

Porty I/O oraz wykorzystanie przycisków i diod świecących

Sterowanie z wykorzystaniem PWM

Opracował opiekun przedmiotu dr inż. Krzysztof Chudzik

Wydział Informatyki i Telekomunikacji
Politechnika Wrocławskiego

Wrocław, 2021.08.25 18:42:23

Spis treści

1 Wyprowadzenia elektryczne w Arduino UNO R3	1
2 Porty I/O i piny cyfrowe	2
2.1 Czym sa porty I/O	2
2.2 Tryby pracy pinów cyfrowych	2
2.3 Konfiguracja portów I/O w trybie wyjścia (output) i sterowanie ich stanem	3
2.4 Konfiguracja portów I/O w trybie wejścia (input) i odczyt ich stanu	4
2.4.1 Konfiguracja wejścia w trybie zwykłym	4
2.4.2 Konfiguracja wejścia z rezystorem podciągającym	5
2.4.3 Odczyt stanu podłączonych do portów przełączników przyciskanych	6
3 Dioda LED RGB (analogowa) i sterowanie PWM	7
3.1 Budowa i podłączenie diody RGB	7
3.2 Obliczanie wartości rezystora ograniczającego prąd diody przy świeceniu ciągłym	7
3.3 Sterowanie z modulacją szerokości impulsu (PWM)	7

Lista zadań

1 Konfiguracja, odczyt i sterowanie pinami cyfrowymi	6
2 Sterowanie jasnością świecenia diody z wykorzystaniem PWM	9
3 Uzyskiwanie kolorów pośrednich w diodzie RGB	9

1 Wyprowadzenia elektryczne w Arduino UNO R3

Przez wyprowadzenia będziemy rozumieć złącza i umieszczone w nich piny (styki) służące do przyłączania przewodów elektrycznych łączących płytę Arduino z innymi układami elektronicznymi. Dzięki tym wyprowadzeniom może dokonywać się interakcja pomiędzy płytą Arduino i jej otoczeniem z wykorzystaniem dodatkowych elementów lub układów elektronicznych. Mogą to być przełączniki, diody świecące, wyświetlacze, czujniki odległości lub ruchu, serwomechanizmy, i inne, których mnogość jest tak duża, że nie sposób ich wszystkich tu wymienić.

Prawidłowe podłączenie zewnętrznych elementów lub układów elektronicznych, wymaga znajomości funkcjonalności poszczególnych pinów płytki Arduino. Na witrynie producenta Arduino, na stronie [Arduino Uno Rev3 \(link\)](#) można znaleźć dokumentację techniczną do płytki Arduino UNO Rev. 3. W szczególności, dokument [Pinout UNO rev3 latest \(link\)](#) opisuje

piny dostępne na płytce Arduino UNO Rev. 3, która jest częścią zestawu laboratoryjnego. Proszę pobrać ten dokument i zapoznać się z nim. W części laboratorium poświęconej Arduino, jeśli zostanie użyty zwrot „schemat wyprowadzeń” lub „schemat pinów” należy go rozumieć jako właśnie ten dokument (lub inny jemu równoważny).

Schemat wyprowadzeń zawiera obraz płytka i przypisane do każdego pinu funkcjonalności w postaci kolorowych etykiet. Legenda, wyjaśniająca ogólne znaczenie etykiet, znajduje się pod rysunkiem na każdej stronie. Strona pierwsza zawiera najczęściej wykorzystywane, podstawowe funkcjonalności. Druga strona, zawiera rozszerzenie pierwszej o kolejne, bardziej zaawansowane funkcjonalności. Będziemy poznawać je w czasie kursu na laboratorium i w ramach wykładu.

Schemat wyprowadzeń informuje, że do większości pinów Arduino przypisana jest więcej niż jedna funkcjonalność. Wynika to z uniwersalności pinów mikrokontrolera ATmega328P. Jednym z najważniejszych trybów pracy pinu jest praca jako wejście lub wyjście cyfrowe. Opisane jest to na schemacie wyprowadzeń jako funkcjonalność *pin cyfrowy*, ang. *Digital Pin*.

2 Porty I/O i piny cyfrowe

2.1 Czym są porty I/O

Piny cyfrowe Arduino czy, bardziej ogólnie, mikrokontrolera, z punktu widzenia architektury mikrokontrolera są zgrupowane w porty. Porty I/O zawierają rejestrystery sterujące pinami. W ATmega 328P liczba pinów w porcie nie przekracza 8, aby umożliwić konfigurowanie i sterowanie portami z wykorzystaniem pojedynczych bajtów, które zawierają odpowiednie ustawienia pinów. Dostęp do pinów I/O na poziomie sprzętowym odbywa się za pomocą specjalnych rejestrów mikrokontrolera skojarzonych z portami, które umożliwiają ich konfigurację. Konfiguracja pinów z wykorzystaniem rejestrów portów zostanie omówiona na wykładzie.

