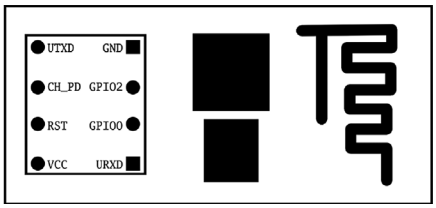
Moduł ESP01 pracujący jako sterownik z 2 wyjściami i 1 wejściem

ESP01 to miniaturowy moduł Wi-Fi zbudowany w oparciu o układ ESP8266. W tej najprostszej wersji do dyspozycji użytkownika są 2 wyprowadzenia I/O oraz port komunikacyjny UART. Nawet jednak z tak ograniczonymi zasobami sprzętowymi, można wykorzystać moduł do zbudowania mikro-serwera z 2 wyjściami i 1 wejściem, ze sterowaniem za pośrednictwem przeglądarki internetowej. Taki serwer może generować dynamiczne strony HTML wysyłane do wyświetlenia przez przeglądarkę. Wszystko sprowadza się do odpowiedniego oprogramowania modułu.



Rysunek 1. Schematyczny wygląd modułu ESP01

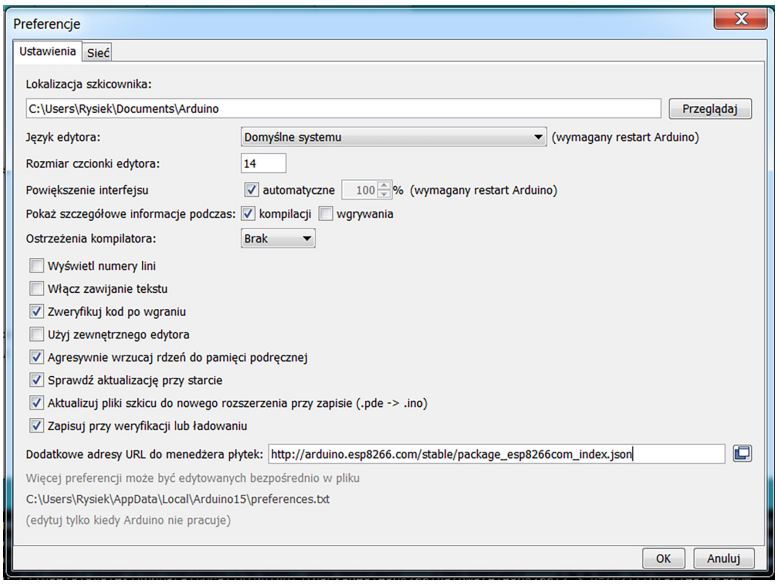
Na **rysunku 1** pokazano schematyczny wygląd modułu Wi-Fi ESP01. Czarnymi prostokątami oznaczono położenie układu ESP8266 i szeregowej pamięci Flash, na prawo jest wytrawiona na płytce ścieżka anteny radiowej pracującej w paśmie 2,4 GHz. Po lewej zamontowano złącze ze wszystkimi dostępnymi sygnałami:

- VCC – zasilanie +3,3 V,
- GND – masa,
- UTXD – wyjście TxD interfejsu szeregowego UART o poziomach 0/3,3 V.
- URXD – wejście RxD interfejsu szeregowego UART o poziomach 0/3,3 V.
- CH_PD – wprowadzanie modułu w tryb obniżonego poboru mocy; przy normalnej pracy należy zasilić wejście napięciem 3,3 V (ustawić poziom wysoki).
- RST – sygnał zerowania modułu, poziom aktywny 0 V; w czasie normalnej pracy należy zasilić wejście napięciem 3,3 V (ustawić poziom wysoki).
- GPIO0 – linia I/O; jeżeli w czasie restartu wejście będzie zwierane do masy, układ ESP8266 wejdzie w tryb programowania.
- GPIO2 – linia I/O, w czasie restartu nie powinna być zerowana.

