



STM &  XILINX.

«Программирование» встраиваемых систем на базе микроконтроллеров и ПЛИС

Владимир Хрусталев
Email : v_crys@mail.ru

Часть I
Знакомство с STM32

План курса

1. Часть I. Программирование микроконтроллеров
 1. Основы STM32. Обзор средств разработки. Первая программа.
 2. Таймеры и прерывания. Разрабатываем секундомер.
 3. **Внешние интерфейсы STM32. «Связываем» микроконтроллер с компьютером.**
2. Часть II. Проектирование микросхем и их отладка на FPGA
 1. Знакомимся с FPGA и SystemVerilog. Обзор средств разработки. Первые опыты.
 2. Цифровая схемотехника. Синхронная и комбинаторная логика. Конечные автоматы. Разрабатываем светофор.
 3. Верификация. Разработка аппаратного модуля UART.
3. Часть III. (опционально) О разработке процессоров. Архитектура и микроархитектура. «Живой» пример: процессор Syntacore SCR1

Занятие №3

Внешние интерфейсы STM32

Занятие №3

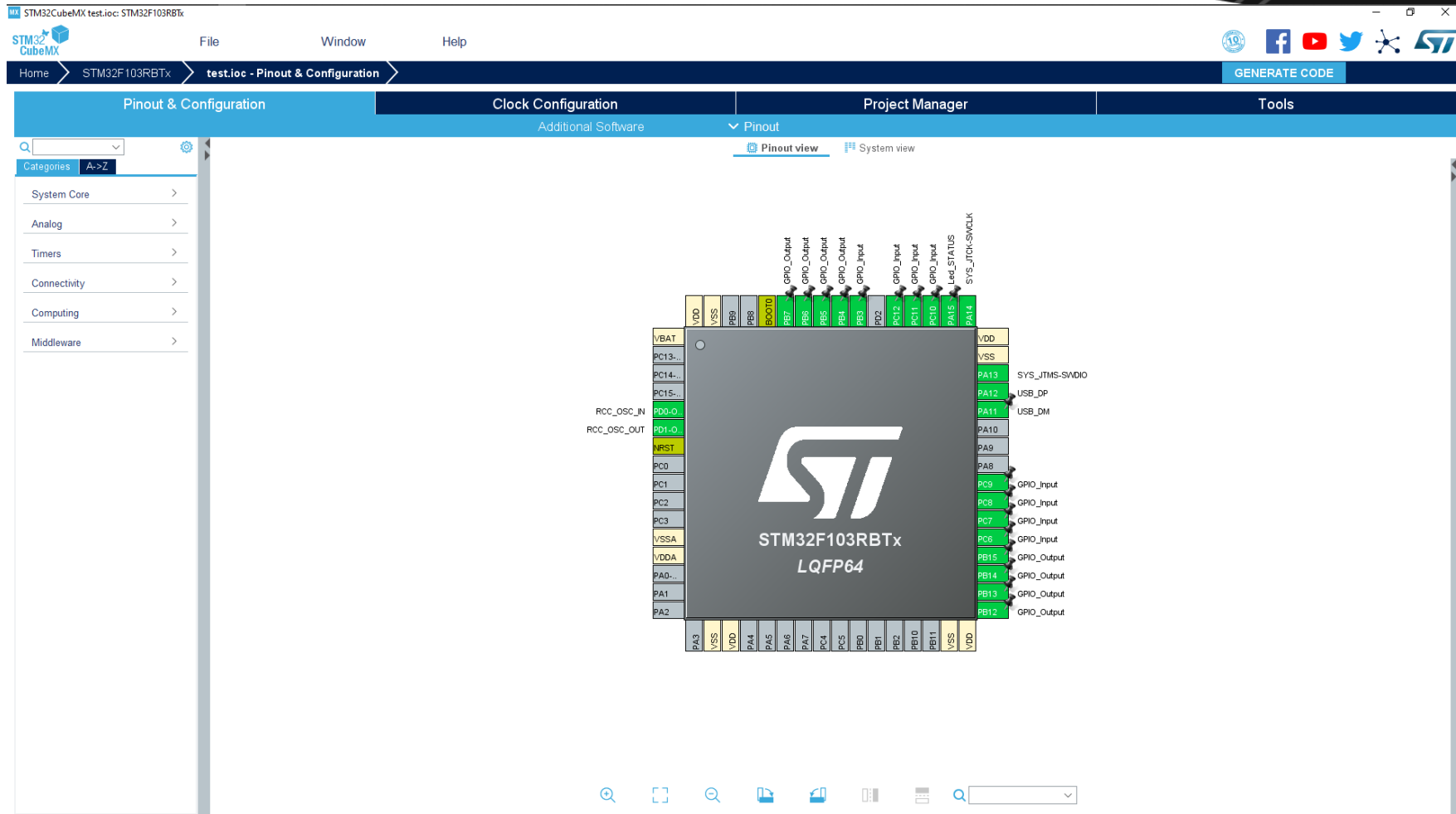
Связываем МК с компьютером

План занятия

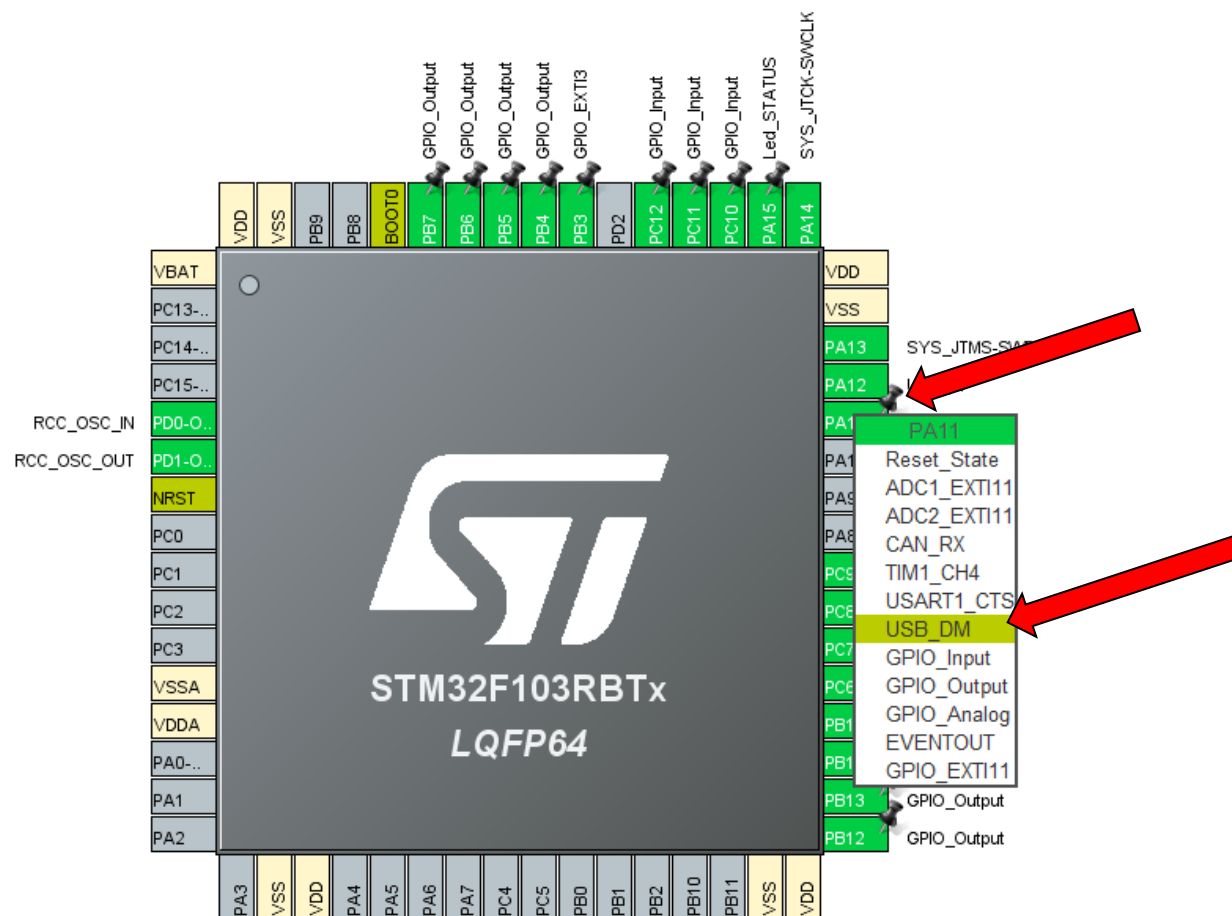
Разработка микропрограммного обеспечения для МК

Разработка ПО для компьютера

Берем за основу проект с предыдущего урока



Включаем USB



Настройки USB можно оставить по-умолчанию

The screenshot displays the STM32CubeMX configuration tool. On the left, the 'Categories' pane shows a tree view with 'USB' selected under the 'Connectivity' category. The main 'Mode' pane shows 'Device (FS)' selected. Below this, the 'Configuration' section includes a 'Reset Configuration' button and tabs for 'Parameter Settings', 'User Constants', 'NVIC Settings', and 'GPIO Settings'. The 'Parameter Settings' tab is active, showing a search bar and a list of parameters. The 'Basic Parameters' section shows 'Speed' set to 'Full Speed 12MBit/s'. The 'Power Parameters' section shows 'Low Power', 'Link Power Management', and 'Battery Charging' all set to 'Disabled'.

