

Podstawy Systemów Wbudowanych

Laboratorium

Laboratorium 1: GPIO, przerwania i PWM

Autorzy: Angelika Tefelska, Dariusz Tefelski

Spis treści

1	Wprowadzenie	1
2	Wyposażenie stanowiska	1
3	Zadanie 0: Instalacja środowiska Arduino IDE	1
4	Zadanie 1: Wyjścia cyfrowe i diody LED	2
5	Zadanie 2: Wejście cyfrowe i przerwanie	3
6	Zadanie 3: Modulacja szerokości impulsu PWM	7
7	Zadanie 4: Programowanie nieblokujące	8
8	Zadanie końcowe: Sterownik sygnalizacji świetlnej	9
9	Podsumowanie	10

1 Wprowadzenie

Cel ćwiczenia



Celem ćwiczenia jest przećwiczenie podstaw pracy z wejściami i wyjściami mikrokontrolera na przykładzie Arduino UNO R4 WiFi oraz wprowadzenie dwóch mechanizmów ważnych w systemach wbudowanych: obsługi zdarzeń za pomocą przerwań i programowania nieblokującego.

Najważniejsza idea tego laboratorium



W prostych przykładach Arduino często stosuje się funkcję `delay()`. W rzeczywistym systemie wbudowanym długie opóźnienie blokuje jednak wykonywanie innych zadań. Dlatego podczas tego laboratorium najpierw wykorzystamy `delay()` do prostych testów, a następnie zastąpimy ją programowaniem opartym na czasie i zdarzeniach.

2 Wyposażenie stanowiska

Do wykonania ćwiczenia potrzebne są:

- Arduino UNO R4 WiFi,
- przewód USB,
- płytki stykowa,
- diody LED,
- rezystory,
- przycisk,
- przewody połączeniowe męsko-męskie.

Co należy zapamiętać?



Arduino UNO R4 WiFi pracuje z logiką 5 V, ale dopuszczalny prąd pojedynczego pinu GPIO jest mniejszy niż w wielu starszych płytkach Arduino. W tym laboratorium przyjmij prąd diody nie większy niż około 5 mA. Nie podłączaj diody LED bez rezystora ograniczającego prąd.

3 Zadanie 0: Instalacja środowiska Arduino IDE

W celu zainstalowania środowiska Arduino IDE:

- Wejdź na stronę: <https://www.arduino.cc> a następnie wybierz: *Products* → *Software* → *Arduino IDE*.

- Wybierz **Linux AppImage** i kliknij *Download*.
- Uruchom terminal i utwórz katalog o nazwie *Arduino* (przypomnienie: komenda tworząca katalog to *mkdir nazwa_katalogu*).
- Skopiuj z katalogu *Downloads* pobrany obraz. Komenda powinna wyglądać mniej więcej tak: *cp ../Downloads/arduino-ide-2.3.10-Linux-64bit.AppImage .*
- Nadaj uprawnienia do wykonywania się obrazu:
chmod +x arduino-ide-2.3.10-Linux-64bit.AppImage
- Uruchom obraz: *./arduino-ide-2.3.10-Linux-64bit.AppImage*
- Następnie wybierz *Arduino UNO R4 WiFi* jako board.
- Oprogramowanie zapyta się czy zainstalować niezbędne biblioteki w prawym dolnym rogu ekranu. Zgódź się.

4 Zadanie 1: Wyjścia cyfrowe i diody LED

Dioda LED przewodzi prąd głównie w jednym kierunku. Dłuższa nóżka diody LED to anoda, a krótsza to katoda. Do ograniczenia prądu stosujemy rezystor szeregowy.

Wartość rezystora można oszacować ze wzoru:

$$R = \frac{U_Z - U_D}{I_D}, \quad (1)$$

gdzie U_Z oznacza napięcie zasilania, U_D spadek napięcia na diodzie, a I_D przyjęty prąd diody.

Zadanie 1.1 — Dobór rezystora i pierwszy program



Przyjmij:

- $U_Z = 5\text{ V}$,
- $U_D = 1,7\text{ V}$,
- $I_D = 5\text{ mA}$.

Oblicz minimalną wartość rezystora, a następnie wybierz najbliższą większą wartość dostępną na stanowisku.

Podłącz diodę LED do pinu D8 przez dobrany rezystor. Ponieważ jest to pierwszy program wykonywany na Arduino, rozpocznij od gotowego przykładu:

```
1  const int LED_PIN = 8;
2
3  void setup() {
4      pinMode(LED_PIN, OUTPUT);
5  }
6
7  void loop() {
```



```
9      digitalWrite(LED_PIN, HIGH);  
      delay(500);  
10  
11      digitalWrite(LED_PIN, LOW);  
12      delay(500);  
13  }
```

Wgraj program do płytki i sprawdź jego działanie. Następnie zmień wartości opóźnień i sprawdź, jak wpływają one na sposób migania diody.

