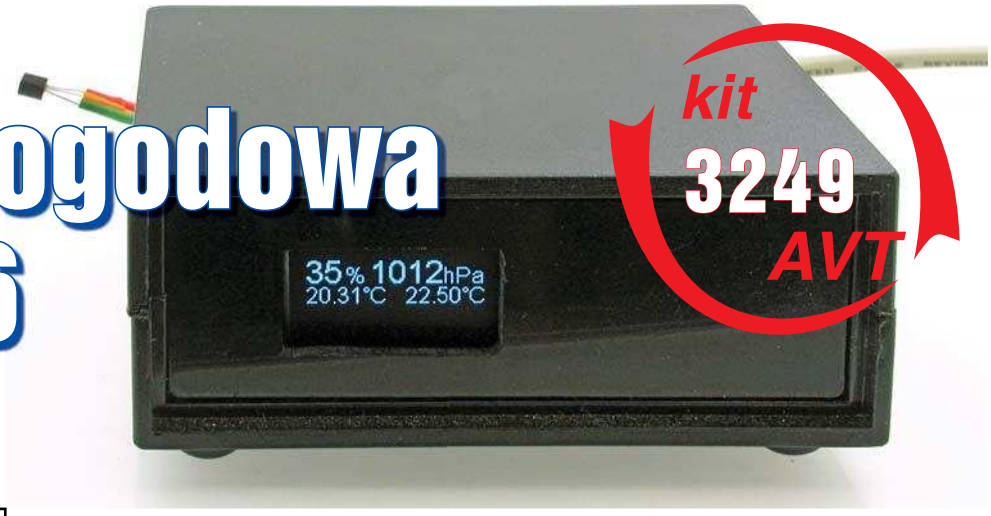
Stacja pogodowa ESP8266 z OLED



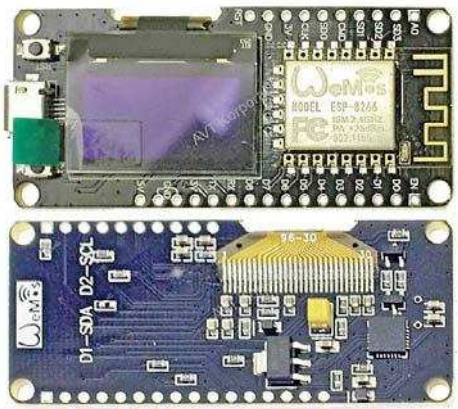
kit 3249 AVT

Popularne i tanie moduły Wi-Fi z ESP8266 pozwalają łatwo i szybko budować stosunkowo skomplikowane konstrukcje. ESP8266 poza podstawową pierwotną funkcją mostka Wi-Fi UART, po wgraniu własnego programu do modułu, może realizować przeróżne funkcje, np. bezprzewodowych czujników czy sterowników. Możliwość programowania z poziomu ArduinoIDE i bogactwo przykładów w Internecie ułatwiają budowę własnych urządzeń.

W artykule przedstawiono konstrukcję prostej stacji pogodowej mierzącej ciśnienie, wilgotność i temperaturę z kilku czujników. Wyniki są prezentowane na wyświetlaczu OLED oraz wysyłane na serwer *thingspeak*. Trwający kurs Arduino i zakończony C niewątpliwie ułatwią zrozumienie zasady działania programu stacji pogodowej.

Zadaniem stacji pogodowej jest pomiar temperatury przez maksymalnie 10 czujników temperatury, ciśnienia i wilgotności. Wyniki na bieżąco są prezentowane na monochromatycznym wyświetlaczu OLED 128x64. Wyświetlacz jest zamontowany na płytce modułu ESP-12F (**fotografia 1**), który można nabyć w sklepie AVT, kod towaru: ARD-9534 i link – *https://sklep.avt.pl/modul-nodemcu-wifi-esp8266-esp-12f-z-wyswietlaczem-oled-0-96-cala-arduino.html, w skrócie: https://bit.ly/2HSIriy.*