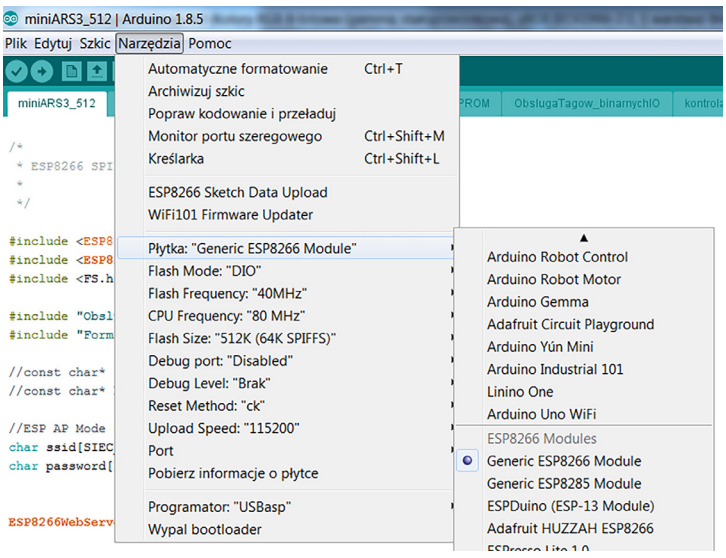
ESP01w trybie interpretacji rozkazów AT

Nowo zakupiony moduł jest zwykle AT zaprogramowany do obsługi (interpretacji) poleceń AT. Są to polecenia tekstowe, rozpoczynające się od przedrostka „AT”, pozwalające na łatwe i wszechstrone sterowanie modułem. Żeby przekonać się czy moduł znajduje się w trybie AT należy:

- Zasilić moduł napięciem +3,3 V.



Rysunek 2. Ścieżka do dodatkowych bibliotek



Rysunek 3. Wybór płytki ESP01 w menu narzędzia

- Do wyprowadzeń URXD, UTXD dołączyć np. konwerter USB-UART. **Uwaga!** Sygnały na wyprowadzeniach UART konwertera muszą pracować z poziomami 0/3,3 V. Podanie na wyprowadzenia modułu ESP01 sygnałów o wyższych poziomach, np. 5 V, grozi jego zniszczeniem.
- Korzystając z dowolnego programu – terminala, należy otworzyć port komputera, do którego dołączono konwerter USB, w trybie: 8 bitów danych, bez parzystości, z szybkością transmisji 115200 bps i wysłać komendę tekstową „AT+GMR" zakończoną znakami CR+LF (szesnastkowo 0x0D i 0x0A).
- W odpowiedzi moduł powinien odesłać tekst z informacjami o wersji interpretera AT i wersji SDK, a na końcu tekst „OK".

Komendy AT są łatwe w użyciu, ale wymagają zewnętrznego sterowania modułu przez komputer lub dodatkowy kontroler. Aby jednak w pełni wykorzystać możliwości modułu, należy wykorzystać API producenta i bezpośrednio oprogramować zamontowany na płytce układ ESP8266.

Narzędzia do programowania ESP8266

Procesor ESP8266 jest wydajnym mikrokontrolerem, zintegrowanym z układami radiowymi do komunikacji w paśmie 2,4 GHz, z wykorzystaniem standardu Wi-Fi. Producent układu, firma Espresiff Systems, opracowała dla niego API *(*Application Programming Interface*)*, czyli zestaw niskopoziomowych procedur pozwalających na korzystanie z możliwości układu. Dostęp do API jest możliwy z poziomu aplikacji pisanych w języku C lub z wykorzystaniem dodatkowo ułatwiających pracę narzędzi, takich jak język LUA i jego środowisko lub środowisko Arduino.

Ze względu na popularność i łatwość użycia, opis oprogramowania mikro-serwera oprzemy właśnie na Arduino. Do tworzenia i zapisu oprogramowania do ESP01, potrzeba będzie trochę dodatkowych narzędzi. W kolejnych punktach opiszę co i jak zainstalować:

