Подключение WI-FI модуля ESP-01 к микроконтроллеру STM32

К.А. Волобуев, И.В. Трубин

Новосибирский государственный технический университет, г. Новосибирск, РФ

Аннотация: В данной статье рассматривается модуль *ESP-01*, на котором установлен чип *ESP8266*, представляющий собой миниатюрный микроконтроллер с *Wi-Fi* передатчиком. Чтобы воспользоваться возможностями данного модуля было осуществлено его подключение к микроконтроллеру *STM32F103C8T6* на отладочной плате *Blue Pill*. Управление осуществляется по интерфейсу *UART* с помощью *AT-команд*. В статье приведены фрагменты кода, позволяющие настроить взаимодействие между модулем *ESP-01* и микроконтроллером *STM32F103C8T6*. В приведенном коде показано, как настроить взаимодействие, чтобы при обращении к *web*-серверу, созданному модулем *ESP-01*, отключать/включать светодиод, подключенный к порту *PC13* на отладочной плате *Blue Pill*. В ответ на запрос клиента сервер отправляет ответ с текущим состоянием светодиода (*ledon/ledoff*). Статья будет полезна к ознакомлению студентам, желающим изучить беспроводные сети и технологии взаимодействия устройств в сети *INTERNET*.

Ключевые слова: ESP-01, ESP8266, Wi-Fi, передача данных, UART, STM32, STM32F103C8T6, отладочная плата Blue Pill, AT-команды.

Введение

В наше время широкого развития технологий мы постоянно сталкиваемся с различными каналами связи и коммуникационными сетями. Яркий пример из жизни каждого - телефон и компьютер, которые с помощью проводного подключения, или *Wi-Fi* соединения взаимодействуют с другими устройствами, а также, набирающая популярность технология «Умный Дом», в которой различные модули и датчики соединяются в единую беспроводную сеть.

Wi-Fi модуль *ESP-01* один из самых популярных модулей серии *ESP8266*. Он может взаимодействовать с другими устройствами (компьютер или микроконтроллер) через *UART* с помощью *AT* команд и устанавливать беспроводное соединение. На *Рис.* 1. можно увидеть популярную схему взаимодействия устройств с помощью беспроводных сетей, где в качестве точки доступа выступает роутер, а в качестве клиентов - различные устройства, такие как телефон или компьютер.

В данном материале рассматривается ситуация, когда модуль *ESP-01* работает в качестве точки доступа (роутера) и разбирается процесс настройки связи отладочной платы *Blue Pill* на базе микроконтроллера *STM32F103C8T6*, модуля *ESP-01* и смартфона.

Рис. 1. Схема взаимодействия домашних устройств

Теоретические сведения

Wi-Fi – это беспроводная технология передачи данных. В принцип работы *Wi-Fi* положена передача зашифрованных сигналов посредством *СВЧ*-волн (сверхвысокочастотные волны) на небольшие (десятки метров) расстояния. Схема сети состоит минимум из двух элементов: точка доступа и клиент.

Для подключения к сети необходимо знать идентификатор сети. Точка доступа передаёт его с помощью специальных сигнальных пакетов на скорости 0,1 *Мбит/с* каждые 100 миллисекунд. Зная идентификатор сети, клиент может выяснить возможность подключения к данной точке доступа. При попадании в зону действия двух точек доступа с идентичными идентификаторами приёмник может выбирать между ними на основании данных об уровне сигнала. Стандарт *Wi-Fi* даёт клиенту полную свободу при выборе критериев для соединения.

TCP – Transmission Control Protocol (Протокол Управления Передачей) – это протокол транспортного уровня, предоставляющий транспортировку (передачу) потока данных, с необходимостью предварительного установления соединения, благодаря чему гарантирует уверенность в целостности получаемых данных, также выполняет повторный запрос данных в случае потери данных или искажения. Помимо этого, протокол *TCP* отслеживает дублирование пакетов и в случае обнаружения - уничтожает дублирующиеся пакеты.

Установление соединения *TCP*

Если процесс, работающий на одном хосте, хочет установить соединение с другим процессом на другом хосте, то тот процесс, инициирующий соединение называется «клиентом», а другой узел называется «сервером».

Перед началом передачи каких-либо данных, согласно протоколу *TCP*, стороны должны установить соединение. Соединение устанавливается в три этапа (процесс «трёх-кратного рукопожатия» *TCP*).

