DIY: zarabianie wtyku RJ45 i tester żył na ESP32

Poziom: średnio

Dotyczy: wtyk RJ45 (8P8C), T568A i T568B, ESP32 (płytka z modułem WROOM), ESPHome 2026.9

Czas: ~90 min

Budujemy dwie rzeczy: kabel krosowy zarobiony własnoręcznie i prosty tester, który sprawdza osiem żył przez dwa gniazda RJ45. Tester na ESP32 pokazuje, czy każda żyła dochodzi do właściwego styku, i oddaje wynik przez stronę WWW. Wracamy tu przy krótkim kablu do szafy, przy naprawie wtyku i przy sprawdzaniu kabla, któremu nie ufamy. Nie jest to certyfikacja okablowania, a granice opisujemy w osobnej sekcji.

Skróty użyte na tej stronie (22)

Skrót	Co znaczy
8P8C	8 Position 8 Contact: ścisła nazwa wtyku modułowego z ośmioma miejscami i ośmioma stykami, potocznie nazywanego RJ45
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
CRC	Cyclic Redundancy Check: suma kontrolna dopisana do odpowiedzi czujnika, dzięki której wykrywamy przekłamany odczyt
DHCP	Dynamic Host Configuration Protocol: usługa, która sama rozdaje komputerom adresy IP
ESP32	Tani mikrokontroler z Wi-Fi i Bluetooth firmy Espressif: malutki komputer, z którego budujemy własne czujniki sieciowe
ESPHome	Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć
I2C	Inter-Integrated Circuit: prosta dwuprzewodowa magistrala, którą mały układ (np. czujnik) rozmawia z mikrokontrolerem
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
JSON	JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami
MCP23017	Układ ekspandera I2C, który dodaje płytce 16 wejść i wyjść cyfrowych
PoE	Power over Ethernet: zasilanie urządzenia tym samym kablem sieciowym, którym płyną dane

Skrót	Co znaczy
PSRAM	Pseudo-Static RAM: dodatkowa pamięć RAM w niektórych modułach ESP32, na przykład WROVER, która zajmuje piny GPIO16 i GPIO17
RJ45	Registered Jack 45: potoczna nazwa ośmiostykowego wtyku i gniazda 8P8C, w którym kończymy kable skrętkowe Ethernetu
T568A	Schemat wtyku RJ45 z normy TIA-568, w którym para zielona leży na stykach 1 i 2, a pomarańczowa na stykach 3 i 6
T568B	Schemat wtyku RJ45 z normy TIA-568, w którym para pomarańczowa leży na stykach 1 i 2, a zielona na stykach 3 i 6
TIA	Telecommunications Industry Association: amerykańskie stowarzyszenie, które wydaje normy okablowania strukturalnego, na przykład TIA-568 i TIA-606
USB	Universal Serial Bus: standard portów i urządzeń podłączanych do komputera, tu pendrive'y i inne nośniki wymienne
Wi-Fi	Sieć bezprzewodowa: wygodna w biurze, ale w serwerowni dokłada kolejną możliwą przyczynę awarii
WROOM	ESP32-WROOM: popularny moduł ESP32 bez dodatkowej pamięci PSRAM, na którym budujemy płytki rozwojowe
WROVER	ESP32-WROVER: moduł ESP32 z dodatkową pamięcią PSRAM, więc część pinów GPIO jest w nim zajęta
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS
YAML	YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach

Fiszka

> Poniższe polecenia wykonujemy w bashu na stacji administratora, z płytką ESP32 podłączoną kablem USB. Konfigurację z tej strony sprawdziliśmy poleceniem `esphome config` na ESPHome 2026.9.1, a logikę wyniku na komputerze z zasymulowanymi danymi. Programu nie kompilowaliśmy i płytki nie uruchamialiśmy, więc pierwsze uruchomienie sprawdzamy sami.

