Использование библиотеки RI_SDK для взаимодействия с электронными компонентами и периферическими устройствами

Цели преследуемые при разработке архитектуры библиотеки (Advantages and Goals)

1. Создать кроссплатформенную библиотеку для взаимодействия с электронными компонентами (Пользователь библиотеки не должен писать платформо-зависимый код)
2. Библиотека не должна зависеть от архитектуры процессора, и должна одинаково корректно работать как на ARM, так и на amd64 и i386.
3. Библиотека должна идеально вписываться в любой стек разработки, будь то Python или например Java, или приложение на PHP.
4. Библиотека должна работать быстрее аналогов на js и python.
5. Без лишних абстракций.
    a. Нет лишней связанности
    b. Принцип “Разделяй и властвуй” Отдельный компонент на каждую сущность (Компонент на шину, Компонент на датчик, Компонент на конвертер сигнала)
6. Каждый пользователь библиотеки может расширять ее возможности используя с помощью плагинов (Например интеграция драйверов для своих устройств)
7. Библиотека должна потреблять минимальное кол-во памяти и ресурсов, чтобы обеспечить возможность её запуска на устройствах IoT (Только когда конкретный драйвер используется, под него выделяется память)
8. Предоставить комплексное решение со своим дизайном и философией, удобством использования, а не просто библиотека которая решает задачу подключения конкретного компонента в конкретных условиях
9. Нацелена на длительное использование и поддержку
10. Скромный порог вхождения
    a. Использование в образовательных организациях для обучения
    b. Статическая типизация
    c. Необходимо минимальное кол-во кода для решения задачи
11. Возможность асинхронного взаимодействия с компонентами
12. Каждый драйвер реализует и предоставляет все возможности устройства.
13. Возможность портативного использования библиотеки, без необходимости установки.

Примеры программ

PHP

Установка PHP

Linux

Идем на сайт php.net и скачиваем оттуда последний релиз. На момент написания статьи 8.0.12. Скачанный архив можно распаковать командой:

```
tar -xzf php-8.0.12.tar.gz
```

Переходим в распакованную директорию

```
cd php-8.0.12
```

Следующим шагом перед установкой выполняем конфигурацию gcc с указанием расширений которые мы хотим использовать в дальнейшем. Здесь перечислены часто используемые расширения. Главное расширение, которое обязательно должно быть это FFI (флаг --with-ffi)

```
./configure --with-openssl --enable-mbstring --with-curl --enable-calendar
--enable-bcmath --enable-exif --enable-gd --enable-ftp --enable-pcntl --enable-sockets
--with-sodium --with-zip --with-pear --enable-mysqlnd --with-pdo-mysql
--with-pgsql=/usr/bin/pg_config --with-pdo-pgsql=/usr/bin/pg_config
--with-config-file-scan-dir=/etc/php/8.0
--with-config-file-path=/etc/php/8.0/cli/php.ini --with-zlib --with-ffi
```

Затем проверяем во сколько потоков может происходить компиляция на вашей машине

```
nproc
```

Для более быстрой компиляции вызываем команду make с флагом -j и указываем кол-во потоков сразу после флага. К примеру у меня на машине 12 потоков:

```
make -j12
```

Выполняем скрипт установки

```
sudo make install
```

После завершения установки проверяем результат командой:

```
php -v
```

Windows

Идем на сайт php.net, выбираем Windows downloads и скачиваем Zip релиз. На момент написания статьи 8.0.12.

Распаковываем скачанный архив командой.

```
mkdir php-8.0.12 && tar -x -f php-8.0.12-nts-Win32-vs16-x64.zip -C php-8.0.12
```

Переходим в распакованную директорию

```
cd php-8.0.12
```

Создайте php.ini для конфигурации из готового шаблона.

```
copy php.ini-development php.ini
```

Откройте php.ini через текстовый редактор и раскомментируйте следующие строки

1. extension_dir = "ext"
2. extension=ffi

Измените значение переменной `short_open_tag=Off` на `On`
Добавьте в переменную окружения PATH путь к папке `php-8.0.12`.

Подключение RI_SDK к приложению на PHP

С помощью расширения FFI подключаем распределенную библиотеку risdk, для этого нам в конструкторе cdef нужно указать пути к файлам .so или .dll и .h
Ниже представлен пример подключения библиотеки, инициализации SDK и создания компонента:

Создадим php файл

Команда для Linux

```
touch shared_client.php
```

Команда для Windows

```
type nul > shared_client.php
```

Структура проекта

. workspace/
--- > shared_client.php
--- > librisdk.h
--- > librisdk.so/.dll

Подключим библиотеку
Обратите внимание на значение переменной $RELATIVE_PATH. Это относительный путь для поиска библиотеки, если она лежит ниже чем php скрипт.
Например при значении '/..' поиск библиотеки будет выполняться на одну директорию ниже.
При значении '' файлы librisdk.h и librisdk.so должны располагаться в той же директории что и php скрипт.

```
<?
$RELATIVE_PATH = '';
$headers = file_get_contents(__DIR__ . $RELATIVE_PATH . '/librisdk.h');
```

```
// Вырежем лишнии заголовки
$headers = preg_replace(['/#ifdef __cplusplus\s*extern "C" {\s*#endif/i', '/#ifdef
__cplusplus\s*}\s*#endif/i'], '', $headers);
$ffi = FFI::cdef($headers, __DIR__ . $RELATIVE_PATH . '/librisdk.so');

print("Success \n");

?>
```

Для запуска скрипта выполним в терминале следующую команду

```
php shared_client.php
```

Создадим компонент

```
$logLevel = 2;
$errorText = $ffi->new('char[1000]', 0); // Выделяем память на строку с ошибкой

// Инициализируем SDK
$errCode = $ffi->RI_SDK_InitSDK($logLevel, $errorText);
if ($errCode) {
    // Выведем текст и код ошибки в терминал, при возникновении
    print("RI_SDK_InitSDK errorText:" . FFI::string($errorText) . " errCode: " .
$errCode . " \n");
    return $errCode;
}

// Создаем базовый компонент
$descriptor = $ffi->new('int', 0); // Выделяем память на переменную с номером
дескриптора
$errCode = $ffi->RI_SDK_CreateBasic(FFI::addr($descriptor), $errorText);
if ($errCode) {
    // Выведем текст и код ошибки в терминал, при возникновении
    print("RI_SDK_CreateBasic errorText:" . FFI::string($errorText). " errCode: " .
$errCode . " \n");
    return $errCode;
}

// Выведем полученный дескриптор компонента
print('descriptor: ' . + $descriptor->cdata . " \n");
```

Запустим скрипт. Результат успешной работы должен выглядеть следующим образом

```
-bash#$ php shared_client.php
descriptor: 1
Success
-bash#$
```

Пример программы на PHP использующий RI_SDK

Описание

Данная программа является примером, использования библиотеки. Она реализует простой сценарий поведения робота. Сперва все сервоприводы встают в условное стартовое положение. После робот берет кубик с позиции слева и перемещает его на позицию справа. После сервоприводы возвращаются в первоначальное положение.
Одновременно с движением робота, происходит свечение светодиода. Светодиод горит красным цветом пока сервоприводы приводятся к стартовому положению, после загорается зеленым. Далее светодиоды моргает зеленым при передвижении тела робота и мерцает синим, пока робот кладет/берет кубик.

Структура проекта

. workspace/
    --- > demo.php
    --- > librisdk.h
    --- > librisdk.dll/.so
    --- > CH341DLL.DLL/ CH341DLLA64.DLL
    --- > SLABHIDDevice.dll
    --- > SLABHIDtoSMBus.dll