Запрашивающая сторона (которая, как правило, называется клиент) отправляет *SYN* сегмент, указывая номер порта сервера, к которому клиент хочет подсоединиться, и исходный номер последовательности клиента (*ISN*).

Сервер отвечает своим сегментом *SYN*, содержащим исходный номер последовательности сервера. Сервер также подтверждает приход *SYN* клиента с использованием *ACK* (*ISN* + 1). На *SYN* используется один номер последовательности.

Клиент должен подтвердить приход *SYN* от сервера своим сегментов *SYN*, содержащий исходный номер последовательности клиента (*ISN*+1) и с использованием *ACK* (*ISN*+1). Бит *SYN* установлен в 0, так как соединение установлено. Полная схема установление *TCP* соединения представлена на *Рис*. 2.

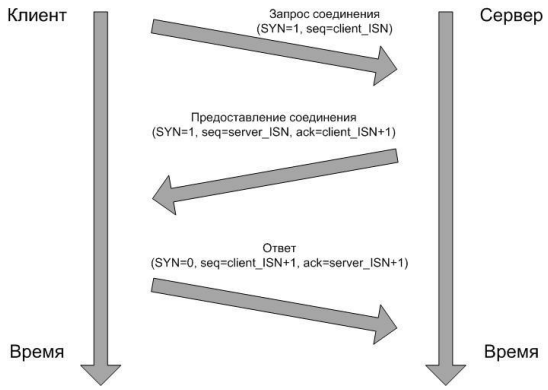

Рис. 2. Схема установление *TCP* соединения

После установления соединения *TCP*, эти два хоста могут передавать данные друг другу, так как соединение является полнодуплексным, они могут передавать данные одновременно.

HTTP (англ. *HyperText Transport Protocol*) - протокол передачи данных, используемый обычно для получения информации с веб-сайтов. Он использует «клиент-серверную» модель. То есть существуют клиенты, которые формируют и отправляют запрос. И серверы, которые слушают запросы и, соответственно, на них отвечают.

Рассмотрим *HTTP* запрос от клиента к серверу на *Рис.* 3.

HTTP

```
Get / HTTP/1.1
Accept-Language: en-us
Accept: */*
Connection: close
Host: 192.168.1.2
```

Рис. 3. *HTTP* запрос

Наибольший интерес здесь представляют первая и последняя строки. В первой строке указан метод чтения данных - *GET*. После *GET* стоит символ "/", что означает, что запрашивается главная (корневая) страница по *URL* (англ. *Uniform Resource Locator*) пути. Простыми словами, *URL* - идентификатор какого-либо ресурса в сети. Запись *HTTP*/1.1 – это версия протокола. В нижней строке указывается адрес сервера или имя, на котором располагается нужный ресурс.

Технические характеристики модуля ESP-01, с чипом ESP8266

Рис. 4. Фото модуля *ESP-01*

ESP8266 ESP-01 V090 (он же Wi07c) самый популярный модуль. *PCB* антенна обеспечивает дальность до 400 *м* на открытом пространстве. Существует старая версия модуля V080, на которой выполнены только 4 контакта.

Назначение выводов модуля

- ***GND:*** Питание модуля «-»;
- ***GPIO2:*** Цифровой вход/выход программируемый;
- ***GPIO0:*** Цифровой вход/выход программируемый, также используется для режимов загрузки;
- ***RX:*** *UART* приемник;
- ***TX:*** *UART* передатчик;
- ***CH_PD:*** Включение / отключение режима низкого энергопотребления, для использования Wi-Fi необходимо подключить к 3.3 *В*;
- ***RST:*** Перезагрузка модуля, для сброса модуля необходимо подключить к выводу 3.3 *В*;
- ***VCC:*** Питание модуля «+» 3.3 *В*.

Технические характеристики чипа

- Напряжение питания: 3 *В* ~ 3.6 *В*;
- Максимальный рабочий ток: 220 *мА*;
- Поддержка беспроводного стандарта: 802.11*b/g/n;*
- Рабочая частота: 2.4 *ГГц*;
- Режимы: *P2P* (клиент), *soft-AP* (точка доступа);
- Количество *GPIO*: 2;
- *FLASH* память: 1024 *кб*;
- Выходная мощность в режиме 802.11*b*: +19.5 *dBm*;
- Габариты: 24.8 x 14.3 x 8 *мм*;
- Рабочий диапазон температур: -40 *°C* ~ +125 *°C*.

