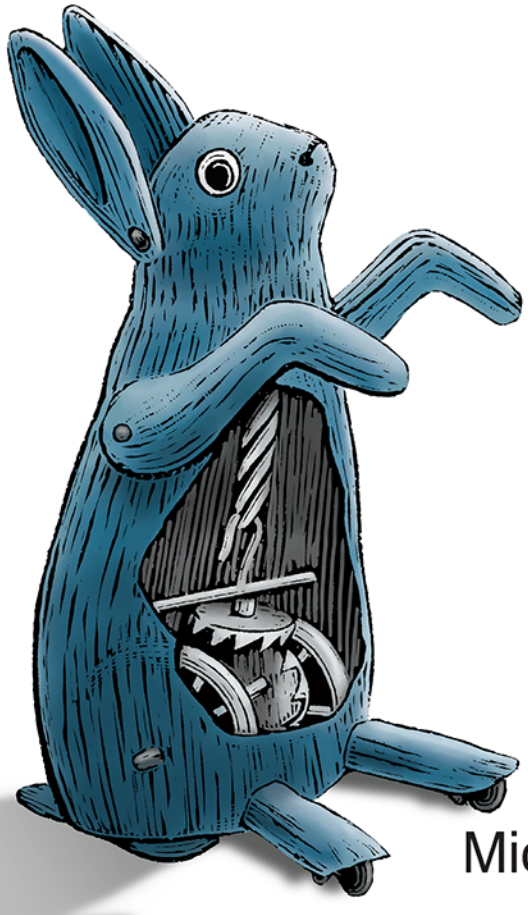
O'REILLY®

Wydanie III

Arduino

Przepisy na rozpoczęcie, rozszerzanie i udoskonalanie projektów



Michael Margolis
Brian Jepson
Nicholas Robert Weldin

Helion

Tytuł oryginału: Arduino Cookbook: Recipes to Begin, Expand, and Enhance Your Projects, 3rd Edition

Tłumaczenie: Anna Mizerska

ISBN: 978-83-283-7161-3

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/ardro3.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/ardro3
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 7.

Optyczne urządzenia wyjścia

7.0. Wprowadzenie

Dzięki optycznym urządzeniom wyjścia Arduino za pomocą szerokiej gamy komponentów LED może przekazywać informacje użytkownikowi. (Arduino może również wyświetlać informacje na graficznych wyświetlaczach, o czym przeczytasz więcej w rozdziale 11.). Zanim zaczniemy zgłębiać receptury z tego rozdziału, omówimy analogowe i cyfrowe wyjście Arduino i wyjaśnimy, jak ta płytka współpracuje z diodami LED (ang. *Light-Emitting Diodes* — diody emitujące światło). To wprowadzenie będzie dobrym punktem wyjścia, jeśli nie masz jeszcze doświadczenia z cyfrowym i analogowym wyjściem (`digitalWrite` i `analogWrite`) lub stosowaniem diody LED w obwodzie. Receptury w tym rozdziale obejmują wszystkie zagadnienia, od obsługi pojedynczej diody LED do stworzenia efektu świetlnego (receptura 7.7) i wyświetlania kształtów (receptura 7.9).

Cyfrowe wyjście

Wszystkie piny, które można wykorzystać jako cyfrowe wejście, można również zastosować jako cyfrowe wyjście. W rozdziale 5. znajduje się przegląd układu pinów Arduino i jeśli nie wiesz, jak podłączać komponenty do Arduino, przeczytaj „Wprowadzenie" do tego rozdziału.

Cyfrowe wyjście powoduje, że napięcie na pinie jest albo wysokie (5 lub 3,3 V w zależności od płytki), albo niskie (0 V). Użyj funkcji `digitalWrite(pinWyjscia, wartosc)`, aby coś włączyć lub wyłączyć. Funkcja przyjmuje dwa argumenty: `pinWyjscia` to pin, którym funkcja ma sterować, a `wartosc` ma wartość `HIGH` (5 lub 3,3 V) albo `LOW` (0 V).

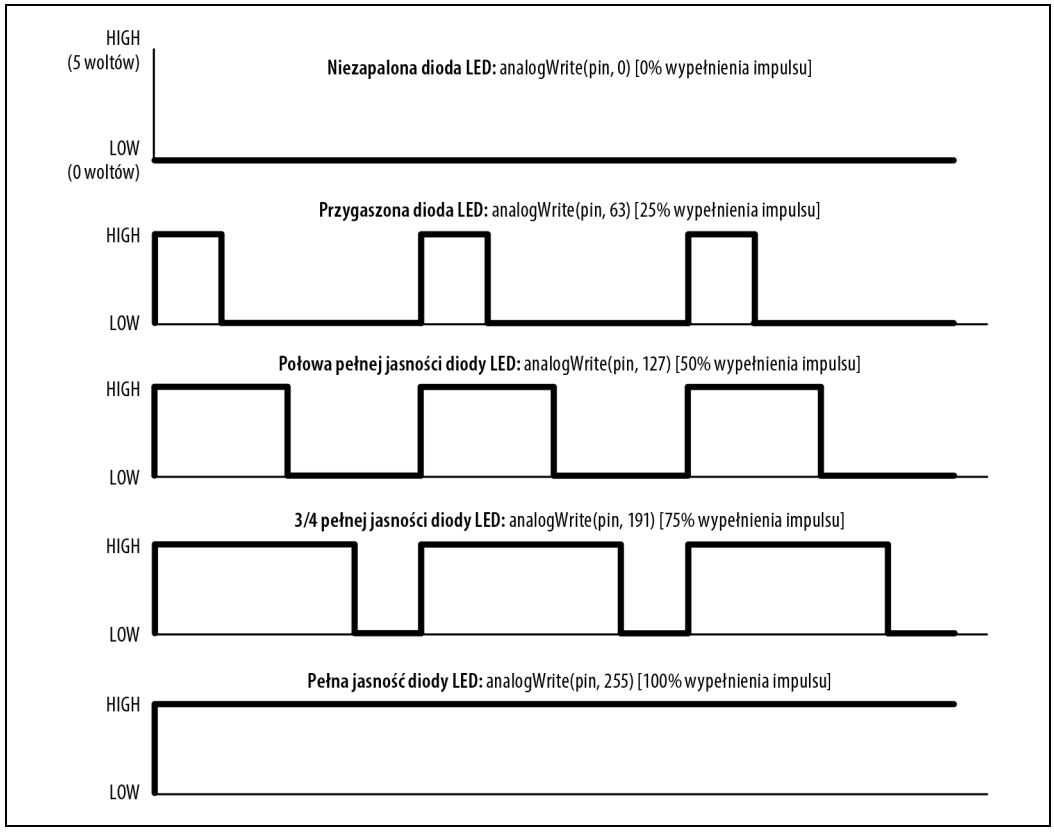
Aby napięcie reagowało na to polecenie, tryb wyjścia musi być ustawiony na wybranym pinie za pomocą komendy `pinMode(pinWyjscia, OUTPUT)`. Szkic z receptury 7.1 pokazuje, jak używać cyfrowego wyjścia.

Analogowe wyjście

Określenie „analogowe" odnosi się do poziomów, które mogą się stopniowo zmieniać, od swoich minimalnych do maksymalnych wartości (można to porównać do ściemniacza światła lub regulacji głośności). Funkcja `analogWrite` pozwala sterować takimi wartościami jak intensywność świecenia diody LED podłączonej do Arduino.

Funkcja `analogWrite` nie jest tak naprawdę analogowa, mimo że może działać jak analogowa, o czym przekonasz się wkrótce. Funkcja ta wykorzystuje technikę modulacji szerokości impulsów (PWM — ang. *Pulse-Width Modulation*), która naśladuje sygnały analogowe za pomocą impulsów cyfrowych.

PWM działa na zasadzie różnicowania długości czasu, w którym impuls jest włączony lub wyłączony tak, jak widać na rysunku 7.1. Wyjście niskiego poziomu jest emulowane za pomocą impulsów, które są włączone tylko przez krótką chwilę. Wyjście wyższego poziomu jest emulowane przez impulsy, które są dłużej włączone niż wyłączone. Gdy impulsy są powtarzane wystarczająco szybko (niemal 500 razy na sekundę lub szybciej, jak w przypadku płytek Arduino), drgania nie są wyczuwalne dla ludzkich zmysłów i w efekcie wydaje się, że wartości takich komponentów jak diody LED są płynnie zmieniane wraz ze zmianą współczynnika wypełnienia impulsu.



Rysunek 7.1. Wyjście PWM dla równych wartości analogWrite

Arduino ma ograniczoną liczbę pinów, które mają funkcję PWM. W przypadku Arduino Uno i innych płytek opartych na ATmega328 możesz użyć pinów 3, 5, 6, 9, 10 i 11. Na Arduino Mega możesz korzystać z pinów od 2 do 13 i od 44 do 46. Płytka Nano ma tylko pięć takich pinów, podczas gdy Zero, SparkFun RedBoard Turbo oraz Adafruit Metro Express M0 obsługują PWM na każdym pinie cyfrowym, z wyjątkiem pinów 2 i 7. Wiele z dalszych receptur wykorzystuje piny, które mogą być użyte zarówno dla cyfrowych wyjść, jak i funkcji PWM, aby ograniczyć potrzebę zmiany szkicu, gdybyś chciał wypróbować inne receptury. Jeśli dla PWM chciałbyś wybrać inne piny, pamiętaj,

aby wybrać jeden z pinów obsługujących funkcję `analogWrite` (inne piny nie dadzą żadnych sygnałów wyjściowych). Płytki Zero, RedBoard Turbo i Metro Express M0 mają pin A0 z przetwornikiem cyfrowo-analogowym (DAC), który wytwarza prawdziwy sygnał analogowy. Nie jest on przeznaczony do sterowania jasnością diody LED czy prędkością silnika (w takich przypadkach PWM działa lepiej), ale jest bardzo przydatny do generowania sygnałów audio, co zostało pokazane w recepturze 1.8.

Sterowanie światłem

Metoda sterowania światłem za pomocą cyfrowego lub analogowego wyjścia jest uniwersalna, efektywna i powszechnie stosowana. Pojedyncze diody LED, matryce złożone z diod LED, wyświetlacze numeryczne są szczegółowo omówione w recepturach tego rozdziału. Tekstowe i graficzne wyświetlacze LCD wymagają zastosowania innych technik, które zostały przedstawione w rozdziale 11.

Specyfikacja diody LED

Dioda LED jest elementem półprzewodnikowym z dwiema nóżkami, **anodą** i **katodą**. Kiedy napięcie na anodzie jest odpowiednio wyższe niż na katodzie (o wielkość zwaną **napięciem przewodzenia**), urządzenie emituje światło (fotony). Dłuższa nóżka jest zazwyczaj anodą, a na obudowie znajduje się często płaskie miejsce, które wskazuje na katodę (zobacz rysunek 7.2). Kolor diody LED i dokładna wartość napięcia przewodzenia zależą od jej budowy.

Standardowo napięcie przewodzenia czerwonej diody LED wynosi około 1,8 V. Jeśli napięcie na anodzie nie jest większe o 1,8 V od napięcia na katodzie, przez diodę LED nie przepłynie prąd i nie będzie żadnego światła. Gdy napięcie na anodzie będzie wyższe o 1,8 V niż na katodzie, dioda LED zapali się (będzie przewodzić prąd) i w efekcie spowoduje zwarcie. Musisz ograniczyć prąd za pomocą opornika, w przeciwnym wypadku dioda LED (wcześniej czy później) się spali. W recepturze 7.1 znajdziesz sposób obliczania wartości oporników, jakie należy zastosować dla danej diody.

Możliwe, że będziesz musiał sprawdzić dokumentację, aby wybrać diodę LED odpowiednią dla swojego projektu. W szczególności zwróć uwagę na wartość napięcia przewodzenia i maksymalne napięcie. Tabele 7.1 i 7.2 pokazują najważniejsze parametry, które należy sprawdzić w dokumentacji diody LED.

Tabela 7.1. Kluczowe dane zawarte w dokumentacji: bezwzględne wartości maksymalne

Parametr	Symbol	Wartość	Jednostki	Komentarz
Prąd przewodzenia	I_F	25	mA	Maksymalny prąd ciągły dla tej diody LED
Szczyt prądu przewodzenia (1/10 wypełnienia impulsu przy częstotliwości 1 kHz)	I_{FP}	160	mA	Maksymalny prąd impulsowy (dla impulsu, który jest w 1/10 włączony, a w 9/10 wyłączony)

Tabela 7.2. Kluczowe dane zawarte w dokumentacji: właściwości elektrooptyczne

Parametr	Symbol	Wartość	Jednostki	Komentarz
Intensywność świecenia	I_V	2	mcd	Jasność, gdy natężenie zasilania jest równe 2 mA
	I_V	40	mcd	Jasność, gdy natężenie zasilania jest równe 20 mA
Kąt widzenia		120	stopnie	Kąt promienia światła
Długość fali		620	nm	Dominanta lub szczytowa długość fali (kolor)
Napięcie przewodzenia	U_F	1,8	V	Napięcie diody, gdy się świeci

Piny płytek Arduino Uno, Leonardo i Mega mogą dostarczyć maksymalnie aż 40 mA. To bardzo dużo jak na typową diodę o średniej intensywności świecenia, ale niewystarczająco, aby zasilić diody o wyżej intensywności lub wiele diod podłączonych do jednego pinu. Receptura 7.3 pokazuje, jak wykorzystać tranzystor, aby zwiększyć natężenie płynące przez diodę LED.

Płytki 3,3 V mają mniejszą obciążalność prądową, dlatego sprawdź dokumentację swojej płytki, aby nie przekroczyć maksymalnych wartości.

Wielokolorowe diody LED składają się z co najmniej dwóch diod LED umieszczonych w jednej obudowie. Mogą mieć więcej niż dwa przewody, aby umożliwić niezależne sterowanie różnymi kolorami. Jest wiele wariantów obudów, więc aby się dowiedzieć, jak podłączyć taką diodę, sprawdź dokumentację.

Multipleksowanie

Aplikacje, które muszą sterować wieloma diodami LED, mogą wykorzystywać technikę zwaną **multipleksowaniem**. Ta technika działa na zasadzie przełączania po kolei grup diod LED (zazwyczaj ułożonych w rzędy lub kolumny). Receptura 7.12 pokazuje, jak 32 pojedyncze diody LED (8 diod LED na liczbę, w tym kropka dziesiętna), które tworzą wyświetlacz 4-cyfrowy, mogą być sterowane za pomocą tylko 12 pinów. Osiem pinów steruje segmentami wszystkich cyfr, a pozostałe cztery wybierają, która cyfra ma być aktywna. Wystarczająco szybki przebieg po wszystkich cyfrach (przynajmniej 25 razy na sekundę) tworzy wrażenie, że cyfry pozostają włączone, a nie pulsują — to **iluzja optyczna**.

Charlieplexing wykorzystuje technikę multipleksowania, biorąc pod uwagę fakt, że diody LED są **spolaryzowane** (świecą tylko wtedy, gdy anoda ma odpowiednio większe napięcie niż katoda), aby przełączać się między dwiema diodami przez odwrócenie ich biegunowości.

Maksymalne natężenie pinu

Diody LED mogą pobierać więcej prądu, niż mikrokontroler Arduino jest w stanie obsłużyć. Dokumentacja podaje bezwzględne wartości maksymalne dla chipa Arduino Uno (ATmega328P) równe 40 mA na pin. Mikrokontroler jest w stanie dostarczać i obniżać w sumie 200 mA, więc

zawsze musisz się upewnić, że natężenie całkowite nie przekracza tej wartości, na przykład pięć pinów zapewniających wyjście równe `HIGH` (dostarczanie) i pięć o wartości `LOW` (obniżanie), przy założeniu, że każdy pin ma 40 mA. Dobrą praktyką jest projektowanie układów w taki sposób, żeby działały dobrze przy maksymalnych bezwzględnych wartościach, co sprawi, że będą bardziej niezawodne. Dlatego warto utrzymywać napięcie nie większe niż 30 mA, aby zapewnić duży margines. W celach hobbystycznych, gdzie wymagane jest większe natężenie pinu, a mniejsza pewność działania jest do zaakceptowania, możesz doprowadzać do pinu natężenie do 40 mA, o ile limit dostarczania i obniżania natężenia równy 200 mA na chip nie zostanie przekroczony.

Zobacz omówienie receptury 7.3, aby się dowiedzieć, jak zwiększyć natężenie bez zewnętrznych tranzystorów.

Dokumentacja podaje 40 mA jako bezwzględną wartość maksymalną i niektórzy inżynierowie mogą nie być chętni do operowania bliskimi temu limitowi wartościami. Jednak 40 mA to już zaniżona przez Atmel wartość maksymalna, co oznacza, że piny mogą bezpiecznie obsługiwać takie natężenie. Dalsze receptury uwzględniają maksimum wynoszące 40 mA, ale jeśli budujesz układ, którego niezawodność działania jest istotna, to rozważnie jest obniżyć tę wartość do 30 mA, tak aby pozostawić margines. Pamiętaj jednak, że w przypadku płytek 3,3 V, a nawet niektórych 5 V, te wartości są niższe, na przykład dla Uno WiFi Rev 2 jest to 20 mA, a dla Zero — 7 mA. Jeśli korzystasz z innej płytki, sprawdź jej dokumentację.

7.1. Podłączanie i wykorzystywanie diod LED

Problem

Chcesz sterować co najmniej jedną diodą LED i dobrać odpowiedni opornik, aby jej nie zniszczyć.