W przypadku Arduino, zdefiniowano dedykowane Arduino funkcje i identyfikatory ułatwiające programowanie. Jak wspomniano w materiałach Laboratorium 1, będziemy nazywać je nieformalnie językiem Arduino. Są one dostępne w ramach narzędzi programistycznych dla Arduino.

W przypadku pinów cyfrowych, zdefiniowano funkcje, które umożliwiają konfigurację pojedynczych pinów. Z całą pewnością, zwiększa to rozmiar kodu programu i zmniejsza jego efektywność, ale osobie początkującej pozwala zrozumieć funkcjonowanie pinu cyfrowego, bez wnikania w pewne zawiłości związane z konfiguracją portów. W dalszej części laboratorium skupimy się na zatem na sposobie dostępu do pinów cyfrowych.

2.2 Tryby pracy pinów cyfrowych

Piny cyfrowe umożliwiają komunikację ze światem zewnętrznym za pomocą sygnałów cyfrowych, czyli takich, gdzie napięcie elektryczne na pinie reprezentuje pewien stan logiczny.

W przypadku sygnałów elektrycznych nie używa się pojęć prawda czy fałsz, ale mówimy o stanie napięciowym wysokim (high) lub niskim (low). Jeśli nie powoduje to niejednoznaczności, używa się pojęć „1” i „0”, które odpowiadają stanom wysokiemu i niskiemu. Ten rodzaj zapisu (notacji) stosowany jest często w dokumentacji. Dzięki takiemu zapisowi w sposób zwięzły wyrażamy stany, na przykład, wszystkich pinów w porcie lub zawartość wybranego rejestru w postaci liczby binarnej. Taka liczebność, aby dalej skrócić zapis, możemy zapisać w formacie szesnastkowym.

Piny cyfrowe mogą być skonfigurowane w dwóch podstawowych trybach:

- wejście (input) - w tym trybie odczytywany jest stan pinu, który może być ustalony przez wyjście innego układu elektronicznego (przelącznik, czujnik, itp.),
- wyjście (output) - w tym trybie mikrokontroler ustawia stan napięciowy pinu, który może być odczytany przez wejście innych układów elektronicznych (sygnalizator, element wykonawczy, itp.).

Piny mają ograniczenia dotyczące napięć elektrycznych na nich występujących i prądów elektrycznych przez nie płynących. W szczególności, do pinów wyjściowych, nie wolno podłączać bezpośrednio elementów o charakterze pojemnościowym i indukcyjnym (w tym przekaźników bez zabezpieczenia przepięciowego, silników, itp.). Nie wolno podłączać też odbiorników rezystancyjnych o dużym poborze mocy (rezystorów o małej rezystancji, diod świecących bez rezystorów ograniczających prąd, itp.). Karta katalogowa ATmega328P ([link - pobrać dokument ATmega48A/PA/88A/PA/168A/PA/328/P Data Sheet](#)) zawiera podrozdziały Electrical Characteristics (dla dwóch wersji temperaturowych mikrokontrolera), a w nich, w ramce, Absolute Maximum Ratings. Są to informacje o parametrach, których nie wolno przekraczać. Po ich przekroczeniu należy liczyć się z nieodwracalnym uszkodzeniem mikrokontrolera. Proszę pobrać kartę katalogową (dla rodziny układów zwykłych, nie automotive) i przejrzeć rozdziały Electrical Characteristics, aby nabyć orientacji w jakich zakresach prądowych i napięciowych poruszamy się na laboratorium.

Uwaga: Nie wolno bezpośrednio łączyć elektrycznie dwóch pinów wyjściowych ze sobą. Na ogół, prowadzi to do nieodwracalnego uszkodzenia tych pinów lub całego mikrokontrolera.

Pin port I/O jest jedynie jedną z wielu funkcjonalności danego pinu. Koegzystencja wielu funkcjonalności na jednym pinie powoduje, że jeśli pin został skonfigurowany jako pin portu I/O, to w tym samym czasie nie może być wykorzystywany przez inny sprzętowy układ wewnętrzny mikrokontrolera, na przykład, szeregowie łącze komunikacyjne.

Jak wynika ze schematu pinów, na płytce Arduino możemy skonfigurować 20 pinów cyfrowych, oznaczonych identyfikatorami od D0 do D19.

Uwaga: Różne układy cyfrowe, w tym mikrokontrolery, pracują z różnymi poziomami napięć logicznych. Nawet wśród rodziny Arduino można znaleźć płytki, które mają logikę 5V (na przykład, Arduino UNO) i 3,3V (na przykład, ARDUINO NANO 33 BLE). Mikrokontroler Raspberry Pi też posiada logikę 3,3V (nie mylić z napięciem zasilania). Nie wolno łączyć bezpośrednio układów (mikrokontrolerów, czujników, wyświetlaczów, itp.) o różnych poziomach logicznych, ponieważ ze względu na nieakceptowalne poziomy napięć dojdzie do uszkodzeń układów. Do łączenia układów o różnych poziomach logicznych wykorzystuje się konwertery poziomów logicznych. Zobaczą Państwo taki układ w zestawie laboratoryjnym z Raspberry Pi.