Categories A-Z

- System Core >
- Analog >
- Timers >
- Connectivity >
 - CAN
 - ⚠ I2C1
 - ⚠ I2C2
 - SPI1
 - ⊗ SPI2
 - USART1
 - USART2
 - ⚠ USART3
 - ✓ USB
- Computing >
- Middleware >
 - FATFS
 - FREERTOS
 - USB_DEVICE

Mode

✓ Device (FS)

Configuration

Reset Configuration

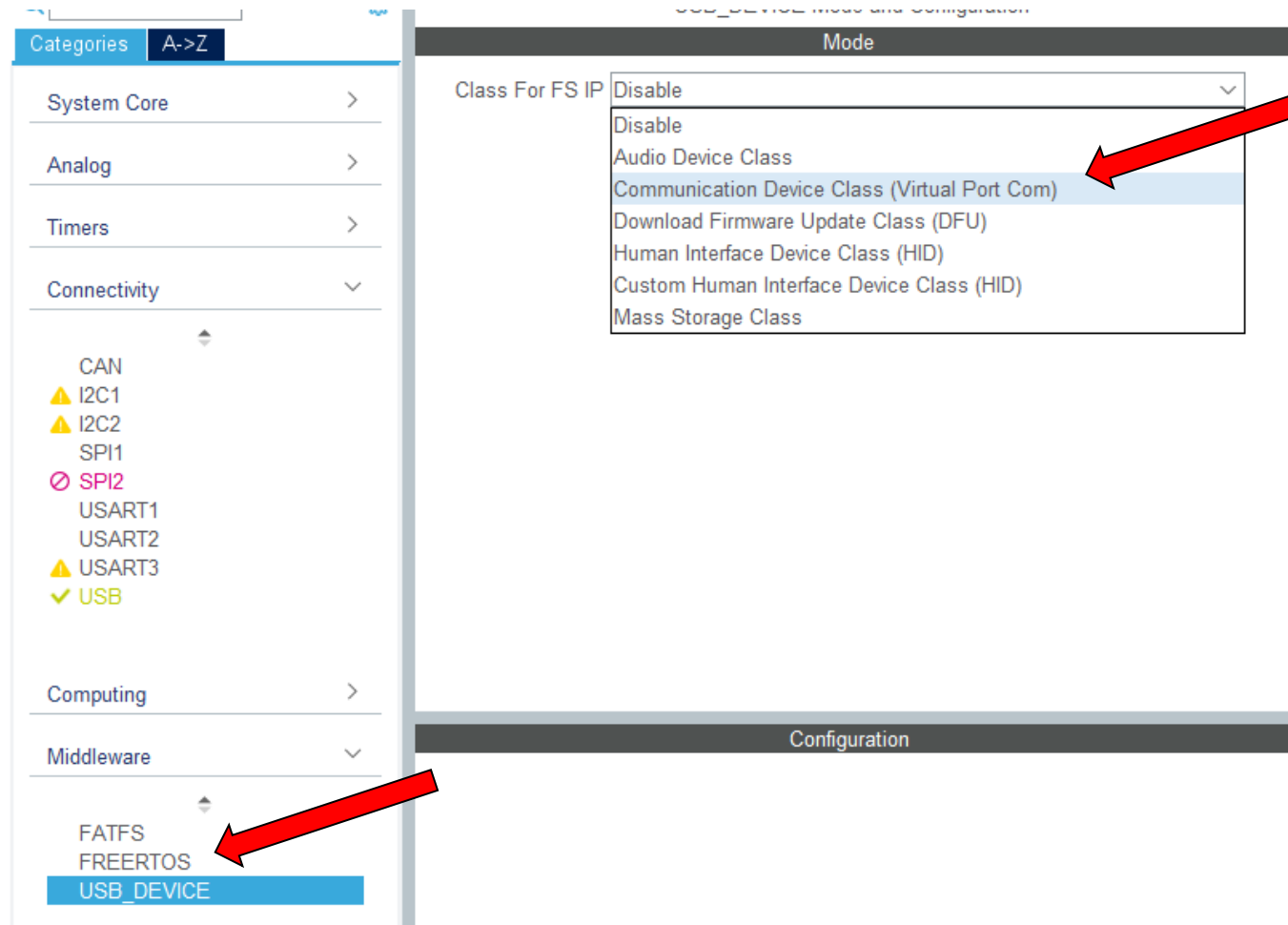
✓ Parameter Settings ✓ User Constants ✓ NVIC Settings ✓ GPIO Settings

Configure the below parameters :

Search (Ctrl+F) ⓘ ⓘ

- Basic Parameters
 - Speed Full Speed 12MBit/s
- Power Parameters
 - Low Power Disabled
 - Link Power Management Disabled
 - Battery Charging Disabled

Подключаем STM драйвера USB



Настройки можно оставить по-умолчанию

The image shows the Altium Designer configuration interface. On the left, a tree view lists various components: CAN, I2C1, I2C2, SPI1, SPI2 (disabled), USART1, USART2, USART3, and USB (checked). Below this, the 'Computing' and 'Middleware' sections are expanded, showing 'FATFS', 'FREERTOS', and 'USB_DEVICE' (checked). On the right, the 'Configuration' window is open, showing the 'Parameter Settings' tab. The 'Basic Parameters' section lists: USB_MAX_NUM_INTERFACE... 1, USB_MAX_NUM_CONFIGURA... 1, USB_MAX_STR_DESC_SIZ (M... 512 bytes, USB_SELF_POWERED (Enabl... Enabled, and USB_DEBUG_LEVEL (USBD ... 0: No debug message. The 'Class Parameters' section lists: USB CDC Rx Buffer Size 1000 Bytes and USB CDC Tx Buffer Size 1000 Bytes.

Configuration

Reset Configuration

Parameter Settings Device Descriptor User Constants

Configure the below parameters :

Search (Ctrl+F)

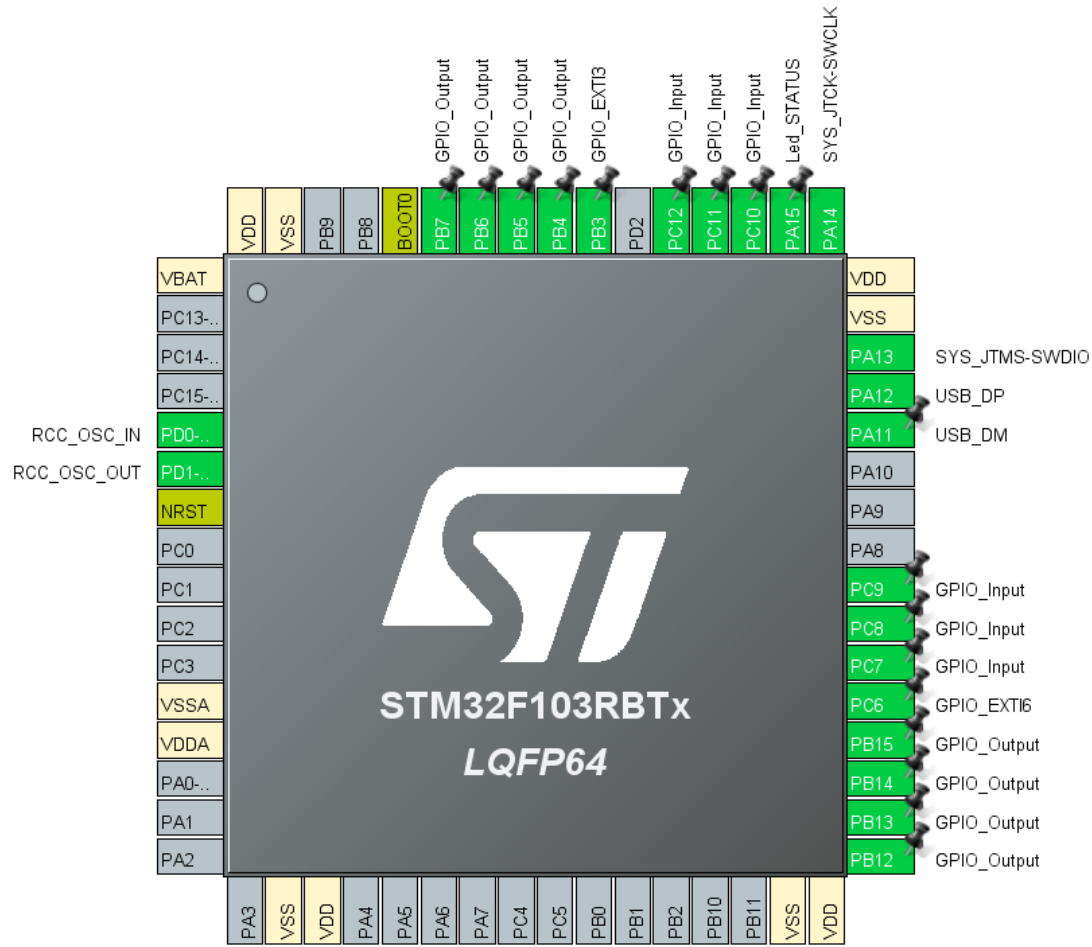
Basic Parameters

- USBD_MAX_NUM_INTERFACE... 1
- USBD_MAX_NUM_CONFIGURA... 1
- USBD_MAX_STR_DESC_SIZ (M... 512 bytes
- USBD_SELF_POWERED (Enabl... Enabled
- USBD_DEBUG_LEVEL (USBD ... 0: No debug message

Class Parameters

- USB CDC Rx Buffer Size 1000 Bytes
- USB CDC Tx Buffer Size 1000 Bytes

Проверяем, что настроены прерывания по двум кнопкам и таймеру



Pinout & Configuration

categories A-Z

- System Core >
- Analog >
- Timers >
 - RTC
 - TIM1
 - TIM2**
 - TIM3
 - TIM4
- Connectivity >
- Computing >
- Middleware >

Clock Configuration

Additional Software

TIM2 Mode and Configuration

Mode	
Slave Mode	Disable
Trigger Source	Disable
Clock Source	Internal Clock
Channel1	Disable
Channel2	Disable
Channel3	Disable
Channel4	Disable
Combined Channels	Disable
<input type="checkbox"/> Use ETR as Clearing Source	
<input type="checkbox"/> XOR activation	
<input type="checkbox"/> One Pulse Mode	

Configuration

Reset Configuration

Parameter Settings User Constants NVIC Settings DMA Settings

Configure the below parameters :

Search (Ctrl+F)

Counter Settings

Prescaler (PSC - 16 bits value)

0

Counter Mode

Up

Counter Period (AutoReload Register - 16 bits val...)

24000

Internal Clock Division (CKD)

No Division

auto-reload preload

Disable

Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit)

Disable (Trigger input effect not delayed)

Trigger Event Selection

Reset (UG bit from TIMx_EGR)

Перегенирируем исходный код в CubeMX и убираем все лишнее

Файл `stm32f1xx_it` можно удалить целиком. После чего CubeMX сгенерирует этот файл с нуля.

В этом уроке будем все описывать в файле `main`

Разрабатываемая нами прошивка должна работать следующим образом:

- 1) Доступно две кнопки для переключения режима отображаемого
- 2) По приходу сообщения по USB декодируем его и обновляем состояние светодиодов
- 3) В основном цикле плавно зажигаем и гасим светодиод

Для этого мы будем использовать прерывания по таймеру, двум кнопкам и usb.