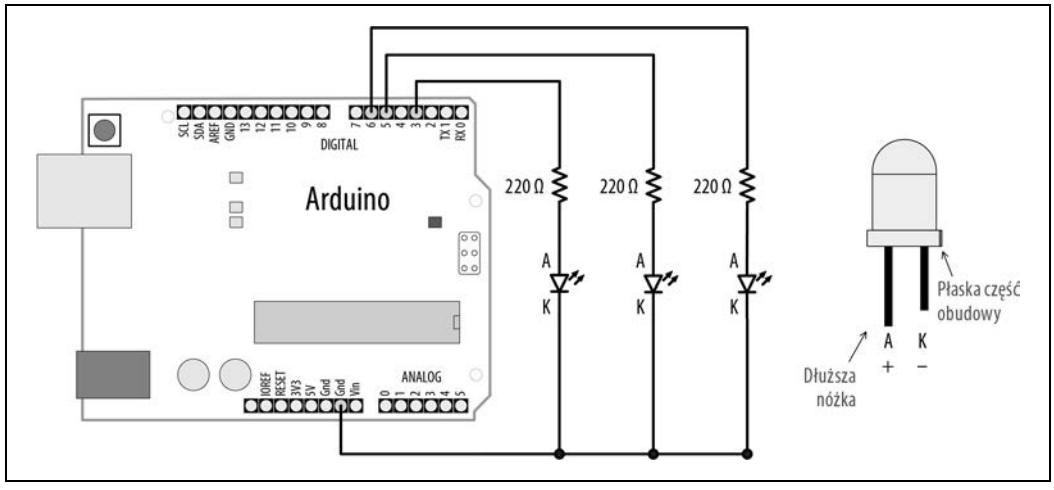
Rozwiązanie

Włączanie i wyłączanie diody LED za pomocą Arduino jest łatwe i niektóre receptury z poprzednich rozdziałów już to wykorzystywały (w recepturze 5.1 znajdziesz przykład sterowania wbudowaną diodą LED podłączoną do pinu 13). Ta receptura radzi, jak dobrać i stosować zewnętrzne diody LED. Na rysunku 7.2 pokazano podłączenie trzech diod LED, ale poniższy szkic możesz również uruchomić z podłączoną mniejszą liczbą diod.

Schematyczny symbol katody (pin ujemny) to *k*.

Poniższy szkic po kolei zapala na jedną sekundę trzy diody LED podłączone do pinów 3, 5 i 6.

```
/*
 * Szkic blink_leds
 * Miganie trzema diodami LED podłączonymi do osobnych pinów
 */
```



Rysunek 7.2. Podłączenie zewnętrznych diod LED

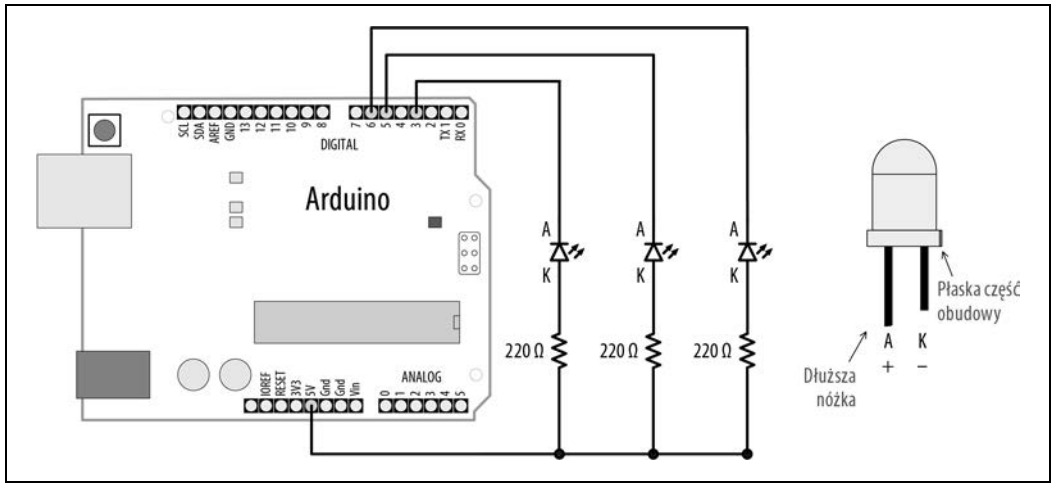
```
const int firstLedPin  = 3; // Pin dla każdej diody LED
const int secondLedPin = 5;
const int thirdLedPin  = 6;

void setup()
{
  pinMode(firstLedPin, OUTPUT);  // Ustawienie pinu diody LED jako wyjścia
  pinMode(secondLedPin, OUTPUT); // Ustawienie pinu diody LED jako wyjścia
  pinMode(thirdLedPin, OUTPUT);  // Ustawienie pinu diody LED jako wyjścia
}

void loop()
{
  // Zapalenie każdej diody LED na 1000 ms (1 s)
  blinkLED(firstLedPin,  1000);
  blinkLED(secondLedPin, 1000);
  blinkLED(thirdLedPin,  1000);
}


// Funkcja powodująca miganie diodą LED podłączoną do podanego pinu i w podanych w  milisekundach
// odstępach czasowych
void blinkLED(int pin, int duration)
{
  digitalWrite(pin, HIGH);      // Zapalenie diody LED
  delay(duration);
  digitalWrite(pin, LOW);       // Zgaszenie diody LED
  delay(duration);
}
```

W funkcji `setup` szkic ustawia tryb wyjścia na pinach, do których podłączone są diody LED. Pętla główna `loop` wywołuje funkcję `blinkLED`, aby każda dioda mignęła. Funkcja `blinkLED` ustawia na 1 sekundę (1000 ms) stan wysoki (`HIGH`) na wskazanym pinie.

Omówienie

Ponieważ anody podłączone są do pinów Arduino, a katody do masy, diody LED zapalą się, gdy pin będzie miał stan `HIGH`, a zgasną, gdy pin osiągnie stan `LOW`. Diodę LED możesz zapalić, kiedy pin ma stan `LOW`, podłączając katody do pinów, a anody do masy (opornik może być podłączony do dowolnej nóżki).

Jeśli anody diod LED są podłączone do +5 V tak, jak to zostało pokazane na rysunku 7.3, diody zaświecą się, gdy pin będzie miał stan `LOW` (efekt wizualny będzie odwrotny — jedna z diod wyłączy się na sekundę, podczas gdy pozostałe dwie będą się świecić).



Rysunek 7.3. Podłączenie katod zewnętrznych diod LED do pinów

Diody do kontroli natężenia wymagają oporników podłączonych szeregowo, inaczej mogłyby szybko się wypalić. Zewnętrzna dioda LED musi mieć podłączony szeregowo opornik, albo do katody, albo do anody.

Opornik połączony szeregowo z diodą LED służy do regulacji ilości prądu, który będzie przepływał przez diodę. Aby obliczyć wielkość opornika, musisz znać napięcie źródła (U_{ZAS}, zazwyczaj 5 V), napięcie przewodzenia diody (U_F) oraz ilość prądu (I), która ma przepływać przez diodę.

Wzór na opór w omach (znany jako prawo Ohma) to:

$$R = (U_{ZAS} - U_F) : I$$

Na przykład w przypadku napięcia przewodzenia równego 1,8 V, prądu równego 15 mA i napięcia zasilania równego 5 V do wzoru należy podstawić następujące wartości:

$U_{ZAS} = 5$ (dla płytek Arduino 5 V)

$U_F = 1{,}8$ (napięcie przewodzenia diody LED)

$I = 0{,}015$ (1 mA to jedna tysięczna ampera, więc 15 mA to 0,015 A)

Napięcie włączonej diody to (U_{ZAS} – U_F), czyli 5 – 1,8, co daje 3,2.

Wartość opornika należy zatem obliczyć w następujący sposób: 3,2 : 0,015 = 213 omów.

Wartość 213 omów nie jest standardową wielkością opornika, więc możesz zaokrąglić ją w górę, do 220 omów.

Opornik pokazany na rysunku 7.2 umieszony jest między katodą a masą, ale może być podłączony do drugiej strony diody LED (między źródłem napięcia a anodą).

Arduino Uno i Mega mają określone maksymalne natężenie na pinach równe 40 mA. Jeśli Twoja dioda LED potrzebuje więcej prądu, niż płytka jest w stanie dostarczyć, zobacz recepturę 7.3.

Zobacz również

Zapoznaj się z recepturą 7.3.

7.2. Dostosowywanie jasności diody LED

Problem

Chcesz za pomocą szkicu sterować jasnością jednej lub kilku diod LED.

Rozwiązanie

Podłącz każdą diodę LED do analogowego wyjścia (PWM). Wzoruj się na schemacie z rysunku 7.2. Szkic będzie rozjaśniał diodę(y) LED, od całkowitego wygaszenia do pełnej jasności, i wygaszał. Każdy taki cykl będzie trwał około 5 sekund.

```
/*
 * Szkic led_brightness
 * Steruje jasnością diod LED podłączonych do analogowych portów wyjścia
 */

const int firstLed   = 3; // Pin dla każdej diody LED
const int secondLed  = 5;
const int thirdLed   = 6;

int brightness = 0;
int increment  = 1;

void setup()
{
  // Piny sterowane przez analogWrite nie muszą być zadeklarowane jako wyjścia
}

void loop()
{
  if(brightness > 255)
```

```
  {
    increment = -1; // Po osiągnięciu 255 zacznij odliczać w dół
  }
  else if(brightness < 1)
  {
    increment =  1; // Po osiągnięciu 0 zacznij odliczać w górę
  }
  brightness = brightness + increment; // Inkrementacja (lub dekrementacja)

  // Zapisanie wartości jasności na pinie diody LED
  analogWrite(firstLed, brightness);
  analogWrite(secondLed, brightness);
  analogWrite(thirdLed, brightness );

  delay(10); // 10 ms na każdą zmianę kroku oznacza 2,55 s na rozjaśnienie lub wygaszenie
}
```

Omówienie

Powyższy szkic korzysta z tego samego obwodu co poprzedni, ale tutaj piny są sterowane za pomocą `analogWrite` zamiast `digitalWrite`. Polecenie `analogWrite` korzysta z funkcji PWM, aby sterować mocą dostarczaną do diody LED. Zobacz wprowadzenie do tego rozdziału, gdzie przeczytasz więcej o analogowym wyjściu.

Przy każdym obiegu pętli szkic zmienia poziom jasności na mniejszy i większy przez zwiększenie wartości zmiennej `brightness` (rozjaśnianie) lub przez jej zmniejszenie (ściemnianie). Następnie zmienna `brightness` przekazywana jest do `analogWrite` dla trzech podłączonych diod LED. Minimalna wartość, jaka może być przekazana do `analogWrite`, to 0, co powoduje utrzymanie na pinie napięcia równego 0. Maksymalna wartość natomiast to 255 (5 V na płytkach 5 V, a 3,3 V na płytkach 3,3 V).

Dobrą praktyką jest ograniczenie wartości zakresu od 0 do 255, gdyż liczby spoza zakresu mogą dawać nieoczekiwane rezultaty (zobacz recepturę 3.5).

Kiedy wartość zmiennej `brightness` osiągnie maksimum, zacznie maleć, ponieważ znak zmiennej `increment` zmienia się z +1 na –1 (dodawanie –1 do danej wartości jest tożsame z odejmowaniem 1 od tej wartości).

Zobacz również

Wprowadzenie do tego rozdziału wyjaśnia, jak działa wyjście analogowe.

W przypadku płytek takich jak Due, Zero czy MKR1000 funkcja PWM może osiągać maksymalnie wartość równą 4095, choć ich wartością domyślną jest standardowo 255. Jeśli potrzebujesz większej rozdzielczości, możesz ją ustawić za pomocą funkcji `analogWriteResolution` (*https://www.arduino.cc/reference/en/language/functions/zero-due-mkr-family/analogwriteresolution/*).

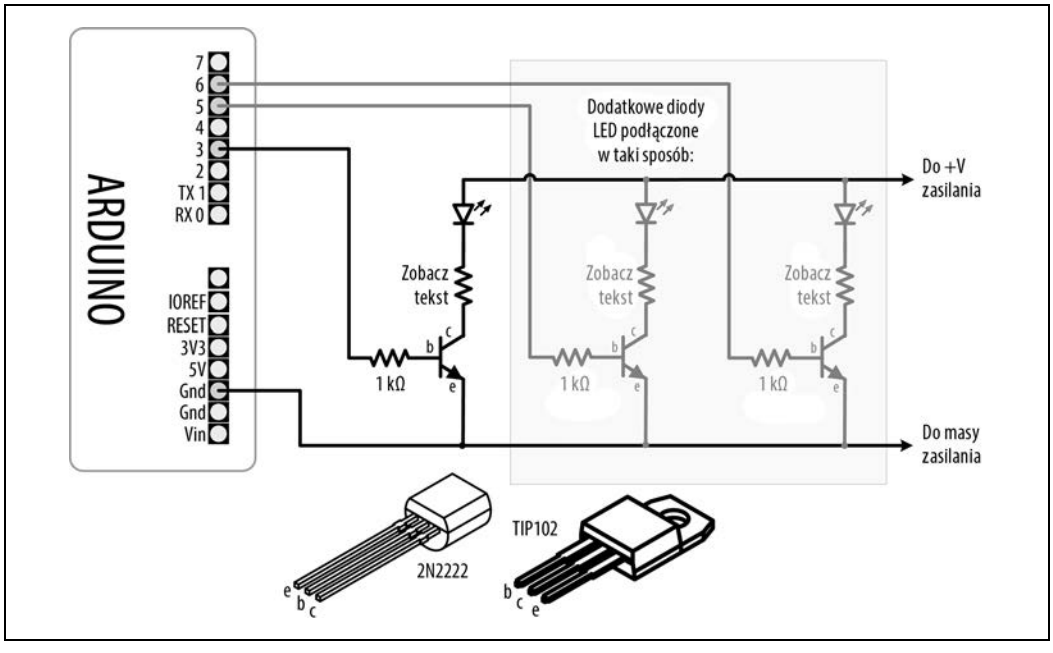
7.3. Korzystanie z diod LED wysokiej mocy

Problem

Musisz przełączać lub sterować jasnością diody LED, która wymaga więcej mocy, niż Arduino jest w stanie dostarczyć. Maksymalne natężenie na pinach Arduino Uno i Mega wynosi 40 mA.

Rozwiązanie

Użyj tranzystora, aby włączać i wyłączać prądy płynące przez diodę LED. Diodę podłącz tak, jak pokazano na rysunku 7.4. Możesz wykorzystać ten sam kod co w recepturach 7.1 i 7.2 (upewnij się tylko, że piny podłączone do bazy tranzystora pasują do numerów pinów podanych w szkicu).



Rysunek 7.4. Wykorzystanie tranzystora do sterowania diodami LED wysokiej mocy

Omówienie

Na rysunku 7.4 widoczna jest strzałka wskazująca +V zasilania. Może to być pin zasilania +5 V, który może dostarczyć maksymalnie 400 mA, jeśli jest zasilany przez USB. Dzięki zasilaniu płytki zewnętrznym źródłem dostępny prąd jest zależny od wartości natężenia i napięcia zasilacza DC (stabilizator odprowadza nadmiar napięcia w postaci ciepła — sprawdź, czy stabilizator płytki, 3-pinowy chip znajdujący się zazwyczaj obok gniazda zasilania, nie jest zbyt gorący, aby go dotykać). Jeśli wymagana jest większa ilość prądu, niż Arduino +5 V jest w stanie zapewnić, musisz zasilać diody LED osobnym źródłem. W dodatku C znajdziesz więcej informacji na temat zewnętrznych źródeł zasilania.

Jeśli korzystasz z zewnętrznego źródła zasilania, pamiętaj, aby podłączyć jego masę do masy Arduino.

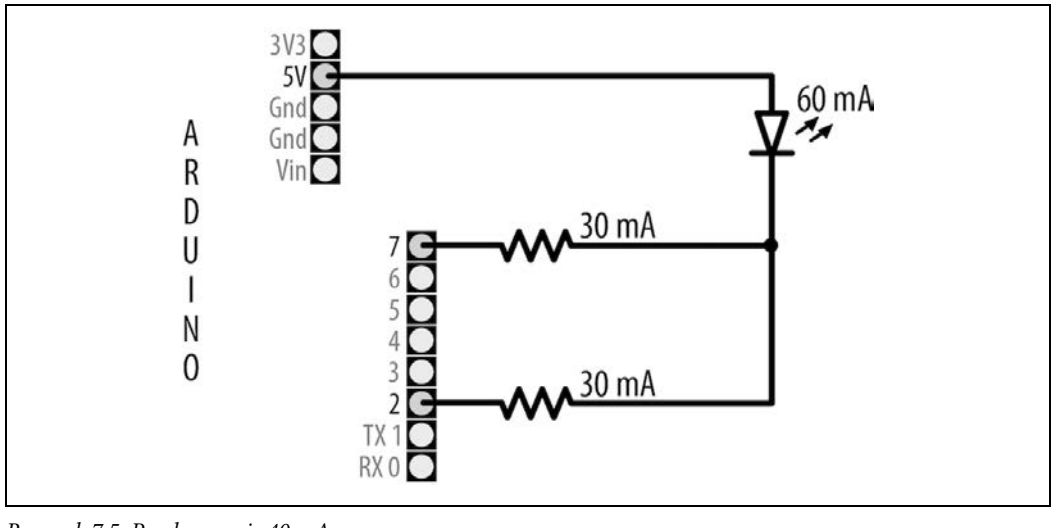
Gdy tranzystor jest włączony, prąd może płynąć od kolektora do emitera. Kiedy jest wyłączony, nie płynie przez niego żaden znaczący prąd. Arduino może włączyć tranzystor przez ustawienie stanu wysokiego (`HIGH`) na pinie za pomocą funkcji `digitalWrite`. Między pinem a bazą tranzystora niezbędny jest opornik, który chroni przed zbyt wysokim prądem. Standardowa wartość takiego opornika to 1 kΩ (do bazy tranzystora dostarczany jest prąd 5 mA). W dodatku B znajdziesz wskazówki, jak czytać dokumentację oraz wybrać i stosować tranzystor. Do sterowania wieloma wyjściami możesz zastosować wyspecjalizowane obwody zintegrowane, na przykład ULN2003A, które zawierają siedem sterowników wyjścia o wysokim natężeniu (0,5 A).

Wartość opornika ograniczającego przepływ prądu przez diodę LED jest obliczana za pomocą wzoru omówionego w recepturze 7.1, ale musisz wziąć pod uwagę zredukowane przez tranzystor napięcie źródła. Zazwyczaj jest to mniej niż trzy czwarte wolta (rzeczywista wartość to napięcie nasycenia kolektor – emiter, zobacz dodatek B). Diody LED wysokiej mocy (1 W lub więcej) najlepiej sterowane są przez źródło prądu stałego (obwód, który aktywnie kontroluje natężenie), co pozwala zarządzać jego przepływem przez diodę.

Przekroczenie 40 mA na chipie ATmega

Jeśli płytka korzysta z chipa ATmega, możesz podłączyć wiele pinów naraz, aby przekroczyć natężenie 40 mA na pinie (zobacz podpunkt „Maksymalne natężenie pinu" we wcześniejszej części rozdziału).

Rysunek 7.5 pokazuje, jak podłączyć diodę LED zasilaną przez 60 mA za pomocą dwóch pinów. Dioda przez piny 2 i 7 łączy oporniki z masą. Oba piny muszą mieć stan niski (`LOW`), aby 60 mA przepływało przez diodę. Są potrzebne dwa osobne oporniki, dlatego nie próbuj użyć jednego opornika w celu podłączenia dwóch pinów.



