmd ==
__ulp_m_system::__CMD::__GET_LOG_DATA))
	{
		__ulp_m_system::__SEND_LOG_DATA_TYPEDEF* pSend_log_data =
(__ulp_m_system::__SEND_LOG_DATA_TYPEDEF*)_ptr;

		//Определяем сколько данных в буфере на момент запроса
```

```
                inBuff_num = pSend_log_data->status.queue;

                //Определяем сколько данных в команде
                uint8_t num_datas = (ndata_bytes - 2) / 10;

                for (int i = 0; i < num_datas; i++)
                {
                        buff_data.push_back(pSend_log_data->data[i]);
                }

        }

}

// Функция очищает информацию в lastCommand
void ulp_clear_last_command()
{
        lastCommand.cmd = 0;
        lastCommand.data_ptr = 0;
        lastCommand.module = 0;
        lastCommand.ndata_bytes = 0;
}

bool ulp_clear_send_exchange(void* ulp_device, uint8_t _module, uint8_t _command,
uint8_t* _data, uint32_t _data_size, bool isReadBeforeWrite = false, uint32_t sleep_time
= ATIME )
{
        ulp_clear_last_command();
        ulp_send_message(ulp_device, _module, _command, _data, _data_size);
        ulp_exchange(ulp_device, sleep_time, isReadBeforeWrite);
        return lastCommand.ndata_bytes;

}


#endif
```

Пример 1. Работа с системным модулем.

```
// Пример программы для работы с устройством, реализующем ULP протокол
// Программа подключается к устройству, запрашивает количество модулей и подмодулей у
каждого модуля и выводит информацию о
// модулях и подмодулях на экран.

#include "stdafx.h"
#include <stdint.h>
#include <iostream>
#include <windows.h>
#include <process.h>
#include "ulp_lib.h"
#include "ulp_tools.h"
#include "module_sensor_types.h"
#include "module_system_types.h"
#include <link_types.h>
#include <vector>
#include <string>



using namespace std;



int _tmain(int argc, _TCHAR* argv[])
{
	// Выбираем тип подключения. В данном случае это USB
	_ulp_link::__CONNECTION_TYPE_TYPEDEF ctype = _ulp_link::__TYPE_USB;

	vector<wstring> deviceList;

	// Запрашиваем и выводим массив имен устройств, подключенных к компьютеру
	if (ulp_get_device_list(ctype, deviceList))
	{
		for (int i = 0; i < deviceList.size(); i++)
		{

			wcout<<"Device "<<i<< ": "<<deviceList[i]<<endl;
		}
	}

	void* handle = NULL;

	// Выделяем память и подключаемся к устройству с номером 0.
	if (ulp_create_and_connect(ctype, 0, &handle)) cout<<"Create & Connect OK
!"<<endl;


	//Если подключаемся у устройству через COM порт, настраиваем COM порт. Указываем
адрес устройства (247)
	if (ctype == _ulp_link::__TYPE_COM)
	{
		_ulp_link::__COM_SETTINGS_TYPEDEF settings;
		settings.baudrate = 115200;
		settings.dev_addr = 247;
		settings.nstop = _ulp_link::__STOP_BITS_1;
		settings.parity = _ulp_link::__PARITY_NO;
```

```
		if (ulp_change_link_settings(handle, &settings)) cout << "COM settings OK
!" << endl;
	}

	// Устанавливаем функцию обратного вызыва для приема сообщений от устройства
	ulp_set_message_callback(handle, callback_handler);

	// Записываем сообщение для модуля "Системный"  с командой "Запрос количества
модулей в устройстве"  в текущую посылку
	ulp_send_message(handle, __ulp_module::__TYPE::__SYSTEM,
__ulp_m_system::__CMD::__GET_MODULES_NUM, NULL, 0);

	// Производим обмен данных устройства с компьютером. При этом просим отправить
посылку устройству,
	//затем подождать ATIME миллисекунд, а затем принять посылку. Задержка ATIME
сделана для того, чтобы устройство успело обработать
	//посылку, ответить и передать посылку компьютеру.
	ulp_exchange(handle, ATIME, false);

	uint8_t modules_num = 0;  // количество модулей в устройстве

	// После того, как отработает callback_handler, в структуре lastCommand окажется
ответ на последнюю команду устройству.
	// В нашем случае там будет количество модулей в устройстве.
	if (lastCommand.ndata_bytes > 0) modules_num = lastCommand.data_ptr[0];

	cout << "Modules number :" << (int )modules_num<<endl;


	for (uint8_t i = 0; i < modules_num; i++)
	{
		//Очищаем информацию в lastCommand
		ulp_clear_last_command();

		// Отправляем команду "Запрос информации о модуле" каждому модулю
устройства и выводим ее на экран
		ulp_send_message(handle, (__ulp_module::__TYPE::__TYPEDEF) i,
__ulp_module::__SHARED_CMD::__GET_MODULE_INFO, NULL, 0);
		ulp_exchange(handle, ATIME, false);

		if (lastCommand.ndata_bytes > 0)
		{
			__ulp_module::__TYPE::__TYPEDEF type =
(__ulp_module::__TYPE::__TYPEDEF)lastCommand.data_ptr[0];
			cout << "Module " << (int)i << " Type: " <<
module_type_strings[type] << "\t\t";
		}

		uint8_t submodules_num = 0;
		ulp_clear_last_command();

		//Отправляем команду "Запрос количества подмодулей" каждому модулю и
выводим ее на экран
		ulp_send_message(handle, (__ulp_module::__TYPE::__TYPEDEF) i,
__ulp_module::__SHARED_CMD::__GET_SUB_MODULE_NUM, NULL, 0);
		ulp_exchange(handle, ATIME, false);


		if (lastCommand.ndata_bytes > 0) submodules_num = lastCommand.data_ptr[0];
		cout << "Submodules number :" << (int)submodules_num << endl;
		for (uint8_t j = 0; j < submodules_num; j++)
		{
			ulp_clear_last_command();
```

```
                        // Отправляем команду "Запрос информации о подмодуле" каждому
подмодулю и выводим ее на экран
                        ulp_send_message(handle, (__ulp_module::__TYPE::__TYPEDEF) i,
__ulp_module::__SHARED_CMD::__GET_SUB_MODULE_INFO, &j, sizeof(j));
                        ulp_exchange(handle, ATIME, false);
                        if (lastCommand.ndata_bytes > 0)
                        {
                                cout << "\t Submodule "<<(int)j<<"  type: " <<
(int)lastCommand.data_ptr[0]<<endl;
                        }
                }
        }

        // Разрываем связь с устройством и освобождаем память
        ulp_disconnect_and_free(handle);


        getchar();
        return 0;
}
```

Пример 2. Работа с модулем инкрементного датчика.