Напишем в мейн код для плавного управления светодиодом (он будет служить индикатором того, что устройство не зависло)

```
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */
    //CDC_Transmit_FS("hello", 5);

    for ( int i = 0; i < 100; i++)
    {
        HAL_Delay(i/5);
        HAL_GPIO_WritePin(Led_STATUS_GPIO_Port, Led_STATUS_Pin, 1);
        HAL_Delay((100 - i)/5);
        HAL_GPIO_WritePin(Led_STATUS_GPIO_Port, Led_STATUS_Pin, 0);
    }

    for ( int i = 99; i >= 0; i--)
    {
        HAL_Delay(i/5);
        HAL_GPIO_WritePin(Led_STATUS_GPIO_Port, Led_STATUS_Pin, 1);
        HAL_Delay((100 - i)/5);
        HAL_GPIO_WritePin(Led_STATUS_GPIO_Port, Led_STATUS_Pin, 0);
    }
    /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}
```

Верхняя закомментированная строка `CDC_Transmit_FS("hello", 5);` может быть использована для передачи сообщения на компьютер. Для того чтобы в этом убедиться можно воспользоваться программой COM Port Toolkit или любым другим монитором COM порта (даже на андроид).

Первый цикл постепенно гасит светодиод (т.к. уменьшается задержка между включенным и выключенным состояниями). А второй цикл – постепенно зажигает светодиод. Принцип ШИМ.

Пишем обработчик события на таймер

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if (time_after_btn < 51)
        time_after_btn++;

    if (time_after_btn2 < 51)
        time_after_btn2++;
}
```

В функции мы инкрементируем переменные, показывающие сколько времени прошло с предыдущего нажатия кнопки (в обработчике кнопок эти переменные сбрасываем в ноль). Если прошло более 50 мс, то считаем, что этого достаточно для избежаниядребезга.

Таймер будем использовать для устранениядребезга контактов кнопок. Как и в прошлой работе, будем считать время, прошедшее с предыдущего нажатия кнопки.

Так как описывать это мы будем в main, а не в _it файле, то необходимо найти интерфейс функции, которую нам необходимо переопределить. Для этого в файле _it находим интересующую нас функцию «TIM2_IRQHandler» и смотрим что делает единственная функция, вызываемая в ней «HAL_TIM_IRQHandler(&htim2);». В функции находим интересующую нас область «/* TIM Update event */» которая отвечает за прерывание по обновлению значения в таймере. В этой области вызывается функция «HAL_TIM_PeriodElapsedCallback(htim);». Эту функцию нам и необходимо переопределить у себя в main.

Пишем обработчик нажатия кнопок

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == GPIO_PIN_6)
    {
        if (time_after_btn > 50){
            mode_load++;
            if (mode_load >= 4) mode_load = 0;

            time_after_btn = 0;

            update_led();
        }
    }
    else if (GPIO_Pin == GPIO_PIN_3) {
        mode_load--;
        if (mode_load < 0) mode_load = 3;

        time_after_btn2 = 0;

        update_led();
    }
}
```

Также будем описывать в main, а не в _it файле. Смотрим _it файл и обнаруживаем в нем две функции по работе с нашими двумя кнопками. Эти функции называются:

```
void EXTI9_5_IRQHandler(void)
void EXTI3_IRQHandler(void)
```

Обе эти функции вызывают одну и ту же функцию HAL_GPIO_EXTI_IRQHandler но в качестве параметра передают в нее номер пина, к которому подключена кнопка. Смотрим содержимое этой функции и находим функцию, которую нам необходимо переопределить в main

```
HAL_GPIO_EXTI_Callback(GPIO_Pin);
```

Эта функция переключает режим отображения информации на светодиоды (переменная mode_load), очищает переменную счетчик времени с предыдущего нажатия кнопки и обновляет состояние светодиодов

Функция для обновления состояния светодиодов Update_led

```
void update_led(void)
{
    int disp_val = 0;
    int tmp_var = 0;

    switch (mode_load){
        case 0: { disp_val = CPU_load; break;}
        case 1: { disp_val = RAM_load; break;}
        case 2: { disp_val = NET_load; break;}

        case 3:
        {
            tmp_var |= (SOUND_right >= 2) << 3;
            tmp_var |= (SOUND_right >= 4) << 2;
            tmp_var |= (SOUND_right >= 6) << 1;
            tmp_var |= SOUND_right >= 7;

            tmp_var |= (SOUND_left >= 2) << 7;
            tmp_var |= (SOUND_left >= 4) << 6;
            tmp_var |= (SOUND_left >= 6) << 5;
            tmp_var |= (SOUND_left >= 7) << 4;

            break;
        }
    }
}
```

```
if ( (mode_load != 3) )
    for (int i = 0; i < (disp_val); i++)
        tmp_var = (tmp_var << 1) | 1;

    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, !((tmp_var & 1)));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, !((tmp_var >> 1) & 1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, !((tmp_var >> 2) & 1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, !((tmp_var >> 3) & 1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, !((tmp_var >> 4) & 1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, !((tmp_var >> 5) & 1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, !((tmp_var >> 6) & 1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, !((tmp_var >> 7) & 1));
}
```

Switch в зависимости от режима переключает переменную, которая будет выведена на светодиоды. Для нашего основного 3-го режима происходит перекодировка двух переменных в левый и правый столбцы светодиодов. Для всех остальных, кроме третьего режима, количество зажженных светодиодов определяется переданным числом.

Прерывание по приходу данных от USB

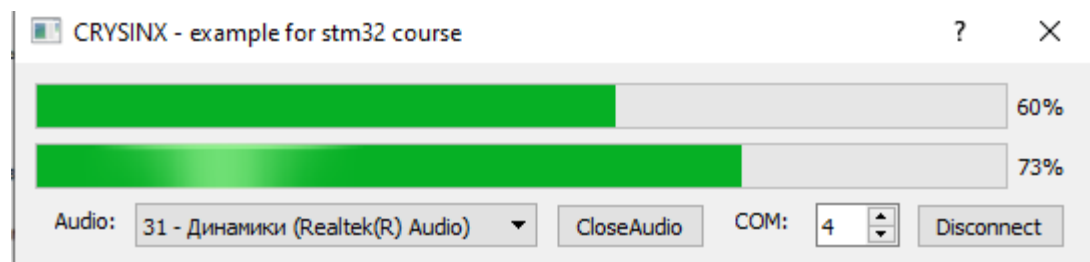
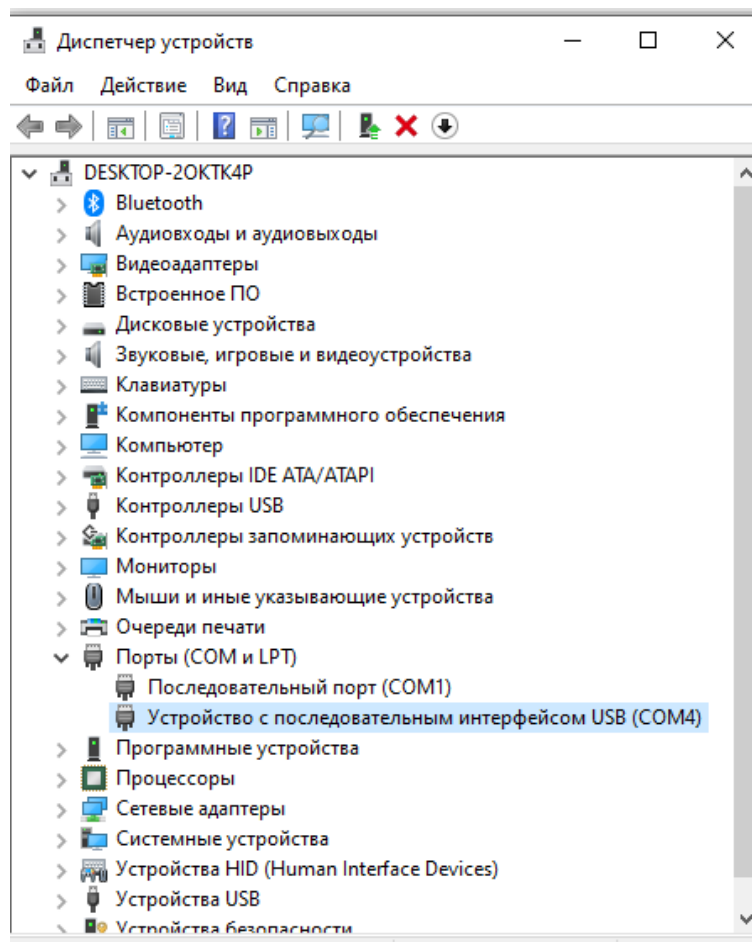
Протокол передачи данных с компьютера на плату следующий:

- 1) Первый символ «S». Показывает начало передачи
- 2) Второй символ – цифра от 0 до 4-х. Цифра устанавливает, значение какой переменной передается
- 3) Третий символ – значение переменной

В файле `usbd_cdc_if` находим функцию `static int8_t CDC_Receive_FS(uint8_t* Buf, uint32_t *Len)` и перед `return` добавляем вызов нашей функции `Resive_USB(Buf, Len);`

```
void Resive_USB(uint8_t* Buf, uint32_t *Len)
{
    for (int i = 0; i < *Len; i++)
    {
        if (Buf[i] == 'S') {
            switch (Buf[i+1]){
                case '0':{
                    CPU_load = Buf[i+2] - '0';
                    break;
                }
                case '1': {
                    RAM_load = Buf[i+2] - '0';
                    break;
                }
                case '2': {
                    NET_load = Buf[i+2] - '0';
                    break;
                }
                case '3': {
                    SOUND_left = Buf[i+2] - '0';
                    break;
                }
                case '4': {
                    SOUND_right = Buf[i+2] - '0';
                    break;
                }
            }
        }
    }
    update_led();
}
```