Rysunek 7.5. Przekroczenie 40 mA

Ta technika może być również stosowana w przypadku źródła prądu. Na przykład odwróć diodę LED — nóżkę, która była podłączona do oporników (katoda), połącz z masą, a drugą (anoda) z opornikami. W takim układzie diodę zapalasz przez ustawienie stanu wysokiego (`HIGH`) na obu pinach.

Najlepiej, jeśli wybierzesz niesąsiadujące ze sobą piny, tak aby zminimalizować obciążenie chipa. Ta technika będzie działać z każdym pinem korzystającym z `digitalWrite;`. Nie będzie natomiast działać z `analogWrite`, więc jeżeli potrzebujesz więcej prądu dla wyjść analogowych (PWM), musisz zastosować tranzystor tak, jak już to zostało wcześniej omówione.

Ta technika nie jest zalecana dla płytek 32-bitowych.

Zobacz również

Dokumentacja sterowników prądu stałego znajduje się na stronie *http://www.neufeld.newton.ks.us/electronics/?p=475* (w języku angielskim).

7.4. Dostosowywanie koloru diody LED

Problem

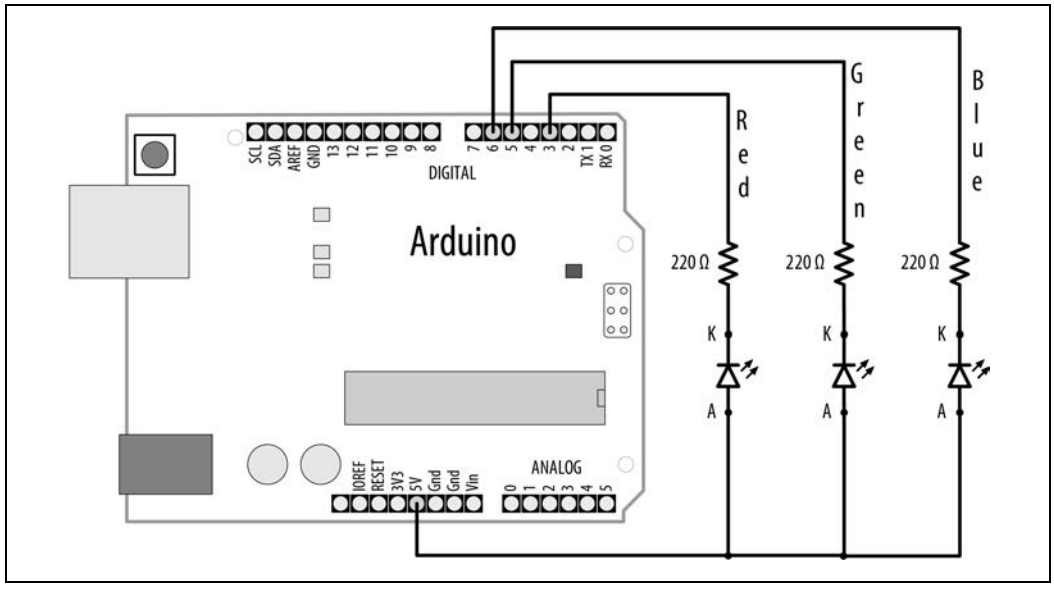
Chcesz sterować kolorem diody RGB LED za pomocą szkicu.

Rozwiązanie

Diody RGB LED mają czerwony, zielony i niebieski element w jednej obudowie, albo z połączonymi ze sobą anodami (znane jako diody RGB LED ze **wspólną anodą**), albo z połączonymi ze sobą katodami (diody RGB LED ze **wspólną katodą**). Diodę RGB LED ze wspólną anodą podłącz tak, jak na rysunku 7.6 (anody podłączone są do +5 V, a katody do pinów). W przypadku diody RGB LED ze wspólną katodą możesz wykorzystać schemat podłączeniowy pokazany na rysunku 7.2.

Ten szkic nieprzerwanie zmienia spektrum koloru przez zmianę intensywności czerwonego, zielonego i niebieskiego.

```
/*
 * Szkic RGB
 * Diody RGB LED sterowane za pomocą pinów wyjścia z funkcją PWM
 */

const int redPin   = 3;    // Pin dla każdej diody LED
const int greenPin = 5;
const int bluePin  = 6;
const bool invert = true; // Ustaw true dla diody ze wspólną anodą, a false w przypadku wspólnej katody

int color = 0; // Reprezentująca odcień wartość od 0 do 255
int R, G, B;    // Składowe elementy koloru: czerwony, zielony, niebieski

void setup()
{
  // Piny sterowane przez analogWrite nie muszą być zadeklarowane jako wyjścia
}
```



Rysunek 7.6. Podłączenie diody RGB LED ze wspólną anodą

```
void loop()
{
  int brightness = 255; // 255 to maksymalna jasność

  hueToRGB(color, brightness); // Wywołanie funkcji zmieniającej odcień na RGB

  // Zapis wartości RGB na pinach
  analogWrite(redPin,   R);
  analogWrite(greenPin, G);
  analogWrite(bluePin,  B);

  color++; // Zwiększenie koloru
  if (color > 255)
    color = 0;
  delay(10);
}

// Funkcja zmieniająca kolor na jego elementy składowe: czerwony, zielony, niebieski
void hueToRGB(int hue, int brightness)
{
  unsigned int scaledHue = (hue * 6);

  // Segment od 0 do 5 na kole kolorów
  unsigned int segment = scaledHue / 256;

  // Pozycja w segmencie
  unsigned int segmentOffset = scaledHue - (segment * 256);

  unsigned int complement = 0;
  unsigned int prev = (brightness * ( 255 -  segmentOffset)) / 256;
  unsigned int next = (brightness *  segmentOffset) / 256;
```

```
  if (invert)
  {
    brightness = 255 - brightness;
    complement = 255;
    prev = 255 - prev;
    next = 255 - next;
  }

  switch (segment) {
    case 0:     // Czerwony
      R = brightness;
      G = next;
      B = complement;
      break;
    case 1:     // Żółty
      R = prev;
      G = brightness;
      B = complement;
      break;
    case 2:     // Zielony
      R = complement;
      G = brightness;
      B = next;
      break;
    case 3:     // Niebieskozielony
      R = complement;
      G = prev;
      B = brightness;
      break;
    case 4:     // Niebieski
      R = next;
      G = complement;
      B = brightness;
      break;
    case 5:     // Magenta
    default:
      R = brightness;
      G = complement;
      B = prev;
      break;
  }
}
```

Omówienie

Kolor diody RGB LED jest zależny od intensywności jego składowych: czerwonego, zielonego i niebieskiego. Kluczowa funkcja szkicu (`hueToRGB`) zamienia zakres wartości odcienia, od 0 do 255, na odpowiedni zakres koloru, od czerwonego do niebieskiego. Widmo widzialnych kolorów jest często przedstawiane w postaci koła złożonego z kolorów podstawowych i pochodnych z gładkimi przejściami między sąsiadującymi kolorami. Każda z sześciu części koła barw, przedstawiających kolory podstawowe i pochodne, w szkicu obsługiwana jest przez sześć wyrażeń `case`. Jeśli zmienna `segment` pasuje do numeru opcji, kod w bloku danego wyrażenia `case` jest wykonywany i ustawiana jest odpowiednia wartość dla każdego koloru (`R`, `G`, `B`). Segment 0 to czerwony, segment 1 to żółty, segment 2 to zielony i tak dalej.

Jeśli dodatkowo chcesz dostosować jasność, możesz zmniejszyć wartość zmiennej `brightness`. Poniższa linijka kodu pokazuje, jak dostosować jasność za pomocą rezystora nastawnego lub czujnika, podłączonego tak, jak na rysunkach 7.14 i 7.18.

```
int brightness = map(analogRead(A0),0,1023,0,255);
```

Zmienna `brightness` będzie mieścić się w zakresie od 0 do 255, relatywnie do zakresu analogowego wejścia, od 0 do 1023, co spowoduje, że dioda będzie świecić jaśniej wraz ze wzrostem wartości wejścia.

Zobacz również

Zapoznaj się z recepturą 2.16.

7.5. Sterowanie wieloma kolorowymi diodami LED

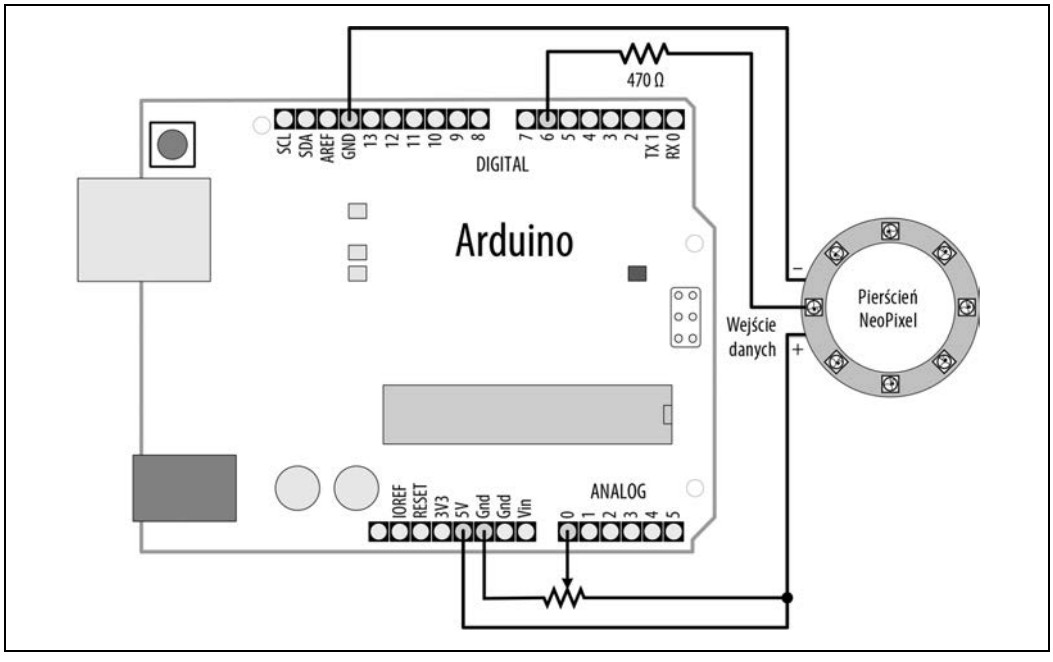
Problem

Chcesz sterować kolorem wielu diod LED za pomocą jednego pinu.

Rozwiązanie

Ta receptura pokazuje, jak stosować inteligentne diody RGB LED z wbudowanym w każdej diodzie maleńkim sterownikiem, który umożliwia sterowanie wieloma diodami RGB LED za pomocą jednego cyfrowego pinu. Do zmiany kolorów diod RGB LED opartych na odczytach z pinu analogowego szkic wykorzystuje bibliotekę Adafruit Neopixels (zainstalowaną za pomocą menedżera bibliotek w Arduino IDE). Rysunek 7.7 pokazuje, jak podłączyć pierścień NeoPixel i potencjometr do sterowania kolorem.

```
/*
 * Szkic neo
 * Kolor diody RGB LED zmienia się zależnie od wartości analogowego pinu
 */

#include <Adafruit_NeoPixel.h>

const int sensorPin = A0; // Analogowy pin czujnika
const int ledPin = 6;     // Pin taśmy LED
const int count  = 16;    // Liczba diod LED w taśmie

// Deklaracja taśmy LED
Adafruit_NeoPixel leds = Adafruit_NeoPixel(count, ledPin, NEO_GRB + NEO_KHZ800);

void setup() {
  leds.begin(); // Inicjalizacja taśmy LED
  for (int i = 0; i < count; i++) {
    leds.setPixelColor(i, leds.Color(0,0,0)); // Wyłączenie każdej diody LED
  }
  leds.show();  // Odświeżenie taśmy LED nowymi wartościami pikseli  (wyłączenie wszystkich)
}
```



Rysunek 7.7. Podłączenie pierścienia NeoPixel

```
void loop() {
  static unsigned int last_reading = -1;

  int reading =  analogRead(sensorPin);
  if (reading != last_reading) { // Jeśli wartość się zmieniła,
    // przeskaluj odczyty analogowe do zakresu kolorów taśmy NeoPixel
    unsigned int mappedSensorReading = map(reading, 0, 1023, 0, 65535);

    // Zaktualizuj piksele z niewielkim opóźnieniem, aby stworzyć efekt pływającego światła
    for (int i = 0; i < count; i++) {
      leds.setPixelColor(i, leds.gamma32(leds.ColorHSV(mappedSensorReading, 255, 128)));
      leds.show();
      delay(25);
    }
    last_reading = reading;
  }
}
```

Jeśli używasz płytki 3,3 V, musisz połączyć zarówno dodatni przewód potencjometru, jak i pierścienia NeoPixel do pinu 3,3 V zamiast 5 V.

Omówienie

Ten szkic steruje pałką, taśmą lub grupą połączonych ośmiu diod RGB LED — NeoPixel firmy Adafruit. Możesz zmienić wartość zmiennej `numOfLeds`, jeśli podłączyłeś inną liczbę diod RGB LED, ale miej na uwadze fakt, że każda z nich może pobierać do 60 mA (w przypadku koloru

białego świecącego pełną jasnością). Port USB może zasilić do ośmiu diod RGB LED, ale dla większej ich liczby musisz podłączyć złącza zasilania taśmy do zasilacza 5 V o wysokim natężeniu, a jego masę podłączyć do masy Arduino. Jeśli używasz płytki 3,3 V, to nie zasilaj NeoPixel napięciem wyższym niż 3,7 V (na przykład polimerową baterią litowo-jonową), ponieważ NeoPixel wymaga sygnału danych, który ma napięcie bliskie swojemu zasilaniu. Jeśli używasz zewnętrznego źródła zasilania, powinieneś również podłączyć kondensator 1000 µf między ujemnym i dodatnim pinem źródła, aby chronić piksele (sprawdź bieguny kondensatora i upewnij się, że podłączasz go prawidłowo).

Zmienna `leds` jest zadeklarowana za pomocą następującej linijki kodu:

```
Adafruit_NeoPixel leds = Adafruit_NeoPixel(count, ledPin, NEO_GRB + NEO_KHZ800);
```

W ten sposób zostaje stworzona struktura, która przechowuje kolor każdej diody LED i komunikuje się z taśmą. Musisz określić liczbę diod LED w taśmie (`count`), pin Arduino, do którego podłączona jest linia danych (`ledPin`), i typ taśmy LED (w tym przypadku `NEO_GRB+NEO_KHZ800`). Dla swojej taśmy LED musisz sprawdzić dokumentację, gdzie znajdziesz informację o bibliotece i dowiesz się, czy musisz użyć innych ustawień, ale nic nie uszkodzisz, jeśli będziesz próbował użyć kolejnych opcji z biblioteki, aby sprawdzić, która działa.

Aby ustawić kolor pojedynczej diody RGB LED, użyj metody `led.setPixelColor`. Musisz podać numer diody (zaczynając od 0 dla pierwszej z nich) i wybrany kolor. W celu przesłania danych do diody LED należy wywołać funkcję `led.show`. Przed jej wywołaniem możesz zmienić kolory wielu diod, aby zmieniły się jednocześnie. Niezmienione diody LED będą miały taki kolor jak poprzednio. Podczas tworzenia obiektu `Adafruit_Neo Pixel` wszystkie wartości mają początkową wartość 0.

Biblioteka NeoPixel zawiera swoją własną funkcję zamiany odcienia na wartość RGB: `ColorHSV`. Pierwszym argumentem jest odcień, drugim nasycenie koloru, a trzeci to jasność. Funkcja `gamma32` konwertuje `ColorHSV` na wyjściu, aby zniwelować różnicę między tym, jak komputery reprezentują kolory, a tym, jak ludzie je postrzegają.

Każdy piksel (dioda LED) jest połączony z wejściem i wyjściem danych, zasilaniem oraz masą. Arduino steruje wejściem danych pierwszego piksela, wyjściem danych, które jest połączone z wejściem danych następnego piksela w łańcuchu. Możesz kupić pojedyncze piksele lub taśmy, które są już połączone.

Jeśli Twoja taśma nie jest wspierana przez bibliotekę Adafruit

Pierwsze taśmy LED używały chipa WS2811. Od tamtego czasu powstały inne wersje, na przykład WS2812, WS2812B i APA102. Jeśli Twoje diody LED nie są obsługiwane przez bibliotekę stworzoną przez Adafruit, wypróbuj bibliotekę *Fast LED* (*http://fastled.io/*).

Kompatybilna z Arduino płytka Teensy 3.x lub wyższa (*https://www.pjrc.com/teensy/*) może sterować ośmioma taśmami na różnych pinach i korzysta z połączenia bardzo szybkiego sprzętu i oprogramowania, aby umożliwić tworzenie wysokiej jakości animacji.

Diody LED są dostępne osobno, ale także na giętkich taśmach nawiniętych na rolki, z różnymi rozstawami diod (określonymi jako liczba diod LED na metr lub stopę). Firma Adafruit produkuje szeroką gamę kształtów płytek drukowanych, w tym koła diod LED, krótkie taśmy i panele, sprzedawane pod marką NeoPixel.

Zobacz również

Więcej informacji na temat produktów NeoPixel znajdziesz w przewodniku na stronie Adafruit: *https://learn.adafruit.com/adafruit-neopixel-uberguide/the-magic-of-neopixels* (w języku angielskim).

Biblioteka Teensy (*https://www.pjrc.com/teensy/td_libs_OctoWS2811.html*), która również zawiera dobre rysunki przedstawiające, jak podłączyć źródła zasilania z dużą liczbą diod LED, oraz program, który wyciąga dane z wideo, które możesz dodać do swojego programu, który będzie wyświetlał to wideo na taśmach LED.