□□ TYLKO ODCZYT

Zarabianie wtyku: zaciskarka RJ45

1. Odcinamy równo koniec kabla i zdejmujemy płaszcz na długość podaną w instrukcji wtyku, bez nacinania izolacji żył.

2. Odginamy ekran, jeśli kabel go ma, i obcinamy krzyżak rozdzielający pary, jeśli jest w kablu.
3. Rozplatamy pary na jak najkrótszym odcinku: dla kategorii 5e i wyżej najwyżej 13 mm od miejsca zakończenia, a dla kategorii 6 jak najmniej [1].
4. Układamy żyły płasko według T568B albo T568A i prostujemy je palcami.
5. Przycinamy żyły równo, na długość podaną w instrukcji wtyku.
6. Wsuwamy żyły do wtyku do oporu, tak żeby miedź było widać w czole wtyku, a płaszcz wszedł pod zacisk.
7. Zaciskamy wtyk zaciskarką do oporu.
8. Drugi koniec zarabiamy według tego samego schematu i sprawdzamy kabel testerem.

Zaciskarka ma gniazdo na wtyk 8P8C, a wtyk musi pasować do kabla: inny dla drutu, inny dla linki, inny dla kabla ekranowanego.

RJ45 = Registered Jack 45: potoczna nazwa ośmiostykowego wtyku i gniazda 8P8C, w którym kończymy kable skrętkowe Ethernetu · **T568B** = Schemat wtyku RJ45 z normy TIA-568, w którym para pomarańczowa leży na stykach 1 i 2, a zielona na stykach 3 i 6 · **T568A** = Schemat wtyku RJ45 z normy TIA-568, w którym para zielona leży na stykach 1 i 2, a pomarańczowa na stykach 3 i 6 · **8P8C** = 8 Position 8 Contact: ścisła nazwa wtyku modułowego z ośmioma miejscami i ośmioma stykami, potocznie nazywanego RJ45

TYLKO ODCZYT

Układ żył we wtyku: T568A i T568B

Styk	T568A	T568B
1	biało-zielona	biało-pomarańczowa
2	zielona	pomarańczowa
3	biało-pomarańczowa	biało-zielona
4	niebieska	niebieska
5	biało-niebieska	biało-niebieska
6	pomarańczowa	zielona
7	biało-brązowa	biało-brązowa
8	brązowa	brązowa

Wtyk trzymamy zatrzaskiem w dół, stykami do góry i do siebie, więc styk 1 jest po lewej. Oba końce kabla krosowego zarabiamy tym samym schematem. W Zielonej przyjmujemy T568B, jeśli nie ma powodu na T568A.

T568A = Schemat wtyku RJ45 z normy TIA-568, w którym para zielona leży na stykach 1 i 2, a pomarańczowa na stykach 3 i 6 · **T568B** = Schemat wtyku RJ45 z normy TIA-568, w którym para

pomarańczowa leży na stykach 1 i 2, a zielona na stykach 3 i 6

□□ ZMIENIA SYSTEM

Instalacja ESPHome: pipx install esphome

```
pipx install esphome
```

ESPHome buduje program dla płytki ESP32 z jednego pliku YAML i wgrywa go przez USB albo przez sieć.

□□ **ESPHome** = Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć · **ESP32** = Tani mikrokontroler z Wi-Fi i Bluetooth firmy Espressif: malutki komputer, z którego budujemy własne czujniki sieciowe · **YAML** = YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach · **USB** = Universal Serial Bus: standard portów i urządzeń podłączanych do komputera, tu pendrive'y i inne nośniki wymienne · **pipx** = instaluje narzedzia napisane w Pythonie, kazde we wlasnym, odizolowanym srodowisku, bez mieszania zaleznosci miedzy programami

□□ TYLKO ODCZYT

Wygenerowanie klucza API: openssl rand -base64 32

```
openssl rand -base64 32
```

Klucz szyfrujący natywne API to 32 losowe bajty w base64, a ESPHome odrzuci każdy inny tekst [2].

□□ **API** = Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą · **ESPHome** = Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć · **openssl** = zestaw narzędzi do certyfikatów i szyfrowania

□□ TYLKO ODCZYT

Plik z hasłami: secrets.yaml

```
# secrets.yaml, obok pliku tester-kabli.yaml
api_klucz: "WynikPolecenia_openssl_rand_base64_32"
ota_haslo: "HasloOta"
wifi_nazwa: "NazwaSieci"
wifi_haslo: "HasloSieci"
ap_haslo: "HasloAwaryjne"
```

```
web_uzytkownik: "nazwa_usera"
web_haslo: "HasloWebServera"
```

Hasła trzymamy w osobnym pliku i odwołujemy się do nich przez `!secret`. Wartości powyżej to placeholdery, a klucz API pochodzi z poprzedniej fiszki.

API = Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą

PLIK KONFIGURACYJNY

Plik konfiguracyjny ESPHome: tester-kabli.yaml

```
esphome:
  name: tester-kabli
  friendly_name: Tester kabli RJ45

esp32:
  board: esp32dev
  framework:
    type: esp-idf

logger:

# Urządzenie warsztatowe: adres z DHCP, bez rezerwacji w sieci serwerów.
api:
  encryption:
    key: !secret api_klucz

ota:
  - platform: esphome
    password: !secret ota_haslo

wifi:
  ssid: !secret wifi_nazwa
  password: !secret wifi_haslo
  ap:
    ssid: "Tester kabli"
    password: !secret ap_haslo
```

```
web_server:
  port: 80
  auth:
    username: !secret web_uzytkownik
    password: !secret web_haslo

# Gniazdo A: wyjścia na styki 1 do 8, każde przez rezystor 330 omów.
output:
  - platform: gpio
    id: wy1
    pin: GPIO13
  - platform: gpio
    id: wy2
    pin: GPIO14
  - platform: gpio
    id: wy3
    pin: GPIO16
  - platform: gpio
    id: wy4
    pin: GPIO17
  - platform: gpio
    id: wy5
    pin: GPIO18
  - platform: gpio
    id: wy6
    pin: GPIO19
  - platform: gpio
    id: wy7
    pin: GPIO21
  - platform: gpio
    id: wy8
    pin: GPIO22

# Gniazdo B: wejścia ze styków 1 do 8, z wewnętrznym rezystorem do masy.
# GPIO5 jest pinem rozruchowym, więc kabel wpinamy dopiero po starcie płytki.
binary_sensor:
  - platform: gpio
    id: we1
    internal: true
```

```
    pin: {number: GPIO23, mode: {input: true, pulldown: true}}
  - platform: gpio
    id: we2
    internal: true
    pin: {number: GPIO25, mode: {input: true, pulldown: true}}
  - platform: gpio
    id: we3
    internal: true
    pin: {number: GPIO26, mode: {input: true, pulldown: true}}
  - platform: gpio
    id: we4
    internal: true
    pin: {number: GPIO27, mode: {input: true, pulldown: true}}
  - platform: gpio
    id: we5
    internal: true
    pin: {number: GPIO32, mode: {input: true, pulldown: true}}
  - platform: gpio
    id: we6
    internal: true
    pin: {number: GPIO33, mode: {input: true, pulldown: true}}
  - platform: gpio
    id: we7
    internal: true
    pin: {number: GPIO4, mode: {input: true, pulldown: true}}
  - platform: gpio
    id: we8
    internal: true
    pin: {number: GPIO5, mode: {input: true, pulldown: true}}

text_sensor:
  - platform: template
    id: wynik
    name: "Wynik"
  - platform: template
    id: mapa
    name: "Mapa żył"

globals:
```

```
  - id: maski
    type: uint8_t[8]

script:
  - id: testuj
    mode: single
    then:
      - lambda: |-
          id(wynik).publish_state("Test trwa");
      - repeat:
          count: 8
          then:
            - lambda: |-
                switch (iter) {
                  case 0: id(wy1).turn_on(); break;
                  case 1: id(wy2).turn_on(); break;
                  case 2: id(wy3).turn_on(); break;
                  case 3: id(wy4).turn_on(); break;
                  case 4: id(wy5).turn_on(); break;
                  case 5: id(wy6).turn_on(); break;
                  case 6: id(wy7).turn_on(); break;
                  default: id(wy8).turn_on(); break;
                }
            - delay: 60ms
            - lambda: |-
                uint8_t m = 0;
                if (id(we1).state) m |= 1;
                if (id(we2).state) m |= 2;
                if (id(we3).state) m |= 4;
                if (id(we4).state) m |= 8;
                if (id(we5).state) m |= 16;
                if (id(we6).state) m |= 32;
                if (id(we7).state) m |= 64;
                if (id(we8).state) m |= 128;
                id(maski)[iter] = m;
                id(wy1).turn_off();
                id(wy2).turn_off();
                id(wy3).turn_off();
                id(wy4).turn_off();
```

```
                id(wy5).turn_off();
                id(wy6).turn_off();
                id(wy7).turn_off();
                id(wy8).turn_off();
            - delay: 30ms
      - lambda: |-
          std::string opis;
          std::string bledy;
          bool prosty = true;
          bool krosowany = true;
          for (int i = 0; i < 8; i++) {
            uint8_t m = id(maski)[i];
            int zadane = 1 << i;
            int krzyz = (i == 0) ? 4 : (i == 2) ? 1 : (i == 1) ? 32 : (i == 5) ? 2 :
zadane;
            if (m != zadane) prosty = false;
            if (m != krzyz) krosowany = false;
            opis += std::to_string(i + 1) + ">";
            if (m == 0) opis += "brak";
            for (int j = 0; j < 8; j++) {
              if (m & (1 << j)) opis += std::to_string(j + 1);
            }
            opis += " ";
            if (m == 0) {
              bledy += "żyła " + std::to_string(i + 1) + " przerwana; ";
            } else if ((m & (m - 1)) != 0) {
              bledy += "żyła " + std::to_string(i + 1) + " zwarta z inną; ";
            }
          }
          id(mapa).publish_state(opis);
          if (prosty) {
            id(wynik).publish_state("OK: kabel prosty, ten sam wzór na obu końcach");
          } else if (krosowany) {
            id(wynik).publish_state("OK: kabel krosowany, T568A na jednym końcu i T568B
na drugim");
          } else if (!bledy.empty()) {
            id(wynik).publish_state("BŁĄD: " + bledy);
          } else {
            id(wynik).publish_state("BŁĄD: żyły zamienione miejscami, patrz mapa");
```

```
          }

button:
  - platform: template
    name: "Testuj kabel"
    on_press:
      - script.execute: testuj
```

