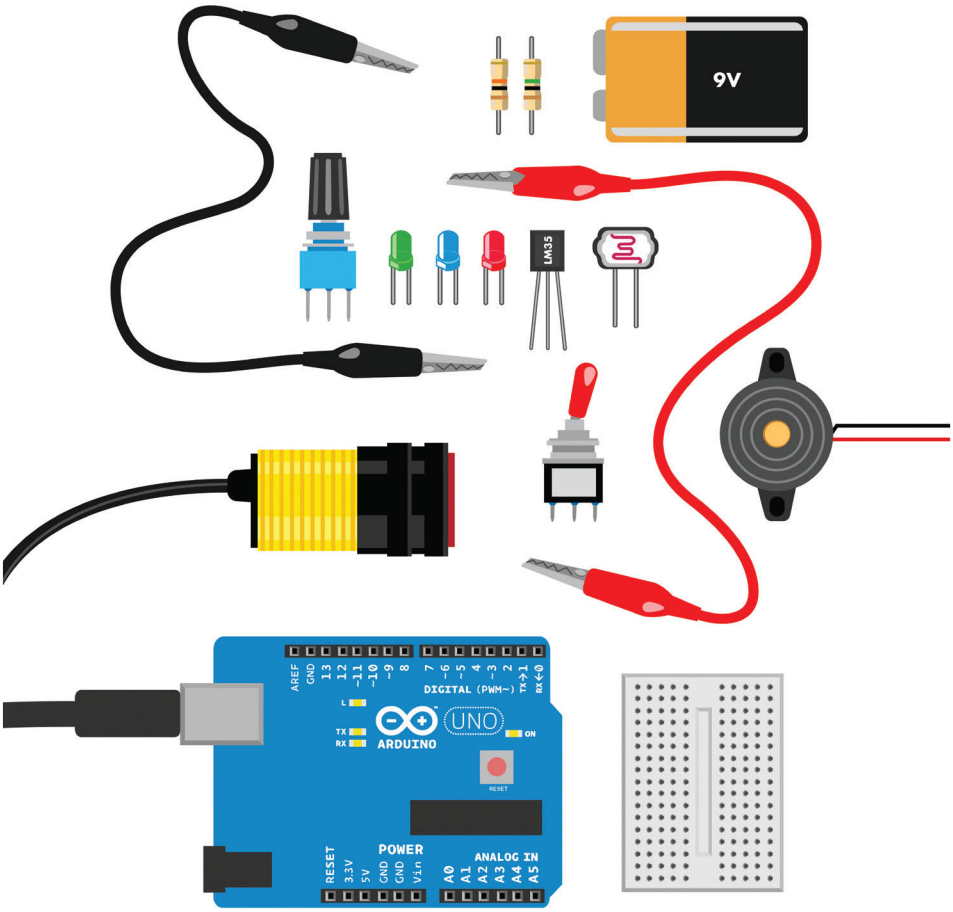
Czujniki dla początkujących



Poznaj otaczający Cię świat za pomocą elektroniki, Arduino i Raspberry Pi

Kimmo Karvinen, Tero Karvinen

Helion

Tytuł oryginału: Getting Started with Sensors

Tłumaczenie: Krzysztof Brauner

ISBN: 978-83-283-0365-2

Authorized Polish translation of the English edition of Getting Started with Sensors,
ISBN 9781449367084 © 2014 Tero Karvinen, Kimmo Karvinen, published by Maker Media Inc.

Wydawnictwo HELION
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)

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

Printed in Poland.

Spis treści

3 Czujniki i Arduino

W tym rozdziale opiszemy, w jaki sposób do dokonywania pomiarów wykorzystać Arduino — popularną platformę programistyczną, której sercem jest mikrokontroler AVR. Korzystanie z Arduino, w porównaniu do układów budowanych za pomocą poszczególnych elementów (tak jak to przedstawiono w rozdziale 2.), ma wiele zalet. Jedną z największych jest oszczędność czasu wynikająca z braku konieczności budowania skomplikowanych obwodów do przetwarzania danych pochodzących z czujników (rysunek 3.1).

Rysunek 3.1. Nasz układ Arduino, płytka stykowa i laptop

Układ Arduino uruchamia *szkice* (ang. *sketch*). To pojęcie oznacza programy, które po napisaniu przesyła się ze swojego komputera do Arduino. Musisz oczywiście napisać kilka linii kodu, ale znacznie szybciej jest utworzyć pętlę warunkową, niż zaprojektować ją za pomocą elementów elektronicznych — w każdym razie dla większości z nas.

Także modyfikacja kodu wydaje się zadaniem prostszym niż modyfikacja całego układu. Załóżmy, że chcesz dodać do swojego układu opóźnienie, z jakim czujnik dokonywał będzie kolejnych pomiarów. Mógłbyś tego dokonać, łącząc z sobą kilka kondensatorów, ale uzyskanie odpowiedniej wartości opóźnienia wymaga pewnych przemyśleń i obliczeń. Znacznie prostsze jest utworzenie prostego układu na Arduino i skorzystanie z funkcji `delay()`, która służy właśnie do generowania opóźnienia. Szybciej napiszesz kilka linii kodu i załadujesz program do Arduino, niż znajdziesz odpowiednie elementy i zmodyfikujesz układ. Na przykład:

```
delay(1000); // opóźnia wykonanie kolejnej instrukcji o 1000 milisekund (1 sekundę)
```

Jeśli jeszcze nigdy w życiu nie programowałeś, nie przejmuj się. Podstawy zrozumiesz w mig, a im bardziej będziesz dopracowywał swój kod, tym więcej się nauczysz. Tworzenie szkiców otworzy przed Tobą całe pole możliwości, jeśli chodzi o układy elektroniczne. Najwyższa pora coś zbudować!

Jeśli po raz pierwszy korzystasz z Arduino, zapoznaj się z dodatkiem B — dowiesz się stamtąd, w jaki sposób zainstalować potrzebne do pracy sterowniki i zintegrowane środowisko programistyczne (IDE). Dobrym pomysłem jest także uruchomienie prostego programu (w rodzaju „Hello, world") w celu przetestowania Twojego układu.

Tak długo, jak używasz czujników i innych elementów elektronicznych zgodnie z ich specyfikacją, korzystanie z Arduino i innych układów opartych na mikrokontrolerach uprości i zwiększy elastyczność Twoich projektów. Jednakże użycie ich niezgodnie ze specyfikacją — np. zasilenie diody LED zbyt dużym napięciem — może doprowadzić do ich uszkodzenia, a nawet uszkodzenia Twojego układu Arduino.

Projekt 6. Przycisk monostabilny[1] i rezystor podciągający

Przyciski nie są może najbardziej wyrafinowanymi elementami, ale z całą pewnością należą do grupy najczęściej używanych czujników. W odróżnieniu od wielu sposobów podłączenia przycisku do układu Arduino w tym projekcie nie jest wymagane wykorzystanie rezystora pomocniczego. A to z tego powodu, że używasz rezystorów podciągających (ang. *pull-up*) wbudowanych w układ wejścia – wyjścia ogólnego przeznaczenia (GPIO), które mogą zostać włączone za pomocą jednej linii kodu. Chyba większej kontroli nad sprzętem z poziomu oprogramowania nie można sobie wyobrazić.

Elementy

Do tego projektu będziesz potrzebował następujących elementów:

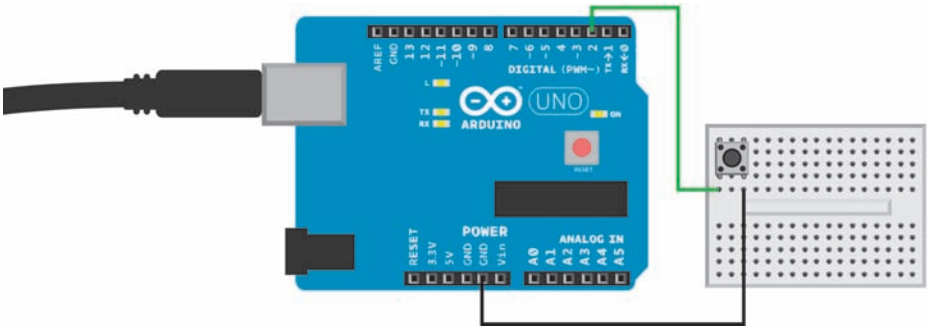
- przycisku monostabilnego (mikrostyku);
- Arduino Uno;
- kabelków połączeniowych;
- płytki stykowej.

[1] Inna nazwa to przełącznik chwilowy — *przyp. tłum.*

Budowa

Oto poszczególne kroki budowy projektu:

1. Ustaw poziomo płytkę stykową — tak jak pokazano na rysunku 3.2.



Rysunek 3.2. Przycisk monostabilny podłączony do Arduino Uno

2. Umieść przycisk w dowolnym miejscu na płytce stykowej.
3. Podłącz kabelek do tej samej kolumny na płytce stykowej, w której znajduje się wyprowadzenie przycisku, a drugi koniec kabelka podłącz do wyprowadzenia oznaczonego GND na płytce Arduino.
4. Drugi kabelek podepnij w tym samym rzędzie co kabelek pierwszy, ale w kolumnie, w której znajduje się druga para wyprowadzeń przycisku. Drugi koniec kabelka podepnij do wyprowadzenia 2 na płytce Arduino.

Uruchom kod

Po połączeniu wszystkich potrzebnych elementów uruchom kod z listingu 3.1. Po naciśnięciu przycisku zaświeci się wbudowana w Arduino dioda LED.