Реализация

Реализация данной программы представлена ниже. Ее также можно скачать по ссылке на репозиторий. Вы можете скопировать данный пример и запустить у себя.

```
<?
$GLOBALS = ['body_start_pulse' => 1500, // стартовая позиция тела
    'arrowR_start_pulse' => 2000,       // стартовая возиция правой стрелы
    'arrowL_start_pulse' => 1000,       // стартовая позиция левой стрелы
    'claw_start_pulse' => 1000,         // стартовая позиция клешни
    'claw_rotate_start_pulse' => 1500,  // стартовая позиция поворота клешни

    'arrowR_over_cube_position' => -30, // позиция правой стрелы над кубиком
    'arrowL_over_cube_position' => -10,  // позиция левой стрелы над кубиком
    'claw_unclenched_position' => 80,   // позиция открытой клешни
    'arrowR_cube_position' => -75,      // позиция правой стрелы на месте кубика
    'arrowL_cube_position' => 55,       // позиция левой стрелы на месте кубика
];

// структура дескрипторов устройства
class device
{
    public $i2c;        // дескриптор i2c
    public $pwm;        // дескриптор pwm
    public $body;       // дескриптор сервопривода тела
```

```
    public $claw;       // дескриптор сервопривода клешни
    public $arrowR;     // дескриптор сервопривода правой стрелы
    public $arrowL;     // дескриптор сервопривода левой стрелы
    public $clawRotate; // дескриптор сервопривода поворота клешни
    public $led;        // дескриптор светодиода

    public function __construct($i2c, $pwm, $body, $claw, $arrowR, $arrowL,
$clawRotate, $led)
    {
        $this->i2c = $i2c;
        $this->pwm = $pwm;
        $this->body = $body;
        $this->claw = $claw;
        $this->arrowR = $arrowR;
        $this->arrowL = $arrowL;
        $this->clawRotate = $clawRotate;
        $this->led = $led;
    }
}

// для хранения дескриптоов сервоприводов
class servo
{
    public $descriptor;   // дескриптор сервопривода
    public $startPosition; // стартовая позиция сервопривода

    public function __construct($descriptor, $startPosition)
    {
        $this->descriptor = $descriptor;
        $this->startPosition = $startPosition;
    }
}

// Обратите внимание на значение переменной $RELATIVE_PATH
// Это относительный путь для поиска библиотеки, если она лежит ниже чем php скрипт
// Например при значении '/..' поиск библиотеки будет выполняться на одну директорию
ниже
// При значении '' файлы librisdk.h и librisdk.so должны располагаться в той же
директории что и php скрипт
$RELATIVE_PATH = '';
$headers = file_get_contents(__DIR__ . $RELATIVE_PATH . '/librisdk.h');
// Вырежем лишнии заголовки
$headers = preg_replace(['/#ifdef __cplusplus\s*extern "C" {\s*#endif/i', '/#ifdef
__cplusplus\s*}\s*#endif/i'], '', $headers);
$ffi = FFI::cdef($headers, __DIR__ . $RELATIVE_PATH . '/librisdk.dll');
$errorText = $ffi->new('char[1000]', 0); // Выделяем память на строку с ошибкой.
Передается как входной параметр,при возникновении ошибки в эту переменную будет записан
текст ошибки

// инициализация структуры устройства
$robohand = new Device($ffi->new('int', 0), $ffi->new('int', 0), $ffi->new('int', 0),
$ffi->new('int', 0), $ffi->new('int', 0),
```

```
    $ffi->new('int', 0), $ffi->new('int', 0), $ffi->new('int', 0));

// массив дескрипторов сервоприводов
$servos = [
    new servo($robohand->body, $GLOBALS['body_start_pulse']),
    new servo($robohand->claw, $GLOBALS['claw_start_pulse']),
    new servo($robohand->arrowR, $GLOBALS['arrowR_start_pulse']),
    new servo($robohand->arrowL, $GLOBALS['arrowL_start_pulse']),
    new servo($robohand->clawRotate, $GLOBALS['claw_rotate_start_pulse'])
];

$errCode = start($ffi, $servos, $robohand, $errorText);
if ($errCode) {
    print("ErrorText: " . FFI::string($errorText) . " ErrCode: " . $errCode . " \n");

}

// start - запускает демо программу
function start($ffi, $servos, $robohand, $errorText)
{
    $speed = 100;  // скорость в градусах в секунду

    // initDevice - инициализация библиотеки и устройств
    $errCode = initDevice($ffi, $servos, $robohand, $errorText);
    if ($errCode) {
        return $errCode;
    }

    // startPositionAllServo - переводит все сервоприводы в стартовую позицию
    $errCode = startPositionAllServo($ffi, $robohand, $servos, $errorText);
    if ($errCode) {
        return $errCode;
    }

    // Двигаем тело к местонахождению кубика
    $errCode = rotateBody($ffi, $robohand, 45, $speed, $errorText);
    if ($errCode) {
        return $errCode;
    }

    // Берем кубик
    $errCode = get($ffi, $robohand, $speed, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //Двигаем тело к новому местонахождению кубика
    $errCode = rotateBody($ffi, $robohand, -90, $speed, $errorText);
    if ($errCode) {
        return $errCode;
    }
```

```
    //Кладем кубик
    $errCode = put($ffi, $robohand, $speed, $errorText);
    if ($errCode) {
        return $errCode;
    }

    // Возвращаем тело в стартовое положение
    $errCode = rotateBody($ffi, $robohand, 45, $speed, $errorText);
    if ($errCode) {
        return $errCode;
    }

    // Уничтожаем компоненты и библиотеку
    $errCode = destruct($ffi, $robohand, $servos, $errorText);
    if ($errCode) {
        return $errCode;
    }

    print("Success \n");

    return 0;
}

// get - берет кубик
function get($ffi, $robohand, $speed, $errorText)
{
    //выполняем мерцание светодиодом,передаем дескриптор светодиода,3 параметра
цвета(RGB),продолжительность,кол-во повторений и включаем асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_RGB_LED_Flicker($robohand->led->cdata, 0, 0, 255, 500,
0, true, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->arrowR->cdata,
$GLOBALS['arrowR_over_cube_position'], $speed, false, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->arrowL->cdata,
$GLOBALS['arrowL_over_cube_position'], $speed, false, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
```

```
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->claw->cdata,
$GLOBALS['claw_unclenched_position'], $speed, false, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->arrowR->cdata,
($GLOBALS['arrowR_cube_position'] - $GLOBALS['arrowR_over_cube_position']), $speed,
false, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->arrowL->cdata,
($GLOBALS['arrowL_cube_position'] - $GLOBALS['arrowL_over_cube_position']), $speed,
false, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->claw->cdata, (-1) *
$GLOBALS['claw_unclenched_position'], $speed, false, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->arrowR->cdata, (-1) *
$GLOBALS['arrowR_cube_position'], $speed, false, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->arrowL->cdata, (-1) *
$GLOBALS['arrowL_cube_position'], $speed, false, $errorText);
    if ($errCode) {
        return $errCode;
    }

    return 0;
}

// rotateBody - вращение тела в указанный угол
```

```
function rotateBody($ffi, $robohand, $angle, $speed, $errorText)
{
    //выполняем мигание с заданной частотой,передаем дескриптор светодиода,3 параметра цвета(RGB),частоту,продолжительность и включаем асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_RGB_LED_FlashingWithFrequency($robohand->led->cdata, 0, 255, 0, 5, 0, true, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор тела,угол,скорость и асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->body->cdata, $angle, $speed, false, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->clawRotate->cdata, 45, $speed, false, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->clawRotate->cdata, -45, $speed, false, $errorText);
    if ($errCode) {
        return $errCode;
    }

    return 0;
}


// initServos - создает сервоприводы и линкует их
function initServos($ffi, $robohand, $servos, $errorText)
{
    //создаем 5 сервоприводов и линкуем их к пинам 0-4
    for ($i = 0; $i < 5; $i++) {
        // создаем компонент сервопривода с конкретной моделью как исполняемое устройство и получаем дескриптор сервопривода
        $errCode = $ffi->RI_SDK_CreateModelComponent("executor", "servodrive", "mg90s", FFI::addr($servos[$i]->descriptor), $errorText);
        if ($errCode) {
            return $errCode;
        }
        //связываем сервопривод с ШИМ,передаем дескриптор сервопривода и шим
        $errCode =
```

```
$ffi->RI_SDK_LinkServodriveToController($servos[$i]->descriptor->cdata,
$robohand->pwm->cdata, $i, $errorText);
        if ($errCode) {
            return $errCode;
        }
    }

    return 0;
}

// initDevice - инициализация библиотеки и устройств
function initDevice($ffi, $servos, $robohand, $errorText)
{
    $logLevel = 2; // уровень логирования

    // Инициализируем SDK
    $errCode = $ffi->RI_SDK_InitSDK($logLevel, $errorText);
    if ($errCode) {
        return $errCode;
    }

    // создаем компонент ШИМ с конкретной моделью как исполняемое устройство,получаем
дескриптор сервопривода
    $errCode = $ffi->RI_SDK_CreateModelComponent("connector", "pwm", "pca9685",
FFI::addr($robohand->pwm), $errorText);
    if ($errCode) {
        return $errCode;
    }

   // создаем компонент i2c адатера
    // Здесь осуществлен примитивное определение подключенной модели адаптера
    // Сначала пробуем создать i2c адаптер модели ch341 и связать с ним ШИМ
    $errCode = $ffi->RI_SDK_CreateModelComponent("connector", "i2c_adapter", "ch341",
FFI::addr($robohand->i2c), $errorText);
    if ($errCode) {
        return $errCode;
    }

    //связываем i2c адаптер с ШИМ по адресу 0x40
    $errCode = $ffi->RI_SDK_LinkPWMToController($robohand->pwm->cdata,
$robohand->i2c->cdata, 0x40, $errorText);
    if ($errCode) {
        // Если не получается то пробуем создать i2c адаптер модели cp2112
        $errCode = $ffi->RI_SDK_CreateModelComponent("connector", "i2c_adapter",
"cp2112", FFI::addr($robohand->i2c), $errorText);
        if ($errCode) {
            return $errCode;
        }

        $errCode = $ffi->RI_SDK_LinkPWMToController($robohand->pwm->cdata,
$robohand->i2c->cdata, 0x40, $errorText);
        if ($errCode) {
```

```
            return $errCode;
        }
    }

    // создаем компонент светодиода с конкретной моделью (ky016) как исполняемое
устройство и получаем дескриптор светодиода
    $errCode = $ffi->RI_SDK_CreateModelComponent("executor", "led", "ky016",
FFI::addr($robohand->led), $errorText);
    if ($errCode) {
        return $errCode;
    }

    //связываем светодиод адаптер с ШИМ,передаем значения трех пинов к которым
подключен светодиод
    $errCode = $ffi->RI_SDK_LinkLedToController($robohand->led->cdata,
$robohand->pwm->cdata, 15, 14, 13, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //инициализируем сервоприводы
    $errCode = initServos($ffi, $robohand, $servos, $errorText);
    if ($errCode) {
        return $errCode;
    }

    return 0;
}

// startPositionAllServo - переводит все сервоприводы в стартовую позицию
function startPositionAllServo($ffi, $robohand, $servos, $errorText)
{
//выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3 параметра
цвета(RGB), и включаем асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_RGB_LED_SinglePulse($robohand->led->cdata, 255, 0, 0,
0, true, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //приводим сервоприводы в стартовое положение
    for ($i = 0; $i < 5; $i++) {
        //выполняем поворот сервопривода в заданный угол,передаем дескриптор
сервоприовда,значение угла
        $errCode =
$ffi->RI_SDK_exec_ServoDrive_TurnByPulse($servos[$i]->descriptor->cdata,
$servos[$i]->startPosition, $errorText);
        if ($errCode) {
            return $errCode;
        }
        sleep(0.5);
    }
```

```
    //выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3 параметра цвета(RGB), и включаем асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_RGB_LED_SinglePulse($robohand->led->cdata, 0, 255, 0, 0, true, $errorText);
    if ($errCode) {
        return $errCode;
    }
    sleep(0.5);
    return 0;
}

// put - кладет кубик
function put($ffi, $robohand, $speed, $errorText)
{
    //выполняем мерцание светодиодом,передаем дескриптор светодиода,3 параметра цвета(RGB),продолжительность,кол-во повторений и включаем асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_RGB_LED_Flicker($robohand->led->cdata, 0, 0, 255, 500, 0, true, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->arrowR->cdata, $GLOBALS['arrowR_cube_position'], $speed, false, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->arrowL->cdata, $GLOBALS['arrowL_cube_position'], $speed, false, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->claw->cdata, $GLOBALS['claw_unclenched_position'], $speed, false, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->arrowR->cdata, (-1) * $GLOBALS['arrowR_cube_position'], $speed, false, $errorText);
    if ($errCode) {
```

```
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->arrowL->cdata, (-1) * $GLOBALS['arrowL_cube_position'], $speed, false, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_ServoDrive_Turn($robohand->claw->cdata, (-1) * $GLOBALS['claw_unclenched_position'], $speed, false, $errorText);
    if ($errCode) {
        return $errCode;
    }
    return 0;
}

// destructServos - уничтожает сервоприводы
function destructServos($ffi, $servos, $errorText)
{
    //уничтожаем сервоприводы
    for ($i = 0; $i < 5; $i++) {
        $errCode = $ffi->RI_SDK_DestroyComponent($servos[$i]->descriptor->cdata, $errorText);
        if ($errCode) {
            return $errCode;
        }
    }

    return 0;
}

// destruct - уничтожает все компоненты и библиотеку
function destruct($ffi, $robohand, $servos, $errorText)
{
//выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3 параметра цвета(RGB), и включаем асинхронный режим работы
    $errCode = $ffi->RI_SDK_exec_RGB_LED_SinglePulse($robohand->led->cdata, 255, 0, 0, 0, true, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //уничтожаем сервоприводы
    $errCode = destructServos($ffi, $servos, $errorText);
    if ($errCode) {
        return $errCode;
    }
```

```
    //останавливаем свечение светодиода с определенным дескриптором
    $errCode = $ffi->RI_SDK_exec_RGB_LED_Stop($robohand->led->cdata, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //удаляем компонент светодиода по дескриптору
    $errCode = $ffi->RI_SDK_DestroyComponent($robohand->led->cdata, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //сбрасываем все порты на ШИМ
    $errCode = $ffi->RI_SDK_sigmod_PWM_ResetAll($robohand->pwm->cdata, $errorText);
    if ($errCode) {
        return $errCode;
    }

    // удаляем компонент ШИМ
    $errCode = $ffi->RI_SDK_DestroyComponent($robohand->pwm->cdata, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //удаляем компонент i2c
    $errCode = $ffi->RI_SDK_DestroyComponent($robohand->i2c->cdata, $errorText);
    if ($errCode) {
        return $errCode;
    }

    //удаляем sdk со всеми компонентами в реестре компонентов
    $errCode = $ffi->RI_SDK_DestroySDK(true, $errorText);
    if ($errCode) {
        return $errCode;
    }
    return 0;
}
```

C

Подключение RI_SDK к приложению на C

Linux / Windows

Создадим C файл

Команда для Linux

```
touch shared_client.c
```

Команда для Windows

```
type nul > shared_client.c
```

Структура проекта

. workspace/
--- > shared_client.c
--- > librisdk.h
--- > librisdk.so/.dll

Подключим библиотеку

```c
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include "librisdk.h"

int main() {
    printf("Success\n");

    return 0;
}
```

Для запуска скрипта выполним в терминале следующие команды.
Укажем компилятору gcc что динамические библиотеки можно искать в текущей директории

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.
```

Выполним компиляцию:

Команда для Linux

```
gcc shared_client.c -o c_shared_client.bin -lrisdk -L.
```

Команда для Windows

```
gcc shared_client.c -o c_shared_client.exe -lrisdk -L.
```

Значение **shared_client.c** указывает на файл, который необходимо собрать
Значение **-o c_shared_client.bin/.exe** указывает на путь к скомпиленному файлу
Значение **-lrisdk** указывает на путь к динамической библиотеке (В данном случае на librisdk.so)
Значение **-L.** указывает в какой директории относительно LD_LIBRARY_PATH искать библиотеку (В данном случае в текущей)

Получим результат выполнения собранной программы

```
./c_shared_client.bin
```

Создадим компонент внутри функции main

```
int logLevel = 2;
int errCode;
char errorText[1000];

// Инициализируем SDK
errCode = RI_SDK_InitSDK(logLevel, errorText);
if (errCode) {
    // Выведем текст и код ошибки в терминал, при возникновении
    printf("RI_SDK_InitSDK errorText:%s; errorCode:%s--\n", errorText, errCode);
    return errCode;
}

int descriptor;

// Создаем базовый компонент
errCode = RI_SDK_CreateBasic(&descriptor, errorText);
if (errCode) {
    // Выведем текст и код ошибки в терминал, при возникновении
    printf("RI_SDK_CreateBasic errorText:%s; errorCode:%s--\n", errorText, errCode);
    return errCode;
}

// Выведем полученный дескриптор компонента
printf("descriptor:%d--\n", descriptor);
```

Компилируем и запустим скрипт. Результат успешной работы должен выглядеть следующим образом

