Zastosowanie moduł Wi-Fi ESP-12 (1)

Wirtualny interfejs szeregowy

UART jest jednym z interfejsów używanych do komunikacji. Jest on łatwy w obsłudze programowej i użyciu, szczególnie w wypadku komunikacji z komputerem PC. Dla uzyskania podstawowej funkcjonalności jest przyłączenie jedynie 3 linii: RxD, TxD oraz masy. Ileż prościej by było, gdyby można zastosować taki interfejs bez używania żadnych kabli. Pozwoliłoby to na bezproblemową komunikację komputera z systemem wbudowanym, bez konieczności ciągnięcia – niekiedy bardzo długiej – wiązki kabli. W artykule zaprezentowano rozwiązanie bazujące na module ESP-12, realizujące bezprzewodową wersję interfejsu szeregowego UART.

Zasada działania transmisji szeregowej – szczypta teorii

Jak sama nazwa mówi, jest to transmisja szeregowa polegająca na wysyłaniu ciągu bitów, które są następnie składane w paczkę danych. Pojedyncza paczka danych składa się z bitu start, danych, bitu stopu i ewentualnie bitu parzystości, jeżeli transmisja jest tak skonfigurowana. Słówko *Asynchronous* w nazwie informuje, że jest to transmisja asynchroniczna, co oznacza, iż zegar taktujący rytm wysyłania danych nie jest przesyłany do odbiornika, tak jak ma to miejsce np. w SPI czy I²C. Dlatego każdorazowo należy ustawić odpowiednią prędkość w nadajniku i odbiorniku – oczywiście, musi ona być taka sama. Efektem ustawienia różnych prędkości będzie odczytanie błędnych danych z interfejsu.

ESP-12 i wirtualny UART

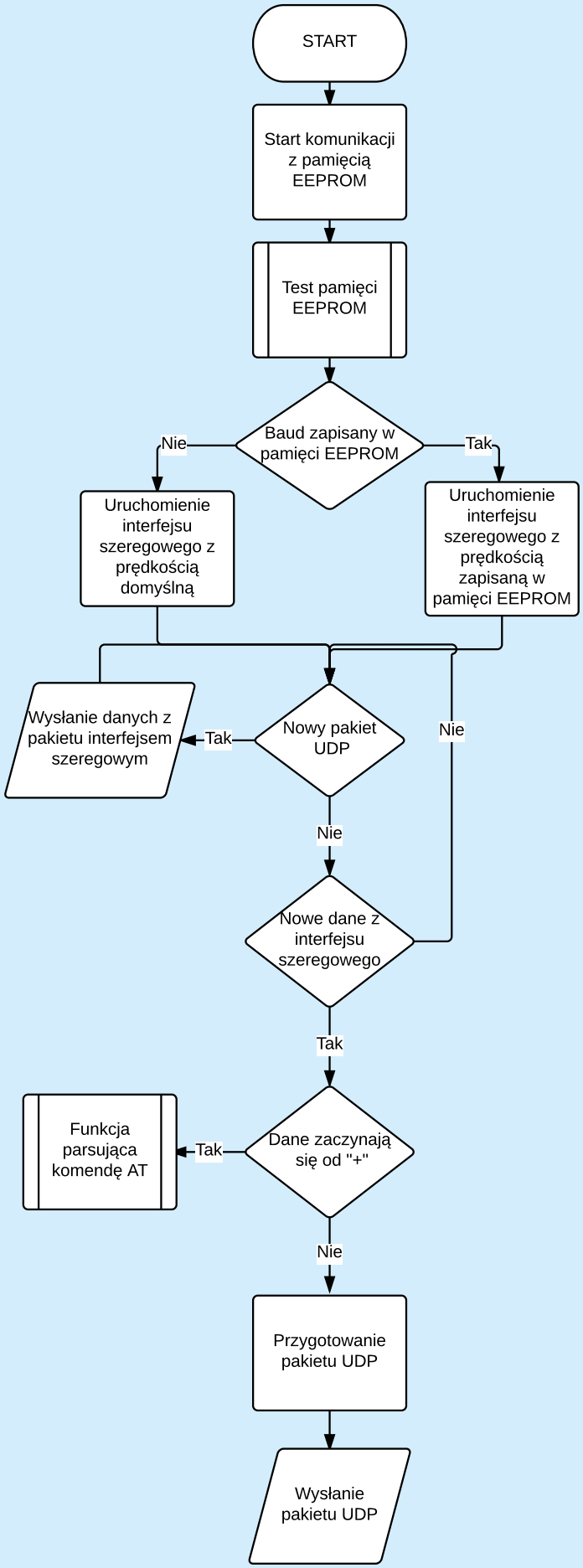
Moduł ESP-12 ma sprzętowy interfejs portu szeregowego, co oznacza, że na wyjściu będą same dane wysłane przez nadajnik. Nie trzeba martwić się o bity startu, stopu itd. Jedyne, co należy zrobić, to skonfigurować ten port. Program źródłowy przedstawiony na **listingu 1** jest rozwiązaniem dla przewodowego UART. Algorytm funkcjonowania programu zaprezentowano na na schemacie blokowym zamieszczonym na **rysunku 1**.