Listing 3.1. Szkic Arduino do wykrywania naciśnięcia przycisku

```
// button.ino – zapalenie diody LED poprzez naciśnięcie przycisku
// (c) BotBook.com – Karvinen, Karvinen, Valtokari
int buttonPin=2; // (1)
int ledPin=13; // (2)
int buttonStatus=LOW; // (3)

void setup() // (4)
{
pinMode(ledPin, OUTPUT); // (5)
pinMode(buttonPin, INPUT); // (6)
digitalWrite(buttonPin, HIGH); // wewnętrzny rezystor podciągający // (7)
}
void loop() // (8)
{
buttonStatus=digitalRead(buttonPin); // (9)
if (LOW==buttonStatus) { // (10)
digitalWrite(ledPin, HIGH); // (11)
} else { // (12)
```

```
digitalWrite(ledPin, LOW); // ⓭
}
} // ⓮
```

❶ Zainicjalizowanie kilku *zmiennych*. Od tego miejsca gdy posłużysz się zmienną `buttonPin`, Arduino wstawi w jej miejsce liczbę 2. Zmienne sprawiają, że kod jest znacznie czytelniejszy i prostszy w edycji, ponieważ wystarczy dokonać zmiany tylko w jednym miejscu. Wszystkie zmienne w tym szkicu są zmiennymi typu całkowitego (ang. *integer*), np.: 1, 2, 3 lub -500.

❷ Utworzenie nowej zmiennej do obsługi diody LED. Wbudowana dioda LED będzie sterowana za pomocą wyprowadzenia cyfrowego 13 (D13).

❸ Mimo że wartość zmiennej `buttonStatus` jest modyfikowana w innej części programu, dobrym pomysłem jest zainicjalizowanie jej do jakiejkolwiek wartości.

❹ Funkcja `setup()` jest uruchamiana jednokrotnie podczas rozruchu Arduino.

❺ Ustawienie wyprowadzenia jako `OUTPUT` (wyjście) pozwoli Ci na sterowanie nim za pomocą funkcji `digitalWrite()`.

❻ Ustawienie wyprowadzenia `buttonPin` jako `INPUT` (wejście) — abyś później mógł odczytać jego wartość za pomocą funkcji `digitalRead()`.

❼ Włączenie wbudowanego rezystora podciągającego. To tutaj dzieje się cała magia (programowa kontrola nad sprzętem). Teorię dotyczącą rezystorów podciągających poznasz w punkcie „Rezystory podciągające i Arduino”. Zauważ, że tego samego możesz dokonać, usuwając tę linię kodu i modyfikując poprzednią do postaci: `pinMode(buttonPin, INPUT_PULLUP);`. Jednak ten sposób zadziała tylko z nowszymi wersjami oprogramowania Arduino — nie jest obsługiwany w starszych wersjach.

❽ Arduino uruchamia funkcję `loop()` zaraz po zakończeniu działania funkcji `setup()`. Jak sama nazwa wskazuje[2], ta funkcja jest uruchamiana raz za razem — aż do wyłączenia Arduino.

❾ W tym miejscu odczytywana jest wartość z wyprowadzenia `buttonPin` (wcześniej zostało zdefiniowane, że jest to wyprowadzenie 2). Następnie ta wartość jest zapisywana w zmiennej `buttonStatus`. Jeśli przycisk jest *wciśnięty*, wyprowadzenie 2 zostanie zwarte do masy i funkcja `digitalRead()` zwróci wartość `LOW`. Jeśli przycisk *nie jest wciśnięty*, wyprowadzenie 2 nie zostanie zwarte do masy, więc wbudowany rezystor podciągający wymusi na nim napięcie +5 V i funkcja zwróci wartość `HIGH`.

❿ Dwa znaki równości (==) oznaczają porównanie z sobą dwóch wyrażeń. Zauważ, że mają zupełnie inne znaczenie niż pojedynczy znak równości (=), który służy do przypisywania wartości zmiennym. W instrukcji `if` — gdy chcesz dokonać porównania — zawsze korzystaj z podwójnego znaku równości. Aby uniknąć popełnienia błędu, z lewej strony operatora porównania wpisz wyrażenie `LOW` — jest to stała, więc nie można do niej przypisać wartości. Jeśli się pomylisz i użyjesz pojedynczego znaku równości, Arduino zgłosi błąd podczas kompilacji i próby załadowania szkicu.

⓫ W tym miejscu jest zapalana dioda LED.

⓬ Instrukcja `else` ma za zadanie obsłużyć sytuację, gdy warunek w instrukcji `if` nie zostanie spełniony. Blok kodu ({}), który znajduje się zaraz za instrukcją `else`, zostanie wykonany, gdy będzie spełniony warunek `LOW != buttonStatus` (co oznacza to samo co `buttonStatus != LOW` — czyli wartość `buttonStatus` jest różna od `LOW`).

[2] Słowo *loop* w języku angielskim oznacza *pętla* — *przyp. tłum.*

⓭ Wyłączenie diody LED.

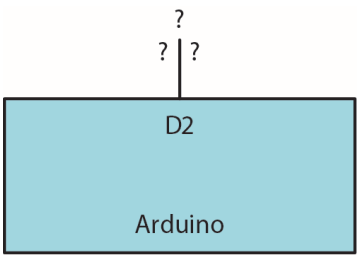
⓮ Gdy tylko funkcja `loop()` zakończy działanie, Arduino uruchomi ją ponownie od samego początku.

Kody źródłowe do przykładów zawartych w tej książce znajdziesz pod adresem *ftp://ftp.helion.pl/przyklady/czujpo.zip*

Rezystory podciągające i Arduino

Przycisk monostabilny działa poprawnie, ponieważ wykorzystałeś *rezystor podciągający* wbudowany w Arduino. Pewnie zastanawiasz się, co podciąga rezystor podciągający — otóż podciąga on dane wyprowadzenie do napięcia zasilającego 5 V.

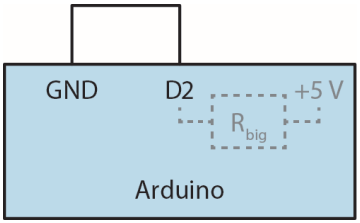
Uważaj na „wiszące" wyprowadzenia. Odczyt z wyprowadzenia, które nie jest do niczego podłączone („wiszące"), jest nieprzewidywalny. Na przykład jeśli nie podłączysz niczego do wejścia cyfrowego `D2` i dokonasz odczytu za pomocą funkcji `digitalRead()`, nie masz żadnej gwarancji, że odczyt będzie poprawny. Może to być wartość `HIGH`, ale równie dobrze może to być wartość `LOW`, która na dodatek zmieni się dziesięć razy w ciągu jednej sekundy — „wiszące" wyprowadzenie daje więc w rezultacie bezużyteczną wartość (rysunek 3.3).



Rysunek 3.3. Odczyt „wiszącego" wyprowadzenia nie ma sensu — dodaj rezystor podciągający

Zintegrowane środowisko programistyczne (IDE) Arduino jest dystrybuowane z wieloma przydatnymi przykładowymi szkicami. Jeśli przyjrzysz się dobrze szkicowi do obsługi przycisku (*menu Plik/Przykłady/02.Digital/Button*), okaże się, że ten szkic wymaga dodatkowego rezystora podciągającego i nasz przykładowy program nie zadziała poprawnie. Wynika to z faktu, że ten szkic wykorzystuje *rezystor ściągający* (ang. *pull-down*) tak, aby stan wysoki `HIGH` oznaczał, że przycisk został naciśnięty, a stan niski `LOW` oznaczał, że przycisk został zwolniony.

Jeśli podłączysz to samo wejście `D2` do masy (`GND`) — funkcja `digitalRead()` z całą pewnością zwróci wartość `LOW` (rysunek 3.4). Odpowiada to sytuacji, gdy przycisk został naciśnięty.

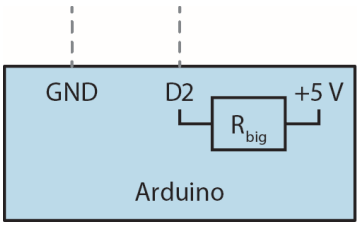


Rysunek 3.4. Stan niski (LOW). Naciśnięcie przycisku podłączonego między wyprowadzenie D2 i masę (GND) jest równoznaczne ze zwarciem wyprowadzenia D2 do masy

A co w przypadku, gdy przycisk nie został naciśnięty? Wiadomo już, że aby odczytać z wejścia jakąś wartość, trzeba je do czegoś podłączyć. Nawet jeśli podłączysz do niego przycisk, to tak naprawdę — dopóki go nie naciśniesz — nie jest on do niczego podłączony.

Wybawieniem z tej sytuacji jest właśnie rezystor podciągający!

Rezystor podciągający łączy wyprowadzenie D2 z napięciem +5 V. Prąd płynący przez rezystor jest bardzo mały — niemniej połączenie istnieje. Jeśli nic innego nie zostanie podłączone do wyprowadzenia D2, rezystor podciągnie je do stanu wysokiego HIGH. Tak więc gdy przycisk nie jest naciśnięty, wyprowadzenie D2 nie jest zwarte do masy — czyli jest podłączone do +5 V poprzez rezystor podciągający. Oznacza to, że na wyprowadzeniu D2 będzie się utrzymywał stan wysoki do momentu naciśnięcia przycisku (rysunek 3.5).



Rysunek 3.5. Stan wysoki (HIGH). Wyprowadzenie D2 jest podłączone poprzez rezystor podciągający do +5 V

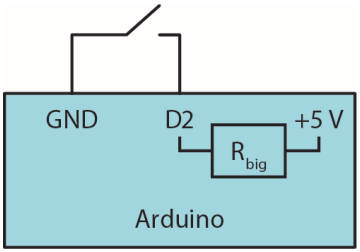
Ale co się stanie, gdy naciśniesz przycisk? Oczywiście wyprowadzenie D2 zostanie zwarte do masy, ale jednocześnie jest podłączone do +5 V, czyż nie? Możesz pomyśleć: „Chyba nie każecie mi łączyć z sobą masy i +5 V? Przecież to zwarcie!”.

