ZUT SKN WiWav
Warsztaty Arduino

Rafał Górczewski
Joanna Lewandowska
Dawid Bindas
Mateusz Moszczyński
Kinga Walawicz
Izabella Hermanowicz
Przemysław Śnihur

Spis treści

Wprowadzenie

Ze względu na to, że na tych warsztatach będziemy mieli styczność z zarówno analogowymi, jak i cyfrowymi elementami elektronicznymi, warto przypomnieć sobie podstawy elektrotechniki oraz elementarną nomenklaturę panującą w tej dziedzinie.

Prąd elektryczny

Potencjał

- Potencjał elektryczny w punkcie oznacza zdolność ładunku do przemieszczenia się z jednego miejsca w inne.

Napięcie (różnica potencjałów)

- Jednostką jest **wolt (volt) [V]**, a oznacza się **U**.
- Kiedy w dwóch różnych punktach układu występuje różny potencjał, mówi się wtedy o jego różnicy, a zatem napięciu. Powoduje to przepływ ładunków między tymi punktami tak, jakby chciały tę różnicę wyrównać. Jeżeli w jednym miejscu potencjał wynosi -3V, a w drugim 3V, to napięcie między nimi będzie równe 6V. Mówi się zawsze o **napięciu między punktami** albo **napięciu na elemencie** (co oznacza napięcie element- masą).

Masa

- Jest to powszechnie używana nazwa dla miejsca, w którym następuje wyrównanie potencjałów.
- Najczęściej można spotkać parę: napięcie zasilania, które powoduje przepływ prądu i masa, do której ten prąd “spływa”.
- W przypadku Arduino mamy do wyboru dwa napięcia - 3,3V oraz 5V i **kilka miejsc masy, które można traktować jednakowo**

Natężenie prądu (prąd elektryczny)

- Jednostką jest **amper [A]**, oznacza się **I**.
- Oznacza “prędkość” z jaką prąd płynie (ile ładunków płynie przez przewodnik w jednostce czasu). Mówi się o **natężeniu w jakimś punkcie**, nie między punktami.

Rezystancja (opór)

- Jednostką jest **om (ohm) [Ω]**, oznacza się **R**.
- Jest to współczynnik regulujący przepływ prądu w obwodzie. Im większy opór, tym ładunkom “trudniej” jest się po nim przemieszczać.

Wybrane komponenty elektroniczne

Rezystor

- Rezystor to podstawowy elektroniczny element bierny, którego główną cechą jest (jak sama nazwa wskazuje) posiadanie rezystancji.
- Służy do regulacji napięcia/natężenia prądu w obwodzie w celu zapewnienia jego prawidłowego działania.
- Ze względu na jego mały rozmiar do odczytania wartości rezystancji oraz tolerancji (zakresu możliwych odchyleń od wartości nominalnej) wykorzystuje się tzw. kod paskowy rezystorów

Cewka

- Inaczej zwojnica lub induktor jest to bierny element układu elektrycznego, wytwarzający pole magnetyczne.
- Cewki idealnie nadają się jako elementy filtrujące zakłócenia w obwodzie.
- Podstawowym parametrem opisującym cewkę jest indukcyjność, czyli stosunek wytwarzanego pola magnetycznego do prądu przechodzącego przez cewkę.

Kondensator

- Stanowi układ co najmniej dwóch elektrod wykonanych z przewodnika oddzielonych od siebie dielektrykiem.
- Służy do gromadzenia ładunku elektrycznego.

- Jeżeli kondensator jest naładowany to w momencie odłączenia napięcia doprowadzonego do obwodu ładunki elektryczne w nim zgromadzone utrzymują się na jego elektrodach.

Dioda prostownicza

- Własnością diod (nie tylko elektroluminescencyjnych) jest **umożliwianie przepływu prądu tylko w jednym kierunku** - od anody do katody, dlatego ważne jest odpowiednie włączenie ich do obwodu, tj. anodą od strony “plusa”, a katodą od strony “minusa”.
- Wykorzystuje się ją np. w układach prostujących (zmieniających prąd zmienny na prąd stały).

LED (DEL, dioda elektroluminescencyjna)

- Jest to element, który charakteryzuje się emisją światła widzialnego pod wpływem prądu elektrycznego.
- Należy pamiętać, żeby razem z diodą świecąca podłączyć **szeregowo rezystor**, aby dioda nie uległa zniszczeniu.

Potencjometr

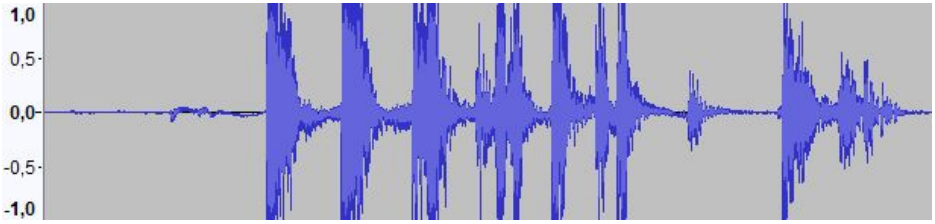
- Jest to, najprościej mówiąc, rezystor o nastawnej rezystancji, którą można zmieniać poprzez np. kręcenie gałką lub przesuwanie suwaka.

Sygnały cyfrowe i analogowe

W elektrotechnice rozróżnia się dwie elementarne grupy sygnałów - sygnały analogowe oraz sygnały cyfrowe. Arduino UNO pozwala nam na obsługiwanie obydwu tych grup, jednakże w przypadku sygnałów analogowych można je jedynie odbierać (stąd wejście analogowe, a nie wejście/wyjście analogowe).
W elektronice sygnałem są zazwyczaj zmieniające się wartości napięć, które wpływają na działanie układu.

Sygnały analogowe

- Są to sygnały, które **mogą przyjąć jedną z nieskończenie wielu wartości**.
- Sygnałem analogowym jest np. sygnał elektryczny powstający na wyjściu prostego mikrofonu, do którego ktoś mówi. W wyniku poruszania się jego membrany, zmienia się wartość napięcia, która jest proporcjonalna do jej wychylenia, zatem do głośności dźwięku. Napięcie to może przyjąć dowolną wartość z zakresu np. 0V-5V, czyli przykładowo 0.4V, 3.333333V, 0.01V itp.

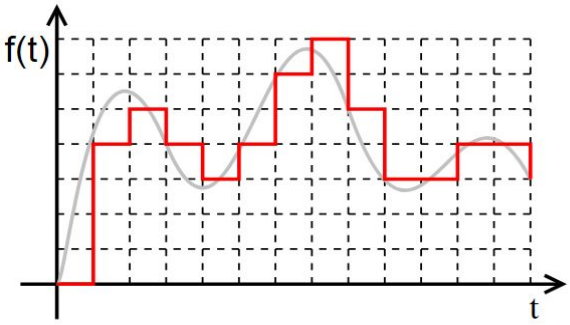


Sygnały cyfrowe

- Charakteryzują się tym, że ich wartości mogą przyjąć tylko konkretne, z góry ustalone wartości.
- Sygnałem cyfrowym jest np. sygnał elektryczny, gdzie napięcie przyjmuje wartości tylko 1V i 5V.

Jako, że elektronika polega na odpowiednim kształtowaniu napięć i natężeń w obwodzie, to w klasycznym tego ujęciu robi się to w sposób

analogowy. W elektronice cyfrowej operuje się głównie na wartościach cyfrowych, które można przypisać do dwóch stanów.



Źródło: https://upload.wikimedia.org/wikipedia/commons/9/9a/Digital.signal.svg

Logika dwuwartościowa

W układach cyfrowych głównie występują dwie wartości sygnału - 0 i 1. Oznacza to, że albo sygnał jest w stanie 1 (logiczna prawda) lub 0 (logiczny fałsz).
Jednak skoro napięcie może przyjmować nieskończenie wiele wartości, to jak stwierdzić w jakim stanie jest sygnał? To proste. Dla obydwu tych stanów przeznaczony jest **pewien zakres napięć**.