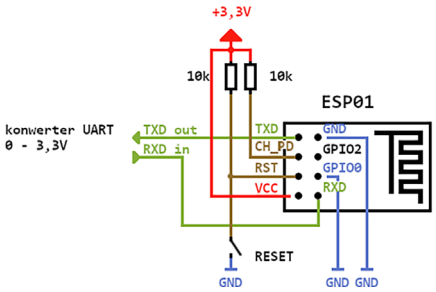
- Plik instalacyjny Arduino IDE można pobrać spod adresu ***http://bit.ly/2ItlykE***.
- W Arduino IDE należy podać ścieżkę do zainstalowania wymaganych bibliotek płytki ESP8266. Korzystając z menu *Plik* → *Preferencje*, w polu *Dodatkowe adresy URL do menadżera płytek* wpisujemy ***http://bit.ly/2G476lA***, jak pokazano na **rysunku 2**.
- W polach *Narzędzia* wybieramy płytkę modułu ESP01, jak pokazano na **rysunku 3**:
  - Płytka: „Generic ESP8266 Module"
  - Flash Mode: „DIO"
  - Flash Frequency „40MHz"
  - CPU Frequency: „80MHz"
  - Flash Size „512K (64K SPIFFS)"
  - Upload speed „115200"
- W kolejnym kroku instalujemy narzędzie do zapisu do systemu plików w pamięci Flash układu ESP8266. Pobieramy plik instalacyjny spod adresu ***http://bit.ly/2Pg80KM***.
- Rozpakowujemy pobrany plik w katalogu Arduino IDE (*C:\Program Files (x86)\Arduino\tools*).
- Otwieramy Arduino IDE. W zakładce *Narzędzia* powinna zostać pokazana dodatkowa pozycja *ESP8266 Sketch Data Upload*.

Aby moc wgrać własne oprogramowanie do pamięci Flash na płytce ESP01, jej wyprowadzenia powinny być ustawione, jak pokazano na **rysunku 4**. W szczególności, w momencie restartu (dołączania zasilania do modułu lub zwierania do masy wyprowadzenia RST), wyprowadzenie GPIO0 powinno być połączone z masą. Do wyprowadzeń TXD i RXD należy dołączyć konwerter UART o poziomach sygnałów 0/3,3 V.

Teraz mamy wszystko co potrzeba do tworzenia oprogramowania mikro-serwera serwera i jego zapisu do pamięci programu na płytce ESP01.

ESP01 obsługa portów IO

Mikro-serwer ma sterować przełączaniem stanu 2 portów wyjściowych i odczytywać stan 1 portu wejściowego. Praca z portami IO, to najprostsza do wyjaśnienia część oprogramowania więc od niej zaczniemy. Składnia



Rysunek 4. Zapis pamięci ESP01 – konfiguracja wyprowadzeń

Listing 1. Przykładowy program

```
int relPin =2;  //GPIO2 do tego IO jest dołączony przekaźnik
int ledPin =0;  //GPIO0 do tego IO jest dołączona dioda LED
int we1Pin =3;  //GPIO3(RXD) do tego IO jest dołączony przycisk WE1

// funkcja setap uruchamia się raz przy uruchomieniu
void setup()
{
    Serial.begin(115200);
    Serial.println(„\n inicjacja\n");

    pinMode(relPin, OUTPUT); // ustawiamy IO jako wyjście
    pinMode(ledPin, OUTPUT);
    pinMode(we1Pin, INPUT_PULLUP);//ustawiamy IO jako wejście
}

// funkcja loop uruchamia się w nieskończonej pętli
void loop()
{
    int stanWE1 =digitalRead(we1Pin);//odczyt stanu IO we1Pin
    if (stanWE1 ==LOW)
    {
        digitalWrite(relPin, LOW);   //przepisanie stanu niskiego na wyjścia
        digitalWrite(ledPin, LOW);
    }
    else
    {
        digitalWrite(relPin, HIGH);   //przepisanie stanu wysokiego na wyjścia
        digitalWrite(ledPin, HIGH);
    }
}
```

Listing 2. Konfigurowanie ESP01

```
#include <ESP8266WiFi.h>
const char* ssid     =”nazwa”; // Tu wpisz nazwę swojego wifi
const char* password =”hasło”; // Tu wpisz hasło do swojego wifi
//dołączenie do sieci (rutera)
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED)
{
    delay(500);
    Serial.print(„.”);
}
Serial.print(„Web Server tryb Stacji IP: „);
Serial.println(WiFi.localIP());
```