Подключение модуля ESP-01 к STM32 и работа с AT-командами

Для работы модуля *ESP-01*, необходим источник питания постоянного тока, который должен выдавать 3.3 *В* и током не менее 220 *мА*. Кроме того, логическая единица для интерфейса *UART* находится в диапазоне 2.7-3.6 *В*, то есть на вывод *RX* необходимо подавать напряжение 2.7-3.6 *В*, а с вывода *TX* будет выводиться напряжение 2.7-3.6 *В* (так же и для других выводов).

Модуль *ESP-01* чувствителен к просадкам по напряжению, в связи с этим, для него лучше всего использовать внешний стабилизатор напряжения на 3.3 *В*, с выходным током 0,5 *А*.

Подключение и настройка

В заводской прошивке *Wi-Fi* модуль общается с управляющей платой через «*AT-команды*» по интерфейсу *UART*. Схема подключения контроллера и модуля представлены на *Рис*. 5. Подключение, собранное на макетной плате по схеме показано на *Рис*. 6.

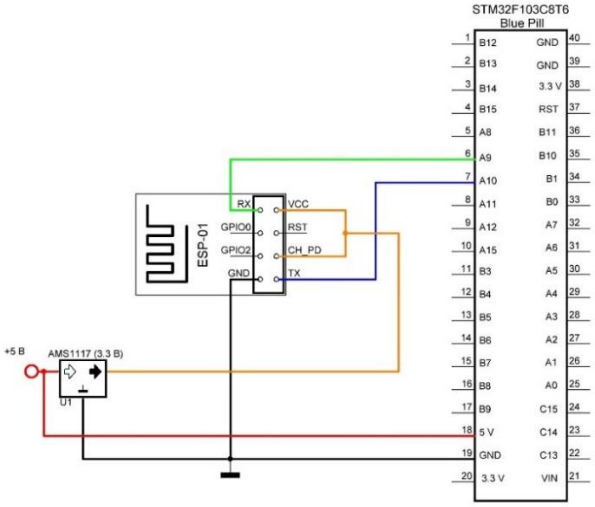

Рис. 5. Схема подключения *ESP-01* к отладочной плате *Blue Pill*

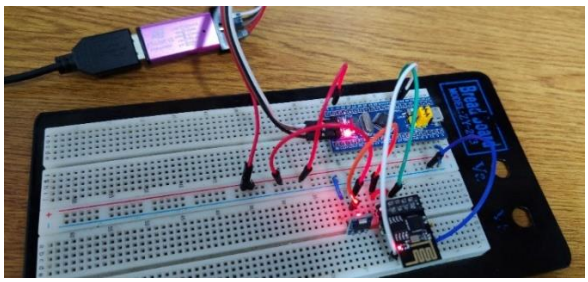

Рис. 6. Собранная схема

Данный модуль можно использовать как в режиме *Station* (*Wi-Fi* клиент), так и в режиме *SoftAP* (точка доступа). В данной статье мы будем использовать подключение с помощью точки доступа.

Таблица 1.

Список основных необходимых АТ-команд

AT	Проверка состояния модуля.
AT+RST	Перезагрузка модуля.
AT+CWMODE_CUR	Переключение режима *Wi-Fi* для текущего сеанса.
AT+CWSAP_CUR	Создание *SoftAP* (точки доступа) для текущего сеанса.
AT+CIPAP_CUR	Посмотреть/установить *IP* адрес в режиме *SoftAP* (точка доступа) для текущего сеанса.
AT+CIPSEND	Команда отправки данных. Длина данных в пакете до 2048 *байт*. После получения данной команды модуль выводит приглашение ">" и переходит в режим приема данных через *UART*, после приема данных необходимой длины передает их в радиоканал. При успешной передаче возвращает "*SEND OK*". При неудаче "*ERROR*".
AT+CIPMUX	Выбрать режим одиночного или множественных подключений.
AT+CIPSERVE	Запустить (перезапустить) *TCP* сервер.
AT+CIPSTO	Установить/посмотреть таймаут сервера. Таймаут в секундах от 0 до 7200.
AT+CIPMODE	Установить сквозной режим.
+IPD	Получить данные.

При корректном выполнении вышеприведенных команд модуль возвращает "*OK*". При неудаче "*ERROR*".

Когда модуль получает данные по сети, то он их отправляет в *UART* командой *+IPD*. Более детальные описания команд приведены в [3].

Код программы

Для понимания взаимодействия напишем код, который каждый раз при обновлении страницы будет включать/выключать светодиод, подключенный к порту *PC13* на отладочной плате и в ответ на запрос посылать текущее состояние светодиода (*ledon/ledoff*).