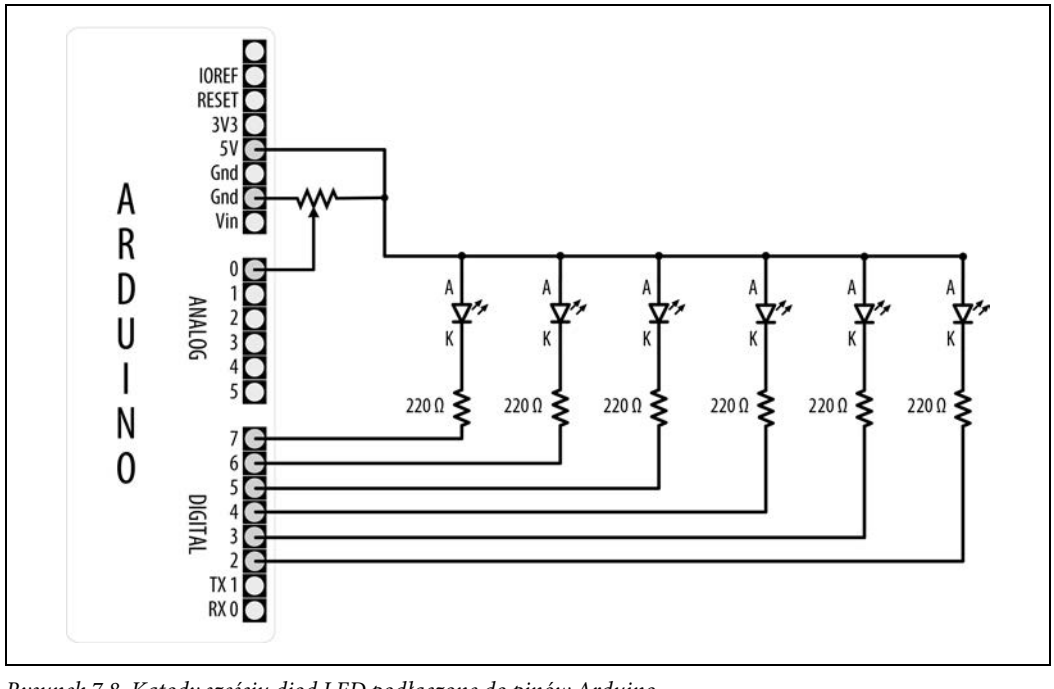
7.6. Sterowanie wieloma diodami LED — tworzenie wykresu słupkowego

Problem

Chcesz stworzyć LED-owy wykres słupkowy przedstawiający wartości z Twojego szkicu lub wartości odczytane z czujnika.

Rozwiązanie

Diody LED możesz podłączyć zgodnie ze schematem pokazanym na rysunku 7.2 (jeśli chcesz mieć więcej diod, użyj dodatkowych pinów). Rysunek 7.8 pokazuje sześć diod LED podłączonych do sąsiadujących ze sobą pinów.



Rysunek 7.8. Katody sześciu diod LED podłączone do pinów Arduino

Następny szkic włącza wiele diod LED, których liczba jest proporcjonalna do wartości podłączonego do analogowego pinu czujnika (zobacz rysunek 7.14 lub rysunek 7.8, aby sprawdzić, jak należy podłączyć czujnik).

```
/*
 * Szkic bargraph
 * Włącza diody LED, których liczba jest proporcjonalna do wartości czujnika analogowego
 * Ten szkic steruje sześcioma diodami LED, ale możesz to zmienić; dostosuj wartości zmiennej NbrLEDs
 * i dodaj piny do tablicy ledPins
 */

const int ledPins[] = {2, 3, 4, 5, 6, 7};
const int NbrLEDs = 6
const int analogInPin = A0; // Analogowy pin wejścia podłączony do zmiennego rezystora


// Zamień wartości dwóch poniższych definicji, jeśli katody podłączone są do masy
#define LED_ON  LOW
#define LED_OFF HIGH


int sensorValue = 0; // Wartość odczytana z czujnika
 int ledLevel = 0;    // Wartość czujnika zamieniona na słupki diod LED

void setup() {
  for (int led = 0; led < NbrLEDs; led++)
  {
    pinMode(ledPins[led], OUTPUT);  // Ustawienie wszystkich pinów diod LED jako wyjścia
  }
}

void loop() {
  sensorValue = analogRead(analogInPin);              // Odczyt wartości analogowej wejścia
  ledLevel = map(storedValue, 10, 1023, 0, NbrLEDs); // Dostosowanie do liczby diod LED

  for (int led = 0; led < NbrLEDs; led++)
  {
    if (led < ledLevel ) {
      digitalWrite(ledPins[led], LED_ON);  // Włączenie pinów poniżej określonego poziomu
    }
    else {
      digitalWrite(ledPins[led], LED_OFF);    // Wyłączenie pinów powyżej określonego poziomu
    }
  }
}
```

Omówienie

Piny, do których podłączone są diody LED, przechowywane są w tablicy `ledPins`. Aby zmienić liczbę diod, możesz dodać (lub usunąć) elementy w tablicy, lecz upewnij się, że stała `NbrLEDs` jest równa rozmiarowi tablicy (liczbie pinów zapisanych w tablicy). Kompilator może za Ciebie obliczyć wartość zmiennej `NbrLEDs`, wystarczy, że tę linijkę:

```
const int NbrLEDs = 6;
```

zastąpisz następującą linijką:

```
const int NbrLEDs = sizeof(ledPins) / sizeof(ledPins[0]);
```

Funkcja `sizeof` zwraca rozmiar (liczbę bajtów) zmiennej, w tym przypadku liczbę bajtów w tablicy `ledPins`. Ponieważ jest to tablica przechowująca liczby całkowite `int` (2 bajty na element), łączna liczba bajtów w tablicy jest dzielona przez rozmiar pojedynczego elementu (`sizeof(ledPins[0])`), co daje liczbę elementów.

Dostępna w Arduino IDE funkcja `map` została wykorzystana do obliczenia proporcjonalnej do wartości czujnika liczby diod LED, które powinny być zapalone. Pętla sprawdza każdą diodę przez włączenie jej, jeżeli wartość jest proporcjonalnie większa niż numer diody. Na przykład jeśli wartość czujnika jest mniejsza niż 10, to żadna dioda nie zostanie zaświecona, a jeśli jest równa połowie tej wartości, połowa diod LED zostanie zapalona. W idealnych warunkach najmniejsza wartość, jaką może zwrócić potencjometr, to zero, ale jest to mało prawdopodobne w rzeczywistym świecie. Kiedy czujnik osiągnie maksymalną wartość, wszystkie diody LED będą świecić. Jeśli zauważysz, że ostatnia z nich miga, gdy potencjometr jest ustawiony na maksymalną wartość, spróbuj zmniejszyć drugi argument funkcji `map` z `1023` na `1000` lub zbliżoną liczbę.

Na rysunku 7.8 widać, że wszystkie anody są ze sobą połączone (wspólna anoda), a katody podłączone są do pinów. Piny muszą mieć stan niski (`LOW`), aby diody LED się świeciły. Natomiast jeśli do pinów podłączone są anody (tak, jak widać na rysunku 7.2), a katody są ze sobą połączone (wspólna katoda), dioda będzie się świecić, gdy pin będzie miał wartość `HIGH`. W tej recepturze szkic używa stałych `LED_OFF` i `LED_ON`, aby było łatwiej wybrać jedną z opcji, wspólną anodę lub wspólną katodę. Żeby przystosować kod do diod LED ze wspólną katodą, zamień w następujący sposób wartości tych stałych:

```
const bool LED_ON = HIGH; // HIGH oznacza zapaloną diodę w przypadku wspólnej katody
const bool LED_OFF = LOW;
```

Być może zechcesz zmniejszyć prędkość zmiany świateł, na przykład aby naśladować wskaźnik poziomu głośności. Oto wersja szkicu, który powoli wygasza słupki diod LED, gdy wartość czujnika spada:

```
/*
 * Szkic bargraph_decay
 *

 */

const int ledPins[] = {2, 3, 4, 5, 6, 7};
const int NbrLEDs = sizeof(ledPins) / sizeof(ledPins[0]);
const int analogInPin = A0; // Analogowy pin wejścia podłączony do zmiennego rezystora
const int decay = 10;       // Zwiększenie tej liczby zmniejszy wartość wygaszania

// Zamień wartości dwóch poniższych definicji, jeśli katody są podłączone do masy
#define LED_ON  LOW
#define LED_OFF HIGH

// Wartość wygaszania
int storedValue = 0;
```

```
void setup() {
  for (int led = 0; led < NbrLEDs; led++)
  {
    pinMode(ledPins[led], OUTPUT);   // Ustawienie wszystkich pinów diod LED jako wyjścia
  }
}

void loop() {
  int sensorValue = analogRead(analogInPin);              // Odczyt wartości analogowej wejścia
  storedValue = max(sensorValue, storedValue);            // Jeśli wartość storedValue jest wyższa,
                                                          // użyj  wartości czujnika
  int ledLevel = map(storedValue, 10, 1023, 0, NbrLEDs); // Dostosowanie do liczby diod LED

  for (int led = 0; led < NbrLEDs; led++)
  {
    if (led < ledLevel ) {
      digitalWrite(ledPins[led], LED_ON);   // Włączenie pinów poniżej określonego poziomu
    }
    else {
      digitalWrite(ledPins[led], LED_OFF); // Wyłączenie pinów powyżej określonego poziomu
    }
  }
  storedValue = storedValue - decay;        // Obniżenie wartości wygaszania o prędkość wygaszania
  delay(10);                                   // Poczekaj 10 s przed wykonaniem następnego obiegu pętli
}
```

Wygaszanie jest obsługiwane przez linijkę, która używa funkcji `max`. W rezultacie zostaje zwrócona albo wartość czujnika, albo zapisana wartość wygaszania, w zależności od tego, która jest wyższa. Jeśli wyższa jest wartość czujnika, zostaje przypisana do zmiennej `storedValue`. W przeciwnym razie zmienna `storedValue` zostaje pomniejszona o stałą wartość zmiennej `decay` przy każdym obiegu pętli (wykonującej się z 10-milisekundowymi przerwami, zgodnie ze znajdującym się na końcu poleceniem `delay`). Zwiększenie wartości stałej `decay` spowoduje skrócenie czasu, w jakim diody LED zgasną.

Możesz stworzyć taki wykres słupkowy z wykorzystaniem produktu NeoPixel z poprzedniej receptury. Kod dla takiego projektu wyglądałby następująco:

```
/*
 * Szkic bargraph_neo
 * Włącza diody LED, których liczba jest proporcjonalna do wartości czujnika analogowego
 */

#include <Adafruit_NeoPixel.h>

const int sensorPin = A0; // Analogowy pin wejścia podłączony do zmiennego rezystora

const int ledsPin = 2;     // Pin, do którego podłączona jest taśma LED
const int numOfLeds = 16; // Liczba diod LED na taśmie

// Stałe do skalowania wartości czujnika
const int minReading = 0;
const int maxReading = 1023;
```

```
// Deklaracja taśmy LED
Adafruit_NeoPixel leds = Adafruit_NeoPixel(numOfLeds, ledsPin, NEO_GRB + NEO_KHZ800);

void setup() {
  leds.begin();    // Inicjalizacja taśmy LED
  leds.setBrightness(25);
}

void loop() {
  int sensorReading =  analogRead(A0);
  int nbrLedsToLight = map(sensorReading, minReading, maxReading, 0, numOfLeds);

  for (int i = 0; i < numOfLeds; i++)
  {
    if ( i < nbrLedsToLight)
      leds.setPixelColor(i, leds.Color(0, 0, 255)); // Niebieski
    else
      leds.setPixelColor(i, leds.Color(0, 255, 0)); // Zielony
  }
  leds.show();
}
```

Zobacz również

Receptura 3.6 wyjaśnia funkcję `max`.

W recepturze 5.6 znajdziesz więcej informacji na temat odczytywania wartości z czujnika za pomocą funkcji `analogRead`.

Receptura 5.7 opisuje funkcję `map`.

Zobacz receptury 12.2 i 12.1, jeśli potrzebujesz większej dokładności czasu wygaszania. Całkowity czas pętli to tak naprawdę więcej niż 10 milisekund, ponieważ reszta operacji z pętli także potrzebuje kilku milisekund, aby się wykonać.

7.7. Sterowanie wieloma diodami LED — efekt pływającego światła

Problem

Chcesz zapalać diody LED w taki sposób, aby powstał efekt pływającego światła. Taka animacja została wykorzystana jako efekt specjalny w serialach *Nieustraszony* i *Battlestar Galactica*, stworzonych przez Glena A. Larsona, dlatego efekt ten jest również nazywany *skanerem Larsona*.

Rozwiązanie

Możesz wykorzystać ten sam obwód co na rysunku 7.8.

```
/* Szkic chaser
 */
```

```
const int NbrLEDs = 6;
const int ledPins[] = {2, 3, 4, 5, 6, 7};
const int wait_time = 30;

// Zamień wartości dwóch poniższych definicji, jeśli katody podłączone są do masy
#define LED_ON  HIGH
#define LED_OFF LOW

void setup() {
  for (int led = 0; led < NbrLEDs; led++)
  {
    pinMode(ledPins[led], OUTPUT);
  }
}

void loop() {
  for (int led = 0; led < NbrLEDs - 1; led++)
  {
    digitalWrite(ledPins[led], LED_ON);
    delay(wait_time);
    digitalWrite(ledPins[led + 1], LED_ON);
    delay(wait_time);
    digitalWrite(ledPins[led], LED_OFF);
    delay(wait_time * 2);
  }
  for (int led = NbrLEDs - 1; led > 0; led--) {
    digitalWrite(ledPins[led], LED_ON);
    delay(wait_time);
    digitalWrite(ledPins[led - 1], LED_ON);
    delay(wait_time);
    digitalWrite(ledPins[led], LED_OFF);
    delay(wait_time * 2);
  }
}
```

Omówienie

Ten kod jest podobny do kodu z receptury 7.6, z tym że piny diod LED są włączane i wyłączane w określonej kolejności, a nie w zależności od poziomu czujnika. W szkicu są dwie pętle `for`. Pierwsza tworzy animowany wzór od lewej do prawej, zapalając po kolei diody od lewej do prawej. Pętla ta zaczyna od pierwszej diody LED (od lewej strony) i zapala sąsiednią diodę LED, i tak po kolei, aż do ostatniej (pierwszej od prawej strony). Druga pętla `for` zapala diody od prawej do lewej, zaczynając od pierwszej od prawej strony i idąc po kolei, aż do pierwszej od lewej strony. Przerwę między zapalaniem się diod określa wartość zmiennej `wait` i możesz ją dowolnie zmieniać, aby osiągnąć efekt, który będzie Ci się podobał najbardziej.

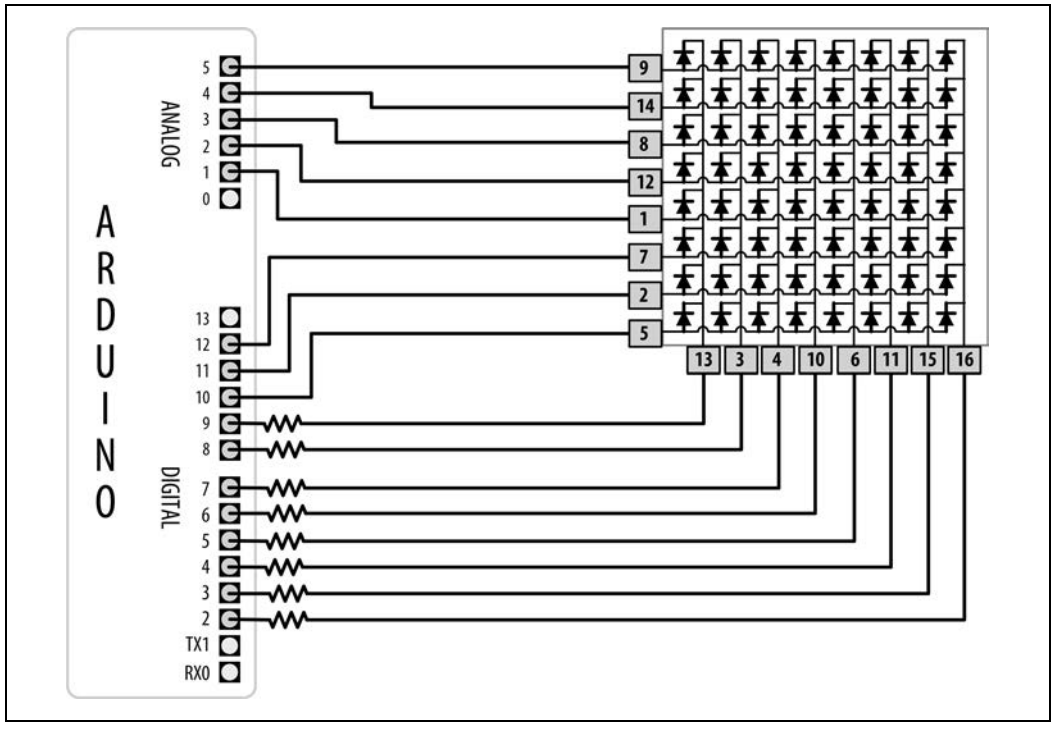
7.8. Sterowanie matrycą diod LED za pomocą multipleksowania

Problem

Masz matrycę LED i chcesz zminimalizować liczbę potrzebnych do jej sterowania pinów Arduino.

Rozwiązanie

Ten szkic wykorzystuje matrycę złożoną z 64 diod LED, gdzie anody połączone są w rzędach, a katody w kolumnach (tak jak w Jameco 2132349). Na rysunku 7.9 pokazano schemat podłączeniowy. (Dwukolorowe wyświetlacze mogą być łatwiej dostępne, a możesz sterować tylko jednym kolorem, jeśli to Ci wystarczy).



