Zastosowanie modułu Wi-Fi ESP-12 (1)

Wprowadzenie

Jednym z ciekawszych modułów Wi-Fi dostępnych na rynku jest ESP-12-Q. Bazuje on na 32-bitowym mikrokontrolerze ESP8266 firmy Espressif. Ten mikrokontroler integruje 32-bitowy rdzeń firmy Tensilica, standardowe, cyfrowe układy interfejsowe, przełącznik antenowy, balun RF, wzmacniacz mocy RF, odbiornik ze wzmacniaczem, filtry oraz moduły zarządzające zasilaniem. Dodatkowo, do zastosowania w module Wi-Fi, zintegrowano w nim stos TCP/IP.

Niewielki pobór mocy oraz moduły komunikacyjne opracowane z użyciem wspomnianego we wstępie mikrokontrolera idealnie nadają się do zastosowania we własnych projektach z dziedziny IoT. Kurs będzie obejmował użycie modułów ESP przy tworzeniu własnych elementów IoT, takich jak: bezprzewodowa stacja pogodowa, bezprzewodowe moduły wykonawcze oraz czujniki.

Na początek

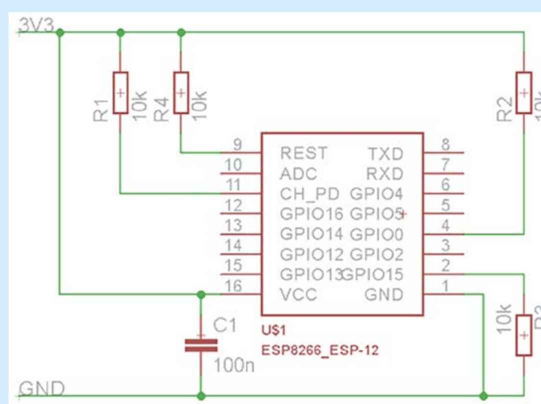
W kursie zostanie użyty moduł ESP-12-Q. Jest on zgodny z wersją ESP-12, różni się jedynie dodatkowymi pinami na spodzie PCB. Moduł wymaga zasilania napięciem 3,3 V. Przy współpracy z systemem nadrzędnym zasilanym napięciem innym niż 3,3 V, należy zastosowań konwerter poziomu napięcia.

Fabrycznie nowy moduł pracuje pod kontrolą oprogramowania wykonanego przez producenta. Komunikacja odbywa się w oparciu o transmisję UART, parametry połączenia są następujące:

1. Prędkość transmisji: 115200 b/s (w starszych wersjach jest to 9600 b/s).
2. 8 bitów danych.
3. Bez bitu parzystości.
4. 1 bit stopu.

Sposób sterowania wykorzystuje zasadę *request/response* i opiera się o komendy AT. Do pierwszych testów wystarczy zwykły konwerter USB/UART. Należy zwrócić uwagę na napięcie na liniach Rx i Tx, aby nie przekraczały poziomu 3,6 V, czyli maksymalnego napięcia dla mikrokontrolera ESP8266.

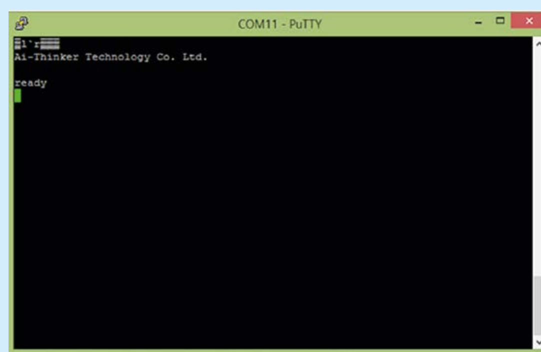
Z racji tego, że układ mikrokontrolera ESP8266 można programować własnym kodem z wykorzystaniem komunikacji UART, pinu GPIO0 użyto do sterowania bootloaderem. Po ustawieniu GPIO0 bootloader ładuje kod programu z pamięci Flash, natomiast po wyzerowaniu oczekuje na nowy kod i zadpisuje go w pamięci Flash. Podstawową aplikację modułu pokazano na **rysunku 1**. Rezystory R1…R3 dodano, aby w momencie pisania własnego oprogramowania nie stworzyć zwarcia poprzez zmianę poziomu pinu podłączonego na stałe do któregoś z biegunów zasilania.