2.3 Konfiguracja portów I/O w trybie wyjścia (output) i sterowanie ich stanem

Konfiguracja pinu cyfrowego za pomocą funkcji dedykowanych Arduino odbywa się z wykorzystaniem funkcji, które widzieliśmy już w dokumentacji laboratorium 1.

Dokumentację funkcji dedykowanych Arduino (języka Arduino) można znaleźć na stronie [Language Reference \(link\)](#).

Funkcja `pinMode()` konfiguruje pin jako wejście lub wyjście. Można ustawić jeden z trzech trybów:

- `OUTPUT` - wyjście,
- `INPUT` - wejście,
- `INPUT_PULLUP` - w wejście z rezystorem podciągającym (będzie omówione dalej).

Proszę zapoznać się z [dokumentacją funkcji `pinMode\(\)` \(link\)](#).

Ustawienie wyjścia w zadanym stanie napięciowym wysokim (HIGH) lub niskim (LOW) odbywa się z wykorzystaniem funkcji `digitalWrite()`. Proszę zapoznać się z [dokumentacją funkcji `digitalWrite\(\)` \(link\)](#).

Wielokrotnie będziemy posługiwać się różnymi stałymi, czy identyfikatorami, zdefiniowanymi dla płytki Arduino. Są one wymienione w [sekcji dokumentacji \(link\)](#) języka Arduino opisującej typy danych i stałe.

Wśród tych stałych znajduje się identyfikator `LED_BUILTIN`, który przyjmuje wartość 13, czyli numer trzynastego pinu cyfrowego (D13). Do tego pinu podłączona jest dioda LED i rezystor zabezpieczający ją (ograniczający prąd diody). Dioda będzie świecić jeśli na pin D13 podany zostanie stan wysoki (HIGH). W stanie niskim (LOW) dioda nie będzie świecić.

Zatem, najprostszy kod włączający wbudowaną diodę po uruchomieniu Arduino będzie wyglądał jak ilustruje to Kod 1.

Kod 1: Włączenie wbudowanej diody przy starcie Arduino.

```
1 void setup()
2 {
3     pinMode(LED_BUILTIN, OUTPUT);
4     digitalWrite(LED_BUILTIN, HIGH);
5 }
6
7 void loop()
8 {
9 }
```

Po skompilowaniu i wgraniu programu, dioda zaświeci się i w takim stanie pozostanie. Jeśli chcielibyśmy aby migała, trzeba ja naprawdę włączać i wyłączać przez zmianę stanu pinu D13.

Ponieważ pętla loop wykonywana jest cyklicznie, w niej należy zatrzymać fragment kodu, który zmieni stan wyjścia na wysoki lub niski, a następnie wstrzyma swoje działanie na określony czas, po którym zmieni stan D13 na przeciwny.

Wstrzymanie wykonania programu na określony czas można wykonać, na przykład, funkcją `delay()`. Proszę zapoznać się z [dokumentacją funkcji `delay\(\)` \(link\)](#). Jak wynika z dokumentacji, funkcja ta ma poważną wadę, która polega na marnowaniu czasu mikrokontrolera. W naszym przypadku, wstrzymuje inne akcje programu, które mogłyby odbywać się w czasie oczekiwania na przełączenie stanu diody świecącej. Wróćmy jeszcze do zagadnienia odkładania wykonania operacji w czasie, choć należy zapamiętać, że użycie funkcji `delay()` może i jest proste, ale na pewno bardzo nieefektywne.

Zatem, aby móc migać diodą, dochodzimy do programu przedstawionego jako Kod 2, a który już był prezentowany w ramach poprzedniego laboratorium.

Kod 2: Przykład z Arduino IDE: Blink.ino

```
1 // the setup function runs once when you press reset or power the board
2 void setup() {
3     // initialize digital pin LED_BUILTIN as an output.
4     pinMode(LED_BUILTIN, OUTPUT);
5 }
6
7 // the loop function runs over and over again forever
8 void loop() {
9     digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
10    delay(1000); // wait for a second
11    digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
12    delay(1000); // wait for a second
13 }
```

2.4 Konfiguracja portów I/O w trybie wejścia (input) i odczyt ich stanu

Konfiguracja pinu w trybie wejścia pozwala odczytać czy na wejściu ustawiony jest poziom napięciowy wysoki (HIGH) czy niski (LOW).

Konfiguracja trybu pracy dokonywana jest funkcją `pinMode()` z argumentem INPUT, który konfiguruje wejście w trybie zwykłym, lub INPUT_PULLUP, który konfiguruje wejście z rezystorem podciągającym.