Для начала подключим необходимые файлы, а именно "*stm32f10x.h*" для работы с микроконтроллером и "*main_init.c*", в котором сконфигурирована инициализация *МК*, настройка таймера и портов.

```
//======= Используемые библиотеки =======
#include "stm32f10x.h"
#include "main_init.c"
```

Затем - создадим массив «*Buffin*», в который будем принимать ответы от модуля *ESP-01*, переменную «*bf*» будем использовать как индекс массива, переменной «*flag*» будем устанавливать включение и выключение светодиода, а состоянием переменной «*msg*» будем проверять наличие принятого запроса.

```
//====Описание глобальных переменных ====
char Buffin[512];
uint16_t bf = 0;
uint8_t flag = 0;
uint8_t msg = 0;
```

Далее, в файле "*main_init.c*" настраиваем порт *PC13* как выход.

```
//=========== Настройка портов =========
// Включаем тактирование порта GPIOC
RCC->APB2ENR = RCC_APB2ENR_IOPCEN;
// Настраиваем порт PC13 как выход
GPIOC->CRH = GPIO_CRH_MODE13;
```

Настроим последовательный интерфейс *UART* на подсистеме *USART*1.

```
//=======Настройка USART1 ===============
// Включаем тактирование порта GPIOA
// и блока альтернативных функций
   RCC->APB2ENR |= (RCC_APB2ENR_IOPAEN |
          RCC_APB2ENR_AFIOEN);
RCC->APB2ENR |= RCC_APB2ENR_USART1EN;
// Включаем тактирование USART1
// PA9 (TX1) AFIO Push-Pull
// PA10 (RX1) HiZ
// Вначале обязательно устанавливаем
// пары бит в "00"
     GPIOA->CRH &= ~(GPIO_CRH_MODE9 |
     GPIO_CRH_CNF9 | GPIO_CRH_MODE10 |
           GPIO_CRH_CNF10);
// Потом нужные биты устанавливаем в '1'
    GPIOA->CRH |= (GPIO_CRH_MODE9_0 |
   GPIO_CRH_CNF9_1 | GPIO_CRH_CNF10_0);
// Рассчитаем частоту для связи с ESP8266
// PCLK2 / Baud = 72000000 / 115200 = 625
USART1->BRR = 625;
// Включаем USART, передатчик и приемник
USART1->CR1 = USART_CR1_UE | USART_CR1_TE |
              USART_CR1_RE;
// Разрешаем прерывание по приёму RX
USART1->CR1 |= USART_CR1_RXNEIE;
// Настройки по умолчанию: 8 инф. бит,
// 1 стоповый бит, контроля чётности нет
// Разрешаем прерывания UART1 в
// контроллере прерываний
NVIC_EnableIRQ(USART1_IRQn);
```

Далее, вернемся в файл "*main.c*" и напишем функцию, которая будет считывать данные с порта *RX* интерфейса *UART* и записывать их в массив «*Buffin*».

```
// ===== Обработка прерывания USART1 =====
// USART1->SR, бит RXNE сбрасывается при
// чтении USART1->DR,
// записывать в него ноль нужно только при
// мультибуферной коммуникации
void USART1_IRQHandler() {
    Buffin[bf++] = USART1->DR;
// Если принят символ конца строки
// считаем, что пришел запрос от клиента
    if(Buffin[bf-1] == '\n'){msg = 1;}}
```

Следующий шаг - создание функций для отправки данных на модуль *ESP-01*. Сначала создадим функцию для передачи символа, а после, с помощью ее вызова будем передавать строки символов.

```
// ========Отправка данных на ESP ========
// Функция передачи символа по USART1
void Tx1(char Symbol) {
while((USART1->SR & USART_SR_TXE)==0) {};
    USART1->DR = Symbol;
}
// Функция передачи массива символов
// (строки) по USART1
// пока не встретится "0" байт
void ESP_print(char *pStr) {
    uint8_t i = 0;
    do Tx1(pStr[i++]);
    while(pStr[i] != 0);
}
// Функция передачи строки по USART1
// с переносом строки
void ESP_println(char *pStr) {
    uint8_t i = 0;
    do Tx1(pStr[i++]);
    while(pStr[i] != 0);
// Вставить символ переноса каретки
    Tx1('\r');
// Вставить символ новой строки
    Tx1('\n');
}
```