Rysunek 1. Podstawowa aplikacja modułu ESP-12

Pierwsze uruchomienie

Moduł należy podłączyć do przejściówki USB/UART krzyżując linie Rx i Tx. Do komunikacji można zastosować dowolny terminal – tutaj zostanie użyty darmowy program *Putty*. Po włączeniu zasilania w konsoli powinny pojawić się wiadomości takie, jak na **rysunku 2** lub bardzo podobne (zależnie od wersji oprogramowania). Wiadomość „ready" oznacza poprawny start oprogramowania i gotowość do pracy.



Rysunek 2. Komunikat odebrany po ustanowieniu komunikacji z modułem ESP-12 przez UART

Aby przetestować poprawną pracę modułu, można sprawdzić wersję oprogramowania za pomocą polecenia *AT+GMR*. Odpowiedź modułu, jak na **rysunku 3**, informuje nas nie tylko w wersji programu, ale też o tym, że wszystko zostało poprawnie połączone i że ustawiliśmy poprawne parametry transmisji.

Kilka słów wyjaśnienia odnośnie do samych komend. Wszystkie zaczynają się od przedrostka „AT", za którym – najczęściej po znaku „+" – jest umieszczana komenda

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 77642, pass: 3220ppmm

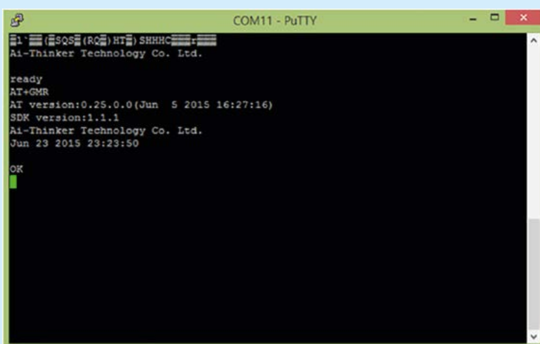
i jej argumenty. Polecenie jest interpretowane w chwili, gdy moduł odbierze znaki powrotu karetki i nowej linii. Należy zwracać szczególną uwagę na kolejność tych znaków, ponieważ przeciwnym razie komenda zostanie zignorowana. W przypadku korzystania z *Putty* kombinacje klawiszy generujące niezbędne kody to:

– Powrót karetki: *CTRL+M*.
– Znak nowej linii: *CTRL+J*.

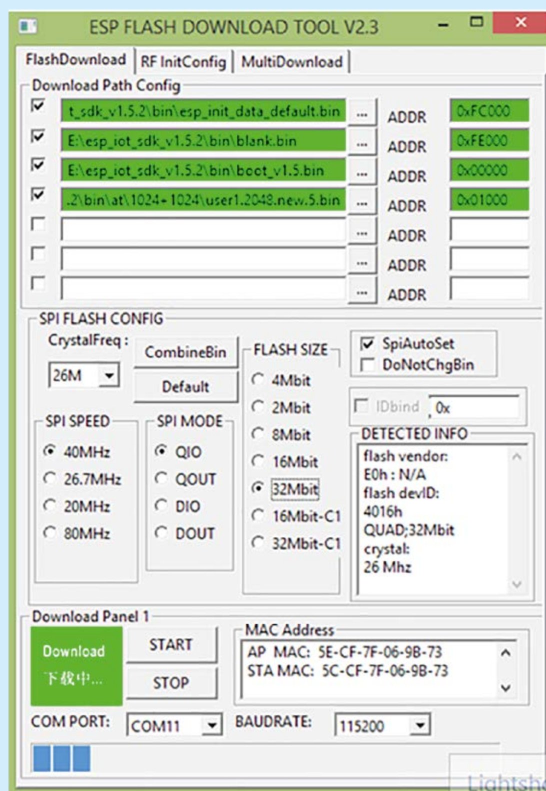
Każde polecenie, oprócz wyniku zawsze zwraca status „OK" lub „ERROR" zależnie od rezultatu jego wykonania. Zależnie od oprogramowania, może też pojawić się komunikat *„Busy p…."*, który informuje o tym, że moduł jest zajęty i nie przyjmie komendy. Listę dostępnych komend znajdziemy na stronie producenta.