Odczyt stanu wejścia realizuje funkcja `digitalRead()`. Proszę zapoznać się z [dokumentacją funkcji digitalRead\(\)](#) ([link](#)).

2.4.1 Konfiguracja wejścia w trybie zwykłym

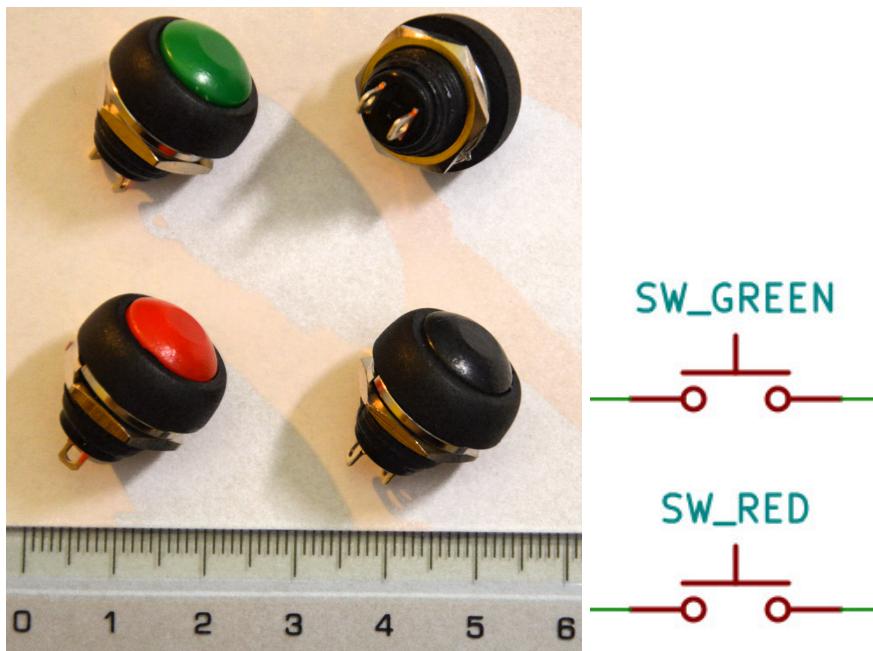
Tryb ten konfigurujemy funkcją `pinMode()` z argumentem INPUT.

W tym trybie, pin ma bardzo dużą impedancję wejściową, czyli pobiera bardzo mały prąd z układu, który wymusza (ustawia) jego stan. W tym trybie, jeśli pin nie jest podłączony elektrycznie do żadnego układu, to należy założyć, że stan napięciowy jest nieustalony lub przypadkowy. Zakłócenia elektryczne, na przykład, pochodzące z instalacji elektrycznej w budynku mogą wpływać na stan niepodłączonego pinu. Odczytywanie stanu takiego wejścia nie ma sensu.

Ustalenie (wymuszenie) stanu na pinie wejściowym może nastąpić poprzez przyłączenie go do wyjścia cyfrowego innego układu.

W wielu sytuacjach chcemy, aby układ reagował na naciśnięcie przełącznika (przycisku). Wejście cyfrowe można w bardzo prosty sposób przystosować do odczytu stanu przycisku.

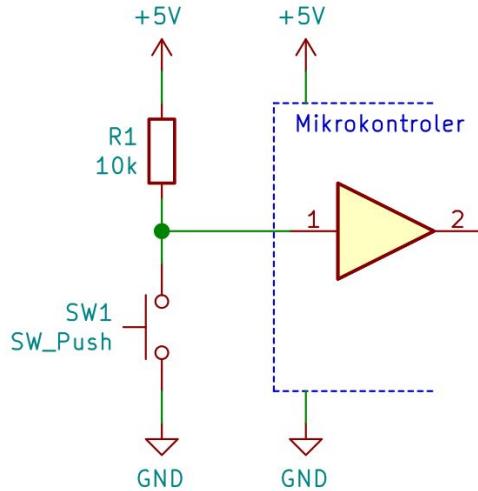
Rola przełącznika jest zamknięcie i otwieranie obwodu elektrycznego. Przykład przełączników przyciskanych jest na fotografii 1.



Fotografia 1: Przełączniki przyciskane (przyciski).

Są to przyciski zamontowane w naszym zestawie laboratoryjnym. Gdy nie są wcisnięte, obwód pomiędzy wyprowadzeniami jest otwarty (dlatego jest nazywany przełącznikiem normalnie otwartym). Jeśli przycisniemy przycisk, to wyprowadzenia są zwierane (obwód zamknięty).

Przykład podłączenia do mikrokontrolera znajduje się na schemacie 1.



Schemat 1: Podłączenie przycisku do wejścia mikrokontrolera z wejściem zwykłym.