- Stan niski: stan niski sygnału odpowiada binarnemu 0 i logicznemu fałszowi.
  Napięcie oznaczające stan niski mieści się zazwyczaj w zakresie 0-0,8V.
- Stan wysoki: stan wysoki sygnału odpowiada binarnemu 1 i logicznej prawdzie.
  Napięcie oznaczające stan wysoki mieści się zazwyczaj w zakresie 2,4-5V.

Interakcja z Arduino

W projektach opartych o Arduino najczęściej korzysta się z wejść/wyjść cyfrowych, ponieważ wiele modułów wytwarza i odbiera właśnie taki typ sygnału (jednak nie jest to regułą).
Przykładowo czujnik wstrząsów po uprzednim potrząśnięciu ustawi na swoim wyjściu sygnał stanu wysokiego, który będzie oznaczał wykrycie

wstrząsu przez moduł. Odczyt wartości z takiego modułu będzie cyfrowy, ponieważ mogą zaistnieć tylko dwa stany - wysoki lub niski.
Czasami jednak istnieje potrzeba użycia wejść analogowych w Arduino, np. gdy korzysta się z modułów posiadających analogowe wyjście, takich jak czujnik natężenia światła. Wówczas odczyt wartości z takiego modułu jest analogowy, ponieważ jest możliwy odczyt pewnego zakresu napięć o rozdzielczości zgodnej z przetwornikiem A/D (analogowo cyfrowym).

Świat Arduino

Arduino jest otwartą platformą elektroniczną, gdzie jej twórcy za cel postawili sobie prostotę i łatwość obsługi. Samych platform z serii Arduino jest kilkanaście, jednak na potrzeby warsztatów wybraliśmy Arduino UNO, ze względu na przystępną cenę i szeroki wachlarz modułów oraz materiałów pomocniczych w sieci.

Słowniczek pojęć

Moduł bazowy

Jest to płytka, której sercem jest reprogramowalny mikrokontroler, potocznie nazywana “Ardu”. Istnieje kilkanaście rodzajów modułów bazowych Arduino, a w tych warsztatach posługiwać się będziemy wersją UNO.

Moduł

Jako moduł przeważnie określa się pełniący wyspecjalizowaną funkcję układ elektroniczny zawarty na odrębnej płytce drukowanej.

Port szeregowy

Arduino podłączone do komputera widziane jest w systemie jako port szeregowy COM, za pomocą którego możliwe jest zaprogramowanie mikrokontrolera. Port ten pełni też funkcję komunikacyjną.

Bootloader

Bootloader służy wgrywaniu wygenerowanego programu z pominięciem programatora, dzięki czemu możemy bezpośrednio po

skompilowaniu programu wgrać go na Arduino. Takie rozwiązanie jest coraz częściej stosowane (każde arduino posiada bootloader)

Sketch

Jest to powszechnie używana nazwa, która oznacza program napisany pod Arduino.

Zworki

Zworki to drobne “złączki” pozwalające na łatwe zwieranie dwóch lub więcej pinów, poprzez ich nałożenie.

Pin

Pinami (nóżkami, potocznie również “goldpinami” ze względu na kolor) nazywamy metalowe, wystające lub wchodzące w płytkę wyprowadzenia, które pozwalają na wygodne podłączenie ścieżek do płytki prototypowej.

Przewód męski/żeński

Ponieważ zarówno wyprowadzenia Arduino jak i modułów są realizowane za pomocą pinów męskich i żeńskich, niezbędne jest używanie odpowiednich przewodów zakończonych właściwą wtyczką lub gniazdkiem.

Typowe oznaczenia

VCC, GND

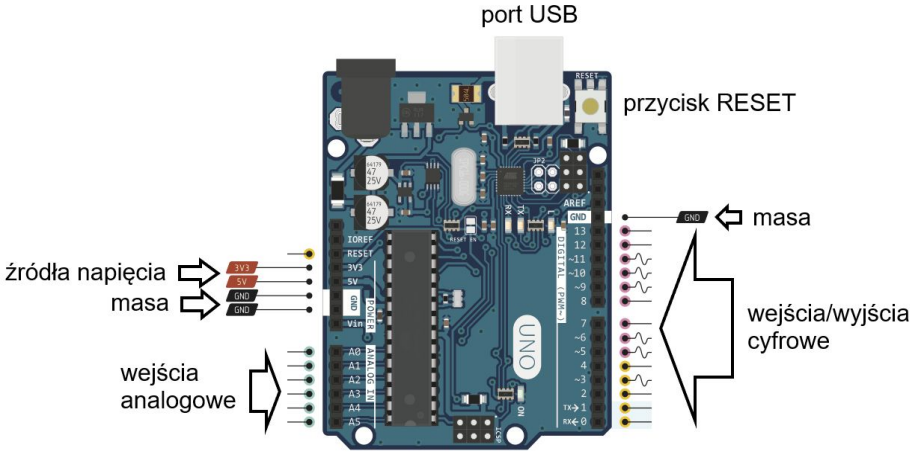
- VCC - oznacza pin, do którego należy podpiąć źródło zasilania. Należy wpierw zapoznać się z **dopuszczalnym zakresem napięć** dla danego modułu, żeby przypadkowo go nie zniszczyć.
- GND - oznacza pin masy

Bardzo ważne jest aby bezpośrednio nie podłączyć omyłkowo złączy VCC i GND, gdyż grozi to **bezpowrotnym uszkodzeniem modułu**.

IN, OUT, D, A

- IN - zazwyczaj oznacza wyprowadzenie układu dla sygnału wejściowego np. pin/port wejściowy.
- OUT - zazwyczaj oznacza wyprowadzenie układu dla sygnału wyjściowego np. pin/port wyjściowy.
- D[0..n] - przeważnie oznaczenie pinu cyfrowego (ang. digital pin) np. D0, D1.
- A[0..n] - przeważnie oznaczenie pinu analogowego (ang. analog pin) np. A0, A1.

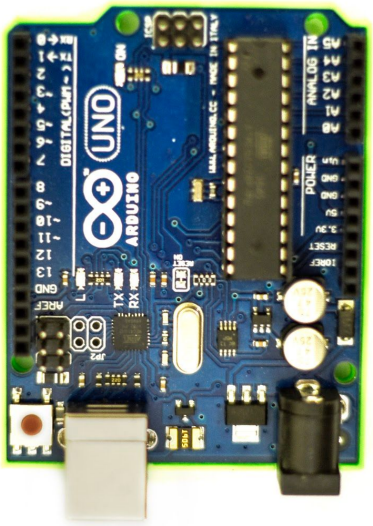
Budowa Arduino UNO



Źródło: http://forum.arduino.cc/index.php?topic=146315.0

Arduino UNO składa się z sekcji zasilania, programatora, oraz mikrokontrolera, który jest sercem całego układu. Zazwyczaj w Arduino UNO instalowany jest mikrokontroler Atmel ATmega328 o 32kB pamięci programu i 1kB pamięci trwałej EEPROM. Mikrokontroler ten posiada także sześcio-pinowy port analogowo-cyfrowy o 10-bitowej rozdzielczości przetwornika A/C.

Do komunikacji z komputerem i programowania mikrokontrolera służy umieszczony na płytce port USB. Jest on również źródłem zasilania płytki na czas etapu programowania i testów niewielkich projektów.



Gdyby zaistniała potrzeba podłączenia do Arduino modułów mających większe zapotrzebowanie na prąd, Arduino wyposażone jest w sekcję zasilania ze stabilizatorem napięcia i gniazdem DC (ang. direct current - prąd stały) za pomocą którego można zasilić płytkę np. baterią 9V lub zasilaczem prądu stałego.

Obok portu USB na płytce znajduje się przycisk RESET, służący do ponownego uruchomienia naszego programu znajdującego się w pamięci mikrokontrolera.

