

Graficzne programowanie mikrokontrolerów

M5Stack / UIFlow

Laboratorium 7

Komunikacja UART i RS485

Czas trwania: 2 × 45 min**Tryb:** studia stacjonarne**Jednostka:** Katedra Elektrotechniki i Technologii Nadprzewodnikowych**Prowadzący:** dr inż. Michał Łanczont

Cel ćwiczenia

Celem ćwiczenia jest poznanie zasad transmisji szeregowej UART oraz magistrali RS485. Student konfiguruje parametry transmisji, wysyła i odbiera dane, tworzy prosty tekstowy protokół poleceń oraz implementuje podstawową walidację, przekroczenie czasu oczekiwania i diagnostykę komunikacji.

UART

UART przesyła dane asynchronicznie. Obie strony muszą używać zgodnych parametrów, przede wszystkim prędkości transmisji, liczby bitów danych, parzystości i liczby bitów stopu. W najprostszym połączeniu wykorzystuje się linie TX i RX oraz wspólny potencjał odniesienia.

RS485

RS485 definiuje warstwę elektryczną wykorzystującą transmisję różnicową. W typowej konfiguracji dwuprzewodowej transmisja jest półdupleksowa: w danej chwili węzeł nadaje albo odbiera. Sama warstwa RS485 nie określa znaczenia komunikatów, dlatego w ćwiczeniu zostanie zbudowany prosty protokół aplikacyjny.

Przykładowe polecenia:

```
USTAW:SILNIK:50
POBIERZ:TEMPERATURA
POBIERZ:STAN
ZATRZYMAJ
```

Modbus RTU jako protokół nad RS485

RS485 określa warstwę elektryczną transmisji, natomiast **Modbus RTU** określa znaczenie przesyłanych bajtów. W typowym układzie jeden węzeł pełni rolę klienta/urządzenia nadrzędnego i inicjuje wymianę, a urządzenie serwer/urządzenie podrzędne odpowiada na żądania skierowane pod jego adres. W tym ćwiczeniu przyjmij:

Core + Demo Board: urządzenie nadrzędne Modbus (master/client) ↔ RS485 A/B ↔ Core2 + interfejs RS485: urządzenie podrzędne Modbus (slave/server)

Core2 należy połączyć z magistralą przez odpowiedni transceiver/Unit RS485 korzystający z UART. **Nie należy mechanicznie instalować Core2 na Demo Board.** Oznaczenia A/B potrafią różnić się pomiędzy producentami transceiverów, dlatego należy kierować się dokumentacją konkretnego modułu i potwierdzić komunikację testem.

Minimalna ramka Modbus RTU zawiera:

Pole	Znaczenie
Adres	adres urządzenia podrzędnego, 1 bajt
Kod funkcji	rodzaj operacji, np. 03 – odczyt rejestrów przechowujących (Holding Registers), 06 – zapis jednego rejestru przechowującego
Dane	adres rejestru, liczba rejestrów albo zapisywana wartość
CRC-16	kontrola poprawności ramki; w RTU bajt młodszy CRC jest transmitowany jako pierwszy

W podstawowej wersji ćwiczenia nie trzeba samodzielnie implementować obliczania CRC, jeżeli wykorzystywane bloki/klasz Modbus wykonują tę operację automatycznie.

Mapa rejestrów Core2 (adres urządzenie podrzędne = 2):

Offset	Nazwa	Zakres	Znaczenie
0	NASTAWA	0–100	nastawa zapisywana przez Core
1	TRYB	0–2	0=ZATRZYMANIE, 1=PRACA, 2=AUTOMAT
2	STAN	0–3	0=OK, 1=ALARM, 2=COMM_ERROR, 3=ZAJETY; logicznie tylko do odczytu
3	LICZNIK	0–65535	licznik aktualizowany lokalnie przez Core2

W ćwiczeniu używaj **offsetów 0,1,2,3**. Niektóre programy diagnostyczne prezentują rejestry przechowujące (Holding Registers) w notacji 40001, 40002 itd.