Чтобы иметь возможность вносить программные задержки во время выполнения кода, напишем функции задержки.

```
// ========= Функции задержки ===========
// Функция задержки в мкс (можно от 1 мкс)
void delay_us(uint32_t us) {
    static volatile uint32_t n;
    n = us * 72;
// SystemCoreClock/1000000 = 72
    DWT_CYCCNT = 0;
    while(DWT_CYCCNT < n) ;
}
// Функция задержки в мс от 1 мс до
// 59 сек., больше нельзя, т.к. происходит
// переполнение
void delay_ms(uint32_t ms) {
    delay_us(ms * 1000);
}
```

Создадим функции для работы с массивами символов. Они необходимы для разбора *HTTP*-запроса и формирования ответа клиенту. Так как большинство из этих функций выполняют стандартные действия, излишние комментарии будут опущены.

```
//= Функции работы с массивами символов =
// Функция подсчета ненулевых символов в
// массиве
uint16_t strlen2(char* p) {
    uint16_t i = 0;
    while(*p++) i++;
    return i;
}

// Функция очистки массива
void clear_mass(char *Buff){
    uint16_t l = strlen2(&Buff[0]);
    for(uint16_t i = 0; i < l; i++){
        Buff[i] = 0;
    }
}
// Функция сложения строк
void mass_add(char *Buffer, char *Buff){
    uint16_t l = strlen2(&Buffer[0]);
    uint16_t k = strlen2(&Buff[0]);
    uint16_t j = 0;
    for(int i = l; i < l + k; i++){
        Buffer[i] = Buff[j];
        j++;
    }
}
// Функция перезаписи массива
void mass_new(char *Buffer, char *Buff){
    uint16_t l = strlen2(&Buffer[0]);
    for(uint16_t i = 0; i < l; i++){
        Buffer[i] = 0;
    }
    uint16_t k = strlen2(&Buff[0]);
    for(uint16_t i=0; i < k; i++){
        Buffer[i] = Buff[i];
    }
}
// Функция поиска символа в массиве,
// возвращает позицию
uint16_t cmp(char* str1, char p){
    uint16_t i = 0;
    uint16_t n = strlen2(str1);
        while(i < n){
        if(str1[i++] == p){
            return i;
        }
    }
    return 0;
}
// Функция для поиска подстроки в строке
// возвращает позицию, начала подстроки
uint16_t pos(char *s, char *c){
    uint16_t  i, j;
    uint16_t  lenC = strlen2(c);
    uint16_t  lenS = strlen2(s);
    if(lenS != 0){
        for(i = 0; i <= lenS - lenC; i++){
            for(j = 0; j <= lenC; j++){
                if(s[i + j] != c[j]){
                    break;
                }
            }
            if(j == lenC){return i;}
        }
    }
    return -1;
}
```

Также, в процессе работы со строками, для передачи чисел, понадобится написать функцию, которая будет преобразовывать число в строку.

```
// Функция преобразования
// знакового 16 битного числа в строку
 void int16ToStr(uint16_t  Number, char
                 *pStr) {
    clear_mass(pStr);
    if(Number < 10){
        pStr[0] = Number + '0';
        pStr[1] = 0; pStr[2] = 0;
```

```
        pStr[3] = 0; pStr[4] = 0;
    } else if(Number < 100){
        pStr[0] = Number / 10 + '0';
        pStr[1] = Number % 10 + '0';
        pStr[2] = 0; pStr[3] = 0;
        pStr[4] = 0;
    } else if(Number < 1000){
        pStr[0] = Number / 100 + '0';
        pStr[1] = Number / 10 % 10 + '0';
        pStr[2] = Number % 100 + '0';
        pStr[3] = 0; pStr[4] = 0;
    }
    pStr[5] = 0; // конец строки
}
```

Теперь, создадим несколько функций необходимых для работы с принятыми данными и формирования ответа клиенту.

```
// ========== Функции ESP-01 ==========
// Функция очистки входного буфера
void clear_Buffer(char *Buff){
    for(uint16_t  i = 0; i < 511; i++){
        Buff[i] = 0;
    }
    bf = 0;
}
```

Далее - создадим функцию, которая будет ждать и проверять ответ о корректном выполнении команды от модуля.

```
// Функция проверки наличия ответа «OK»
// от ESP-01
void OK(char *Buff){
    int16_t n = -1;
    while(n == -1){
        n = pos(Buff, "OK");
        delay_ms(10);
    }
    clear_Buffer(Buff);
}
```

