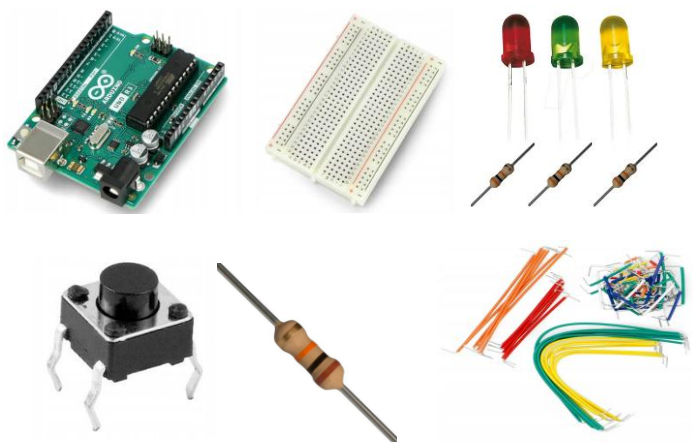
Ćwiczenie 31

Temat: Logika PLC – Układ podtrzymania i sekwencja procesu

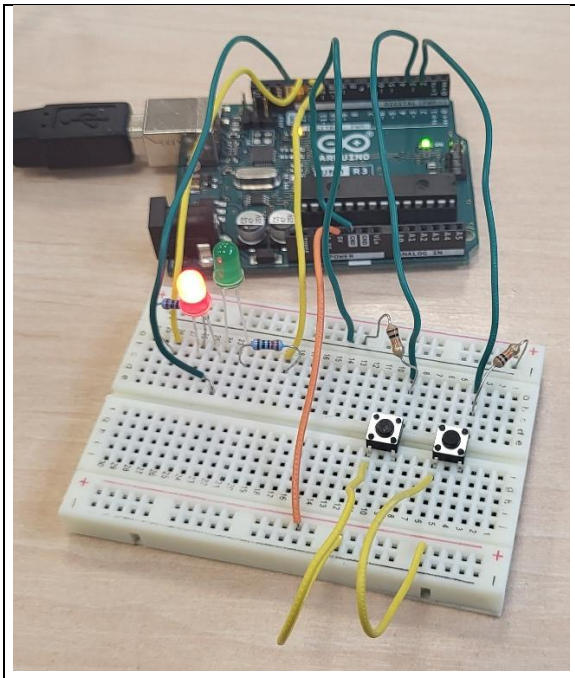
Cel ćwiczenia: Zrozumienie fundamentalnej koncepcji ze świata PLC: układu **podtrzymania** (zatrzasku). Nauczysz się, jak napisać program, w którym jeden przycisk (**START**) trwale uruchamia system, a drugi przycisk (**STOP**) go wyłącza, niezależnie od wszystkiego innego. Następnie rozbudujemy to o prostą sekwencję, tworząc automat stanów – serce każdego programu PLC.



Potrzebne komponenty:

- Płytka Arduino UNO i płytka stykowa
- 3x Diody LED (np. zielona, żółta, czerwona)
- 3x Rezystory 220Ω (do diod LED)
- 3x Przyciski (tactile switch)
- 3x Rezystory 10kΩ (do przycisków, jako pull-down)
- Przewody połączeniowe

Część 1: Podstawowy obwód i kod - Układ Podtrzymania (START/STOP)



W tej części zbudujemy najważniejszy obwód bezpieczeństwa w automatyce. Program nie będzie sterował diodą bezpośrednio z przycisku. Zamiast tego, przycisk będzie zmieniał stan **wewnętrznej zmiennej w pamięci**, a diody będą świecić w zależności od tej zmiennej.

Obwód

1. **Diody LED:**
   - Podłącz diodę **zieloną** (przez rezystor 220Ω) do pinu **12**.
   - Podłącz diodę **czerwoną** (przez rezystor 220Ω) do pinu **13**.
   - Katody obu diod podłącz do **GND**.
2. **Przyciski (z rezystorami pull-down):**
   - Podłącz przycisk **START** do pinu **2**.
   - Podłącz przycisk **STOP** do pinu **3**.
   - Dla obu przycisków: jeden róg połącz z **5V**, a drugi róg (po przekątnej) połącz jednocześnie z pinem cyfrowym Arduino (2 lub 3) oraz przez rezystor **10kΩ** z **GND**.