RS485 i terminacja

Na stanowisku laboratoryjnym stosuj topologię magistralową, krótkie odgałęzienia i zgodne parametry transmisji. Rezystory terminujące 120 Ω stosuje się na obu fizycznych końcach dłuższej magistrali; przed ich dodaniem sprawdź, czy moduł nie posiada już włączonej terminacji. Wszystkie połączenia wykonuj przy wyłączonym zasilaniu.

Uwaga

Nie łącz bezpośrednio sygnałów TTL UART z liniami A/B RS485. Do RS485 wykorzystuj dedykowany transceiver dostępny na Demo Board i właściwe zaciski magistrali.

Przebieg ćwiczenia

Warstwy komunikacji przewodowej

dane aplikacji → protokół (tekstowy / Modbus RTU) → UART →
nadajnik-odbiornik RS485 → A/B → drugi węzeł

Zadanie 1. Konfiguracja UART

Skonfiguruj kanał komunikacji szeregowej z parametrami wskazanymi przez prowadzącego, np. 9600 lub 115200 bit/s, 8 bitów danych, bez parzystości, 1 bit stopu. Wysyłaj co sekundę licznik ramek.

Sposób realizacji i weryfikacja

1. Ustaw identyczne parametry po obu stronach: prędkość, 8 bitów danych, parzystość i bit stopu; zapisz konfigurację jako 8N1, jeżeli używasz braku parzystości i jednego bitu stopu.
2. Nadaj ramkę testową zawierającą numer sekwencyjny i znak końca linii; odbiornik powinien jednoznacznie rozdzielać kolejne komunikaty.
3. Porównaj zachowanie dla prawidłowej i celowo błędnej prędkości transmisji, aby rozpoznać typowe objawy niezgodnej konfiguracji.

Pseudokod / zapis algorytmu:

```
co 1 s:
    wyslij("RAMKA:" + nr_ramki + "\n")
    nr_ramki = nr_ramki + 1
```

Zadanie 2. Odbiór i odesłanie wiadomości

Program powinien odbierać linię tekstu i odesłać ją z prefiksem POWTORZ:. Dodaj licznik poprawnie odebranych komunikatów.

Sposób realizacji i weryfikacja

1. Odbieraj dane do bufora aż do znaku końca linii albo do osiągnięcia ustalonej długości maksymalnej.
2. Po złożeniu pełnej wiadomości usuń znak końca linii, zwiększ licznik RX i odeślij POWTORZ:...
3. Dodaj przekroczenie czasu oczekiwania bufora: niekompletna wiadomość nie może blokować programu bez końca.

Pseudokod / zapis algorytmu:

```
dopoki dane_dostepne:
    znak = odczytaj_znak()
    if znak == KONIEC_LINII:
        przetworz(bufor); bufor = ""
    else:
        bufor += znak
```

Zadanie 3. Analizator poleceń poleceń

Zaimplementuj rozpoznawanie poleceń POBIERZ:STAN, POBIERZ:WARTOSC, USTAW:WARTOSC:n oraz ZATRZYMANIE. Dla nieznanego polecenia zwróć BLAD:POLECENIE.

Sposób realizacji i weryfikacja

1. Najpierw rozpoznaj nazwę polecenia, a dopiero potem liczbę argumentów i ich typy.
2. Każde żądanie kończy się dokładnie jedną odpowiedzią: OK, danymi albo BLAD:<powod>.
3. Nieznane lub niepełne polecenie nie może zmieniać stanu urządzenia.

Pseudokod / zapis algorytmu:

```
pola = podziel(wiadomosc, ":")
if pola[0] == "POBIERZ": ...
elif pola[0] == "USTAW": ...
elif wiadomosc == "ZATRZYMAJ": ...
else: odpowiedz("BLAD:POLECENIE")
```

Zadanie 4. Walidacja argumentu

Dla USTAW:WARTOSC:n sprawdź, czy argument jest liczbą i należy do zakresu 0–100. W przypadku błędu zwróć odpowiedni komunikat bez zmiany poprzedniej wartości.

Sposób realizacji i weryfikacja

1. Konwertuj argument dopiero po sprawdzeniu, że polecenie ma właściwą liczbę pól.
2. Po konwersji sprawdź zakres 0–100; dla wartości spoza zakresu zwróć błąd bez zmiany poprzedniej nastawy.
3. Przetestuj: 0, 100, -1, 101, pusty argument oraz tekst zamiast liczby.