Для того, чтобы послать ответ на запрос клиента необходимо сформировать заголовок и тело посылки. Необходимо знать «*id*» клиента, от которого поступил запрос, длину заголовка и тела ответной посылки, а также правильно указать все переносы строки и возвраты каретки.

```
// Функция формирования ответа клиенту
  void otvet_klienty(uint16_t  ch_id, char
                  *Buff) {
// Массив для трансформации числа в строку
    char mass_int[6];
// Массив для отправки посылки
    char send[22] = "AT+CIPSEND=";
// Массив тела посылки
    char Content[20];
// Переменная размера посылки
    uint16_t  lc = 0;
// Включение/выключение светодиода,
// в зависимости от флага
    if(flag == 0){
        GPIOC->BRR |= GPIO_BRR_BR13;
        flag = 1;
        mass_new(Content, "ledon");
    } else{
        GPIOC->BSRR |= GPIO_BRR_BR13;
        flag = 0;
        mass_new(Content, "ledoff");
    }
// Подсчитываем длину контента страницы
// Переводим числа в строки
// После этого добавляем строки к
// массиву для отправки
// 80-Длина заголовка
    lc = strlen2(Content);
    int16ToStr(ch_id, &mass_int[0]);
    mass_add(send, mass_int);
    mass_add(send, ",");
    int16ToStr(80 + lc, &mass_int[0]);
    mass_add(send, mass_int);
    ESP_print(send);
    Tx1('\r'); Tx1('\n');
    delay_ms(50);
```

Отправка посылки происходит только после того, как модуль сообщит о готовности принимать пакет для отправки с помощью символа '>'.

```
uint16_t  yes = cmp(Buff, '>');
    if(yes) {
// Отправка заголовка
        ESP_print("HTTP/1.1 200 OK");
        Tx1('\r'); Tx1('\n');
ESP_print("Content-Type: text/html");
        Tx1('\r'); Tx1('\n');
        ESP_print("Connection: close");
        ESP_print("Content-Length: ");
        int16ToStr(lc, &mass_int[0]);
        ESP_print(mass_int);
        Tx1('\r'); Tx1('\n');
        Tx1('\r'); Tx1('\n');
// Отправка содержимого пакета
        ESP_print(Content);
        delay_ms(10);
}}
```

Перейдем к главному коду программы. Проинициализируем микроконтроллер, выключим светодиод, подключенный к порту *PC13*, и проведем первоначальную настойку модуля *ESP-01*.

```
// =============== MAIN =================
int  main(void) {
    main_init(); // Инициализация МК
// ============ GPIO off  ===============
// Отключаем светодиод
    GPIOC->BSRR |= GPIO_BRR_BR13;
// ========== Конфигурация ESP ==========
// Отправка команды перезагрузки ESP-01
    ESP_println("AT+RST");
    delay_ms(1000);
// Очищаем входной буфер от мусора
    clear_Buffer(&Buffin[0]);
// Выставляем режим работы 2
// (точка доступа/станция)
    ESP_println("AT+CWMODE_CUR=2");
// Ждем ответа "OK" от модуля ESP-01
    OK(&Buffin[0]);
// Задаем имя, пароль, канал,
// тип шифрования 3-WPA2_PSK
ESP_println("AT+CWSAP_CUR=\"ESP8266\",\"123
             4567890\",5,3");
    OK(&Buffin[0]);
// Устанавливаем локальный IP-адрес
// точки доступа
        ESP_println("AT+CIPAP_CUR=\
            "192.168.4.1\"");
    OK(&Buffin[0]);
// Включаем сквозной режим передачи
// данных
    ESP_println("AT+CIPMODE=0");
    OK(&Buffin[0]);
// Включаем возможность множественного
// подключения
```

```
    ESP_println("AT+CIPMUX=1");
    OK(&Buffin[0]);
// Запускаем TCP-сервер на 80-ом порту
    ESP_println("AT+CIPSERVER=1,80");
    OK(&Buffin[0]);
// Ставим таймаут сервера 1 сек
    ESP_println("AT+CIPSTO=1");
    OK(&Buffin[0]);
```

Далее, в бесконечном цикле будем проверять переменную «*msg*» на равенство единице, и находить во входном буффере подстроку "*IPD*", которая обозначает, что принят пакет данных от клиента. При нахождении данной подстроки будем вызывать функцию ответа клиенту.

```
// ============ ГЛАВНЫЙ ЦИКЛ ============
while(1) {
    if(msg == 1){
        msg = 0;
        int16_t n = -1;
        uint16_t ch_id = 0;
        n = pos(Buffin, "IPD");
        if(n >= 0){
            ch_id = Buffin[n+4] - '0';
        otvet_klienty(ch_id, &Buffin[0]);
        }
        clear_Buffer(&Buffin[0]);
        }
        delay_ms(10);
}}
```

Следует пояснить, что отправка ответа на запрос происходит в следующем формате: "*AT+CIPSEND=0,82*", где 0 - «*id*»-клиента, 82 - длина пакета.