Listing 3. Zmiana adresu IP

```
#include <ESP8266WiFi.h>

IPAddress local_IP(192,168,4,1);  //adres IP
IPAddress gateway(192,168,4,9);  //opcja
IPAddress subnet(255,255,255,0);  //opcja

Serial.print(„Tryb soft-AP ... „);
Serial.println(WiFi.softAPConfig(local_IP, gateway, subnet) ? „Ready” : „Failed!”);
//Inicjacja trybu AP
WiFi.softAP(„ARS3_miniAP”);  //nadanie nazwy soft-AP, dostęp bez hasła
IPAddress myIP = WiFi.softAPIP();
Serial.print(„Web Server ARS3_miniAP IP:”);
Serial.println(myIP);
```

Listing 4. Procedura zapisu do pamięci nazwy sieci

```
//zapis do EEPROM nazwy sieci SSID
#include <EEPROM.h> //plik nagłówkowy procedur EEPROM
#define SIEC_NAZWA_ADR          0          //adres w pamięci EEPROM
#define SIEC_NAZWA_ROZMIAR_MAX  32

bool EEPROM_Zapis_SSID(char *p_buf_SSID)
{
    int x, ile;
    ile =strlen(p_buf_SSID);
    if (ile>=SIEC_NAZWA_ROZMIAR_MAX-1) return false;
    EEPROM.begin(SIEC_NAZWA_ROZMIAR_MAX);
    delay(10);
    for (x=SIEC_NAZWA_ADR;x<SIEC_NAZWA_ADR+ile;x++)
    {
        EEPROM.write(x, *p_buf_SSID);
        p_buf_SSID++;
    }
    EEPROM.write(x, '\0');
    EEPROM.commit();
    EEPROM.end();
    return true;
}
```

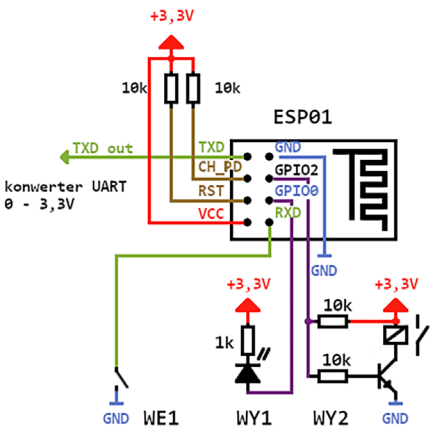
Listing 5. Odczyt identyfikatora sieci

```
//odczyt z EEPROM nazwy sieci SSID
void EEPROM_Odczyt_SSID(char *p_buf_SSID)
{
    int x;
    EEPROM.begin(SIEC_NAZWA_ROZMIAR_MAX);
    delay(10);
    for (x=SIEC_NAZWA_ADR;x<SIEC_NAZWA_ADR+SIEC_NAZWA_ROZMIAR_MAX-1;x++)
    {
        *p_buf_SSID =EEPROM.read(x);
        if (*p_buf_SSID =='\0')
        {
            p_buf_SSID++;
            break;
        }
        p_buf_SSID++;
    }
    *p_buf_SSID = '\0';
    EEPROM.end();
}
```

rozkazów związanych z manipulacją portami IO jest arduinowa:

- *pinMode(numer_pinu, OUTPUT)* – konfigurowanie I/O jako wyjścia.
- *pinMode(numer_pinu, INPUT_PULLUP)* – konfigurowanie I/O jako wejścia z wewnętrznym podciąganiem do +3,3 V.
- *digitalWrite(numer_pinu, LOW)* – wyzerowanie wyjścia I/O.
- *digitalWrite(numer_pinu, HIGH)* – ustawianie wyjścia I/O.
- *digitalRead(numer_pinu)* – odczyt poziomu na wejściu I/O.

Na **listingu 1** pokazano przykładowy program. W pętli badany jest stan WE1 i przepisywany na 2 wyjścia WY1, WY2. Po skompilowaniu należy zapisać program wynikowy do ESP01 w konfiguracji takiej, jak na **rysunku 4**. Do testowania i pracy należy połączyć wyprowadzenia ESP01 tak jak na **rysunku 5**. Będzie to konfiguracja mikro-serwera z obwodami wykonawczymi. Dla przykładu pokazano, jak do GPIO0, GPIO2 pracujących jako wyjścia WY1, WY2 dołączyć diodę LED i przekaźnik. Jako wejście WE1 użyto wyprowadzenia RXD.