Pseudokod / zapis algorytmu:

```
nowa_wartosc = odczytaj_liczbe(pole)
if blad_konwersji: BLAD:TYP
elif nowa_wartosc < 0 or nowa_wartosc > 100: BLAD:ZAKRES
else: wartosc = nowa_wartosc; OK
```

Zadanie 5. RS485

Uruchom wymianę danych przez tor RS485. Jeden węzeł pełni rolę nadrzędną i wysyła żądanie, drugi odpowiada. Jeżeli dostępny jest tylko jeden węzeł RS485, prowadzący może zapewnić drugi terminal lub konwerter.

Sposób realizacji i weryfikacja

1. Połącz dwa węzły przewodami A/B zgodnie z oznaczeniami zastosowanych transceiverów; dla Core2 użyj zewnętrznego interfejsu RS485/UART, nie montuj Core2 mechanicznie na Demo Board.
2. Uruchom najpierw prostą wymianę żądanie–odpowiedź bez Modbus: Core wysyła żądanie, Core2 odpowiada. Sprawdź, czy tylko jeden węzeł nadaje w danej chwili.
3. Dodaj licznik zapytań i odpowiedzi; brak odpowiedzi w ustalonym czasie traktuj jako błąd komunikacji.

Pseudokod / zapis algorytmu:

```
CORE: wyslij(zadanie); czekaj_na_odpowiedz(limit_czasu)
CORE2: if odebrano_zadanie: przetworz(); wyslij(odpowiedz)
```

Zadanie 5A. Mapa rejestrów Modbus RTU

Skonfiguruj Core2 jako urządzenie podrzędne Modbus (slave/server) o adresie 2 i odwzoruj cztery rejestry z tabeli we wstępie. Na ekranie Core2 pokaż bieżące wartości NASTAWA, TRYB, STAN i LICZNIK. Rejestr LICZNIK zwiększaj lokalnie raz na sekundę, aby urządzenie nadrzędne mogło odczytywać zmieniającą się wartość.

Sposób realizacji i weryfikacja

1. Uruchom warstwę RS485 z identycznymi parametrami po obu stronach, np. 9600 bit/s, 8N1.
2. Zainicjalizuj urządzenie podrzędne Modbus (slave/server) pod adresem 2 i przypisz zmienne programu do offsetów 0–3.
3. Ustaw STAN=0 i TRYB=0 po starcie. Program powinien rozpocząć pracę w stanie bezpiecznym ZATRZYMANIE.
4. Zmień lokalnie LICZNIK; potwierdź na ekranie Core2, że zmienna rzeczywiście się zmienia przed testem odczytu z urządzenia nadrzędnego.

Zadanie 5B. Odczyt i zapis rejestrów Core–Core2

Skonfiguruj Core jako urządzenie nadrzędne Modbus (master/client). Wykonaj cykliczny odczyt rejestrów Core2 kodem funkcji 03 oraz zapis NASTAWA lub TRYB kodem funkcji 06. Po każdej operacji wyświetl wynik i zwiększ odpowiedni licznik komunikacji.

Sekwencja testowa:

1. Nadrzędny: odczytaj rejestry 0..3 z urządzenia podrzędnego 2
2. Podrzędny: zwróć NASTAWA, TRYB, STAN, LICZNIK
3. Nadrzędny: zapisz rejestr 0 = 50
4. Nadrzędny: odczytaj rejestr 0 i sprawdź wartość == 50
5. Nadrzędny: zapisz rejestr 1 = 1 (PRACA)
6. Podrzędny: sprawdź polecenie i aktualizuj GUI

Testy obowiązkowe

Wykonaj: poprawny odczyt, poprawny zapis, odczyt z błędnego adresu urządzenia podrzędne, przerwę w przewodzie/wyłączenie drugiego węzła oraz próbę zapisu niedozwolonej wartości, np. NASTAWA=150. Niedozwolona wartość nie może zostać użyta przez logikę sterowania nawet wtedy, gdy warstwa protokołu dostarczy ją poprawnie.