Компилируем\прошиваем

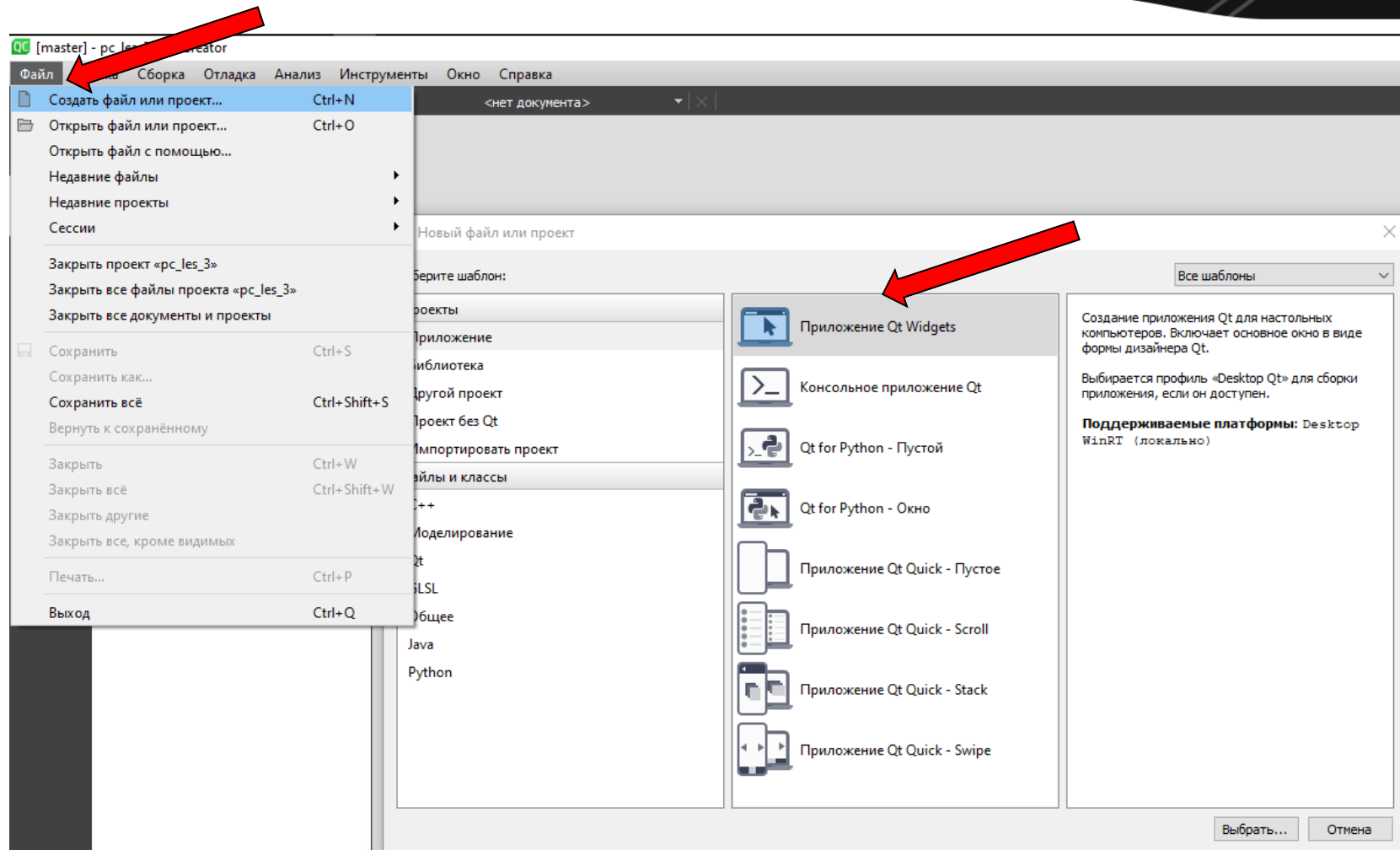


После подключения устройства к компьютеру смотрим в Диспетчере устройств номер присвоенного устройству COM порта. Открываем тестовую программу и настраиваем ее на соответствующий ком порт и выходное аудио устройство (колонки, наушники).

Включаем какую-нибудь музыку\видео и смотрим как моргают светодиоды на плате в соответствии с громкостью музыки на левой и правой колонках.

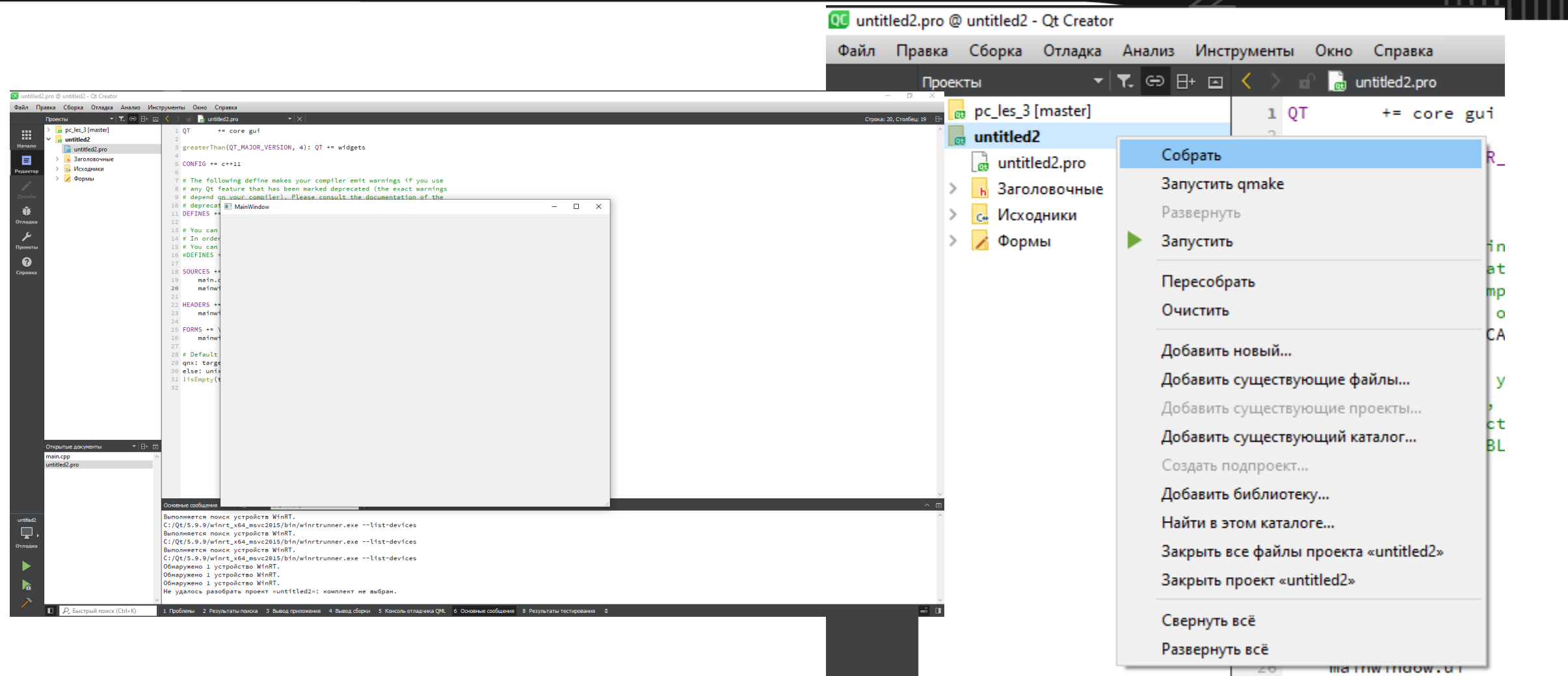
П.с. Кнопками на устройстве не забываем переключиться в необходимый режим.

Пишем программу для компьютера (будем использовать QT Creator)



Все настройки проекта оставляем по умолчанию

Собираем и компилируем пустой проект



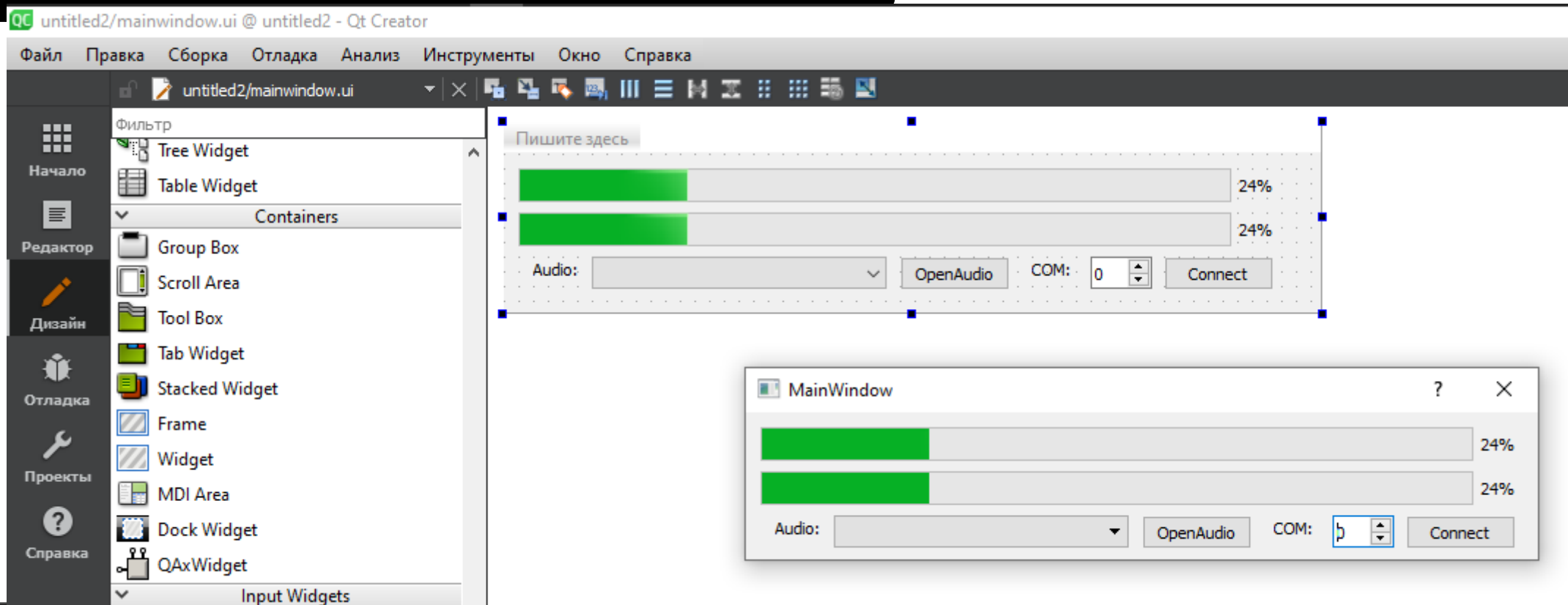
Редактируем форму приложения

The screenshot displays the Qt Creator application with the Qt Designer interface open. The main window is titled "untitled2/mainwindow.ui @ untitled2 - Qt Creator". The interface is divided into several panels:

- Left Panel (Project Explorer):** Shows the project structure. The "untitled2" project is expanded, showing files like "untitled2.pro", "Заголовочные", "Исходники", and "Формы". The "mainwindow.ui" file is selected under "Формы".
- Top Panel (Menu Bar):** Contains standard menus: "Файл", "Правка", "Сборка", "Отладка", "Анализ", and "Инструменты".
- Central Canvas:** A large area for designing the widget form. It contains a grid and a text label "Пишите здесь".
- Right Panel (Object Inspector):** Displays the hierarchy of objects in the form. The "MainWindow" object is selected, showing its class "QMainWindow" and its children: "centralwidget" (QWidget), "menubar" (QMenuBar), and "statusbar" (QStatusBar).
- Bottom Panel (Property Inspector):** Shows the properties of the selected "MainWindow" object. The "objectName" property is set to "MainWindow". Other properties like "windowModality", "enabled", "geometry", "sizePolicy", "palette", "font", "cursor", "mouseTracking", "tabletTracking", "focusPolicy", and "contextMenuPolicy" are listed with their values.