```
// Пример работы с модулем инкрементного датчика
//

#include "stdafx.h"
#include "ulp_lib.h"
#include "ulp_tools.h"
#include <iostream>
#include <conio.h>

int _tmain(int argc, _TCHAR* argv[])
{



	setlocale(LC_ALL, "Russian");
	void* handle = NULL;
	_ulp_link::__CONNECTION_TYPE_TYPEDEF ctype = _ulp_link::__TYPE_USB;

	// Выделяем память и подключаемся к устройству с номером 0.
	if (ulp_create_and_connect(ctype, 0, &handle)) cout << "Создание и подключение:OK
!" << endl;
	else
	{
		cout << "Ошибка подключения!";
		return 1;
	}

	// Устанавливаем функцию обратного вызыва для приема сообщений от устройства
	ulp_set_message_callback(handle, callback_handler);

	// Проверяем, что модуль 1 - это модуль датчика

	uint8_t sensor_module_num = 1;
	ulp_send_message(handle, sensor_module_num,
__ulp_module::__SHARED_CMD::__GET_MODULE_INFO, NULL, 0);
	ulp_exchange(handle, ATIME, false);
	if (lastCommand.ndata_bytes)
	{
		__ulp_module::__TYPE::__TYPEDEF module_type =
(__ulp_module::__TYPE::__TYPEDEF)lastCommand.data_ptr[0];
		if (module_type == __ulp_module::__TYPE::__SENSOR) cout << "Тип модуля
"<<(int)sensor_module_num<<": Датчик"<<endl;

	}

	// Очищаем структуру
	ulp_clear_last_command();

	//Проверяем включен ли модуль датчика и, если не включен, включаем
	ulp_send_message(handle, sensor_module_num,
__ulp_module::__SHARED_CMD::__GET_MODULE_STATE, NULL, 0);
	ulp_exchange(handle, ATIME, false);
	uint8_t isModuleDisable;
	if (lastCommand.ndata_bytes)
	{
		isModuleDisable = lastCommand.data_ptr[0]; // 0 -включен, 1- выключен
		if (isModuleDisable)
		{
```

ОАО «СКБИС» 195009, Санкт-Петербург, Кондратьевский проспект, дом 2, литер А тел. (812) 334-17-72 доб.175, факс (812) 540-29-33 www.skbis.ru

```
                        ulp_clear_last_command();
                        // Включаем модуль датчика
                        uint8_t data = 0;
                        ulp_send_message(handle, sensor_module_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_STATE, &data, sizeof(data));
                        ulp_exchange(handle, ATIME, false);
                        if (lastCommand.ndata_bytes && (lastCommand.data_ptr[0] == ACK))
                        {
                                isModuleDisable = 0;
                        }

                }
                if (!isModuleDisable) cout << "Модуль датчика включен !" << endl;
                else
                {
                        cout << "Модуль датчика выключен !" << endl;

                }

        }

        // Устанавливаем режим работы модуля датчика "Инкрементный".

        // Выбираем подмодуль инкрементных датчиков
        uint8_t submodule = __ulp_m_sensor::__SUB_MODULE_TYPES::__INCREMENTAL;
        ulp_clear_last_command();
        ulp_send_message(handle, sensor_module_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_MODE, &submodule, sizeof(submodule));
        ulp_exchange(handle, ATIME, false);
        if (lastCommand.ndata_bytes && (lastCommand.data_ptr[0] = ACK))
                cout << "Режим работы модуля: Инкрементный"<<endl;


        for (int i = 0; i < 10; i++)
        {

                // Обновляем и запрашиваем текущую координату датчика

                ulp_clear_last_command();
                ulp_send_message(handle, sensor_module_num,
__ulp_m_sensor::__CMD::__UPD_SENSOR_DATA, 0, 0);
                uint8_t axis = 0;
                ulp_send_message(handle, sensor_module_num,
__ulp_m_sensor::__CMD::__GET_SENSOR_DATA, &axis, sizeof(axis));
                ulp_exchange(handle, ATIME, false);

                if (lastCommand.ndata_bytes)
                {
                        __ulp_m_sensor::__SENS_DATA_TYPEDEF* psensor_data =
(__ulp_m_sensor::__SENS_DATA_TYPEDEF*)lastCommand.data_ptr;
                        __ulp_m_sensor::__SENS_DATA_TYPEDEF sensor_data(psensor_data);
                        cout << "Текущая координата:" << sensor_data.coordinate << endl;
                        cout << "Биты ошибок:" << sensor_data.status.alarm_bits << endl;
                        cout << "Ошибка коррекции:" << sensor_data.status.correction_error <<
endl;
                        cout << "Не захвачена референтная метка:" <<
sensor_data.status.no_ref << endl;
                        cout << "Референтна метка захвачена:" <<
sensor_data.status.ref_capture << endl;
                        cout << "Ошибка датчика:" << sensor_data.status.sensor_error << endl
<< endl;


                }
```

```
            Sleep(1000);
        }

        ulp_disconnect_and_free(handle);

        _getch();
        return 0;
}
```

ОАО «СКБИС» 195009, Санкт-Петербург, Кондратьевский проспект, дом 2, литер А тел. (812) 334-17-72 доб.175, факс (812) 540-29-33 www.skbis.ru

Пример 3. Работа с таблицей коррекции датчика.

```
// Пример работы с таблицей коррекции для датчика.
//

#include "stdafx.h"
#include "ulp_lib.h"
#include "ulp_tools.h"
#include <iostream>
#include <conio.h>

using namespace std;

typedef struct {
	int64_t coordinate;
	int64_t correction : 8;


} __CORRECTION_POINT_TYPEDEF;


uint8_t DrawMainMenu()
{
	cout << endl;
	cout << "1. Вывести таблицу коррекции из ОЗУ" << endl;
	cout << "2. Стереть таблицу коррекции в ОЗУ" << endl;
	cout << "3. Добавить точку коррекции в ОЗУ" << endl;
	cout << "4. Удалить точку коррекции из ОЗУ" << endl;
	cout << "5. Сохранить таблицу коррекции в ПЗУ" << endl;
	cout << "6. Загрузить таблицу коррекции из ПЗУ" << endl;
	cout << "0. Выход" << endl;
	cout << "Выберите пункт:";
	uint8_t result;
	cin >> result;
	cout << endl;
	return result;

}

bool DeleteCorrectionPoint(void* handle)
{
	uint8_t point_number;
	cout << "Введите номер точки для удаления:";
	cin >> point_number;
	point_number -= '0';
	bool result = false;
	if (ulp_clear_send_exchange(handle, 1,
__ulp_m_sensor::__CMD::__DELETE_CORRECTION_POINT, &point_number, sizeof(point_number))
		&& (lastCommand.data_ptr[0] == ACK))
	{
		cout << "Точка " << (int)point_number << " удалена"<<endl;
		result = true;
	}
	return result;
}

//Считываем значение координаты и корекции с консоли. Возвращаем структуру с введенными
значениями.
bool ReadCorrectionPoint(__CORRECTION_POINT_TYPEDEF* _correction_point)
{
```