Rysunek 5. Połączenie wyprowadzeń ESP01

ESP01 jako softAP i Stacja

W najprostszym wypadku komunikacja z mikro-serwerem odbywa się w jednym z 2 trybów: mikro-serwer pracuje jako Stacja lub samodzielny punkt dostępowy AP.

Pracując jako Stacja serwer najpierw musi zostać dołączony do lokalnej sieci Wi-Fi za pośrednictwem rutera. W procedurze *WiFi.begin(ssid, password)* należy podać nazwę sieci (ssid) i jeżeli jest używane hasło (password). W razie akceptacji, mikro-serwer otrzymuje własny adres IP. Od tej chwili w obrębie sieci jest możliwe połączenie z serwerem używając nadanego adresu IP. Do prawidłowego działania procedura wymaga dołączenia pliku nagłówkowego *ESP8266WiFi.h*. Nadany adres IP jest dodatkowo wysyłany portem szeregowym TXD out.

Jako AP, należy nadać mikro-serwerowi unikatową nazwę za pomocą której będzie identyfikowany wśród innych sieci Wi-Fi. Domyślnym numerem IP jest 192.169.4.1, ale może być zmieniony.

Jeżeli chcemy zachować kontrolę nad wyborem trybu pracy mikro-serwera, obie procedury trzeba połączyć. Wybór trybu pracy można uzależnić np. od badania bezpośrednio po resecie stanu pinu wejściowego WE1 (rys. 5). Zależnie od poziomu podanego na WE1 realizowana będzie procedura ustawiania mikro-serwera do pracy jako Stacja lub AP.

Pamięci EEPROM i SPIFFS

Niektóre typy danych, takie jak nazwa sieci czy hasło powinny być zapamiętywane w pamięci nieulotnej i odtwarzane po załączeniu zasilania mikro-serwera. W module ESP01 użytkownik może, za pośrednictwem bibliotek, mieć dostęp do 2 typów pamięci nieulotnej: EEPROM i SPIFFS. Obie są symulowane w pamięci Flash jako wydzielone obszary obok pamięci programu.

Pojemność symulowanej pamięci EEPROM wynosi 4096 bajtów. Dostęp do danych odbywa się za pomocą bufora automatycznie tworzonego w pamięci RAM, którego pojemność (od 4 do 4096 bajtów) deklaruje

Listing 6. Zapisanie przykładowego pliku „plik.txt"

```
#include <FS.h>

Serial.begin(115200);
//inicjacja systemu plików
if(SPIFFS.begin()) Serial.println(„SPIFFS zainicjowany....ok");
else
{
    Serial.println(„SPIFFS błąd inicjacji");
    return;
}
//tworzenie pliku
//w=otwarcie pliku w trybie do zapisu
File f = SPIFFS.open(„/plik.txt", „w");
if (!f) Serial.println(„błąd otwarcia pliku");
else
{
    f.print(„Przykładowy tekst zapisywany do pliku");
    f.close(); //zamykanie pliku
}
```

Listing 7. Odczytanie zawartości pliku

```
int i;
SPIFFS.begin()
//otwarcie pliku w trybie do odczytu
File f = SPIFFS.open(„/plik.txt", „r");
if (!f) Serial.println(„błąd otwarcia pliku");
else
{
    //odczyt zawartości pliku i wysłanie portem UART
    for(i=0;i<f.size();i++) Serial.print((char)f.read());
    f.close(); //zamykanie pliku
}
```

się poleceniem *EEPROM.begin(pojemność)*. Po umieszczeniu danych w buforze komenda *EEPROM.commit()* przepisuje dane do obszaru pamięci EEPROM. Przykładową procedurę zapisu do pamięci nazwy sieci (SSID) pokazano na **listingu 4**, natomiast procedurą odczytującą identyfikator SSID sieci na **listingu 5**.