The status bar at the bottom indicates the current mode: "Редактор действий" (Action Editor) and "Редактор сигналов и слотов" (Signal and Slot Editor).

Добавляем на форму два прогресс бара, spinBox, ComboBox и две кнопки. Также добавляем необходимые подписи.




Для того чтобы зафиксировать размеры окна добавим в файл main.cpp две строки

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.setWindowFlags(Qt::Dialog);
    w.setFixedSize(540,100);
    w.show();
    return a.exec();
}
```



Разбираемся с COM

Подключим заголовочный файл
`#include <QtSerialPort/QtSerialPort>`

ВАЖНО! В файле *.pro необходимо
добавить для корректной сборки
проекта.

QT += serialport
CONFIG += serialport

В h файл добавляем указатель на экземпляр
класса COM порта в private область
MainWindow. Также добавляем интерфейсы
трех функций, которые мы далее напишем.

```
QSerialPort *serial;  
  
void initSerialPort();  
int connectSerialPort();  
void closeSerialPort();
```



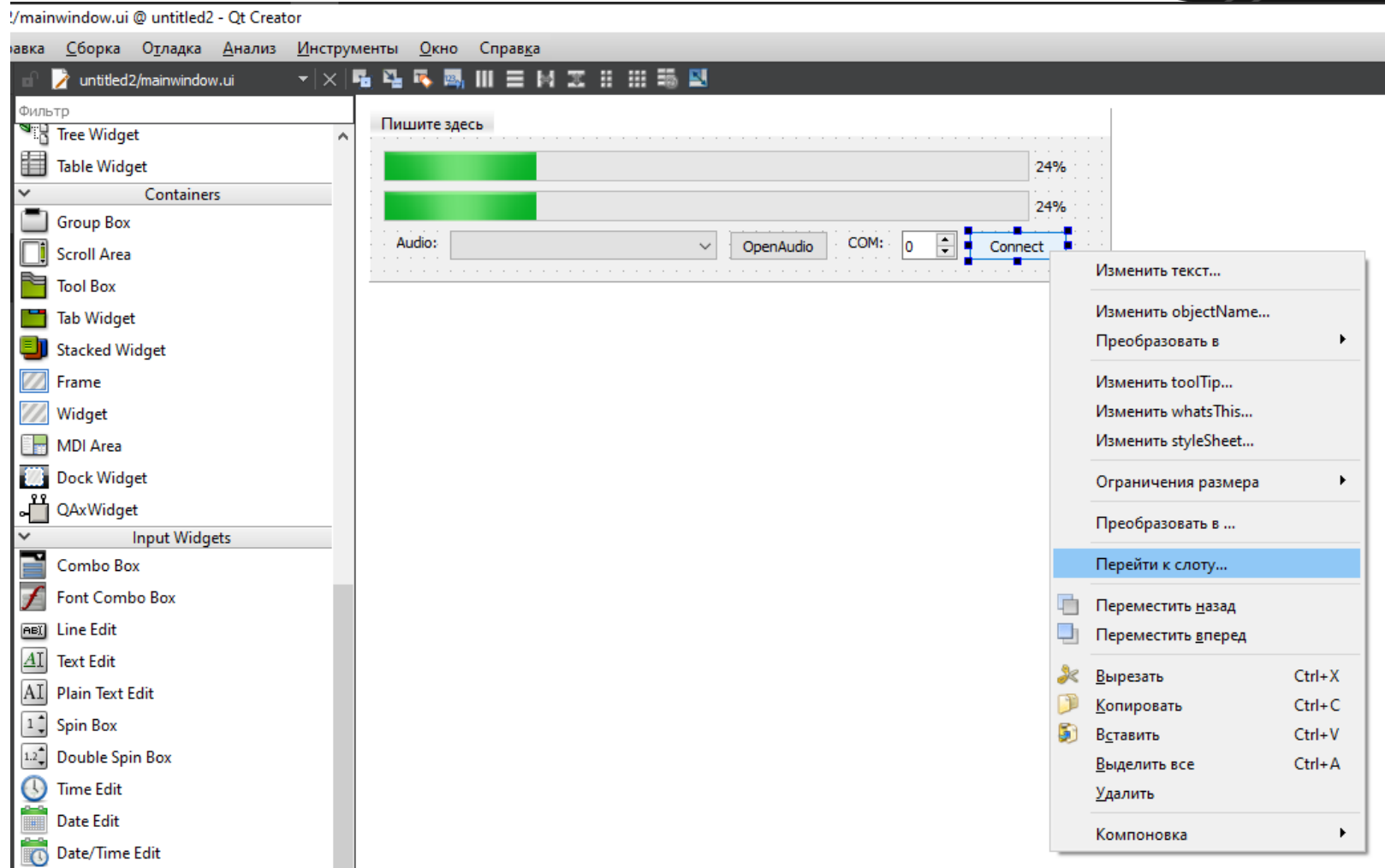
```
void MainWindow::initSerialPort(){
    serial = new QSerialPort(this);
}

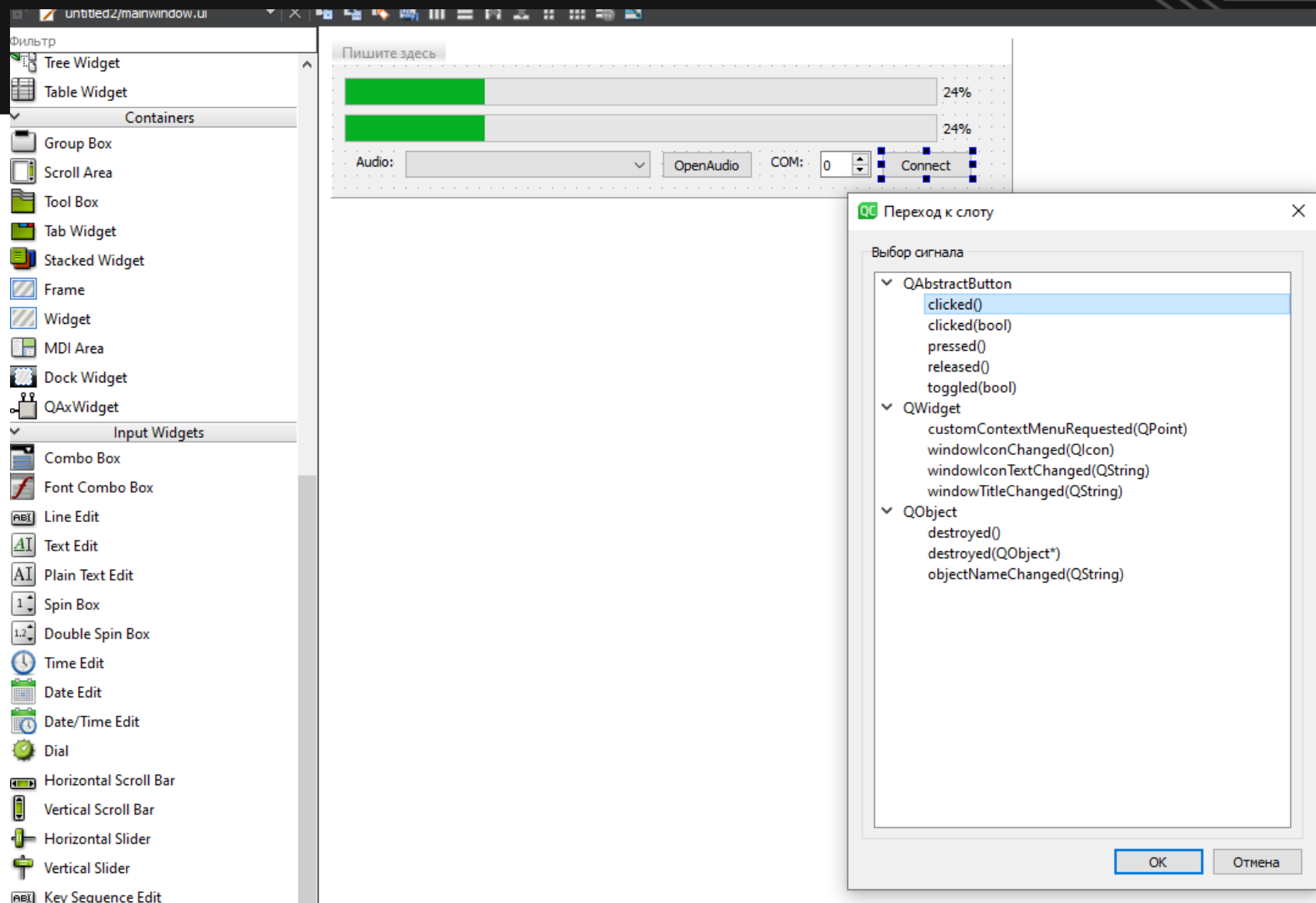
int MainWindow::connectSerialPort()
{
    serial->setPortName( QString::fromLocal8Bit("COM") + ui->spinBox->text());
    serial->setBaudRate(serial->Baud9600);
    serial->setDataBits( serial->Data8 );
    serial->setParity(serial->NoParity);
    serial->setStopBits(serial->OneStop);
    serial->setFlowControl( serial->NoFlowControl);

    return serial->open(QIODevice::ReadWrite);
}

void MainWindow::closeSerialPort()
{
    if (serial->isOpen())
        serial->close();
}
```

Добавляем обработчик события по кнопке





Код обработчика события по кнопке

Переменную `flag_connect` опишем в `private` области класса `MainWindow` (h файл).

Флаг показывает, наличие\отсутствие подключения к COM порту.

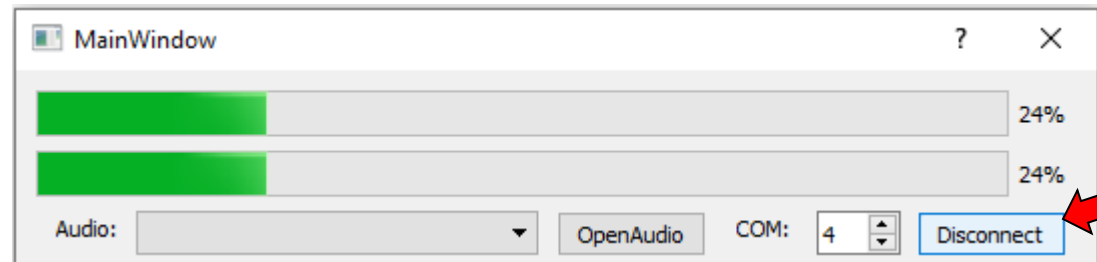
Проинициализируем COM порт в конструкторе `MainWindow`.