```
	try
	{

		cout << "Введите координату:";
		int32_t coord;
		cin >> coord;
		cout << "Введите коррекцию:";
		int16_t correction;
		cin >> correction;
		_correction_point->coordinate = coord;
		_correction_point->correction= correction;
	}
	catch (exception ex)
	{
		return false;
	}
	return true;
}

// Добавить точку коррекции в таблицу
bool AddCorrectionPoint(void* handle)
{
	__CORRECTION_POINT_TYPEDEF correction_point;
	__CORRECTION_POINT_TYPEDEF* _pcorrection_point = &correction_point;



	return ReadCorrectionPoint(_pcorrection_point) &&
		ulp_clear_send_exchange(handle, 1,
__ulp_m_sensor::__CMD::__ADD_CORRECTION_POINT_MANUAL, (uint8_t*)_pcorrection_point,
sizeof(__CORRECTION_POINT_TYPEDEF)) &&
		lastCommand.data_ptr[0] == ACK;
}

// Вывод текущей таблицы коррекции на экран
void DrawCorrectionTable(void* handle)
{
	// Определяем количество точек в таблице коррекции
	uint8_t sensor_module_num = (uint8_t) __ulp_module::__TYPE::__SENSOR;
	uint8_t cpoints_num = 0;
	if (ulp_clear_send_exchange(handle, sensor_module_num,
__ulp_m_sensor::__CMD::__GET_CORRECTION_POINT_NUM, NULL, 0))
	{
		cpoints_num = lastCommand.data_ptr[0];
	}

	if (!cpoints_num) cout << "Таблица пуста !"<<endl;
	else
	{
		cout << "Номер точки\tКоордината\tКоррекция"<<endl;
	}

	for (uint8_t i = 0; i < cpoints_num; i++)
	{

		if (ulp_clear_send_exchange(handle, 1,
__ulp_m_sensor::__CMD::__GET_CORRECTION_POINT_INFO, &i, sizeof(i)))
		{
			__CORRECTION_POINT_TYPEDEF* pcorrection_data =
(__CORRECTION_POINT_TYPEDEF*)lastCommand.data_ptr;

			cout << (int16_t)i << "\t\t" << pcorrection_data->coordinate <<
"\t\t" << (int16_t)pcorrection_data->correction << endl;
		}
```

```
	}

}


int _tmain(int argc, _TCHAR* argv[])
{
	setlocale(LC_ALL, "Russian");
	void* handle = NULL;
	_ulp_link::__CONNECTION_TYPE_TYPEDEF ctype = _ulp_link::__TYPE_USB;

	// Выделяем память и подключаемся к устройству с номером 0.
	if (ulp_create_and_connect(ctype, 0, &handle)) cout << "Создание и подключение к
устройству: OK !" << endl;
	else
	{
		cout << "Ошибка подключения!";
		return 1;
	}

	// Устанавливаем функцию обратного вызыва для приема сообщений от устройства
	ulp_set_message_callback(handle, callback_handler);


	// Включаем модуль датчика
	uint8_t data = 0;
	ulp_send_message(handle, 1, __ulp_module::__SHARED_CMD::__SET_MODULE_STATE, &data,
sizeof(data));
	ulp_exchange(handle, ATIME, false);
	if (lastCommand.ndata_bytes && (lastCommand.data_ptr[0] == ACK))
	{
		cout << "Модуль датчика включен. "<<endl;
	}

	uint8_t input = 0;

	while ((input = DrawMainMenu()) != '0')
	{
		switch (input)
		{
		case '1':
		{
					DrawCorrectionTable(handle);
					break;
		}
		case '2':
		{
					if (ulp_clear_send_exchange(handle, 1,
__ulp_m_sensor::__CMD::__INIT_CORRECTION_PROC, NULL, 0)
						&& lastCommand.data_ptr[0] == ACK)
					{
						cout << "Таблица коррекции обнулена." << endl;
					}
					break;
		}
		case '3':
		{
					if (AddCorrectionPoint(handle)) cout<<"Точка
сохранена."<<endl;
					break;
		}
		case '4':
		{
```

```
                                        DeleteCorrectionPoint(handle);
                                        break;
                }
                case '5':
                {
                                        if (ulp_clear_send_exchange(handle, 1,
__ulp_m_sensor::__CMD::__SAVE_CORRECTION_TABLE, NULL, 0)     && (lastCommand.data_ptr[0]
= ACK))
                                                cout << "Таблица коррекции успешно сохранена в
ПЗУ." << endl;
                                        break;
                }
                case '6':
                {
                                        if (ulp_clear_send_exchange(handle, 1,
__ulp_m_sensor::__CMD::__ABORT_CORRECTION_PROCESS, NULL, 0) && (lastCommand.data_ptr[0] =
ACK))
                                                cout << "Таблица коррекции успешно загружена из
ПЗУ." << endl;
                                        break;
                }
                default:
                        break;
                }
        }

        ulp_disconnect_and_free(handle);


        return 0;
}
```

Пример 4. Работа с модулем входов/выходов.

```
// Пример работы с модулем входов/выходов. Программа считывает состояние входов и
устанавливает такое же состояние выходов
//
#include "stdafx.h"
#include "ulp_lib.h"
#include "ulp_tools.h"
#include <iostream>
#include <module_io_types.h>
#include <conio.h>

using namespace std;



int _tmain(int argc, _TCHAR* argv[])
{
	setlocale(LC_ALL, "Russian");
	void* handle = NULL;
	_ulp_link::__CONNECTION_TYPE_TYPEDEF ctype = _ulp_link::__TYPE_USB;

	// Выделяем память и подключаемся к устройству с номером 0.
	if (ulp_create_and_connect(ctype, 0, &handle)) cout << "Создание и подключение к
устройству: OK !" << endl;
	else
	{
		cout << "Ошибка подключения!";
		return 1;
	}


	// Устанавливаем функцию обратного вызыва для приема сообщений от устройства
	ulp_set_message_callback(handle, callback_handler);


	uint8_t module_num = 3;
	// Включаем модуль входов/выходов
	uint8_t  state = 0;
	if (ulp_clear_send_exchange(handle, module_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_STATE, &state, sizeof(state))
		&& (lastCommand.data_ptr[0] = ACK))
	{
		cout << "Модуль входов/выходов включен." << endl;
	}


	do
	{


		cout << endl;
		cout << "Считываем состояние выходов:" << endl;
		// Считываем состояние входов
		uint8_t inputs_num = 4; // количество входов устройства
		uint16_t inputs_state; // состояние входов

		if (ulp_clear_send_exchange(handle, module_num,
__ulp_m_io::__CMD::__GET_INPUTS, NULL, 0))
		{
```

ОАО «СКБИС» 195009, Санкт-Петербург, Кондратьевский проспект, дом 2, литер А тел. (812) 334-17-72 доб.175, факс (812) 540-29-33 www.skbis.ru