Mimo że zarówno wyprowadzenie GND, jak i D2 są podłączone do +5 V, to jest to połączenie za pomocą rezystora. A ponieważ połączenie między GND i D2 charakteryzuje się o wiele mniejszą rezystancją niż połączenie do +5 V, to prąd płynie właśnie przez GND i D2. Inaczej mówiąc: rezystor podciągający wymusza na wyprowadzeniu D2 stan wysoki, gdy nie jest ono zwarte do masy.

Typowy rezystor podciągający ma rezystancję w zakresie od kilkudziesięciu tysięcy omów do kilku milionów omów (np. od 20 kΩ do 2 MΩ).

Oto jak działa Twój układ — naciśnij przycisk, a na wyprowadzeniu D2 pojawi się stan niski (LOW). Zwolnij przycisk, a rezystor podciągający wymusi na wejściu stan wysoki (HIGH). Jak się zorientować, że w układzie jest „wiszące” wyprowadzenie? Jeśli chcesz podłączyć do Arduino jakikolwiek czujnik, który ma dwa wyprowadzenia, zastanów się, czy istnieje taki stan, w którym nastąpi jego rozłączenie (w przypadku przycisku jest

to stan, gdy przycisk nie jest naciśnięty). Jeśli tak, to prawdopodobnie będziesz musiał zastosować rezystor podciągający, co pozwoli na wymuszenie stanu wysokiego na wejściu (rysunek 3.6).



Rysunek 3.6. Wbudowany rezystor podciągający. Włączysz go poprzez pinMode(pin, INPUT), a następnie digitalWrite(pin, HIGH) lub tylko poprzez pinMode(pin, INPUT_PULLUP)

Jeśli wolisz jednak wykorzystać zewnętrzny rezystor pomocniczy, to popularniejsze jest zastosowanie go w funkcji rezystora ściągającego — aby wymusić na wyprowadzeniu stan niski `LOW`. Pamiętaj, aby w takim przypadku odwrócić logikę w Twoim programie (stan wysoki `HIGH` będzie sygnalizował naciśnięcie przycisku, a stan niski `LOW` — zwolnienie przycisku).

To tyle, jeśli chodzi o teorię. Jak widzisz, zbudowanie nawet takiego przykładowego prostego przełącznika niesie z sobą sporo wiedzy o tym, w jaki sposób podłącza się czujniki do układu. Teraz nadeszła pora, aby pobawić się czymś bardziej praktycznym.

Projekt 7. Wykrywanie przedmiotów za pomocą czujnika zbliżeniowego na podczerwień

Czujniki podczerwieni (takie jak przełącznik pokazany na rysunku 3.7) składają się z dwóch części: nadajnika podczerwieni i odbiornika. Nadajnik to po prostu dioda LED, która emituje światło niewidoczne dla ludzkiego oka — długość fali takiego światła jest dłuższa od fal wchodzących w zakres pasma widzialnego. Diody podczerwieni znajdują się np. w pilotach do telewizorów.

Rysunek 3.7. Przełącznik na podczerwień

Zadaniem odbiornika podczerwieni jest — jak sama nazwa wskazuje — odebranie odbitego promienia światła. Każda przeszkoda ustawiona na drodze powoduje, że odbije się od niej większa ilość światła niż normalnie, a to z kolei informuje, że przed czujnikiem znajduje się jakiś przedmiot.

Tego typu czujnik może działać nieprawidłowo w otoczeniu silnego światła słonecznego lub gdy obiekt będący celem detekcji jest wykonany z bardzo ciemnego materiału.

Przełącznik tego typu można nabyć w sklepach internetowych pod nazwą „cyfrowy czujnik odległości" lub „czujnik zbliżeniowy". Oznaczony jest symbolem E18-D50NK lub E18-D80NK.

Czujnik zbliżeniowy może być przydatny w wielu projektach. Szczególnie pomocny okaże się w sytuacji, gdy trzeba określić, czy w pobliżu znajduje się jakiś przedmiot, ale dokładna informacja o jego odległości jest niepotrzebna. Zasięg detekcji czujnika ustawia się na pomocą małego potencjometru z tyłu czujnika — tak jak pokazano to na rysunku 3.8.

Rysunek 3.8. Regulacja czujnika zbliżeniowego

Elementy

Do tego projektu będziesz potrzebował następujących elementów:

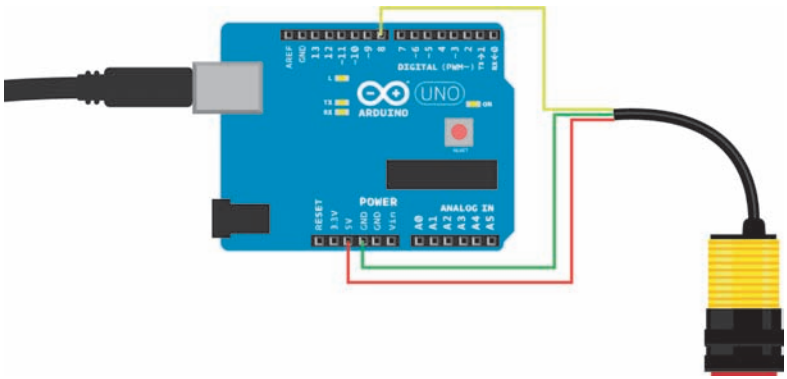
- czujnika zbliżeniowego na podczerwień;
- Arduino Uno;
- kabelków połączeniowych.

Jeśli Twój czujnik zakończony jest złączem z wyprowadzeniami żeńskimi, będziesz potrzebował kabelków połączeniowych, aby podłączyć go do Arduino. Jeśli natomiast zakończony jest złączem z wyprowadzeniami męskimi, będziesz potrzebował kabelków oraz płytki stykowej.

Jeśli masz czujnik z osobnymi wyprowadzeniami dla masy, 5 V i sygnału, ten diagram sprawdzi się w Twoim przypadku. My użyliśmy czujnika z rozdzielonymi wyprowadzeniami, więc mogliśmy je podłączyć bezpośrednio do układu Arduino.

Budowa

Upewnij się, że Arduino jest wyłączone. Podłącz układ w sposób przedstawiony na rysunku 3.9, a następnie uruchom szkic z listingu 3.2. Gdy umieścisz jakikolwiek przedmiot bezpośrednio przed czujnikiem, zapali się wbudowana w Arduino dioda LED.



Rysunek 3.9. Podłączenie czujnika zbliżeniowego do Arduino

Podłącz czujnik zgodnie z poniższą instrukcją.

1. Kabelek żółty (sygnałowy) podłącz do wyprowadzenia cyfrowego 8 na płytce Arduino.
2. Kabelek czerwony (plus zasilania) podłącz do wyprowadzenia +5 V na płytce Arduino.
3. Kabelek zielony (masa) podłącz do wyprowadzenia GND na płytce Arduino.

Uruchom kod

Listing 3.2. Szkic Arduino do detekcji obiektów

```
// infrared_proximity.ino – dioda LED zapali się, gdy w pobliżu czujnika
// znajdzie się jakiś obiekt
// (c) BotBook.com – Karvinen, Karvinen, Valtokari
int irPin=8; // ❶
int ledPin=13;
int objectDetected=LOW; // ❷
void setup() {
pinMode(ledPin, OUTPUT);
pinMode(irPin, INPUT);
digitalWrite(irPin, HIGH); // wewnętrzny rezystor podciągający
}
void loop() {
objectDetected=digitalRead(irPin);
if (LOW==objectDetected) {
digitalWrite(ledPin, HIGH);
} else {
digitalWrite(ledPin, LOW);
}
}
```

❶ Podobnie jak w programie z listingu 3.1 — z tą różnicą, że nazwa zmiennej `buttonPin` została zmieniona na `irPin`.

❷ Podobnie — nazwa zmiennej `buttonStatus` została zmieniona na `objectDetected`.

Powinieneś rozpoznać ten kod. Gdy się mu przyjrzysz, zauważysz, że to ten sam kod, z którego korzystałeś w listingu 3.1. Zmieniły się dwie zmienne — ale są to właściwie tylko kosmetyczne zmiany. Spróbuj poeksperymentować: czy uda Ci się tak zmodyfikować układ z czujnikiem podczerwieni, aby zadziałał z programem z listingu 3.1? Czy uda Ci się tak zmodyfikować układ z przyciskiem, aby zadziałał z programem z listingu 3.2?

Wiele czujników ma taki sam interfejs. Teraz, gdy już potrafisz korzystać z dwóch podobnych przełączników (nazywamy je *cyfrowymi czujnikami rezystancyjnymi*), możesz spożytkować tę wiedzę podczas pracy z innymi czujnikami tego samego rodzaju.

Analogowe czujniki rezystancyjne a dzielniki napięcia

Większość czujników, z których będziesz korzystał, to analogowe czujniki rezystancyjne. Odpowiedzią czujnika jest stopniowa zmiana rezystancji. Na przykład: im większa będzie siła wywierana na czujnik nacisku, tym mniejsza będzie rezystancja czujnika.

Jako że Arduino (podobnie jak Raspberry Pi) nie jest w stanie bezpośrednio mierzyć rezystancji, warto dowiedzieć się, w jaki sposób wykorzystać do tego celu *dzielniki napięcia*. Zasada działania dzielnika napięcia jest zbliżona do działania rezystorów podciągających i ściągających — zamiast jednego rezystora używa się dwóch szeregowo połączonych rezystancji, wpiętych między +5 V a GND (masę):

- rezystor pomocniczy;
- czujnik, którego rezystancja się zmienia w zależności od stanu, w jakim się znajduje.

Do pomiaru napięcia, które się pojawi między tymi dwiema rezystancjami, można wykorzystać wejście analogowe. Wartość napięcia będzie się zmieniała proporcjonalnie do stanu czujnika.