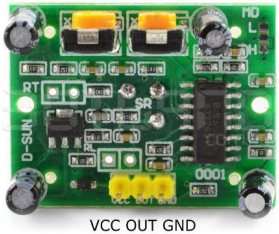
Na obu dłuższych krawędziach płytki znajdują się listwy kołkowe typu żeńskiego, na których zostały wyprowadzone:

- cyfrowe porty wejścia/wyjścia, mogące obsługiwać sygnały cyfrowe pochodzące z modułów oraz wyprowadzać własne sygnały na inne moduły;
- wejścia analogowe - służące do odbierania sygnałów analogowych;
- dodatkowe linie zasilania (źródła napięcia i masy).

Dodatkowo na płytce znajdują się dwa złącza magistrali szeregowej ISP umożliwiającej komunikację z niektórymi modułami zewnętrznymi.

Moduły i ich wykorzystanie z Arduino

Do Arduino stworzono wiele modułów elektronicznych, które pozwalają na szybką i prostą budowę relatywnie zaawansowanych projektów. Moduły mają swoją dedykowaną funkcjonalność, którą można wykorzystać do własnych celów. W zależności od ich cech moduły znacznie różnią się swoją budową i wyprowadzeniami, dlatego warto mieć pod ręką dokumentację stworzoną przez producenta danego modułu.



Źródło: https://botland.com.pl/czujniki-ruchu/1655-czujnik-ruchu-pir-hc-sr501-zielony.html

Na powyższym zdjęciu znajduje się przykładowy moduł czujnika ruchu PIR HC-SR501. Jak widać składa się on z bardzo wielu elementów elektronicznych, którymi na szczęście nie musimy sobie zawracać głowy, dopóki nie zainteresuje nas jak działa on od środka.

Większość modułów ma opisane złącza zasilania (VCC i GND) oraz złącza komunikacyjne. Przywołany tutaj moduł posiada tylko jeden pin

komunikacyjny (OUT), lecz inne mogą posiadać dużo bardziej skomplikowane wyprowadzenia i oznaczenia.

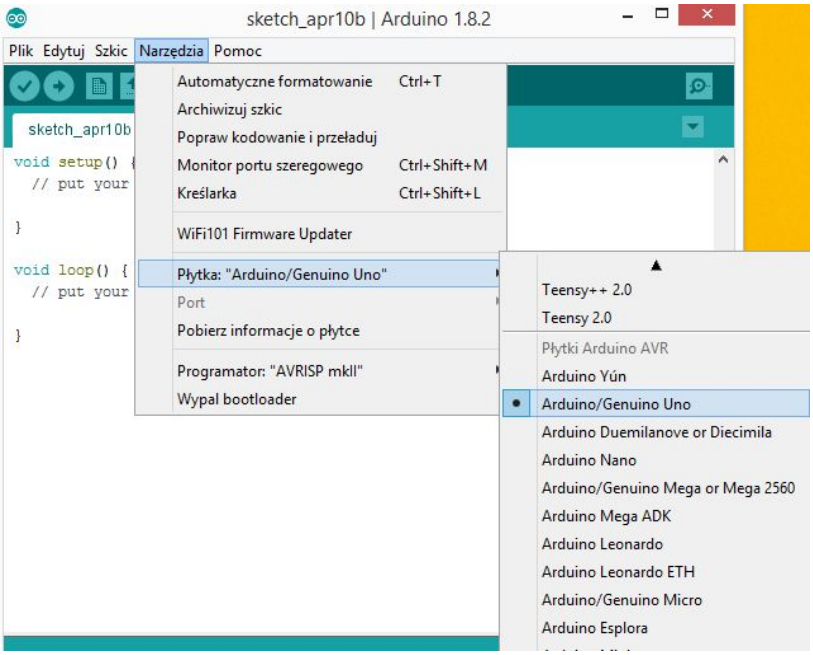
Moduł tego konkretnie czujnika obsługuje się w bardzo prosty sposób: po podłączeniu do zasilania i wykryciu ruchu, na wyjściu (OUT) ustala się stan wysoki, który taki pozostaje przez okres czasu, dopóki z pola widoczności nie zniknie ruch.

Dzięki temu że moduły mają swoją odrębną funkcjonalność, podczas realizacji własnego projektu należy skupić się jedynie na obsłudze komunikatów lub bezpośrednich wartości jakie moduły podają na swoje wyjścia. Oszczędza to czas i wysiłek włożony w pracę, stąd przygoda z Arduino jest łatwym i przyjemnym sposobem na wkroczenie w świat elektroniki cyfrowej i programowania.

Programowanie Arduino

Arduino bez wgranego programu określanego jako “sketch” jest tylko złożonym układem elektronicznym, który nie pełni żadnej funkcji. Sketche można pisać, kompilować oraz wgrywać do Arduino za pomocą wielu środowisk. Jednym z popularniejszych jest Arduino IDE. Jest to program open-source, który można pobrać ze strony https://www.arduino.cc.

Po uruchomieniu oprogramowania Arduino musimy przejść do zakładki Narzędzia, a następnie wybrać wersję Arduino, z której będziemy korzystać. W naszym przypadku będzie to Arduino UNO. Dodatkowo musimy wybrać port COM, pod który podłączona jest nasza płytka.



Domyślnie po uruchomieniu Arduino IDE pokazuje nam się nowy sketch , na którym będziemy mogli zacząć pisać nasz kod.

 - za pomocą tego przycisku uruchamiamy kompilację naszego kodu źródłowego

 - za pomocą tego przycisku wgrywamy program do pamięci Arduino. Jeśli wcześniej nie uruchomimy kompilacji, zostanie ona uruchomiona przed wgraniem programu.

Podstawowe elementy Sketchy

- **pinMode(pin, mode)** - służy do ustawienia odpowiedniego pinu w tryb wyjścia (OUTPUT) albo wejścia (INPUT), np.

```
pinMode(2, OUTPUT); // ustawia tryb pinu 2 na wyjście
```

- **digitalRead(pin)** - służy do odczytania wartości (HIGH/LOW) z danego pinu cyfrowego ustawionego w tryb wejścia (INPUT) np.

```
if (digitalRead(2) == HIGH) { // jeżeli na pinie 2 jest stan
                              // wysoki
    ...
}
```

- **analogRead(pin)** - zwraca wartość odczytaną z pinu analogowego, która mieści się w przedziale <0, 1023>, np.

```
int x = analogRead(2); // odczytaj wartość wejścia
                       // analogowego o numerze 2
```

- **digitalWrite(pin, value)** - podaje na wybrane wyjście cyfrowe wartość stanu wysokiego (HIGH) lub niskiego (LOW), np.

```
digitalWrite(2, HIGH); // ustawi wyjście pinu 2 w stan wysoki
```

- **delay(ms)** - wstrzymuje działanie programu na podaną liczbę milisekund, np.

```
delay(1000); // wstrzymuje działanie programu na sekundę
```

Pierwszy sketch

Pierwszym sketchem, który wykonamy na warsztatach będzie sterował diodą - na przemian zapalał i gasił.

W pierwszym kroku należy zdefiniować pin, z którego będziemy korzystali, aby sterować diodą.

```
#define LED_PIN 8;
```

Następnie za pomocą funkcji pinMode() należy określić typ naszego pinu. Biorąc pod uwagę fakt, że będziemy wysyłali sygnał z Arduino do diody należy określić go jako OUTPUT. Specyfikacja ta powinna znaleźć w funkcji setup() wywoływanej jednokrotnie podczas uruchomienia urządzenia.

```
void setup() {
  pinMode(LED_PIN, OUTPUT);
}
```

Funkcja loop() swoim działaniem przypomina pętlę nieskończoną - jest ciągle wykonywana. W niej należy zamieścić ciało naszego programu. Aby ustawić stan pinu cyfrowego określonego jako OUTPUT wykorzystuje się funkcję digitalWrite(). Przyjmuje ona dwa parametry: numer pinu oraz stan - LOW lub HIGH. Dodatkowo, aby być w stanie zaobserwować zmianę stanów diody wykorzystamy funkcję delay(), która przyjmuje jako parametr ilość milisekund, na które ma zatrzymać działanie programu.