```
                        inputs_state = *((uint16_t*)lastCommand.data_ptr);
                        for (int i = 0; i < inputs_num; i++)
                        {
                                cout << "Вход номер:" << i << " Состояние входа:"
<<((inputs_state & (1 << i))>0) << endl;

                        }
                }

                //Устанавливаем состояние входов на выхода устройства
                cout << endl;
                uint8_t outputs_num = 4;
                if (ulp_clear_send_exchange(handle, module_num,
__ulp_m_io::__CMD::__SET_OUTPUTS, (uint8_t*)&inputs_state, sizeof(inputs_state))
                        && (lastCommand.data_ptr[0] = ACK))
                {
                        cout << "Выхода установлены." << endl<<endl;
                }

                inputs_state = 0;
                //Считываем состояние выходов
                cout << "Считываем состояние выходов:" << endl;
                if (ulp_clear_send_exchange(handle, module_num,
__ulp_m_io::__CMD::__GET_OUTPUTS, NULL, 0))

                {
                        inputs_state = *((uint16_t*)lastCommand.data_ptr);
                        for (int i = 0; i < outputs_num; i++)
                        {
                                cout << "Выход номер:" << i << " Состояние выхода:" <<
((inputs_state & (1 << i))>0) << endl;

                        }
                }

                cout <<endl<< "Нажмите 'q' для выхода ..."<<endl;
        } while (_getch() != 'q');


        ulp_disconnect_and_free(handle);


        return 0;
}
```

Пример 5. Работа с буфером устройства. Запись координаты датчика по таймеру

```
// TimerSensor.cpp
// Пример демонстрирует работу с буфером устройства. По заданному периоду таймера
значения с датчика записываются в буфер, а затем
// сохраняются в файл.
//

#include "stdafx.h"
#include "../../inc/ulp_lib.h"
#include "../../inc/ulp_tools.h"
#include <iostream>
#include <conio.h>
#include "../../inc/module_virtual_io_types.h"
#include  "../../inc/module_func_remap_types.h"
#include "../../inc/sub_module_sensor_inc_types.h"
#include <fstream>

int _tmain(int argc, _TCHAR* argv[])
{


	setlocale(LC_ALL, "Russian");
	void* handle = NULL;
	_ulp_link::__CONNECTION_TYPE_TYPEDEF ctype = _ulp_link::__TYPE_USB;

	// Выделяем память и подключаемся к устройству с номером 0.
	if (ulp_create_and_connect(ctype, 0, &handle)) cout << "Создание и подключение:OK
!" << endl;
	else
	{
		cout << "Ошибка подключения!";
		return 1;
	}

	// Устанавливаем функцию обратного вызыва для приема сообщений от устройства
	ulp_set_message_callback(handle, callback_handler);

	//!!!!!!!!!!!




	ulp_send_message(handle, __ulp_module::__TYPE::__SYSTEM,
__ulp_m_system::__CMD::__GET_LOG_DATA, NULL, 0);

	ulp_exchange(handle, 0, false);



	// Проверяем, что модуль 1 - это модуль датчика

	uint8_t sensor_module_num = 1;
	ulp_send_message(handle, sensor_module_num,
__ulp_module::__SHARED_CMD::__GET_MODULE_INFO, NULL, 0);
	ulp_exchange(handle, ATIME, false);
	if (lastCommand.ndata_bytes)
	{
		__ulp_module::__TYPE::__TYPEDEF module_type =
(__ulp_module::__TYPE::__TYPEDEF)lastCommand.data_ptr[0];
		if (module_type == __ulp_module::__TYPE::__SENSOR) cout << "Тип модуля " <<
(int)sensor_module_num << ": Датчик" << endl;

	}
```

```
	// Очищаем структуру
	ulp_clear_last_command();

	//Проверяем включен ли модуль датчика и, если не включен, включаем
	ulp_send_message(handle, sensor_module_num,
__ulp_module::__SHARED_CMD::__GET_MODULE_STATE, NULL, 0);
	ulp_exchange(handle, ATIME, false);
	uint8_t isModuleDisable;
	if (lastCommand.ndata_bytes)
	{
		isModuleDisable = lastCommand.data_ptr[0]; // 0 -включен, 1- выключен
		if (isModuleDisable)
		{
			ulp_clear_last_command();
			// Включаем модуль датчика
			uint8_t data = 0;
			ulp_send_message(handle, sensor_module_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_STATE, &data, sizeof(data));
			ulp_exchange(handle, ATIME, false);
			if (lastCommand.ndata_bytes && (lastCommand.data_ptr[0] == ACK))
			{
				isModuleDisable = 0;
			}

		}
		if (!isModuleDisable) cout << "Модуль датчика включен !" << endl;
		else
		{
			cout << "Модуль датчика выключен !" << endl;

		}

	}

	// Устанавливаем режим работы модуля датчика "Инкрементный".

	// Выбираем подмодуль инкрементных датчиков
	uint8_t submodule = __ulp_m_sensor::__SUB_MODULE_TYPES::__INCREMENTAL;
	ulp_clear_last_command();
	ulp_send_message(handle, sensor_module_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_MODE, &submodule, sizeof(submodule));
	ulp_exchange(handle, ATIME, false);
	if (lastCommand.ndata_bytes && (lastCommand.data_ptr[0] == ACK))
		cout << "Режим работы модуля: Инкрементный" << endl;

	ulp_clear_last_command();
	ulp_send_message(handle, sensor_module_num,
__ulp_sm_sensor_inc::__CMD::__CLEAR_POSITION, NULL, 0);
	ulp_exchange(handle, ATIME, false);
	if (lastCommand.ndata_bytes && (lastCommand.data_ptr[0] == ACK))
		cout << "Координата обнулена" << endl;

	// Включаем модуль виртуальных входов/выходов
	uint8_t virtual_io_module_num = 4; // Порядковый номер модуля виртуальных
входов/выходов
	ulp_clear_last_command();
	uint8_t data = 0;
	ulp_send_message(handle, virtual_io_module_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_STATE, &data, sizeof(data));
	ulp_exchange(handle, ATIME, false);
	if (lastCommand.ndata_bytes && (lastCommand.data_ptr[0] == ACK))
	{
		cout << "Модуль виртуальных входов/выходов включен." << endl;
```

```
	}

	int freq;
	cout << "Введите частоту таймера в Гц:";
	cin >> freq;
	uint32_t dfreq = 1000 / freq; // Определяем делитель частоты для записи в
устройтсво
	// Настраиваем таймер в режим меандра, период 10 мс
	ulp_clear_last_command();
	// Создаем структуру, содержащую 4 таймера, 8 виртуальных входов и 8 виртуальных
выходов
	__ulp_m_virtual_io::__SETTINGS_TYPEDEF<4, 8, 8> v_io_settings;
	//Настраиваем 0-й таймер
	v_io_settings.timers[0].pulse = false; // режим работы меандр
	v_io_settings.timers[0].max_cntr = dfreq; // записываем делитель частоты
	ulp_send_message(handle, virtual_io_module_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_SETTINGS, (uint8_t*)&v_io_settings,
sizeof(__ulp_m_virtual_io::__SETTINGS_TYPEDEF<4, 8, 8>));
	ulp_exchange(handle, ATIME, false);
	if (lastCommand.ndata_bytes && (lastCommand.data_ptr[0] == ACK))
	{
		cout << "Таймер настроен." << endl;