Fot. 1

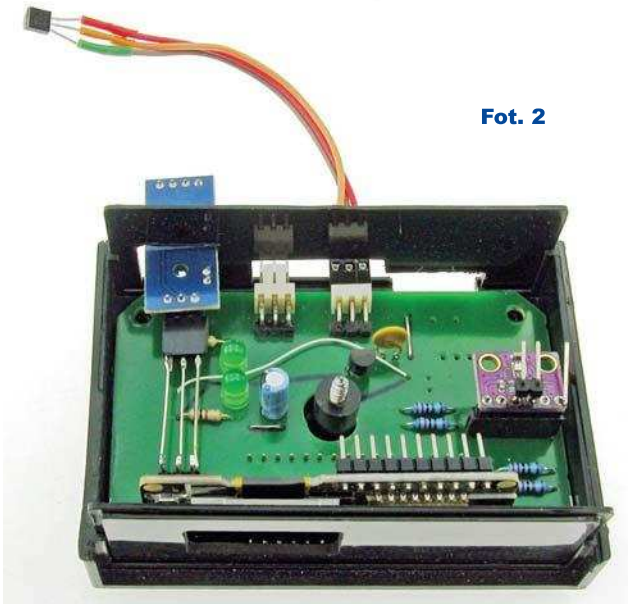


Ciśnienie mierzy czujnik BMP280, kod towaru ARD-7206, link – *https://sklep.avt.pl/product/search?query=bmp280, w skrócie: https://bit.ly/2HRLPtX.*

Wilgotność mierzy moduł z czujnikiem DHT11 ARD-8608, link – *https://sklep.avt.pl/czujnik-wilgotnosci-i-temperatury-dht11-modul-arduino.html* w skrócie: *https://bit.ly/2EOuCQ8.*

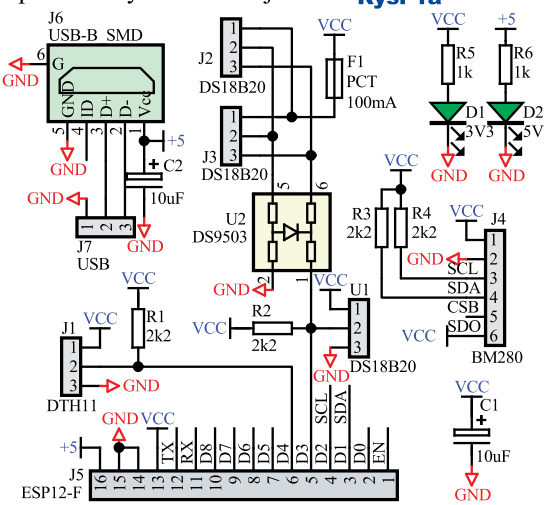
Temperaturę mierzy popularny termometr cyfrowy DS18B20, link – *https://sklep.avt.pl/product/search?query=ds18b20 w skrócie: https://bit.ly/2WJBXuB.*

Ze względu na to, że moduły te były opisywane na łamach EdW, nie ma sensu powielać tych informacji.



Fot. 2

Rys. 1a

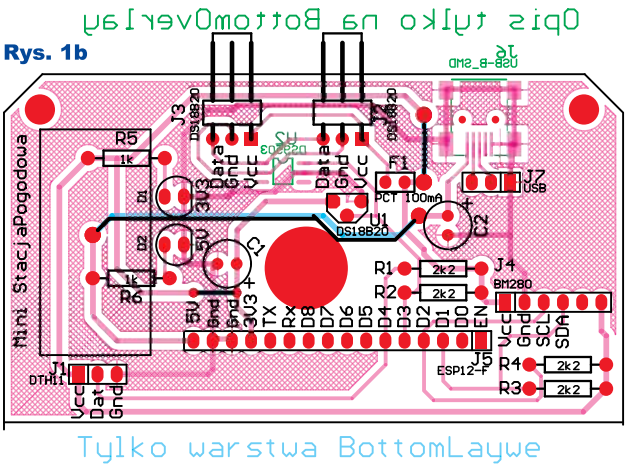


Opis układu

Schemat stacji pogodowej pokazany jest na **rysunku 1a**. Zasilanie układu czerpane jest z dowolnego zasilacza z wtykiem mini USB. Złącze J5 łączy płytę bazową z modułem ESP8266. Na płycie bazowej znajdują się złącza J2, J3 dla zewnętrznych termometrów DS18B20. Na płytce jest miejsce dla jednego termometru (U1) oraz barometru BM280 (J4) i higrometru DH11 (J1). Układ U2 zabezpiecza mikrokontroler przed uszkodzeniem przez ładunki ESD i zakłócenia mogące pojawić się w przewodach łączących czujnik z mikrokontrolerem. Niestety ten element jest często pomijany w amatorskich konstrukcjach albo zastępowany przez diodę Zenera, która nie jest w stanie zapewnić dobrej ochrony, a wprowadza niepotrzebnie dużą pojemność.