Zadanie 6. Przekroczenie czasu oczekiwania i diagnostyka

Dodaj licznik czasu od ostatniej poprawnej ramki. Jeżeli przez ustalony czas nie pojawi się komunikat, ustaw stan BŁĄD KOMUNIKACJI. Na ekranie pokazuj liczbę odebranych ramek, liczbę błędów i czas od ostatniej ramki.

Sposób realizacji i weryfikacja

1. Mierz czas od wysłania żądania do poprawnej odpowiedzi oraz czas od ostatniej poprawnej ramki.
2. Rozróżnij co najmniej: przekroczenie czasu oczekiwania, błędny adres/funkcję, niepoprawną wartość oraz błąd CRC Modbus.
3. Po utracie komunikacji przejdź do bezpiecznego stanu i wymagaj ponownego URUCHOM po odzyskaniu łącza.

Pseudokod / zapis algorytmu:

```
if czas_biezacy - ostatnia_poprawna > limit_czasu_komunikacji:
    stan_komunikacji = BŁĄD_KOMUNIKACJI
    bezpieczne_zatrzymanie()
```

Wymagany rezultat

Dwa węzły wymieniają polecenia i odpowiedzi według ustalonego formatu, błędne polecenia nie zmieniają stanu urządzenia, a utrata komunikacji jest wykrywana.

Zadanie końcowe**Integracja zadań etapowych**

Zadanie końcowe należy wykonać przez rozbudowę jednego toru komunikacyjnego przygotowanego w zadaniach 1–6. Konfiguracja UART z zadania 1 pozostaje podstawą transmisji, buforowanie z zadania 2 jest wykorzystywane przez analizator poleceń z zadania 3, walidacja z zadania 4 zabezpiecza dane, zadanie 5 przenosi wymianę na RS485, zadania 5A–5B dodają Modbus RTU, a zadanie 6 uzupełnia całość o diagnostykę i bezpieczne zachowanie po przekroczeniu czasu oczekiwania.

Uruchom dwa warianty sterowania tym samym stanowiskiem po RS485. W pierwszym użyj polskiego protokołu tekstowego, np. `POBIERZ:STAN`, `POBIERZ:WARTOSC`, `USTAW:WARTOSC:n`, `ZATRZYMAJ`. W drugim użyj Modbus RTU pomiędzy Core jako urządzeniem nadrzędnym i Core2 jako urządzeniem podrzędnym. Rejestry nazwij `NASTAWA`, `TRYB`, `STAN` i `LICZNIK`. Core ma cyklicznie odczytywać rejestry, zapisywać nastawę 0–100 i tryb pracy oraz wykrywać przekroczenie czasu oczekiwania. Core2 waliduje wartości i po utracie komunikacji przechodzi do `ZATRZYMANIE` oraz wyświetla **BŁĄD KOMUNIKACJI**.

Rozszerzenie: licz oddzielnie poprawne zapytania, poprawne odpowiedzi i błędy oraz pokaż czas od ostatniej poprawnej wymiany.

Kryteria zaliczenia

- poprawna konfiguracja transmisji,
- nadawanie i odbiór danych tekstowych,
- analizator poleceń co najmniej czterech poleceń,
- walidacja argumentów,
- uruchomienie toru RS485,
- poprawna konfiguracja Modbus RTU Core–Core2 i mapują rejestrów,
- odczyt FC03 oraz zapis co najmniej jednego rejestru (np. FC06),
- diagnostyka i przekroczenie czasu oczekiwania komunikacji.

Sprawozdanie

Podstawą zaliczenia ćwiczenia jest sprawozdanie dotyczące wykonania zadania końcowego. Zadania etapowe są zaliczane na zasadzie demonstracji działania programu na stanowisku. Sprawozdanie należy przesłać w formie pliku **.pdf* oraz pliku projektu programu do weryfikacji. W dokumencie należy przedstawić:

- kod programu w UIFlow,
- zdjęcia działającego stanowiska lub czytelne zrzuty ekranu,
- krótki opis zastosowanego algorytmu wraz ze schematem blokowym, diagramem stanów lub pseudokodem,

- opis sposobu testowania programu,
- wnioski.