	}

	// Включаем модуль функциональных сигналов
	uint8_t module_func_remap_num = 5; // Порядковый номер модуля функциональных
сигналов
	ulp_clear_last_command();
	data = 0;  // 0 - Включить модуль
	ulp_send_message(handle, module_func_remap_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_STATE, &data, sizeof(data));
	ulp_exchange(handle, ATIME, false);
	if (lastCommand.ndata_bytes && (lastCommand.data_ptr[0] == ACK))
	{
		cout << "Модуль функциональных сигналов включен." << endl;

	}

	// Запрос списка входных сигналов и определение порядкового номера сигнала записи
в буфер
	ulp_clear_last_command();
	ulp_send_message(handle, module_func_remap_num, 0x17, NULL, 0);
	ulp_exchange(handle, ATIME, false);
	uint8_t commands_num ; // Количество сигналов
	uint8_t write_buffer_command_num; // Порядковый номер сигнала записи в буфер
	uint8_t buff_overflow_command_num; // Порядковый номер сигнала переполнения буфера
	if (lastCommand.ndata_bytes)
	{
		commands_num = lastCommand.ndata_bytes;
		cout << "Модуль поддерживает " << (int)commands_num<<" входных сигналов."<<
endl;
		for (int i = 0; i < lastCommand.ndata_bytes; i++)
		{
			if (lastCommand.data_ptr[i] ==
__ulp_m_func_remap::__IN_FUNCTION_ID::__LOG_WRITE)
			{
				write_buffer_command_num = i;
			}
			if (lastCommand.data_ptr[i] ==
__ulp_m_func_remap::__OUT_FUNCTION_ID::__LOG_OVERFLOW)
			{
				buff_overflow_command_num = i;
```

ОАО «СКБИС» 195009, Санкт-Петербург, Кондратьевский проспект, дом 2, литер А тел. (812) 334-17-72 доб.175, факс (812) 540-29-33 www.skbis.ru

```
                }

            }

            cout << "Порядковый номер сигнала записи в буфер:" <<
(int)write_buffer_command_num<<endl;
            cout << "Порядковый номер сигнала переполнения буфера:" <<
(int)buff_overflow_command_num << endl;


      }

      uint32_t time_to_read;
      cout << "Введите время чтения значений из буфера в секундах:";
      cin >> time_to_read;

      // Включаем функциональный сигнал, указываем таймер в качестве источника,
устанавливаем триггер на работу по обоим фронтам
      ulp_clear_last_command();
      //Создаем структуру настроек модуля функциональных сигналов. В нашем случае у ЛИР-
919 есть 12 входных сигналов, 19 выходных функциональных
      // сигналов и 4 байпаса
      __ulp_m_func_remap::__SETTINGS_TYPEDEF<12, 19, 4> func_remap_settings;

      // Настраиваем входной сигнал записи в буфер
      func_remap_settings.in[write_buffer_command_num].ena = true; // Включаем входящий
сигнал записи в буфер
      func_remap_settings.in[write_buffer_command_num].src.addr = 0; // Номер таймера,
созданного ранее
      func_remap_settings.in[write_buffer_command_num].src.src =
__ulp_m_virtual_io::__SOURCE::__TIMERS ; // Тип таймер
      func_remap_settings.in[write_buffer_command_num].src.trg =
__ulp_m_virtual_io::__TRIGGERS::__EDGE; // Работа по обоим фронтам

      // Настраиваем выходной сигнал переполнения буфера. При записи буфера будет
установлен выход номер 4 (загорится светодиод)
      func_remap_settings.out[buff_overflow_command_num].ena = true; // Включаем
      func_remap_settings.out[buff_overflow_command_num].dst.addr = 4; // Номер
      func_remap_settings.out[buff_overflow_command_num].dst.dst =
__ulp_m_virtual_io::__DESTINATION::__SELF_OUTPUTS; // Тип "собственные выходы" устройства



      ulp_send_message(handle, module_func_remap_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_SETTINGS, (uint8_t*)&func_remap_settings,
sizeof(__ulp_m_func_remap::__SETTINGS_TYPEDEF<12, 19, 4>));

      ulp_exchange(handle, ATIME, false);
      if (lastCommand.ndata_bytes && (lastCommand.data_ptr[0] == ACK))
      {
            cout << "Функциональный сигнал включен и настроен." << endl;

      }




      //Считываем значения из буфера

      uint32_t tiks  = GetTickCount();
      while ((GetTickCount()- tiks) < time_to_read*1000)
      {
```

```
                //cout << inBuff_num << endl;
                ulp_send_message(handle, __ulp_module::__TYPE::__SYSTEM,
__ulp_m_system::__CMD::__GET_LOG_DATA, NULL, 0);
                ulp_exchange(handle, 0, false);


        }

        //Выводим значения в файл
        ofstream f;
        f.open("result.txt", ios::out);
        for (int i = 0; i < buff_data.size();  i++)
        {
                f << i<< "\t"<<buff_data[i].coordinate << "\t"<<
buff_data[i].status.all<<endl;

        }
        f.close();


        ulp_disconnect_and_free(handle);


        return 0;
}
```

Пример 6. Работа с модулем позиционирования.

```
// PositionerSample.cpp
// Пример демонстрирует работу модуля позиционирования. В данном примере ЛИР-919
подключен к шаговому, двигателю, линейному датчику
//  и двум концевым выключателям
//

#include "stdafx.h"

#include "stdafx.h"
#include "ulp_lib.h"
#include "ulp_tools.h"
#include <iostream>
#include <conio.h>
#include "module_virtual_io_types.h"
#include  "module_func_remap_types.h"
#include "sub_module_sensor_inc_types.h"
#include "module_positioner_types.h"
#include "sub_module_positioner_discrete_types.h"
uint8_t DrawMainMenu()
{
	cout << endl;
	cout << "1. Поиск референтной метки" << endl;
	cout << "2. Переместиться в координату" << endl;
	cout << "3. Переместиться в координату со скоростью" << endl;
	cout << "4. Стоп" << endl;
	cout << "0. Выход" << endl;
	cout << "Выберите пункт:";
	uint8_t result;
	cin >> result;
	cout << endl;
	return result;