Projekt 8. Pomiar obrotu z wykorzystaniem potencjometru

Potencjometr (przedstawiony na rysunku 3.10) jest rezystorem nastawnym. Do zmiany rezystancji służy pokrętło lub suwak. Potencjometry są elementami urządzeń, które wykorzystujemy w życiu codziennym — np. w odbiorniku radiowym bądź żelazku. W sterowanych zdalnie samolotach serwomechanizmy określają swoje położenie kątowe właśnie za pomocą potencjometrów.

W pewnym stopniu analogowe czujniki rezystancyjne mogą się zachowywać jak czujniki cyfrowe — przy założeniu składającym się z dwóch warunków:

- przy ustawieniu najmniejszej rezystancji powinny przewodzić taką ilość prądu, aby poziom napięcia został zinterpretowany jako wysoki — `HIGH`;
- przy ustawieniu największej rezystancji powinny stawiać taki opór, aby poziom napięcia został zinterpretowany jako niski — `LOW`.

Sprawdź to sam! Powróć na chwilę do projektu 6. „Przycisk monostabilny i rezystor podciągający” — zbuduj ten układ i uruchom program. Następnie zastąp przycisk monostabilny potencjometrem — podłącz jedną

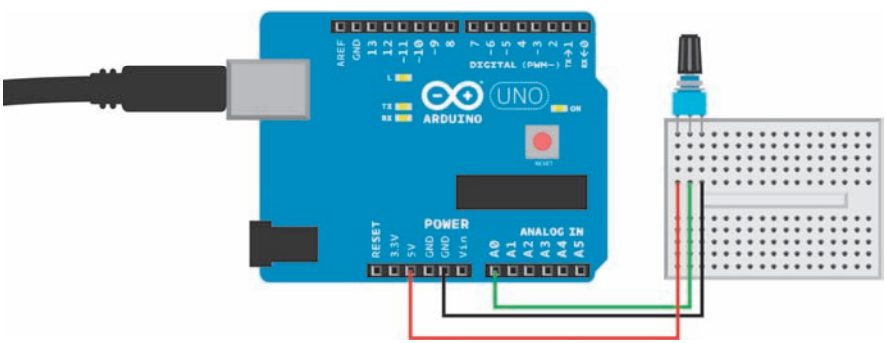
Rysunek 3.10. Potencjometry mogą posłużyć do pomiaru obrotu

skrajną nóżkę potencjometru do wyprowadzenia +5 V, drugą skrajną nóżkę do wyprowadzenia GND, a środkową nóżkę do cyfrowego wyprowadzenia 2. Teraz pokręć potencjometrem w zakresie minimum – maksimum. Jeśli rezystancja potencjometru jest wystarczająco duża, dioda LED powinna się włączać i wyłączać.

Ponieważ ten program utworzyliśmy z myślą o pokazaniu sposobu użycia przełącznika, jego funkcjonalność ogranicza się do włączania albo wyłączania diody. Aby wykorzystać w pełni właściwości potencjometru do płynnej zmiany napięcia, należy sięgnąć po funkcję `analogRead()`. W przypadku czujników analogowych ta funkcja zwraca wartości z zakresu 0 – 1023 (w przeciwieństwie do wartości `HIGH` – `LOW`).

Ten projekt posłuży Ci do nauczenia się sposobu kontrolowania prędkości migania diody LED za pomocą potencjometru. Po pokręceniu potencjometrem w jedną stronę dioda będzie migała powoli. Natomiast jeśli nim pokręcisz w przeciwną stronę, będzie migała szybciej.

Na rysunku 3.11 przedstawiono diagram układu.



Rysunek 3.11. Podłączenie potencjometru do wyprowadzeń +5 V,GND i A0 na płytce Arduino Uno

Potencjometr odgrywa tutaj rolę dzielnika napięcia. Jedno skrajne wyprowadzenie jest podłączone do +5 V, drugie skrajne wyprowadzenie do GND. Wyprowadzenie środkowe jest podłączone do wejścia analogowego A0. W miarę jak kręcisz potencjometrem, rezystancja po obu stronach wyprowadzenia środkowego się zmienia. W przypadku potencjometru 10 kΩ (podane tutaj wartości są przybliżone, ponieważ dokładność tanich potencjometrów jest ograniczona) możemy określić trzy przykładowe stany.

Jeśli pokrętło jest skręcone maksymalnie w jedną stronę[3]:

- rezystancja pomiędzy plusem napięcia a wyprowadzeniem środkowym jest zerowa;
- rezystancja pomiędzy minusem napięcia a wyprowadzeniem środkowym jest równa 10 kΩ;
- na wejściu A0 pojawia się napięcie +5 V;
- funkcja `analogRead(potPin)` zwraca wartość `1023`.

Jeśli pokrętło jest skręcone w przeciwną stronę:

- rezystancja pomiędzy plusem napięcia a wyprowadzeniem środkowym jest równa 10 kΩ;
- rezystancja pomiędzy minusem napięcia a wyprowadzeniem środkowym jest zerowa;
- na wejściu A0 pojawia się napięcie 0 V;
- funkcja `analogRead(potPin)` zwraca wartość `0`.

Jeśli pokrętło jest wypośrodkowane:

- rezystancja pomiędzy plusem napięcia a wyprowadzeniem środkowym jest równa 5 kΩ;
- rezystancja pomiędzy minusem napięcia a wyprowadzeniem środkowym jest równa 5 kΩ;
- na wejściu A0 pojawia się napięcie 2,5 V;
- funkcja `analogRead(potPin)` zwraca wartość ok. `512`.

Elementy

Do tego projektu będziesz potrzebował następujących elementów:

- potencjometru (zalecamy taki o rezystancji ok.10 kΩ);
- Arduino Uno;
- kabelków połączeniowych;
- płytki stykowej.

Budowa

Połącz elementy w sposób przedstawiony na rysunku 3.11.

Uruchom kod

Uruchom kod z listingu 3.3.

Listing 3.3. Szkic Arduino do odczytu obrotu potencjometru

```
// pot.ino – kontroluje szybkość migania diody LED za pomocą potencjometru
// (c) BotBook.com – Karvinen, Karvinen, Valtokari
int potPin=A0; //❶
int ledPin=13; //❷
int x=0; // 0..1023 //❸
```

[3] To, która to jest strona, zależy od sposobu podpięcia potencjometru do płytki — wypróbuj podłączenie w obie strony.

```
void setup() { //❹
pinMode(ledPin, OUTPUT); //❺
}
void loop() { //❻
x=analogRead(potPin); //❼
digitalWrite(ledPin, HIGH); //❽
delay(x/10); //❾
digitalWrite(ledPin, LOW); //❿
delay(x/10); //⓫
} //⓬
```

❶ W poprzednich projektach korzystałeś tylko z wejść – wyjść cyfrowych. Odwoływałeś się do nich poprzez numer (np. `13` to wyprowadzenie cyfrowe numer 13). Analogowe wejścia są także ponumerowane, ale dodatkowo przed numerem dodaje się literkę `A` (np. `A0`, `A1` itd.). Wejścia analogowe znajdują się na płytce Arduino po przeciwnej stronie niż wyprowadzenia cyfrowe.

❷ Pin `13` jest połączony z wbudowaną w Arduino diodą LED.

❸ Zmienna `x` przechowuje wartość odczytaną przez funkcję `analogRead()`. Zauważ, że w komentarzu wskazaliśmy zakres zwracany przez funkcję. Tę informację można znaleźć także w dokumentacji Arduino, ale dobrze jest mieć ją pod ręką.

❹ Funkcja `setup()` służy do jednorazowej inicjalizacji.

❺ Ustawienie wyprowadzenia `D13` jako wyjście (`OUTPUT`), co pozwoli później na sterowanie nim za pomocą funkcji `digitalWrite()`. Zauważ, że mimo iż do odczytu wykorzystane zostanie wyprowadzenie `A0`, nie ma konieczności konfigurowania go jawnie jako wejścia, ponieważ funkcja `analogRead()` działa w inny sposób niż funkcja `digitalRead()`.

❻ Funkcja `loop()` jest wywoływana automatycznie po zakończeniu działania funkcji `setup()`.

❼ Odczytanie napięcia z wejścia `potPin` (`A0`) za pomocą funkcji `analogRead()`. W wyniku otrzymasz wartość z przedziału pomiędzy `0` (0 V — stan niski `LOW`) a `1023` (+5 V — stan wysoki `HIGH`). Ta wartość zapisywana jest w zmiennej `x`.

❽ Włączenie wbudowanej diody LED.

❾ Odczekanie x/10 milisekund. Oznacza to, że przy skręconym do maksimum potencjometrze będzie to 0 ms (`A0` odczyta 0 V), a przy skręconym potencjometrze w przeciwną stronę będzie to 0,1 ms (1023/10 ms).

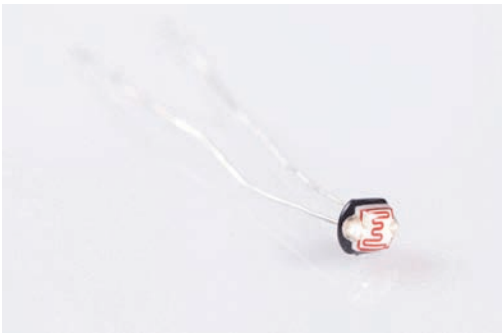
❿ Wyłączenie wbudowanej diody LED.

⓫ Ponownie odczekanie, aby dioda LED została przez moment wyłączona.

⓬ Po zakończeniu działania funkcji `loop()` jest ona wywoływana ponownie.