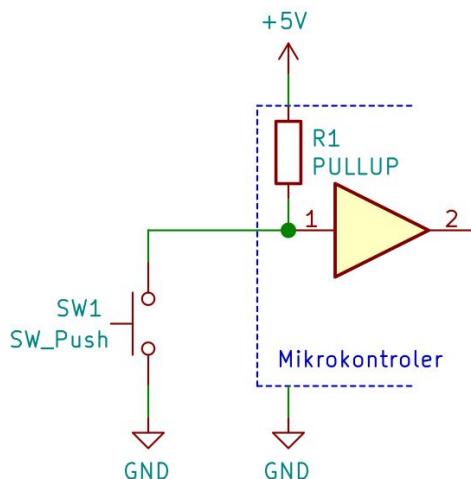
Układ działa w sposób następujący. Gdy przycisk jest zwolniony (nie jest naciśnięty), to stanowi przerwę w obwodzie. Rezystancja pinu wejściowego jest bardzo duża, dużo większa niż rezystancja rezystora R1 (tu przyjęto wartość 10kOhm - czytaj 10 kilo Ohmów). Możemy przyjąć, że przez rezystor R1 nie płynie prawie żaden prąd, wobec tego napięcie na jego obu końcach jest równe 5V. Oznacza to, że na wejście podany jest stan wysoki (HIGH), jeśli przycisk jest zwolniony, ponieważ potencjał tego wejścia został podciągnięty do 5V.

Jeśli naciśniemy przycisk, to zewrze on do potencjału 0V (GND - *ground*) wejście i końcówkę rezystora R1, która jest do wejścia podłączona. Przez rezystor popłynie oczywiście prąd, który łatwo można wyliczyć z prawa Ohma. Jednak potencjał wejścia pozostanie na poziomie 0V. Oznacza to, że na wejście podany jest stan niski (LOW) jeśli przycisk jest wciśnięty.

Zwróćmy uwagę, że rezystora R1 nie można usunąć, bo przy zwolnionym przycisku stan wejścia będzie nieustalony. Nie można tego rezystora zastąpić zwarciem, bo po naciśnięciu przycisku zwarlibyśmy źródło zasilającego układ i popłynąłby bardzo duży prąd, który najprawdopodobniej uszkodziłby układ. Wartość rezystora R1, nie może być zbyt mała, bo po naciśnięciu przycisku płynąłby większy prąd, co ma znaczenie szczególnie przy zasilaniu baterijnym lub akumulatorowym. Z drugiej strony rezystancja nie może być zbyt duża, bo wtedy wpływ zakłóceń może być większy niż działanie podciągające rezystora i przestanie on pełnić swoją rolę. Typowa wartość tego rezystora to 10kOhm.

2.4.2 Konfiguracja wejścia z rezystorem podciągającym

Przyciski, czy inne elementy elektroniczne zwierające styki zwierne czy rozwierne, na przykład, enkodery, wyłączniki krańcowe, itp., są powszechnie wykorzystywane w sprzęcie elektronicznym. Producenci mikrokontrolerów zdecydowali się ułatwić pracę projektantom i uprościć montaż takich elementów. Rezystor R1 podciągający wejście do napięcia 5V ze schematu 1 włączono do układów elektronicznych we wnętrzu mikrokontrolera. Nazywany jest on, po prostu, rezystorem podciągającym. Pomimo, iż fizycznie jest to bardziej złożony układ niż pojedynczy rezystor, to podstawowa funkcja jest taka sama. Ilustruje to schemat 2.



Schemat 2: Podłączenie przycisku do wejścia mikrokontrolera z rezystorem podciągającym (pullup).

Rezystor podciągający włącza się lub wyłącza ustawienia w rejestrach konfiguracyjnych portu I/O, w ramach którego funkcjonuje pin. Jest to więc operacja programowa. W języku Arduino, włączenie rezystora podciągającego odbywa się przez

podanie argumentu INPUT_PULLUP, gdy konfigurowane pin jako wejście przy pomocy funkcji pinMode().

2.4.3 Odczyt stanu podłączonych do portów przełączników przyciskanych

Zatem odczyt stanu przełączników podłączonych do portów będzie odbywał się tak jak pokazano w kodzie 3. W zestawie laboratoryjnym poszczególne diody w ramach diody RGB, podłączone są do pinów: D6 - czerwona, D5 - zielona i D3 - niebieska. Przyciski są podłączone do pinów: D2 - czerwony, D4 - zielony.

Aby program był czytelniejszy, dyrektywą preprocessora `#define` wprowadza się identyfikatory diod i przycisków z przypisaniem im numerów pinów, do których są podłączone. W przypadku dyrektywy `#define` nie są rezerwowane obszary pamięci do przechowywania stałych, ale wartości te są podstawiane za identyfikator bezpośrednio w kodzie programu przed komplikacją. Proszę zauważać, że jest to nieco inny mechanizm, niż stała wprowadzana ze słowem kluczowym `const`.