В результате, после загрузки данного кода и подачи питания, контроллер перезагрузит и настроит модуль *ESP-01* и в списке *Wi-Fi* сетей телефона увидим сеть с именем "*ESP8266*" и паролем "*1234567890*" (*Рис*. 7.), подключившись к которой и перейдя по адресу http://192.168.4.1/ отправим запрос на контроллер.

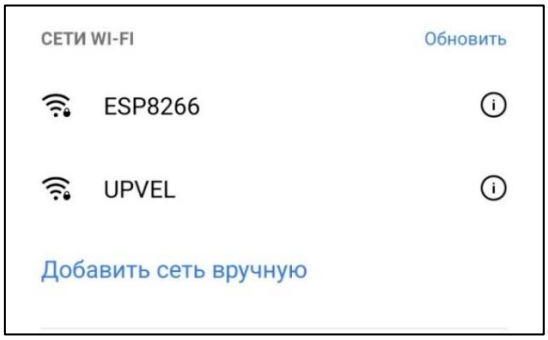

Рис. 7. Доступные для подключения *Wi-Fi* сети

Для более удобной работы разберем *HTTP*-запрос клиента. Запрошенная страница имеет адрес http://192.168.4.1/, что является корневым адресом. Но можно обратиться и к другому адресу на сервере, например, http://192.168.4.1/ledon/ и http://192.168.4.1/ledoff/ для того чтобы выполнять только определенное действие, а не просто менять состояние при обращении.

Для этого потребуется немного изменить функцию формирования ответа клиенту, а если конкретнее, то заменить условия с флагом:

```
if(flag == 0){
    GPIOC->BRR |= GPIO_BRR_BR13;
    flag = 1;
    mass_new(Content, "ledon");
}else{
    GPIOC->BSRR |= GPIO_BRR_BR13;
    flag = 0;
    mass_new(Content, "ledoff");
}
```

На следующий код:

```
if(pos(Buff, "/ledon") >= 0){
    GPIOC->BRR |= GPIO_BRR_BR13;
    flag = 1;
    mass_new(Content, "ledon");
}
if(pos(Buff, "/ledoff")>=0){
    GPIOC->BSRR |= GPIO_BRR_BR13;
    flag = 0;
    mass_new(Content, "ledoff");
}
```

Прежде чем изменить состояние светодиода, после принятого запроса, рассмотрим пришедший запрос. К примеру, если пришел запрос "*GET /ledon HTTP/1.1*", то включим светодиод и отправим соответствующий ответ. Соответственно если пришел запрос "*GET /ledoff HTTP/1.1*", то выключим светодиод.

Если придет запрос, в котором не будет искомой подстроки, то светодиод никак не изменит своего состояния.

Дальнейшие настройки будут опущены, т.к. это уже в меньшей степени относится к программированию *STM32*, и в большей к разбору логики работы *WEB-сети* и протокола *HTTP*.

Выводы

- Модуль *ESP-01* является хорошим и доступным модулем для создания каналов связи по беспроводной сети *Wi-Fi*.
- Модулем *ESP-01* в стандартной прошивке возможно управлять по протоколу *UART* с помощью *AT-команд*.
- Процесс подключения модуля к микроконтроллеру достаточно прост и имеет довольно много настроек, более детально описанных в [3].
- Использование *Wi-Fi* – модулей в проектах с микроконтроллером *STM32F103C8T6* дает возможность управления различными устройствами беспроводным путем.
- *Wi-Fi* – модули крайне актуальны в современном мире, в том числе в проектах *IoT* и технологиях умного дома.

