Zestawienie funkcji czasowych i timerów w Arduino

Marcin Brzozowski 5TP

1. Wbudowane funkcje czasowe

- *delay(ms)* - wstrzymuje wykonywanie programu przez nadaną ilość milisekund. Blokuje wykonywanie innych instrukcji takich jak odczyty z sensorów czy obliczenia. Z tego powodu jest zazwyczaj unikana przy tworzeniu programów. Poniżej widzimy program (źródło), który cykliczne włącza i wyłącza diodę co 1s.

```
int ledPin = 13;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

Listing 1: Przykład programu z użyciem funkcji *delay*

- *millis()* - zwraca ilość czasu (w milisekundach), która mineła od uruchomienia mikrokontrolera. Sama w sobie nie pozwala na zarządzanie instrukcjami w czasie, ale może być do tego wykorzystana. Poniżej program (źródło) analogiczny do poprzedniego, tym razem wykorzystujący funkcję *millis*.

```
const int ledPin = LED_BUILTIN;
int ledState = LOW;
unsigned long previousMillis = 0;
const long interval = 1000;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    if (ledState == LOW) {
      ledState = HIGH;
    } else {
      ledState = LOW;
    }

    digitalWrite(ledPin, ledState);
  }
}
```

Listing 2: Przykład programu z użyciem funkcji *millis*

Porównanie programów

Pierwszy program jak to wynika z natury funkcji delay wstrzymuje wykonywanie innych instrukcji i nie robi nic przez 1s natomiast drugi program drugi nie blokuje wykonywania innych instrukcji, więc w jego trakcie możemy wykonywać także inne niezwiązane z czekaniem operacje.

2. Timery

Wstęp. Układ Atmega-328P posiada trzy timery oznaczone kolejno timer0, timer1 i timer2. Timer0 jest timerem 8-bitowym jest wykorzystywany w funkcjach takich jak *delay* czy *millis*. Timer1 jest timerem 16-bitowym i wykorzystwany jest przez bibliotekę *servo*. Timer2 jest timerem 8-bitowym wykorzystywanym przez funkcję *tone*. Timery 1 oraz 2 możemy także skonfigurować do własnych potrzeb za pomocą zmian w rejestrze, podobnie możemy postapić z *timer0*, lecz nie jest to zalecane w związku z jego silnym powiązaniem z funkcjami wbudowanymi.

2.1. Biblioteka Timers.h

Dokumentacja biblioteki: https://github.com/centaq/arduino-simple-timers

Biblioteka *Timers.h* udostępnia nam prosty interfejs do pracy z operacjami cyklicznymi. Poniżej już trzeci przykład programu "BLINK" tym razem z wykorzystaniem wspomnianej biblioteki.

```
#include "Timers.h"

Timers ledTimer;
const int ledPin = LED_BUILTIN;
int ledState = LOW;
const long interval = 1000;


void setup() {
    timer.start(interval);
}

void loop() {
  if (timer.available()) {
    timer.stop();

    if (ledState == LOW) {
        ledState = HIGH;
    } else {
        ledState = LOW;
    }

    digitalWrite(ledPin, ledState);

    timer.start(interval);
  }
}
```

Listing 3: Przykład programu z użyciem biblioteki *Timers.h*

3. Podsumowanie

Podsumowując, delay() jest prostym sposobem na zatrzymanie programu na określony czas, ale może spowodować opóźnienia w wykonywaniu innych operacji. millis() pozwala na planowanie operacji w oparciu o upływ czasu, bez blokowania reszty programu. Timery natomiast oferują zaawansowaną kontrolę nad czasem i pozwalają na bardziej skomplikowane operacje czasowe, takie jak przerwania czy generowanie sygnałów PWM. Wybór metody zależy od konkretnych wymagań projektu.