3.2. Лабораторная работа. Wi-Fi в Mbed

Site: Samsung Innovation Campus
Course: Internet of Things
Book: 3.2. Лабораторная работа. Wi-Fi в Mbed

Printed by: Антон Файтельсон
Date: Saturday, 21 October 2023, 7:37 PM

Table of contents

3.2.1. Соединение ESP8266 и STM32Nucleo

AT-команды - это проверенный десятилетиями устоявшийся способ коммуникации. Пользуясь AT-командами, можно заставить модуль подключиться к сети и сделать любые другие команды. Но нам бы хотелось делать всё это не вручную, а программно, и с этой целью подключим нашу микроконтроллерную плату.

Здесь будет рассмотрено физическое подключение ESP8266. Соединение должно быть по четырем выводам: два коммуникационных вывода TX и RX, питание (V) и земля (GND). Со стороны микроконтроллерной платы выводы будут такие: RX и TX (крест-накрест). питание 3.3 В и земля.

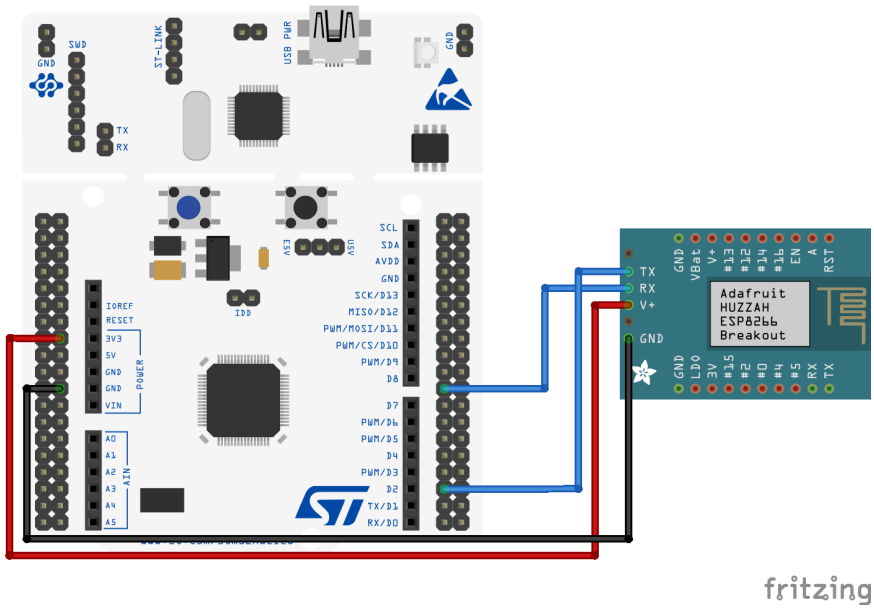
Если у вас Slot Shield, то к сожалению, подключить удобным образом плату к нему не получится, поскольку он позволяет использовать для коммуникации из UART-выводов только D0 и D1, а они у нас уже заняты для общения с компьютером. Поэтому подключим модуль просто проводами (или, как вариант, через Troika Shield - будет удобнее, если он у вас есть). Выберите, к каким из выводов UART вы хотите подключиться, главное, чтобы это было одно и то же устройство UART (то есть вывод TX и вывод RX должны принадлежать одному и тому же UART).

Всего у нас 4 UART на борту, из них UART2 мы задействовать не можем, поскольку он уже занят под коммуникацию с компьютером через USB. Поэтому можно выбрать UART1, UART4 или UART5.

Наглядно табличка свободных UART:

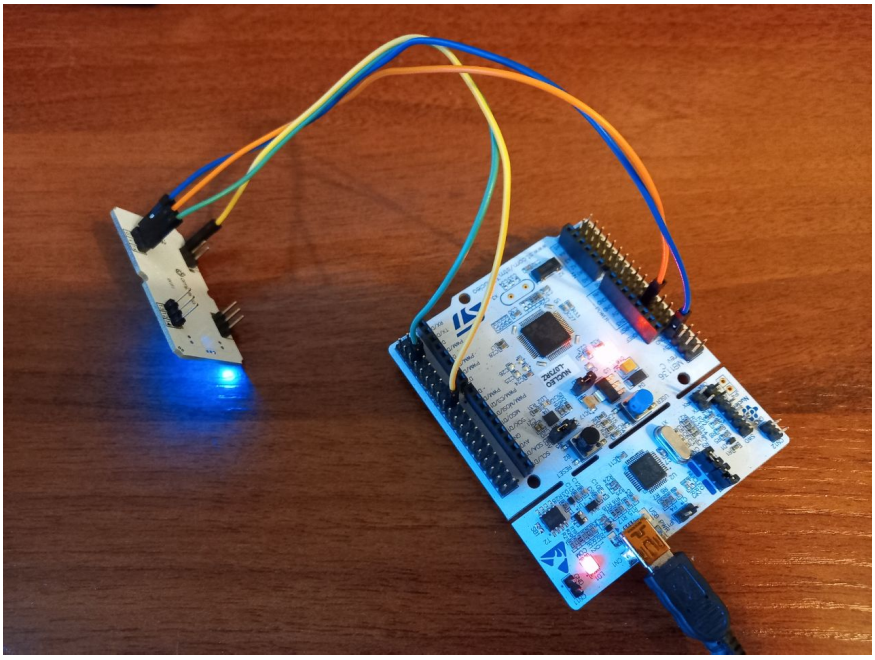
Номер UART	Вывод TX	Вывод RX
UART1	A4, D6, D8, D10	A5, D2
UART4	A0	A1
UART5	D3	D5

Мы будем использовать для примера UART1, а подключаться будем к выводам D8 и D2.



Здесь на рисунке другая WiFi-плата, но это не принципиально, нам важно, что подключаются те же самые 4 провода.

Выглядеть это будет так:



Для начала просто подключимся к плате, ничего не программируя. Протестируем соединение. STM32Nucleo будет выступать в роли “моста”. Подключаться будем к одному из UART, можно выбрать любой. Мы для определенности взяли UART1 (выводы D8 и D2), но это не принципиально. UART2 (D1 и D0) уже занят под коммуникацию с компьютером по USB, поэтому его использовать не станем.

Подключим четыре провода:

ESP8266	STM32Nucleo
V	3.3V
GND	GND
RX	D8
TX	D2

Кое-что насчет питания:

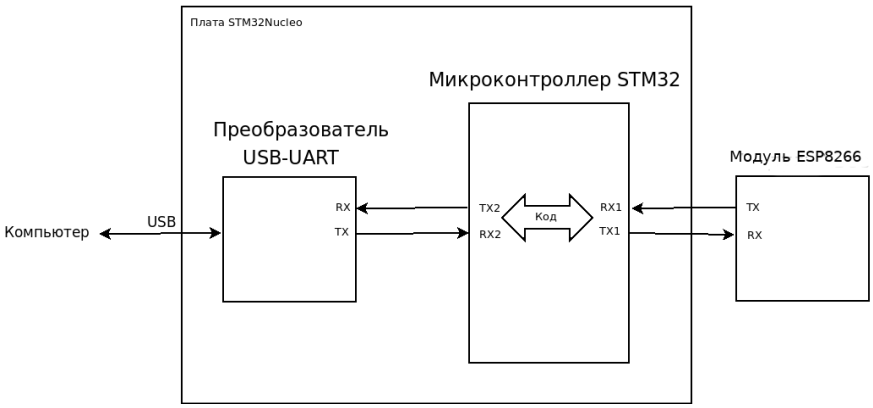
- Если на плате ESP8266 есть встроенный преобразователь 5В в 3.3В, то лучше воспользоваться им. Например, такой преобразователь есть на плате от “Амперки”. Причина, почему лучше воспользоваться им: на самой плате Nucleo встроенные преобразователь слабый, и если вы берете с нее 3.3В, то есть риск нестабильной работы. Итак, в этом случае, вы берете с платы Nucleo питание через вывод, маркированный 5V.
- Если на плате ESP8266 нет встроенного преобразователя, то брать 5V ни в коем случае нельзя! Тогда вы берете 3.3В, как указано в таблице выше.

Воспользуемся готовым кодом.Создайте пустой проект (назовите его, например, serial-passthrough), и скопируйте туда следующий код:

```
#include "mbed.h"

RawSerial  pc(USBTX, USBRX);
RawSerial  dev(D8, D2);
DigitalOut led1(LED1);

void dev_recv()
{
    led1 = !led1;
    while(dev.readable()) {
        pc.putc(dev.getc());
    }
}

void pc_recv()
{
    while(pc.readable()) {
        dev.putc(pc.getc());
    }
}

int main()
{
    pc.baud(115200);
    dev.baud(115200);

    pc.attach(&pc_recv, Serial::RxIrq);
    dev.attach(&dev_recv, Serial::RxIrq);

    while(1) {
        sleep();
    }
}
```

! В Mbed 6 RawSerial заменен на UnbufferedSerial.