}

static const char* state_str[]{
	"STOP",
	"WAIT_AXIS_SEL_SIGNAL",
	"RUN",
	"STOPPING",
	"BRAKING" };
static const char* result_str[]{
		"WORKING ",
		"REF_SEARCHING",
		"MANUAL",
		"DONE",
		"POSITIONING_ERROR",
		"SENSOR_ERROR",
		"LIMITER_DETECT",
		"REF_NOT_CAPTURED",
		"STOP_CMD",
		"WRONG_DIRECTION",
		"NOT_MOVING",
		"INTERNAL_ERROR"};
static const char* direction_str[]{
		"LEFT",
		"RIGHT"};

bool GetResult(void* handle)
{
	__ulp_m_positioner::__RESULT::__TYPEDEF result;
	if (ulp_clear_send_exchange(handle, 6,
__ulp_m_positioner::__CMD::__GET_CURRENT_RESULT, 0, 0))
```

```
	{
		result = *((__ulp_m_positioner::__RESULT::__TYPEDEF*)lastCommand.data_ptr);
		cout << "Результат: " << result_str[result]<<endl;
		return true;
	}

	return false;
}
bool PositionerIndication(void* handle)
{
	char* clrstr = "\r
";
	cout << endl;
	__ulp_m_positioner::__STATE::__TYPEDEF state;
	__ulp_m_positioner::__MOVE_DIRECTION::__TYPEDEF direction;
	__ulp_m_positioner::__RESULT::__TYPEDEF result;
	uint16_t speed;
	do
	{


		if (ulp_clear_send_exchange(handle, 6,
__ulp_m_positioner::__CMD::__GET_CURRENT_STATE, 0, 0))
		{
			state =
*((__ulp_m_positioner::__STATE::__TYPEDEF*)lastCommand.data_ptr);
		}
		if (ulp_clear_send_exchange(handle, 6,
__ulp_m_positioner::__CMD::__GET_CURRENT_DIRECTION, 0, 0))
		{
			direction =
*((__ulp_m_positioner::__MOVE_DIRECTION::__TYPEDEF*)lastCommand.data_ptr);
		}
		if (ulp_clear_send_exchange(handle, 6,
__ulp_m_positioner::__CMD::__GET_CURRENT_RESULT, 0, 0))
		{
			result=
*((__ulp_m_positioner::__RESULT::__TYPEDEF*)lastCommand.data_ptr);
		}
		if (ulp_clear_send_exchange(handle, 6,
__ulp_m_positioner::__CMD::__GET_CURRENT_SPEED, 0, 0))
		{
			speed= *(uint16_t*)lastCommand.data_ptr;
		}


		cout << clrstr;
		cout << "\r" << "Статус: " << state_str[state] << " Направление: " <<
direction_str[direction] << " Результат: " << result_str[result] << " Скорость: " <<
speed;


	}  while (state != __ulp_m_positioner::__STATE::__STOP)
	;
	cout << endl<<endl;

	return true;
}

int _tmain(int argc, _TCHAR* argv[])
{
	setlocale(LC_ALL, "Russian");
	void* handle = NULL;
	_ulp_link::__CONNECTION_TYPE_TYPEDEF ctype = _ulp_link::__TYPE_USB;
```

```
	// Выделяем память и подключаемся к устройству с номером 0.
	if (ulp_create_and_connect(ctype, 0, &handle)) cout << "Создание и подключение к
устройству: OK !" << endl;
	else
	{
		cout << "Ошибка подключения!";
		return 1;
	}

	// Устанавливаем функцию обратного вызыва для приема сообщений от устройства
	ulp_set_message_callback(handle, callback_handler);


	// Включаем модуль датчика
	uint8_t data = 0;
	ulp_send_message(handle, 1, __ulp_module::__SHARED_CMD::__SET_MODULE_STATE, &data,
sizeof(data));
	ulp_exchange(handle, ATIME, false);
	if (lastCommand.ndata_bytes && (lastCommand.data_ptr[0] == ACK))
	{
		cout << "Модуль датчика включен. " << endl;
	}

	__ulp_m_sensor::__SETTINGS_TYPEDEF sensor_settings;
	sensor_settings.axis_type = __ulp_m_sensor::__AXIS::__LINEAR;
	sensor_settings.is_reverse = true;
	if (ulp_clear_send_exchange(handle, 1,
__ulp_module::__SHARED_CMD::__SET_MODULE_SETTINGS, (uint8_t* )&sensor_settings,
sizeof(sensor_settings))
		&& (lastCommand.data_ptr[0] == ACK))
	{
		cout << "Настройки датчика установлены." << endl;
	}


	// Включаем модуль входов/выходов
	uint8_t io_module_num = 3;
	uint8_t  state = 0;
	if (ulp_clear_send_exchange(handle, io_module_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_STATE, &state, sizeof(state))
		&& (lastCommand.data_ptr[0] == ACK))
	{
		cout << "Модуль входов/выходов включен." << endl;
	}

	// Включаем модуль виртуальных входов/выходов
	uint8_t virtual_io_module_num = 4; // Порядковый номер модуля виртуальных
входов/выходов
	state = 0;
	if (ulp_clear_send_exchange(handle, virtual_io_module_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_STATE, &state, sizeof(state))
		&& (lastCommand.data_ptr[0] == ACK))
	{
		cout << "Модуль виртуальных входов/выходов включен." << endl;
	}

	// Создаем структуру, содержащую 4 таймера, 8 виртуальных входов и 8 виртуальных
выходов
	__ulp_m_virtual_io::__SETTINGS_TYPEDEF<4, 8, 8> v_io_settings;

	v_io_settings.inputs[__ulp_m_virtual_io::__SOURCE::__SELF_INPUTS].inverse |= 1 <<
0; // 0- левый концевой выключатель
```

```
	v_io_settings.inputs[__ulp_m_virtual_io::__SOURCE::__SELF_INPUTS].inverse |= 1 <<
1; // 1- правый концевой выключатель

	v_io_settings.outputs[__ulp_m_virtual_io::__DESTINATION::__SELF_OUTPUTS].inverse
|= 1 << 1; // 1 - направление вращения драйвера шагового двигателя
	v_io_settings.outputs[__ulp_m_virtual_io::__DESTINATION::__SELF_OUTPUTS].inverse
|= 1 << 2; // 2 - сигнал Enable драйвера шагового двигателя

	v_io_settings.timers[0].max_cntr = 10;
	v_io_settings.timers[0].pulse = false;

	v_io_settings.timers[1].max_cntr = 5;
	v_io_settings.timers[1].pulse = false;

	v_io_settings.timers[2].max_cntr = 1;
	v_io_settings.timers[2].pulse = false;

	v_io_settings.log_outputs[0].ena = true;
	v_io_settings.log_outputs[0].dst.dst =
__ulp_m_virtual_io::__DESTINATION::__SELF_OUTPUTS;
	v_io_settings.log_outputs[0].dst.addr = 0;
	v_io_settings.log_outputs[0].logic = __ulp_m_virtual_io::__LOGIC_FUNCTION::__AND;

	v_io_settings.log_outputs[1].ena = true;
	v_io_settings.log_outputs[1].dst.dst =
__ulp_m_virtual_io::__DESTINATION::__SELF_OUTPUTS;
	v_io_settings.log_outputs[1].dst.addr = 0;
	v_io_settings.log_outputs[1].logic = __ulp_m_virtual_io::__LOGIC_FUNCTION::__AND;

	v_io_settings.log_outputs[2].ena = true;
	v_io_settings.log_outputs[2].dst.dst =
__ulp_m_virtual_io::__DESTINATION::__SELF_OUTPUTS;
	v_io_settings.log_outputs[2].dst.addr = 0;
	v_io_settings.log_outputs[2].logic = __ulp_m_virtual_io::__LOGIC_FUNCTION::__AND;

	if (ulp_clear_send_exchange(handle, virtual_io_module_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_SETTINGS, (uint8_t*)&v_io_settings,
sizeof(__ulp_m_virtual_io::__SETTINGS_TYPEDEF<4, 8, 8>))
		&& (lastCommand.data_ptr[0] == ACK))
	{
		cout << "Произведены настройки модуля виртуальных входов/выходов." << endl;
	}