Projekt 9. Pomiar jasności światła za pomocą fotorezystora

Jest jasno czy ciemno? Fotorezystor stawia mniejszy opór przepływającemu przez niego prądowi, gdy zostanie oświetlony silnym światłem (rysunek 3.12). Nasi uczniowie wykorzystują fotorezystory przy konstruowaniu robotów, które „lubią" światło (bądź go „nie lubią"), układów, które automatycznie zapalają światła po zmroku, lub do budowy alarmów włamaniowych.



Rysunek 3.12. Fotorezystor lepiej przewodzi prąd, jeśli zostanie oświetlony jasnym światłem

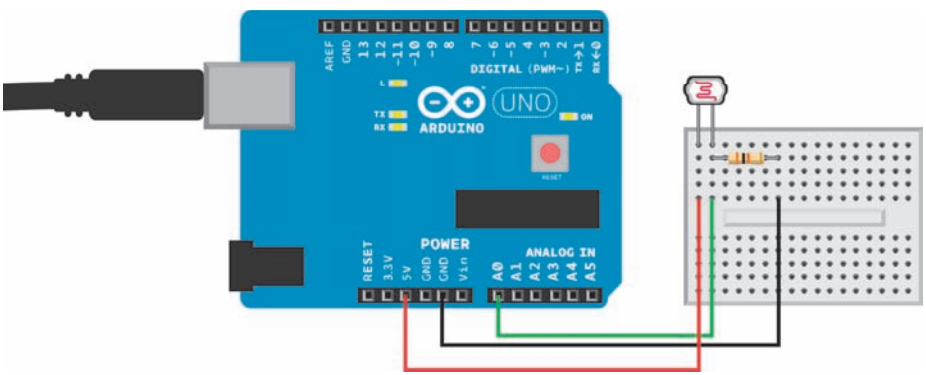
Elementy

Do tego projektu będziesz potrzebował następujących elementów:

- fotorezystora (zalecany 10 kΩ);
- rezystora 10 kΩ (kod czteropaskowy — brązowy, czarny, pomarańczowy; kod pięciopaskowy — brązowy, czarny, czarny, czerwony; odpowiednio: czwarty i piąty pasek jest zależny od tolerancji rezystora);
- Arduino Uno;
- kabelków połączeniowych;
- płytki stykowej.

Budowa

Rysunek 3.13 przedstawia diagram układu, który zbudujesz w tym projekcie. Złóż układ zgodnie z diagramem oraz uruchom szkic z listingu 3.4. Zwróć uwagę na obecność rezystora 10 kΩ — wchodzi on w skład dzielnika napięcia. W projekcie 8. „Pomiar obrotu z wykorzystaniem potencjometru" nie było potrzeby używania zewnętrznego rezystora, ponieważ już sam potencjometr pełni funkcję dzielnika napięcia.



Rysunek 3.13. Diagram układu z fotorezystorem

Aby wytworzyć zmianę napięcia zależną od zmiany rezystancji fotorezystora, musisz wykorzystać dzielnik napięcia złożony z dwóch rezystancji (opornik 10 kΩ i fotorezystor) oraz zmierzyć wartość napięcia, które pojawi się między nimi. Na tej podstawie będzie można określić, czy w otoczeniu jest ciemno, czy też jasno.

Uruchom kod

Listing 3.4. Szkic Arduino korzystający z fotorezystora

```
// photoresistor.ino – szybsze miganie diody w ciemności, wolniejsze w świetle
// (c) BotBook.com – Karvinen, Karvinen, Valtokari
int photoPin=A0;
int ledPin=13;
int x=-1; // 0..1023

void setup() {
pinMode(ledPin, OUTPUT);
}
void loop() {
x=analogRead(photoPin);
digitalWrite(ledPin, HIGH);
delay(x/10); //
digitalWrite(ledPin, LOW);
delay(x/10);
}
```

Jesteśmy pewni, że zauważyłeś, że powyższy kod wygląda podobnie jak listing 3.3! To ten sam kod — zmieniliśmy jedynie nazwę zmiennej `potPin` na `photoPin` oraz komentarz w pierwszej linii. Uruchom program i sprawdź go. Co się dzieje, gdy umieścisz fotorezystor w cieniu bądź go całkowicie zasłonisz?

Większość popularnych czujników to analogowe czujniki rezystancyjne, więc ich podłączenie do układu wygląda podobnie. W przyszłości — gdy będziesz miał do czynienia z innymi czujnikami — możesz skorzystać z tego samego podejścia.

Projekt 10. Pomiar siły nacisku za pomocą czujnika FlexiForce[4]

Rezystancyjny czujnik siły nacisku, taki jak FlexiForce (rysunek 3.14), ma najczęściej postać płaskiej folii zakończonej obszarem aktywnym i może zostać wykorzystany do badania działającej na niego siły. W stanie spoczynku charakteryzuje się on bardzo dużą rezystancją, natomiast po przyłożeniu siły do obszaru aktywnego czujnika jego rezystancja maleje.

[4] Podobne czujniki można nabyć w sklepach internetowych pod symbolem CZN-CP1 — *przyp. tłum.*

Rysunek 3.14. Rezystancyjny czujnik siły nacisku mierzy wywieraną na niego siłę

Elementy

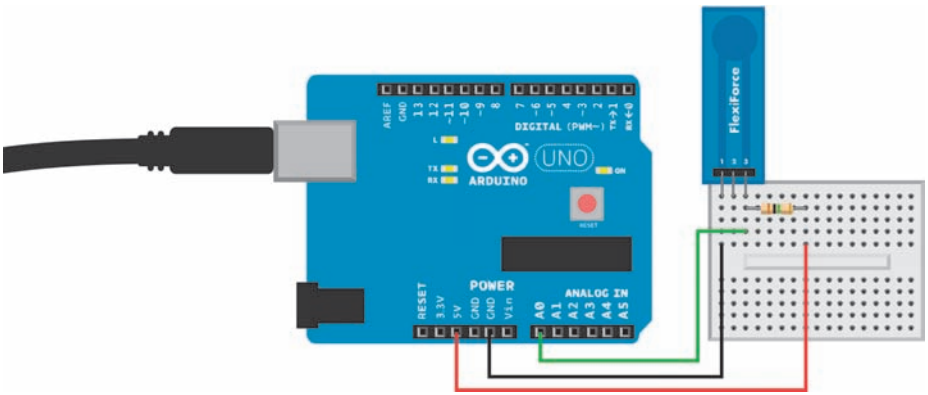
Do tego projektu będziesz potrzebował następujących elementów:

- rezystancyjnego czujnika siły;
- rezystora 1 MΩ (kod czteropaskowy — brązowy, czarny, zielony; kod pięciopaskowy — brązowy, czarny, czarny, żółty; odpowiednio czwarty i piąty pasek jest zależny od tolerancji rezystora);
- Arduino Uno;
- kabelków połączeniowych;
- płytki stykowej.

Budowa

Podobnie jak było w projekcie 9. „Pomiar jasności światła za pomocą fotorezystora", wykorzystasz tutaj dzielnik napięcia. Również w tym przypadku musisz użyć rezystora o wartości zbliżonej do rezystancji czujnika. W naszym czujniku odpowiedni okazał się opornik 1 MΩ.

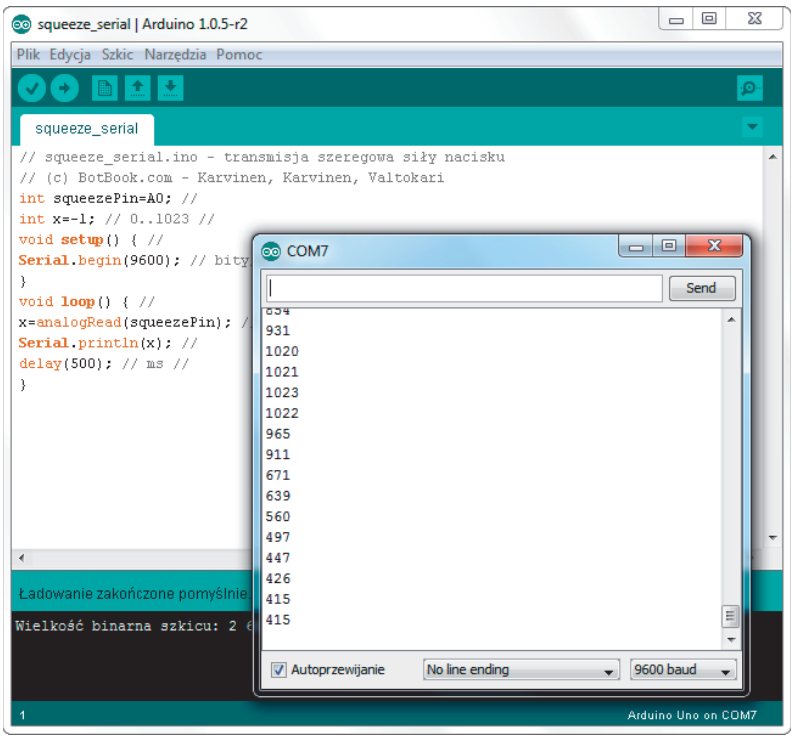
Zadaniem wyprowadzenia A0 będzie pomiar napięcia z naszego dzielnika (zob. ramka „Analogowe czujniki rezystancyjne a dzielniki napięcia"). To napięcie będzie się zmieniać w odpowiedzi na przyłożoną do czujnika siłę. W efekcie pojawi się wartość pomiędzy 0 (napięcie bliskie 0 V) a 1023 (napięcie bliskie +5 V). Zbuduj układ przedstawiony na rysunku 3.15 i załaduj szkic z listingu 3.5.



Rysunek 3.15. Diagram układu z czujnikiem FlexiForce

Nasłuch na porcie szeregowym

Nie musisz się domyślać tego, co dzieje się wewnątrz Arduino. Układ Arduino potrafi komunikować się z Twoim komputerem poprzez wbudowany port szeregowy i przesyłać do niego informacje (istnieje także możliwość przesyłania informacji w drugą stronę — z komputera do Arduino), co przedstawiono na rysunku 3.16. Transmisja następuje poprzez port USB wbudowany w Arduino.