Plik opisujemy w sekcji o projekcie niżej. Każdą żyłę pobudzamy po kolei na gnieździe A i patrzymy, które wejścia gniazda B zobaczyły napięcie.

ESPHome = Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć · **RJ45** = Registered Jack 45: potoczna nazwa ośmiostykowego wtyku i gniazda 8P8C, w którym kończymy kable skrętkowe Ethernetu · **id** = pokaż użytkownika, jego numer UID oraz grupy, do których należy · **T568A** = Schemat wtyku RJ45 z normy TIA-568, w którym para zielona leży na stykach 1 i 2, a pomarańczowa na stykach 3 i 6 · **T568B** = Schemat wtyku RJ45 z normy TIA-568, w którym para pomarańczowa leży na stykach 1 i 2, a zielona na stykach 3 i 6

TYLKO ODCZYT

Sprawdzenie pliku: esphome config tester-kabli.yaml

```
esphome config tester-kabli.yaml
```

Polecenie sprawdza składnię pliku bez płytki i bez kompilacji. Poprawny plik kończy się napisem `Configuration is valid!`.

esphome = ESPHome: buduje i wgrywa oprogramowanie na ESP32 z pliku YAML, pokazuje logi urządzenia

TYLKO ODCZYT

Wgranie programu przez USB: esphome run tester-kabli.yaml

```
esphome run tester-kabli.yaml
```

Pierwszy raz wgrywamy program przez USB, a kolejne razy ESPHome wgrywa go przez sieć. Polecenie pyta o port, a po wgraniu pokazuje logi płytki.

USB = Universal Serial Bus: standard portów i urządzeń podłączanych do komputera, tu pendrive'y i inne nośniki wymienne · **ESPHome** = Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć · **esphome** =

ESPHome: buduje i wgrywa oprogramowanie na ESP32 z pliku YAML, pokazuje logi urządzenia

TYLKO ODCZYT

Podgląd logów: esphome logs tester-kabli.yaml

```
esphome logs tester-kabli.yaml
```

Logi pokazują adres IP, jaki płytka dostała z DHCP, i ewentualne błędy połączenia z Wi-Fi.