```
-bash#$ ./c_shared_client.bin
descriptor:1--
Success
-bash#$
```

Пример программы на C, использующий RI_SDK

Описание

Данная программа является примером, использования библиотеки. Она реализует простой сценарий поведения робота. Сперва все сервоприводы встают в условное стартовое положение. После робот берет кубик с позиции слева и перемещает его на позицию справа. После сервоприводы возвращаются в первоначальное положение.
Одновременно с движением робота, происходит свечение светодиода. Светодиод горит красным цветом пока сервоприводы приводятся к стартовому положению, после загорается зеленым. Далее светодиоды моргает зеленым при передвижении тела робота и мерцает синим, пока робот кладет/берет кубик.

Структура проекта

. workspace/
        --- > demo.c

--- > librisdk.h
--- > librisdk.dll/.so
--- > CH341DLL.DLL/ CH341DLLA64.DLL
--- > SLABHIDDevice.dll
--- > SLABHIDtoSMBus.dll

Реализация

Реализация данной программы представлена ниже. Ее также можно скачать по ссылке на репозиторий. Вы можете скопировать данный пример и запустить у себя.

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <windows.h>
#include <unistd.h>
#include "./librisdk.h"

char errorText[1000]; // текст ошибки. Передается как входной параметр,при возникновении ошибки в эту переменную будет записан текст ошибки

int body_start_pulse = 1500;         // стартовая позиция тела
int arrowR_start_pulse = 2000;       // стартовая возиция правой стрелы
int arrowL_start_pulse = 1000;       // стартовая позиция левой стрелы
int claw_start_pulse = 1000;         // стартовая позиция клешни
int claw_rotate_start_pulse = 1500; // стартовая позиция поворота клешни

int arrowR_over_cube_position = -30; // позиция правой стрелы над кубиком
int arrowL_over_cube_position = -10; // позиция левой стрелы над кубиком
int claw_unclenched_position = 60;   // позиция открытой клешни
int arrowR_cube_position = -75;      // позиция правой стрелы на месте кубика
int arrowL_cube_position = 55;       // позиция левой стрелы на месте кубика

int logLevel = 2; // уровень логирования
int speed = 100;  // скорость в градусах в секунду

// структура дескрипторов устройства
struct device
{
    int i2c;         // дескриптор i2c
    int pwm;         // дескриптор pwm
    int body;        // дескриптор сервопривода тела
    int claw;        // дескриптор сервопривода клешни
    int arrowR;      // дескриптор сервопривода правой стрелы
    int arrowL;      // дескриптор сервопривода левой стрелы
    int clawRotate; // дескриптор сервопривода поворота клешни
    int led;         // дескриптор светодиода
};

// инициализация структуры устройства
```

```
struct device robohand;

struct servo
{
    int *descriptor;    // дескриптор сервопривода
    int *startPosition; // стартовая позиция сервопривода
};

// массив дескрипторов сервоприводов
struct servo servos[5] = {
    &robohand.body,
    &body_start_pulse,
    &robohand.claw,
    &claw_start_pulse,
    &robohand.arrowR,
    &arrowR_start_pulse,
    &robohand.arrowL,
    &arrowL_start_pulse,
    &robohand.clawRotate,
    &claw_rotate_start_pulse,
};

// initServos - создает сервоприводы и линкует их
int initServos()
{
    int errCode; // код ошибки

    //создаем 5 сервоприводов и линкуем их к пинам 0-4
    for (int i = 0; i < 5; i++)
    {
        // создаем компонент сервопривода с конкретной моделью как исполняемое
устройство и получаем дескриптор сервопривода
        errCode = RI_SDK_CreateModelComponent("executor", "servodrive", "mg90s",
servos[i].descriptor, errorText);
        if (errCode)
        {
            return errCode;
        }

        //связываем сервопривод с ШИМ,передаем дескриптор сервопривода и ШИМ
        errCode = RI_SDK_LinkServodriveToController(*servos[i].descriptor,
robohand.pwm, i, errorText);
        if (errCode)
        {
            return errCode;
        }
    }

    return 0;
}

// initDevice - инициализация библиотеки и устройств
```

```
int initDevice()
{
    int errCode;

    // Инициализируем SDK
    errCode = RI_SDK_InitSDK(logLevel, errorText);
    if (errCode)
    {
        return errCode;
    }

    // создаем компонент ШИМ с конкретной моделью как соединитель и получаем
дескриптор компонента
    errCode = RI_SDK_CreateModelComponent("connector", "pwm", "pca9685", &robohand.pwm,
errorText);
    if (errCode)
    {
        return errCode;
    }

    // создаем компонент i2c адаптера
    // Здесь осуществлен примитивное определение подключенной модели адаптера
    // Сначала пробуем создать i2c адаптер модели ch341 и связать с ним ШИМ
    errCode = RI_SDK_CreateModelComponent("connector", "i2c_adapter", "ch341",
&robohand.i2c, errorText);
    if (errCode)
    {
        return errCode;
    }

    //связываем i2c адаптер с ШИМ по адресу 0x40
    errCode = RI_SDK_LinkPWMToController(robohand.pwm, robohand.i2c, 0x40, errorText);
    if (errCode)
    {
        // Если не получается то пробуем создать i2c адаптер модели cp2112
        errCode = RI_SDK_CreateModelComponent("connector", "i2c_adapter", "cp2112",
&robohand.i2c, errorText);
        if (errCode)
        {
            return errCode;
        }

        //связываем i2c адаптер с ШИМ по адресу 0x40
        errCode = RI_SDK_LinkPWMToController(robohand.pwm, robohand.i2c, 0x40,
errorText);
        if (errCode)
        {
            return errCode;
        }
    }

    // создаем компонент светодиода с конкретной моделью (ky016) как исполняемое
```

```
устройство и получаем дескриптор светодиода
    errCode = RI_SDK_CreateModelComponent("executor", "led", "ky016", &robohand.led,
errorText);
    if (errCode)
    {
        return errCode;
    }

    //связываем светодиод адаптер с ШИМ,передаем значения трех пинов к которым
подключен светодиод
    errCode = RI_SDK_LinkLedToController(robohand.led, robohand.pwm, 15, 14, 13,
errorText);
    if (errCode)
    {
        return errCode;
    }

    //инициализируем сервоприводы
    errCode = initServos();
    if (errCode)
    {
        return errCode;
    }

    return 0;
}

// startPosition - переводит сервопривод в стартовое положение
int startPosition(struct servo s)
{
    int errCode;
    //выполняем поворот сервопривода в заданный угол,передаем дескриптор
сервопривода,значение угла
    errCode = RI_SDK_exec_ServoDrive_TurnByPulse(*s.descriptor, *s.startPosition,
errorText);
    if (errCode)
    {
        return errCode;
    }

    //небольшая пауза для последовательного движения
    Sleep(500);
    return 0;
}

// startPositionAllServo - переводит все сервоприводы в стартовую позицию
int startPositionAllServo()
{
    int errCode; //код ошибки
                 //выполняем одиночное свечение светодиодом,передаем дескриптор
светодиода,3 параметра цвета(RGB), и включаем асинхронный режим работы
    errCode = RI_SDK_exec_RGB_LED_SinglePulse(robohand.led, 255, 0, 0, 0, true,
```

```
errorText);
    if (errCode)
    {
        return errCode;
    }

    //приводим сервоприводы в стартовое положение
    for (int i = 0; i < 5; i++)
    {
        errCode = startPosition(servos[i]);
        if (errCode)
        {
            return errCode;
        }
    }

    //выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3
параметра цвета(RGB), и включаем асинхронный режим работы
    errCode = RI_SDK_exec_RGB_LED_SinglePulse(robohand.led, 0, 255, 0, 0, true,
errorText);
    if (errCode)
    {
        return errCode;
    }
    //небольшая пауза для последовательного движения
    Sleep(500);
    return 0;
}

// roteteBody - вращение тела в указанный угол
int roteteBody(int angle, int speed)
{

    int errCode; //код ошибки

    //выполняем мигание с заданной частотой,передаем дескриптор светодиода,3 параметра
цвета(RGB),частоту,продолжительность и включаем асинхронный режим работы
    errCode = RI_SDK_exec_RGB_LED_FlashingWithFrequency(robohand.led, 0, 255, 0, 5, 0,
true, errorText);
    if (errCode)
    {
        return errCode;
    }
    //выполняем поворот на заданный угол,передаем дескриптор тела,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.body, angle, speed, false,
errorText);
    if (errCode)
    {
        return errCode;
    }
```

```
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.clawRotate, 45, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.clawRotate, -45, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }
    return 0;
}

// get - берет кубик
int get()
{
    int errCode; //код ошибки
    //выполняем мерцание светодиодом,передаем дескриптор светодиода,3 параметра цвета(RGB),продолжительность,кол-во повторений и включаем асинхронный режим работы
    errCode = RI_SDK_exec_RGB_LED_Flicker(robohand.led, 0, 0, 255, 500, 0, true, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowR, arrowR_over_cube_position, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowL, arrowL_over_cube_position, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
```

```
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.claw, claw_unclenched_position,
speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowR, (arrowR_cube_position -
arrowR_over_cube_position), speed,
                                          false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowL, (arrowL_cube_position -
arrowL_over_cube_position), speed,
                                          false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.claw, (-1) *
claw_unclenched_position, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowR, (-1) * arrowR_cube_position,
speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowL, (-1) * arrowL_cube_position,
speed, false, errorText);
    if (errCode)
    {
```

```
        return errCode;
    }

    return 0;
}

// put - кладет кубик
int put()
{
    int errCode; //код ошибки
    //выполняем мерцание светодиодом,передаем дескриптор светодиода,3 параметра цвета(RGB),продолжительность,кол-во повторений и включаем асинхронный режим работы
    errCode = RI_SDK_exec_RGB_LED_Flicker(robohand.led, 0, 0, 255, 500, 0, true, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowR, arrowR_cube_position, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowL, arrowL_cube_position, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.claw, claw_unclenched_position, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowR, (-1) * arrowR_cube_position, speed, false, errorText);
    if (errCode)
    {
```

```
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowL, (-1) * arrowL_cube_position, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.claw, (-1) * claw_unclenched_position, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    return 0;
}

// destructServos - уничтожает сервоприводы
int destructServos()
{
    int errCode; //код ошибки

    for (int i = 0; i < 5; i++)
    {
        errCode = RI_SDK_DestroyComponent(*servos[i].descriptor, errorText);
        if (errCode)
        {
            return errCode;
        }
    }

    return 0;
}

// destruct - уничтожает все окомпоненты и библиотеку
int destruct()
{

    int errCode; //код ошибки

    //выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3 параметра цвета(RGB), и включаем асинхронный режим работы
    errCode = RI_SDK_exec_RGB_LED_SinglePulse(robohand.led, 255, 0, 0, 0, true, errorText);
    if (errCode)
```

```
    {
        return errCode;
    }

    //уничтожаем сервоприводы
    errCode = destructServos();
    if (errCode)
    {
        return errCode;
    }
    //останавливаем свечение светодиода с определенным дескриптором
    errCode = RI_SDK_exec_RGB_LED_Stop(robohand.led, errorText);
    if (errCode)
    {
        return errCode;
    }

    //удаляем компонент светодиода по дескриптору
    errCode = RI_SDK_DestroyComponent(robohand.led, errorText);
    if (errCode)
    {
        return errCode;
    }

    //сбрасываем все порты на ШИМ
    errCode = RI_SDK_sigmod_PWM_ResetAll(robohand.pwm, errorText);
    if (errCode)
    {
        return errCode;
    }

    //удаляем компонент ШИМ
    errCode = RI_SDK_DestroyComponent(robohand.pwm, errorText);
    if (errCode)
    {
        return errCode;
    }
    //удаляем компонент i2c
    errCode = RI_SDK_DestroyComponent(robohand.i2c, errorText);
    if (errCode)
    {
        return errCode;
    }

    //удаляем sdk со всеми компонентами в реестре компонентов
    errCode = RI_SDK_DestroySDK(true, errorText);
    if (errCode)
    {
        return errCode;
    }
    return 0;
}
```

```
// start - запускает демо программу
int start()
{
    int errCode;

    // Инициализируем библиотеку и компоненты
    errCode = initDevice();
    if (errCode)
    {
        return errCode;
    }

    // Приводим сервоприводы к стартовой позиции
    errCode = startPositionAllServo();
    if (errCode)
    {
        return errCode;
    }

    // Двигаем тело к местонахождению кубика
    errCode = roteteBody(45, speed);
    if (errCode)
    {
        return errCode;
    }

    // Берем кубик
    errCode = get();
    if (errCode)
    {
        return errCode;
    }

    //Двигаем тело к новому местонахождению кубика
    errCode = roteteBody(-90, speed);
    if (errCode)
    {
        return errCode;
    }

    //Кладем кубик
    errCode = put();
    if (errCode)
    {
        return errCode;
    }

    // Возвращаем тело в стартовое положение
    errCode = roteteBody(45, speed);
    if (errCode)
    {
```

```
        return errCode;
    }

    // Уничтожаем компоненты и библиотеку
    errCode = destruct();
    if (errCode)
    {
        return errCode;
    }
}

int main()
{

    int errCode; //код ошибки

    //запускаем программу
    errCode = start();
    if (errCode)
    {
        printf("errorText:%s\n", errorText);
        return errCode;
    }

    return 0;
}
```

C++

Подключение RI_SDK к приложению на C++

Создадим C++ файл

Команда для Linux

```
touch shared_client.cpp
```

Команда для Windows

```
type nul > shared_client.cpp
```

Структура проекта

. workspace/
    --- > shared_client.cpp
    --- > librisdk.h
    --- > librisdk.so/.dll

Подключим библиотеку

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include "librisdk.h"
#include <iostream>

int main(void) {
    std::cout << "Success\n";

    return 0;
}
```

Для запуска скрипта выполним в терминале следующие команды.
Укажем компилятору gcc что динамические библиотеки можно искать в текущей директории

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.
```

Выполним компиляцию:

Команда для Linux

```
g++ -I. -L. shared_client.cpp  -o cpp_shared_client.bin -lrisdk
```

Команда для Windows

```
g++ -I. -L. shared_client.cpp  -o cpp_shared_client.exe -lrisdk
```

Значение **shared_client.cpp** указывает на файл, который необходимо собрать
Значение **-o cpp_shared_client.bin/.exe** указывает на путь к скомпиленному файлу
Значение **-lrisdk** указывает на путь к динамической библиотеке (В данном случае на librisdk.so)
Значение **-L.** указывает в какой директории относительно LD_LIBRARY_PATH искать библиотеку (В данном случае в текущей)

Получим результат выполнения собранной программы

```
./cpp_shared_client.bin
```

Создадим компонент внутри функции main

```
int logLevel = 2;
int errCode;
char errorText[1000];

// Инициализируем SDK
errCode = RI_SDK_InitSDK(logLevel, errorText);
if (errCode) {
    // Выведем текст и код ошибки в терминал, при возникновении
    printf("RI_SDK_InitSDK errorText:%s; errorCode:%d--\n", errorText, errCode);
    return errCode;
}
```

```
int descriptor;

// Создаем базовый компонент
errCode = RI_SDK_CreateBasic(&descriptor, errorText);
if (errCode) {
    // Выведем текст и код ошибки в терминал, при возникновении
    printf("RI_SDK_CreateBasic errorText:%s; errorCode:%d--\n", errorText, errCode);
    return errCode;
}

// Выведем полученный дескриптор компонента
printf("descriptor:%d--\n", descriptor);
```

Компилируем и запустим скрипт. Результат успешной работы должен выглядеть следующим образом

```
-bash#$ ./cpp_shared_client.bin
descriptor:1--
Success
-bash#$
```

Пример программы на C++, использующий RI_SDK

Описание

Данная программа является примером, использования библиотеки. Она реализует простой сценарий поведения робота. Сперва все сервоприводы встают в условное стартовое положение. После робот берет кубик с позиции слева и перемещает его на позицию справа. После сервоприводы возвращаются в первоначальное положение.
Одновременно с движением робота, происходит свечение светодиода. Светодиод горит красным цветом пока сервоприводы приводятся к стартовому положению, после загорается зеленым. Далее светодиоды моргает зеленым при передвижении тела робота и мерцает синим, пока робот кладет/берет кубик.