Литература

[1] Документация на микроконтроллер STM32F103. – URL: https://www.st.com/content/ccc/resource/technical/document/reference_manual/59/b9/ba/7f/11/af/43/d5/CD001

71190.pdf/files/CD00171190.pdf/jcr:content/translations/en.CD00171190.pdf (дата обращения: 08.07.2022).
[2] Документация на микроконтроллер ESP8266: Ultrasonic Ranging Module. – URL: https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf (дата обращения: 08.07.2022).
[3] Список АТ-команд для ESP8266. – URL: https://esp8266.ru/esp8266-at-commands-v022/ (дата обращения: 08.07.2022).
[4] Документация на модуль ESP-01. – URL: https://docs.ai-thinker.com/_media/esp8266/esp8266_series_modules_user_manual_en.pdf (дата обращения: 08.07.2022).
[5] Технические параметры модуля ESP-01. – URL: https://robotchip.ru/obzor-modulya-esp-01-na-chipe-esp8266/ (дата обращения: 08.07.2022).
[6] Документация на отладочную плату Blue Pill. – URL: https://www.belchip.by/sitedocs/31025.pdf (дата обращения: 08.07.2022).
[7] Принцип работы Wi-Fi. – URL: https://wifigid.ru/besprovodnye-tehnologii/kak-rabotaet-wi-fi/ (дата обращения: 08.07.2022).
[8] Протокол *TCP*. – URL: https://pc.ru/docs/network/tcp (дата обращения: 08.07.2022).
[9] Протокол *HTTP*. – URL: https://habr.com/ru/post/307714/ (дата обращения: 08.07.2022).

Кирилл Андреевич Волобуев – студент группы АА-07 кафедры Автоматики НГТУ.
E-mail: kirya.volobuev@mail.ru

Игорь Витальевич Трубин – ассистент кафедры Защиты Информации НГТУ, зам. директора ООО «КБ Автоматика».
E-mail: tiv.kba@gmail.com

Статья поступила 10.11.2022.

Coniungens WI-FI ESP-01 Moduli ad STM32 Microcontroller

K.A. Volobuev, I.V. Trubin
Novosibirscum Civitatis Technicae Universitas, Novosibirscum, Foederatio Russica

Abstract: Hic articulus de ESP-01 moduli disserit, in quo chip ESP8266 inaugurata est, quae minima microcontroller est cum transmisso Wi-FI. Uti facultatibus huius moduli, coniunctum erat cum STM32F103C8T6 microcontroller in tabula Pillula caerulea. Imperium exercetur per UART instrumenti utendi AT imperat. Articulus codicem continet fragmenta quae te permittunt commercium inter ESP-01 moduli et STM32F103C8T6 microcontrollerum configurare. Codex infra ostendit commercium configurare ut cum accessu interretiali servo ab ESP-01 modulo creato, disable / efficiat ducatur ad PC13 portum in tabula progressionis Venetae Pill. Propter petitionem clientis, servo responsionem mittit cum statu praesentis LED (ledon/ledoff). Articulus usui erit alumnis notis qui wireless retiacula et technologias studere volentibus pro commercio machinis in retis interretialibus.

Key words: ESP-01, ESP8266, Wi-Fi, notitia translationis, UART, STM32, STM32F103C8T6, Pillula progressionis tabulae caeruleae, AT imperat.

References

[1] Microcontroller documentation STM32F103. – URL: https://www.st.com/content/ccc/resource/technical/document/reference_manual/59/b9/ba/7f/11/af/43/d5/CD00171190.pdf/files/CD00171190.pdf/jcr:content/translations/en.CD00171190.pdf.
[2] Microcontroller documentation ESP8266: Ultrasonic Ranging Module. – URL: https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf .
[3] List of AT commands for ESP8266. – URL: https://esp8266.ru/esp8266-at-commands-v022/.
[4] Documentation for the module ESP-01. – URL: https://docs.ai-thinker.com/_media/esp8266/esp8266_series_modules_user_manual_en.pdf.
[5] Technical parameters for the module ESP-01. – URL: https://robotchip.ru/obzor-modulya-esp-01-na-chipe-esp8266/.
[6] Документация на отладочную плату Blue Pill. – URL: https://www.belchip.by/sitedocs/31025.pdf.
[7] Principle of working of Wi-Fi. – URL: https://wifigid.ru/besprovodnye-tehnologii/kak-rabotaet-wi-fi/.
[8] Protocol *TCP*. – URL: https://pc.ru/docs/network/tcp.
[9] Protocol *HTTP*. – URL: https://habr.com/ru/post/307714/.

Kirill Volobuev is a student of group AA-07 of the Department of Automation of the Novosibirsk State Technical University.
E-mail: kirya.volobuev@mail.ru

Igor Trubin - assistant of the Department of Information Protection, NSTU, deputy. director of KB Avtomatika LLC.
E-mail: tiv.kba@gmail.com

The paper has been recieved 10.11.2022.