IP = Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection) · **DHCP** = Dynamic Host Configuration Protocol: usługa, która sama rozdaje komputerom adresy IP · **Wi-Fi** = Sieć bezprzewodowa: wygodna w biurze, ale w serwerowni dokłada kolejną możliwą przyczynę awarii · **esphome** = ESPHome: buduje i wgrywa oprogramowanie na ESP32 z pliku YAML, pokazuje logi urządzenia

TYLKO ODCZYT

Uruchomienie testu z konsoli: curl

```
curl -s -u nazwa_usera:HasloWebServera -X POST "http://tester-
kabli.local/button/testuj_kabel/press"
curl -s -u nazwa_usera:HasloWebServera "http://tester-kabli.local/text_sensor/wynik"
```

Pierwsze polecenie naciska przycisk testu, a drugie odczytuje wynik w JSON. Adresy zasobów powstają z nazw elementów, więc przy odpowiedzi 404 sprawdzamy listę elementów na stronie WWW płytki. Zwykle wystarcza przycisk na stronie WWW, a polecenia przydają się przy skryptach.

JSON = JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami ·**WWW** = World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS ·**curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

Z życia wzięte

Z ŻYCIA WZIĘTE

Środa, 13:40. Tomek zarabia sam kabel krosowy do szafy ściennej w Swarzędzu, bo gotowego nie ma w magazynie. Prosty tester z ośmioma diodami zapala wszystkie żyły po kolei, więc kabel wędruje do szafy, a łącze wstaje na 100 Mb/s i pokazuje błędy CRC. Okazuje

> się, że Tomek zarobił oba końce tak samo, ale ze złą kolejnością: pary nie leżały na właściwych stykach, a tester tego błędu nie widzi.

Projekt DIY krok po kroku

Lista części

- Kabel skrętkowy kategorii 5e albo 6 do ćwiczeń, najlepiej linka, bo kable krosowe są z linki.
- Wtyki RJ45 dopasowane do kabla (linka albo drut, nieekranowane albo ekranowane).
- Zaciskarka z gniazdem 8P8C, ściągacz płaszcza i szczypce tnące.
- Prosty tester kabli z ośmioma diodami, do porównania z naszym.
- Płytka ESP32 z modułem WROOM, na przykład typowa płytka rozwojowa z gniazdem USB.
- Dwa gniazda RJ45 z wyprowadzeniami na piny (moduły do płytki stykowej).
- Szesnaście rezystorów 330 omów i przewody połączeniowe.

Wtyk RJ45 zarabiany na kablu skrętkowym. Żyły leżą płasko i równo przycięte, a płaszcz wchodzi po

Wtyk RJ45 zarabiany na kablu skrętkowym. Żyły leżą płasko i równo przycięte, a płaszcz wchodzi pod zacisk. Źródło: Wikimedia Commons (autor i wolna licencja podane na stronie pliku).

Zarabianie wtyku

Kroki podaje fiszka na początku strony. Tu zapamiętujemy trzy rzeczy: kolejność żył według jednego schematu na obu końcach, jak najkrótszy odcinek rozplecionych par i płaszcz wciśnięty pod zacisk, żeby kabel nie wysunął się z wtyku.

Zaciskarka do wtyków RJ45. Wtyk wsuwamy w gniazdo, a zaciskarkę zamykamy do oporu.

Zaciskarka do wtyków RJ45. Wtyk wsuwamy w gniazdo, a zaciskarkę zamykamy do oporu. Źródło: Wikimedia Commons (autor i wolna licencja podane na stronie pliku).



Rysunek 1. Układ żył według T568A i T568B: osiem styków wtyku od lewej, z kolorami żył i parami. Warto zauważyć, że oba schematy różnią się tylko miejscem pary zielonej i pomarańczowej. (kliknięcie otwiera pełny rozmiar)

W schemacie T568A para zielona leży na stykach 1 i 2, a w T568B na stykach 3 i 6. Para niebieska i para brązowa zostają bez zmian. Dlatego kabel z A na jednym końcu i B na drugim zamienia miejscami pary i wychodzi z niego kabel krosowany, a nie prosty [3].