SPIFFS to system plików zapisywanych w wydzielonym obszarze pamięci Flash. Jego wielkość zależy od modułu ESP i zamontowanej pamięci SPI. Dla ESP01 najbezpieczniej przyjąć rozmiar 64 kB. Ponieważ dane mogą być przechowywane w formie plików, jest to dobre miejsce na źródła stron HTML i grafik, które nasz mikro-serwer będzie udostępniał do wyświetlenia przez przeglądarkę.

Aby mieć dostęp do systemu plików, należy zainstalować bibliotekę procedur jego obsługi. Do programu należy dodać plik nagłówkowy FS.h. Na **listingu 6** pokazano, w jaki sposób można zapisać do systemu SPIFFS przykładowy plik „plik.txt". Najpierw inicjowany jest dostęp do systemu plików. W następnej kolejności jest tworzony w systemie SPIFFS sam plik, zapisywana jego zawartość, a na koniec dostęp do pliku jest zamykany. Na **listingu 7** pokazano, jak uzyskać dostęp do pliku zapisanego w systemie SPIFFS i odczytać jego zawartość.

W mikro-serwerze zawartość stron HTML do wyświetlenia powinna znajdować się w systemie plików SPIFFS przed uruchomieniem serwera. Do „hurtowego" zapisu danych można posłużyć się narzędziem *ESP8266 Sketch Data Upload*. Opis jego instalacji podano na początku.

Posługiwanie się narzędziem przebiega następująco:

- W katalogu projektu arduinowego, w którym znajduje się plik z rozszerzeniem .ino, należy utworzyć podkatalog o nazwie *data*.
- Do utworzonego katalogu należy przepisać pliki, które mają zostać umieszczone w systemie plików SPIFFS.
- Płytkę ESP01 należy połączyć jak do zapisu do pamięci Flash (konfiguracja z rys. 4).
- Na pulpicie projektu należy wybrać opcję *Narzędzia* → *ESP8266 Sketch Data Upload*.
- Zawartość katalogu *data* zostanie zapisana w systemie plików SPIFFS.

Od tej chwili pliki są dostępne do odczytu. Można je oczywiście także usuwać i zmieniać.

Procedury Web Serwera

Procedury Serwera odpowiedzialne za obsługę zapytań z przeglądarki internetowej, najwygodniej oprzeć o specjalnie w tym celu stworzoną bibliotekę *ESP8266WebServer*. Biblioteka zawiera mechanizm wywołań zwrotnych obsługujących każde zapytanie

Listing 8. Przykład obsługi zapytania przez serwer

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
const char *ssid="nazwa sieci";
const char *password="hasło sieci";
ESP8266WebServer webServer(80);

void indexHandler()
{
    webServer.send(200, „text/plain", „Czas uruchomienia serwera: „ +
String(millis())+"ms");
}

void notFoundHandler()
{
    webServer.send(404, „text/plain", „Nie ma takiej strony");
}

void setup()
{
    Serial.begin(115200);
    Serial.println(„");
    WiFi.begin(ssid, password);
    //Wait for connection
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(„.");
    }
    Serial.println(„");
    Serial.print(„SSID „);
    Serial.println(ssid);
    Serial.print(„IP address: „);
    Serial.println(WiFi.localIP());
    webServer.on(„/", indexHandler);
    webServer.onNotFound(notFoundHandler);
    webServer.begin();
    Serial.println(„Start Web Servera");
}