Структура проекта

. workspace/
- --- > demo.cpp
- --- > librisdk.h
- --- > librisdk.dll/so
- --- > CH341DLL.DLL/ CH341DLLA64.DLL
- --- > SLABHIDDevice.dll
- --- > SLABHIDtoSMBus.dll

Реализация

Реализация данной программы представлена ниже. Ее также можно скачать по ссылке на репозиторий. Вы можете скопировать данный пример и запустить у себя.

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <windows.h>
#include <unistd.h>
#include <iostream>
#include "./librisdk.h"

char errorText[1000]; // текст ошибки. Передается как входной параметр,при возникновении ошибки в эту переменную будет записан текст ошибки

int body_start_pulse = 1500;         // стартовая позиция тела
int arrowR_start_pulse = 2000;       // стартовая возиция правой стрелы
int arrowL_start_pulse = 1000;       // стартовая позиция левой стрелы
int claw_start_pulse = 1000;         // стартовая позиция клешни
int claw_rotate_start_pulse = 1500; // стартовая позиция поворота клешни

int arrowR_over_cube_position = -30; // позиция правой стрелы над кубиком
int arrowL_over_cube_position = -10; // позиция левой стрелы над кубиком
int claw_unclenched_position = 80;   // позиция открытой клешни
int arrowR_cube_position = -75;      // позиция правой стрелы на месте кубика
int arrowL_cube_position = 55;       // позиция левой стрелы на месте кубика

int logLevel = 2; // уровень логирования
int speed = 100;  // скорость в градусах в секунду

// структура дескрипторов устройства
struct device
{
    int i2c;        // дескриптор i2c
    int pwm;        // дескриптор pwm
    int body;       // дескриптор сервопривода тела
    int claw;       // дескриптор сервопривода клешни
    int arrowR;     // дескриптор сервопривода правой стрелы
    int arrowL;     // дескриптор сервопривода левой стрелы
    int clawRotate; // дескриптор сервопривода поворота клешни
    int led;        // дескриптор светодиода
};

// инициализация структуры устройства
struct device robohand;

struct servo
{
    int *descriptor;    // дескриптор сервопривода
    int *startPosition; // стартовая позиция сервопривода
};

// массив дескрипторов сервоприводов
```

```
struct servo servos[5] = {
    &robohand.body,
    &body_start_pulse,
    &robohand.claw,
    &claw_start_pulse,
    &robohand.arrowR,
    &arrowR_start_pulse,
    &robohand.arrowL,
    &arrowL_start_pulse,
    &robohand.clawRotate,
    &claw_rotate_start_pulse,
};

// initServos - создает сервоприводы и линкует их
int initServos()
{
    int errCode;
    char executor[] = "executor";
    char servodrive[] = "servodrive";
    char mg90s[] = "mg90s";

    //создаем 5 сервоприводов и линкуем их к пинам 0-4
    for (int i = 0; i < 5; i++)
    {
        // создаем компонент сервопривода с конкретной моделью как исполняемое
устройство и получаем дескриптор сервопривода
        errCode = RI_SDK_CreateModelComponent(executor, servodrive, mg90s,
servos[i].descriptor, errorText);
        if (errCode)
        {
            return errCode;
        }
        //связываем сервопривод с ШИМ,передаем дескриптор сервопривода и ШИМ
        errCode = RI_SDK_LinkServodriveToController(*servos[i].descriptor,
robohand.pwm, i, errorText);
        if (errCode)
        {
            return errCode;
        }
    }

    return 0;
}

// initDevice - инициализация библиотеки и устройств
int initDevice()
{
    int errCode; //код ошибки
    char executor[] = "executor";
    char connector[] = "connector";
    char pwm[] = "pwm";
    char cp2112[] = "cp2112";
```

```
    char ch341[] = "ch341";
    char pca9685[] = "pca9685";
    char led[] = "led";
    char ky016[] = "ky016";

    // Инициализируем SDK
    errCode = RI_SDK_InitSDK(logLevel, errorText);
    if (errCode)
    {
        return errCode;
    }

    // создаем компонент сервопривода с конкретной моделью как исполняемое
устройство,получаем дескриптор сервопривода
    errCode = RI_SDK_CreateModelComponent(connector, pwm, pca9685, &robohand.pwm,
errorText);
    if (errCode)
    {
        return errCode;
    }

    // создаем компонент i2c адаптера
    // Здесь осуществлен примитивное определение подключенной модели адаптера
    // Сначала пробуем создать i2c адаптер модели ch341 и связать с ним ШИМ
    errCode = RI_SDK_CreateModelComponent(connector, pwm, ch341, &robohand.i2c,
errorText);
    if (errCode)
    {
        return errCode;
    }

    //связываем i2c адаптер с ШИМ по адресу 0x40
    errCode = RI_SDK_LinkPWMToController(robohand.pwm, robohand.i2c, 0x40, errorText);
    if (errCode)
    {
        // Если не получается то пробуем создать i2c адаптер модели cp2112
        errCode = RI_SDK_CreateModelComponent(connector, pwm, cp2112, &robohand.i2c,
errorText);
        if (errCode)
        {
            return errCode;
        }

        //связываем i2c адаптер с ШИМ по адресу 0x40
        errCode = RI_SDK_LinkPWMToController(robohand.pwm, robohand.i2c, 0x40,
errorText);
        if (errCode)
        {
            return errCode;
        }
    }
```

```
    // создаем компонент светодиода с конкретной моделью (ky016) как исполняемое
устройство и получаем дескриптор светодиода
    errCode = RI_SDK_CreateModelComponent(executor, led, ky016, &robohand.led,
errorText);
    if (errCode)
    {
        return errCode;
    }

    //связываем светодиод адаптер с ШИМ,передаем значения трех пинов к которым
подключен светодиод
    errCode = RI_SDK_LinkLedToController(robohand.led, robohand.pwm, 15, 14, 13,
errorText);
    if (errCode)
    {
        return errCode;
    }

    //инициализируем сервоприводы
    errCode = initServos();
    if (errCode)
    {
        return errCode;
    }

    return 0;
}

// startPosition - переводит сервопривод в стартовое положение
int startPosition(struct servo s)
{
    int errCode; //код ошибки
        //выполняем поворот сервопривода в заданный угол,передаем дескриптор
сервопривода,значение угла
    errCode = RI_SDK_exec_ServoDrive_TurnByPulse(*s.descriptor, *s.startPosition,
errorText);
    if (errCode)
    {
        return errCode;
    }

    //небольшая пауза для последовательного движения
    Sleep(500);
    return 0;
}

// startPositionAllServo - переводит все сервоприводы в стартовую позицию
int startPositionAllServo()
{
    int errCode; //код ошибки

    //выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3
```

```
параметра цвета(RGB), и включаем асинхронный режим работы
    errCode = RI_SDK_exec_RGB_LED_SinglePulse(robohand.led, 255, 0, 0, 0, true,
errorText);
    if (errCode)
    {
        return errCode;
    }
    //приводим сервоприводы в стартовое положение
    for (int i = 0; i < 5; i++)
    {
        errCode = startPosition(servos[i]);
        if (errCode)
        {
            return errCode;
        }
    }

    //выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3
параметра цвета(RGB), и включаем асинхронный режим работы
    errCode = RI_SDK_exec_RGB_LED_SinglePulse(robohand.led, 0, 255, 0, 0, true,
errorText);
    if (errCode)
    {
        return errCode;
    }
    //небольшая пауза для последовательного движения
    Sleep(500);
    return 0;
}

// rotateBody - вращение тела в указанный угол
int rotateBody(int angle, int speed)
{

    int errCode; //код ошибки

    //выполняем мигание с заданной частотой,передаем дескриптор светодиода,3 параметра
цвета(RGB),частоту,продолжительность и включаем асинхронный режим работы
    errCode = RI_SDK_exec_RGB_LED_FlashingWithFrequency(robohand.led, 0, 255, 0, 5, 0,
true, errorText);
    if (errCode)
    {
        return errCode;
    }
    //выполняем поворот на заданный угол,передаем дескриптор тела,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.body, angle, speed, false,
errorText);
    if (errCode)
    {
        return errCode;
    }
```

```
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.clawRotate, 45, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.clawRotate, -45, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }
    return 0;
}

// get - берет кубик
int get()
{
    int errCode; //код ошибки
        //выполняем мерцание светодиодом,передаем дескриптор светодиода,3 параметра цвета(RGB),продолжительность,кол-во повторений и включаем асинхронный режим работы
    errCode = RI_SDK_exec_RGB_LED_Flicker(robohand.led, 0, 0, 255, 500, 0, true, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowR, arrowR_over_cube_position, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowL, arrowL_over_cube_position, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }
```

```
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.claw, claw_unclenched_position,
speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowR, (arrowR_cube_position -
arrowR_over_cube_position), speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowL, (arrowL_cube_position -
arrowL_over_cube_position), speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.claw, (-1) *
claw_unclenched_position, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowR, (-1) * arrowR_cube_position,
speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowL, (-1) * arrowL_cube_position,
speed, false, errorText);
    if (errCode)
    {
        return errCode;
```

```
    }

    return 0;
}

// put - кладет кубик
int put()
{
    int errCode; //код ошибки
        //выполняем мерцание светодиодом,передаем дескриптор светодиода,3 параметра
цвета(RGB),продолжительность,кол-во повторений и включаем асинхронный режим работы
    errCode = RI_SDK_exec_RGB_LED_Flicker(robohand.led, 0, 0, 255, 500, 0, true,
errorText);
    if (errCode)
    {
        return errCode;
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowR, arrowR_cube_position, speed,
false, errorText);
    if (errCode)
    {
        return errCode;
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowL, arrowL_cube_position, speed,
false, errorText);
    if (errCode)
    {
        return errCode;
    }
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.claw, claw_unclenched_position,
speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowR, (-1) * arrowR_cube_position,
speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.arrowL, (-1) * arrowL_cube_position,
```

```
speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errCode = RI_SDK_exec_ServoDrive_Turn(robohand.claw, (-1) *
claw_unclenched_position, speed, false, errorText);
    if (errCode)
    {
        return errCode;
    }

    return 0;
}

// destructServos - уничтожает сервоприводы
int destructServos()
{
    int errCode; //код ошибки
    for (int i = 0; i < 5; i++)
    {
        errCode = RI_SDK_DestroyComponent(*servos[i].descriptor, errorText);
        if (errCode)
        {
            return errCode;
        }
    }

    return 0;
}

// destruct - уничтожает все компоненты и библиотеку
int destruct()
{

    int errCode; //код ошибки
    //выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3
параметра цвета(RGB), и включаем асинхронный режим работы
    errCode = RI_SDK_exec_RGB_LED_SinglePulse(robohand.led, 255, 0, 0, 0, true,
errorText);
    if (errCode)
    {
        return errCode;
    }
    //уничтожаем сервоприводы
    errCode = destructServos();
    if (errCode)
    {
        return errCode;
```

```
    }
    //останавливаем свечение светодиода с определенным дескриптором
    errCode = RI_SDK_exec_RGB_LED_Stop(robohand.led, errorText);
    if (errCode)
    {
        return errCode;
    }
    //удаляем компонент светодиода по дескриптору
    errCode = RI_SDK_DestroyComponent(robohand.led, errorText);
    if (errCode)
    {
        return errCode;
    }
    //сбрасываем все порты на ШИМ
    errCode = RI_SDK_sigmod_PWM_ResetAll(robohand.pwm, errorText);
    if (errCode)
    {
        return errCode;
    }
    //удаляем компонент ШИМ
    errCode = RI_SDK_DestroyComponent(robohand.pwm, errorText);
    if (errCode)
    {
        return errCode;
    }
    //удаляем компонент i2c
    errCode = RI_SDK_DestroyComponent(robohand.i2c, errorText);
    if (errCode)
    {
        return errCode;
    }
    //удаляем sdk со всеми компонентами в реестре компонентов
    errCode = RI_SDK_DestroySDK(true, errorText);
    if (errCode)
    {
        return errCode;
    }
    return 0;
}

// start - запускает демо программу
int start()
{
    int errCode; //код ошибки

    // Инициализируем библиотеку и компоненты
    errCode = initDevice();
    if (errCode)
    {
        return errCode;
    }
```

```
    // Приводим сервоприводы к стартовой позиции
    errCode = startPositionAllServo();
    if (errCode)
    {
        return errCode;
    }

    // Двигаем тело к местонахождению кубика
    errCode = rotateBody(45, speed);
    if (errCode)
    {
        return errCode;
    }

    // Берем кубик
    errCode = get();
    if (errCode)
    {
        return errCode;
    }

    //Двигаем тело к новому местонахождению кубика
    errCode = rotateBody(-90, speed);
    if (errCode)
    {
        return errCode;
    }

    //Кладем кубик
    errCode = put();
    if (errCode)
    {
        return errCode;
    }