Rysunek 7.9. Matryca LED podłączona do 16 pinów cyfrowych

Jest to rozwiązanie, które pobiera stosunkowo dużo mocy i jest odpowiednie tylko dla Arduino Uno i innych płytek opartych na ATmega328. Arduino Uno WiFi Rev2 i Nano Every, podobnie jak większość (jeśli nie wszystkie) 32-bitowych płytek, nie są w stanie bezpiecznie dostarczyć wystarczająco dużo prądu, aby zasilić wszystkie diody LED. W recepturach 7.10 i 7.14 znajdziesz odpowiednie rozwiązania.

```
/*
 * Szkic matrix_mpx
 *
 * Sekwencja zapalająca po kolei wszystkie  diody LED, począwszy od pierwszej kolumny i pierwszego
wiersza, aż do ostatnich
 * Dzięki multipleksowaniu 64 diody LED są sterowane za pomocą 16 pinów
 */

const int columnPins[] = {2, 3, 4, 5, 6, 7, 8, 9};
const int rowPins[]    = {10,11,12,A1,A2,A3,A4,A5};

int pixel       = 0; // Od 0 do 63 diod LED w tablicy
int columnLevel = 0; // Wartość piksela zamieniona na kolumnę LED
int rowLevel    = 0; // Wartość piksela zamieniona na wiersz LED

void setup()
{
  for (int i = 0; i < 8; i++)
  {
    pinMode(columnPins[i], OUTPUT);   // Ustawienie wszystkich pinów diod LED jako wyjścia
    pinMode(rowPins[i], OUTPUT);
  }
}

void loop()
{
  pixel = pixel + 1;
  if(pixel > 63)
     pixel = 0;

  columnLevel = pixel / 8;   // Skalowanie do liczby kolumn
  rowLevel = pixel % 8;      // Oblicz wartość ułamkową

  for (int column = 0; column < 8; column++)
  {
    digitalWrite(columnPins[column], LOW);      // Podłączenie kolumny do masy
    for(int row = 0; row < 8; row++)
    {
      if (columnLevel > column)
      {
        digitalWrite(rowPins[row], HIGH);  // Podłączenie wszystkich diod LED w wierszu do napięcia
      }
      else if (columnLevel == column && rowLevel >= row)
      {
          digitalWrite(rowPins[row], HIGH);
      }
      else
      {
        digitalWrite(columnPins[column], LOW); // Wyłączenie wszystkich diod LED w wierszu
      }
      delayMicroseconds(300);     // Funkcja delay określa ramy czasowe (20 ms) dla 64 diod LED
      digitalWrite(rowPins[row], LOW);  // Wyłączenie diody LED
    }
```

```
    // Odłączenie kolumny od masy
    digitalWrite(columnPins[column], HIGH);
  }
}
```

Schemat podłączeniowy opiera się na wyświetlaczu Jameco (numer produktu 2132349), który ma typową budowę dla tego typu matrycy. Jednak nie wszystkie matryce mają taki sam układ pinów, dlatego musisz sprawdzić dokumentację swojego wyświetlacza. Wiersze anod i kolumny katod podłącz tak, jak pokazano na rysunku 7.16 lub rysunku 7.17, lecz użyj numerów pinów podanych w Twojej dokumentacji.

Rysunek 7.9 pokazuje logiczne ułożenie pinów oraz ich odniesienie do wierszy i kolumn. Numery pinów na schemacie odpowiadają fizycznemu układowi. Zasadniczo piny ponumerowane są według wzoru litery U, począwszy od lewego górnego rogu (1 – 8 od góry lewej kolumny pinów). Szkopuł tkwi w odpowiednim ustawieniu wyświetlacza, tak aby pin 1 był w lewym górnym rogu. Musisz poszukać oznaczenia — zazwyczaj jest to mała kropka, która oznacza pin o numerze 1. Prawdopodobnie znajdziesz ją na obudowie. Sprawdź dokumentację w razie wątpliwości

Omówienie

Należy dobrać odpowiednią wielkość opornika, aby maksymalne natężenie prądu przepływającego przez pin Arduino Uno (i innych płytek opartych na ATmega328; nie stosuj tego rozwiązania dla płytek 3,3 V ani żadnych innych, których pojedynczy pin nie może obsłużyć 40 mA) nie przekraczało 40 mA. Ponieważ prąd aż dla ośmiu diod LED może płynąć przez każdy pin kolumny, maksymalne natężenie dla każdej z nich musi wynosić jedną ósmą 40 mA, czyli 5 mA. Każda czerwona dioda w typowej matrycy ma napięcie przewodzenia równe 1,8 V. Z obliczeń wynika, że wartość opornika, który da natężenie 5 mA przy napięciu przewodzenia równym 1,8 V, będzie równa 680 Ω). Sprawdź w dokumentacji napięcie przewodzenia dla swojej matrycy. Każda kolumna jest połączona z cyfrowym pinem przez podłączony szeregowo opornik. Kiedy pin kolumny przechodzi w stan niski, a pin wiersza w wysoki, zaświeci się odpowiednia dioda LED. W przypadku każdej diody, gdy pin kolumny ma stan wysoki, a pin jej wiersza niski, nie będzie przez nią przepływał prąd i w rezultacie nie będzie ona się świecić.

Pętla `for` sprawdza każdy wiersz i każdą kolumnę i zapala po kolei diody LED, dopóki wszystkie nie będą się świecić. Pętla zaczyna od pierwszej kolumny i pierwszego wiersza, a następnie zwiększa licznik wiersza, aż do momentu, kiedy wszystkie diody z danego wiersza będą się świecić. Potem przechodzi do następnej kolumny i tak dalej, aż wszystkie diody zostaną zapalone.

Możesz sterować liczbą zapalonych diod LED proporcjonalną do wartości odczytanej z czytnika (zobacz recepturę 5.6, aby się dowiedzieć, jak podłączyć czujnik do analogowego pinu) przez wprowadzenie poniższych zmian do szkicu.

Umieść znaczniki komentarza lub usuń poniższe trzy linijki znajdujące się na początku pętli:

```
pixel = pixel + 1;
if(pixel > 63)
   pixel = 0;
```

Zastąp je następującymi linijkami, które odczytują wartość z czujnika podłączonego do pinu 0, i przeskaluj odczytaną wartość do liczby pikseli z zakresu od 0 do 63:

```
int sensorValue = analogRead(0);              // Odczyt wartości analogowej
pixel = map(sensorValue, 0, 1023, 0, 63);     // Przeskalowanie wartości czujnika do liczby pikseli (diod)
```

Zmieniony szkic możesz przetestować z rezystorem nastawnym podłączonym do analogowego pinu 0 tak, jak na rysunku 5.7 w rozdziale 5. Liczba zapalonych diod LED będzie proporcjonalna do wartości odczytanych z czujnika.

Nie musisz zapalać całego wiersza naraz. Poniższy szkic będzie zapalał diody po kolei:

```
/*
 * Szkic matrix_mpx_only_one
 *
 * Sekwencja zapalająca pojedynczo wszystkie diody LED, od pierwszej kolumny i pierwszego wiersza do ostatnich
 * Dzięki multipleksowaniu 64 diody LED są sterowane za pomocą 16 pinów
 */

const int columnPins[] = {2, 3, 4, 5, 6, 7, 8, 9};
const int rowPins[]    = {10,11,12,A1,A2,A3,A4,A5};

int pixel       = 0; // Od 0 do 63 diod LED w tablicy

void setup()
{
  for (int i = 0; i < 8; i++)
  {
    pinMode(columnPins[i], OUTPUT);   // Ustawienie wszystkich pinów diod LED jako wyjścia
    pinMode(rowPins[i], OUTPUT);
    digitalWrite(columnPins[i], HIGH);
  }
}

void loop()
{
  pixel = pixel + 1;
  if(pixel > 63)
     pixel = 0;

  int column = pixel / 8;  // Skalowanie do liczby kolumn
  int row = pixel % 8;     // Oblicz wartość ułamkową

  digitalWrite(columnPins[column], LOW);  // Podłączenie kolumny do masy
  digitalWrite(rowPins[row], HIGH);       // Ustawienie stanu wysokiego dla wiersza

  delay(125);  // Krótka pauza

  digitalWrite(rowPins[row], LOW);        // Ustawienie stanu niskiego dla wiersza
  digitalWrite(columnPins[column], HIGH); // Odłączenie kolumny od masy
}
```

7.9. Wyświetlanie obrazków na matrycy LED

Problem

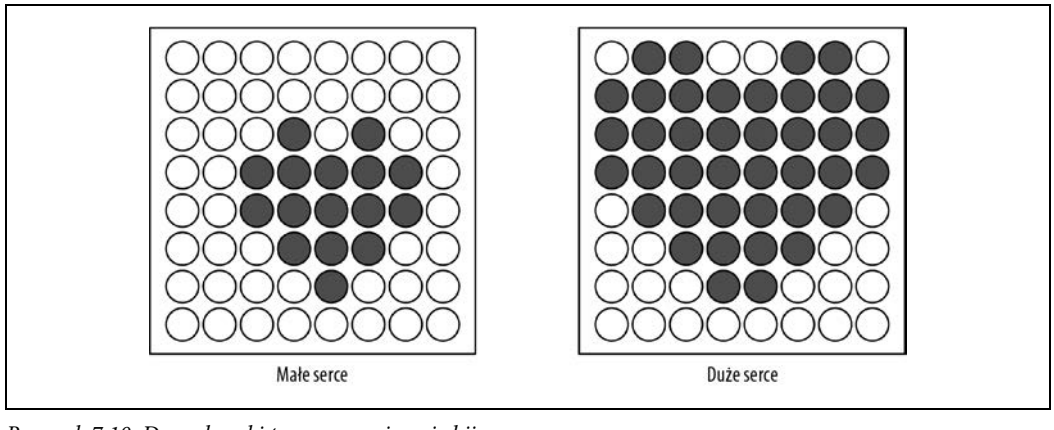
Chcesz wyświetlić jeden lub więcej obrazków na matrycy LED. Być może chcesz szybko zmieniać kilka obrazków, aby powstała animacja.

Rozwiązanie

Obwód potrzebny w tej recepturze może być taki sam jak w recepturze 7.8. Szkic tworzy animację bijącego serca przez zaświecanie na krótko diod LED ułożonych w kształt serca. Małe serce, po którym wyświetla się większe, reprezentuje jedno uderzenie (na rysunku 7.10 pokazano obrazek serca ułożony z diod LED).

```
/*
 * Szkic matrix_animate
 * Animacja dwóch obrazków serca ilustrująca bicie serca
 */

// Obrazki serc są przechowywane jako bitmapy — każdy bit odpowiada jednej diodzie LED
// 0 oznacza zgaszoną diodę, a 1 zapaloną
byte bigHeart[] = {
  B01100110,
  B11111111,
  B11111111,
  B11111111,
  B01111110,
  B00111100,
  B00011000,
  B00000000};

byte smallHeart[] = {
  B00000000,
  B00000000,
  B00010100,
  B00111110,
  B00111110,
  B00011100,
  B00001000,
  B00000000};

const int columnPins[] = { 2, 3, 4, 5, 6, 7, 8, 9};
const int rowPins[]    = {10,11,12,A1,A2,A3,A4,A5};

void setup() {
  for (int i = 0; i < 8; i++)
  {
    pinMode(rowPins[i], OUTPUT);          // Ustawienie wszystkich pinów diod LED jako wyjścia
    pinMode(columnPins[i], OUTPUT);
    digitalWrite(columnPins[i], HIGH);  // Odłączenie pinu kolumny od masy
  }
}
```

```
void loop() {
  int pulseDelay = 800 ; // Odstęp (w milisekundach) między uderzeniami serca

  show(smallHeart, 80);  // Wyświetlenie małego serca przez 80 ms
  show(bigHeart, 160);   // Wyświetlenie dużego serca przez 160 ms
  delay(pulseDelay);     // Nie pokazuj nic między uderzeniami
}

// Wyświetlenie ramki obrazka przechowywanej w tablicy, której wskazaniem jest parametr obrazka
// Ramka jest powtarzana przez czas określony za pomocą drugiego parametru podanego w milisekundach
void show(byte * image, unsigned long duration)
{
 unsigned long start = millis();              // Początek odliczania czasu animacji
 while (start + duration > millis())          // Wykonuj tak długo, aż minie określony czas animacji
  {
    for(int row = 0; row < 8; row++)
    {
      digitalWrite(rowPins[row], HIGH);           // Podłączenie wiersza do napięcia
      for(int column = 0; column < 8; column++)
      {
        bool pixel = bitRead(image[row],column);
        if(pixel == 1)
        {
          digitalWrite(columnPins[column], LOW);  // Podłączenie kolumny do masy
        }
        delayMicroseconds(300);                   // Małe opóźnienie dla każdej diody LED
        digitalWrite(columnPins[column], HIGH);   // Odłączenie kolumny od masy
      }
      digitalWrite(rowPins[row], LOW);            // Odłączenie diod LED
    }
  }
}
```



Rysunek 7.10. Dwa obrazki tworzące animację bijącego serca

Omówienie

Kolumny i wiersze są multipleksowane (przełączane) podobnie jak w recepturze 7.8, z tą różnicą, że wartości wysyłane do diody LED opierają się na obrazkach przechowywanych w tablicach `bigHeart`

i `smallHeart`. Każdy element tych tablic reprezentuje **piksel** (pojedynczą diodę LED), a każdy wiersz tablic odpowiada wierszowi wyświetlacza. Wiersz składa się z 8 bitów przedstawionych w formacie binarnym (na co wskazuje litera *B* umieszczona na początku każdego wiersza). Bit o wartości 1 oznacza, że odpowiadająca mu dioda LED powinna się świecić, a 0, że powinna pozostać zgaszona. Efekt animacji powstaje na skutek szybkiego przełączania się pomiędzy tablicami.

Funkcja `loop` czeka krótką chwilę (800 ms) pomiędzy uderzeniami serca, a następnie wywołuje funkcję `show`, najpierw z tablicą `smallHeart`, a potem z `bigHeart`. Funkcja `show` przechodzi przez każdy element w wierszu i kolumnie, zapalając odpowiednią diodę LED, jeśli dany element ma wartość 1. Funkcja `bitRead` (zobacz recepturę 2.20) jest stosowana do określenia wartości każdego bitu.

Krótka, 300-mikrosekundowa przerwa między pikselami, daje człowiekowi wystarczająco dużo czasu, aby zauważył obrazek. Co więcej, dzięki temu każdy obrazek powtarza się wystarczająco często (50 razy na sekundę) i miganie nie jest zauważalne.

Poniżej znajduje się kod ze zmienioną częstotliwością uderzeń serca, opartą na wartości odczytanej z czujnika. Możesz przetestować ten szkic za pomocą rezystora nastawnego podłączonego do analogowego pinu 0 tak, jak pokazano w recepturze 5.6. Użyj poprzedniego obwodu i kodu, w którym wystarczy tylko zamienić funkcję `loop` na następującą:

```
void loop() {
  int sensorValue = analogRead(A0);                  // Odczyt wejściowej wartości analogowej
  int pulseRate =  map(sensorValue,0,1023,40,240); // Zamień na bity na minutę
  int pulseDelay = (60000 / pulseRate);  // Odstęp pomiędzy uderzeniami serca (w milisekundach)

  show(smallHeart, 80);              // Wyświetlenie małego serca przez 80 ms
  show(bigHeart, 160);               // Wyświetlenie dużego serca przez 160 ms
  delay(pulseDelay);                 // Nie pokazuj nic między uderzeniami
}
```

Ta wersja oblicza przerwę między uderzeniami serca za pomocą funkcji `map` (zobacz recepturę 5.7), aby zamienić wartość odczytaną z czujnika na liczbę uderzeń na minutę. Obliczenie nie bierze pod uwagą czasu potrzebnego na wyświetlenie serca, ale możesz odjąć 240 ms (80 ms i 160 ms dla obu obrazków), jeśli chcesz uzyskać większą dokładność.

Zobacz również

W recepturach 7.13 i 7.14 znajdziesz informacje, jak używać rejestrów przesuwnych do sterowania wieloma diodami matrycy LED i jednocześnie zmniejszyć liczbę potrzebnych do tego pinów Arduino.

Receptury 12.2 i 12.1 zawierają więcej informacji, jak zarządzać czasem za pomocą funkcji `millis`.

7.10. Sterowanie matrycą LED — charlieplexing

Problem

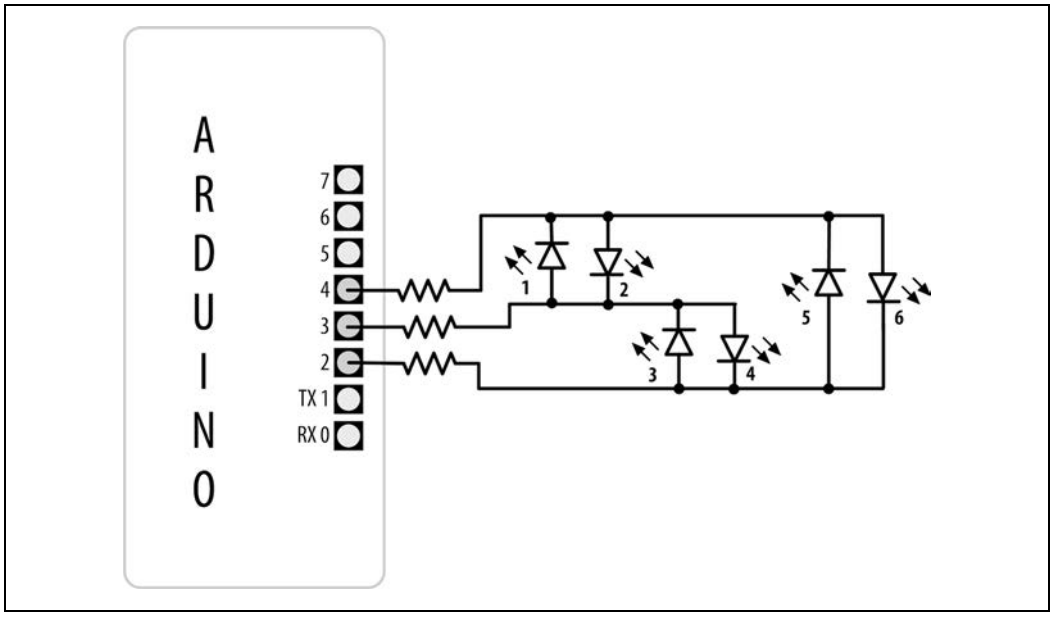
Masz matrycę LED i chcesz zminimalizować liczbę pinów potrzebnych do włączania i wyłączania którejkolwiek z diod.