Kod (C++)

Program wprowadza kluczową zmienną ***systemGotowy***, która działa jak "pamięć" lub "cewka podtrzymująca" w sterowniku PLC.

```
// Definicje pinów
const int pinStart = 2;
const int pinStop = 3;
const int pinLedZielona = 12; // Sygnalizacja pracy
const int pinLedCzerwona = 13; // Sygnalizacja zatrzymania

// Zmienna globalna przechowująca stan systemu. To jest nasz "zatrzask" PLC.
bool systemGotowy = false;

void setup() {
  // Definiujemy wejścia (przyciski)
  pinMode(pinStart, INPUT);
  pinMode(pinStop, INPUT);

  // Definiujemy wyjścia (diody)
  pinMode(pinLedZielona, OUTPUT);
  pinMode(pinLedCzerwona, OUTPUT);
}

void loop() {
  // === 1. Odczyt Wejść ===
  // Odczytujemy stan przycisków na początku każdej pętli
  bool startNacisniety = digitalRead(pinStart);
  bool stopNacisniety = digitalRead(pinStop);

  // === 2. Przetwarzanie Logiki (Serce PLC) ===
  // Logika "SET" (Ustaw): Jeśli naciśnięto START, system ma być gotowy.
  if (startNacisniety == HIGH) {
    systemGotowy = true;
  }

  // Logika "RESET" (Resetuj): Jeśli naciśnięto STOP, system ma być zatrzymany.
  // Logika STOP zawsze ma priorytet nad START.
  if (stopNacisniety == HIGH) {
    systemGotowy = false;
  }

  // === 3. Aktualizacja Wyjść ===
  // Na samym końcu pętli, aktualizujemy diody na podstawie stanu naszej pamięci.
  // Ta sekcja jest oddzielona od odczytu przycisków!
  if (systemGotowy == true) {
    // Stan PRACY: zielona świeci, czerwona zgaszona
    digitalWrite(pinLedZielona, HIGH);
    digitalWrite(pinLedCzerwona, LOW);
  } else {
    // Stan STOP: zielona zgaszona, czerwona świeci
    digitalWrite(pinLedZielona, LOW);
    digitalWrite(pinLedCzerwona, HIGH);
  }
```

```
    // Pętla loop() wykonuje się bardzo szybko, ciągle skanując wejścia i aktualizując wyjścia.
  // To jest dokładnie tak, jak działa cykl skanu PLC.
}
```

Kluczowe wyjaśnienie: Zauważ, że naciśnięcie przycisku START nie zapala diody! Ono tylko **ustawia zmienną** *systemGotowy* na *true*. Dopiero oddzielna część kodu sprawdza tę zmienną i decyduje o zapaleniu diod. To jest fundamentalna różnica między programowaniem Arduino a logiką PLC.

Część 2: Samodzielna rozbudowa - Symulator Linii Produkcyjnej (Automat Stanów)

Teraz, gdy mamy już działający układ START/STOP, zamienimy go w prosty symulator linii, np. napełniania butelek. Wykorzystamy do tego **automat stanów** (State Machine), czyli centralną koncepcję programowania sekwencyjnego w TIA Portal (GRAPH/SFC).

Obwód:

- Dodaj do układu:
  - Przycisk "**CZUJNIK**" (symuluje butelkę na miejscu) na pinie **4** (z rezystorem 10kΩ).
  - Diodę żółtą (symuluje "proces napełniania") na pinie **11** (z rezystorem 220Ω).