Как вы видите, код очень простой, он перенаправляет данные от нашего устройства на D8 и D2 на компьютер через D1 и D0, создавая прозрачный “мост”.

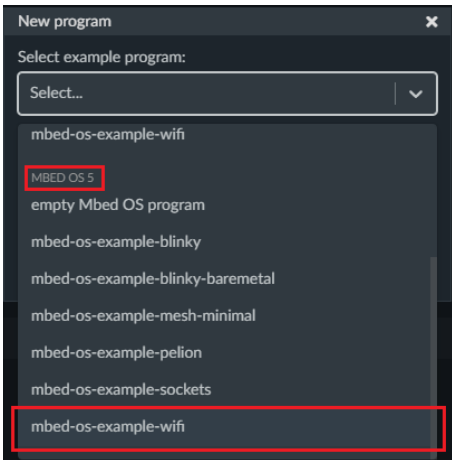


Теперь вы можете общаться с модулем через терминал последовательного порта. В Mbed Studio откройте терминал на скорости 115200 и попробуйте пообщаться посредством AT-команд. То есть введите те же самые команды AT и AT+GMR, и посмотрите на выдачу. Если всё в порядке, то можно двигаться к следующему пункту!

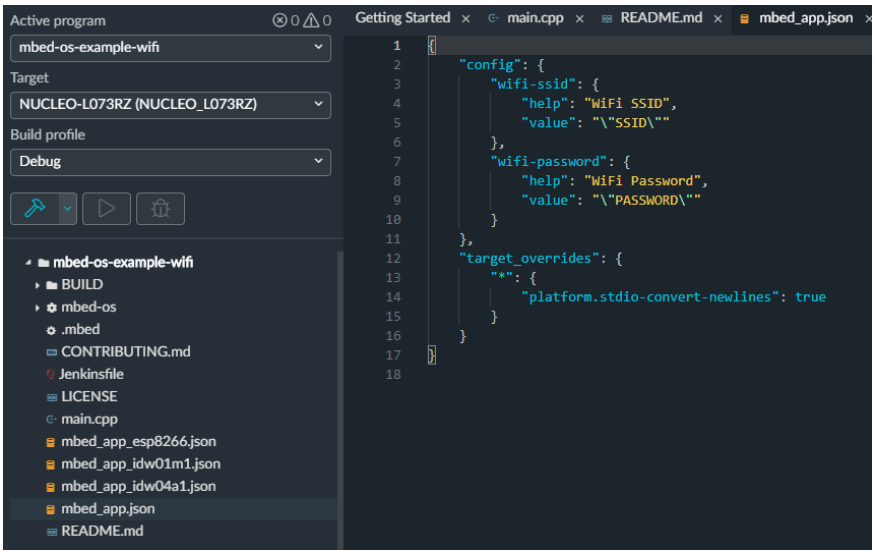
3.2.2. Изучение примера WiFi

Прежде, чем что-то программировать, проверьте, что плата STM32Nucleo коммуницирует с ESP8266, и что вы всё правильно подсоединили (см. предыдущий шаг). Если получается в терминале отправлять AT-команды и получать на них ответы, то всё OK.

Откройте в Mbed Studio пример для Wifi, либо посмотрите его же в онлайн-компиляторе. Он называется mbed-os-example-wifi-5. Как найти его в Mbed Studio: File - New Program - Select Example Program. Важно, что вам могут предложить пример для версии Mbed 6.0, но в настоящий момент (июнь 2020) он неработоспособен, выдавая ошибку при той же конфигурации. Поэтому брать нужно пример для Mbed 5.0, до тех пор, пока пока баг не исправят (соответствующий Issue уже открыт в GitHub разработчиков). Данный пример был успешно протестирован для mbed с библиотеками версии 5.12.0. Возможно, с выходом новых версий ситуация поменяется.



Вы увидите простой пример. В директории проекта откройте файл mbed_app.json



Если посмотрите на структуру проекта, то увидите, что в нем фигурирует набор json-файлов:

- mbed_app.json
- mbed_app_esp8266.json
- mbed_app_idw01m1.json
- mbed_app_idw04a1.json

Первый файл - конфигурационный, в нем задаются параметры подключения, такие как имя и пароль сети, интерфейс, свойства этого интерфейса (к примеру, выводы для коммуникации). Другие файлы - это примеры настроек для различных WiFi- интерфейсов. Если вы используете какой-то из этих интерфейсов, то копируете настройки в mbed_app.json - именно к нему будет обращаться программа.

Итак, работаем с mbed_app.json.

Введите там параметры своей сети: название и пароль (в секции config).