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

```
  delay(1000);
}
```

Projekty do realizacji

Projekt dla początkujących

Opis projektu

Czy kiedykolwiek męczyło was ręczne odsłanianie i zasłanianie żaluzji? Jest na to sposób - wystarczy kilka modułów i płytka Arduino.
W naszym projekcie stworzymy prototyp automatycznego sterowania systemu rolet. Użyjemy do tego serwomechanizmu, który będzie obsługiwany za pomocą pilota na podczerwień.

Potrzebne elementy

Do realizacji projektu potrzebne będą:

- Pilot IR + odbiornik VS1838B
- Serwo TowerPro
- Przewody połączeniowe.

Opis modułów

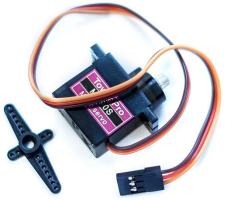
Pilot IR + odbiornik VS1838B

Pilot nadający w paśmie podczerwieni wraz z kompatybilnym odbiornikiem. Umożliwia on komunikację bezprzewodową z dowolnym mikrokontrolerem lub zestawem.

Piny przeznaczone do użycia prezentują się następująco:
VCC i GND - piny zasilania,
Out - pin wyjściowy.
Aby możliwe było użycie pilota, konieczne jest pobranie jego kodu.
My będziemy korzystać z przycisków “|<<” ,“>>|” oraz “>||” .

Serwo TowerPro

Serwomechanizm to silnik, przekładnia oraz dedykowany sterownik zamknięty w jednej obudowie. Napędy te nie są jednak przystosowane do wykonywania pełnego obrotu. Najczęściej serwomechanizmy mogą poruszać zamontowanym ramieniem o kąt 0-180º.



Z każdego serwomechanizmu wyprowadzone są 3 przewody:

1. Masa (czarny, ciemnobrązowy)
2. Zasilanie (czerwony)
3. Sygnał sterujący (żółty/pomarańczowy)

Realizacja

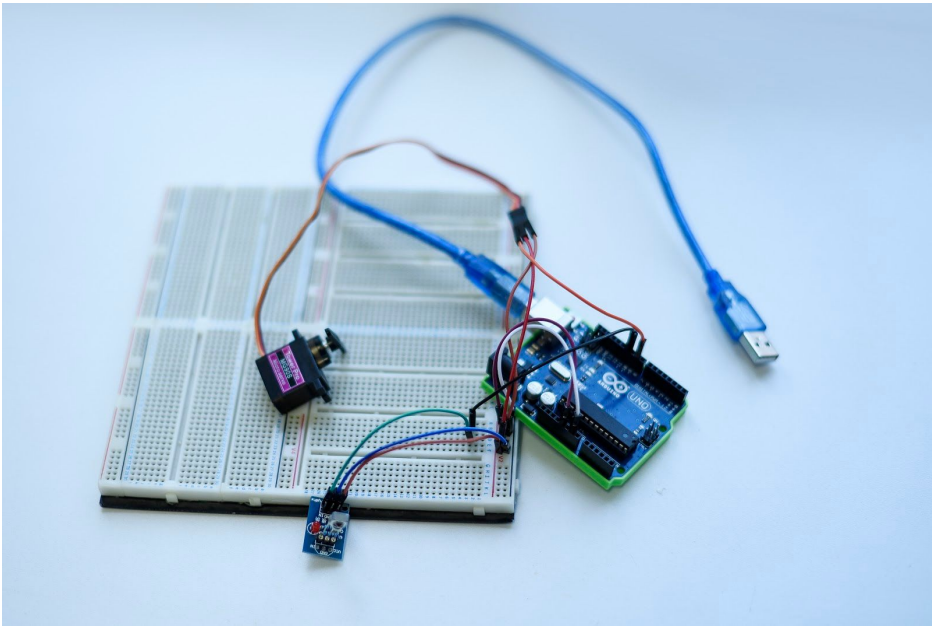
Na początek podłączmy odpowiednio moduły do płytki Arduino, z których będziemy korzystać, a następnie przygotujmy kod programu. Wszystkie moduły mają złącza VCC i GND, które należy odpowiednio podpiąć do pinów w Arduino.

Zacznijmy od podłączenia serwomechanizmu.

1) Sygnał sterujący (żółty/pomarańczowy) podepniemy do pinu numer 9 (na płytce Arduino)
2) Zasilanie(kabel środkowy) podepniemy do płytki stykowej oraz poprowadzimy kolejny przewód do płytki Arduino do pinu (GND)
3) Ostatni przewód(czarny, ciemnobrązowy) podepniemy również do płytki stykowej tylko teraz do “-”oraz poprowadzimy kolejny przewód do płytki arduino do pinu (VCC).

Kolejnym etapem jest podłączenie odbiornika.

1) Przewód po stronie diody LED podłączamy do pinu nr 11
2) Przewód środkowy podłączamy do “-”
3) Ostatni przewód podłączamy do “+”



Po podpięciu modułów, zajmijmy się pisaniem programu.

Na początku powinniśmy dołączyć bibliotekę służącą do obsługi pilota i serwomechanizmu, wpisując poniższe instrukcje na samym szczycie pliku:

```
#include <Servo.h>
#include <IRremote.h>
```

Następnie zadeklarujemy zmienną (obiekt) myservo, który posłuży nam do obsługi serwa, a następnie zdefiniujemy pin odpowiedzialny za obsługę pilota.

```
Servo myservo;          // obiekt myservo do obsługi serwo
int RECV_PIN = 11;      // pin używany przez odbiornik pilota
```

Deklaracja zmiennych pomocniczych:

```
int pos = 0; //początkowa pozycja serwo
int dir = 0;  // początkowy kierunek serwo
int key = storeCode(&results); //odczytanie kodu przycisku
                                //wciśniętego na pilocie
```

Ta część kodu pozwala nam na obsługę serwo za pomocą pilota.

```
 switch(key){

    case 8925:{          //w przypadku, kiedy został wciśnięty przycisk
                         //PREV
    dir=1;              // ustawiamy kierunek na prawo
    }break;

    case 765:{           //w przypadku, kiedy został wciśnięty przycisk
                        //NEXT
    dir=-1;             // ustawiamy kierunek na lewo
    }break;

    case -15811:{        //w przypadku, kiedy został wciśnięty przycisk
                         //PLAY/PAUSE
    dir=0;    // zatrzymanie serwa
    }break;
  }
```

Po wciśnięciu przycisku, którym odpowiada wartość:
-PREV (8925);
-NEXT (765);
-PLAY/PAUSE (-15811);
ustawiamy wartość zmiennej dir, która w kolejnych linijkach kodu umożliwi nam obracanie nakładką serwomechanizmu.

Wartość zmiennej pos określa kąt obrotu śmigła - w naszym przypadku są to wartości z zakresu 0-180. Gdy kąt osiągnie jedną z wartości granicznych, serwo przestaje się obracać.

```
if(dir<0){
  pos-=1;                  //odejmujemy od pos, które w tym momencie
                           //równa się 180

   if(pos==-1){             //jeśli pos równa się -1
   dir=0;
   pos=0;
   }

  myservo.write(pos);       //obraca serwo
  delay(10);                //z opóźnieniem, żeby nie było za szybkie
  }

  if(dir>0){               //w tym momencie nasze pos ma wartość 0
  pos+=1;                  //dodajemy do niego jeden

   if(pos==181){           //kiedy pos osiągnie 181
   dir=0;                  //zmieniamy wartość dir na 0, co spowoduje
                           //zatrzymanie mechanizmu
   pos=180;                //a nasze pos ustawiamy na 180
   }

  myservo.write(pos);
  delay(10);
  }
}
```

Projekt średniozaawansowany

Opis projektu

Kiedy znajdujemy się w potrzebie zmierzenia mijającego czasu, zwykły zegar nie zawsze wystarcza. Z pomocą przychodzi urządzenie zwane stoperem, którego budowa jest celem tego projektu.
Nasz stoper będzie umożliwiał wyświetlanie mierzonego okresu czasu, zatrzymywanie odmierzania oraz “pauzowanie”. Czas będzie wyświetlany w sposób graficzny, a wciśnięcie każdego z przycisków będzie potwierdzane sygnałem dźwiękowym pochodzącym z buzzera.