    // Возвращаем тело в стартовое положение
    errCode = rotateBody(45, speed);
    if (errCode)
    {
        return errCode;
    }

    // Уничтожаем компоненты и библиотеку
    errCode = destruct();
    if (errCode)
    {
        return errCode;
    }
    return 0;
}

int main()
```

```
{
    int errCode; //код ошибки
    //запускаем программу
    errCode = start();
    if (errCode)
    {
        printf("errorText:%s\n", errorText);
        return errCode;
    }

    std::cout << "Success\n";

    return 0;
}
```

Golang

Подключение RI_SDK динамической библиотеки к приложению на Golang

Linux

Создадим Golang файл

```
touch shared_client.go
```

Структура проекта

. workspace/
        --- > shared_client.go
        --- > librisdk.h
        --- > librisdk.so

Подключим библиотеку

```
package main

/*
#cgo CFLAGS: -I.
#cgo LDFLAGS: -L. -lrisdk
#include <librisdk.h>
*/
import "C"
import (
      "fmt"
)

func main() {

      fmt.Println("Success")
```

```
}
```

Для запуска скрипта выполним в терминале следующие команды.
Укажем компилятору gcc что динамические библиотеки можно искать в текущей директории

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.
```

Выполним компиляцию:

```
go build -o go_shared_client.bin shared_client.go
```

Значение **shared_client.go** указывает на файл, который необходимо собрать
Значение **-o go_shared_client.bin** указывает на путь к скомпиленному файлу

Получаем результат выполнения собранной программы

```
./go_shared_client.bin
```

Создадим компонент внутри функции main

```
var (
	logLevel   = C.int(1)   //уровень логирования
	errorTextC [1000]C.char //текст ошибки
	errorCode  C.int        //код ошибки
)

// Инициализируем SDK
errorCode = C.RI_SDK_InitSDK(logLevel, &errorTextC[0])
if errorCode != 0 {
	// Возвращаем текст и код ошибки
	errorText := C.GoString(&errorTextC[0])
	err = fmt.Errorf("errorCode:%d - errorText:%s", errorCode, errorText)
	return
}

var descriptor = C.int(0)

// Создаем базовый компонент
errorCode = C.RI_SDK_CreateBasic(&descriptor, &errorTextC[0])
if errorCode != 0 {
	// Возвращаем текст и код ошибки
	errorText := C.GoString(&errorTextC[0])
	err = fmt.Errorf("errorCode:%d - errorText:%s", errorCode, errorText)
	return
}

// Выведем полученный дескриптор компонента
fmt.Printf("descriptor:%d--\n", descriptor)
```

Компилируем и запустим скрипт. Результат успешной работы должен выглядеть следующим образом

```
-bash#$ ./go_shared_client.bin
descriptor:1--
Success
-bash#$
```

Windows

На данный момент Golang не поддерживает работу с Shared Object на Windows.

Пример программы на Golang, использующий RI_SDK

Описание

Данная программа является примером, использования библиотеки. Она реализует простой сценарий поведения робота. Сперва все сервоприводы встают в условное стартовое положение. После робот берет кубик с позиции слева и перемещает его на позицию справа. После сервоприводы возвращаются в первоначальное положение.
Одновременно с движением робота, происходит свечение светодиода. Светодиод горит красным цветом пока сервоприводы приводятся к стартовому положению, после загорается зеленым. Далее светодиоды моргает зеленым при передвижении тела робота и мерцает синим, пока робот кладет/берет кубик.

Структура проекта

. workspace/
    --- > demo.go
    --- > librisdk.h
    --- > librisdk.so

Реализация

Реализация данной программы представлена ниже. Ее также можно скачать по ссылке на репозиторий. Вы можете скопировать данный пример и запустить у себя.

```
package main

/*
#cgo CFLAGS: -I.
#cgo LDFLAGS: -L. -lrisdk
#include <librisdk.h>
*/
```

```
import "C"
import (
    "fmt"
    "time"
)

var (
    body_start_pulse        = C.int(1500) // стартовая позиция тела
    arrowR_start_pulse      = C.int(2000) // стартовая возиция правой стрелы
    arrowL_start_pulse      = C.int(1000) // стартовая позиция левой стрелы
    claw_start_pulse        = C.int(1000) // стартовая позиция клешни
    claw_rotate_start_pulse = C.int(1500) // стартовая позиция поворота клешни

    arrowR_over_cube_position = C.int(-30) // позиция правой стрелы над кубиком
    arrowL_over_cube_position = C.int(-10) // позиция левой стрелы над кубиком
    claw_unclenched_position  = C.int(80)  // позиция открытой клешни
    arrowR_cube_position      = C.int(-75) // позиция правой стрелы на месте кубика
    arrowL_cube_position      = C.int(55)  // позиция левой стрелы на месте кубика

    logLevel = C.int(2)   // уровень логирования
    speed    = C.int(100) // скорость в градусах в секунду
)

// структура дескрипторов устройства
type device struct {
    i2c        C.int // дескриптор i2c
    pwm        C.int // дескриптор pwm
    body       C.int // дескриптор сервопривода тела
    claw       C.int // дескриптор сервопривода клешни
    arrowR     C.int // дескриптор сервопривода правой стрелы
    arrowL     C.int // дескриптор сервопривода левой стрелы
    clawRotate C.int // дескриптор сервопривода поворота клешни
    led        C.int // дескриптор светодиода
}

// инициализация структуры устройства
var robohand device

type servo struct {
    descriptor    *C.int // дескриптор сервопривода
    startPosition *C.int // стартовая позиция сервопривода
}

// массив дескрипторов сервоприводов
var servos = [5]servo{
    {
        descriptor:    &robohand.body,
        startPosition: &body_start_pulse,
    },
    {
        descriptor:    &robohand.claw,
        startPosition: &claw_start_pulse,
```

```
    },
    {
        descriptor:    &robohand.arrowR,
        startPosition: &arrowR_start_pulse,
    },
    {
        descriptor:    &robohand.arrowL,
        startPosition: &arrowL_start_pulse,
    },
    {
        descriptor:    &robohand.clawRotate,
        startPosition: &claw_rotate_start_pulse,
    },
}

// initServos - создает сервоприводы и линкует их
func initServos() error {
    var (
        errorTextC [1000]C.char //текст ошибки. Передается как входной параметр,при
возникновении ошибки в эту переменную будет записан текст ошибки
        errCode    C.int        //код ошибки
    )
    //создаем 5 сервоприводов и линкуем их к пинам 0-4
    for i := 0; i < 5; i++ {
        // создаем компонент сервопривода с конкретной моделью как исполняемое
устройство и получаем дескриптор сервопривода
        errCode = C.RI_SDK_CreateModelComponent(C.CString("executor"),
C.CString("servodrive"), C.CString("mg90s"), servos[i].descriptor, &errorTextC[0])
        if errCode != 0 {
            return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
        }
        //связываем сервопривод с ШИМ,передаем дескриптор сервопривода и ШИМ
        errCode = C.RI_SDK_LinkServodriveToController(*servos[i].descriptor,
robohand.pwm, C.int(i), &errorTextC[0])
        if errCode != 0 {
            return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
        }
    }

    return nil
}

// initDevice - инициализация библиотеки и устройств
func initDevice() error {
    var (
        errorTextC [1000]C.char //текст ошибки
        errCode    C.int        //код ошибки
    )

    // Инициализируем SDK
```

```
    errCode = C.RI_SDK_InitSDK(logLevel, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    // создаем компонент ШИМ с конкретной моделью как соединитель и получаем
дескриптор компонента
    errCode = C.RI_SDK_CreateModelComponent(C.CString("connector"), C.CString("pwm"),
C.CString("pca9685"), &robohand.pwm, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    // создаем компонент i2c адатера
    // Здесь осуществлен примитивное определение подключенной модели адаптера
    // Сначала пробуем создать i2c адаптер модели ch341 и связать с ним ШИМ
    errCode = C.RI_SDK_CreateModelComponent(C.CString("connector"),
C.CString("i2c_adapter"), C.CString("ch341"), &robohand.i2c, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //связываем i2c адаптер с ШИМ по адресу 0x40
    errCode = C.RI_SDK_LinkPWMToController(robohand.pwm, robohand.i2c, 0x40,
&errorTextC[0])
    if errCode != 0 {
        // Если не получается то пробуем создать i2c адаптер модели cp2112
        errCode = C.RI_SDK_CreateModelComponent(C.CString("connector"),
C.CString("i2c_adapter"), C.CString("cp2112"), &robohand.i2c, &errorTextC[0])
        if errCode != 0 {
            return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
        }
        //связываем i2c адаптер с ШИМ по адресу 0x40
        errCode = C.RI_SDK_LinkPWMToController(robohand.pwm, robohand.i2c, 0x40,
&errorTextC[0])
        if errCode != 0 {
            return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
        }
    }
    // создаем компонент светодиода с конкретной моделью (ky016) как исполняемое
устройство и получаем дескриптор светодиода
    errCode = C.RI_SDK_CreateModelComponent(C.CString("executor"), C.CString("led"),
C.CString("ky016"), &robohand.led, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //связываем светодиод адаптер с ШИМ,передаем значения трех пинов к которым
подключен светодиод
    errCode = C.RI_SDK_LinkLedToController(robohand.led, robohand.pwm, 15, 14, 13,
```

```
&errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }

    //инициализируем сервоприводы
    err := initServos()
    if err != nil {
        return err
    }

    return nil
}

// startPosition - переводит сервопривод в стартовое положение
func startPosition(s servo) error {
    var (
        errorTextC [1000]C.char //текст ошибки
        errCode    C.int        //код ошибки
    )
    //выполняем поворот сервопривода в заданный угол,передаем дескриптор
сервопривода,значение угла
    errCode = C.RI_SDK_exec_ServoDrive_TurnByPulse(*s.descriptor, *s.startPosition,
&errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }

    //небольшая пауза для последовательного движения
    time.Sleep(500 * time.Millisecond)
    return nil
}

// startPositionAllServo - переводит все сервоприводы в стартовую позицию
func startPositionAllServo() error {
    var (
        errorTextC [1000]C.char //текст ошибки
        errCode    C.int        //код ошибки
    )

    //выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3
параметра цвета(RGB), и включаем асинхронный режим работы
    errCode = C.RI_SDK_exec_RGB_LED_SinglePulse(robohand.led, 255, 0, 0, 0, true,
&errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }

    //приводим сервоприводы в стартовое положение
```

```
    for i := 0; i < 5; i++ {
        err := startPosition(servos[i])
        if err != nil {
            return err
        }
    }

    //выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3
параметра цвета(RGB), и включаем асинхронный режим работы
    errCode = C.RI_SDK_exec_RGB_LED_SinglePulse(robohand.led, 0, 255, 0, 0, true,
&errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //небольшая пауза для последовательного движения
    time.Sleep(500 * time.Millisecond)
    return nil
}

// rotateBody - вращение тела в указанный угол
func rotateBody(angle, speed C.int) error {

    var (
        errorTextC [1000]C.char //текст ошибки
        errCode    C.int        //код ошибки
    )
    //выполняем мигание с заданной частотой,передаем дескриптор светодиода,3 параметра
цвета(RGB),частоту,продолжительность и включаем асинхронный режим работы
    errCode = C.RI_SDK_exec_RGB_LED_FlashingWithFrequency(robohand.led, 0, 255, 0, 5,
0, true, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //выполняем поворот на заданный угол,передаем дескриптор тела,угол,скорость и
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.body, angle, speed, false,
&errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.clawRotate, C.int(45), speed,
false, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
```

```
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.clawRotate, C.int(-45), speed,
false, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    return nil
}

// get - берет кубик
func get() error {
    var (
        errorTextC [1000]C.char //текст ошибки
        errCode    C.int        //код ошибки
    )
    //выполняем мерцание светодиодом,передаем дескриптор светодиода,3 параметра
цвета(RGB),продолжительность,кол-во повторений и включаем асинхронный режим работы
    errCode = C.RI_SDK_exec_RGB_LED_Flicker(robohand.led, C.int(0), C.int(0),
C.int(255), C.int(500), C.int(0), true, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.arrowR, arrowR_over_cube_position,
speed, false, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.arrowL, arrowL_over_cube_position,
speed, false, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.claw, claw_unclenched_position,
speed, false, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.arrowR, (arrowR_cube_position -
arrowR_over_cube_position), speed, false, &errorTextC[0])
```

```
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.arrowL, (arrowL_cube_position -
arrowL_over_cube_position), speed, false, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.claw,
(-1)*claw_unclenched_position, speed, false, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.arrowR, (-1)*arrowR_cube_position,
speed, false, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.arrowL, (-1)*arrowL_cube_position,
speed, false, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    return nil
}

// put - кладет кубик
func put() error {
    var (
        errorTextC [1000]C.char //текст ошибки
        errCode    C.int        //код ошибки
    )
    //выполняем мерцание светодиодом,передаем дескриптор светодиода,3 параметра
цвета(RGB),продолжительность,кол-во повторений и включаем асинхронный режим работы
    errCode = C.RI_SDK_exec_RGB_LED_Flicker(robohand.led, C.int(0), C.int(0),
C.int(255), C.int(500), C.int(0), true, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
```

```
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.arrowR, arrowR_cube_position,
speed, false, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.arrowL, arrowL_cube_position,
speed, false, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.claw, claw_unclenched_position,
speed, false, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.arrowR, (-1)*arrowR_cube_position,
speed, false, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.arrowL, (-1)*arrowL_cube_position,
speed, false, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errCode = C.RI_SDK_exec_ServoDrive_Turn(robohand.claw,
(-1)*claw_unclenched_position, speed, false, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }

    return nil
}

// destructServos - уничтожает сервоприводы
```

```
func destructServos() error {
    var (
        errorTextC [1000]C.char //текст ошибки
        errCode    C.int        //код ошибки
    )
    for i := 0; i < 5; i++ {
        errCode = C.RI_SDK_DestroyComponent(*servos[i].descriptor, &errorTextC[0])
        if errCode != 0 {
            return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
        }
    }

    return nil
}

// destruct - уничтожает все компоненты и библиотеку
func destruct() error {

    var (
        errorTextC [1000]C.char //текст ошибки
        errCode    C.int        //код ошибки
    )

    //выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3
параметра цвета(RGB), и включаем асинхронный режим работы
    errCode = C.RI_SDK_exec_RGB_LED_SinglePulse(robohand.led, C.int(255), C.int(0),
C.int(0), C.int(0), true, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //уничтожаем сервоприводы
    err := destructServos()
    if err != nil {
        return err
    }
    //останавливаем свечение светодиода с определенным дескриптором
    errCode = C.RI_SDK_exec_RGB_LED_Stop(robohand.led, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //удаляем компонент светодиода по дескриптору
    errCode = C.RI_SDK_DestroyComponent(robohand.led, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //сбрасываем все порты на ШИМ
    errCode = C.RI_SDK_sigmod_PWM_ResetAll(robohand.pwm, &errorTextC[0])
    if errCode != 0 {
```

```
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //удаляем компонент ШИМ
    errCode = C.RI_SDK_DestroyComponent(robohand.pwm, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //удаляем компонент i2c
    errCode = C.RI_SDK_DestroyComponent(robohand.i2c, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    //удаляем sdk со всеми компонентами в реестре компонентов
    errCode = C.RI_SDK_DestroySDK(true, &errorTextC[0])
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode,
C.GoString(&errorTextC[0]))
    }
    return nil
}

// start - запускает демо программу
func start() error {
    // Инициализируем библиотеку и компоненты
    err := initDevice()
    if err != nil {
        return err
    }

    // Приводим сервоприводы к стартовой позиции
    err = startPositionAllServo()
    if err != nil {
        return err
    }

    // Двигаем тело к местонахождению кубика
    err = rotateBody(45, speed)
    if err != nil {
        return err
    }

    // Берем кубик
    err = get()
    if err != nil {
        return err
    }

    //Двигаем тело к новому местонахождению кубика
    err = rotateBody(-90, speed)
```

```
    if err != nil {
        return err
    }

    //Кладем кубик
    err = put()
    if err != nil {
        return err
    }

    // Возвращаем тело в стартовое положение
    err = rotateBody(45, speed)
    if err != nil {
        return err
    }

    // Уничтожаем компоненты и библиотеку
    err = destruct()
    if err != nil {
        return err
    }
    return nil
}

func main() {
    //запускаем программу
    err := start()
    if err != nil {
        fmt.Println(err.Error())
        return
    }

    return
}
```