Wszystkie dane, które zostaną odebrane w pakiecie UDP od razu są wysyłane przez port szeregowy. Analogicznie, wszystkie dane, które nie zawierają znaku „+” są pakowane do pakietu UDP i wysyłane do odbiorcy.

Komunikacja UDP została zastosowana z powodu prostoty oraz szybkości wysyłania danych. Testy wykazały, że przy większej ilości danych wysyłanych pakietami TCP, w pewnym momencie następowało przepełnienie, gdyż czas potrzebny na: nawiązanie połączenia TCP, wysłanie pakietu, oczekiwanie na potwierdzenie, ewentualną retransmisję pakietu oraz zamknięcie połączenia, powodował zgromadzenie zbyt dużej ilości danych oczekujących w buforze. Przy wykorzystaniu transmisji UDP, każdy znak odczytany z interfejsu jest natychmiastowo wysyłany pakietem. W wypadku UDP nie nawiązuje się połączenia oraz nie trzeba go zamykać. Wadą tego rozwiązania jest możliwość „zagubienia” pakietu danych. Kontrolę błędów transmisji należałoby wykonać programowo, np. wysyłając na koniec ciągu znaków sumę kontrolną CRC.

Tabela 1. Komendy AT realizowane przez moduł ESP-12

<Status>	<Status> : *OK* – wszystko działa, moduł gotowy do pracy *ERROR* – błąd przy uruchamianiu, należy zrestartować moduł.
+AT	Komenda testująca komunikację Odpowiedź: *<Status>*
+AT_Set_Recv_IP=”<IP>”	Ustawianie adresu IP odbiornika, adres powinien być w formie: xxx.xxx.xxx.xxx np.: **+AT_Set_Recv_IP=”192.168.0.1”** Odpowiedź: *<Status>*
+AT_Recv_IP	Zapytanie o adres IP odbiornika Odpowiedź: *Recv IP: <IP>* *<Status>*
+AT_IP	Zapytanie o adres modułu w lokalnej sieci WiFi Odpowiedź: *Local IP:<IP>* *<Status>*
+AT_Port	Zapytanie o nr portu na którym działa nasłuch oraz wysyłanie pakietów UDP Odpowiedź: Port:<port> <Status>
+AT_Set_Baud=”<Baud>”	Ustawienie prędkości transmisji szeregowej UART, domyślnie jest to wartość 115200 <Baud> - prędkość transmisji Np.: ***+AT_Set_Baud=”115200”*** Odpowiedź: *<Status>*
+AT_Baud	Zapytanie o aktualnie ustawioną prędkość transmisji szeregowej Odpowiedź: *<Baud>* *<Status>*
+AT_RST	Programowy reset modułu
+AT_Connect=”<SSID>”,”<PASS>”	Ustawianie parametrów połączenia z siecią WiFi <SSID> - nazwa sieci WiFi <PASS> - hasło do sieci WiFi Np.: *+AT_Connect=”<dom123>”,”<admin1>”*



Rysunek 1. Schemat blokowy działania programu

Firmware interpretuje i realizuje 9 komend AT. Każda komenda rozpoczyna się członem „+AT_" i musi być zakończona znakiem końca linii. Dopiero wtedy moduł zaakceptuje komendę i będzie próbować ją przetworzyć. Wysyłanie danych za pomocą UDP nie wymaga znaku końca linii, jeżeli przed danymi nie wystąpił znak „+", to dane będą przesyłane paczkami UDP do odbiornika.

W tabeli 1 wymieniono komendy realizowane przez moduł Wi-Fi ESP-12. Nie są one specjalnie skomplikowane, a w razie wątpliwości warto sięgnąć albo do źródła programu z tego odcinka cyklu, albo do dokumentacji samego modułu.