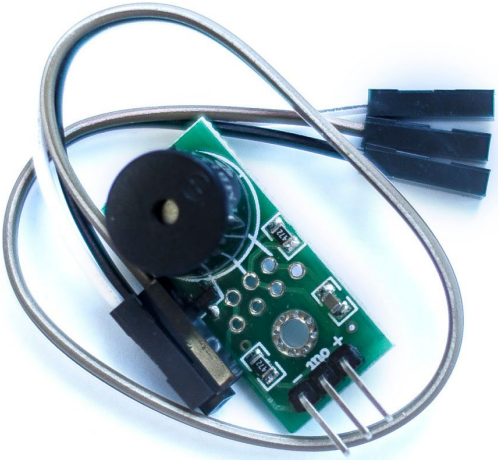
Potrzebne elementy

Do realizacji projektu potrzebne zatem będą:

- Klawiatura 4*4 z wyjściem analogowym - rezystancyjnym,
- Moduł wyświetlacza LED 4 cyfrowego 7seg. TM1637,
- Buzzer aktywny z generatorem,
- Przewody połączeniowe.

Opis modułów

Buzzer aktywny z generatorem

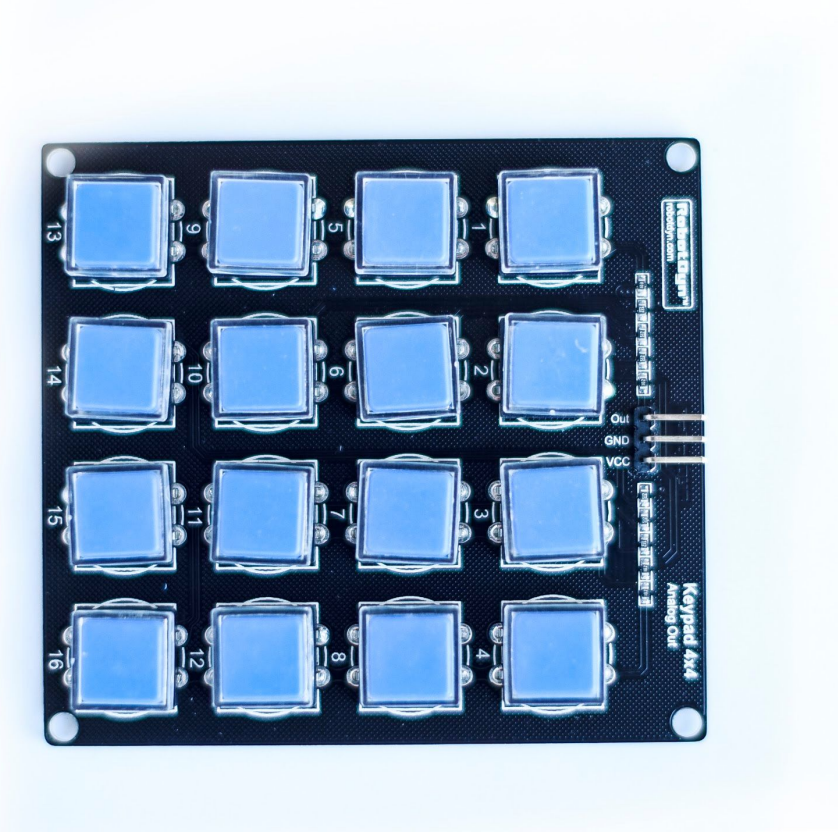


Moduł ten służy do wydawania prostego, charakterystycznego dźwięku o stałym tonie, którego użyjemy do sygnalizowania wciśnięcia przycisku. Jest aktywny, gdyż sygnał dźwiękowy nie jest dostarczany przez użytkownika w postaci analogowej, a jest on generowany na płytce po “włączeniu” go stanem niskim (umożliwieniu odpłynięcia prądu na wyjściu Out).

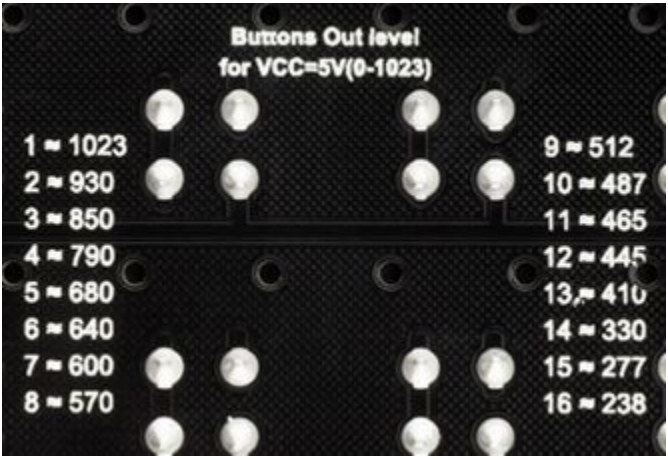
Piny przeznaczone do użycia prezentują się następująco:
VCC i GND - piny zasilania,
Out - pin wyjściowy.

Buzzer zaczyna wydawać dźwięk, kiedy na wyjściu (wejściu) modułu pojawia się **stan niski**, a przestaje po zmianie sygnału na stan wysoki.

Klawiatura analogowa



Ze względu na to, że klawiatura ma **wyjście analogowe**, należy je podpiąć do wejścia analogowego Arduino.
Klawiatura wydaje na wyjście sygnał o wartości napięcia charakterystycznej dla danego klawisza. Odczytując wartość tego sygnału w Arduino, musimy kierować się rozpiską stworzoną przez producentów tego modułu:



Po lewej stronie znaków = są podane numery (indeksy) klawiszy, które mogą być wciśnięte, zaś po prawej stronie są podane wartości odczytane przez Arduino z sygnału analogowego.
Przykład: wciskając klawisz oznaczony numerem 4, po odczytaniu wartości w Arduino, uzyskamy wartość w przybliżeniu równą 790.

Jest to możliwe dzięki temu, że sygnał analogowy o napięciu maksymalnym 5V jest dzielony na 1024 możliwe wartości cyfrowe, co pozwala na przypisanie określonym wartościom napięć jedną z tych wartości cyfrowych.

Wyświetlacz LED 7-segmentowy

Korzystanie z tego typu wyświetlacza jest nieco bardziej skomplikowane, gdyż jego jedyne wejście jest cyfrowe, a co za tym idzie może przyjąć jedynie stan niski i wysoki, zaś do obsłużenia mamy aż 4*7=28 segmentów LED. Jak zatem obsługuje się taki wyświetlacz? Rozwiązaniem jest komunikacja szeregowa. Wejście wyświetlacza zmienia swoje stany w określonej kolejności i w bardzo szybkim czasie, dzięki czemu możliwy jest wybór segmentów jedynie za pomocą jednego wejścia cyfrowego. Do prostej obsługi wyświetlacza posłuży nam biblioteka TM1637.

Realizacja

Zaczynając projekt warto najpierw odpowiednio podłączyć do Arduino moduły, z których będziemy korzystać, a następnie przygotować kod programu.
Wszystkie moduły mają piny VCC i GND, które należy odpowiednio podpiąć do pinów w Arduino.
Out (buzzer) - należy podpiąć do wyjścia cyfrowego Arduino,
CLK/DIO (wyświetlacz) - należy podpiąć do wyjść cyfrowych Arduino,
Out (klawiatura) - należy podpiąć do wejścia **analogowego** Arduino.