```
initSerialPort();
```

```
void MainWindow::on_pushButton_clicked()
{
    if (flag_connect == 0)
    {
        if (!connectSerialPort()) return;

        ui->pushButton->setText("Disconnect");
    } else {
        closeSerialPort();
        ui->pushButton->setText("Connect");
    }
    flag_connect = !flag_connect;
}
```

Скомпилируем и попробуем подключиться к плате



Отправка данных на плате будет производиться по таймеру. Добавим его в проект.

Подключим заголовочный файл
`#include <QTimer>`

В h файл добавляем указатель на экземпляр класса таймера в private область MainWindow.
`QTimer *timer;`

Добавляем обработчик срабатывания таймера в private slot секцию класса MainWindow.
`void slotTimerAlarm();`

В конструкторе MainWindow проинициализируем и запустим таймер с периодом 80 мс:

```
timer = new QTimer();  
  
connect(timer, SIGNAL(timeout()), t  
his, SLOT(slotTimerAlarm()));  
  
timer->start(80);
```

Опишем обработчик события по срабатыванию таймера

Переменные left и right будут содержать текущий уровень громкости левой и правой колонок. Сейчас для проверки запишем в них константы.

По таймеру будет обновляться состояние ProgressBar'ов которые показывают уровень громкости на компьютере.

Если подключение к плате активно, то в соответствии с описанным выше протоколом передаем на плату данные об уровне громкости.

```
void MainWindow::slotTimerAlarm()
{
    QByteArray send_data = "";

    unsigned int left = 20000; // the left level
    unsigned int right = 10000; // the right level

    ui->progressBar->setValue(left / (32768.0 / 100));
    ui->progressBar_2->setValue(right / (32768.0 / 100));

    if (flag_connect)
    {
        send_data.append("S");
        send_data.append("3");
        send_data.append(left / (32768.0 / 8) + '0');

        send_data.append("S");
        send_data.append("4");
        send_data.append(right / (32768.0 / 8) + '0');

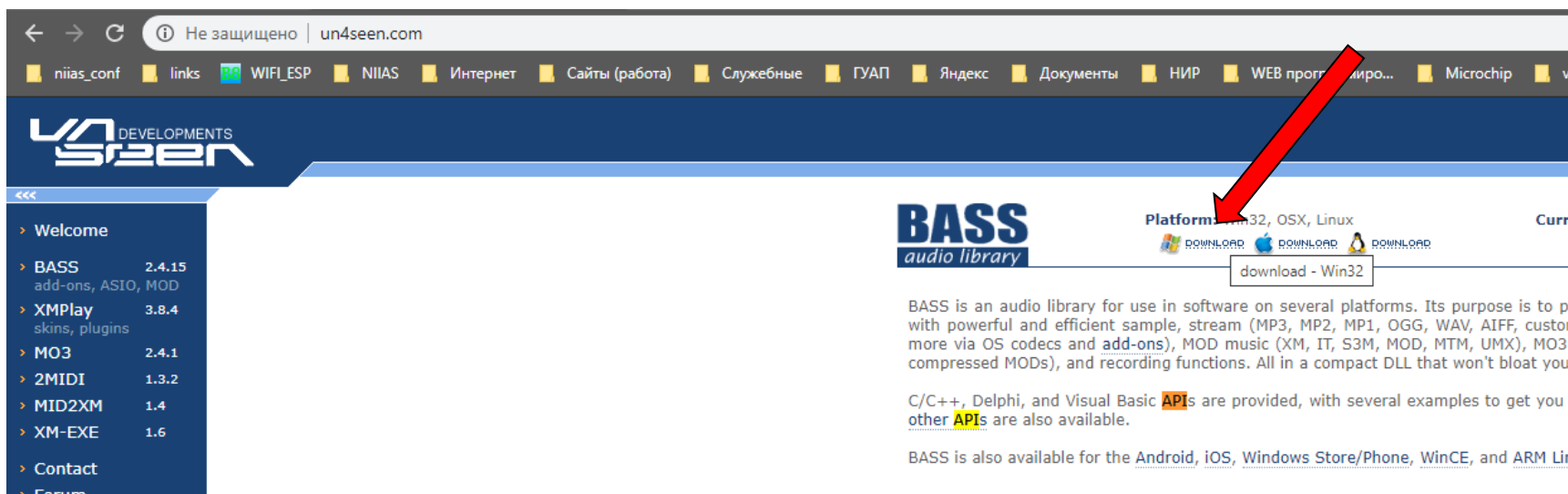
        serial->write( send_data );
    }
}
```

Работаем со звуком. Библиотека bass.dll

При работе с этой библиотекой я использовал следующий источник:

<http://www.un4seen.com/>

Для работы с библиотекой ее необходимо подключить к проекту. Для этого создадим в папке с проектом подкаталог lib. Туда поместим все необходимые файлы для работы с библиотекой.






















Подготавливаем библиотеку

Из загруженного архива распаковываем в папку lib файл bass.dll который лежит в корне архива. И к нему кладем заголовочный файл из папки "с" с именем base.h

Кроме этой основной библиотеки нам понадобится дополнение для нее. Дополнение называется: BASSWASAPI. Скачивается с той же страницы в разделе Add-ons.

По аналогии «вытаскиваем» из архива dll и h файлы и кладем их в каталог lib.

- 
-  [DOWNLOAD](#) (on OSX/iOS), or user-provided encoders. Also features streaming of encoded data to clients directly or via Shoutcast and Icecast servers, and PCM WAV/AIFF file writing. C/C++, Delphi, and Visual Basic APIs are included.
 -  [DOWNLOAD](#) **BASSenc_MP3** 2.4.1.1
 [DOWNLOAD](#) An extension to BASSenc that provides MP3 encoding of BASS channels, with support for LAME options. C/C++, Delphi, and Visual Basic APIs are included.
 -  [DOWNLOAD](#)
 -  [DOWNLOAD](#) **BASSenc_FLAC** 2.4.2.1
 [DOWNLOAD](#) An extension to BASSenc that provides FLAC encoding of BASS channels. C/C++, Delphi, and Visual Basic APIs are included.
 -  [DOWNLOAD](#)
 -  [DOWNLOAD](#) **BASSenc_OGG** 2.4.0.1
 [DOWNLOAD](#) An extension to BASSenc that provides Ogg Vorbis encoding of BASS channels, with support for OGGENC options. C/C++, Delphi, and Visual Basic APIs are included.
 -  [DOWNLOAD](#)
 -  [DOWNLOAD](#) **BASSenc_OPUS** 2.4.1.1
 [DOWNLOAD](#) An extension to BASSenc that provides Opus encoding of BASS channels, with support for OPUSENC options. C/C++, Delphi, and Visual Basic APIs are included.
 -  [DOWNLOAD](#)
 -  [DOWNLOAD](#) **BASSmix** 2.4.10
 [DOWNLOAD](#) An extension providing the ability to mix together multiple BASS channels, with resampling and matrix mixing features. Also provides the ability to split a BASS channel into multiple channels. C/C++, Delphi, and Visual Basic APIs are included.
 -  [DOWNLOAD](#)
 -  [DOWNLOAD](#) **BASSWASAPI** 2.4.2
An extension enabling the use of WASAPI input and output on Windows Vista and above, with support for exclusive and shared modes.
- [More add-ons](#)
-  [DOWNLOAD](#) **BASS FX** 2.4.12.1

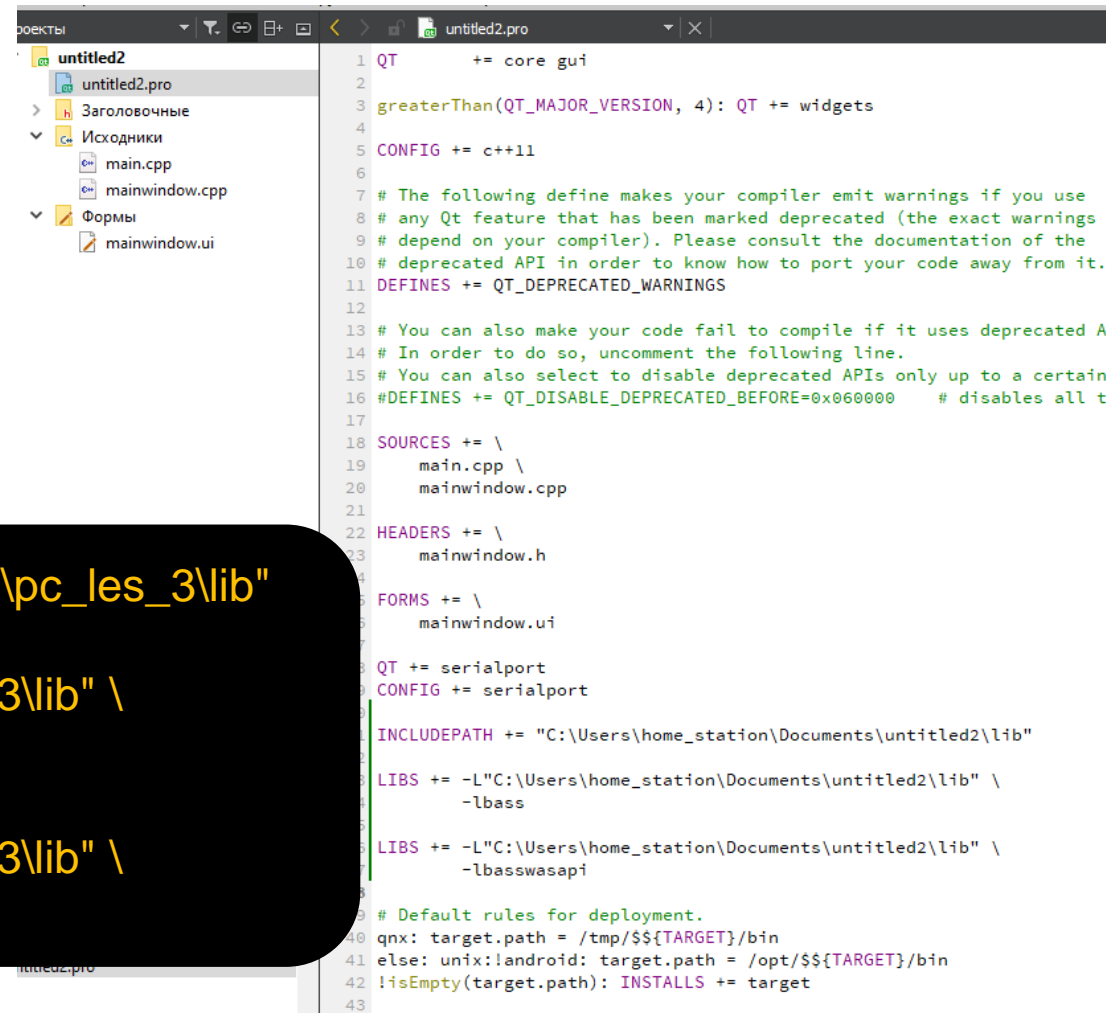
Настраиваем QT

В *.pro необходимо добавить указание на папку с заголовочными файлами и подключить сами библиотеки к проекту.