Program sterujący pokazano na **listingu 1**. Jedynym ograniczeniem jest brak możliwości stosowania znaku „+" w wiadomościach. Komentarze są przy każdej ważniej części.

Wykorzystanie w praktyce – bezprzewodowy czujnik

Do testu użyjemy modułu ESP-12 realizującego program z listingu 1. Do tego będzie nam potrzebne Arduino Uno R3 oraz potencjometr. Odbiornikiem będzie komputer pracujący jako serwer UDP na porcie 8080. Odbiornikiem może być każde urządzenie mogące obsłużyć protokół UDP, jednak najlepiej sprawowałby się tutaj bliźniaczy moduł ESP-12 z wgranym programem wirtualnego portu szeregowego.

Mikrokomputer Arduino będzie mierzył napięcie na wejściu przetwornika A/C i wysyłał za pomocą interfjesu szeregowego, więc zmiany będzie można zaobserwować w programie ustawionym na nasłuchiwanie transmisji odbywającej się przez port 8080.

Najpierw jednak należy skonfigurować wirtualny UART. Dla tego celu trzeba wysłać dwie komendy: komendę ustanawiającą połączenie z siecią Wi-Fi (bez tego żadne połączenie nie zostanie nawiązane). Następnie ustawiamy adres odbiornika i możemy zacząć wysyłać dane. W tym przykładzie dane będą tylko wysyłane, nic nie będzie odbierane przez UART. Przykładowy program pokazano na **listingu 2**.

Przed wgraniem wsadu nie należy przyłączać modułu ESP-12 do Arduino, ponieważ zajmiemy wyprowadzenia RxD i TxD, co może zakłócić wgrywanie wsadu do pamięci modułu – najlepiej zrobić to tuż po wgraniu. Po zaprogramowaniu modułu należy zrestartować Arduino. Efektem działania programu będą dane odebrane przez program, który nasłuchuje na odpowiednim porcie.

Program sprawuje się bardzo dobrze, wysyła każdy znak w formie pakietu. Teraz wystarczy złożyć te dane w jedną zmienną, aby łatwo zrealizować pomiar ze zdalnego przetwornika A/C. Należy w tym wypadku zwrócić szczególną uwagę na to, że sam kod czujnika był bardzo krótki. Właściwie zawierał tylko konfigurację interfejsu, wykonywał pomiary i przesyłał je przez interfejs szeregowy. Całą komunikacją zajmuje się moduł ESP-12.

Oprogramowanie jest na tyle uniwersalne, że możemy podłączyć każdy mikroprocesor i/lub czujnik z interfejsem szeregowym. Program ma też wadę. Jest nią brak możliwości wstawiania znaków „+" w paczki danych, jeżeli to nie jest komenda AT. Można też ustawić wysyłanie danych dopiero po otrzymaniu znaku potwierdzenia (może to być znak *'\0'* albo *'\n'* ewentualnie *'\r\n'*).

Jest wiele możliwości zmian w programie i dzięki dostępności źródła każdy może przygotować odpowiedni dla systemu, który jest przygotowywany.