В target_overrides необходимо указать тип интерфейса связи. По умолчанию это Ethernet, также можно указать WiFi, Cellular или Mesh (об этом есть в руководстве). Мы укажем WiFi.

```
"target.network-default-interface-type": "WIFI",
```

Также в target_overrides добавьте строчку, которая фигурирует в файле mbed_app_esp8266.json:

```
"esp8266.provide-default" : true
```

Тогда программа будет знать, какой WiFi-интерфейс ей подключать: вы увидите в main.cpp, что там создается экземпляр общего базового класса WiFiInterface. В исходном коде явным образом не прописано, он достаточно абстрактный.

Кроме того, туда нужно добавить строчки, указывающие, к каким выводам подключена плата ESP8266:

```
"esp8266.tx": "D8",
"esp8266.rx": "D2"
```

Какие еще бывают конфигурационные параметры, можно посмотреть в драйвере ESP8266 (путь иногда меняется в новых версиях mbed, бывает, что файл перекладывают в другую директорию - в этом случае ищите папку esp8266-driver - это делается простым поиском по репозиторию GitHub).

То есть в целом конфигурационный файл mbed_app.json будет выглядеть примерно так:

```
{
    "config": {
        "wifi-ssid": {
            "help": "WiFi SSID",
            "value": "\"myssid\""
        },
        "wifi-password": {
            "help": "WiFi Password",
            "value": "\"mypassword\""
        },
        "tx": {
            "help": "TX pin for serial connection",
            "value": "D8"
        },
        "rx": {
            "help": "RX pin for serial connection",
            "value": "D2"
        },
        "socket-bufsize": {
            "help": "Max socket data heap usage",
            "value": "1024"
        }
    },
    "target_overrides": {
        "*": {
            "target.network-default-interface-type": "WIFI",
            "esp8266.tx": "D8",
            "esp8266.rx": "D2",
            "esp8266.socket-bufsize": "1024",
            "esp8266.debug": false,
            "platform.stdio-convert-newlines": true,
            "esp8266.provide-default" : true
        }
    }
}
```

По непонятной причине требуется указывать некоторые параметры дважды - иначе программа рискует не увидеть доступные сети или вывалиться с ошибкой. Предлагаемый же выше конфигурационный файл - рабочий и проверенный.

Скомпилируйте пример и загрузите в плату. Имейте в виду, что в этом примере скорость в терминале должна быть выставлена на 9600, в противном случае вы ничего не увидите. Либо, альтернативный вариант: добавление строчек:

```
"platform.stdio-baud-rate": 115200,
"platform.default-serial-baud-rate": 115200
```

в mbed_app.json в target_overrides позволит общаться на "привычной" 115200

Если вы забыли указать в конфигурационном файле, какой именно WiFi интерфейс использовать по умолчанию, то увидите следующую ошибку:

```
WiFi example
Mbed OS version 5.12.0

ERROR: No WiFiInterface found.
```

Если вы увидите такую ошибку, то у вас скорее всего неправильно подключена плата ESP8266 или вы неправильно указали выводы в конфигурационном файле:

```
Scan:
AT
AT
AT
AT
AT
scan() failed with return value: -3012
No WIFI APs found - can't continue further.
```

3.2.3. Запуск примера с WiFi

Далее смотрите пример на скорости 9600 бод (или 115200, если вы это явно указали в файле конфигурации). Увидите примерно следующее: вначале появится надпись Scan, она будет висеть довольно долго, пока идет сканирование доступных сетей. Затем появится их список. И наконец, вы увидите процесс подключения.

```
WiFi example
Mbed OS version 5.12.0

Scan:
Network: MGTS_GPON_6736 secured: WPA2 BSSID: D4:60:E3:21:94:7a RSSI: -94 Ch: 1
Network: Valentina secured: WPA2 BSSID: F8:C0:91:14:e6:a7 RSSI: -81 Ch: 11
Network: OnLime_10 secured: WPA/WPA2 BSSID: 7C:39:53:8f:a4:6 RSSI: -30 Ch: 11
3 networks available.

Connecting to OnLime_10...

Connection error: -3015
```

Здесь непонятная ошибка с кодом -3015, чтобы узнать, что означают все эти ошибки, просто посмотрите в таблице кодов. В данном случае “ошибка” заключается в том, что соединение уже установлено.