Zadanie 31.1: Wprowadzenie Stanów (State Machine)

- Stwórz nową zmienną globalną *int stanSystemu = 0;*.
- Zdefiniujemy 3 stany: 0 (Zatrzymany), 1 (Gotowy/Oczekiwanie), 2 (Napełnianie).
- Zmodyfikuj kod tak, aby przycisk START ustawiał *stanSystemu* na 1, a STOP zawsze ustawiał na 0.

Zadanie 31.2: Logika Stanów (Przejścia)

- W funkcji *loop()* stwórz strukturę *switch (stanSystemu)* lub serię *if...else if*....
- **Stan 0 (ZATRZYMANY)**: Świeci się dioda **czerwona**. Jeśli naciśnięto START, przejdź do *stanSystemu = 1*.
- **Stan 1 (GOTOWY)**: Świeci się dioda **zielona**. Program czeka. Jeśli naciśnięto CZUJNIK (pin 4), *przejdź do stanSystemu = 2*.
- **Stan 2 (NAPEŁNIANIE)**: Świeci się dioda **żółta**. W tym stanie program ma symulować napełnianie przez **3 sekundy**. Po upływie tego czasu, program musi **automatycznie** wrócić do *stanSystemu = 1* (Gotowy) i czekać na kolejną "butelkę".
- **Logika STOP**: Przycisk STOP musi działać **ZAWSZE**, niezależnie od stanu. Najlepiej umieścić odczyt przycisku STOP na samym początku pętli *loop()*, aby natychmiast ustawiał *stanSystemu = 0*, przerywając każdy inny proces.

Zadanie 31.3: Timer bez blokowania (Klucz do PLC!)

- W zadaniu 31.2 musisz zrealizować 3-sekundowe opóźnienie w Stanie 2.
- **ABSOLUTNIE NIE WOLNO** używać funkcji *delay(3000);*! Zatrzymanie programu (tzw. "blocking code") jest niedopuszczalne w PLC, ponieważ w tym czasie sterownik nie czytałby przycisku STOP.
- Musisz użyć logiki opartej o funkcję *millis()*, tak jak robi to timer **TON** w TIA Portal.
- **Podpowiedź**: Będziesz potrzebował/a dodatkowej zmiennej globalnej *unsigned long czasStartuNapelniania;*. Kiedy wchodzisz do Stanu 2, zapisz w niej aktualny czas: *czasStartuNapelniania = millis();*. Następnie, będąc w Stanie 2, ciągle sprawdzaj warunek:
  if (millis() - czasStartuNapelniania >= 3000) { ... }.

Zadanie 31.4: Licznik (Counter)

- Stwórz zmienną *int licznikButelek = 0;*.
- Za każdym razem, gdy proces napełniania (Stan 2) się zakończy, zwiększ licznik o 1.
- Gdy *licznikButelek* osiągnie wartość 5, program ma automatycznie przejść do Stanu 0 (STOP) i zapalić **czerwoną** diodę, symulując pełną paletę i konieczność interwencji operatora (wciśnięcia START, aby zresetować licznik i zacząć od nowa). To jest bezpośredni odpowiednik bloku licznika **CTU (Count Up)**.

Pliki: [zad31-1.INO, zad31-2.INO, zad31-3.INO, zad31-4.INO] przesłać na haukeprojekt@gmail.com ,
w tytule podać „**3XY(Z)_cwiczenie31**” gdzie X=klasa, Y=grupa, Z=numer_z_dziennika

31.x – Rozszerzone instrukcje:

Do Zadania 31.1: Wprowadzenie Stanów (State Machine)

- Pomyślcie o zmiennej *int stanSystemu* jak o sygnalizacji na skrzyżowaniu. W danej chwili może świecić się tylko jedno światło (być tylko w jednym stanie).
- Wasza poprzednia zmienna *bool systemGotowy* była automatem o dwóch stanach: *true* i *false*. Teraz po prostu zamieniacie ją na zmienną typu *int*, która może przechowywać *więcej* niż dwa stany (np. 0, 1, 2).
- Zamiast *systemGotowy = true;* będziecie pisać *stanSystemu = 1;*.
- Zamiast *systemGotowy = false;* będziecie pisać *stanSystemu = 0;*.