Jak zbudowany jest program Arduino?



Każdy podstawowy program Arduino zawiera dwie szczególnie ważne funkcje:

- `setup()` — wykonuje się jeden raz po uruchomieniu lub zresetowaniu płytki; umieszcza się w niej m.in. konfigurację pinów,
- `loop()` — po zakończeniu `setup()` wykonywana jest w pętli bez końca.

Wykorzystane wyżej funkcje mają następujące znaczenie:

- `pinMode(pin, OUTPUT)` — konfiguruje pin jako wyjście,
- `digitalWrite(pin, HIGH)` — ustawia na wyjściu stan wysoki,
- `digitalWrite(pin, LOW)` — ustawia na wyjściu stan niski,
- `delay(ms)` — zatrzymuje wykonywanie programu na podaną liczbę ms.

Zadanie 1.2 — Trzy diody



Podłącz trzy diody LED do pinów D8, D9 i D10. Każda dioda musi mieć własny rezystor ograniczający prąd.

Napisz program realizujący kolejno dwa tryby pracy:

1. wszystkie diody jednocześnie zmieniają stan co 500 ms,
2. diody zapalają się kolejno jedna po drugiej co 100 ms, a po zapaleniu ostatniej sekwencja rozpoczyna się od początku.

5 Zadanie 2: Wejście cyfrowe i przerwanie

Przycisk może być odczytywany przez cykliczne sprawdzanie stanu wejścia w pętli programu. Taki sposób nazywamy *pollingiem*. Alternatywnie mikrokontroler może zareagować na zmianę sygnału automatycznie, wywołując przerwanie.

Po co jest rezystor podciągający?

Wejście cyfrowe mikrokontrolera ma bardzo dużą rezystancję wejściową. Jeżeli nie zostanie połączone ani ze stanem wysokim, ani z masą, jego potencjał może być nieokreślony. Takie wejście nazywamy *plywającym* (*floating input*). Może ono przypadkowo zmieniać odczytywany stan pod wpływem zakłóceń.

Rezystor podciągający (*pull-up*) wymusza zdefiniowany stan logiczny wtedy, gdy przycisk jest rozłączony. Arduino umożliwia włączenie wewnętrznego rezystora podciągającego za pomocą konfiguracji:

```
1 pinMode(BUTTON_PIN, INPUT_PULLUP);
```

Przycisk podłączamy wtedy pomiędzy wejście a masę. W stanie spoczynku rezystor podciąga wejście do napięcia zasilania, dlatego odcytujemy HIGH. Po naciśnięciu przycisku wejście zostaje zwarte z masą i odcytujemy LOW. Oznacza to, że przy takim podłączeniu przycisk jest **aktywny stanem niskim**.

Zadanie 2.1 — Polling



Podłącz przycisk do pinu D2. W programie zdefiniuj numer pinu i skonfiguruj go jako wejście z wewnętrznym rezystorem podciągającym:

```
1 const int BUTTON_PIN = 2;
2
3 void setup() {
4     pinMode(BUTTON_PIN, INPUT_PULLUP);
5 }
```

Funkcja

`digitalRead(pin)`

odczytuje aktualny stan logiczny wskazanego wejścia i zwraca wartość HIGH albo LOW.

Napisz program, w którym:

- naciśnięcie przycisku włącza diodę na pinie D8,
- kolejne naciśnięcie przycisku gasi diodę.

Sprawdź, jaki stan logiczny zwraca `digitalRead()` dla przycisku nienaciśniętego i naciśniętego.

Drgania styków przycisku

Styki mechanicznego przycisku nie zwierają się idealnie w jednej chwili. Przez kilka lub kilkadziesiąt milisekund mogą wielokrotnie łączyć się i rozłączać. Mikrokontroler jest znacznie szybszy od człowieka, dlatego jedno fizyczne naciśnięcie może zostać odczytane jako kilka kolejnych zmian stanu.