Поэтому имеет смысл слегка модифицировать main так, чтобы ошибка корректно обрабатывалась и программа не вылетала в случае, если сеть уже подключена:

```
    if (ret == -3015) {
        printf("Already connected!\n\r");
    }
    else if (ret != 0) {
        printf("\nConnection error: %d\n", ret);
        return -1;
    }
```

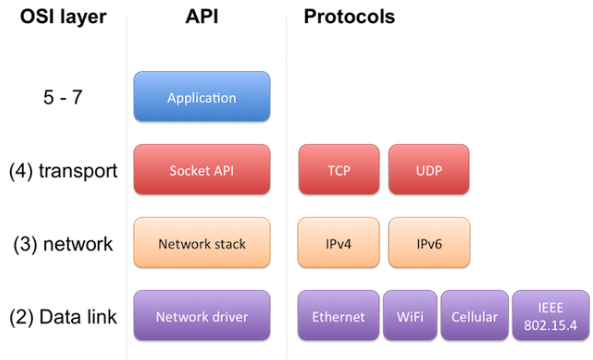
Наконец, по итогам всего увидите долгожданную надпись.

```
Success

MAC: 2c:f4:32:81:6a:00
IP: 192.168.0.48
Netmask: 255.255.255.0
Gateway: 192.168.0.1
RSSI: -37
```

3.2.4. Как устроена работа с сетями в Mbed?

После того, как заработал самый простой пример с WiFi, необходимо сделать небольшое отступление. Как вообще абстрагируется работа с сетями в Mbed? На нижеприведенной картинке исчерпывающе показана вся работа с сетевым стеком.

Мы сейчас посмотрели работу нижнего уровня - на уровне 2 (Data Link) мы подключились к точке доступа по WiFi. Дальше мы посмотрим, как устроен транспортный уровень на примере Socket API, но это будет следующий шаг



Обратим внимание вот на что. Есть 4 разных протокола канального уровня - это Ethernet, WiFi, Cellular, и Mesh (или, как на этой картинке он называется, IEEE 802.15.4). А API у них одинаковый! То есть теоретически, при смене протокола программа не должна вообще поменяться. Меняется только конфигурационный файл с параметрами подключения, а сам код должен быть достаточно абстрактно устроен, чтобы его не пришлось переписывать каждый раз при смене Ethernet на WiFi, к примеру.

Как это достигается?

В Mbed есть общий класс под названием Network Interface. Обращение к этому классу выглядит примерно так:

```
NetworkInterface *net = NetworkInterface::get_default_instance();

if (!net) {
    printf("Error! No network inteface found.\n");
    return 0;
}

net->connect();
```

В идеале, каждая ваша программа должна выглядеть именно так. Тот пример, который мы смотрели с WiFi - слишком конкретный. В дальнейшем мы используем этот подход при обращении к другому протоколу канального уровня, а пока лишь дополним наш пример новым функционалом.

3.2.5. Пример с сокетами

Теперь, когда есть интернет-соединение, самое время посмотреть, как происходит работа с сокетами.

Попробуем добавить следующую функцию в наш WiFi-пример:

```
void http_demo(NetworkInterface *net)
{
    // Open a socket on the network interface, and create a TCP connection to mbed.org
    TCPSocket socket;
    socket.open(net);

    SocketAddress a;
    net->gethostbyname("ifconfig.io", &a);
    a.set_port(80);
    socket.connect(a);
    // Send a simple http request
    char sbuffer[] = "GET / HTTP/1.1\r\nHost: ifconfig.io\r\n\r\n";
    int scount = socket.send(sbuffer, sizeof sbuffer);
    printf("sent %d [%.*s]\n", scount, strstr(sbuffer, "\r\n") - sbuffer, sbuffer);

    // Recieve a simple http response and print out the response line
    char rbuffer[64];
    int rcount = socket.recv(rbuffer, sizeof rbuffer);
    printf("recv %d [%.*s]\n", rcount, strstr(rbuffer, "\r\n") - rbuffer, rbuffer);

    // Close the socket to return its memory and bring down the network interface
    socket.close();
}
```

И вызовем эту функцию в main, непосредственно перед закрытием соединения по WiFi:

```
http_demo(wifi);
```

Вы видите, что здесь передается уже указатель на базовый класс NetworkInterface, то есть используется полиморфизм, о котором было сказано выше.

После запуска примера увидите следующую выдачу. Посылается простой GET-запрос и на него приходит ответ:

```
Success

MAC: 2c:f4:32:81:6a:00
IP: 192.168.0.60
Netmask: 255.255.255.0
Gateway: 192.168.0.1
RSSI: -46

sent 38 [GET / HTTP/1.1]
recv 64 [HTTP/1.1 200 OK]

Done
```

Reset user tour on this page