void loop()
{
    webServer.handleClient();
}
```

Listing 9. Zmodyfikowana, rozszerzona obsługa zapytań

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <FS.h>
const char *ssid="nazwa sieci";
const char *password="hasło sieci";
ESP8266WebServer webServer(80);

void indexHandler()
{
    webServer.sendHeader(„Location", „/index.html",true);   //przekierowanie na stronę
index.html
    webServer.send(302, „text/plane","");
}

void notFoundHandler()
{
    webServer.send(404, „text/plain", „Nie ma takiej strony");
}

bool loadFromSpiffs(String path)
{
    String dataType = „text/plain";
    if(path.endsWith(„/")) path += „index.html";
    if(path.endsWith(„.src")) path = path.substring(0, path.lastIndexOf(„."));
    else if(path.endsWith(„.html")) dataType = „text/html";
        else if(path.endsWith(„.htm")) dataType = „text/html";
            else if(path.endsWith(„.png")) dataType = „image/png";
                else if(path.endsWith(„.gif")) dataType = „image/gif";
                    else if(path.endsWith(„.jpg")) dataType = „image/jpeg";
    File dataFile = SPIFFS.open(path.c_str(), „r");
    if (!dataFile) return false;//w systemie SPIFFS brak pliku o takiej nazwie
    if (webServer.streamFile(dataFile, dataType) != dataFile.size())
    {
    }
    dataFile.close();
    return true;
}

void setup() {
    Serial.begin(115200);
    Serial.println(„");
    WiFi.begin(ssid, password);
    //Wait for connection
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(„.");
    }
    SPIFFS.begin();
    Serial.println(„");
    Serial.print(„SSID „);
    Serial.println(ssid);
    Serial.print(„IP address: „);
    Serial.println(WiFi.localIP());
    webServer.on(„/", indexHandler);
    webServer.onNotFound(notFoundHandler);
    webServer.begin();
    Serial.println(„Start Web Servera");
}

void loop() {
    webServer.handleClient();
}
```

z przeglądarki. Najpierw w programie należy napisać procedurę obsługi konkretnego zapytania. Potem należy zadeklarować je jako wywołanie zwrotne (callback). Na przykład może to wyglądać tak:

- *void indexHandler()* {} – procedura obsługi zapytania, gdy w pasku przeglądarki wpiszemy adres IP mikro-serwera, ewentualnie zakończony ukośnikiem '/'.
- *void notFoundHandler()* {} – procedura obsługi pozostałych zapytań.
- *ESP8266WebServer.on(„/", indexHandler);* – deklaracja wywołania procedury *indexHandler()* po odbiorze z przeglądarki zapytania będącego adresem IP mikro-serwera.
- *ESP8266WebServer.onNotFound(notFoundHandler);* – deklaracja wywołania procedury *notFoundHandler()* będących obsługą pozostałych zapytań.

Deklaracji *ESP8266WebServer.on* może być wiele, natomiast *ESP8266WebServer.onNotFound* tylko jedna. Dodatkowo, w pętli głównej jest wywoływana funkcja *ESP8266WebServer. handleClient sterująca* obsługą zapytań. Na **listingu 8** pokazano kompletny przykład serwera obsługującego w taki sposób zapytania z przeglądarki. Serwer w opisany wcześniej sposób skonfigurowany jest do pracy jako Stacja.

Dla serwera, który wyświetla dane zapisane w systemie plików SPIFFS, oprogramowanie musi być trochę zmienione. Przykład pokazano został na **listingu 9**. Zmiany w procedurze *indexHandler()* polegają na tym, że jest wysyłane do przeglądarki żądanie przekierowania zapytania o zapisany w systemie SPIFFS plik „index.html". Wprowadzono także obsługę przesyłania plików graficznych, z rozszerzeniami .png .gif .jpg, które razem ze źródłami stron HTML mogą być zapisane w systemie plików SPIFFS.

Mikro-serwer dynamiczne strony HTML

Opisany wyżej sposób obsługi zapytań przeglądarki sprawdzi się w przypadku wyświetlania statycznych stron HTML. Jeżeli jednak chcielibyśmy, żeby mikro-serwer wyświetlał rzeczywisty, zmieniający się stan wejść i wyjść, potrzebna będzie obsługa i tworzenie dynamicznych stron HTML. W tym celu potrzebny będzie mechanizm programowy po stronie mikro-serwera jak i specjalnie przygotowane strony HTML. Na **listingu 10** pokazano kod źródłowy strony HTML, której wygląd będzie mógł być modyfikowany w zależności od stanu wejść i wyjść mikro-serwera.