Test prostym testerem

Wpinamy kabel w oba gniazda testera i patrzymy na diody. Osiem diod ma zapalać się po kolei od 1 do 8, a kabel z A na jednym końcu i B na drugim pokaże zamianę 1 z 3 i 2 z 6. Brak jednej diody znaczy przerwę, a dwie diody naraz zwarcie. Prosty tester nie widzi jednak błędu, który jest taki sam na obu końcach, jak w scenie wyżej.

Prosty tester kabli sieciowych. Kabel wpinamy w dwa gniazda i patrzymy na kolejność diod.

Prosty tester kabli sieciowych. Kabel wpinamy w dwa gniazda i patrzymy na kolejność diod. Źródło: Wikimedia Commons (autor i wolna licencja podane na stronie pliku).

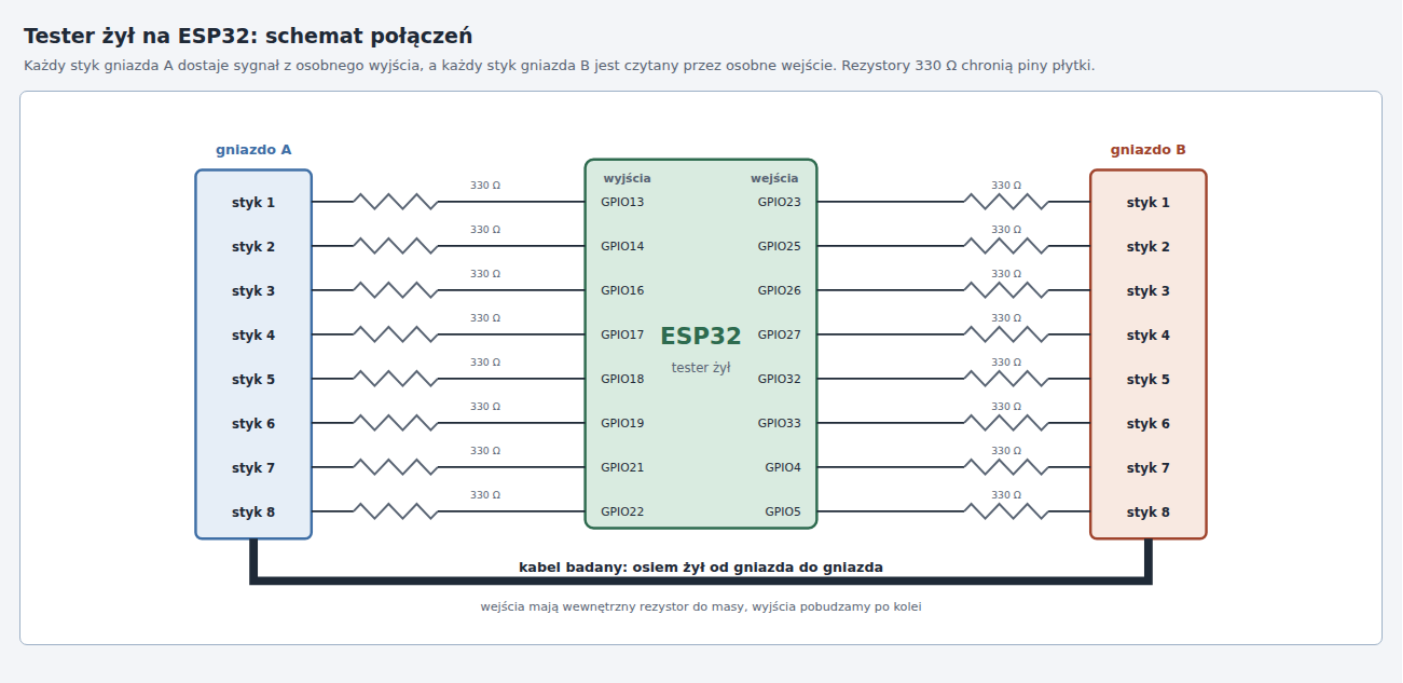
Tester na ESP32

Nasz tester ma dwa gniazda: z gniazda A wysyłamy sygnał po kolei na każdy styk, a gniazdo B odczytuje, na którym styku sygnał się pojawił. Z tego mamy mapę żył i wynik w formie tekstu na stronie WWW. Jest to tester warsztatowy z Wi-Fi i adresem z DHCP, więc nie rezerwujemy mu

miejsca w sieci serwerów.