```
INCLUDEPATH += "D:\files\work\soft\windows\crysinox_les_3\pc_les_3\lib"
```

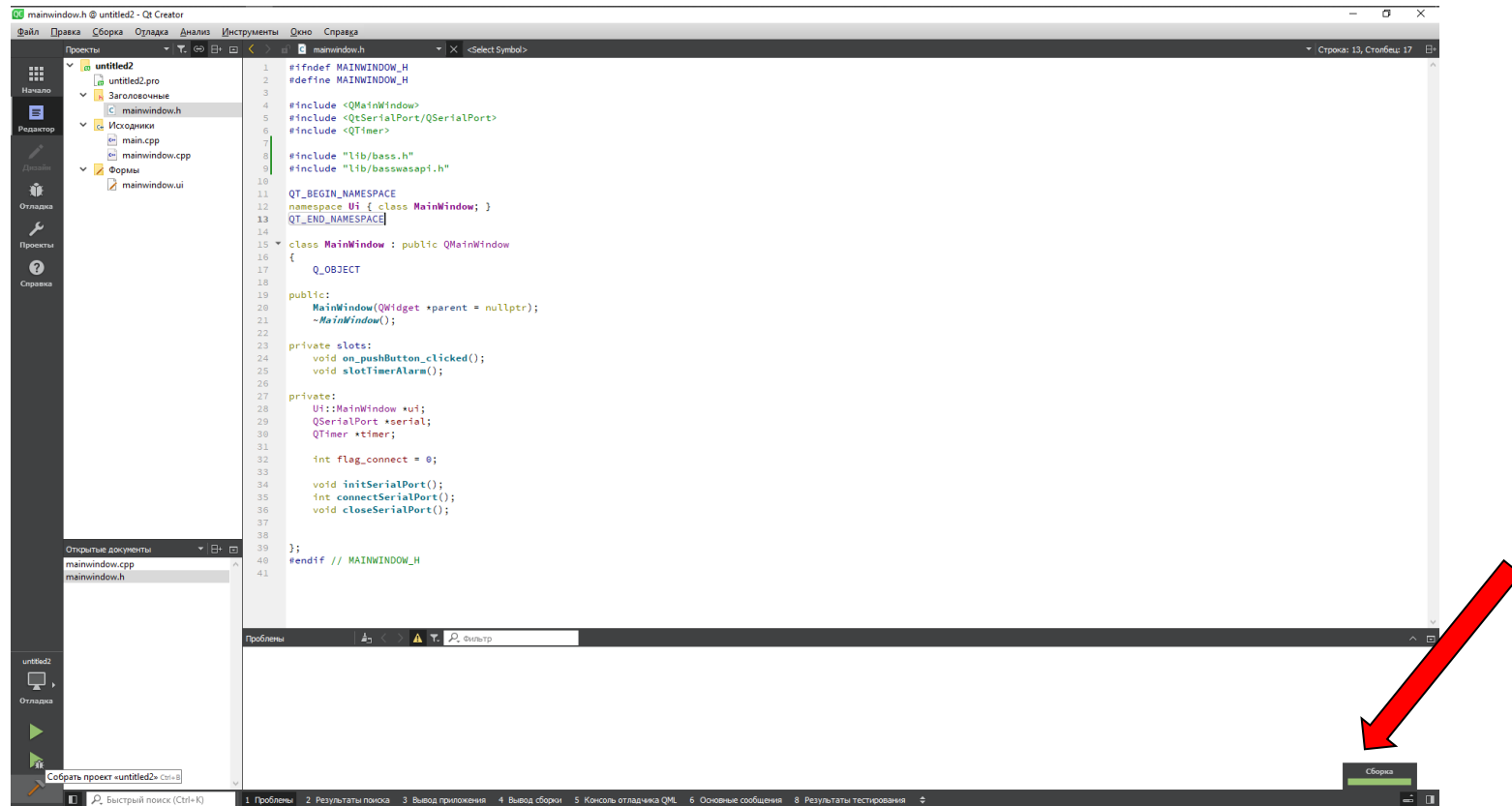
```
LIBS += -L"D:\files\work\soft\windows\crysinox_les_3\pc_les_3\lib" \  
-lbass
```

```
LIBS += -L"D:\files\work\soft\windows\crysinox_les_3\pc_les_3\lib" \  
-lbasswasapi
```



Прописываем заголовочные файл в нашем проекте и пробуем собрать приложение

```
#include "lib/bass.h"  
#include "lib/basswasapi.h"
```



Нам понадобится вывод диалоговых сообщений в случае ошибок, подключим соответствующее QT дополнение

```
#include <QMessageBox>
```

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtSerialPort/QSerialPort>
#include <QTimer>
#include <QMessageBox>


#include "lib/bass.h"
#include "lib/basswasapi.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
```



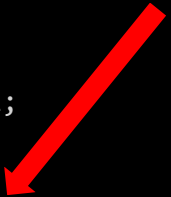
```
void on_pushButton_clicked();
void slotTimerAlarm();

private:
    Ui::MainWindow *ui;
    QSerialPort *serial;
    QTimer *timer;
    QMessageBox msgBox;

    int flag_connect = 0;

    void initSerialPort();
    int connectSerialPort();
    void closeSerialPort();

};
#endif // MAINWINDOW_H
```



Пробуем инициализировать библиотеку и воспроизвести любимую песню

Добавим данный код в конструктор MainWindow. При запуске приложения должна начать воспроизводиться мелодия, находящаяся по пути C://1.mp3.

```
DWORD stream;

BASS_SetConfig(BASS_CONFIG_UPDATETHREADS, 1);
int res = BASS_Init(-1, 44100, BASS_DEVICE_DEFAULT, 0, NULL);

if (!res){
    msgBox.setText("Init error");
    msgBox.exec();
    exit(0);
}

char filename[] = "C://1.mp3";
stream = BASS_StreamCreateFile(FALSE, filename, 0, 0, 0);

if (res == 0){
    msgBox.setText("Init error");
    msgBox.exec();
    exit(0);
}

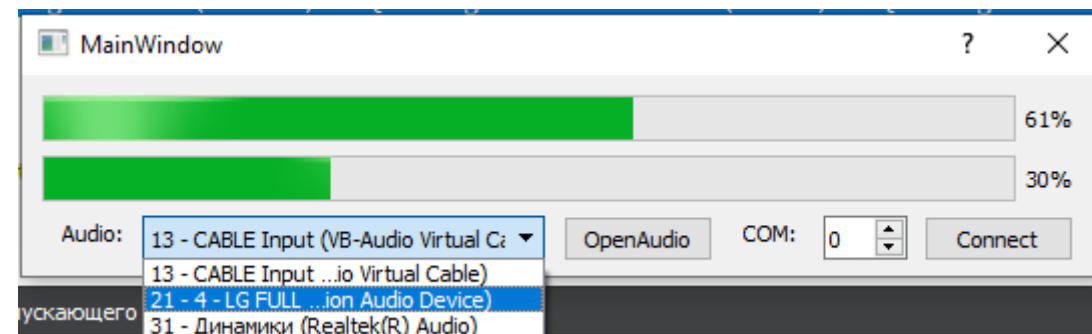
BASS_ChannelPlay(stream, TRUE);
```

При запуске приложения считаем доступные устройства для вывода звука

```
BASS_WASAPI_DEVICEINFO info_dev;
for (unsigned long device=0;BASS_WASAPI_GetDeviceInfo(device,&info_dev);device++)
{
    if ((info_dev.flags & BASS_DEVICE_ENABLED ) && (info_dev.flags & BASS_DEVICE_LOOPBACK ))
    {
        ui->comboBox->addItem(QString::number(device) + QString::fromLocal8Bit(" - ") + QString::fromLocal8Bit(info_dev.name));
    }
}
```

Важный момент, каждое звуковое устройство имеет индивидуальный номер. Нумерация производится начиная с нуля. В цикле мы перебираем все устройства. Цикл заканчивается когда невозможно считать информацию об очередном устройстве.

Далее для работы с устройством используется именно этот номер. Например у меня динамики имеют номер 31



Напишем обработчик события для кнопки OpenAudio

Для определения того, открыто какое-либо устройство или нет используется флаг `flag_open_audio`. Флаг описан аналогично флагу для ком порта.

```
void MainWindow::on_pushButton_2_clicked()
{
    if (!flag_open_audio)
    {
        QString dev_text = ui->comboBox->currentText();
        int id_dev = ((dev_text.split(' ')[0]).toInt());

        bool result = BASS_WASAPI_Init(id_dev, 0, 0, BASS_WASAPI_BUFFER, 1.0f, 0.05f, Process, NULL);

        if (!result){
            msgBox.setText("Init error");
            msgBox.exec();
            return;
        }

        flag_open_audio = !flag_open_audio;
        ui->pushButton_2->setText("CloseAudio");
        BASS_WASAPI_Start();
    } else {
        BASS_WASAPI_Stop(true);
        ui->pushButton_2->setText("OpenAudio");
        flag_open_audio = !flag_open_audio;
    }
}
```

В качестве параметра функция `BASS_WASAPI_Init` принимает указатель на функцию. Мы не будем использовать эту функцию, но так как она есть в списке параметров, то поставим заглушку.

```
// WASAPI callback, required for continuous recording
DWORD CALLBACK Process(void *buffer, DWORD length, void *user)
{
    return length;
}
```


При попытке открыть какое-нибудь аудио устройство мы получим ошибку

Дело в том, что ранее в `BASE_Init` мы уже открыли аудиоустройство и воспроизводим в него звук. Находим документацию на `Base_init`

Нас интересует первый параметр, который отвечает за номер открываемого устройства.