W podobny sposób, dyrektywą `#define` można wprowadzić całe makra z argumentami, które będą wprowadzane do kodu w miejscu występowania identyfikatorów makr.

Kod 3: Odczyt stanu przycisków i zaświecanie diod.

```
1 #define LED_RED 6
2 #define LED_GREEN 5
3 #define LED_BLUE 3
4
5 #define RED_BUTTON 2
6 #define GREEN_BUTTON 4
7
8 void initRGB()
9 {
10     pinMode(LED_BUILTIN, OUTPUT);
11     digitalWrite(LED_BUILTIN, LOW);
12
13     pinMode(LED_RED, OUTPUT);
14     digitalWrite(LED_RED, LOW);
15
16     pinMode(LED_GREEN, OUTPUT);
17     digitalWrite(LED_GREEN, LOW);
18
19     pinMode(LED_BLUE, OUTPUT);
20     digitalWrite(LED_BLUE, LOW);
21 }
22
23 void initButtons()
24 {
25     pinMode(RED_BUTTON, INPUT_PULLUP);
26     pinMode(GREEN_BUTTON, INPUT_PULLUP);
27 }
28
29 void setup()
30 {
31     initRGB();
32     initButtons();
33 }
34
35 void loop()
36 {
37     if (digitalRead(RED_BUTTON) == LOW)
38         digitalWrite(LED_RED, HIGH);
39     else
40         digitalWrite(LED_RED, LOW);
41
42     if (digitalRead(GREEN_BUTTON) == LOW)
43         digitalWrite(LED_GREEN, HIGH);
44     else
45         digitalWrite(LED_GREEN, LOW);
46 }
```

Przyciski to elementy elektro-mechaniczne. Zamknięcie lub otwarcie obwodu elektrycznego wiąże się tu ze zjawiskiem określonym mianem „drążenia styków”. Powoduje ono, że mamy pewne stany przejściowe, które mogą być interpretowane jako bardzo szybkie wielokrotne zamknięcie i otwarcie obwodu. Tutaj ten problem jedynie sygnalizujemy, ale do jego rozwiązania wróćmy jeszcze.

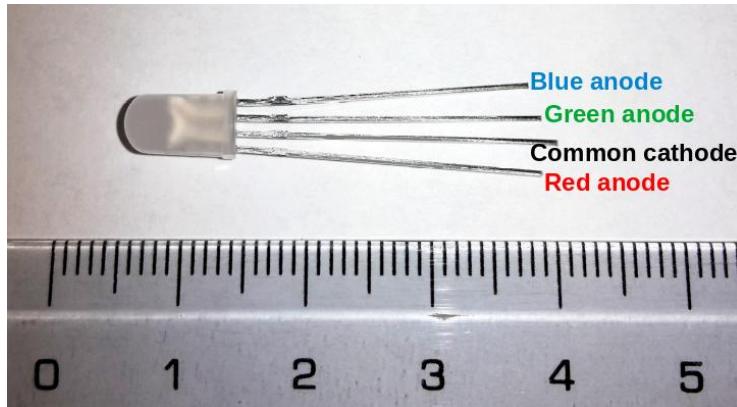
Zadanie 1: Konfiguracja, odczyt i sterowanie pinami cyfrowymi

Przygotuj prosty program wykorzystujący przełączniki przyciskane (zielony i czerwony) oraz diody świecące według wskaźówek Prowadzącego laboratorium. Przykładowy program: Po naciśnięciu przycisku zielonego zmień kolor świecącej diody. Przycisk czerwony włącza i wyłącza świecenie.

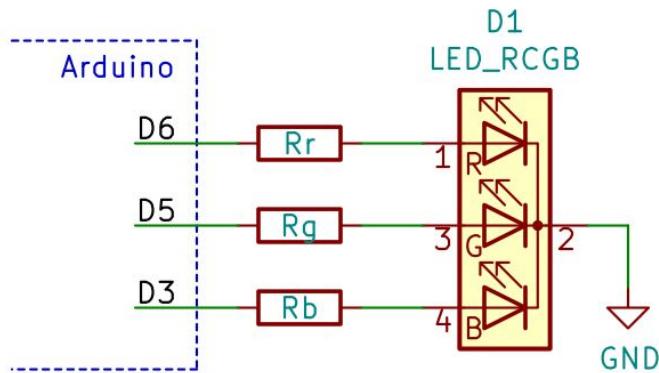
3 Dioda LED RGB (analogowa) i sterowanie PWM

3.1 Budowa i podłączenie diody RGB

Obudowa diody zawiera trzy struktury świecące (trzy diody elektroluminescencyjne) w trzech podstawowych barwach: czerwonej (R), zielonej (G) i niebieskiej (B). Odpowiednio dobrane średnie wartości natężenia prądów płynących przez diody pozwalały uzyskać dowolny kolor i jego jasność. Regulację prądu dokonuje się poprzez rezystory ograniczające prąd oraz współczynnik wypełnienia przebiegu elektrycznego, który pozwala regulować średnie natężenie prądu. Diody analogowe RGB występują w układzie wspólnej anody lub katody. W układzie wspólnej katody (na fotografii 2) mamy wyprowadzenie katody wspólnej dla trzech diod, którą przyłącza się do potencjału ujemnego, oraz trzech anodach, które poprzez rezystory ograniczające prąd, podłącza się do potencjału dodatniego, tak jak pokazano to na schemacie 3.