Płytka rozwojowa ESP32 z modułem WROOM. Do jej pinów podłączamy dwa gniazda RJ45 przez rezy

Płytka rozwojowa ESP32 z modułem WROOM. Do jej pinów podłączamy dwa gniazda RJ45 przez rezystory. Źródło: Wikimedia Commons (autor i wolna licencja podane na stronie pliku).



Rysunek 2. Schemat testera: osiem wyjść ESP32 przez rezystory 330 omów do gniazda A, osiem wejść z rezystorami do gniazda B i kabel badany między gniazdami. Warto zauważyć, że żaden z pinów nie jest wystawiony bezpośrednio na złącze. (kliknięcie otwiera pełny rozmiar)

Osiem żył wymaga szesnastu pinów, osiem wyjść i osiem wejść, bo każdy styk badamy na obu końcach. Wybraliśmy piny bez funkcji rozruchowych, poza GPIO5, i bez pinów tylko do odczytu (GPIO34 do GPIO39), które nie mają wewnętrznych rezystorów do zasilania i do masy [4]. Gdy brakuje pinów, rozszerzamy płytkę o ekspander I2C, na przykład MCP23017.

Styk RJ45	Gniazdo A (wyjście)	Gniazdo B (wejście)
1	GPIO13	GPIO23
2	GPIO14	GPIO25
3	GPIO16	GPIO26
4	GPIO17	GPIO27
5	GPIO18	GPIO32
6	GPIO19	GPIO33

Styk RJ45	Gniazdo A (wyjście)	Gniazdo B (wejście)
7	GPIO21	GPIO4
8	GPIO22	GPIO5

Każdą z szesnastu linii prowadzimy do gniazda przez rezystor 330 omów, żeby pomyłka przy łączeniu nie uszkodziła wyjścia płytki. W modułach z pamięcią PSRAM, na przykład WROVER, piny GPIO16 i GPIO17 zajmuje ta pamięć, więc dla takich płytek wybieramy inne piny [4].

Pierwsze uruchomienie i odczyt

Po wgraniu programu otwieramy stronę WWW płytki pod adresem z logów i naciskamy przycisk „Testuj kabel”. W polach „Wynik” i „Mapa żył” widzimy rozkład. Mapa ma postać ciągu par „żyła z gniazda A>styki z gniazda B”, więc zapis `1>3` oznacza, że sygnał z styku 1 gniazda A pojawił się na styku 3 gniazda B.

Mapa	Wynik	Znaczenie
`1>1 2>2 ... 8>8`	kabel prosty	ten sam schemat na obu końcach
`1>3 2>6 3>1 4>4 5>5 6>2 7>7 8>8`	kabel krosowany	T568A na jednym końcu, T568B na drugim
`4>brak`	żyła przerwana	wtyk źle zaciśnięty albo przerwana żyła
`4>45 5>45`	zwarcie	dwie żyły się stykają
`7>8 8>7`	żyły zamienione	zła kolejność na jednym końcu

Przy kablu z błędem poprawiamy wtyk, który podejrzewamy, i powtarzamy test. Dobry wynik testu oznacza tylko, że styki połączone są tak, jak to widać na mapie.

Bezpieczeństwo i ograniczenia

Tester nie jest certyfikatorem. Nie mierzy długości, tłumienności, przesłuchu ani odbić, więc nie dowodzi, że kabel obsłuży 1 ani 10 Gb/s. Do odbioru okablowania służy certyfikator, jak opisuje strona Odbiór okablowania od wykonawcy: lista kontrolna, testy, protokoły.

Kilka ograniczeń wynika z konstrukcji.

- Nie wykrywamy błędu zrobionego tak samo na obu końcach, w tym rozdzielonych par, bo mapa pokaże wtedy kabel prosty.
- Nie testujemy ekranu kabla, bo gniazda nie mają do niego styku.
- Do gniazd testera podłączamy tylko kabel, który badamy. Wpięcie w żywy port przełącznika, a zwłaszcza w port PoE z napięciem około 48 V, zniszczy płytkę.