	// Включаем модуль функциональных сигналов
	uint8_t module_func_remap_num = 5; // Порядковый номер модуля функциональных
сигналов
	state = 0;
	if (ulp_clear_send_exchange(handle, module_func_remap_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_STATE, &state, sizeof(state))
		&& (lastCommand.data_ptr[0] == ACK))
	{
		cout << "Модуль функциональных сигналов включен." << endl;
	}

	//Создаем структуру настроек модуля функциональных сигналов. В нашем случае у ЛИР-
919 есть 12 входных сигналов, 19 выходных функциональных
	// сигналов и 4 байпаса
	__ulp_m_func_remap::__SETTINGS_TYPEDEF<12, 19, 4> func_remap_settings;

	func_remap_settings.in[3].ena = true;		// 3 - правый концевой выключатель
	func_remap_settings.in[3].src.src = __ulp_m_virtual_io::__SOURCE::__SELF_INPUTS;
	func_remap_settings.in[3].src.addr = 1; // выключатель подключен к 1-му входу
```

```
	func_remap_settings.in[4].ena = true;		// 4 - левый концевой выключатель
	func_remap_settings.in[4].src.src = __ulp_m_virtual_io::__SOURCE::__SELF_INPUTS;
	func_remap_settings.in[4].src.addr = 0; // выключатель подключен к 0-му входу

	func_remap_settings.ic[0].ena = true;
	func_remap_settings.ic[0].src.src = __ulp_m_virtual_io::__SOURCE::__TIMERS;
	func_remap_settings.ic[0].src.addr = 0; // используем 0-й таймер
	func_remap_settings.ic[0].src.trg = __ulp_m_virtual_io::__TRIGGERS::__HIGHL_LEVEL;
	func_remap_settings.ic[0].dst.dst =
__ulp_m_virtual_io::__DESTINATION::__LOG_OUTPUTS;
	func_remap_settings.ic[0].dst.addr = 8; // Перенаправляем сигнал с таймера 0 на 2-
й вход логического выхода номер 0, который имеет индекс 8

	func_remap_settings.out[8].ena = true;  // 8 - порядковый номер выходного сигнала
"первая скорость мотора"
	func_remap_settings.out[8].dst.dst =
__ulp_m_virtual_io::__DESTINATION::__LOG_OUTPUTS;
	func_remap_settings.out[8].dst.addr = 0; // Подключаем выходной сигнал "первая
скорость мотора" к первому входу логического выхода номер 0

	func_remap_settings.ic[1].ena = true;
	func_remap_settings.ic[1].src.src = __ulp_m_virtual_io::__SOURCE::__TIMERS;
	func_remap_settings.ic[1].src.addr = 1; // используем 1-й таймер
	func_remap_settings.ic[1].src.trg = __ulp_m_virtual_io::__TRIGGERS::__HIGHL_LEVEL;
	func_remap_settings.ic[1].dst.dst =
__ulp_m_virtual_io::__DESTINATION::__LOG_OUTPUTS;
	func_remap_settings.ic[1].dst.addr = 9; // Перенаправляем сигнал с таймера 1 на 2-
й вход логического выхода номер 1, который имеет индекс 9

	func_remap_settings.out[9].ena = true;  // 9 - порядковый номер выходного сигнала
"вторая скорость мотора"
	func_remap_settings.out[9].dst.dst =
__ulp_m_virtual_io::__DESTINATION::__LOG_OUTPUTS;
	func_remap_settings.out[9].dst.addr = 1; // Подключаем выходной сигнал "вторая
скорость мотора" к первому входу логического выхода номер 1

	func_remap_settings.ic[2].ena = true;
	func_remap_settings.ic[2].src.src = __ulp_m_virtual_io::__SOURCE::__TIMERS;
	func_remap_settings.ic[2].src.addr = 2; // используем 2-й таймер
	func_remap_settings.ic[2].src.trg = __ulp_m_virtual_io::__TRIGGERS::__HIGHL_LEVEL;
	func_remap_settings.ic[2].dst.dst =
__ulp_m_virtual_io::__DESTINATION::__LOG_OUTPUTS;
	func_remap_settings.ic[2].dst.addr = 10; // Перенаправляем сигнал с таймера 2 на
2-й вход логического выхода номер 2, который имеет индекс 10

	func_remap_settings.out[10].ena = true;  // 10 - порядковый номер выходного
сигнала "третья скорость мотора"
	func_remap_settings.out[10].dst.dst =
__ulp_m_virtual_io::__DESTINATION::__LOG_OUTPUTS;
	func_remap_settings.out[10].dst.addr = 2; // Подключаем выходной сигнал "третья
скорость мотора" к первому входу логического выхода номер 2

	func_remap_settings.out[3].ena = true; // 3- порядковый номер выходного сигнала
"мотор включен"
	func_remap_settings.out[3].dst.dst =
__ulp_m_virtual_io::__DESTINATION::__SELF_OUTPUTS;
	func_remap_settings.out[3].dst.addr = 2; // Подключаем сигнал "мотор включен ко 2-
му физическому выходу устройства

	func_remap_settings.out[5].ena = true; // 5- порядковый номер выходного сигнала
"направление мотора"
	func_remap_settings.out[5].dst.dst =
__ulp_m_virtual_io::__DESTINATION::__SELF_OUTPUTS;
```

```
	func_remap_settings.out[5].dst.addr = 1; // Подключаем сигнал "мотор включен к 1-
му физическому выходу устройства

	if (ulp_clear_send_exchange(handle, module_func_remap_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_SETTINGS, (uint8_t*)&func_remap_settings,
sizeof(__ulp_m_func_remap::__SETTINGS_TYPEDEF<12, 19, 4>)))
	{
		cout << "Функциональные сигналы настроены" << endl;
	}