Aktualizacja firmware

Do aktualizacji zostanie wykorzystany program *Flash_Download_Tools* następnie należy pobrać oprogramowanie ze strony producenta *bbs.espressif.com* kategoria *SDKs*. W chwili pisania artykułu najnowszą wersją jest



Rysunek 3. Komunikat informujący o numerze wersji modułu ESP-12



Rysunek 4. Okno program służącego do zapisu pamięci Flash modułu ESP-12

1.5.3 – opcja *Non-OS SDK*. Do wgrania oprogramowania na ESP wymagane są 4 pliki:

– esp_init_data_default – adres 0xFC000.
– blank.bin – adres 0xFE000.
– boot_v1.5.bin – adres 0x00000.
– user1.2048.new.5.bin – adres 0x01000.

Należy zauważyć, że nazwy mogą się nieznacznie różnić w zależności od wersji oprogramowania. Przed aktualizacją trzeba przełączyć ESP-12 w stan aktualizacji oprogramowania, wymuszając poziom niski na GPIO0 i restartując moduł. Okno oprogramowania służącego do zapisu pamięci Flash modułu pokazano na **rysunku 4**. Prędkość transmisji pokazana na rysunku może różnić się, zależnie od ustawień oprogramowania. Po aktualizacji i przełączeniu modułu w tryb normalnej pracy (ustawienie GPIO0) i wydaniu komendy *AT+GMR* moduł powinien odpowiedzieć podając numer nowej wersji oprogramowania. Po aktualizacji oprogramowania warto przywrócić nastawy fabryczne, co uchroni nas od potencjalnych problemów. Można to zrobić za pomocą komendy *AT+RESTORE*.

„Hello World"

Czas na najciekawsze – pokazanie możliwości modułu ESP-12 w wersji sterowanej komendami AT. Pierwszym programem będzie klasyczne „Hello World" dla mikrokontrolerów, czyli miganie diodą LED. Do tego celu oprócz modułu ESP-12 wykorzystano również płytkę Arduino UNO R3, ale można zastosować dowolny mikrokontroler wyposażony w interfejs UART lub USART. Środowisko wykorzystane do napisania kodu to Arduino IDE z powodu integracji z modułem ESP-12 i możliwości tworzenia kodu na ten moduł, co przyda się w następnych częściach. Uproszczony schemat ideowy połączeń pokazano na **rysunku 5**.

Oprogramowanie testowe jest nieskomplikowane. Moduł ESP nasłuchuje na porcie 80, kiedy przyjdzie nowy pakiet zasygnalizuje to wysyłając przez UART *+IPD,<ID>,<len>:<data>*, gdzie:

– ID pojawia się, gdy ESP jest ustawione na opcję pozwalającą zawierać więcej niż jedno połączenie (maksymalnie 5) i zawiera ID połączenia.
– Len długość pakietu danych.
– Data dane pakietu.

Teraz trzeba przygotować ESP do współpracy z Arduino, w tym celu wysłana zostanie komenda zmieniająca prędkość transmisji UART. Z racji tego, że Arduino ma tylko jeden sprzętowy interfejs UART, który będzie wykorzystany do komunikacji z komputerem, ESP będzie korzystał z programowej transmisji UART.

Zmiana ustawień transmisji polega na wydaniu polecenia *AT+UART_DEF=9600,8,1,0,0\r\n*, gdzie:

– 9600 jest prędkością transmisji.
– 8 jest liczbą bitów przypadających na jedną paczkę danych.
– 1 bit stopu.
– 0 – bez bitu parzystości.
– 0 – wyłączona kontrola przepływu.

Dopisek *_DEF* oznacza, że ta konfiguracja zostanie zapisana w pamięci nieulotnej Flash w sekcji *user parametr*. Po tym, należy przyłączyć ESP do Arduino – uproszczony schemat ideowy połączeń pokazano na **rysunku 1**. Zamiast pinów 2 i 3 na płycie Arduino

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **77642**, pass: **3220ppmm**

można wykorzystać dowolne inne, ale wtedy należy zmodyfikować oprogramowanie. Z racji tego, że Arduino UNO R3 ma tylko jedno wyprowadzenie 3,3 V, zostawimy pin GPIO0 niepodłączony. Dokumentacja mówi, że ten pin przy normalnej pracy może być albo ustawiony, albo w stanie nieustalonym.