Fotografia 2: Dioda RGB z zestawu laboratoryjnego



Schemat 3: Podłączenie diody RGB w zestawie laboratoryjnym.

3.2 Obliczanie wartości rezystora ograniczającego prąd diody przy świeceniu ciągłym

Jeśli decydujemy się podłączyć diodę świecącą do pinu wyjścia mikrokontrolera lub płytki Arduino musimy ograniczyć prąd płynący przez diodę za pomocą dodatkowego rezystora włączonego w szereg z diodą. Dioda ma nieliniową charakterystykę napięciowo-prądową. Podłączenie diody bezpośrednio spowoduje przepływ bardzo dużego prądu i uszkodzenie mikrokontrolera, albo diody świecącej. Stąd dodatkowe rezystory ograniczające prąd dla każdej diody wewnętrznej diody RGB na schemacie 3.

Temat ten jest dobrze omówiony w artykule na blogu Forbot: [Jak dobrać rezistor do diody? Różne metody zasilania LED! \(link\)](#). Nie stawiam tego jako wymogu, ale sugeruję zapoznać się z nim, szczególnie jeśli chcecie Państwo budować układy z Arduino, Raspberry Pi, itp., samodzielnie.

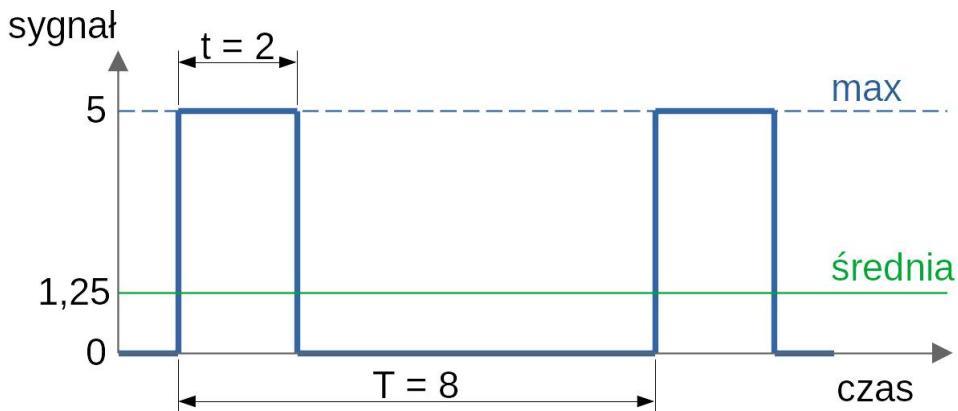
3.3 Sterowanie z modulacją szerokości impulsu (PWM)

Jasność świecenia ciągłego można regulować przez zmianę wielkości prądu, który płynie przez diodę. Można to zrobić dobierając odpowiednio rezystor w szeregu z diodą. Niestety, przy takim sterowaniu, aby zmienić jasność, należy zmienić rezystor. Nie jest rozwiązaniem praktyczne. Można zbudować mniej lub bardziej złożony analogowy układ elektroniczny, który będzie sterował prądem, ale nie zawsze jest to ekonomicznie uzasadnione.

Jasność świecenia diody odbierana przez człowieka jest pewną funkcją wartości prądu płynącego przez diodę. Przy czym, jeśli wymusimy przepływ prądu zmieniającego się okresowo w czasie, to jeżeli zmiany te będą następować na tyle szybko (krótki okres), że nie zauważymy ich oko ludzkie, to jasność odbierana będzie funkcją wartości średniej przepływanego prądu.

Sygnal taki można generować zaświecając i gasząc diodę odpowiednio szybko. Jeśli w kodzie 2 zmniejszymy opóźnienia w funkcji `delay()`, to zauważymy, że dioda przestała dla nas migać, a jedynie świeci nieco ciemniej. Można jednak ten problem rozwiązać bardziej efektywnie.

Mając do dyspozycji mikrokontroler możemy skorzystać ze sterowania z modulacją szerokości impulsu (PWM - Pulse Width Modulation). Przebieg wartości sygnału z modulacją szerokości impulsu w czasie ilustruje rysunek 1.



Rysunek 1: Modulacja szerokości impulsu (PWM).