Rozwiązanie

Charlieplexing to szczególny rodzaj multipleksowania zwiększający liczbę diod LED, które mogą być sterowane za pomocą grupy pinów. Ten szkic steruje sześcioma diodami LED za pomocą tylko trzech pinów. Na rysunku 7.11 pokazano schemat podłączeniowy (aby obliczyć poprawną wartość oporników dla diod LED, zobacz recepturę 7.1).

```
/*
 * Szkic charlieplex
 * Zapalenie po kolei sześciu diod LED podłączonych do pinów 2, 3 i 4
 */

int pins[] = {2,3,4};  // Piny, do których podłączone są diody LED

// Następne dwie linie obliczają liczbę pinów i diod LED na podstawie powyższych tablic
const int NUMBER_OF_PINS = sizeof(pins)/ sizeof(pins[0]);
const int NUMBER_OF_LEDS = NUMBER_OF_PINS * (NUMBER_OF_PINS-1);

byte pairs[NUMBER_OF_LEDS/2][2] = { {2,1}, {1,0}, {2,0} }; // Mapowanie pinów do diod LED

void setup()
{
   // Żadne ustawienia nie są wymagane
}

void loop(){
    for(int i=0; i < NUMBER_OF_LEDS; i++)
    {
       lightLed(i);  // Zapalanie po kolei wszystkich diod LED
       delay(1000);
    }
}

// Funkcja zapalająca podaną diodę LED, pierwsza ma numer 0
void lightLed(int led)
{
  // Poniższe cztery linijki zamieniają numer diody LED na numer pinu
  int indexA = pairs[led/2][0];
  int indexB = pairs[led/2][1];
  int pinA = pins[indexA];
  int pinB = pins[indexB];

  // Wyłączenie wszystkich pinów niepodłączonych do podanej diody LED
  for(int i=0; i < NUMBER_OF_PINS; i++)
  {
    if(pins[i] != pinA && pins[i] != pinB)
    {  // Jeśli ten pin nie jest żadnym z naszych pinów,
       pinMode(pins[i], INPUT);    // ustaw tryb na wejście,
       digitalWrite(pins[i],LOW); // upewnij się, że rezystory podciągające są wyłączone
    }
  }
  // Teraz włącz piny dla podanej diody LED
  pinMode(pinA, OUTPUT);
  pinMode(pinB, OUTPUT);
  if( led % 2 == 0)
```

```
    {
       digitalWrite(pinA,LOW);
       digitalWrite(pinB,HIGH);
    }
    else
    {
       digitalWrite(pinB,LOW);
       digitalWrite(pinA,HIGH);
    }
}
```



Rysunek 7.11. Sześć diod LED sterowanych trzema pinami za pomocą techniki charlieplexing

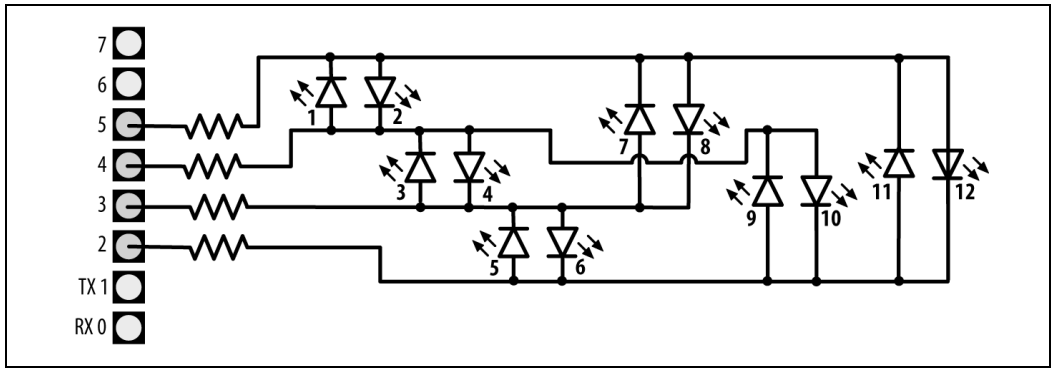
Omówienie

Termin *charlieplexing* pochodzi od imienia Charlie'ego Allena (z firmy Microchip Technology), który opublikował tę metodę. Technika opiera się na fakcie, że dioda LED zapali się tylko wtedy, gdy zostanie podłączona w „dobrą stronę" (z odpowiednio większym napięciem na anodzie niż na katodzie). Oto tabela pokazująca numery diod LED (zobacz rysunek 7.9), które są zapalone, kiedy występuje odpowiednia kombinacja stanów trzech pinów. `L` oznacza `LOW` (niski), `H` oznacza `HIGH` (wysoki), a `i` odnosi się do trybu `INPUT` (wejście). Ustawienie na pinie trybu `INPUT` skutecznie odłącza go od obwodu.

```
Piny     Diody LED
4 3 2    1 2 3 4 5 6
L L L    0 0 0 0 0 0
L H i    1 0 0 0 0 0
H L i    0 1 0 0 0 0
i L H    0 0 1 0 0 0
i H L    0 0 0 1 0 0
```

```
L i H    0 0 0 0 1 0
H i L    0 0 0 0 0 1
```

Możesz podwoić liczbę diod LED poprzez dodanie tylko jednego pinu. Pierwsze sześć diod jest podłączone tak samo jak w poprzednim przykładzie. Dodaj jeszcze sześć, aby obwód wyglądał jak na rysunku 7.12.



Rysunek 7.12. Sterowanie 12 diodami LED za pomocą czterech pinów z wykorzystaniem techniki charlieplexing

W poprzednim szkicu dodaj pin do tablicy `pins`:

```
byte pins[] = {2,3,4,5}; // Piny, do których podłączone są diody LED
```

Dodaj elementy do tablicy `pairs`:

```
byte pairs[NUMBER_OF_LEDS/2][2] = { {0,1}, {1,2}, {0,2}, {2,3}, {1,3}, {0,3} };
```

Pozostała część kodu może pozostać bez zmian. Pętla będzie przechodzić przez wszystkie 12 diod, ponieważ kod określa ich liczbę na podstawie rozmiaru tablicy `pins`.

Z uwagi na fakt, że charlieplexing działa na zasadzie sterowania pinami Arduino, aby tylko jedna dioda LED w danym momencie była zapalona, o wiele bardziej skomplikowane jest stworzenie wrażenia zaświecania wielu diod. Jednak możesz zapalić wiele diod za pomocą multipleksowania dostosowanego do techniki charlieplexing.

Poniższy szkic tworzy wykres słupkowy przez zapalenie grupy diod LED w oparciu o wartość odczytaną z czujnika podłączonego do analogowego pinu 0:

```
byte pins[] = {2,3,4};
const int NUMBER_OF_PINS = sizeof(pins)/ sizeof(pins[0]);
const int NUMBER_OF_LEDS = NUMBER_OF_PINS * (NUMBER_OF_PINS-1);

byte pairs[NUMBER_OF_LEDS/2][2] = { {2,1}, {1,0}, {2,0} }; // Mapowanie pinów do diod LED

int ledStates = 0; // Zmienna przechowująca stan maksymalnie 15 diod LED
int refreshedLed;  // Odświeżana dioda LED

void setup()
{
   // Żadne ustawienia nie są wymagane
```

```
}

void loop()
{
const int analogInPin = 0; // Analogowy pin wejścia podłączony do opornika nastawnego

  // Kod ze szkicu bargraph
  int sensorValue = analogRead(analogInPin);            // Odczyt wejściowej wartości analogowej
  // Skalowanie odczytu do liczby diod LED
  int ledLevel = map(sensorValue, 0, 1023, 0, NUMBER_OF_LEDS);
  for (int led = 0; led < NUMBER_OF_LEDS; led++)
  {
    if (led < ledLevel ) {
      setState(led, HIGH);      // Włączenie pinów poniżej poziomu
    }
    else {
      setState(led, LOW);       // Wyłączenie pinów powyżej poziomu
    }
  }
  ledRefresh();

}

void setState( int led, bool state)
{
   bitWrite(ledStates,led, state);
}

void ledRefresh()
{
   // Przy każdym wywołaniu funkcji odśwież inną diodę LED
   if( refreshedLed++ > NUMBER_OF_LEDS) // Przejście do następnej diody LED
      refreshedLed = 0; // Jeśli wszystkie zostały odświeżone, zacznij od początku

   if( bitRead(ledStates, refreshedLed ) == HIGH)
         lightLed( refreshedLed );
   else
       if(refreshedLed == 0) // Wyłącz wszystkie diody, jeśli pin 0 jest wyłączony
         for(int i=0; i < NUMBER_OF_PINS; i++)
           digitalWrite(pins[i],LOW);
}

// Ta funkcja jest dokładnie taka sama jak ta w szkicu pokazanym w „Rozwiązaniu”
// Funkcja zapalająca podaną diodę LED, pierwsza ma numer 0
void lightLed(int led)
{
  // Poniższe cztery linijki zamieniają numer diody LED na numer pinu
  int indexA = pairs[led/2][0];
  int indexB = pairs[led/2][1];
  int pinA = pins[indexA];
  int pinB = pins[indexB];

  // Wyłączenie wszystkich pinów niepodłączonych do podanej diody LED
  for(int i=0; i < NUMBER_OF_PINS; i++)
  {
```

```
    if(pins[i] != pinA && pins[i] != pinB)
    {  // Jeśli ten pin nie jest żadnym z naszych pinów,
        pinMode(pins[i], INPUT);    // ustaw tryb na wejście,
        digitalWrite(pins[i],LOW); // upewnij się, że rezystory podciągające są wyłączone
    }
  }
  // Teraz włącz piny dla podanej diody LED
  pinMode(pinA, OUTPUT);
  pinMode(pinB, OUTPUT);
  if( led % 2 == 0)
  {
     digitalWrite(pinA,LOW);
     digitalWrite(pinB,HIGH);
  }
  else
  {
     digitalWrite(pinB,LOW);
     digitalWrite(pinA,HIGH);
  }
}
```

Szkic używa wartości bitów zapisanych w zmiennej `ledStates`, które określają stan diody (0 to wyłączona, a 1 to włączona). Funkcja `refresh` sprawdza każdy bit i zapala diodę odpowiadającą danemu bitowi, który ma wartość 1. Funkcja `refresh` musi być wywołana szybko i powtarzalnie, inaczej będzie widać, że diody migają.

Dodanie opóźnień do kodu może zakłócać iluzję optyczną, dzięki której nie widzimy, że diody LED migają.

Do obsługi funkcji `refresh` w tle możesz zastosować przerwanie (bez konieczności wywoływania jej bezpośrednio w funkcji `loop`). Przerwania od licznika zostały omówione szerzej w rozdziale 18., ale to jest przykład jednego z ich zastosowań, które obsługuje odświeżanie matrycy. Do tworzenia przerwań poniższy szkic korzysta z zewnętrznej biblioteki FrequencyTimer2, która jest dostępna w menedżerze bibliotek (instrukcje dotyczące instalowania bibliotek zewnętrznych znajdziesz w recepturze 16.2).

```
#include <FrequencyTimer2.h>  // Biblioteka potrzebna do obsługi odświeżania wyświetlacza

byte pins[] = {2,3,4};
const int NUMBER_OF_PINS = sizeof(pins)/ sizeof(pins[0]);
const int NUMBER_OF_LEDS = NUMBER_OF_PINS * (NUMBER_OF_PINS-1);

byte pairs[NUMBER_OF_LEDS/2][2] = { {2,1}, {1,0}, {2,0} };

int ledStates = 0; // Zmienna przechowująca stan maksymalnie 15 diod LED
int refreshedLed;  // Odświeżana dioda LED

void setup()
{
  FrequencyTimer2::setPeriod(20000/NUMBER_OF_LEDS); // Ustawienie okresu
  // Następna linijka informuje FrequencyTimer2, jaka funkcja ma być wywołana (ledRefresh)
  FrequencyTimer2::setOnOverflow(ledRefresh);
```

```
  FrequencyTimer2::enable();
}

void loop()
{
const int analogInPin = 0; // Analogowy pin wejścia podłączony do opornika nastawnego

  // Kod ze szkicu bargraph
  int sensorValue = analogRead(analogInPin);            // Odczyt wejściowej wartości analogowej
  // Skalowanie odczytu do liczby diod LED
  int ledLevel = map(sensorValue, 0, 1023, 0, NUMBER_OF_LEDS);
  for (int led = 0; led < NUMBER_OF_LEDS; led++)
  {
    if (led < ledLevel ) {
      setState(led, HIGH);      // Włączenie pinów poniżej poziomu
    }
    else {
      setState(led, LOW);       // Wyłączenie pinów powyżej poziomu
    }
  }
   // Dioda LED nie jest już odświeżana w funkcji loop; jest to obsługiwane przez FrequencyTimer2
}
// Reszta kodu jest taka sama jak w poprzednim przykładzie
```

Biblioteka FrequencyTimer2 ma ustawiony okres na 1,666 mikrosekundy (20 µs podzielone na 12, czyli liczbę diod LED). Metoda `FrequencyTimer2setOnOverflow` wywołuje funkcję `ledRefresh` za każdym razem, gdy licznik da taką informację. Biblioteka FrequencyTimer2 jest kompatybilna z ograniczoną liczbą płytek: Arduino Uno (i prawdopodobnie większość płytek opartych na ATmega328), Arduino Mega oraz wiele wersji płytek Teensy. Na stronie PJRC (*https://www.pjrc.com/teensy/td_libs_FrequencyTimer2.html*) znajdziesz więcej szczegółów (w języku angielskim) na temat tej biblioteki.

Zobacz również

W rozdziale 18. znajdziesz więcej informacji na temat przerwań od licznika.

7.11. Sterowanie 7-segmentowym wyświetlaczem LED

Problem

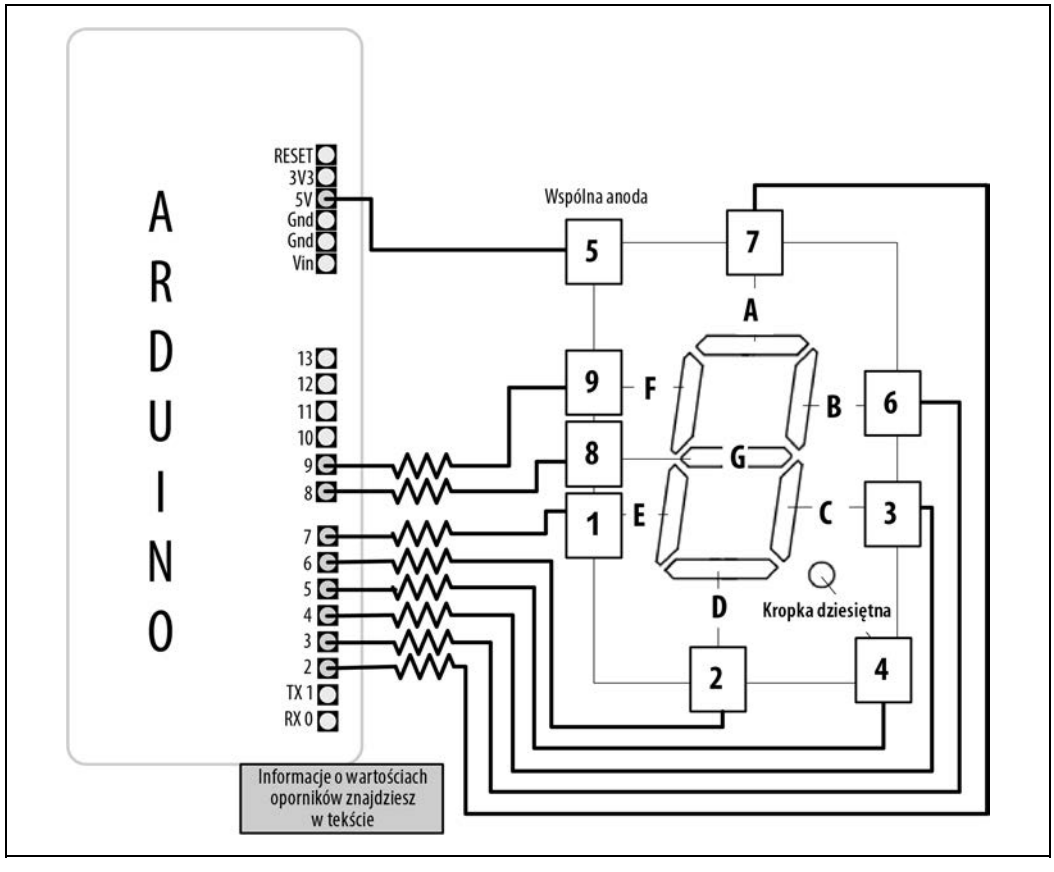
Chcesz wyświetlać liczby za pomocą numerycznego wyświetlacza 7-segmentowego.

Rozwiązanie

Poniższy szkic wyświetla liczby od 0 do 9 na jednej cyfrze złożonej z 7 segmentów. Rysunek 7.13 pokazuje schemat podłączeniowy dla wyświetlacza ze wspólną anodą. Przydział pinów dla Twojego wyświetlacza może być inny, dlatego sprawdź dokumentację. Jeśli wyświetlacz ma wspólną katodę, podłącz ją do masy. Efekt końcowy powstaje przez włączanie kombinacji segmentów, które reprezentują cyfry.

```
/*
 * Szkic SevenSegment
 * Pokazuje liczby od 0 do 9 na jednej cyfrze wyświetlacza
 * Ten szkic odlicza sekundy od 0 do 9
 */

// Bity reprezentujące segmenty od A do G (i kropkę dziesiętną) dla liczb 0 – 9
const byte numeral[10] = {
  //ABCDEFG + kropka dziesiętna
  B11111100,  // 0
  B01100000,  // 1
  B11011010,  // 2
  B11110010,  // 3
  B01100110,  // 4
  B10110110,  // 5
  B00111110,  // 6
  B11100000,  // 7
  B11111110,  // 8
  B11100110,  // 9
};

// Piny dla kropki dziesiętnej i każdego segmentu
// Kropka dziesiętna, G,F,E,D,C,B,A
const int segmentPins[8] = { 5,8,9,7,6,4,3,2};

void setup()
{
  for(int i=0; i < 8; i++)
  {
    pinMode(segmentPins[i], OUTPUT); // Ustawienie pinów kropki dziesiętnej i segmentów jako wyjścia
  }
}

void loop()
{
  for(int i=0; i <= 10; i++)
  {
    showDigit(i);
    delay(1000);
  }
  // Ostatnią wartością jest 10, która spowoduje wyłączenie wyświetlacza
  delay(2000);   // Wyświetlacz będzie wyłączony przez 2 sekundy
}

// Wyświetlanie cyfr od 0 do 9 na wyświetlaczu 7-segmentowym
// Każda wartość spoza zakresu 0 – 9 wyłącza wyświetlacz
void showDigit(int number)
{
  bool isBitSet;

  for(int segment = 1; segment < 8; segment++)
  {
    if( number < 0 || number > 9){
      isBitSet = 0;    // Wyłączenie wszystkich segmentów
    }
```

```
    else{
      // isBitSet będzie równe true, jeśli dany bit jest równy 1
      isBitSet = bitRead(numeral[number], segment);
    }
    isBitSet = ! isBitSet; // Usuń tę linijkę jeśli wyświetlacz ma wspólną katodę
    digitalWrite(segmentPins[segment], isBitSet);
  }
}
```



Rysunek 7.13. Podłączenie wyświetlacza 7-segmentowego

Omówienie

Segmenty, który powinny być zapalane dla każdej liczby, są przechowywane w tablicy o nazwie `numeral`. Znajduje się w niej po 1 bajcie na każdą cyfrę, gdzie każdy bit odpowiada jednemu z siedmiu segmentów (lub kropce dziesiętnej).