Po podpięciu modułów, zajmijmy się pisaniem programu.
Na początku powinniśmy dołączyć bibliotekę służącą do obsługi wyświetlacza i plik z funkcją do obsługi klawiatury, wpisując poniższe instrukcje na samym szczycie pliku:

```
#include "TM1637.h"
#include “Keyboard.h”
```

Pozwoli to na dalsze wykorzystanie funkcji i klas bibliotecznych znajdujących się w innych plikach.
Zdefiniujmy także piny, których użyliśmy do podłączenia modułów, np.:

```
#define BUZZER 3
#define KEYBOARD A2
#define DISPLAY_CLK 4
#define DISPLAY_DIO 5
```

Następnie musimy ustalić, które piny będą wyjściowe, a które wejściowe, oraz jaki będą miały początkowy stan. Początkowy stan układu ustala się w funkcji setup():

```
void setup() {
   pinMode(KEYBOARD, INPUT); //pin klawiatury w trybie wejścia
   pinMode(BUZZER, OUTPUT); //pin buzzera w trybie wyjścia

   digitalWrite(BUZZER, HIGH); //ustalenie stanu wysokiego na pinie
                               //buzzera
}
```

Pin klawiatury ustawiamy w tryb wejścia, gdyż to z niego będziemy wczytywać dane, zaś pin buzzera w tryb wyjścia, gdyż będziemy tam podawać sygnał wyjściowy.
Należy pamiętać, że ustawiamy wyjście buzzera w stan wysoki, gdyż stan niski spowoduje jego uruchomienie (nasz buzzer jest aktywowany stanem niskim).

Zastanówmy się zatem, w jakiej kolejności chcemy zaimplementować funkcjonalności naszego stopera - odmierzanie czasu po wciśnięciu przycisku, odtwarzanie dźwięku po wciśnięciu jakiegokolwiek przycisku, pauzowanie działającego stopera, zatrzymywanie odmierzania czasu.

Najbardziej trywialnym zadaniem wydaje się zatem odtwarzanie dźwięku, gdyż “nie obchodzi nas” który przycisk będzie wciśnięty.
Na początku warto stworzyć zmienną globalną, która będzie dostępna w ciele każdej z funkcji, a która będzie służyła do oznaczania, który przycisk jest aktualnie wciśnięty. Stwórzmy zatem zmienną całkowitoliczbową key, którą umieścimy pod instrukcjami #include oraz #define:

```
#include “...”
#define … ...
int key = 0;
```

Teraz możemy napisać prostą funkcję, która w przypadku wciśnięcia jakiegokolwiek klawisza ustawi stan pinu buzzera na niski, a w przypadku niewykrycia żadnego ustawi ten stan na wysoki:

```
void beepIfClicked(int button) {
   if (button != 0) { //jeżeli numer przycisku jest różny od zera
      digitalWrite(BUZZER, LOW); //ustaw pin buzzera w stan niski
   }                                          //tj. włącz buzzer
   else { //w przeciwnym wypadku
      digitalWrite(BUZZER, HIGH); //wyłącz buzzer (stan wysoki)
   }
}
```

Ciało tej funkcji umieścimy nad funkcjami setup() oraz loop(), zaś pod instrukcjami #include oraz #define i zmiennymi globalnymi, czyli:

```
#include "..."
#define … …

int key = 0;

void beepIfClicked(int button) {
…
}

void setup() {
…
}
void loop() {
…
}
```

Możemy teraz użyć naszej funkcji w programie. Jako że sprawdzanie przycisku jest wykonywane przez cały czas jego działania, z funkcji będziemy korzystać w funkcji loop().

Na początku musimy sprawdzać, jaki przycisk jest aktualnie wciśnięty. Skorzystamy z funkcji znajdującej się w pliku Keyboard.h, który dołączyliśmy na samym początku pisania. Wygląda ona następująco:

```
int getButtonClicked(int x);
```

Jako argument funkcji przekazujemy wartość x odczytaną z pinu wejścia analogowego, a w zamian dostajemy informację, który z pinów klawiatury został wciśnięty.
Brak wciśniętego przycisku oznacza liczba 0.

W funkcji loop() zatem najpierw przypiszemy do naszej globalnej zmiennej key wartość aktualnie wciśniętego klawisza, a następnie przekażemy ją do funkcji beepIfClicked, która spowoduje zabrzęczenie buzzera za każdym razem, gdy jakikolwiek przycisk zostanie wciśnięty:

```
void loop() {
   int analogKeyboardValue = analogRead(KEYBOARD);
//odczytujemy wartość analogowego pinu KEYBOARD

   key = getButtonClicked(analogKeyboardValue);
//przekazujemy ją do funkcji getButtonClicked, która zwraca numer
//wciśniętego klawisza

   beepIfClicked(key);
//brzęczenie buzzera gdy wciśnięty jest jakiś klawisz

}
```

Udało nam się zaimplementować sprawdzanie wciśniętego klawisza, oraz brzęczenie buzzera w przypadku kliknięcia.
Teraz zajmiemy się odmierzaniem czasu. Przydadzą nam się do tego dodatkowe cztery zmienne globalne:

```
int msCounter = 0; //licznik milisekund

bool stopwatchRunning = false; //flaga określająca, czy mierzenie
                               //czasu jest włączone

int digits[4] = {0, 0, 0, 0}; //tablica z czterema wartości, gdzie
                              //każda oznacza cyfrę wyświetlacza

TM1637 display{DISPLAY_CLK, DISPLAY_DIO};
//zmienna (obiekt) służąca do obsługi wyświetlacza
```

Wyświetlacz należy także odpowiednio przygotować w funkcji setup():

```
void setup() {
  ...
  display.init(); //przygotowanie wyświetlacza do działania
  display.set(BRIGHT_TYPICAL); //ustawienie jasności segmentów
}
```

W funkcji loop() dodajmy zatem proste instrukcje warunkowe, które zajmą się włączaniem i wyłączaniem odmierzania czasu:

```
void loop() {
  …
  if (key == 1) { //jeżeli wciśnięty przycisk 1
    stopwatchRunning = true; //odmierzanie czasu włączone
  }
  if (key == 2) { //jeżeli wciśnięty przycisk 2
    stopwatchRunning = false; //odmierzanie czasu wyłączone
  }
}
```

Przygotujmy zatem funkcję obsługującą odmierzanie czasu:

```
void handleStopwatch() {
  //tutaj będziemy wpisywać instrukcje
}
```

Będziemy zatrzymywać działanie programu na jedną milisekundę, jednocześnie zwiększając licznik msCounter o 1. Pozwoli to na płynniejsze działanie stopera, gdyż funkcja delay() całkowicie wstrzymuje działanie całego programu, co nie pozwala na odpowiednią obsługę sygnałów wejściowych np. klawiatury.

```
delay(1); //zatrzymanie programu na 1 milisekundę
msCounter++; //zwiększenie licznika milisekund o 1
```

Po osiągnięciu 150 licznika (1 pełna sekunda), zmienimy ostatnią cyfrę na kolejną, o jeden większą, ponieważ jest to cyfra jedności całej liczby sekund, a liczbę milisekund zerujemy.

```
if (msCounter > 150) { //jeżeli wartość licznika przekroczy 150
    digits[3]++; //zwiększenie ostatniej cyfry o 1 (indeks 3, gdyż tablice
                 //numerujemy od 0)
    msCounter = 0; //zerowanie licznika milisekund
}
```

Oczywiście ostatnią cyfrą jest cyfra 9, zatem po przekroczeniu tej liczby musimy zwiększyć liczbę dziesiątek, zaś liczbę jedności ustawić na 0.

```
if (digits[3] > 9) { //jeżeli cyfra jedności przekroczy 9
    digits[2]++; //zwiększenie liczby dziesiątek o 1
    digits[3] = 0; //wyzerowanie liczby jedności
}
```

To samo musimy powtórzyć dla cyfry drugiej i pierwszej, tyle że po przekroczeniu 9 cyfry w liczbie tysięcy, musimy wyzerować wszystkie wartości.