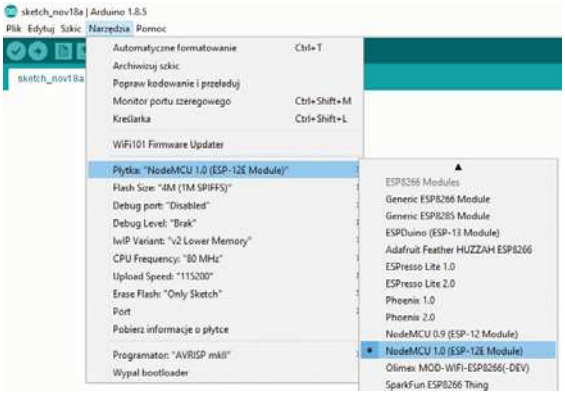
Montaż i uruchomienie

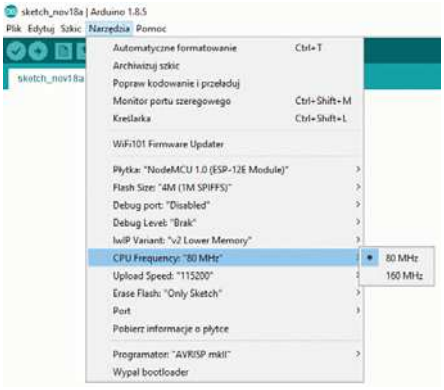
Rys. 1b

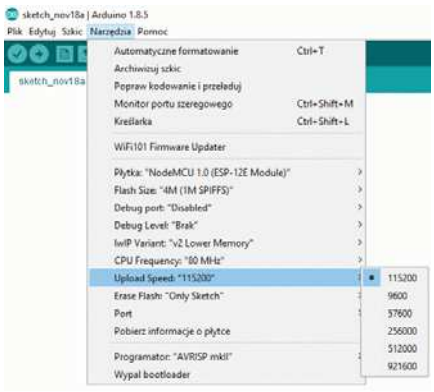


Podstawą układu jest gotowy moduł zawierający procesor i wyświetlacz (**fotografia 2**). Montaż całości możliwy jest więc na wiele sposobów. Można wykorzystać płytkę drukowaną, pokazaną na **rysunku 1b**, która będzie bazą dla pozostałych elementów i modułów (dokumentacja dostępna w Elportalu). Higrometr powinien wystawać poza obudowę, w przeciwnym wypadku wyniki pomiaru będą zaniżone. Temperatura wskazywana przez termometr U1 jest zawyżona o jakieś dwa stopnie gdy czujnik znajduje się w obudowie. Jeśli więc przewiduje się podłączenie zewnętrznego czujnika (czujników), montowanie U1 w obudowie nie ma sensu.

Uruchomienie należy rozpocząć od skompilowania programu. Trzeba pamiętać o wybraniu odpowiedniej wersji płytki. Ze względu na to, że wersji ESP jest kilka, na poniższych zrzutach ekranowych pokazano prawidłowy wybór.







Naturalnie muszą być zainstalowane wymagane biblioteki. Procesu instalacji bibliotek nie będę opisywał, ponieważ zajęłoby to kilka stron a ponadto na łamach EdW jest prowadzony kurs, w którym wszystko jest szczegółowo opisane i nie ma sensu powielać tych informacji.

Jeśli program kompiluje się prawidłowo, można go wgrać do modułu ESP. Po uruchomieniu powinien pojawić się napis „Restart”, po chwili ekran główny:

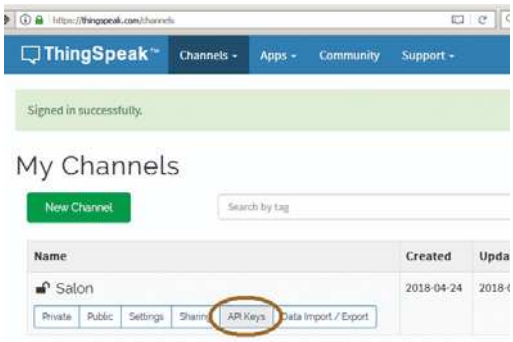
Ze względu na brak interfejsu użytkownika w postaci np. klawiatury, dane sieci Wi-Fi (SSID, hasło) i serwera thingspeak należy umieścić w kodzie źródłowym, po czym skompilować i wgrać do ESP. Aby zmienić dane sieci Wi-Fi, należy odszukać (CTRL+F) ciągu znaków „password”, po czym wpisać dane swojej sieci:

```
//parametry logowania do sieci WiFi
const char* ssid = "nazwaSieci";
const char* password = "haslo";
```

Jeśli dane mają być wysyłane na serwer *thingspeak*, należy odszukać:

```
//dane konta na thingspeak
const char * apiKey = "klucz";
unsigned long myChannelNumber = 1;
```

i zmodyfikować apiKey. Skąd wziąć apiKey? Najpierw trzeba zarejestrować się na stronie *thingspeak.com*. Po rejestracji, potwierdzonej e-mailem i zalogowaniu na stronie, w zakładce „Chanels”



klikamy na „Api Keys” i zobaczymy

gdzie można odczytać lub zmienić klucz.