W tablicy o nazwie `segmentPins` zapisane są numery pinów każdego segmentu. Funkcja `showDigit` sprawdza, czy podana liczba mieści się w zakresie od 0 do 9, a jeśli tak, to sprawdza każdy bit i zapala

odpowiedni segment dla bitów równych 1. W recepturze 3.12 znajdziesz więcej informacji na temat funkcji `bitRead`.

Jak już wspomnieliśmy w recepturze 7.4, segment zapala się, gdy pin ma wartość `HIGH` w przypadku wyświetlacza ze wspólną katodą, a `LOW` dla wyświetlacza ze wspólną anodą. Poniższy kod jest przeznaczony dla wariantu ze wspólną anodą, więc w pokazany tu sposób zamienia wartości (0 na 1 i 1 na 0):

```
isBitSet = ! isBitSet; // Usuń tę linijkę, jeśli wyświetlacz ma wspólną katodę
```

Znak *!* jest operatorem negacji (zobacz recepturę 2.20). Jeżeli Twój wyświetlacz ma wspólną katodę (wszystkie katody są ze sobą połączone; sprawdź dokumentację, jeśli nie masz pewności), usuń tę linijkę.

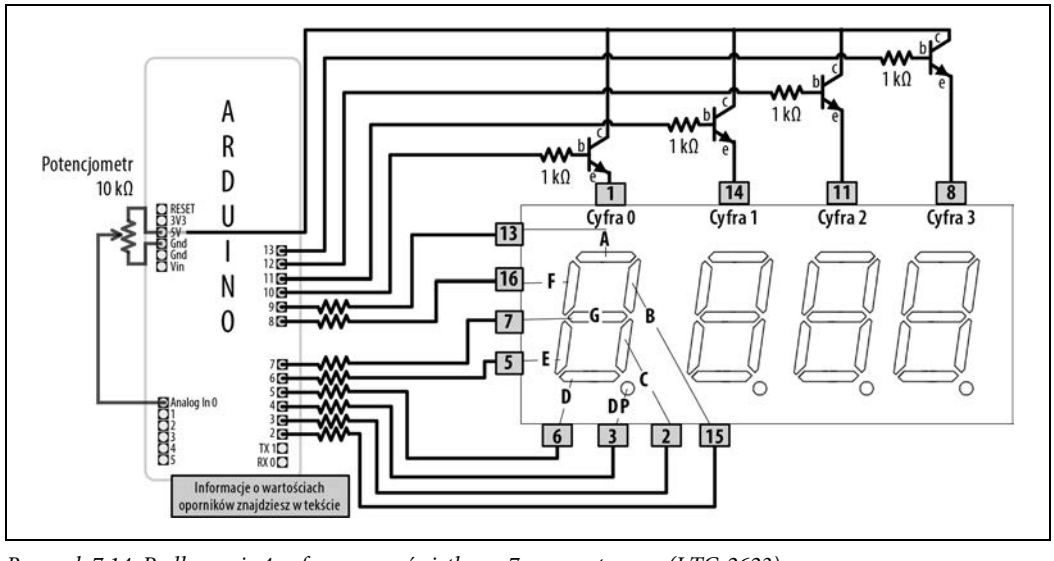
7.12. Sterowanie kilkucyfrowymi wyświetlaczami 7-segmentowymi: multipleksowanie

Problem

Na wyświetlaczu 7-segmentowym chcesz pokazywać liczby złożone z co najmniej dwóch cyfr.

Rozwiązanie

Wielocyfrowe wyświetlacze 7-segmentowe zazwyczaj wykorzystują multipleksowanie. W poprzednich recepturach zmultipleksowane wiersze i kolumny diod LED były ze sobą połączone, tworząc tablicę. Tutaj odpowiadające sobie segmenty każdej cyfry są ze sobą połączone (rysunek 7.14 pokazuje połączenia dla wyświetlacza Lite-On LTC-2623, a jeśli używasz innego, sprawdź dokumentację).



Rysunek 7.14. Podłączenie 4-cyfrowego wyświetlacza 7-segmentowego (LTC-2623)

Pokazany schemat podłączeniowy odnosi się do wyświetlacza Lite-On LTC-2623. Jeśli korzystasz z innego wyświetlacza, możesz użyć tych samych pinów Arduino, ale musisz sprawdzić w dokumentacji, które piny wyświetlacza do nich podłączyć. Wyświetlacz LTC-2623 ma wspólną anodę. Jeżeli Twój ma wspólną katodę, musisz zmienić dwie rzeczy. Po pierwsze, podłącz tranzystory inaczej, tak, aby wszystkie emitery były połączone ze sobą i z masą, a każdy kolektor był podłączony do odpowiedniego pinu wyświetlacza. Po drugie, zamień na komentarz lub usuń w szkicu linijkę `isBitSet = ! isBitSet;`.

```
/*
 * Szkic sevenseg_mpx
 * Pokazuje liczby z zakresu od 0 do 9999 na 4-cyfrowym wyświetlaczu
 * Ten przykład wyświetla wartości odczytane z czujnika podłączonego do analogowego pinu wejścia
 */

// Bity reprezentujące segmenty od A do G (i kropkę dziesiętną) dla cyfr 0 – 9
const int numeral[10] = {
  // ABCDEFG + kropka dziesiętna
  B11111100,  // 0
  B01100000,  // 1
  B11011010,  // 2
  B11110010,  // 3
  B01100110,  // 4
  B10110110,  // 5
  B00111110,  // 6
  B11100000,  // 7
  B11111110,  // 8
  B11100110,  // 9
};

// Piny dla kropki dziesiętnej i każdego segmentu
// Kropka dziesiętna, G,F,E,D,C,B,A
const int segmentPins[] = { 4, 7,8,6,5,3,2,9};

const int nbrDigits= 4;  // Liczba cyfr wyświetlacza
                         // Cyfry 0 1 2 3
const int digitPins[nbrDigits] = { 10,11,12,13};

void setup()
{
  for(int i=0; i < 8; i++)
    pinMode(segmentPins[i], OUTPUT); // Ustawienie pinów kropki dziesiętnej i segmentów jako wyjścia

  for(int i=0; i < nbrDigits; i++)
    pinMode(digitPins[i], OUTPUT);
}

void loop()
{
  int value = analogRead(0);
  showNumber(value);
}

void showNumber(int number)
```

```
{
  if(number == 0)
    showDigit( 0, nbrDigits-1) ; // Wyświetlenie 0 na pierwszej cyfrze od prawej strony
  else
  {
    // Wyświetlenie wartości odpowiadającej każdej cyfrze
    // Pierwsza cyfra z lewej strony ma numer 0, pierwsza z prawej jest o 1 mniejsza niż liczba miejsc
    for( int digit = nbrDigits-1; digit >= 0; digit--)
    {
      if(number > 0)
      {
         showDigit( number % 10, digit)  ;
         number = number / 10;
      }
    }
  }
}

// Funkcja pokazująca podaną liczbę na wyświetlaczu 7-segmentowym na podanym miejscu
void showDigit( int number, int digit)
{
  digitalWrite( digitPins[digit], HIGH );
  for(int segment = 1; segment < 8; segment++)
  {
    bool isBitSet = bitRead(numeral[number], segment);
    // isBitSet będzie równe true, jeśli dany bit jest równy 1
    isBitSet = ! isBitSet; // Usuń tę linijkę, jeśli wyświetlacz ma wspólną katodę
    digitalWrite( segmentPins[segment], isBitSet);
  }
  delay(5);
  digitalWrite( digitPins[digit], LOW );
}
```

Omówienie

Powyższy szkic zawiera funkcję `showDigit`, która jest podobna do tej z receptury 7.11. Tutaj parametrami funkcji są liczba i pozycja. Logika zapalająca odpowiednie segmenty jest taka sama, lecz dodatkowo kod włącza pin (wartość `HIGH`) opowiadający pozycji cyfry i w efekcie tylko ta cyfra będzie wyświetlona (zobacz wcześniejsze wyjaśnienia techniki multipleksowania).

7.13. Sterowanie kilkucyfrowymi wyświetlaczami 7-segmentowymi za pomocą mniejszej liczby pinów

Problem

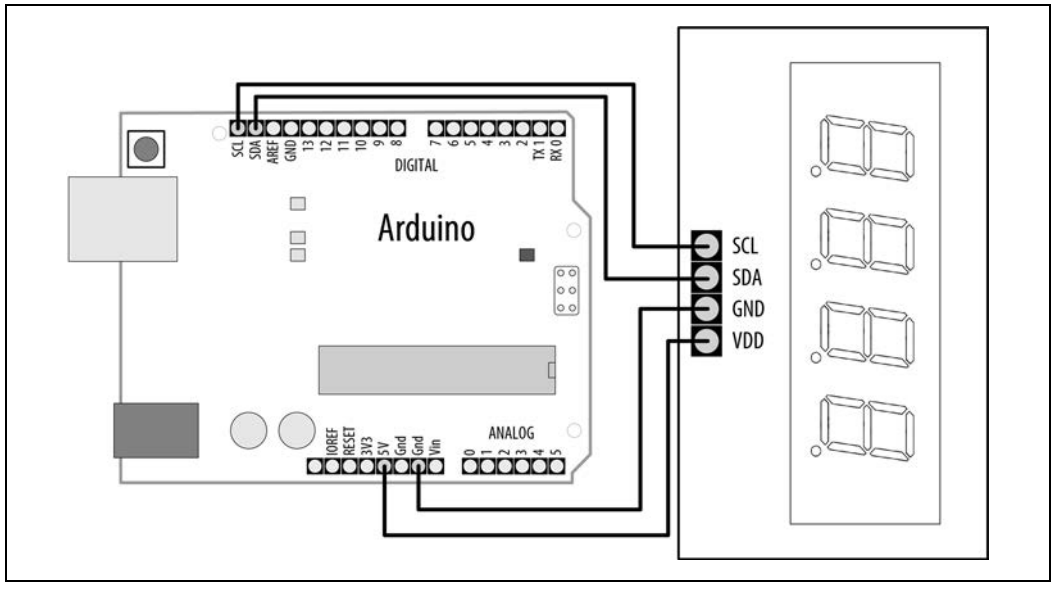
Chcesz sterować kilkucyfrowym wyświetlaczem 7-segmentowym, ale przy zminimalizowanej liczbie wymaganych pinów Arduino.

Rozwiązanie

Ta receptura wykorzystuje płytkę drukowaną opartą na HT16K33 służącą do sterowania 4-cyfrowymi wyświetlaczami ze wspólną katodą, na przykład LuckyLight KW4-56NXBA-P lub Betlux BLQ56C-43. HT16K33 zapewnia prostsze rozwiązanie niż to z receptury 7.12, gdyż obsługuje multipleksowanie i odkodowywanie liczb na poziomie sprzętu. Płytki drukowane oparte na HT16K33 możesz kupić w wielu sklepach. Firma Adafruit produkuje moduł złożony z czterech 7-segmentowych cyfr (numer produktu 877), który jest dostępny w różnych kolorach (*https://www.adafruit.com/category/103*).

Oto szkic, który wyświetli liczbę z zakresu do 0 do 9999 (na rysunku 7.15 został pokazany schemat podłączeniowy):

```
/*
 Szkic HT16K33
 */

#include <Wire.h>
#include <Adafruit_GFX.h>
#include "Adafruit_LEDBackpack.h"

Adafruit_7segment matrix = Adafruit_7segment();

const int numberOfDigits = 4; // Zmień tę wartość, aby pasowała do liczby podłączonych cyfr
const int maxCount       = 9999;

int number = 0;

void setup()
{
  Serial.begin(9600);
  matrix.begin(0x70);      // Inicjalizacja wyświetlacza
  matrix.println(number); // Wysłanie liczy do wyświetlacza (na początku 0)
  matrix.writeDisplay();  // Aktualizacja wyświetlacza
}

void loop()
{
  // Wyświetlenie liczby otrzymanej z portu szeregowego zakończonej znakiem końca linii
  if (Serial.available())
  {
    char ch  = Serial.read();
    if ( ch == '\n')
    {
      if (number <= maxCount)
      {
        matrix.println(number);
        matrix.writeDisplay();
        number = 0; // Wyzerowanie liczby
      }
    }
    else
      number = (number * 10) + ch - '0'; // Szczegóły znajdziesz w rozdziale 4.
  }
}
```



Rysunek 7.15. Sterownik HT16K33 dla kilkucyfrowego wyświetlacza 7-segmentowego ze wspólną katodą

Omówienie

Receptura wykorzystuje protokół I2C do komunikacji z chipem HT16K33. Biblioteka Adafruit_LED ↪Backpack zapewnia interfejs za pośrednictwem obiektu `Adafruit_7segment` (w tym szkicu nazywany `matrix`). W rozdziale 13. omówiliśmy bardziej szczegółowo protokół I2C.

Powyższy szkic wyświetla otrzymaną z portu szeregowego liczbę pod warunkiem, że ma ona maksymalnie cztery cyfry. W rozdziale 4. wyjaśniliśmy zawarty w funkcji `loop` kodu obsługujący port szeregowy. Funkcja `matrix.println` wysyła wartości do sterownika HT16K33, a funkcja `matrix.writeDisplay` aktualizuje wyświetlacz, pokazując ostatnią otrzymaną liczbę.

Sterownik wykorzystuje 4-cyfrowy wyświetlacz 7-segmentowy, ale jeśli kupisz bardziej ogólną płytkę drukowaną HT16K33 (na przykład produkt numer 1427 firmy Adafruit), możesz jej używać z wyświetlaczami od jedno- do ośmiocyfrowych. Jeśli używasz kilku wyświetlaczy, musisz podłączyć wszystkie piny odpowiadających sobie segmentów razem. (W recepturze 13.1 pokazaliśmy, jak podłączyć popularny 2-cyfrowy wyświetlacz). Sprawdź dokumentację swojego wyświetlacza i sterownika HT16K33.

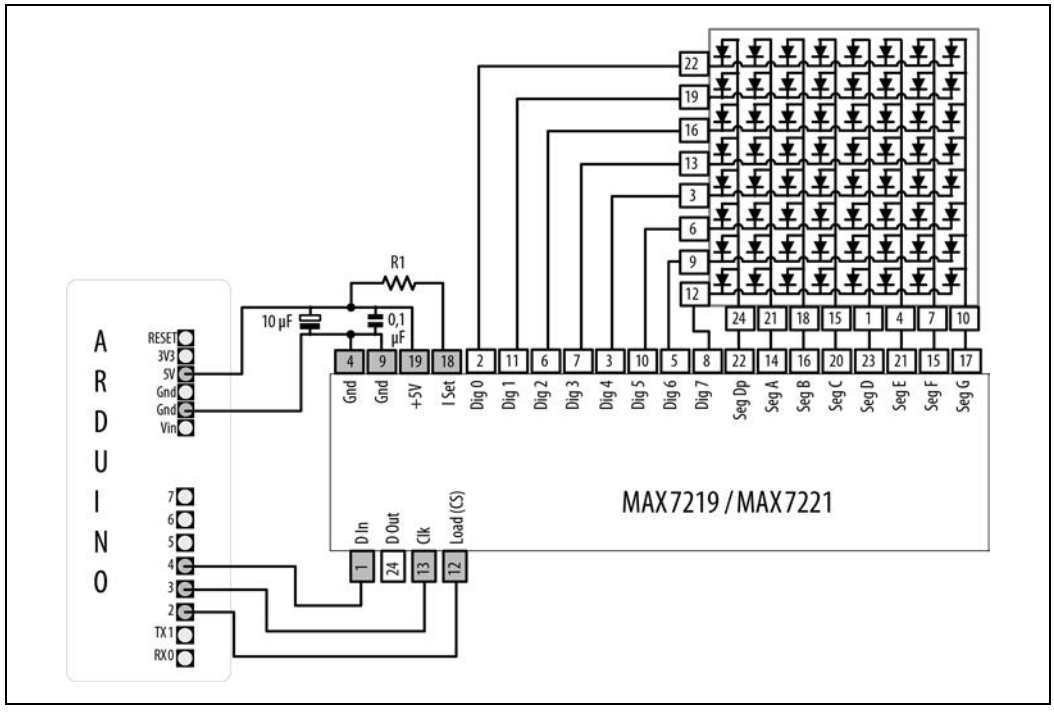
7.14. Sterowanie matrycą diod LED za pomocą rejestrów przesuwnych MAX72xx

Problem

Masz matrycę diod LED 8×8 i chcesz zminimalizować liczbę pinów potrzebnych do sterowania nią.

Rozwiązanie

Podobnie jak w recepturze 7.13, możesz użyć rejestru przesuwnego, aby zmniejszyć liczbę pinów potrzebnych do sterowania matrycą diod LED. W tym celu to rozwiązanie używa popularnego sterownika LED MAX7219 lub MAX7221. Podłącz swoje Arduino, matrycę i MAX72xx tak, jak pokazano na rysunku 7.16.



Rysunek 7.16. MAX72xx sterujący matrycą LED 8×8

Szkic opiera się na mającej ogromne możliwości bibliotece MD_MAX72XX, która może wyświetlać tekst, rysować obiekty na wyświetlaczu oraz przekształcać w różny sposób dane na wyświetlaczu. Możesz ją znaleźć w menedżerze bibliotek (zobacz recepturę 16.2).

```
/*
  Szkic_7219_matrix
 */

#include <MD_MAX72xx.h>

// Piny sterujące 7219
#define LOAD_PIN 2
#define CLK_PIN  3
#define DATA_PIN 4

// Konfiguracja sprzętu
#define MAX_DEVICES  1
```

```
#define HARDWARE_TYPE MD_MAX72XX::PAROLA_HW
MD_MAX72XX mx = MD_MAX72XX(HARDWARE_TYPE, DATA_PIN, CLK_PIN, LOAD_PIN, MAX_DEVICES);

void setup()
{
  mx.begin();
}

void loop()
{
  mx.clear(); // Wyczyszczenie wyświetlacza

  // Rysowanie w wierszach i kolumnach
  for (int r = 0; r < 8; r++)
  {
    for (int c = 0; c < 8; c++) {
      mx.setPoint(r, c, true); // Zapalanie każdej diody LED
      delay(50);
    }

    // Przejście przez wszystkie dostępne poziomy jasności
    for (int k = 0; k <= MAX_INTENSITY; k++) {
      mx.control(MD_MAX72XX::INTENSITY, k);
      delay(100);
    }
  }
}
```

Omówienie

Podczas tworzenia obiektu matrycy należy podać typ sprzętu, numery pinów dla danych, wgrywania i zegara oraz maksymalną liczbę urządzeń (w przypadku gdy łączysz szeregowo kilka modułów). Funkcja `loop` czyści wyświetlacz, a następnie używa metody `setPoint`, aby włączyć piksele. Po odpowiednim wypełnieniu wiersza szkic przechodzi przez wszystkie dostępne poziomy jasności i przechodzi do następnego wiersza.