Jedną z najprostszych metod programowej eliminacji drgań styków jest dwukrotne sprawdzenie stanu przycisku w niewielkim odstępie czasu:

```
1 int state1 = digitalRead(BUTTON_PIN);
2 delay(20);
3 int state2 = digitalRead(BUTTON_PIN);
4
5 if (state1 == state2) {
6     // nacisnieto przycisk
```

Zadanie 2.2 — Eliminacja drgań styków



Zmodyfikuj program z zadania 2.1 tak, aby stan przycisku był akceptowany dopiero wtedy, gdy dwa odczyty wykonane w odstępie 20 ms są takie same. Sprawdź działanie programu przy szybkim naciskaniu i zwalnianiu przycisku. W tym miejscu dopuszczamy krótkie użycie `delay()`. W dalszej części laboratorium pokażemy, jak unikać blokowania programu.

Czym jest przerwanie?

W przypadku pollingu funkcja `loop()` musi stale sprawdzać, czy przycisk zmienił stan. Przerwanie działa inaczej. Gdy wystąpi skonfigurowane zdarzenie sprzętowe, mikrokontroler na chwilę przerywa wykonywanie bieżącego fragmentu programu, uruchamia specjalną funkcję obsługi przerwania, a po jej zakończeniu wraca do wcześniej wykonywanego kodu.

Funkcję obsługi przerwania nazywamy ISR (*Interrupt Service Routine*).

Jak intuicyjnie zrozumieć przerwanie? Wyobraź sobie, że czytasz książkę i w pewnym momencie dzwoni telefon. Przestajesz czytać książkę, odbierasz telefon i po skończonej rozmowie wracasz do czytania książki. W tym przypadku przerwanie to dzwonek telefonu, rozmowa przez telefon to wywołanie funkcji *ISR* a po jej zakończeniu dalsze czytanie to powrót do górnego nurtu wykonywania programu.

Arduino korzysta z wielu źródeł przerw, np. od timerów, interfejsów komunikacyjnych czy urządzeń peryferyjnych mikrokontrolera. W tym laboratorium zajmujemy się **przerwaniem zewnętrznymi**, czyli wywoływanymi zmianą stanu na wejściu cyfrowym.

W Arduino UNO R4 WiFi piny D2 i D3 są oznaczone jako piny przerw zewnętrznych. Przerwanie może być wywołane m.in. przez:

- **RISING** — zbocze narastające, czyli przejście LOW -> HIGH,
- **FALLING** — zbocze opadające, czyli przejście HIGH -> LOW,
- **CHANGE** — dowolną zmianę stanu: LOW -> HIGH lub HIGH -> LOW,
- **LOW** — utrzymywanie się stanu niskiego; jest to wyzwalanie poziomem, a nie zboczem.

Ponieważ w naszym układzie przycisk z `INPUT_PULLUP` po naciśnięciu zmienia stan z HIGH na LOW, naciśnięcie odpowiada zboczu **FALLING**.

Składnia funkcji `attachInterrupt()`



Do dołączenia funkcji obsługi przerwania służy:

```
1 attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);
```



Poszczególne argumenty oznaczają:

- `digitalPinToInterrupt(pin)` — zamienia numer pinu Arduino na identyfikator odpowiadającego mu przerwania,
- `ISR` — nazwa funkcji, która ma zostać wykonana po wystąpieniu przerwania; podajemy ją bez nawiasów,
- `mode` — sposób wyzwalania przerwania, np. `RISING`, `FALLING` lub `CHANGE`.

Funkcja `ISR` nie przyjmuje argumentów i nie zwraca wyniku, dlatego definiujemy ją np. jako:

```
1 void buttonISR() {  
2     // bardzo krótka obsługa zdarzenia  
3 }
```

Zadanie 2.3 — Przerwanie zewnętrzne



Zmodyfikuj program tak, aby kolejne naciśnięcia przycisku przełączały stan diody: włączona ↔ wyłączona ale tym razem do wykrycia naciśnięcia przycisku wykorzystaj przerwanie na pinie D2:

```
1 const int BUTTON_PIN = 2;  
2 volatile bool buttonEvent = false;  
3  
4 void buttonISR() {  
5     buttonEvent = true;  
6 }  
7  
8 void setup() {  
9     pinMode(BUTTON_PIN, INPUT_PULLUP);  
10    attachInterrupt(digitalPinToInterrupt(BUTTON_PIN),  
11                    buttonISR, FALLING);  
12 }
```

Funkcja obsługi przerwania powinna jedynie zgłosić wystąpienie zdarzenia. Zmianę stanu diody wykonaj w funkcji `loop()`.

Pamiętaj, że drgania styków występują także wtedy, gdy przycisk obsługiwany jest przez przerwanie.

Co należy zapamiętać?