Rysunek 3.16. Monitor portu szeregowego

Systemy wbudowane bardzo często wykorzystują do komunikacji porty szeregowe. W naszej książce na temat sterowania robotem za pomocą fal mózgowych (*Make a Mind-Controlled Arduino Robot*) opisaliśmy, w jaki sposób posłużyliśmy się portem szeregowym w celu zhakowania dostępnej na rynku opaski na głowę wykorzystywanej przy elektroencefalografii. Pozwoliło nam to na kontrolowanie robota zbudowanego na bazie Arduino za pomocą fal mózgowych! Kolejnym dobrym przykładem jest sterowany przez telefon robot do gry w piłkę nożną — został opisany w innej naszej książce *Make: Arduino Bots and Gadgets*. Tutaj z kolei do komunikacji z telefonem działającym pod kontrolą systemu Android wykorzystaliśmy transmisję poprzez łącze Bluetooth. Jednak port szeregowy służy nie tylko do przekazywania urządzeniom instrukcji. Podczas budowania wczesnej wersji prototypu naszego satelity użyliśmy portu szeregowego do debugowania projektu. I właśnie w ten sam sposób wykorzystamy go tutaj.

Uruchom kod

Listing 3.5. Szkic Arduino do odczytu czujnika siły

```
// squeeze_serial.ino – transmisja szeregowa siły nacisku
// (c) BotBook.com – Karvinen, Karvinen, Valtokari
int squeezePin=A0; // ❶
int x=-1; // 0..1023 // ❷
void setup() { // ❸
Serial.begin(9600); // bity/s // ❹
}
void loop() { // ❺
x=analogRead(squeezePin); // ❻
Serial.println(x); // ❼
delay(500); // ms // ❽
}
```

❶ To wyprowadzenie odpowiada za odczyt czujnika (wejście analogowe `A0`).

❷ Zmienna globalna `x` zadeklarowana została w ten sposób, aby umożliwić dostęp do niej wewnątrz innych funkcji w szkicu. Zainicjalizowana została wartością –1 (której funkcja `analogRead()` nie zwraca), aby ułatwić debugowanie — jeśli zauważysz tę wartość jako wynik odczytu, najprawdopodobniej w programie jest błąd.

❸ Funkcja `setup()` jest wywoływana automatycznie; służy do inicjalizacji.

❹ Prędkość transmisji portu szeregowego (wybraliśmy prędkość 9600, ponieważ jest to domyślna wartość używana przez monitor portu szeregowego).

❺ Funkcja `loop()` jest wykonywana automatycznie (i wielokrotnie) po zakończeniu działania funkcji `setup()`.

❻ Funkcja `analogRead()` zwraca wartość pomiędzy `0` (poziom niski — `LOW`, 0 V) a `1023` (poziom wysoki — `HIGH`, +5 V). Ta wartość jest zapisywana w zmiennej `x`.

❼ Przesłanie wartości zmiennej `x` do portu szeregowego — ta wartość pojawi się w oknie monitora portu szeregowego. Funkcja `Serial.println()` przesyła każdą wartość w nowej linii — w efekcie w oknie monitora zobaczysz szereg przewijających się wartości.

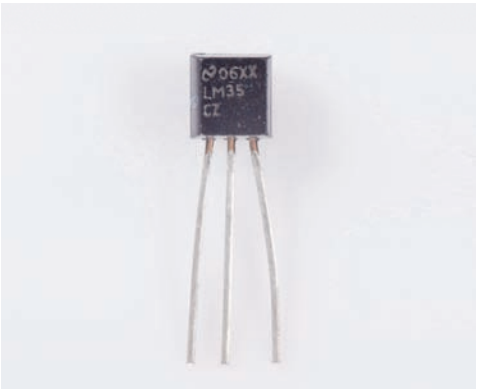
❽ Odczekanie 500 ms — zapewni Ci to wystarczającą ilość czasu, abyś mógł odczytać poszczególne wartości w oknie monitora portu szeregowego; zapobiega także zbyt dużemu obciążeniu procesora Arduino przez Twój program.

Poeksperymentuj: czy potrafisz podłączyć do układu potencjometr i odczytać jego wartość poprzez port szeregowy? Jeśli jesteś gotowy na kolejne wyzwanie, spróbuj przekonwertować zwracaną wartość na kąt obrotu potencjometru lub jako procent maksymalnego obrotu.

Czujesz się na siłach, aby spróbować czegoś więcej? Gdy już przekażesz dane do komputera, możesz je odczytywać za pomocą dowolnego języka programowania. W projekcie dotyczącym interaktywnego rysowania zawartym w książce *Make: Arduino Bots and Gadgets* skorzystaliśmy z biblioteki do obsługi portu szeregowego w Pythonie, aby odczytywać dane z Arduino i za pomocą gestów rysować obrazy na komputerze.

Projekt 11. Pomiar temperatury — czujnik LM35

Jest ciepło czy zimno? Czujnik LM35 jest niedrogim i łatwym w użyciu termometrem (rysunek 3.17). Jedyne, co trzeba zrobić, to zmierzyć napięcie za pomocą funkcji `analogRead()` oraz obliczyć temperaturę za pomocą wzoru: napięcie · 100 C/V.



Rysunek 3.17. Czujnik temperatury LM35

Nasi uczniowie wykorzystują czujnik LM35 np. do mierzenia temperatury w domkach letniskowych i szklarniach.

Elementy

Do tego projektu będziesz potrzebował następujących elementów:

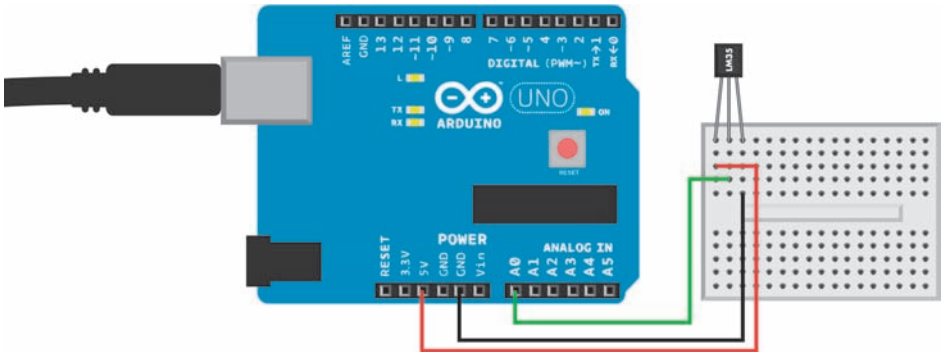
- czujnika temperatury LM35;
- Arduino Uno;
- kabelków połączeniowych;
- płytki stykowej.

Czujnik LM35 różni się nieco od analogowych czujników rezystancyjnych, z których korzystałeś wcześniej. Ma trzy wyprowadzenia: 5 V, masę oraz sygnał. Dzięki temu łatwiej jest go podłączyć, gdyż nie wymaga użycia dodatkowego rezystora podciągającego.

Budowa

Rysunek 3.18 przedstawia diagram układu. Zbuduj układ zgodnie z diagramem, a następnie uruchom kod z listingu 3.6. Aby zobaczyć aktualny odczyt temperatury, otwórz okno monitora portu szeregowego. Będziesz musiał wykonać następujące czynności:

1. Podłącz czujnik LM35 do Arduino za pomocą płytki stykowej. Jeśli masz najbardziej popularną wersję LM35 (w obudowie typu TO-92 — typowej dla wielu półprzewodników), ustaw czujnik tak, aby był zwrócony płaską stroną w Twoim kierunku i z wyprowadzeniami skierowanymi ku dołowi. Lewa nóżka to +5 V, środkowa to sygnał wyjściowy (`VOUT`), a prawa to wyprowadzenie masy (`GND`).



Rysunek 3.18. Czujnik temperatury podłączony za pomocą płytki stykowej

2. Podłącz wyprowadzenie +5 V do +5 V na płytce Arduino, a GND do GND na płytce Arduino. Ale czym jest sygnał VOUT? Otóż ten sygnał to napięcie wyjściowe informujące o temperaturze, którą rejestruje czujnik. Należy więc podłączyć je do wejścia analogowego na płytce Arduino (w listingu 3.6 jest to wejście A0).

Gdy już dokonasz pomiaru temperatury w swoim pokoju, możesz pokusić się o dalsze testowanie czujnika LM35. Gdybyś włożył czujnik do lodówki, musiałbyś odczekać dłuższą chwilę — powietrze jest całkiem dobrym izolatorem i nie przewodzi zbyt dobrze ciepła. W celu szybkiego schłodzenia czujnika posłuż się więc kostką lodu (rysunek 3.19). Wilgotne kostki lodu szybko odprowadzą ciepło z czujnika LM35 i otrzymasz natychmiastowy odczyt temperatury.

Rysunek 3.19. Testowanie czujnika LM35 za pomocą kostek lodu

Uruchom kod

Listing 3.6. Szkic Arduino korzystający z czujnika LM35

```
// temperature_lm35.ino – transmisja szeregowa temperatury z czujnika LM35
// (c) BotBook.com – Karvinen, Karvinen, Valtokari
int lmPin = A0; //❶
void setup() //❷
{
```

```
Serial.begin(9600); // ❸
}
float tempC() // ❹
{
float raw = analogRead(lmPin); // ❺
float percent = raw/1023.0; // ❻
float volts = percent*5.0; // ❼
return 100.0*volts; // ❽
}
void loop() // ❾
{
Serial.println(tempC()); // ❿
delay(200); // ms // ⓫
}
```

❶ Do odczytu sygnału wyjściowego czujnika wykorzystasz wejście analogowe 0 (A0).