Program

Program, oczywiście w postaci szkicu .ino, jest dość prosty, zawiera komentarze, nie ma sensu omawiać go szczegółowo. W pętli „loop()” nie ma żadnych „delay” (co jednak wcale nie dowodzi, że funkcje biblioteczne z „delay” nie korzystają). Wszystkie opóźnienia realizowane są przy użyciu „millis()”, co oznacza kłopoty co ok. 49 dni. Najprostszym rozwiązaniem problemu jest reset np. co 24 godziny. W kodzie realizuje to linia

```
CzasDoResetu = millis() + 24 * 60 * 60 * 1000UL;
```

wykonane pod koniec funkcji „setup()” i

```
if( millis()> CzasDoResetu ){
    Serial.print("\n\*** Abort ***");
    abort();      }
```

„w loop()”. Nie jest to jedyny sposób rozwiązania problemu z przepełnieniem licznika systemowego i nie zawsze możliwy do zaakceptowania.

Jedyne w całym kodzie polecenie *delay()* jest użyte w „startup()” w funkcji inicjalizującej sieć Wi-Fi. Nie było sensu, aby w tej sytuacji używać millis(), bo program kręci się w pętli do czasu nawiązania połączenia z siecią Wi-Fi. Jeśli to się nie uda w ciągu 30 sekund, realizowana jest dalsza część programu, bez komunikacji Wi-Fi.

```
n = 30;
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(“.”);
    char txt[20];
    sprintf( txt, “Wait %2d”, n );
    display.clear();
    display.drawString(30, 20, txt );
    //      display.drawLine(0, 0, 64, 32);
    // poziom, pion, poziom, pion
    display.display();
    if ( !n-- ) {
      break;
      //ESP.restart(); Serial.println(“restart “);
    }
  }
  if ( WiFi.status() == WL_CONNECTED ) {
    Serial.println(“\n\rPolaczenie WiFi OK”);
    Serial.println(WiFi.localIP());
  }
  else {
    Serial.println(“\n\rBlad polaczenia Wi-Fi”);
  }
```

Po pięciu minutach zostanie podjęta kolejna próba połączenia z siecią Wi-Fi:

```
if ( millis() > CzasDoRstWifi ) {
       millis() + DEF_RESTART_WIFI;
       WiFi.begin();
      display.drawString(128 / 3, STAT_POZ_Y, “RESTART Wi-Fi”);
       Serial.println(“*Restart Wi-Fi*”); }
```

Nawiązanie połączenia jest realizowane przez funkcję „WiFi.begin();” bez parametrów (ssid, password). Jest to podyktowane tym, że każde wywołanie: WiFi.begin(ssid, passphrase) WiFi.softAP(ssid, passphrase, channel) WiFi.disconnect WiFi.softAPdisconnect zapisuje dane w pamięci FLASH. Częste używanie tych funkcji może spowodować szybkie jej zużycie. Funkcja „WiFi.begin();” wywoływana jest nieco na wyrost, ponieważ ESP sam próbuje nawiązać połączenie z ostatnio używaną siecią. Można jednak zastąpić „WiFi.begin();” funkcją z parametrami i próbować nawiązać połączenie z inną preferowaną siecią. Aby niepotrzebnie nie zapisywać pamięci Flash, przed „WiFi.begin(ssid, password);” należy wywołać „WiFi.persistent(false)”.

W obsłudze barometru dziwić może umieszczenie „bme.begin()” w pętli „loop()”, a nie „setup()”. Dzięki takiemu rozwiązaniu czujnik może być odłączany i podłączany w trakcie pracy urządzenia. Oczywiście nie zaleca się takiego postępowania, ale w programie trzeba przewidzieć możliwość utraty komunikacji, spowodowany np. zakłóceniami. Fragment

```
byte static cntDS = 1;
    if ( !--cntDS ) {
      cntDS = 45000UL / DEF_REFRESH_LCD;
// Co 45 sekund (45/5 = 9 obiegów pętli
wyszukiwanie czujników
```

powoduje, że co 45 sekund przeszukiwana jest magistrala 1-Wire. Dzięki temu wykrywane jest odłączenie czy podłączenie nowych czujników. Tablica

```
const uint8_t charStopnie16x12[] = {
        0B00110000, 0B01111100,
        0B01001000, 0B10000010,
        0B01001001, 0B00000001,
        0B00110010, 0B00000000,
        0B00000010, 0B00000000,
        0B00000010, 0B00000000,
        0B00000010, 0B00000000,
        0B00000010, 0B00000000,

        0B00000010, 0B00000000,
        0B00000001, 0B00000001,
        0B00000000, 0B10000010,
        0B00000000, 0B01111100,  };
```