Funkcja obsługi przerwania (ISR) powinna wykonywać się możliwie krótko. Nie umieszczaj w niej długich obliczeń ani funkcji `delay()`. Najlepiej, aby ISR jedynie zapisała informację, że zdarzenie wystąpiło, a właściwa reakcja została wykonana później w funkcji `loop()`.

Co oznacza volatile?



Kompilator podczas optymalizacji programu może założyć, że zmienna nie zmieni się, jeśli w aktualnie analizowanym fragmencie kodu nie ma instrukcji, która ją modyfikuje. Zmienna używana zarówno w programie głównym, jak i w ISR może jednak zmienić wartość w dowolnej chwili, ponieważ funkcja obsługi przerwania może zostać uruchomiona niezależnie od miejsca wykonywania programu.

Słowo kluczowe

volatile

informuje kompilator, że wartość takiej zmiennej może zmienić się w sposób nieprzewidywalny dla zwykłego przebiegu programu i należy ją za każdym razem odczytać z pamięci.

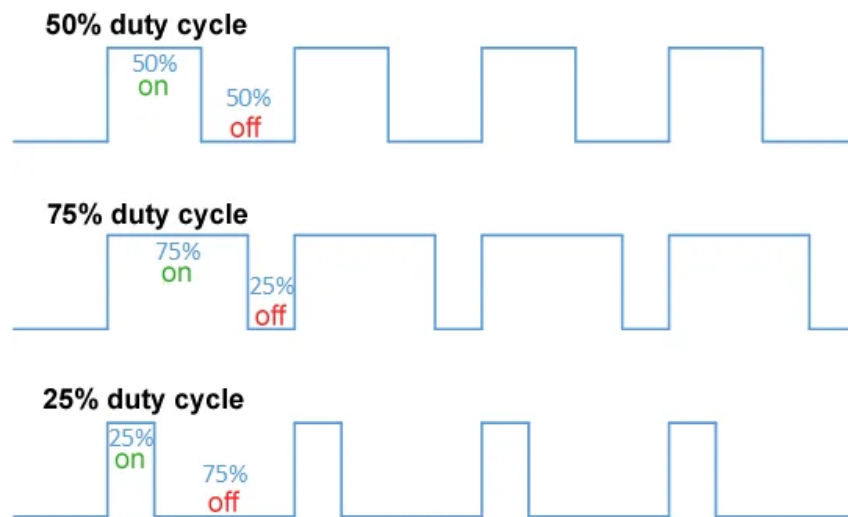
Dlatego flaga ustawiana w ISR została zadeklarowana jako:

```
1 volatile bool buttonEvent = false;
```

volatile nie zastępuje jednak poprawnego projektowania obsługi przerwań — przede wszystkim ISR nadal powinna być możliwie krótka.

6 Zadanie 3: Modulacja szerokości impulsu PWM

PWM (*Pulse Width Modulation*) polega na bardzo szybkim przełączaniu wyjścia cyfrowego pomiędzy stanami LOW i HIGH. Zmieniamy przy tym współczynnik wypełnienia, czyli część okresu, przez którą sygnał pozostaje w stanie wysokim.



Rysunek 1: Idea techniki PWM. Źródło: https://www.corsair.com/pl/pl/explorer/diy-builder/fans/what-is-pwm/?srsltid=AfmBOoqTLsqWkzvVDSMfDPLJ9Zrp1AWCcQLEw-zD-_7QTt_tAP8cK0mt

W przypadku diody LED nie ustawiamy więc jej jasności przez płynną zmianę napięcia na

wyjściu. Dioda jest szybko włączana i wyłączana. Układ wzrokowy człowieka nie nadąża za tak szybkimi zmianami i odbiera je jako uśrednioną, pozornie stałą jasność. Im większy współczynnik wypełnienia, tym większa jest obserwowana jasność świecenia.

W Arduino UNO R4 WiFi piny PWM są oznaczone znakiem ~ na pinocie płytki; należą do nich m.in. D3, D5, D6, D9, D10 i D11. W tym zadaniu wykorzystamy D5.

Zadanie 3.1 — Płynna zmiana obserwowanej jasności



Podłącz diodę LED do pinu D5. Napisz program, który:

1. zwiększa wartość PWM od 0 do 255,
2. następnie zmniejsza ją od 255 do 0,
3. powtarza sekwencję w sposób ciągły.

Na tym etapie dopuszczalne jest użycie funkcji `delay()` do ustalenia szybkości zmian.

Skorzystaj z funkcji `analogWrite` wyjaśnionej poniżej.

Przydatna funkcja



W podstawowym użyciu funkcja `analogWrite(pin, value)` przyjmuje wartość z zakresu 0–255. Wartość 0 odpowiada wypełnieniu 0%, a 255 — wypełnieniu bliskiemu 100%.