Pokazane tutaj numery pinów dotyczą zielonych diod LED dwukolorowej matrycy 8×8 firmy Adafruit (numer produktu 458). Jeśli używasz innego wyświetlacza, sprawdź w dokumentacji, które piny odpowiadają poszczególnym kolumnom i wierszom. Ten szkic będzie także działał z jednokolorową matrycą, ponieważ wykorzystuje tylko jeden kolor. Jeśli zauważysz, że Twój wyświetlacz pokazuje tekst od tyłu lub w inny niż zamierzony sposób, możesz zmienić typ sprzętu w linijce `#define HARDWARE_TYPE MD_MAX72XX::PAROLA_HW` z `PAROLA_HW` na `GENERIC_HW`, `ICSTATION_HW` lub `FC16_HW`. Do biblioteki MD_MAX72xx jest dołączony przykładowy szkic, *MD_MAX72xx_HW_Mapper*, który pomoże Ci wybrać odpowiedni typ sprzętu.

Opornik (zaznaczony na rysunku 7.16) kontroluje maksymalne natężenie sterujące diodą LED. W dokumentacji MAX72xx znajduje się tabela z zakresami wartości (zobacz tabelę 7.3).

Tabela 7.3. Tabela wartości opornika (z dokumentacji MAX72xx)

	Napięcie przewodzenia diody LED				
Natężenie	**1,5 V**	**2,0 V**	**2,5 V**	**3,0 V**	**3,5 V**
40 mA	12 kΩ	12 kΩ	11 kΩ	10 kΩ	10 kΩ
30 mA	18 kΩ	17 kΩ	16 kΩ	15 kΩ	14 kΩ
20 mA	30 kΩ	28 kΩ	26 kΩ	24 kΩ	22 kΩ
10 mA	68 kΩ	64 kΩ	60 kΩ	56 kΩ	51 kΩ

Zielona dioda LED oraz matryce LED pokazane na rysunku 7.16 mają napięcie przewodzenia równe 2 V i natężenie przewodzenia równe 20 mA. Zatem według tabeli 7.3 opornik powinien mieć wartość 28 kΩ, ale żeby dodać mały margines bezpieczeństwa, opornik 30 kΩ lub 33 kΩ będzie odpowiednim wyborem. W celu uniknięcia szumów powstających, gdy dioda LED jest zapalana i gaszona, wymagane są kondensatory (0.1 µF i 10 µF). Jeśli nie miałeś okazji podłączać kondensatorów odsprzęgających, zobacz sekcję „Stosowanie kondensatorów odsprzęgających" w dodatku C.

Zobacz również

Przejrzyj dokumentację MAX72xx (*https://pdfserv.maximintegrated.com/en/ds/MAX7219-MAX7221.pdf*).

7.15. Zwiększenie liczby analogowych wyjść za pomocą generatorów PWM

Problem

Chcesz mieć indywidualną kontrolę na poziomem jasności większej liczby diod LED, niż Arduino jest w stanie obsłużyć.

Rozwiązanie

Generator PCA9685 może sterować maksymalnie 16 diodami LED za pomocą tylko dwóch pinów danych I2C. Adafruit produkuje moduły, które mogą sterować wieloma serwomechanizmami lub diodami LED (numer produktu 815). Na rysunku 7.17 pokazano schemat podłączeniowy. Szkic opiera się na bibliotece Adafruit_PWMServoDriver, którą możesz zainstalować za pomocą menedżera bibliotek w Arduino IDE (zobacz recepturę 16.2).

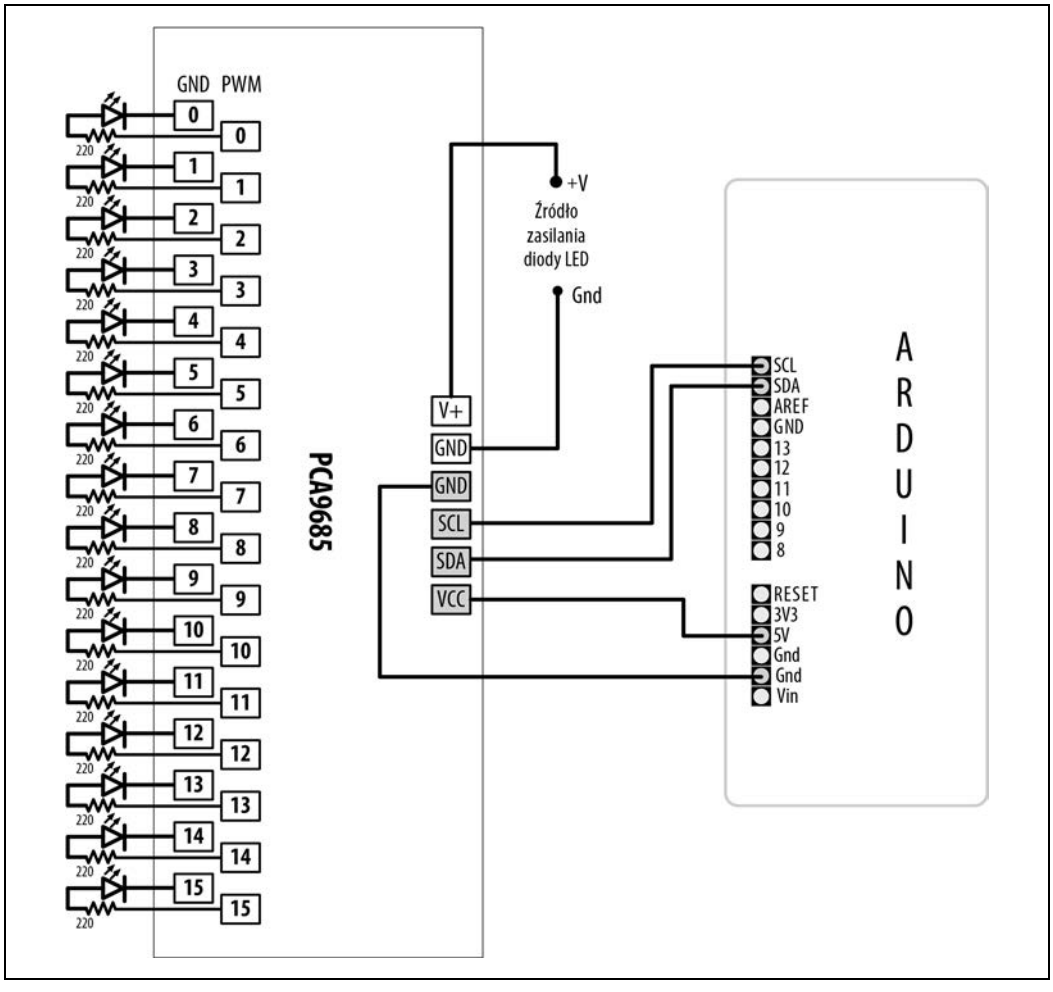
```
/*
  Szkic pwm_PCA9685
  Efekt pływającego światła stworzony z wykorzystaniem diod LED podłączonych do wszystkich wyjść PCA9685
  Wersja dla jednego PCA9685 z podłączonymi 16 diodami LED
 */

#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(); // Domyślny adres I2C to 0x40
```

```
void setup()
{
  pwm.begin(); // Inicjalizacja modułu I2C
}

int channel = 0;
int channel_direction = 1;
int intensity = 4095;       // Maksymalna jasność
int dim = intensity / 4;    // Jasność przyciemnionej diody LED

void loop()
{
  channel += channel_direction; // Inkrementacja (lub dekrementacja) numeru kanału

  // Wyłączenie wszystkich pinów
  for (int pin = 0; pin < 16; pin++) {
    pwm.setPin(pin, 0);
  }

  // Jeśli doszliśmy do kanału 0, ustaw kierunek na 1
  if (channel == 0) {
    channel_direction = 1;
  }
  else { // Jeśli jesteśmy na kanale 1 lub wyższym, ustaw poprzedni na stan przyciemniony
    pwm.setPin(channel - 1, dim);
  }

  // Ustawienie na podanym kanale maksymalnej jasności
  pwm.setPin(channel, intensity);

  if (channel < 16) { // Jeśli jesteśmy poniżej kanału 16, ustaw następny kanał na stan przyciemniony
    pwm.setPin(channel + 1, dim);
  }
  else { // Jeśli doszliśmy do kanału 16, ustaw kierunek na –1
    channel_direction = -1;
  }

  delay(75);
}
```

Omówienie

Szkic przechodzi przez każdy kanał (diodę LED). Przygasza poprzednią diodę LED, bieżący kanał maksymalnie rozjaśnia, a następny przygasza. Diody LED są sterowne za pomocą kilku podstawowych metod. Szkic zakłada, że PCA9685 jest skonfigurowany z domyślnym adresem 0x40.

Przed wywołaniem jakiejkolwiek innej funkcji szkic za pomocą metody `Adafruit_PWMServo ↪Driver.begin` inicjalizuje sterownik. Metoda `pwm.setPin` ustawia współczynnik wypełnienia impulsu danego kanału na podaną liczbę taktów z zakresu od 0 do 4095. Pierwszym argumentem jest numer kanału, a drugim poziom jasności. Każdy cykl modulacji szerokości impulsu podzielony jest na 4096 taktów. Podana dla parametru jasności wartość to liczba taktów, podczas których dioda LED powinna być włączona. Częstotliwość PWM możesz zmienić za pomocą metody `pwm.setPWMFreq` (podaj wartość w hercach, od 40 do 1600).



Rysunek 7.17. Szesnaście diod LED sterowanych za pomocą zewnętrznego generatora PWM

Zobacz również

Listę wszystkich funkcji biblioteki Adafruit_PWMServoDriver znajdziesz na stronie *https://adafruit.github.io/Adafruit-PWM-Servo-Driver-Library/html/class_adafruit___p_w_m_servo_driver.html*.

Możesz połączyć wiele modułów, łącząc szeregowo ich piny. Każdy musi mieć unikalny adres, który ustawiasz przez zlutowanie zworek oznaczonych od A0 do A5. Możesz podłączyć maksymalnie 62 moduły (*https://learn.adafruit.com/16-channel-pwm-servo-driver/chaining-drivers*).

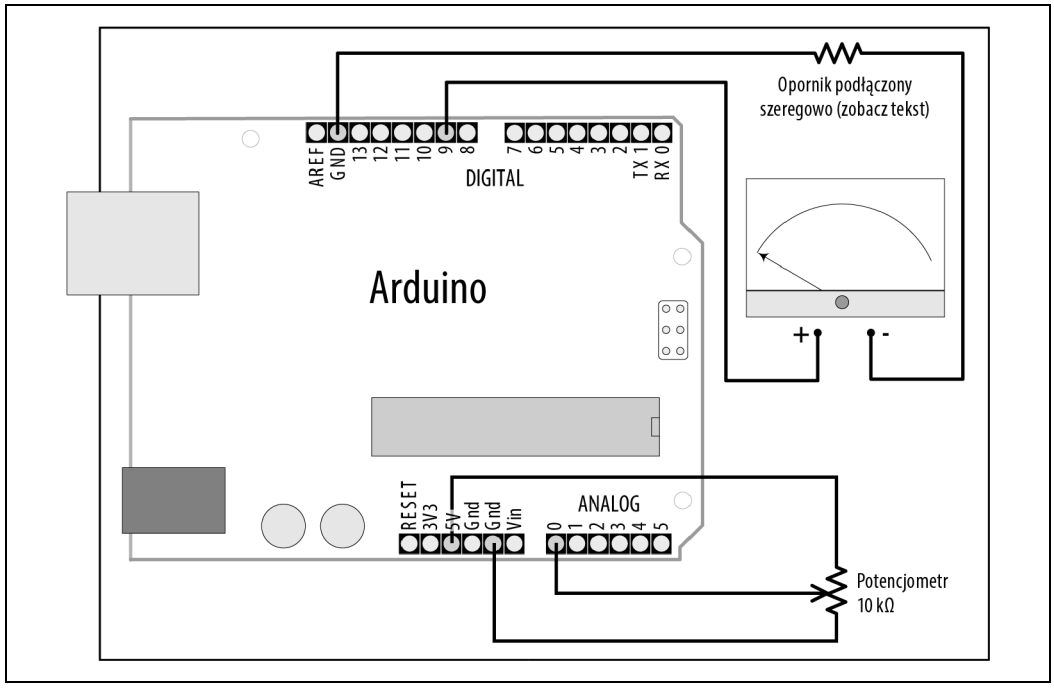
7.16. Wykorzystanie miernika analogowego jako wyświetlacza

Problem

Chcesz sterować wskazówką miernika analogowego za pomocą szkicu. Często zmieniające się odczyty są łatwiejsze do przedstawienia na analogowym mierniku, a ponadto projekt nabierze charakteru retro.

Rozwiązanie

Do analogowego wyjścia (PWM) podłącz miernik przez umieszczenie między nimi opornika (5 kΩ dla typowego miernika 1 mA). Obwód pokazano na rysunku 7.18.



Rysunek 7.18. Sterowanie analogowym miernikiem

Ruch wskaźnika zależy od pozycji potencjometru (rezystora nastawnego).

```
/*
 * Szkic analog_meter
 * Za pomocą pinu PWM steruje analogowym miernikiem
 * Poziom miernika jest kontrolowany przez rezystor nastawny podłączony do analogowego pinu wejścia
 */

const int analogInPin = A0;    // Rezystor nastawny podłączony do analogowego wejścia
const int analogMeterPin = 9; // Miernik podłączony do analogowego wyjścia
```

```
int sensorValue = 0; // Wartość odczytana z potencjometru
int outputValue = 0; // Wartość wyjściowa dla PWM (wyjście analogowe)

void setup()
{
// Nie są wymagane żadne ustawienia
}

void loop()
{
  sensorValue = analogRead(analogInPin); // Odczyt analogowej wartości wejściowej
  outputValue = map(sensorValue, 0, 1023, 0, 255); // Skalowanie dla analogowego wyjścia
  analogWrite(analogMeterPin, outputValue); // Zapisanie na pinie miernika analogowej wartości
wyjściowej
}
```

Omówienie

W tej wersji receptury 7.2 funkcja `analogWrite` steruje miernikiem, który jest zazwyczaj bardziej czuły niż diody LED. Między wyjściem Arduino a miernikiem musi być podłączony opornik obniżający natężenie Arduino do poziomu natężenia miernika.

Wartość takiego opornika będzie zależeć od czułości miernika. Rezystor 5 kΩ daje pełen zakres ruchu wskazówki miernika 1 mA. Możesz użyć opornika 4,7 kΩ, gdyż łatwiej je dostać niż te 5 kΩ, jednak prawdopodobnie będziesz musiał zmniejszyć maksymalną wartość przekazaną do analogWrite na 240 lub zbliżoną wartość. Oto jak możesz zmienić zakres w funkcji `map` dla rezystora 4,7 kΩ podłączonego do miernika 1 mA:

```
outputValue = map(sensorValue, 0, 1023, 0, 240); // Skalowanie do zakresu miernika
```

Jeśli miernik ma inną czułość niż 1 mA, musisz użyć opornika o innej rezystancji. Jego wartość można obliczyć z następującego wzoru:

$$opornik = 5000 : mA$$

I tak dla miernika o czułości 500 mikroamperów (0,5 mA) jest to 5000 : 0,5, co daje 10000 Ω (10 kΩ). Miernik 10 mA wymaga 500 Ω, a do 20 mA musimy podłączyć opornik 250 Ω.

Niektóre mierniki mają już wbudowany opornik i możliwe, że będziesz musiał poeksperymentować, aby się dowiedzieć, jaką ma wartość, ale uważaj i nie podłączaj za dużego napięcia do swojego miernika.

Zobacz również

Zapoznaj się z recepturą 7.2.

Jaki zadziwiający projekt zrobisz dziś z Arduino?

Popularność Arduino szybko rośnie. Dla niektórych jest to uzależniające hobby, dla innych — praktyczny, tani komputer, który może komunikować się ze światem zewnętrznym i obsługiwać przeróżne urządzenia elektroniczne. Arduino jest przy tym stosunkowo prosty w obsłudze nawet dla osób bez doświadczenia w programowaniu i elektronice. Wystarczy dobry pomysł i dzięki Arduino można łatwo tworzyć projekty związane z internetem rzeczy, czujniki monitorujące otoczenie, a także urządzenia reagujące na dotyk, dźwięk, temperaturę czy światło.

Książka stanowi zbiór ponad dwustu receptur ułatwiających wykorzystanie potencjału Arduino 1.8. Jest dostosowana do potrzeb osób, które chcą po prostu szybko znaleźć rozwiązanie problemu ze sprzętem czy z kodem. Zawiera informacje potrzebne do pomyślnej realizacji szerokiej gamy projektów oraz dostosowania ich do szczególnych potrzeb. Nie ma tu przydługawych rozważań teoretycznych, są za to wskazówki pozwalające na błyskawiczne napisanie działającego kodu. Praktyczne receptury umożliwiające wykonanie wielu popularnych zadań przydadzą się zarówno adeptom Arduino, jak i doświadczonym programistom, którzy chcą skutecznie korzystać z niskopoziomowych zasobów kontrolera AVR.

W książce znajdziesz receptury dotyczące:

- koncepcji pracy z programowaniem płytki Arduino
- odczytywania sygnałów cyfrowych i analogowych
- wykorzystywania różnorodnych czujników i urządzeń wejścia
- stosowania wyświetlaczy, generowania dźwięków i sterowania pracą silników
- komunikowania się ze zdalnie sterowanymi urządzeniami domowymi

Michael Margolis od ponad trzydziestu lat zajmuje się systemami czasu rzeczywistego. Doświadczenie zdobywał w takich firmach jak Sony, Microsoft i Lucent/Bell Lab.

Brian Jepson pracuje w LinkedIn Learning, gdzie zarządza kursami z zakresu projektowania i inżynierii.

Nicholas Robert Weldin pracuje w Rix Centre przy University of East London — pomaga osobom z trudnościami korzystać ze źródeł internetowych.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-7161-3

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 99,00 zł