Podczas normalnego wyświetlania w przeglądarce źródła takiej strony, użytkownik zobaczyłby tylko nagłówek. Napisy informujące o stanie wyjść i wejść są opatrzone

Listing 10. Program źródłowy strony

```
<!DOCTYPE html>
<html>
<head>
<title>Mikro-Serwer ESP8266</title>
<meta name="description" content="Mikro-Serwer ESP8266">
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<style>
p {font-size: 30px;}
</style>
</head>

<body style="background-color:#EAEAEA;">
<div style="text-align: center"><h1>Mikro-Serwer ESP8266</h1>
    <p><b>sterowanie 2 wyjściami<br>
            odczyt 1 wejścia</b></p>

<!--#tag_w_out1=1 <p>WYJŚCIE1 =ON</p>-->
<!--#tag_w_out1=0 <p>WYJŚCIE1 =OFF</p>-->

<!--#tag_w_out2=1 <p>WYJŚCIE2 =ON</p>-->
<!--#tag_w_out2=0 <p>WYJŚCIE2 =OFF</p>-->

<!--#tag_w_in1=0 <p>WEJŚCIE1  =ON</p>-->
<!--#tag_w_in1=1 <p>WEJŚCIE1  =OFF</p>-→

</div>

</body>
</html>
```

Rysunek 6. Zrzut ekranu Mikro-serwera

komentarzem i nie będą wyświetlane. Jednak przystosowane do tego oprogramowanie mikro-serwera, będzie w stanie tak przekształcić źródło strony, żeby zależnie od stanu wyjść i wejścia wyświetlić w przeglądarce stosowny napis. W tym celu, najpierw strona jest pobierana z systemu SPIFFS i zapisywana w buforze w pamięci RAM modułu ESP8266. Następnie identyfikowana jest pozycja znaków „<!--#". Potem analizowany jest tag znajdujący się za tymi znakami. Procedura rozpoznaje następujące tagi:

- *tag_w_out1* – tag wyjścia1.
- *tag_w_out2* – tag wyjścia2.
- *tag_w_in1* – tag wejścia1.

Następnie jest badany rzeczywisty poziom logiczny na wyjściu, odpowiadający zidentyfikowanemu tagowi. Jeżeli jest to poziom wysoki, a za tagiem znajduje się przypisanie „=1", procedura usunie znaki komentarza i etykiety „<!--#tag_w_out1=1", „-->" i przeglądarka wyświetli napis „WYJŚCIE1 =ON". Jeżeli stan wyjścia nie odpowiada przypisaniu, znaki komentarza nie zostaną usunięte. Tak samo procedura zadziała z przypadku tagu oznaczającego wyjście 2 jak i wejście 1. Procedura jest wywoływana z wywołania zwrotnego (callback) obsługującego zapytanie z przeglądarki. Żeby wyróżnić potrzebujące obsługi przez procedurę strony dynamiczne, rozszerzenie nazwy ich plików zmieniono na .shtml.

Ustawianie poziomu na wyjściu też odbywa się na zasadzie tagów i przypisania stanu jaki na wyjściu ma być ustawiony. Tagi należy wpisać za adresem IP modułu w pasku przeglądarki. I tak dla wyjścia 1, które ma zostać załączone w pasku przeglądarki powinno się wpisać *http://192.168.0.185/index.shtml?tag_f_out1=1*, a dla wyjścia 2 polecenie wyłączenia będzie wyglądało *http://192.168.0.185/index.shtml?tag_f_out2=0*. Oczywiście, należy podać rzeczywisty adres IP przydzielony modułowi ESP01.

Tak sformułowane zapytanie przeglądarki trafi do procedury wywołania zwrotnego (callback), oprogramowania modułu. Dwie komendy API ESP8266WebServer.argName() i ESP8266WebServer.arg() biblioteki ESP8266WebServer, wyłuskają z przesłanego zapytania przeglądarki tag i przypisaną wartość. Procedura mikro-serwera powinna potrafić zidentyfikować i połączyć przesłane tagi z odpowiednimi wyjściami i zależnie od przypisania ustawić na odpowiednim wyjściu stan wysoki albo niski.

Na **rysunku 6** pokazano zrzut ekranu opisanego powyżej Mikro-serwera działającego na module ESP01. Pliki projektu o nazwie *Demo6_serwer_strony_dynamiczne* dla platformy ARDUINO został dodany do tego artykułu.

Ryszard Szymaniak, EP