Jakub Kisiel
www.microgeek.eu

Listing 1. Program realizujący transmisję za pomocą UART

```
#include <ESP8266WiFi.h>
#include <WiFiUDP.h>
#include <EEPROM.h>

extern „C” {
#include „user_interface.h”
}

#define DEVICENUMBER 123 //wartość zapisywana do pamięci EEPROM pod adresem 0, jeżeli przy następnym
uruchomieniu ta wartość nie zostanie tam znaleziona,
//do pamięci zostaną wgrane wartości domyślne
typedef union
  {
    uint32_t baudrate;
    struct
    {
      uint8_t byte1;
      uint8_t byte2;
      uint8_t byte3;
      uint8_t byte4;
    };
  } baudrate_unia; //unia zamieniająca 32bit wartość baudrate na 4 8bitowe zmienne, gotowe
do zapisu w pamięci EEPROM

WiFiUDP Udp;

//port, na którym odbędzie się komunikacja
unsigned int localPort = 8080;
byte  packetBuffer[512];
int noBytes = 0;
String received_command = „”;
int at_package = 0;
char at_data[50];
int at_cnt = 0;
char *Tx_IP;
uint16_t Tx_PORT = 8080;
//Adresy do zapisu zmiennych w EEPROM
int SW_Eeprom_Addr = 0;
int Baud_Eeprom_Addr = 1;
int Baud_set_Eeprom_Addr = 5;
//Flagi sprawdzające ustawienie odpowiednich parametrów
int Tx_IP_set = 0;
//int SSID_set = 0;
//int PASS_set = 0;
int Baud_set = 0;
//z powodu że funkcja indexOf przenosi wskaźnik w stringu na element szukany albo na koniec, trzeba
operować na kopii danych
int find_text(String szukany, String zrodlo)
{
  return zrodlo.indexOf(szukany);
}

//ustawianie pamięci EEPROM
void EEprom_Configure()
{
  if (DEVICENUMBER != EEPROM.read(SW_Eeprom_Addr))
  {
    EEPROM.write(SW_Eeprom_Addr, DEVICENUMBER);
    EEPROM.write(Baud_set_Eeprom_Addr, Baud_set);
    EEPROM.commit();
  }
  else
  {
    Baud_set = EEPROM.read(Baud_set_Eeprom_Addr);
  }
}

//operacje na pamięci EEPROM
void EEprom_Baud_Save(uint32_t Baud)
{
  baudrate_unia baud;
  baud.baudrate = Baud;
  EEPROM.write(Baud_Eeprom_Addr, baud.byte1);
  EEPROM.write(Baud_Eeprom_Addr + 1, baud.byte2);
  EEPROM.write(Baud_Eeprom_Addr + 2, baud.byte3);
  EEPROM.write(Baud_Eeprom_Addr + 3, baud.byte4);
  EEPROM.write(Baud_set_Eeprom_Addr, Baud_set);
  EEPROM.commit();
}

uint32_t EEprom_Baud_Read()
{
  baudrate_unia baud;
  baud.byte1 = EEPROM.read(Baud_Eeprom_Addr);
  baud.byte2 = EEPROM.read(Baud_Eeprom_Addr + 1);
  baud.byte3 = EEPROM.read(Baud_Eeprom_Addr + 2);
  baud.byte4 = EEPROM.read(Baud_Eeprom_Addr + 3);
  return baud.baudrate;
}

//funkcja sprawdzająca czy w buforze danych znajduje się polecenie +AT
void AT_Parse(char* data)
{
  String AT_data(data);
  if (find_text(„AT_Set_Recv_IP”, AT_data) != -1)
  {
    String Reciver_IP;
    int poz1 = AT_data.indexOf(„\””);
    int poz2 = AT_data.indexOf(„\””, poz1 + 1);
    Reciver_IP = AT_data.substring(poz1 + 1, poz2);
    Tx_IP = new char[Reciver_IP.length() + 1];
```

Listing 1. cd.

```
    Reciver_IP.toCharArray(Tx_IP, Reciver_IP.length() + 1);
    Tx_IP[strlen(Tx_IP)] = '\0';
    Tx_IP_set = 1;
    Serial.println();
    Serial.println("OK");
    return;
  }
  else if (find_text("AT_Connect", AT_data) != -1)
  {
    String ssid_temp;
    String pass_temp;
    int i = 0;
    int poz1 = AT_data.indexOf("\"");
    int poz2 = AT_data.indexOf("\"", poz1 + 1);
    int poz3 = AT_data.indexOf("\"", poz2 + 1);
    int poz4 = AT_data.indexOf("\"", poz3 + 1);
    ssid_temp = AT_data.substring(poz1 + 1, poz2);
    pass_temp = AT_data.substring(poz3 + 1, poz4);
    char ssid[ssid_temp.length()];
    char pass[pass_temp.length()];
    ssid_temp.toCharArray(ssid, ssid_temp.length() + 1);
    pass_temp.toCharArray(pass, pass_temp.length() + 1);
    WiFi.begin(ssid, pass);
    Serial.println(ssid);
    Serial.println(pass);
    while (WiFi.status() != WL_CONNECTED && i++ < 20) delay(500);
    if (i == 21) {
      Serial.print("ERROR");
      return;
    }
    Udp.begin(localPort);
    Serial.println();
    Serial.println("OK");
    return;
  }
  else if (find_text("AT_Recv_IP", AT_data) != -1)
  {
    Serial.print("Recv IP:");
    Serial.println(Tx_IP);
    Serial.println();
    Serial.println("OK");
    return;
  }
  else if (find_text("AT_IP", AT_data) != -1)
  {
    Serial.print("Local IP:");
    Serial.println(WiFi.localIP());
    Serial.println();
    Serial.println("OK");
    return;
  }
  if (find_text("AT_Port", AT_data) != -1)
  {
    Serial.print("Port:");
    Serial.println(localPort);
    Serial.println();
    Serial.println("OK");
    return;
  }
  else if (find_text("AT_Set_Baud", AT_data) != -1)
  {
    String Baud;
    int poz1 = AT_data.indexOf("\"");
    int poz2 = AT_data.indexOf("\"", poz1 + 1);
    Baud = AT_data.substring(poz1 + 1, poz2);
    Baud_set = 1;
    EEprom_Baud_Save(Baud.toInt());
    delay(500);
    system_restart();
    Serial.println();
    Serial.println("OK");
    return;
  }
  else if (find_text("AT_Baud", AT_data) != -1)
  {
    Serial.print("Baud:");
    Serial.println(EEprom_Baud_Read());
    Serial.println();
    Serial.println("OK");
    return;
  }
  else if (find_text("AT_RST", AT_data) != -1)
  {
    system_restart();
  }
  else if (find_text("AT", AT_data) != -1)
  {
    Serial.println();
    Serial.println("OK");
    return;
  }
  else
  {
    Serial.println();
    Serial.println("ERROR");
  }
}

void setup() {
  uint8_t i = 0;
  EEPROM.begin(512);
```