Następnie wyświetlamy ustalone cyfry na wyświetlaczu za pomocą funkcji (metod) skojarzonych z naszą zmienną (obiektem) przedstawiających wyświetlacz:

```
display.display(0, digits[0]); //wyświetlenie cyfry z liczbą tysięcy
                               //na pierwszej pozycji
display.display(1, digits[1]); //setki na drugiej pozycji
display.display(2, digits[2]); //dziesiątki na trzeciej pozycji
display.display(3, digits[3]); //jedności na czwartej pozycji
```

Cały kod naszej funkcji prezentuje się następująco:

```
void handleStopwatch() {
   delay(1);
   msCounter++;
   if (msCounter > 150) {
      digits[3]++;
      msCounter = 0;
   }
   if (digits[3] > 9) {
      digits[2]++;
      digits[3] = 0;
   }
   if (digits[2] > 9) {
      digits[1]++;
      digits[2] = 0;
   }
   if (digits[1] > 9) {
      digits[0]++;
      digits[1] = 0;
   }
   if (digits[0] > 9) {
      digits[3] = 0;
      digits[2] = 0;
      digits[1] = 0;
      digits[0] = 0;
   }
```

```
    display.display(0, digits[0]);
    display.display(1, digits[1]);
    display.display(2, digits[2]);
    display.display(3, digits[3]);
}
```

Teraz funkcji tej używamy za każdym razem, kiedy odmierzanie czasu jest włączone (stopwatchRunning jest prawdą), zatem w funkcji loop() dopisujemy:

```
void loop() {
  …
    if (stopwatchRunning == true) {
        handleStopwatch();
    }
}
```

Teraz nasz stoper powinien działać tak jak prawdziwy stoper, z możliwością jego pauzy. Brakuje jeszcze tylko całkowitego resetowania stopera, ale można tego dokonać bardzo prosto, poprzez sprawdzenie czy wybrany przycisk (np. 3) jest wciśnięty i ustawienie wszystkich wartości na 0:

```
if (key == 3) {
    stopwatchRunning = false;
    msCounter = 0;
    digits[3] = 0;
    digits[2] = 0;
    digits[1] = 0;
    digits[0] = 0;
    display.display(0, digits[0]);
    display.display(1, digits[1]);
    display.display(2, digits[2]);
    display.display(3, digits[3]);
}
```

Cały potrzebny kod wygląda teraz tak:

```
#include "TM1637.h"
#include “Keyboard.h”

#define BUZZER 3
#define KEYBOARD A2
#define DISPLAY_CLK 4
#define DISPLAY_DIO 5

int key = 0;
int msCounter = 0;
bool stopwatchRunning = false;
int digits[4] = {0, 0, 0, 0};
TM1637 display{DISPLAY_CLK, DISPLAY_DIO};

void beepIfClicked(int button) {
    if (button != 0) {
        digitalWrite(BUZZER, LOW);
    }
    else { //w przeciwnym wypadku
        digitalWrite(BUZZER, HIGH);
    }
}

void handleStopwatch() {
    delay(1);
    msCounter++;

    if (msCounter > 150) {
        digits[3]++;
        msCounter = 0;
    }
    if (digits[3] > 9) {
        digits[2]++;
        digits[3] = 0;
    }
```

```
    if (digits[2] > 9) {
        digits[1]++;
        digits[2] = 0;
    }
if (digits[1] > 9) {
        digits[0]++;
        digits[1] = 0;
    }
    if (digits[0] > 9) {
        digits[3] = 0;
        digits[2] = 0;
        digits[1] = 0;
        digits[0] = 0;
    }
    display.display(0, digits[0]);
    display.display(1, digits[1]);
    display.display(2, digits[2]);
    display.display(3, digits[3]);
}

void setup() {
    pinMode(KEYBOARD, INPUT);
    pinMode(BUZZER, OUTPUT);
    digitalWrite(BUZZER, HIGH);

    display.init();
    display.set(BRIGHT_TYPICAL);
}

void loop() {
    int analogKeyboardValue = analogRead(KEYBOARD);
    key = getButtonClicked(analogKeyboardValue);

    beepIfClicked(key);

    if (key == 1) {
        stopwatchRunning = true;
    }
    if (key == 2) {
```

```
        stopwatchRunning = false;
    }
    if (key == 3) {
        stopwatchRunning = false;
        msCounter = 0;
        digits[3] = 0;
        digits[2] = 0;
        digits[1] = 0;
        digits[0] = 0;
        display.display(0, digits[0]);
        display.display(1, digits[1]);
        display.display(2, digits[2]);
        display.display(3, digits[3]);
    }
    if (stopwatchRunning == true) {
        handleStopwatch();
    }
}
```

Stoper można rozszerzyć także o wyświetlanie minut i sekund oraz tworzenie międzyczasów.

Projekt zaawansowany

Opis projektu

Celem tego zadania jest stworzenie własnego budzika, który będziemy mogli ustawić za pomocą pilota IR. Bieżący czas będzie wyświetlany na wyświetlaczu 7-segmentowym, a gdy nadejdzie ustawiona godzina to zostaniemy poinformowani o tym fakcie za pomocą buzzera.

Potrzebne elementy

- Moduł Wyświetlacza
- Buzzer
- Pilot IR wraz z odbiornikiem
- Moduł RTC
- Przewody Połączeniowe

Opis modułów

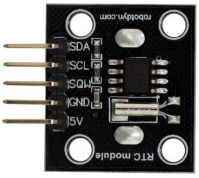
Pilot IR i odbiornik VS1838

Za pomocą pilota z tego modułu będziemy mogli zrealizować sterowanie zdalne naszym budzikiem, czyli wpisać konkretną godzinę i minutę. Każdy naciśnięty przycisk będzie generował swój unikatowy kod, który będziemy odczytywać za pomocą odbiornika.

Piny przeznaczone do użycia:

- VCC, zasilanie
- GND, masa
- OUT, dane

RTC DS1307



Jest to moduł zegara czasu rzeczywistego z rezerwowym zasilaniem bateryjnym. Pozwala nam na odczyt czasu w postaci: roku, miesiąca, dnia, godziny, minuty oraz sekundy. Komunikacja z modułem odbywa się po magistrali I2C.

Piny przeznaczone do użycia :

- SDA, komunikacja
- SCL, komunikacja
- GND, masa
- VCC, zasilanie

Buzzer

Moduł służący wydawaniu charakterystycznego dźwięku o stałym tonie. Po bardziej szczegółowy opis wróć do poprzedniego ćwiczenia.

Pozwoli nam wyświetlić aktualną godzinę i minutę w czasie rzeczywistym. Po bardziej szczegółowy opis wróć do poprzedniego ćwiczenia.

Realizacja

Na początek podłączymy odpowiednio moduły do płytki Arduino, z których będziemy korzystać, a następnie przygotujemy kod od naszego programu.
Wszystkie moduły mają piny *VCC* i *GND*, które należy odpowiednio podpiąć do pinów oznaczonych w Arduino jako *+5V* i *GND*. Pozostałe piny łączymy jak poniżej:

- Port *OUT* w buzzer podpinamy do wyjścia cyfrowego Arduino nr 4
- W wyświetlaczu port CLK podłączamy do portu nr 2, a DIO do portu nr 3
- Port *OUT* w odbiorniku IR podłączamy do wejścia cyfrowego Arduino nr 11
- Porty SDA i SCL w module RTC podłączamy do analogowych pinów A4 i A5.