	// Включаем модуль позиционирования
	uint8_t module_potitioner_num = 6; // Порядковый номер модуля позиционирования
	state = 0;
	if (ulp_clear_send_exchange(handle, module_potitioner_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_STATE, &state, sizeof(state))
		&& (lastCommand.data_ptr[0] == ACK))
	{
		cout << "Модуль позиционирования включен." << endl;
	}

	uint8_t pos_submod_num = 0;
	if (ulp_clear_send_exchange(handle, module_potitioner_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_STATE, &pos_submod_num, sizeof(pos_submod_num))
		&& (lastCommand.data_ptr[0] == ACK))
	{
		cout << "Подмодуль модуля позиционирования выбран." << endl;
	}


	__ulp_m_positioner::__SETTINGS_TYPEDEF positioner_settings;

	//cout << sizeof(positioner_settings) << endl;

	positioner_settings.positioning_tolerance = 5; // Допуск позиционирования 5 ед
перемещения
	positioner_settings.stop_detect_range = 2; // Изменение координаты, считаемой
перемещением в заданный промежуток времени
	positioner_settings.stop_detect_time = 500; // Промежуток времени для расчета
stop_detect_range (мс)
	positioner_settings.backlash_timeout = 10000; // Максимальное время операции
выборки люфта (мс)
	positioner_settings.max_speed = 3; // Ограничение по максимальной скорости
	positioner_settings.backlash_speed = 1; //  Скорость выборки люфта
	positioner_settings.manual_speed = 2; // Скорость перемещения при  ручном
управлении
	positioner_settings.backlash = true; // Выбирать люфт
	positioner_settings.max_sens_series_error = 10; // Критическое количество
последовательных ошибок датчика
	positioner_settings.ena_axis_signal = false; // Сигнал выбора оси
	//Первое действие автоматического поиска референтной метки
	positioner_settings.ref_cmd[0].cmd =
__ulp_m_positioner::__REF_SEARCH_ACTION::__MOVING_LEFT; // Направление поиска влево
	positioner_settings.ref_cmd[0].capture =
__ulp_m_positioner::__REF_SEARCH_CAPTURE::__CAPTURE_NOT; // Не захватываем референтной
метки
	positioner_settings.ref_cmd[0].speed = 1;

	//Второе действие автоматического поиска референтной метки.
	positioner_settings.ref_cmd[1].cmd =
__ulp_m_positioner::__REF_SEARCH_ACTION::__SET_OFFSET; // Обнуляем показание датчика
(Используем инкрементный датчик без Р.М.)
```

```
	if (ulp_clear_send_exchange(handle, module_potitioner_num,
__ulp_module::__SHARED_CMD::__SET_MODULE_SETTINGS, (uint8_t*)&positioner_settings,
sizeof(__ulp_m_positioner::__SETTINGS_TYPEDEF))
		&& (lastCommand.data_ptr[0] == ACK))
	{
		cout << "Модуль позиционирования настроен." << endl;
	}

	__ulp_sm_positioner_discrete::__SETTINGS_TYPEDEF sm_positioner_settings;

	sm_positioner_settings.stop_zone[0] = 0; // Зона тормоза
	sm_positioner_settings.stop_zone[1] = 1; // Зона торможения 1
	sm_positioner_settings.stop_zone[2] = 50; // Зона торможения 2
	sm_positioner_settings.stop_zone[3] = 200; // Зона торможения 3

	if (ulp_clear_send_exchange(handle, module_potitioner_num,
__ulp_module::__SHARED_CMD::__SET_SUB_MODULE_SETTINGS, (uint8_t*)&sm_positioner_settings,
sizeof(__ulp_sm_positioner_discrete::__SETTINGS_TYPEDEF))
		&& (lastCommand.data_ptr[0] == ACK))
	{
		cout << "Подмодуль модуля позиционирования настроен." << endl;
	}


	uint8_t input = 0;

	while ((input = DrawMainMenu()) != '0')
	{
		switch (input)
		{
		case '1':
		{
				if (ulp_clear_send_exchange(handle,
module_potitioner_num, __ulp_m_positioner::__CMD::__START_REF_SEARCH, NULL, 0)
					&& (lastCommand.data_ptr[0] == ACK))
				{
					cout << "Запущен поиск референтной метки." <<
endl;
					PositionerIndication(handle);
				}
				else
				{
					cout << "Ошибка !" << endl;
					GetResult(handle);
				}



				break;
		}
		case '2':
		{
				cout << "Введите координату для перемещения:";
				int64_t dest;
				cin >> dest;

				if (ulp_clear_send_exchange(handle,
module_potitioner_num, __ulp_m_positioner::__CMD::__MOVE_TO, (uint8_t*)&dest,
sizeof(dest))
					&& (lastCommand.data_ptr[0] == ACK))
				{
					cout << "Запущено перемещение. " << endl;
					PositionerIndication(handle);
```

```
                                }
                                else
                                {
                                        cout << "Ошибка !" << endl;
                                        GetResult(handle);
                                }

                                break;
                }

                case '3':
                {

                                __ulp_m_positioner::__MOVE_W_SPEED_CMD_DATA_TYPEDEF
dest_speed;

                                cout << "Введите координату для перемещения:";
                                int64_t dest;
                                cin >> dest_speed.coordinate;
                                uint16_t speed;
                                cout << "Задайте максимальную скорость:";
                                cin >> dest_speed.speed;

                                if (ulp_clear_send_exchange(handle,
module_potitioner_num, __ulp_m_positioner::__CMD::__MOVE_TO_W_SPEED,
(uint8_t*)&dest_speed, sizeof(dest_speed))
                                        && (lastCommand.data_ptr[0] == ACK))
                                {
                                        cout << "Запущено перемещение. " << endl;
                                        PositionerIndication(handle);

                                }
                                else
                                {
                                        cout << "Ошибка !" << endl;
                                        GetResult(handle);
                                }

                                break;
                }

                case '4':
                {
                                if (ulp_clear_send_exchange(handle,
module_potitioner_num, __ulp_m_positioner::__CMD::__STOP, 0, 0)
                                        && (lastCommand.data_ptr[0] == ACK))
                                {
                                        cout << "Остановлено ! " << endl;


                                }
                                else
                                {
                                        cout << "Ошибка !" << endl;
                                        GetResult(handle);
                                }
                }

                        /*
                case '5':
                {
                                if (ulp_clear_send_exchange(handle, 1,
__ulp_m_sensor::__CMD::__SAVE_CORRECTION_TABLE, NULL, 0) && (lastCommand.data_ptr[0] =
ACK))
```

```
                                        cout << "Таблица коррекции успешно сохранена в
ПЗУ." << endl;
                                break;
                }
                case '6':
                {
                                if (ulp_clear_send_exchange(handle, 1,
__ulp_m_sensor::__CMD::__ABORT_CORRECTION_PROCESS, NULL, 0) && (lastCommand.data_ptr[0] =
ACK))
                                        cout << "Таблица коррекции успешно загружена из
ПЗУ." << endl;
                                break;
                }*/
                default:
                        break;
                }

        }



        ulp_disconnect_and_free(handle);


        return 0;
}
```