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **11754**, pass: **208655ee**

Listing 1. cd.

```
  EEprom_Configure();
  //Jeżeli już był ustawiony baudrate, odczytaj go z EEPROM, jeżeli nie użyj domyślnej prędkośći
  if (Baud_set)
  {
    Serial.begin(EEprom_Baud_Read());
  }
  else
  {
    Serial.begin(115200);
  }
}

void loop() {
  noBytes = Udp.parsePacket();
  if ( noBytes ) {
    received_command = „”;
    Udp.read(packetBuffer, noBytes);
    for (int i = 1; i <= noBytes; i++)
    {
      received_command = received_command + char(packetBuffer[i - 1]);
    } // end for
    Serial.println(received_command);
    Serial.println();
  }
  //Jeżeli jakieś dane czekają na odbiór na porcie szeregowym odczytaj je i sprawdź czy zaczynają
się od +AT, jeżeli nie to wyślij je jako pakiet UDP,
  //w innym przypadku zbierz dane aż do znaku '\n' i przekaż do funkcji parse
  if (Serial.available()) {
    size_t len = Serial.available();
    char data;
    if (!at_package)
    {
      data = Serial.read();
    }
    uint8_t sbuf[len];
    if (data == '+' && at_package == 0)
    {
      at_package = 1;
      at_cnt = 0;
      at_data[at_cnt++] = data;
      while (Serial.available())
      {
        at_data[at_cnt++] = Serial.read();
        //        Serial.print(at_data[at_cnt - 1]);
        if (at_data[at_cnt - 1] == '\n')
        {
          at_package = 0;
          AT_Parse(at_data);
          break;
        }
      }
    }
    else if (at_package)
    {
      while (len)
      {
        at_data[at_cnt++] = Serial.read();
        //        Serial.print(at_data[at_cnt - 1]);
        if (at_data[at_cnt - 1] == '\n')
        {
          at_package = 0;
          AT_Parse(at_data);
          break;
        }
        len--;
      }
    }
    //IP odbiornika musi być ustawione, inaczej nic nie zostanie wysłane
    else if (Tx_IP_set)
    {
      sbuf[0] = data;
      Serial.readBytes(&sbuf[1], len - 1);
      //push UART data to all connected telnet clients
      Udp.beginPacket(Tx_IP, Tx_PORT);
      Udp.write(sbuf, len);
      Udp.endPacket();
    }
    else
    {
      Serial.println(„ERROR”);
    }
  }
}
```

Listing 2. Użyteczny przykład – transmisja wyników pomiarów przetwornika A/C

```
int odczytanaWartosc = 0; //Odczytana wartość z ADC
float napiecie = 0; //Wartość przeliczona na napięcie w V

void setup() {
  Serial.begin(9600);//Uruchomienie komunikacji przez USART

  //Konfiguracja wirtualnego portu szeregowego
  Serial.println(„+AT_Connect=\”UPC0045472\”,\”AAAAAAQA\””);
  Serial.println(„+AT_Set_Recv_IP=\”192.168.0.25\””);
}

void loop() {
  odczytanaWartosc = analogRead(A5);//Odczytujemy wartość napięcia
  napiecie = odczytanaWartosc * (5.0/1023.0); //Przeliczenie wartości na napięcie
  Serial.print(napiecie);//Wysyłamy zmierzone napięcie przez wirtualny interfejs szeregowy
  delay(500);
}
```