Do Zadania 31.2: Logika Stanów (Przejścia)

- Najlepszą strukturą do zarządzania stanami w C++ jest *switch...case*. W funkcji *loop()* wasz kod powinien mieć taki szkielet:

```
// Przycisk STOP musi być sprawdzany jako pierwszy i mieć priorytet!
if (digitalRead(pinStop) == HIGH) {
  stanSystemu = 0;
}

// Teraz sprawdzamy, co robić w zależności od aktualnego stanu
switch (stanSystemu) {

  case 0: // Stan ZATRZYMANY
    // Ustaw diodę czerwoną na HIGH, inne na LOW
    // Sprawdź, czy naciśnięto START
    if (digitalRead(pinStart) == HIGH) {
      stanSystemu = 1; // Zmień stan na GOTOWY
    }
    break; // Ważne! Zakończ ten stan

  case 1: // Stan GOTOWY
    // Ustaw diodę zieloną na HIGH, inne na LOW
    // Sprawdź, czy naciśnięto CZUJNIK
    if (digitalRead(pinCzujnik) == HIGH) {
      stanSystemu = 2; // Zmień stan na NAPEŁNIANIE
    }
    break;

  case 2: // Stan NAPEŁNIANIE
    // Ustaw diodę żółtą na HIGH, inne na LOW
    // Tutaj przyjdzie kod z zadania 31.3 (timer)
    // ...
    break;
}
```

Do Zadania 31.3: Timer bez blokowania (Klucz do PLC!)

To najtrudniejsza, ale najważniejsza część. Chcecie odmierzyć 3 sekundy, ale **nie możecie** zatrzymać pętli *loop()* za pomocą *delay()*.

1. Stwórzcie **na górze programu** (jako zmienną globalną) "stoper":
   unsigned long czasStartuNapelniania = 0;

2. Musicie "włączyć" stoper dokładnie w momencie przejścia ze Stanu 1 do Stanu 2. Zmodyfikujcie swój *case 1:*

```
case 1: // Stan GOTOWY
  // ...
  if (digitalRead(pinCzujnik) == HIGH) {
    stanSystemu = 2; // Zmień stan
    czasStartuNapelniania = millis(); // ZAPISZ CZAS STARTU!
  }
  break;
```

3. Teraz w *case 2* musicie ciągle sprawdzać, czy minęły 3 sekundy. Zróbcie to, odejmując czas startu od czasu obecnego:

```
case 2: // Stan NAPEŁNIANIE
  digitalWrite(pinLedZolta, HIGH); // ... (inne diody LOW)

  // Ciągle sprawdzaj zegarek
  if (millis() - czasStartuNapelniania >= 3000) {
    // Minęły 3 sekundy!
    stanSystemu = 1; // Wróć do stanu GOTOWY
  }
  break;
```

Do Zadania 31.4: Licznik (Counter)

1. Stwórzcie **globalną** zmienną licznika: *int licznikButelek = 0;*

2. Gdzie należy zliczać butelki? W **Stanie 2**, dokładnie w tym momencie, gdy timer (*millis()*) zakończył odliczanie 3 sekund. To oznacza, że napełnianie się udało.

```
case 2:
  // ...
  if (millis() - czasStartuNapelniania >= 3000) {
    licznikButelek++; // Zwiększ licznik o 1

    // Sprawdź, czy licznik osiągnął limit
    if (licznikButelek >= 5) {
      stanSystemu = 0; // Zatrzymaj linię (pełna paleta)
    } else {
      stanSystemu = 1; // Wróć do stanu GOTOWY
    }
  }
  break;
```

3. Nie zapomnijcie **wyzerować licznika**! Gdzie to zrobić? Najlepiej wtedy, gdy operator resetuje linię, czyli po naciśnięciu START w Stanie 0.

```
case 0:
  // ...
  if (digitalRead(pinStart) == HIGH) {
    stanSystemu = 1;
    licznikButelek = 0; // ZERUJ LICZNIK przy nowym starcie
  }
  break;
```