- Dla długiego okablowania stałego potrzebujemy testera ze zdalną częścią, bo oba końce muszą być przy nim.
- GPIO5 jest pinem rozruchowym, więc kabel wpinamy po starcie płytki, a nie przy wciskaniu przycisku reset.

Strona WWW płytki ma hasło, ale działa bez szyfrowania, więc tester trzymamy w sieci warsztatowej i nie wystawiamy go na zewnątrz.

> Więcej możliwości albo gotowe rozwiązanie od ręki (out of the box): prosty tester kabli z częścią zdalną, który sprawdza także kable stałe, tester z wyświetlaczem pokazującym mapę żył i długość kabla, albo certyfikator z raportem do wczytania do dokumentacji. W dużych centrach danych stosuje się certyfikację każdego łącza przyrządem klasy certyfikacyjnej, a kable krosowe kupuje się fabrycznie zarobione i przetestowane.

Częste błędy

Objaw	Przyczyna	Co robimy
Wtyk wysuwa się z kabla	płaszcz nie wszedł pod zacisk	skracamy żyły do długości z instrukcji i zaciskamy ponownie
Mapa pokazuje `1>3 2>6` na kablu, który miał być prosty	jeden koniec w T568A, drugi w T568B	zarabiamy jeden koniec ponownie według jednego schematu
Mapa jest poprawna, a łącze ma błędy CRC	rozdzielone pary albo zbyt długie rozplecenie	zarabiamy wtyk od nowa z krótkim rozpleceniem i właściwą kolejnością
Wszystkie żyły „brak”	kabel nie jest wpięty w oba gniazda albo gniazda są źle połączone z płytką	sprawdzamy kabel i połączenia gniazd z płytką
`esphome config` zgłasza błąd klucza API	klucz nie jest 32 bajtami w base64	generujemy klucz poleceniem `openssl rand -base64 32`
Płytka nie startuje poprawnie z wpiętym kablem	GPIO5 jest pinem rozruchowym, a kabel może zmienić jego stan przy starcie	wpinamy kabel po starcie płytki

□□ Ćwiczenia

1. Tester pokazuje mapę `1>3 2>6 3>1 4>4 5>5 6>2 7>7 8>8`. Czy kabel jest uszkodzony?

Odpowiedź

Nie musi być. To kabel krosowany: T568A na jednym końcu i T568B na drugim. Jeśli chcemy kabla prostego, poprawiamy jeden koniec.

1. Kabel dostaje „OK: kabel prosty”, ale łącze gigabitowe działa z błędami. Jak to możliwe?

Odpowiedź

Tester sprawdza tylko ciągłość żył. Oba końce mogą mieć tę samą złą kolejność, więc pary skręcone w kablu leżą na niewłaściwych stykach, a mapa pokazuje kabel prosty. Poprawiamy kolejność i, jeśli to ważne łącze, mierzymy je certyfikatorem.

1. Dlaczego nie wpinamy testera w port przełącznika z PoE?

Odpowiedź

Port PoE podaje napięcie około 48 V, a piny ESP32 pracują przy 3,3 V. Napięcie zniszczy płytkę, więc badamy tylko kable odłączone od sprzętu.

Powiązane: Okablowanie miedziane: kategorie, klasy, ekranowanie i długości kanału, Porządek i dokumentacja: patchpanele, etykiety, kolory i NetBox, Odbiór okablowania od wykonawcy: lista kontrolna, testy, protokoły, Mapa cieplna na ESP32: ΔT szafy i różnica ciśnień między korytarzami (DIY)

Źródła:

1. Fluke Networks, „Terminating a Category 6, 5e and 5 connector”, maksymalne rozplecenie par: flukenetworks.com
2. ESPHome, dokumentacja szyfrowania natywnego API: esphome.io
3. Wikipedia, „TIA/EIA-568”, schematy T568A i T568B: en.wikipedia.org/wiki/TIA/EIA-568
4. Espressif, „ESP32 Series Datasheet”, piny rozruchowe i piny tylko do odczytu: espressif.com

Revision #12
Created 2026-10-03 15:34:14 UTC by Tomasz Konieczny
Updated 2026-10-26 14:23:07 UTC by Tomasz Konieczny