Подключение RI_SDK gRPC микросервиса к приложению на Golang

Создадим Golang файл и файл с для перечисления внешних зависимостей

Команда для Linux:

```
touch grpc_client.go
touch go.mod
```

Команда для Windows:

```
type nul > grpc_client.go
```

```
type nul > go.mod
```

Если вы используете текстовый редактор,то необходимо изменить кодировку `grpc_client.go` на `UTF-8 без BOM, либо использовать IDE (VsCode,etc.).`

Структура проекта

. workspace/
--- > grpc_client.go
--- > risdk.bin/.exe
--- > go.mod

В файле go.mod укажем что мы будем использовать пакет go-risdk последней версии

```
module mod

go 1.17

require github.com/rbs-ri/go-risdk latest
```

Подключим пакет в go файле grpc_client.go
Создадим компонент внутри функции main

```
package main

import (
	"fmt"
	"github.com/rbs-ri/go-risdk"
)

func main() {
	//Открываем соединение для работы с API SDK по gRPC
	client := risdk.GetClientRPC()
	//Закрываем соединение с gRPC
	defer client.Client.Kill()

	// Инициализруем SDK
	errorText, errorCode, err := client.RoboSdkApi.RI_SDK_InitSDK(2)
	if errorCode != 0 || err != nil {
		fmt.Printf("\n Не удалось инициализировать SDK. код ошибки:%d, описание:%s, ошибка:%s", errorCode, errorText, err)
		return
	}

	// Создаем базовый компонент SDK
	descriptor, errorText, errorCode, err := client.RoboSdkApi.RI_SDK_CreateBasic()
	if errorCode != 0 || err != nil {
		fmt.Printf("\n Не удалось создать базовый компонент. код ошибки:%d, описание:%s, ошибка:%s", errorCode, errorText, err)
		return
	}
```

```
      fmt.Printf("Компонент с дескриптором:%d создан \n", descriptor)
}
```

Обновим зависимости внутри пакета:

```
go mod tidy
```

Выполним компиляцию:

```
go build -o grpc_client.bin grpc_client.go
```

Значение **grpc_client.go** указывает на файл, который необходимо собрать
Значение **-o grpc_client.bin** указывает на путь к скомпиленному файлу

Получаем результат выполнения собранной программы
Теперь запустим скрипт. Результат успешной работы должен выглядеть следующим образом

```
-bash#$ ./grpc_client.bin
Current path:  /home/mihail/Documents/projects_go/src/ri_sdk
2021-11-08T15:15:04.817+0400 [DEBUG] plugin: starting plugin:
path=/home/mihail/Documents/projects_go/src/ri_sdk/risdk.bin
args=[/home/mihail/Documents/projects_go/src/ri_sdk/risdk.bin]
2021-11-08T15:15:04.818+0400 [DEBUG] plugin: plugin started:
path=/home/mihail/Documents/projects_go/src/ri_sdk/risdk.bin pid=9570
2021-11-08T15:15:04.818+0400 [DEBUG] plugin: waiting for RPC address:
path=/home/mihail/Documents/projects_go/src/ri_sdk/risdk.bin
2021-11-08T15:15:04.823+0400 [DEBUG] plugin: using plugin: version=1
2021-11-08T15:15:04.823+0400 [DEBUG] plugin.risdk.bin: plugin address:
address=/tmp/plugin1567377908 network=unix timestamp=2021-11-08T15:15:04.823+0400
2021-11-08T15:15:04.824+0400 [TRACE] plugin.stdio: waiting for stdio data
Компонент с дескриптором:1 создан
2021-11-08T15:15:04.827+0400 [DEBUG] plugin.stdio: received EOF, stopping recv loop:
err="rpc error: code = Unavailable desc = transport is closing"
2021-11-08T15:15:04.827+0400 [DEBUG] plugin: plugin process exited:
path=/home/mihail/Documents/projects_go/src/ri_sdk/risdk.bin pid=9570
2021-11-08T15:15:04.827+0400 [DEBUG] plugin: plugin exited

-bash#$
```

Пример программы на Golang, использующий RI_SDK gRPC микросервис

Описание

Данная программа является примером, использования библиотеки. Она реализует простой сценарий поведения робота. Сперва все сервоприводы встают в условное стартовое положение. После робот берет кубик с позиции слева и перемещает его на позицию справа. После сервоприводы возвращаются в первоначальное положение.

Одновременно с движением робота, происходит свечение светодиода. Светодиод горит красным цветом пока сервоприводы приводятся к стартовому положению, после загорается зеленым. Далее светодиоды моргает зеленым при передвижении тела робота и мерцает синим, пока робот кладет/берет кубик.

Структура проекта

. workspace/
--- > demo.go
--- > risdk.exe/.bin
--- > go.mod
--- > CH341DLL.DLL/ CH341DLLA64.DLL
--- > SLABHIDDevice.dll
--- > SLABHIDtoSMBus.dll