Odpowiednie oprogramowanie sterujące pokazano na **listingu 1**. Po zaprogramowaniu mikrokontrolera, na monitorze szeregowym (ważne: należy wybrać monitor szeregowy, a nie monitor portu szeregowego), znajdującym się w IDE zostaną wyświetlone się informacje o tym, co obecnie wykonuje ESP. Zostanie tam też pokazany adres IP, który należy wpisać w oknie przeglądarki. Po jego wpisaniu, zostanie pokazana tabela z 2 możliwościami *ON – OFF*, którymi steruje się diodą LED.

Programowanie modułu ESP-12

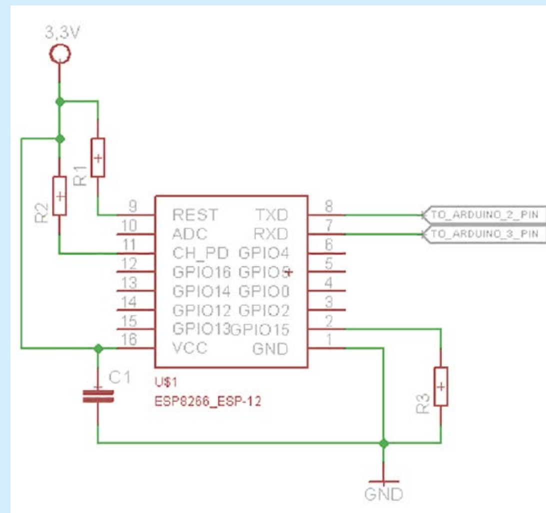
Do programowania modułu ESP-12 wykorzystane zostanie, jak było wspomniane wcześniej, środowisko Arduino IDE z powodu gotowych bibliotek wspomagających pisanie oraz możliwości programowania modułu bezpośrednio z tego środowiska omijając dodatkowe programy. Wszystko, czego obecnie potrzebujemy, to przejściówka *USB* → *UART* oraz zasilanie 3,3 V. Kolejnym krokiem jest dodanie tzw. „płytek" do Menedżera Płytek w Arduino IDE. Dokładną instrukcję można znaleźć pod adresem *www.github.com/esp8266/Arduino*. Teraz można wybrać już opcję *Generic ESP8266 Module* z menu *Narzędzia* → Płytka. Konfiguracja powinna wyglądać następująco:

- Flash Mode: „DIO".
- Flash Frequency: „40MHz".
- Upload Using: „Serial".
- CPU Frequency: „80 MHz".
- Flash Size: „2M (1M SPIFFS)".
- Debug port: „Disabled".
- Debug Level: „Brak".
- Reset Method: „ck".
- Upload Speed: „256000".
- Port: (tutaj należy wybrać port, na którym jest przyłączony moduł).

Upload Speed może się różnić w zależności od wersji IDE i/lub płytki, w przypadku opisywanym w artykule jest to 256000, trzeba doświadczalnie dobrać prędkość by wgrywanie skończyło się sukcesem.

Kod „Hello World" na ESP-12 działa tak samo jak w wersji na Arduino. Dzięki społeczności Arduino, można użyć wielu gotowych już bibliotek przy rozwijaniu oprogramowania dla ESP-12. W tym wypadku zostały użyte biblioteki do połączenia z siecią Wi-Fi oraz realizujące komunikację TCP.

Po wgraniu kodu, wyprowadzenie GPIO0 nie może już być wyzerowane. Po jego ustawieniu lub pozostawieniu go niepodłączonym, moduł wymaga restartu. W konsoli pojawią się informacje o starcie modułu, potem o sukcesie lub niepowodzeniu połączenia do sieci i adres IP. Wpisanie adresu IP, który zostanie wyświetlony przez konsolę na pasku adresowym w oknie przeglądarki WWW, pokaże nam znów taką stronę, jak we wcześniejszym przykładzie. Zauważalną zmianą w tych dwóch przykładach jest szybkość ładowania się stron oraz wykonywania poleceń, na korzyść ESP, co pokazuje dosadnie, jakie możliwości ma ten mały moduł. Mając do wykorzystania w przypadku ESP-12-Q 16 pinów (trzeba zwrócić szczególną uwagę na piny wymagające podłączenia przez rezystory do któregoś z biegunów zasilania) oraz interfejsy komunikacyjne, takie jak I²C, SPI czy UART, przetwornik A/C, można zrealizować sterowanie wieloma urządzeniami z interfejsem w postaci WWW. Interfejsy I²C lub SPI będą świetnie się sprawdzały w przypadku wykorzystania modułu jako czujnika bezprzewodowego. Zwykłe piny GPIO mogą być wykorzystane do sterowania przekaźnikami, co daje możliwość zdalnego sterowania urządzeniami.