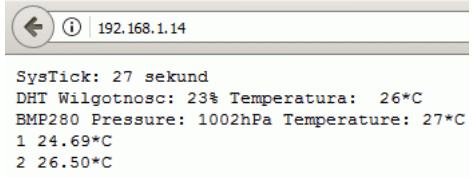
definiuje wzór znaku „°C”. Tablica jest umieszczona w pamięci FLASH. Użytkowników ArduinoUNO i podobnych na AVR zdziwi tu brak atrybutu PROGMEM. Mikrokontroler, który zastosowano w ESP, ma liniową przestrzeń adresową i o tym, czy dane znajdują się w RAM, FLASH, czy też w przestrzeni IO, decyduje adres w pamięci, a nie rozkaz CPU.

Co 100 sekund wyniki pomiarów są wysyłane na serwer, za co odpowiada fragment:

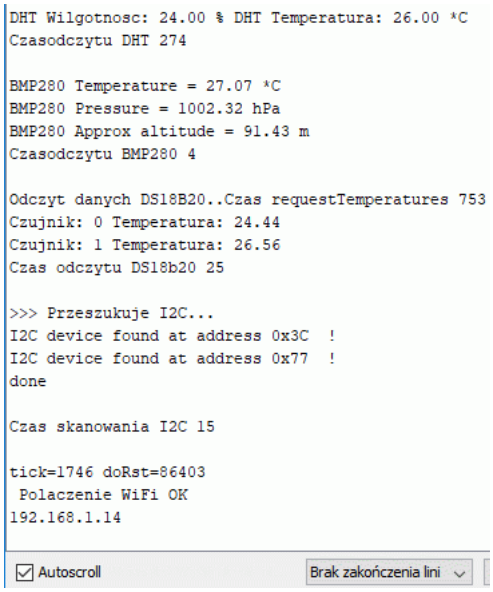
```
uint32_t static CzasDoWWW = 10000;
 if ( millis() > CzasDoWWW  ) {
      CzasDoWWW  =  millis()  +
      100000UL;
```

Wysłanie danych na serwer trwa ponad 5 sekund. Można się o tym dowiedzieć z monitora portu szeregowego. Odpowiadają za to rozkazy, wyróżnione kolorem na **listingu 1**.

Poza wysyłaniem danych na serwer i wyświetlaniem informacji na LCD dane o wilgotności, ciśnieniu i temperaturze są wyświetlane na stronie WWW generowanej przez stronę stacji pogodowej



Adres IP stacji pogodowej można odczytać na routerze, monitorze portu szeregowego lub dowolnym terminalu



W kodzie programu „Serial.print” jest używany dość często, umożliwiając łatwiejsze debugowanie programu.

Strona WWW jest generowana przez funkcję z **listingu 2**.

Ciąg dalszy na stronie 73

Listing 1

```
CzasObiegu = millis();
    ThingSpeak.writeFields(myChannelNumber, apiKey);
     Serial.print(“Czas wysŁania do ThingSpeak “);
     Serial.println( millis() - CzasObiegu);
```

Listing 2

```
void IndexHandler()
{
  char static buf[1500];
  char txt[100];

  buf[0] = 0;
  sprintf( txt, “SysTick: %d sekund”, millis() / 1000 );
  strcat( buf, txt );

  if ( dth_presence ) {
    sprintf( txt, “\nDHT Wilgotnosc: %1.0f%% Temperatura: % 1.0f*C”,
    dth_h, dth_t );
     strcat( buf, txt );
  }
  else {
    strcat( buf, “DTH Err” );
  }

  if ( StBme280 ) {
    sprintf( txt, “\nBMP280 Pressure: %1.0fhPa Temperature:
    %1.0f*C”,
    bme_presure / 100, bme_temp );
    strcat( buf, txt );
  }
  else {
    strcat( buf, “BMP Err” );
  }

  for (byte n = 0; n < MAX_SENSORS && n < device_count; n++) {
    sprintf( txt, “\n%d %1.2f*C”, n+1, temp[n + 2] );
    strcat( buf, txt );
  }

  webServer.send( 200, “text / plain”, buf );
}
```