Nazwa `analogWrite()` może być myląca: na pinie PWM funkcja nie wytwarza stałego napięcia analogowego, lecz przebieg cyfrowy o zmiennym współczynniku wypełnienia.

7 Zadanie 4: Programowanie nieblokujące

Funkcja `delay()` zatrzymuje wykonywanie programu. Jeżeli mikrokontroler ma równocześnie sterować kilkoma elementami i reagować na zdarzenia, takie rozwiązanie szybko staje się niewygodne.

Funkcja `millis()` zwraca liczbę milisekund od uruchomienia programu. Zamiast czekać, można sprawdzać, czy od ostatniego zdarzenia upłynął odpowiedni czas.

```
1 unsigned long now = millis();
2
3 if (now - previousTime >= interval) {
4     previousTime = now;
5     // operacja wykonywana okresowo
6 }
```

Zadanie 4.1 — Dwa niezależne procesy



Napisz program bez użycia funkcji `delay()`, w którym:

- dioda na pinie D8 zmienia stan co 500 ms,



- dioda na pinie D9 zmienia stan co 100 ms,
- przycisk na pinie D2 przełącza stan trzeciej diody na pinie D10.

Każdy z tych mechanizmów powinien działać niezależnie. Naciśnięcie przycisku nie może zatrzymywać migania pozostałych diod.

Typ zmiennej czasu



Do przechowywania wartości zwracanej przez `millis()` używaj typu `unsigned long`. Porównuj czasy przez odejmowanie: `now - previousTime >= interval`. Taka konstrukcja poprawnie działa również po przepełnieniu licznika `millis()`.

8 Zadanie końcowe: Sterownik sygnalizacji świetlnej

W zadaniu końcowym połączymy GPIO, przerwania i odmierzanie czasu w jeden prosty system wbudowany.

Zadanie 5.1 — Sygnalizacja z przyciskiem dla pieszego



Zbuduj układ składający się z trzech diod LED symbolizujących sygnalizator dla pojazdów oraz przycisku zgłoszenia przejścia dla pieszego.

Program powinien spełniać następującą specyfikację:

1. w stanie podstawowym świeci zielona dioda,
2. naciśnięcie przycisku zgłasza żądanie przejścia za pomocą przerwania,
3. po zgłoszeniu system przechodzi przez sekwencję:
zielone → żółte → czerwone → zielone,
4. światło żółte świeci przez 1 s,
5. światło czerwone świeci przez 3 s,
6. kolejne naciśnięcia przycisku podczas trwania sekwencji nie mogą uruchamiać jej od początku,
7. w całym programie nie wolno używać funkcji `delay()`.

Podziel program na czytelne stany. Możesz zastosować typ wyliczeniowy `enum` lub zmienną przechowującą aktualny stan systemu.

Dlaczego to jest system wbudowany?



Program nie wykonuje jednej sekwencji od początku do końca. Przez cały czas obserwuje wejścia, pamięta aktualny stan i wykonuje odpowiednią akcję, gdy wystąpi zdarzenie lub upłynie określony czas. Taki sposób organizacji programu będzie wykorzystywany również w kolejnych laboratoriach.

9 Podsumowanie

Po tych zajęciach powinieneś umieć



- skonfigurować pin jako wejście lub wyjście,
- wyjaśnić, po co stosuje się rezystor *pull-up* i wykorzystać `INPUT_PULLUP`,
- odczytać stan wejścia funkcją `digitalRead()` oraz wykryć naciśnięcie i zwolnienie przycisku,
- wyjaśnić zjawisko drgań styków i zastosować prostą metodę ich eliminacji,
- wyjaśnić zasadę działania przerwania zewnętrznego,
- skonfigurować przerwanie za pomocą `attachInterrupt()` i `digitalPinToInterrupt()`,
- wyjaśnić znaczenie `RISING`, `FALLING` i `CHANGE`,
- wyjaśnić, po co zmienna współdzielona z ISR jest oznaczana jako `volatile`,
- wykorzystać PWM do zmiany obserwowanej jasności LED i wyjaśnić, dlaczego oko widzi zmianę jasności,
- zastąpić `delay()` mechanizmem opartym na `millis()`,
- zbudować prostą maszynę stanów reagującą na zdarzenia i czas.

Co należy zapamiętać?



Przed opuszczeniem stanowiska odłącz zasilanie Arduino, uporządkuj przewody i elementy oraz pozostaw stanowisko w stanie umożliwiającym rozpoczęcie pracy następnej grupie.