❷ Podczas rozruchu Arduino wykonuje inicjalizację, jednokrotnie wywołując funkcję `setup()`.

❸ Otwarcie portu szeregowego. Korzystając z monitora portu szeregowego (*Narzędzia/Monitor portu szeregowego*), możesz zobaczyć, co dzieje się na wyjściu portu szeregowego. Pamiętaj, aby ustawić taką samą prędkość transmisji zarówno w kodzie, jak i w monitorze portu szeregowego.

❹ Definicja nowej funkcji o nazwie `tempC()`. Będziesz ją wywoływał w programie, aby odczytać temperaturę z czujnika.

❺ Funkcja `analogRead()` zwraca liczbę całkowitą z przedziału 0 – 1023. Wartość 0 jest równoznaczna z napięciem 0 V, a wartość 1023 oznacza +5 V.

❻ Dzieląc wynik odczytu przez wartość maksymalną, otrzymasz wartość odczytu wyrażoną procentowo. W ten sposób wartość `0.0` oznacza 0%, a wartość `1.0` oznacza 100%. Aby w wyniku dzielenia otrzymać wartość zmiennoprzecinkową (dziesiętną), należy jako dzielnika użyć także liczby zmiennoprzecinkowej `1023.0` zamiast liczby całkowitej `1023`. W językach programowania C i C++ wyrażenie 1/2 zwróci wartość `0`, natomiast wyrażenie 1/2.0 zwróci wartość `0.5`.

❼ Zmienna `percent` przedstawia wynik wyrażony procentowo w stosunku do wartości maksymalnej 5 V. Aby otrzymać napięcie wyjściowe w woltach, należy pomnożyć wynik procentowy razy 5.0.

❽ W danych katalogowych czujnika LM35 (wyszukaj w sieci frazy `LM35 datasheet`) można znaleźć informację, że współczynnik skalujący wynosi 10 mV/C, a napięcie 0 V oznacza temperaturę 0° C. Teraz trochę matematyki: 10 mV/C = 0.01 V/C. Odwrotność to 1/0.01 C/V = 100 C/V. Aby więc otrzymać temperaturę wyrażoną w stopniach Celsjusza, należy napięcie wyjściowe pomnożyć razy 100.

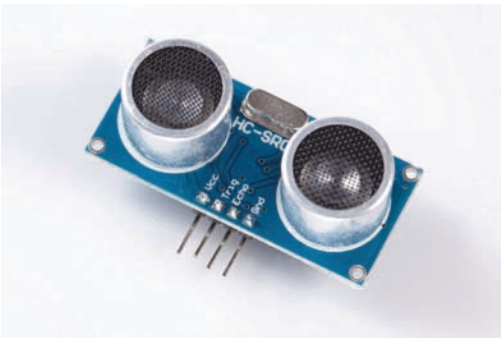
❾ Funkcja `setup()` kończy działanie i aż do odłączenia zasilania od Arduino uruchamiana jest w pętli funkcja `loop()`.

❿ Wywołanie funkcji `tempC()`, która zwraca liczbę (np. `20.0`). Ten wynik jest przesyłany do portu szeregowego i dodatkowo wstawia się znak końca linii. Funkcję `tempC()` można bardzo łatwo wykorzystać w innych projektach. Gdy masz już pewność, że działa ona poprawnie, możesz zapomnieć o szczegółach implementacyjnych. Gdy chcesz zmierzyć temperaturę, podłącz po prostu do swojego układu czujnik LM35 i wywołaj funkcję `tempC()`.

⓫ Odczekanie chwili. Zapewni Ci to wystarczającą ilość czasu, aby odczytać wynik, zabezpieczy bufor portu szeregowego przez przepełnieniem i zapobiegnie pożarciu przez Twój program 100% zasobów Arduino.

Projekt 12. Pomiar odległości za pomocą czujnika ultradźwiękowego HC-SR04

Ultradźwiękowy czujnik odległości (przedstawiony na rysunku 3.20) to jeden z naszych ulubionych czujników. Generuje on falę ultradźwiękową, a następnie rejestruje, po jakim czasie ta fala (echo) do niego powróci — można powiedzieć, że to taki mały sonar. Jak nazwa czujnika sama wskazuje, częstotliwość generowanej fali jest wystarczająco duża, by wykraczała poza pasmo słyszalne przez człowieka. Zaletą tego czujnika — w stosunku do czujnika na podczerwień, z którym zapoznałeś się wcześniej (projekt 7. „Wykrywanie przedmiotów za pomocą czujnika zbliżeniowego na podczerwień"), jest to, że potrafi on określić odległość od obiektu (istnieją także czujniki na podczerwień, które są zdolne do oceny odległości — np. z serii Sharp GP2Y0A).



Rysunek 3.20. Ultradźwiękowy czujnik odległości

Czujniki ultradźwiękowe mają jednak też pewną wadę — pewnego rodzaju obiektów nie potrafią wykryć w ogóle. Chodzi tutaj o wyjątkowo miękkie obiekty oraz obiekty składające się z nachylonych płaszczyzn. To właśnie z tego powodu samoloty niewykrywalne dla radarów mają takie a nie inne kształty.

Na rynku istnieje wiele ultradźwiękowych czujników odległości. Przez pewien czas naszym ulubionym był Parallax PING, jednakże jego koszt (ok. 150 zł) nie pozwalał na zakup zbyt wielu sztuk. W tej chwili można już kupić czujniki ultradźwiękowe w cenie kilkunastu złotych — HC-SR04 jest jednym z nich.

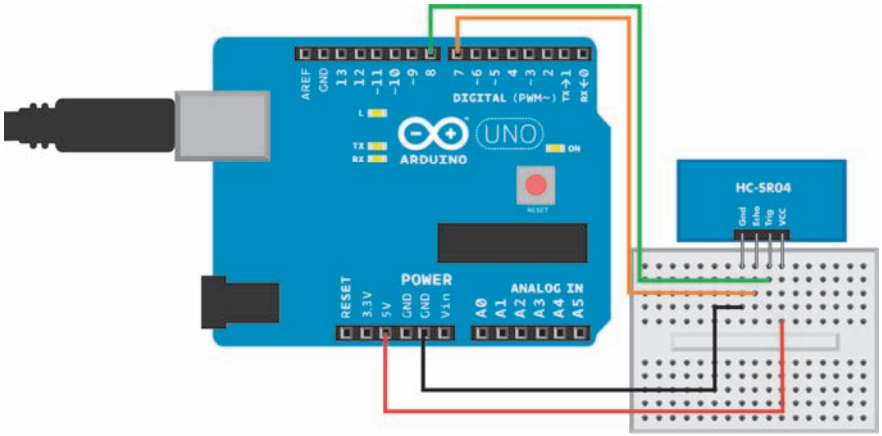
Elementy

Do tego projektu będziesz potrzebował następujących elementów:

- ultradźwiękowego czujnika odległości HC-SR04;
- Arduino Uno;
- płytki stykowej;
- kabelków połączeniowych.

Budowa

Połącz układ w sposób przedstawiony na rysunku 3.21 i uruchom kod z listingu 3.7.



Rysunek 3.21. Czujnik HC-SR04 podłączony za pomocą płytki stykowej

Uruchom kod

Listing 3.7. Szkic Arduino korzystający z czujnika HC-SR04

```
// hc-sr04.ino – pomiar odległości z wykorzystaniem czujnika HC-SR04
// (c) BotBook.com – Karvinen, Karvinen, Valtokari
int trigPin = 8; // ❶
int echoPin = 7; // ❷
float v=331.5+0.6*20; // m/s // ❸
void setup() // ❹
{
Serial.begin(9600); // ❺
pinMode(trigPin, OUTPUT); // ❻
pinMode(echoPin, INPUT); // ❼
}
float distanceCm(){ // ❽
// send sound pulse // ❾
digitalWrite(trigPin, LOW); // ❿
delayMicroseconds(3); // ⓫
digitalWrite(trigPin, HIGH); // ⓬
delayMicroseconds(5); // ⓭
digitalWrite(trigPin, LOW); // ⓮
// listen for echo // ⓯
float tUs = pulseIn(echoPin, HIGH); // mikrosekundy // ⓰
float t = tUs / 1000.0 / 1000.0 / 2.0; // s // ⓱
float d = t*v; // m // ⓲
return d*100; // cm // ⓳
}
void loop() // ⓴
{
int d=distanceCm(); // ㉑
Serial.println(d, DEC); // ㉒
delay(200); // ms // ㉓
}
```

❶ Wyprowadzenie oznaczone jako „Trig” na płytce czujnika HC-SR04 służy do wyzwalania sygnału ultradźwiękowego. Jest ono podłączone do wyprowadzenia cyfrowego `D8` na płytce Arduino i przy zmianie stanu na wysoki (`HIGH`) nastąpi wygenerowanie sygnału.

❷ Na wyprowadzeniu oznaczonym jako „Echo” na płytce czujnika HC-SR04 pojawi się stan wysoki (`HIGH`), gdy powróci do niego echo sygnału. Na płytce Arduino jest ono podłączone do wyprowadzenia cyfrowego `D7`.

❸ Prędkość dźwięku to 340 m/s. Tutaj została ona wyznaczona dla temperatury powietrza równej 20° C (do 331,5 dodajemy 20 · 0,6). Dla zachowania przejrzystości zawsze przedstawiamy takie obliczenia w kodzie. Nie stanowi to dla Arduino dodatkowego obciążenia, ponieważ kompilator przed przesłaniem programu do Arduino zamienia to wyrażenie na stałą.

❹ Funkcja `setup()` jest wywoływana automatycznie podczas rozruchu Arduino.