Wartość średnia sygnału jest wprost proporcjonalna do wartości ilorazu t/T , który nazywany jest współczynnikiem wypełnienia. Zazwyczaj jest on podawany w procentach. Wartość 0% oznacza, że sygnał ma stale wartość zero, 100% oznacza, że stale ma wartość maksymalną. Dla przebiegu z rysunku 1 wartość współczynnika wypełnienia wynosi 25%. Zmieniając współczynnik wypełnienia sygnału o małym okresie, możemy zmieniać jasność świecenia diody. A to już jest realizowalne programowo, a nawet wspierane sprzętowo przez mikrokontroler, w oparciu o liczniki czasu (timers) wbudowane w mikrokontroler.

Płytki Arduino ma wyprowadzenia, które dostarczają sygnał z modulacją szerokości impulsu, o okresie na tyle małym, że jego zmienność w czasie umyka percepcji ludzkiej. Sygnały takie obserwuje się przy pomocy specjalizowanych narzędzi, na przykład, przy pomocy oscyloskopu.

Sygnały takie są wyprowadzane na wybrane piny. Można je zlokalizować na schemacie pinów Arduino. W oznaczeniach wybranych portów cyfrowych obsługujących PWM, przed ich identyfikatorem, znajduje się znak tyldy „~”. Są to, między innymi, `~D6`, `~D5` i `~D3`, czyli piny, do których przyłączona jest dioda RGB w zestawie laboratoryjnym. Zatem, wszystkie kolory diody można sterować przy pomocy PWM.

Samo sterowanie wypełnieniem impulsu realizowane jest sprzętowo przez mikrokontroler, zatem nie wpływa ono na przebieg programu. Pin, na którym ma funkcjonować PWM, konfiguruje się jako wyjście cyfrowe (`pinMode(ledPin, OUTPUT);`). Natomiast wartość współczynnika wypełnienia konfiguruje się funkcją `analogWrite()`. Proszę zapoznać się z dokumentacją funkcji `analogWrite()` ([link](#)). Nazwa funkcji może dziwić, ale wynika ona najprawdopodobniej z faktu, że wartość średnia przebiegu może przyjmować 256 wartości, a nie tylko dwie (`LOW` i `HIGH`). Po odpowiednim odfiltrowaniu sygnału, można więc generować wolnozmiennie sygnały analogowe. Od razu jednak wyjaśniam, że w przypadku Arduino UNO, gdy korzystamy z funkcji `analogWrite()`, okres przebiegu PWM jest na tyle długi, że nie pozwala generować wysokiej jakości dźwięku.

Pamiętajmy, że funkcja `analogWrite()`, ma współczynnik wypełnienia podawany jako wartość całkowitoliczbową z przedziału od 0 do 255. Wartości 0 dopowiada współczynnik wypełnienia 0%, a wartości 255 odpowiada współczynnik wypełnienia 100%. Stąd, aby skonfigurować współczynnik wypełnienia 25%, należy podać wartość $0.25 * 255 = 64$ (zaokrąglamy do najbliższej liczby całkowitej).

Poniżej, jako kod 4 został przedstawiony najprostszy program konfigurujący w zestawie laboratoryjnym diodę czerwoną (w diodzie RGB) do świecenia z wypełnieniem 25%.

Kod 4: Wysterowanie diody świecącej przez PWM (współczynnik wypełnienia 25%).

```

1 #define LED_RED 6
2
3 void setup()
4 {
5     pinMode(LED_RED, OUTPUT);
6     analogWrite(LED_RED, 64);
7 }
8
9 void loop()
10{
11}

```

Kod 5 ilustruje jak wygląda przyciemnienie diody w stosunku do maksymalnego świecenia.

Kod 5: Przyciemnianie diody LED.

```
1 #define LED_RED 6
2
3 void setup()
4 {
5     pinMode(LED_RED, OUTPUT);
6 }
7
8 void loop()
9 {
10    analogWrite(LED_RED, 64);
11    delay(1000);
12    analogWrite(LED_RED, 255);
13    delay(1000);
14 }
```

Zadanie 2: Sterowanie jasnością świecenia diody z wykorzystaniem PWM

Napisz program, który w reakcji na przyciski będzie rozjaśniał i ściemniał wybraną diodę. Na przykład, dioda zielona, po przytrzymaniu przycisku zielonego dioda będzie rozjaśniała się, a po przytrzymaniu czerwonego będzie ściemniać. Prowadzący laboratorium może zmodyfikować zadanie.

Zadanie 3: Uzyskiwanie kolorów pośrednich w diodzie RGB

Napisz program, który w sposób łagodny będzie zmieniał kolory diody RGB w cyklu, na przykład, czerwony > zielony > niebieski > czerwony ..., wyświetlając kolory pośrednie. Prowadzący laboratorium może zmodyfikować zadanie.