Реализация

Реализация данной программы представлена ниже. Ее также можно скачать по ссылке на репозиторий. Вы можете скопировать данный пример и запустить у себя.

```
package main

import (
    "fmt"
    "time"

    "github.com/rbs-ri/go-risdk"
)

var (
    body_start_pulse        int64 = 1500// стартовая позиция тела
    arrowR_start_pulse      int64 = 2000 // стартовая возиция правой стрелы
    arrowL_start_pulse      int64 = 1000 // стартовая позиция левой стрелы
    claw_start_pulse        int64 = 1000 // стартовая позиция клешни
    claw_rotate_start_pulse int64 = 1500 // стартовая позиция поворота клешни

    arrowR_over_cube_position int64 = -30 // позиция правой стрелы над кубиком
    arrowL_over_cube_position int64 = -10 // позиция левой стрелы над кубиком
    claw_unclenched_position  int64 = 80  // позиция открытой клешни
    arrowR_cube_position      int64 = -75 // позиция правой стрелы на месте кубика
    arrowL_cube_position      int64 = 55  // позиция левой стрелы на месте кубика

    logLevel int64            = 2   // уровень логирования
    speed    int64            = 100 // скорость в градусах в секунду
    client   *risdk.ClientRPC       // клиент
)

// структура дескрипторов устройства
type device struct {
    i2c        int64 // дескриптор i2c
```

```
    pwm         int64 // дескриптор pwm
    body        int64 // дескриптор сервопривода тела
    claw        int64 // дескриптор сервопривода клешни
    arrowR      int64 // дескриптор сервопривода правой стрелы
    arrowL      int64 // дескриптор сервопривода левой стрелы
    clawRotate int64 // дескриптор сервопривода поворота клешни
    led         int64 // дескриптор светодиода
}

// инициализация структуры устройства
var robohand device

type servo struct {
    descriptor    *int64 // дескриптор сервопривода
    startPosition *int64 // стартовая позиция сервопривода
}

// массив дескрипторов сервоприводов
var servos = [5]servo{
    {
        descriptor:    &robohand.body,
        startPosition: &body_start_pulse,
    },
    {
        descriptor:    &robohand.claw,
        startPosition: &claw_start_pulse,
    },
    {
        descriptor:    &robohand.arrowR,
        startPosition: &arrowR_start_pulse,
    },
    {
        descriptor:    &robohand.arrowL,
        startPosition: &arrowL_start_pulse,
    },
    {
        descriptor:    &robohand.clawRotate,
        startPosition: &claw_rotate_start_pulse,
    },
}

// initServos - создает сервоприводы и линкует их
func initServos() error {
    var (
        errorText string // текст ошибки. Передается как входной параметр,при
возникновении ошибки в эту переменную будет записан текст ошибки
        errCode   int64  // код ошибки
    )
    //создаем 5 сервоприводов и линкуем их к пинам 0-4
    for i := 0; i < 5; i++ {
        // создаем компонент сервопривода с конкретной моделью как исполняемое
устройство и получаем дескриптор сервопривода
```

```
        *servos[i].descriptor, errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_CreateModelComponent("executor", "servodrive", "mg90s")
        if errCode != 0 {
            return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
        }
        //связываем сервопривод с ШИМ,передаем дескрипторы сервопривода и ШИМ
        errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_LinkServodriveToController(*servos[i].descriptor,
robohand.pwm, int64(i))
        if errCode != 0 {
            return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
        }
    }

    return nil
}

// initDevice - инициализация библиотеки и устройств
func initDevice() error {
    var (
        errorText string //текст ошибки
        errCode   int64  //код ошибки
        err       error  //ошибка
    )

    // Инициализируем SDK
    errorText, errCode, err = client.RoboSdkApi.RI_SDK_InitSDK(logLevel)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    // создаем компонент ШИМ с конкретной моделью как соединитель и получаем
дескриптор компонента
    robohand.pwm, errorText, errCode, err =
client.RoboSdkApi.RI_SDK_CreateModelComponent("connector", "pwm", "pca9685")
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    // создаем компонент i2c адатера
    // Здесь осуществлен примитивное определение подключенной модели адаптера
    // Сначала пробуем создать i2c адаптер модели ch341 и связать с ним ШИМ
    robohand.i2c, errorText, errCode, err =
client.RoboSdkApi.RI_SDK_CreateModelComponent("connector", "i2c_adapter", "ch341")
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }

    //связываем i2c адаптер с ШИМ по адресу 0x40
    errorText, errCode, err =
client.RoboSdkApi.RI_SDK_LinkPWMToController(robohand.pwm, robohand.i2c, 0x40)
    if errCode != 0 {
        // Если не получается то пробуем создать i2c адаптер модели cp2112
        robohand.i2c, errorText, errCode, err =
```

```
client.RoboSdkApi.RI_SDK_CreateModelComponent("connector", "i2c_adapter", "cp2112")
        if errCode != 0 {
            return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
        }

        //связываем i2c адаптер с ШИМ по адресу 0x40
        errorText, errCode, err =
client.RoboSdkApi.RI_SDK_LinkPWMToController(robohand.pwm, robohand.i2c, 0x40)
        if errCode != 0 {
            return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
        }
    }
    // создаем компонент светодиода с конкретной моделью (ky016) как исполняемое
устройство и получаем дескриптор светодиода
    robohand.led, errorText, errCode, err =
client.RoboSdkApi.RI_SDK_CreateModelComponent("executor", "led", "ky016")
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //связываем светодиод адаптер с ШИМ,передаем значения трех пинов к которым
подключен светодиод
    errorText, errCode, err =
client.RoboSdkApi.RI_SDK_LinkLedToController(robohand.led, robohand.pwm, 15, 14, 13)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //инициализируем сервоприводы
    err = initServos()
    if err != nil {
        return err
    }

    return nil
}

// startPosition - переводит сервопривод в стартовое положение
func startPosition(s servo) error {
    var (
        errorText string
        errCode   int64
    )
    //выполняем поворот сервопривода в заданный угол,передаем дескриптор
сервопривода,значение угла
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_TurnByPulse(*s.descriptor, *s.startPosition)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }

    time.Sleep(500 * time.Millisecond)
    return nil
}
```

```
// startPositionAllServo - переводит все сервоприводы в стартовую позицию
func startPositionAllServo() error {
    var (
        errorText string //текст ошибки
        errCode   int64  //код ошибки
    )
    //выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3
параметра цвета(RGB), и включаем асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_RGB_LED_SinglePulse(robohand.led, 255, 0, 0, 0, true)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //приводим сервоприводы в стартовое положение
    for i := 0; i < 5; i++ {
        err := startPosition(servos[i])
        if err != nil {
            return err
        }
    }
    //выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3
параметра цвета(RGB), и включаем асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_RGB_LED_SinglePulse(robohand.led, 0, 255, 0, 0, true)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //небольшая пауза для последовательного движения
    time.Sleep(500 * time.Millisecond)
    return nil
}

// rotateBody - вращение тела в указанный угол
func rotateBody(angle, speed int64) error {
    var (
        errorText string //текст ошибки
        errCode   int64  //код ошибки
    )
    //выполняем мигание с заданной частотой,передаем дескриптор светодиода,3 параметра
цвета(RGB),частоту,продолжительность и включаем асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_RGB_LED_FlashingWithFrequency(robohand.led, 0, 255, 0, 5,
0, true)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор тела,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.body, angle, speed, false)
    if errCode != 0 {
```

```
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.clawRotate, 45, speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.clawRotate, -45, speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    return nil
}

// get - берет кубик
func get() error {
    var (
        errorText string //текст ошибки
        errCode   int64  //код ошибки
    )
    //выполняем мерцание светодиодом,передаем дескриптор светодиода,3 параметра
цвета(RGB),продолжительность,кол-во повторений и включаем асинхронный режим работы
    errorText, errCode, _ = client.RoboSdkApi.RI_SDK_Exec_RGB_LED_Flicker(robohand.led,
0, 0, 255, 500, 0, true)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.arrowR,
arrowR_over_cube_position, speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.arrowL,
arrowL_over_cube_position, speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.claw, claw_unclenched_position,
```

```
speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.arrowR, (arrowR_cube_position -
arrowR_over_cube_position), speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.arrowL, (arrowL_cube_position -
arrowL_over_cube_position), speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.claw,
(-1)*claw_unclenched_position, speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.arrowR,
(-1)*arrowR_cube_position, speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.arrowL,
(-1)*arrowL_cube_position, speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }

    return nil
}

// put - кладет кубик
func put() error {
    var (
        errorText string //текст ошибки
```

```
        errCode   int64  //код ошибки
    )

    //выполняем мерцание светодиодом,передаем дескриптор светодиода,3 параметра
цвета(RGB),продолжительность,кол-во повторений и включаем асинхронный режим работы
    errorText, errCode, _ = client.RoboSdkApi.RI_SDK_Exec_RGB_LED_Flicker(robohand.led,
0, 0, 255, 500, 0, true)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.arrowR, arrowR_cube_position,
speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.arrowL, arrowL_cube_position,
speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.claw, claw_unclenched_position,
speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.arrowR,
(-1)*arrowR_cube_position, speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.arrowL,
(-1)*arrowL_cube_position, speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
```

```
    errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_Exec_ServoDrive_Turn(robohand.claw,
(-1)*claw_unclenched_position, speed, false)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }

    return nil
}

// destructServos - уничтожает сервоприводы
func destructServos() error {
    var (
        errorText string //текст ошибки
        errCode   int64  //код ошибки
    )

    for i := 0; i < 5; i++ {
        errorText, errCode, _ =
client.RoboSdkApi.RI_SDK_DestroyComponent(*servos[i].descriptor)
        if errCode != 0 {
            return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
        }
    }

    return nil
}

// destruct - уничтожает все компоненты и библиотеку
func destruct() error {

    var (
        errorText string //текст ошибки
        errCode   int64  //код ошибки
        err       error
    )
    //выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3
параметра цвета(RGB), и включаем асинхронный режим работы
    errorText, errCode, err =
client.RoboSdkApi.RI_SDK_Exec_RGB_LED_SinglePulse(robohand.led, 255, 0, 0, 0, true)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //уничтожаем сервоприводы
    err = destructServos()
    if err != nil {
        return err
    }
    //останавливаем свечение светодиода с определенным дескриптором
    errorText, errCode, err = client.RoboSdkApi.RI_SDK_Exec_RGB_LED_Stop(robohand.led)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
```

```
    }
    //удаляем компонент светодиода по дескриптору
    errorText, errCode, err = client.RoboSdkApi.RI_SDK_DestroyComponent(robohand.led)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //сбрасываем все порты на ШИМ
    errorText, errCode, err =
client.RoboSdkApi.RI_SDK_Sigmod_PWM_ResetAll(robohand.pwm)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //удаляем компонент ШИМ
    errorText, errCode, err = client.RoboSdkApi.RI_SDK_DestroyComponent(robohand.pwm)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //удаляем компонент i2c
    errorText, errCode, err = client.RoboSdkApi.RI_SDK_DestroyComponent(robohand.i2c)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }
    //удаляем sdk со всеми компонентами в реестре компонентов
    errorText, errCode, err = client.RoboSdkApi.RI_SDK_DestroySDK(true)
    if errCode != 0 {
        return fmt.Errorf("errorCode:%d - errorText:%s", errCode, errorText)
    }

    return nil
}

// start - запускает демо программу
func start() error {
    // Инициализируем библиотеку и компоненты
    err := initDevice()
    if err != nil {
        return err
    }

    // Приводим сервоприводы к стартовой позиции
    err = startPositionAllServo()
    if err != nil {
        return err
    }

    // Двигаем тело к местонахождению кубика
    err = rotateBody(45, speed)
    if err != nil {
        return err
    }

    // Берем кубик
```

```
    err = get()
    if err != nil {
        return err
    }

    //Двигаем тело к новому местонахождению кубика
    err = rotateBody(-90, speed)
    if err != nil {
        return err
    }

    //Кладем кубик
    err = put()
    if err != nil {
        return err
    }

    // Возвращаем тело в стартовое положение
    err = rotateBody(45, speed)
    if err != nil {
        return err
    }

    // Уничтожаем компоненты и библиотеку
    err = destruct()
    if err != nil {
        return err
    }
    return nil
}

func main() {
    //Открываем соединение для работы с API SDK по RPC
    client = risdk.GetClientRPC()
    //Закрываем соединение с RPC
    defer client.Client.Kill()
    //запускаем программу
    err := start()
    if err != nil {
        fmt.Println(err.Error())
        return
    }

    return
}
```

Python

Подключение RI_SDK к приложению на Python

Создадим Python файл.

Команда для Linux

```
touch shared_client.py
```

Команда для Windows

```
type nul > shared_client.py
```

Если вы используете текстовый редактор,то необходимо изменить кодировку `shared_client.py` на `UTF-8 без BOM, либо использовать IDE (VsCode,etc.).`

Структура проекта

. workspace/
    --- > shared_client.py
    --- > librisdk.h
    --- > librisdk.so/.dll

Подключим библиотеку

```
from ctypes import *
import platform

# Подключаем внешнюю библиотеку для работы с SDK
platform=platform.system()
if platform=="Windows":
    lib = cdll.LoadLibrary("librisdk.dll")
if platform=="Linux":
    lib = cdll.LoadLibrary("librisdk.so")

def main():

    print("Success")

main()
```

Для запуска скрипта выполним в терминале следующую команду.

```
python3 shared_client.py
```

Создадим компонент внутри функции main

```
    # Инициализируем SDK
    errCode, errText = RI_SDK_InitSDK_()
    if errCode != 0:
        # Выведем текст и код ошибки в терминал, при возникновении
        print(errCode, decodeErr(errText))
        return

    # Создаем базовый компонент
    descriptor, errCode, errText = RI_SDK_CreateBasic_()
    if errCode != 0:
        # Выведем текст и код ошибки в терминал, при возникновении
        print(errCode, decodeErr(errText))
        return

    # Выведем полученный дескриптор компонента
    print("descriptor:", descriptor, "--")

# для получения текста ошибки
def decodeErr(err):
    return err.value.decode('utf-8')


# RI_SDK_InitSDK_ -  вызов функции из Shared Object для инициализации SDK
def RI_SDK_InitSDK_():
    logLevel = 2
    errTextC = c_char_p()  # Текст ошибки. C type: char*

    # Указываем типы аргументов для функции
    lib.RI_SDK_InitSDK.argtypes = [c_int, POINTER(c_char_p)]
    # Вызываем функцию инициализации
    errCode = lib.RI_SDK_InitSDK(logLevel, errTextC)
    if errCode != 0:
        return errCode, errTextC

    return errCode, errTextC.value


# RI_SDK_CreateBasic_ - вызов функции из Shared Object для создания компонента
def RI_SDK_CreateBasic_():
    descriptor = c_int()  # Дескриптор компонента. C type: int

    errTextC = c_char_p()  # Текст ошибки.  C type: char*
    # Указываем типы аргументов
    lib.RI_SDK_CreateBasic.argtypes = [POINTER(c_int), POINTER(c_char_p)]
    # Создаем компонент
    errCode = lib.RI_SDK_CreateBasic(descriptor, errTextC)
    if errCode != 0:
        return descriptor.value, errCode, errTextC

    return descriptor.value, errCode, errTextC.value
```

Компилируем и запустим скрипт. Результат успешной работы должен выглядеть следующим образом

```
-bash#$ python3 shared_client.py
Success
descriptor:1--
-bash#$
```

Пример программы на PHP использующий RI_SDK

Описание

Данная программа является примером, использования библиотеки. Она реализует простой сценарий поведения робота. Сперва все сервоприводы встают в условное стартовое положение. После робот берет кубик с позиции слева и перемещает его на позицию справа. После сервоприводы возвращаются в первоначальное положение.
Одновременно с движением робота, происходит свечение светодиода. Светодиод горит красным цветом пока сервоприводы приводятся к стартовому положению, после загорается зеленым. Далее светодиоды моргает зеленым при передвижении тела робота и мерцает синим, пока робот кладет/берет кубик.

Структура проекта

. workspace/
- --- > demo.php
- --- > librisdk.h
- --- > librisdk.dll/so
- --- > CH341DLL.DLL/ CH341DLLA64.DLL
- --- > SLABHIDDevice.dll
- --- > SLABHIDtoSMBus.dll