Rysunek 5. Połączenie modułu z płytką Arduino UNO R3

Następnymi rzeczami omówionymi w tym cyklu będzie sterowanie bezprzewodowe pinami, z wykorzystaniem modułów przekaźnikowych, odczyt danych z czujników i gromadzenie ich na stronie WWW, a także sterowanie bardziej skomplikowanymi rzeczami jak radio FM.

Moduł może być programowany w języku „C" „C++ (Arduino)" oraz „LUA". W cyklu przeważać będzie język „C++ (Arduino)" z powodu ilości bibliotek dostępnych dla użytkownika, co znacznie usprawnia pisanie softu.

Jakub Kisiel
www.microgeek.eu

Listing 1. Program demonstracyjny – załączanie i gaszenie diody LED

```
#include <SoftwareSerial.h>
#define DEBUG true //W tym stanie Arduino będzie wyświetlać informacje na konsoli
SoftwareSerial esp8266(2, 3); // Pin 2 jest pinem Rx a pin 3 jest Tx

void ESP8266_Init();

void setup()
{
  Serial.begin(9600);
  esp8266.begin(9600); // Ustawianie prędkosci uart. Wartość 9600 jest wartością optymalną
  ESP8266_Init();
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);
```

Listing 1. cd.

```
}

void loop()
{
  if (esp8266.available()) //sprawdzanie czy ESP nadaje
  {
    if (esp8266.find(„+IPD,”)) //jeżeli ESP nadaje to czy wysyła pakiet
    {
      String html;
        delay(1000);
      int connectionId = esp8266.read() - 48; //Pierwszy znak po +IPD, jest nr połączenia w ASCII
      if (esp8266.find(„update.html?LedState=”)) //przeglądarka odeśle metodą GET stan diody LED
      {
        int State = esp8266.read() - 48; //Pierwszy znak po LedState= jest wartością dla diody LED
        if (!State)
        {
          digitalWrite(13, LOW);   // Stan wysoki na pinie 13 spowoduje włączenie diody LED
          Serial.println(„OFF”);
        }
        else if (State)
        {
          Serial.println(„ON”);
          digitalWrite(13, HIGH);   // Stan niski na pinie 13 spowoduje wyłączenie diody LED
        }
        html =  „<html>”
                   „<body>”
                   „<h1> Data Update</h1>”
                   „</body>”
                   „</html>”; //dane które zostaną wysłane po udanej zmianie stanu diody
      }
      else
      {
        html =  „<html>”
                   „<body>”
                   „<form action=\”update.html\” method=\”get\”>”
                   „<fieldset>”
                   „<legend>LED State</legend>”
                   „<input type=\”radio\” name=\”LedState\” value=\”1\”checked=\”checked\”> ON”
                   „<input type=\”radio\” name=\”LedState\” value=\”0\”> OFF<br>”
                   „</fieldset>”
                   „<input type=\”submit\” value=\”Submit\”>”
                   „</form>”
                   „</body>”
                  „</html>”; //pakiet opisujący wygląd strony głównej + nagłówki HTTP
      }
      //przygotowywanie komendy wysyłającej pakiet TCP
      String SendCommand = „AT+CIPSEND=”;
      SendCommand += connectionId;
      SendCommand += „,”;
      SendCommand += html.length();
      SendCommand += „\r\n”;
      sendData(SendCommand, 1000, DEBUG); //wysłanie komendy do ESP
      sendData(html, 1000, DEBUG); //wysłanie danych pakietu TCP
      delay(1000);
      String closeCommand = „AT+CIPCLOSE=”;
      closeCommand += connectionId; //należy dodać ID połączenia do polecenia zamknięcia
      closeCommand += „\r\n”;
      sendData(closeCommand, 3000, DEBUG); //zamknięcie połączenia TCP
    }
  }
}
void ESP8266_Init()
{
  sendData(„AT+RST\r\n”, 1000, DEBUG); //Reset modułu ESP12
  delay(1000);
  sendData(„AT+CWMODE_CUR=3\r\n”, 1000, DEBUG); //Ustawienie modułu w tryb AP/CLIENT
  delay(1000);
  sendData(„AT+CWJAP_CUR=\”SSID\”,\”PASSWORD\”\r\n”, 1000, DEBUG); // Podłączenie do sieci WiFi
  while (1)
  {
    if (esp8266.available() && esp8266.find(„OK”))
       {
      Serial.print(„CONNECTED\n\r”);
      break;
    }
  }
  sendData(„AT+CIPMUX=1\r\n”, 1000, DEBUG); //Pozwolenie na ustawianie wielu połączeń, wymagane dla 
serwera
  sendData(„AT+CIPSERVER=1,80\r\n”, 1000, DEBUG); //Start serwera na porcie 80
  delay(1000);
  sendData(„AT+CIFSR\r\n”, 1000, DEBUG); //Pobranie adresów MAC oraz IP
}
String sendData(String command, const int waitForResponse, boolean debug)
{
  String response = „”;
  esp8266.print(command); //Wysłanie komendy do ESP
  long int time = millis();
  while ( (time + waitForResponse) > millis())
  {
    while (esp8266.available())
    {
      char znak = esp8266.read();
      response += znak;
    }
  }
  if (debug)
  {
    Serial.print(response); //Wyświetlenie odpowiedzi na konsoli
  }
  return response;
}
```