❺ Otwarcie połączenia szeregowego, aby można było obserwować wynik poprzez monitor portu szeregowego (*Narzędzia/Monitor portu szeregowego*). Pamiętaj, aby ustawić taką samą prędkość transmisji (wyrażoną w b/s lub bodach) zarówno w kodzie, jak i w monitorze portu szeregowego.

❻ Skonfigurowanie wyprowadzenia `trigPin` jako wyjścia (`OUTPUT`), aby można było za pomocą funkcji `digitalWrite()` ustawiać na nim stan wysoki (`HIGH`) lub niski (`LOW`).

❼ Skonfigurowanie wyprowadzenia `echoPin` jako wejścia (`INPUT`), aby można było za pomocą funkcji `digitalRead()` sprawdzać, czy występuje na nim stan niski (`LOW`), czy wysoki (`HIGH`).

❽ Zdefiniowanie nowej funkcji. Funkcja `distanceCm()` zwraca wartość zmiennoprzecinkową typu `float`. Nie przyjmuje ona żadnych parametrów, więc miejsce pomiędzy nawiasami po nazwie funkcji jest puste.

❾ Funkcja składa się z dwóch części. Najpierw informujemy czujnik HC-SR04, aby wygenerował sygnał ultradźwiękowy, a później sprawdzamy, jaki czas upłynął do momentu odebrania echa sygnału.

❿ Aby czujnik wygenerował sygnał, najpierw musimy ustawić na wyjściu stan niski (`LOW`). Muzyka zawsze zaczyna się od ciszy.

⓫ Odczekanie chwili — aby stan na wyjściu zdążył się ustawić. Pracując z krotnościami jednostek miary, warto wyrazić je w jednostkach podstawowych — tak aby mieć poczucie skali. Trzy mikrosekundy to trzy milionowe części sekundy lub liczbowo 0,000003 s.

⓬ Ustawienie na wyprowadzeniu wyzwalającym stanu wysokiego (`HIGH`, +5 V). W tym momencie następuje transmisja sygnału ultradźwiękowego.

⓭ Kolejne odczekanie chwili — 5 μs (mikrosekund). Pięć mikrosekund == pięć milionowych części sekundy == 0,000005 s. Przez ten czas na wyjściu będzie się utrzymywał stan wysoki.

⓮ Ustawienie na wyprowadzeniu wyzwalającym stanu niskiego `LOW`, kończącego krótki sygnał ultradźwiękowy.

⓯ Komentarz informujący, że właśnie następuje przejście do drugiej części funkcji.

⓰ Teraz trzeba ustalić, po jakim czasie na wyprowadzeniu `D7` pojawi się stan niski (`LOW`). Czas zwracany przez funkcję `pulseIn()` jest wyrażony w mikrosekundach. W kodzie najlepiej jest używać prostych znaków ASCII, więc przedrostek mikro zamiast greckiej literki μ oznacza się literką u lub U.

⓱ Korzystając z dzielenia, zamień mikrosekundy na sekundy. Często bardziej praktyczne jest operowanie na jednostkach podstawowych układu SI — takich jak: sekunda, metr, kilogram. Aby uniknąć przedwczesnego zaokrąglenia i w wyniku otrzymać liczbę dziesiętną, użyj jako dzielnika liczby zmiennoprzecinkowej (1000.0 zamiast liczby całkowitej 1000). Czas tUs to czas od wysłania sygnału do powrotu odbitego sygnału (w obie strony), więc trzeba go podzielić na pół, aby otrzymać czas od wysłania sygnału do chwili zetknięcia z przeszkodą.

18. Aby na podstawie czasu otrzymać dystans, należy pomnożyć go przez prędkość. Na przykład: jeśli podróż z prędkością 100 km/h do Krakowa i z powrotem zajęłaby 2 godziny, to odległość oblicza się jako 2h/2 · 100 km/h — czyli 100 km.

19. Aby otrzymać odległość w centymetrach, trzeba pomnożyć liczbę metrów przez 100. W przypadku czujnika HC-SR04 jest to wygodniejsza jednostka.

20. Po zakończeniu funkcji `setup()`, automatycznie jest uruchamiana w pętli funkcja `loop()`. To właśnie wywoływanie funkcji `loop()` jest głównym zadaniem Arduino.

21. To jest właśnie serce programu. Ponieważ funkcja `distanceCm()` została zdefiniowana osobno, nic nie stoi na przeszkodzie, abyś skorzystał z niej w innych projektach. Mimo że napisanie jej wymagało trochę wysiłku, to używając jej, nie musisz się już zagłębiać w szczegóły implementacji. Aby zmierzyć odległość, wystarczy podłączyć czujnik HC-SR04, dołączyć do kodu funkcję `distanceCm()` i wywołać ją. W tym przypadku rzutujesz (konwertujesz) wynik funkcji na liczbę całkowitą — ignorując część dziesiętną. Tak więc w zmiennej `d` będzie zapisywana odległość w pełnych centymetrach (np. 23).

22. Przesłanie odległości do monitora portu szeregowego (*Narzędzia/Monitor portu szeregowego*). Funkcja `println()` formatuje wartość do postaci liczby dziesiętnej i dodaje znak końca linii.

23. Aby zapobiec zbyt dużemu obciążeniu procesora Arduino przez pętlę nieskończoną, wprowadzamy małe opóźnienie. Zapewni Ci to także wystarczającą ilość czasu, aby odczytać wynik z monitora portu szeregowego — ponieważ kolejne wartości będą się przewijały przez okno wolniej.

Poeksperymentuj! Czy uda Ci się ukryć przed ultradźwiękami? Jakie powierzchnie odbijają dźwięk — dzięki czemu czujnik działa poprawnie, a jakie go absorbują lub przekierowują w zupełnie inną stronę, stając się niewidoczne dla sonaru?

A może spróbujesz zrobić coś bardziej skomplikowanego? Zmodyfikuj funkcję `distanceCm()` tak, aby przyjmowała jeden parametr, którym będzie numer wyprowadzenia cyfrowego (np. `distanceCm(3)` w przypadku czujnika HC-SR04 podłączonego do wyprowadzenia cyfrowego `3`). Tym sposobem będziesz mógł podłączyć do swojego Arduino kilka czujników HC-SR04 jednocześnie.

Czujniki ultradźwiękowe typu PING, HC-SR04 i podobne są używane w wielu projektach. Skorzystaliśmy z nich np. w projekcie interaktywnego rysowania czy też chodzącego robota insekta. Nasi uczniowie użyli ich przy budowie robotów, myszki obsługiwanej gestami, alarmów włamaniowych czy też interaktywnych pluszaków.

Podsumowanie

Nauczyłeś się pisać programy na Arduino, które pozwalają na dokonywanie pomiarów w otaczającym Cię świecie. Twoje Arduino potrafi mierzyć nacisk, temperaturę, światło, a także inne wielkości. Jeśli chcesz zdobyć większą wiedzę na temat tego układu, budować roboty oraz inne urządzenia, zajrzyj do książki *Make: Arduino Bots and Gadgets*.

Arduino jest układem prostym i zarazem posiadającym duże możliwości, więc zawsze chętnie z niego korzystamy podczas budowy układów prototypowych. Być może jesteś zainteresowany poznaniem innej platformy, na której można uruchamiać znacznie poważniejsze systemy (takie jak np. serwery sieciowe), czy też takiej, do której można podłączać większe urządzenia — np. projektory lub telewizory? W następnym rozdziale wypłyniesz na głęboką wodę i poznasz komputer Raspberry Pi.

Skorowidz

R

S

Ś

T

U

W

Z

Zbuduj wymarzony układ, reagujący na dane ze środowiska zewnętrznego!

Arduino oraz Raspberry Pi to płytki, które sprawiły, że świat elektroniki stał się dostępny dla wszystkich. Z ich pomocą każdy amator może sprawnie zrealizować projekt, o którym marzył od zawsze. Fantastyczne możliwości oraz łatwość, z jaką można je wykorzystać, przyczyniły się do ogromnej popularności tych dwóch platform. Jeżeli jednak chcesz zbudować bardziej wyrafinowany układ, będziesz potrzebować informacji o świecie zewnętrznym. Dostarczą Ci ich czujniki!

Jeżeli chcesz zorientować się, jak szeroki jest wybór czujników dla Raspberry Pi oraz Arduino, trafiłeś na doskonałą książkę. Znajdziesz w niej bogato ilustrowane opisy zastosowania przeróżnych sensorów. Pomiar obrotu, jasności światła, temperatury oraz odległości to tylko niektóre z opcji. Z kolejnych rozdziałów dowiesz się, jak wykorzystać przycisk monostabilny oraz zbudować urządzenie regulowane czujnikiem podczerwieni. Książka ta jest znakomitym źródłem informacji dla pasjonatów chcących tworzyć zaawansowane projekty z użyciem dostępnych na rynku czujników.

Dzięki tej książce:

- wykorzystasz czujnik temperatury
- zareagujesz na zmiany w oświetleniu
- zastosujesz czujnik FlexiForce
- stworzysz jeszcze bardziej zaawansowany układ elektroniczny

Kimmo Karvinen — magister sztuki. Dyrektor do spraw technicznych w firmie wytwarzającej osprzęt dla inteligentnych budynków. Wcześniej zajmował stanowisko dyrektora kreatywnego oraz specjalizował się w komunikacji marketingowej.

Tero Karvinen — magister ekonomii. Wykładowca na Haaga-Helia University of Applied Sciences. Prowadzi zajęcia związane z systemem Linux oraz sieciami bezprzewodowymi. Bada sieci bezprzewodowe w Helsinkach i okolicy. Wcześniej zajmował stanowisko CEO w małej firmie.

Helion
30814 numer katalogowy
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900
Informatyka w najlepszym wydaniu

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-283-0365-2
9 788328 303652
cena: 34,90 zł

Make:
makezine.com