Реализация

Реализация данной программы представлена ниже. Ее также можно скачать по ссылке на репозиторий. Вы можете скопировать данный пример и запустить у себя.

```
from ctypes import *
import platform
import time

# Подключаем внешнюю библиотеку для работы с SDK
platform = platform.system()
if platform == "Windows":
    lib = cdll.LoadLibrary("./librisdk.dll")
if platform == "Linux":
    lib = cdll.LoadLibrary("./librisdk.so")
```

```
# Инициализируем глобальные переменные
body_start_pulse = 1500 # стартовая позиция тела
arrowR_start_pulse = 2000  # стартовая позиция правой стрелы
arrowL_start_pulse = 1000  # стартовая позиция левой стрелы
claw_start_pulse = 1000  # стартовая позиция клешни
claw_rotate_start_pulse = 1500  # стартовая позиция поворота клешни

arrowR_over_cube_position = -30  # позиция правой стрелы над кубиком
arrowL_over_cube_position = -10  # позиция левой стрелы над кубиком
claw_unclenched_position = 80  # позиция открытой клешни
arrowR_cube_position = -75  # позиция правой стрелы на месте кубика
arrowL_cube_position = 55  # позиция левой стрелы на месте кубика

logLevel = 2  # уровень логирования
speed = 100  # скорость в градусах в секунду

# структура дескрипторов устройства: ключ-имя - значение дескриптора
device = {
    "i2c": c_int(),  # дескриптор i2c
    "pwm": c_int(),  # дескриптор pwm
    "body": c_int(),  # дескриптор сервопривода тела
    "claw": c_int(),  # дескриптор сервопривода клешни
    "arrowR": c_int(),  # дескриптор сервопривода правой стрелы
    "arrowL": c_int(),  # дескриптор сервопривода левой стрелы
    "clawRotate": c_int(),  # дескриптор сервопривода поворота клешни
    "led": c_int(),  # дескриптор светодиода
}

# массив дескрипторов сервоприводов и их стартовых позиций
servos = [
    {"descriptor": device["body"],
     "start_position": body_start_pulse
     },
    {"descriptor": device["claw"],
     "start_position": claw_start_pulse
     },
    {"descriptor": device["arrowR"],
     "start_position": arrowR_start_pulse
     },
    {"descriptor": device["arrowL"],
     "start_position": arrowL_start_pulse
     },
    {"descriptor": device["clawRotate"],
     "start_position": claw_rotate_start_pulse
     },
]

# Указываем типы аргументов для функций
lib.RI_SDK_InitSDK.argtypes = [c_int, POINTER(c_char_p)]
lib.RI_SDK_CreateModelComponent.argtypes = [c_char_p, c_char_p, c_char_p,
POINTER(c_int), POINTER(c_char_p)]
lib.RI_SDK_LinkPWMToController.argtypes = [c_int, c_int, c_uint8, POINTER(c_char_p)]
```

```
lib.RI_SDK_LinkLedToController.argtypes = [c_int, c_int, c_int, c_int, c_int,
POINTER(c_char_p)]
lib.RI_SDK_LinkServodriveToController.argtypes = [c_int, c_int, c_int,
POINTER(c_char_p)]
lib.RI_SDK_exec_RGB_LED_SinglePulse.argtypes = [c_int, c_int, c_int, c_int, c_int,
c_bool, POINTER(c_char_p)]
lib.RI_SDK_exec_ServoDrive_TurnByPulse.argtypes = [c_int, c_int, POINTER(c_char_p)]
lib.RI_SDK_DestroyComponent.argtypes = [c_int, POINTER(c_char_p)]
lib.RI_SDK_exec_RGB_LED_Stop.argtypes = [c_int, POINTER(c_char_p)]
lib.RI_SDK_sigmod_PWM_ResetAll.argtypes = [c_int, POINTER(c_char_p)]
lib.RI_SDK_DestroySDK.argtypes = [c_bool, POINTER(c_char_p)]
lib.RI_SDK_exec_RGB_LED_FlashingWithFrequency.argtypes = [c_int, c_int, c_int, c_int,
c_int, c_int, c_bool,
                                                          POINTER(c_char_p)]
lib.RI_SDK_exec_RGB_LED_Flicker.argtypes = [c_int, c_int, c_int, c_int, c_int, c_int,
c_bool, POINTER(c_char_p)]
lib.RI_SDK_exec_ServoDrive_Turn.argtypes = [c_int, c_int, c_int, c_bool,
POINTER(c_char_p)]


# для получения текста ошибки
def decodeErr(err):
    return err.value.decode()


# initServos - создает сервоприводы и линкует их
def initServos():

    errTextC = c_char_p()  # Текст ошибки. C type: char*
    errCode = c_int
    # создаем 5 сервоприводов и линкуем их к пинам 0-4
    for i in range(len(servos)):
        # создаем компонент сервопривода с конкретной моделью как исполняемое устройство
и получаем дескриптор сервопривода
        errCode = lib.RI_SDK_CreateModelComponent("executor".encode(),
"servodrive".encode(), "mg90s".encode(),
                                                  servos[i]["descriptor"], errTextC)
        if errCode != 0:
            return errCode, errTextC
        # связываем сервопривод с ШИМ,передаем дескриптор сервопривода и ШИМ
        errCode = lib.RI_SDK_LinkServodriveToController(servos[i]["descriptor"],
device["pwm"], i, errTextC)
        if errCode != 0:
            return errCode, errTextC
    return errCode, errTextC


# initDevice - инициализация библиотеки и устройств
def initDevice():
    errTextC = c_char_p()  # Текст ошибки. C type: char*
    # Вызываем функцию инициализации
    errCode = lib.RI_SDK_InitSDK(logLevel, errTextC)
```

```
    if errCode != 0:
        return errCode, errTextC
    # создаем компонент ШИМ с конкретной моделью как исполняемое устройство,получаем дескриптор сервопривода
    errCode = lib.RI_SDK_CreateModelComponent("connector".encode(), "pwm".encode(), "pca9685".encode(), device["pwm"],
                                              errTextC)
    if errCode != 0:
        return errCode, errTextC

    # создаем компонент i2c адатера
    # Здесь осуществлен примитивное определение подключенной модели адаптера
    # Сначала пробуем создать i2c адаптер модели ch341 и связать с ним ШИМ
    errCode = lib.RI_SDK_CreateModelComponent("connector".encode(), "i2c_adapter".encode(), "ch341".encode(),
                                              device["i2c"], errTextC)
    if errCode != 0:
        return errCode, errTextC

    # связываем i2c адаптер с ШИМ по адресу 0x40
    errCode = lib.RI_SDK_LinkPWMToController(device["pwm"], device["i2c"], c_uint8(0x40), errTextC)
    if errCode != 0:

        # Если не получается то пробуем создать i2c адаптер модели cp2112
        errCode = lib.RI_SDK_CreateModelComponent("connector".encode(), "i2c_adapter".encode(), "cp2112".encode(),
                                                  device["i2c"], errTextC)
        if errCode != 0:
            return errCode, errTextC
        # связываем i2c адаптер с ШИМ по адресу 0x40
        errCode = lib.RI_SDK_LinkPWMToController(device["pwm"], device["i2c"], c_uint8(0x40), errTextC)

        if errCode != 0:
            return errCode, errTextC



    # создаем компонент светодиода с конкретной моделью (ky016) как исполняемое устройство и получаем дескриптор светодиода
    errCode = lib.RI_SDK_CreateModelComponent("executor".encode(), "led".encode(), "ky016".encode(), device["led"],
                                              errTextC)
    if errCode != 0:
        return errCode, errTextC
    # связываем светодиод с ШИМ,передаем значения трех пинов к которым подключен светодиод
    errCode = lib.RI_SDK_LinkLedToController(device["led"], device["pwm"], 15, 14, 13, errTextC)
    if errCode != 0:
        return errCode, errTextC
```

```
    # инициализируем сервоприводы
    errCode, errTextC = initServos()
    if errCode != 0:
        return errCode, errTextC
    return errCode, errTextC.value


# startPosition - переводит сервопривод в стартовое положение
def startPosition(servo):
    errTextC = c_char_p()  # Текст ошибки. C type: char*
    errCode = 0
    # выполняем поворот сервопривода в заданный угол,передаем дескриптор
сервопривода,значение угла
    errCode = lib.RI_SDK_exec_ServoDrive_TurnByPulse(servo["descriptor"],
servo["start_position"], errTextC)
    if errCode != 0:
        return errCode, errTextC

    time.sleep(0.5)

    return errCode, errTextC


# startPositionAllServo - переводит все сервоприводы в стартовую позицию
def startPositionAllServo():
    errTextC = c_char_p()  # Текст ошибки. C type: char*
    # выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3
параметра цвета(RGB), и включаем асинхронный режим работы
    errCode = lib.RI_SDK_exec_RGB_LED_SinglePulse(device["led"], 255, 0, 0, 0,
c_bool(True), errTextC)
    if errCode != 0:
        return errCode, errTextC
    # приводим сервоприводы в стартовое положение
    for i in range(len(servos)):
        errCode, errTextC = startPosition(servos[i])
        if errCode != 0:
            return errCode, errTextC
    # выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3
параметра цвета(RGB), и включаем асинхронный режим работы
    errCode = lib.RI_SDK_exec_RGB_LED_SinglePulse(device["led"], 0, 255, 0, 0,
c_bool(True), errTextC)
    if errCode != 0:
        return errCode, errTextC
    # небольшая пауза для последовательного движения
    time.sleep(0.5)

    return errCode, errTextC.value


# destructServos - уничтожает сервоприводы
def destructServos():
    errTextC = c_char_p()  # Текст ошибки. C type: char*
```

```
    for i in range(len(servos)):
        errCode = lib.RI_SDK_DestroyComponent(servos[i]["descriptor"], errTextC)
        if errCode != 0:
            return errCode, errTextC

    return errCode, errTextC


# destruct - уничтожает все компоненты и библиотеку
def destruct():
    errTextC = c_char_p()  # Текст ошибки. C type: char*
    # выполняем одиночное свечение светодиодом,передаем дескриптор светодиода,3
параметра цвета(RGB), и включаем асинхронный режим работы
    errCode = lib.RI_SDK_exec_RGB_LED_SinglePulse(device["led"], 255, 0, 0, 0,
c_bool(True), errTextC)
    if errCode != 0:
        return errCode, errTextC
    # уничтожаем сервоприводы
    errCode, errTextC = destructServos()
    if errCode != 0:
        return errCode, errTextC
    # останавливаем свечение светодиода с определенным дескриптором
    errCode = lib.RI_SDK_exec_RGB_LED_Stop(device["led"], errTextC)
    if errCode != 0:
        return errCode, errTextC
    # удаляем компонент светодиода по дескриптору
    errCode = lib.RI_SDK_DestroyComponent(device["led"], errTextC)
    if errCode != 0:
        return errCode, errTextC
    # сбрасываем все порты на ШИМ
    errCode = lib.RI_SDK_sigmod_PWM_ResetAll(device["pwm"], errTextC)
    if errCode != 0:
        return errCode, errTextC
    # удаляем компонент ШИМ
    errCode = lib.RI_SDK_DestroyComponent(device["pwm"], errTextC)
    if errCode != 0:
        return errCode, errTextC
    # удаляем компонент i2c
    errCode = lib.RI_SDK_DestroyComponent(device["i2c"], errTextC)
    if errCode != 0:
        return errCode, errTextC
    # удаляем sdk со всеми компонентами в реестре компонентов
    errCode = lib.RI_SDK_DestroySDK(c_bool(True), errTextC)
    if errCode != 0:
        return errCode, errTextC

    return errCode, errTextC

# rotateBody - вращение тела в указанный угол
def rotateBody(angle, speed):
    errTextC = c_char_p()  # Текст ошибки. C type: char*
```

```
    # выполняем мигание с заданной частотой,передаем дескриптор светодиода,3 параметра цвета(RGB),частоту,продолжительность и включаем асинхронный режим работы
    errCode = lib.RI_SDK_exec_RGB_LED_FlashingWithFrequency(device["led"], 0, 255, 0, 5, 0, c_bool(True), errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор тела,угол,скорость и асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["body"], angle, speed, c_bool(False), errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["clawRotate"], 45, speed, c_bool(False), errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["clawRotate"], -45, speed, c_bool(False), errTextC)
    if errCode != 0:
        return errCode, errTextC

    return errCode, errTextC


# get - берет кубик
def get():
    errTextC = c_char_p()  # Текст ошибки. C type: char*
    # выполняем мерцание светодиодом,передаем дескриптор светодиода,3 параметра цвета(RGB),продолжительность,кол-во повторений и включаем асинхронный режим работы
    errCode = lib.RI_SDK_exec_RGB_LED_FlashingWithFrequency(device["led"], 0, 0, 255, 500, 0, c_bool(True), errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["arrowR"], arrowR_over_cube_position, speed, c_bool(False),
                                              errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["arrowL"], arrowL_over_cube_position, speed, c_bool(False),
                                              errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
```

```
асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["claw"], claw_unclenched_position,
speed, c_bool(False), errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["arrowR"], (arrowR_cube_position -
arrowR_over_cube_position),
                                              speed, c_bool(False), errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["arrowL"], (arrowL_cube_position -
arrowL_over_cube_position),
                                              speed, c_bool(False), errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["claw"], (-1) *
claw_unclenched_position, speed, c_bool(False),
                                              errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["arrowR"], (-1) *
arrowR_cube_position, speed, c_bool(False),
                                              errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["arrowL"], (-1) *
arrowL_cube_position, speed, c_bool(False),
                                              errTextC)
    if errCode != 0:
        return errCode, errTextC

    return errCode, errTextC


# put - кладет кубик
def put():
    errTextC = c_char_p()  # Текст ошибки. C type: char*
    # выполняем мерцание светодиодом,передаем дескриптор светодиода,3 параметра
цвета(RGB),продолжительность,кол-во повторений и включаем асинхронный режим работы
    errCode = lib.RI_SDK_exec_RGB_LED_Flicker(device["led"], 0, 0, 255, 500, 0,
c_bool(True), errTextC)
    if errCode != 0:
```

```
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["arrowR"], arrowR_cube_position,
speed, c_bool(False), errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["arrowL"], arrowL_cube_position,
speed, c_bool(False), errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["claw"], claw_unclenched_position,
speed, c_bool(False), errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["arrowR"], (-1) *
arrowR_cube_position, speed, c_bool(False),
                                              errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор стрелы,угол,скорость и
асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["arrowL"], (-1) *
arrowL_cube_position, speed, c_bool(False),
                                              errTextC)
    if errCode != 0:
        return errCode, errTextC
    # выполняем поворот на заданный угол,передаем дескриптор клешни,угол,скорость и
асинхронный режим работы
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(device["claw"], (-1) *
claw_unclenched_position, speed, c_bool(False),
                                              errTextC)
    if errCode != 0:
        return errCode, errTextC

    return errCode, errTextC


# start - запускает демо программу
def start():
    # Инициализируем библиотеку и компоненты
    errCode, errText = initDevice()
    if errCode != 0:
        return errCode, errText

    # Приводим сервоприводы к стартовой позиции
```

```
    errCode, errText = startPositionAllServo()
    if errCode != 0:
        return errCode, errText

    # Двигаем тело к местонахождению кубика
    errCode, errText = rotateBody(45, speed)
    if errCode != 0:
        return errCode, errText

    # Берем кубик
    errCode, errText = get()
    if errCode != 0:
        return errCode, errText

    # Двигаем тело к новому местонахождению кубика
    errCode, errText = rotateBody(-90, speed)
    if errCode != 0:
        return errCode, errText

    # Кладем кубик
    errCode, errText = put()
    if errCode != 0:
        return errCode, errText

    # Возвращаем тело в стартовое положение
    errCode, errText = rotateBody(45, speed)
    if errCode != 0:
        return errCode, errText

    # Уничтожаем компоненты и библиотеку
    errCode, errText = destruct()
    if errCode != 0:
        return errCode, errText
    return errCode, errText


# Главная функция запускающая все остальное
def main():
    errCode, errText = start()
    if errCode != 0:
        print(errCode, errText)
        return

    print("Success", device)


main()
```