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **77642**, pass: **3220ppmn**

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 77642, pass: 3220ppmm

Listing 2. Program demonstracyjny – sterowanie diodą za pomocą modułu i sieci Wi-Fi

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

//zmienne zawierające dane do połaczenia z siecią WiFi
const char* ssid = „SSID”;
const char* password = „PASSWORD”;

ESP8266WebServer server(80);

//Pakiet TCP z danymi o stronie z wyborem stanu LED
String webpage =  „<html>”
                  „<body>”
                  „<form action=\”/update\” method=\”GET\”>”
                  „<fieldset>”
                  „<legend>LED State</legend>”
                  „<input type=\”radio\” name=\”LedState\” value=\”Led_ON\”> ON”
                  „<input type=\”radio\” name=\”LedState\” value=\”Led_OFF\” checked=\”checked\”>
OFF<br>”
                  „</fieldset>”
                  „<input type=\”submit\” value=\”Submit\”>”
                  „</form>”
                  „</body>”
                  „</html>”;

//Numer pinu podłączonego do diody LED
const int led = 2;

//
void handleRoot() {
  server.send(200, „text/html”, webpage);
}

void handleNotFound() {
  digitalWrite(led, 1);
  String message = „File Not Found\n\n”;
  message += „URI: „;
  message += server.uri();
  message += „\nMethod: „;
  message += (server.method() == HTTP_GET) ? „GET” : „POST”;
  message += „\nArguments: „;
  message += server.args();
  message += „\n”;
  for (uint8_t i = 0; i < server.args(); i++) {
    message += „ „ + server.argName(i) + „: „ + server.arg(i) + „\n”;
  }
  server.send(404, „text/plain”, message);
}

void setup(void) {
  pinMode(led, OUTPUT);
  digitalWrite(led, 0);
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  Serial.println(„”);

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(„.”);
  }
  Serial.println(„”);
  Serial.print(„Connected to „);
  Serial.println(ssid);
  Serial.print(„IP address: „);
  Serial.println(WiFi.localIP());


//Przypisywanie callbackow do odpowiednich adresów URL które przyjdą w pakietach TCP
  server.on(„/”, handleRoot);
//Można napisać ciało funkcji jako argument do funkcji on()
  server.on(„/update”, []() {

//parsowanie pakietu
    if (server.hasArg(„LedState”))
    {
      for (uint8_t i = 0; i < server.args(); i++)
      {
        if(server.argName(i) == „LedState”)
        {
          if(server.arg(i) == „Led_ON”)
          {
              digitalWrite(led, 0);
                server.send(200, „text/plain”, „Led status: „+server.arg(i));
                break;
          }
          if(server.arg(i) == „Led_OFF”)
          {
              digitalWrite(led, 1);
              server.send(200, „text/plain”, „Led status: „+server.arg(i));
              break;
          }
        }
      }
    }
  });
//wysyła stronę 404
  server.onNotFound(handleNotFound);

  server.begin();
  Serial.println(„HTTP server started”);
}

void loop(void) {
  //Wątek zajmujący się obsługą stosu TCP
  server.handleClient();
}
```