Parameters
device The device to use... -1 = default device,
0 = no sound, 1 = first real output device.
`BASS_GetDeviceInfo` can be used to enumerate the
available devices.

```
BOOL BASS_Init(  
    int device,  
    DWORD freq,  
    DWORD flags,  
    HWND win,  
    GUID *clsid  
);
```

Поменяем в `BASE_Init` первый параметр с -1 на 0 и закоментируем код для воспроизведения музыки.

Правим обработчик таймера для того, чтобы он захватывал громкость с колонок и выводил эту информацию в COM порт и на ProgressBar.

```
void MainWindow::slotTimerAlarm()
{
    QByteArray send_data = "";
    DWORD level = BASS_WASAPI_GetLevel();
    unsigned int left = (unsigned int) (level & 0xffff); // the left level
    unsigned int right =(unsigned int) (level >> 16); // the right level
    .....
}
```

Полный кодmainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtSerialPort/QtSerialPort>
#include <QTimer>
#include <QMessageBox>

#include "lib/bass.h"
#include "lib/basswasapi.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
```

```
private slots:
    void on_pushButton_clicked();
    void slotTimerAlarm();

    void on_pushButton_2_clicked();

private:
    Ui::MainWindow *ui;
    QSerialPort *serial;
    QTimer *timer;

    QMessageBox msgBox;

    int flag_connect = 0;
    int flag_open_audio = 0;

    void initSerialPort();
    int connectSerialPort();
    void closeSerialPort();

};
#endif // MAINWINDOW_H
```

Mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QtSerialPort/QSerialPort>
#include <minwindef.h>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    initSerialPort();

    DWORD stream;

    BASS_SetConfig(BASS_CONFIG_UPDATETHREADS, 1);
    int res = BASS_Init(0, 44100, BASS_DEVICE_DEFAULT, 0, NU
LL);

    if (!res){
        msgBox.setText("Init error");
        msgBox.exec();
        exit(0);
    }
```

```
/*
char filename[] = "C://1.mp3";
stream = BASS_StreamCreateFile(FALSE, filename, 0, 0, 0);

if (res == 0){
    msgBox.setText("Init error");
    msgBox.exec();
    exit(0);
}

BASS_ChannelPlay(stream,TRUE);*/

BASS_WASAPI_DEVICEINFO info_dev;
for (unsigned long device=0;BASS_WASAPI_GetDeviceInfo(device,&info_dev);device++)
{
    if ((info_dev.flags & BASS_DEVICE_ENABLED ) && (info_dev.flags & BASS_DEVICE_LOOPBACK ))
    {
        ui->comboBox->addItem(QString::number(device) + QString::fromLocal8Bit(" - ") + QString::fromLocal8Bit(info_dev.name));
    }
}

timer = new QTimer();
connect(timer, SIGNAL(timeout()), this, SLOT(slotTimerAlarm()));
timer->start(80);
}
```

Mainwindow.cpp продолжение

```
void MainWindow::initSerialPort(){
    serial = new QSerialPort(this);
}

int MainWindow::connectSerialPort()
{
    serial-
>setPortName( QString::fromLocal8Bit("COM") + ui->spinBox-
>text());
    serial->setBaudRate(serial->Baud9600);
    serial->setDataBits( serial->Data8 );
    serial->setParity(serial->NoParity);
    serial->setStopBits(serial->OneStop);
    serial->setFlowControl( serial->NoFlowControl);

    return serial->open(QIODevice::ReadWrite);
}

void MainWindow::closeSerialPort()
{
    if (serial->isOpen())
        serial->close();
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

```
void MainWindow::slotTimerAlarm()
{
    QByteArray send_data = "";
    DWORD level = BASS_WASAPI_GetLevel();
    unsigned int left = (unsigned int) (level & 0xffff); //
the left level
    unsigned int right =(unsigned int) (level >> 16); // th
e right level

    ui->progressBar->setValue(left / (32768.0 / 100));
    ui->progressBar_2->setValue(right / (32768.0 / 100));

    if (flag_connect)
    {
        send_data.append("S");
        send_data.append("3");
        send_data.append(left / (32768.0 / 8) + '0');

        send_data.append("S");
        send_data.append("4");
        send_data.append(right / (32768.0 / 8) + '0');

        serial->write( send_data );
    }
}
```

Mainwindow.cpp продолжение

```
void MainWindow::on_pushButton_clicked()
{
    if (flag_connect == 0)
    {
        if (!connectSerialPort()) return;

        ui->pushButton->setText("Disconnect");
    } else {
        closeSerialPort();
        ui->pushButton->setText("Connect");
    }
    flag_connect = !flag_connect;
}

// WASAPI callback, required for continuous recording
DWORD CALLBACK Process(void *buffer, DWORD length, void *user)
{
    return length;
}
```

```
void MainWindow::on_pushButton_2_clicked()
{
    if (!flag_open_audio)
    {
        QString dev_text = ui->comboBox->currentText();
        int id_dev = ((dev_text.split(' ')[0])).toInt();

        bool result = BASS_WASAPI_Init(id_dev, 0, 0, BASS_WASAPI_BUFFER, 1.0f, 0.05f, Process, NULL);

        if (!result){
            msgBox.setText("Init error");
            msgBox.exec();
            return;
        }

        flag_open_audio = !flag_open_audio;
        ui->pushButton_2->setText("CloseAudio");
        BASS_WASAPI_Start();
    } else {
        BASS_WASAPI_Stop(true);
        ui->pushButton_2->setText("OpenAudio");
        flag_open_audio = !flag_open_audio;
    }
}
```

Main.c для STM32

```
#include "main.h"
#include "usb_device.h"
TIM_HandleTypeDef htim2;

int CPU_load;
int RAM_load;
int NET_load;
int SOUND_left;
int SOUND_right;
int mode_load = 0;

int time_after_btn = 0,
    time_after_btn2 = 0;

int status_ind_cntr = 0;

void update_led(void);
void Reseive_USB(uint8_t* Buf, uint32_t *Len);

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
```

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim
)
{
    if (time_after_btn < 51)
        time_after_btn++;

    if (time_after_btn2 < 51)
        time_after_btn2++;
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == GPIO_PIN_6)
    {
        if (time_after_btn > 50){
            mode_load++;
            if (mode_load >= 4) mode_load = 0;

            time_after_btn = 0;

            update_led();
        }
    } else if (GPIO_Pin == GPIO_PIN_3) {
        mode_load--;
        if (mode_load < 0) mode_load = 3;

        time_after_btn2 = 0;

        update_led();
    }
}
```

Main.c для STM32 продолжение

```
void update_led(void)
{
    int disp_val = 0;
    int tmp_var = 0;

    switch (mode_load){
        case 0: { disp_val = CPU_load; break;}
        case 1: { disp_val = RAM_load; break;}
        case 2: { disp_val = NET_load; break;}

        case 3:
        {
            tmp_var |= (SOUND_right >= 2) << 3;
            tmp_var |= (SOUND_right >= 4) << 2;
            tmp_var |= (SOUND_right >= 6) << 1;
            tmp_var |= SOUND_right >= 7;

            tmp_var |= (SOUND_left >= 2) << 7;
            tmp_var |= (SOUND_left >= 4) << 6;
            tmp_var |= (SOUND_left >= 6) << 5;
            tmp_var |= (SOUND_left >= 7) << 4;

            break;
        }
    }
}
```

```
if ( (mode_load != 3) )
    for (int i = 0; i < (disp_val); i++)
        tmp_var = (tmp_var << 1) | 1;

    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, !((tmp_var & 1)))
;
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, !((tmp_var >> 1)
& 1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, !((tmp_var >> 2)
& 1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, !((tmp_var >> 3)
& 1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, !((tmp_var >> 4) &
1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, !((tmp_var >> 5) &
1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, !((tmp_var >> 6) &
1));
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, !((tmp_var >> 7) &
1));
}
```


Main.c для STM32 продолжение

```
void Reseive_USB(uint8_t* Buf, uint32_t *Len)
{
    for (int i = 0; i < *Len; i++)
    {
        if (Buf[i] == 'S') {
            switch (Buf[i+1]){
                case '0':{
                    CPU_load = Buf[i+2] - '0';
                    break;
                }

                case '1': {
                    RAM_load = Buf[i+2] - '0';
                    break;
                }

                case '2': {
                    NET_load = Buf[i+2] - '0';
                    break;
                }
            }
        }
    }
}
```

```
        case '3': {
            SOUND_left = Buf[i+2] - '0';
            break;
        }

        case '4': {
            SOUND_right = Buf[i+2] - '0';
            break;
        }
    }
}

update_led();
}
```

Main.c для STM32 продолжение

```
int main(void)
{
    HAL_Init();
    SystemClock_Config();

    MX_GPIO_Init();
    MX_USB_DEVICE_Init();
    MX_TIM2_Init();

    HAL_TIM_Base_Start_IT(&htim2);
    while (1)
    {
        for ( int i = 0; i < 100; i++)
        {
            HAL_Delay(i/5);
            HAL_GPIO_WritePin(Led_STATUS_GPIO_Port, Led_STATUS_Pin, 1);
            HAL_Delay((100 - i)/5);
            HAL_GPIO_WritePin(Led_STATUS_GPIO_Port, Led_STATUS_Pin, 0);
        }
    }
}
```

```
for ( int i = 99; i >= 0; i--)
{
    HAL_Delay(i/5);
    HAL_GPIO_WritePin(Led_STATUS_GPIO_Port, Led_STATUS_Pin, 1);
    HAL_Delay((100 - i)/5);
    HAL_GPIO_WritePin(Led_STATUS_GPIO_Port, Led_STATUS_Pin, 0);
}
}
```



STM &  XILINX.

**Спасибо за внимание,
спасибо за старания!**

GitHub

https://github.com/v-crys/course_stm32_fpga