Nasz kod zaczniemy od załadowania bibliotek niezbędnych do obsługi pilota, buzzera oraz wyświetlacza 7-segmentowego:

```
#include <IRremote.h>
#include <DS3231.h>
#include <Wire.h>
#include “TM1637.h”
```

Następnie zdefiniujemy piny, których użyliśmy podczas łączenia modułów z Arduino oraz zdefiniujemy wartości mówiące o rozmiarze dwóch tablic, o których zaraz opowiemy:

```
#define RECV_PIN 11
#define DISPLAY_CLK 2
#define DISPLAY_DIO 3
#define BUZZER 4

#define NUMBERS_SIZE 10
#define CLOCK_SIZE 4
```

Teraz stworzymy zmienne, które będa niezbędne do obsługi naszego budzika. Jak widać poniżej pierwsza zmienna *irreciv* jest odwzorowaniem odbiornika od naszego pilota (w nawiasach należy podać do którego pinu jest podłączony nasz odbiornik, dlatego podajemy wcześniej zdefiniowany *RECV_PIN*). Kolejna zmienna *result* będzie przechowywała ostatni kod zczytany z naszego odbiornika. Za pomocą *option* będziemy formatować odczytany kod do bardziej zrozumiałej dla nas informacji.
Poniżej znajdują się dodatkowe dwie zmienne *numbers[ ]* i *alarm_time[ ]*, które są typu tablicowego. Oznacza to, że mogą one przechowywać kilkanaście wartości, do których możemy się odwoływać za pomocą odpowiedniego indeksu (w programowaniu indeksowanie zazwyczaj zaczynamy od 0). Jak zauważymy przy tworzeniu zmiennej w formie tablicowej musimy podać w nawiasach kwadratowych maksymalny rozmiar takiej tablicy. W tym celu korzystamy z wyżej zdefiniowanych *NUMBERS_SIZE* i *CLOCK_SIZE*. *Numbers[ ]* reprezentuje zbiór kodów z naszego pilota odpowiadającym klawiszom od 0 do 9, a *alarm_time[ ]* będzie zawierał pojedyncze cyfry reprezentujące godzinę, o której nasz alarm ma się uruchomić.
Kolejna zmienna *tm1637* służy do sterowania i wyświetlania informacji na wyświetlaczu 7-segmentowym. W nawiasach należy podać piny cyfrowe obsługujące porty CLK i DIO od tego wyświetlacza. W tym celu

korzystamy z wcześniej zdefiniowanych *DISPLAY_CLK* i *DISPLAY_DIO*.
Rtc będzie służyć do pobierania danych z naszego modułu zegara rzeczywistego RTC. Niżej znajdują się dwie zmienne *h12* i *PM*, o których powiemy w dalszej części tego konspektu.
Hours i *minutes* będą służyły do przetrzymywania informacji o obecnej godzinie, natomiast *a_hours* i *a_minutes* będą godziną, o której nasz alarm ma się uruchomić. *Alarm_switch* będzie pewnym przełącznikiem, za pomocą którego będziemy uruchamiać nasz budzik, a *can_run* jest tylko zmienną pomocniczą, żeby nasz budzik uruchamiał się tylko raz.

```
 IRrecv irrecv(RECV_PIN);
decode_results results;
String option;

String numbers[NUMBERS_SIZE] = { "ff6897", "ff30cf", "ff18e7",
"ff7a85", "ff10ef", "ff38c7", "ff5aa5", "ff42bd", "ff4ab5", "ff52ad" };
byte alarm_time[CLOCK_SIZE] = { 0, 0, 0, 0 };

TM1637 tm1637(DISPLAY_CLK, DISPLAY_DIO);

DS3231 rtc;
bool h12;
bool PM;

byte hours, minutes;
byte a_hours, a_minutes;
bool alarm_switch = false;
bool can_run = false;
```

W funkcji setup zrobimy wstępną konfigurację. Pierwsza instrukcja mówi o szybkości przesyłania informacji z Arduino do naszego komputera. Ustawiamy to na wartość *9600*, ponieważ przyda nam się to później w celu wypisywania informacji w konsoli w środowisku Arduino IDE. W następnej linijce uruchamiamy nasz odbiornik od pilota. Kolejna komenda uruchamia nasz wyświetlacz oraz ustawia jego

jasność, obok podane są inne opcje więc wartość w nawiasach można zmienić według własnych preferencji. Aby móc korzystać z buzzera od naszego budzika musimy najpierw ustawić jego pin na tryb *OUTPUT* w celu wysyłania do niego informacji. Dodatkowo za pomocą polecenia *digitalWrite* ustalamy jego sygnał na *HIGH* żeby nie wydawał dźwięku w trakcie uruchamiania Arduino. Ostatnie polecenie *Write.begin()* służy do uruchomienia magistrali I2C, która jest niezbędna do działania modułu zegara rzeczywistego RTC.

```
void setup()
{
  Serial.begin(9600);

  Serial.println("IR: wlaczanie...");
  irrecv.enableIRIn();
  Serial.println("IR: uruchomiony");
  Serial.println();

  Serial.println("Wyswietlacz: wlaczanie...");
  tm1637.init();
  tm1637.set(BRIGHT_TYPICAL);   //BRIGHT_TYPICAL =
                  //2,BRIGHT_DARKEST = 0,BRIGHTEST = 7;
  Serial.println("Wyswietlacz: uruchomiony");
  Serial.println();

  Serial.println("Buzzer: wlaczanie...");
  pinMode(BUZZER, OUTPUT);
  digitalWrite(BUZZER, HIGH);
  Serial.println("Buzzer: uruchomiony");
  Serial.println();

  Wire.begin();
}
```

Napiszmy teraz kilka funkcji, za pomocą których będziemy obsługiwać nasz budzik. Ważne jest to aby każdą funkcję napisać **przed funkcją setup(), lub funkcją loop().**

Zacznijmy od funkcji o nazwie *ResetAlarm()*. Funkcja ta będzie resetowała tablicę *alarm_clock[ ]*, czyli będzie ustawiać wszystkie cztery liczby na 0:

```
void ResetAlarm()
{
  alarm_time[0] = 0;
  alarm_time[1] = 0;
  alarm_time[2] = 0;
  alarm_time[3] = 0;
}
```

W związku z tym, że informacja którą zwraca nam odbiornik jest w postaci kodu to musimy coś zrobić, aby po naciśnięciu klawiszy od 0 do 9 była zwracana wartość odpowiadająca temu klawiszowi. W przypadku naciśnięcia innego klawisza niech będzie zwracana wartość -1. Całą tą obsługę będzie wykonywać funkcja *GetNumber()*:

```
int GetNumber()
{
  option = String(results.value, HEX);

  for(int i = 0; i < NUMBERS_SIZE; i++)
  {
          if(numbers[i] == option)
          {
          return i;
          }
  }

  return -1;
}
```

Następnie zdefiniujmy funkcje *ReadPilotKeys*(), która umożliwi nam sprawdzanie kodu przycisków na porcie szeregowym. Jednym z naszych zadań będzie napisanie kodu do obsługi wybranych przez nas

przycisków od tego pilota za pomocą kodu znajdującego się w zmiennej *option.*

```
void ReadPilotKeys()
{
  if (irrecv.decode(&results))
  {
        option = String(results.value, HEX);
        Serial.print("Odbiornik: ");
        Serial.println(option);

        //ZADANIE: W tym miejscu piszemy obsługę naciśnięcia
//poszczególnych klawiszy. Należy zrobić ustawianie naszego alarmu
//(SetClock()) oraz wyłączenie budzika (alarm_switch)
        // PODPOWIEDŹ: Kliknijcie w ikonkę lupy w programie
//Arduino IDE, a potem naciskajcie klawisze od pilota w celu
//sprawdzenia jakie kody są przez niego wysyłane

        irrecv.resume();
  }
}
```

Kolejna funkcja będzie pokazywała na wyświetlaczu aktualnie ustawiony alarm, który jest przechowywany w tablicy *alarm_time[ ].* Pokazywanie na wyświetlaczu będzie samodzielnym zadaniem do wykonania, lecz w celach testowych można włączyć konsolę w Arduino IDE klikając na ikonę lupy w prawym górnym rogu programu.

```
void ShowAlarmTime()
{
  for(int i = 0; i < CLOCK_SIZE; i++)
  {
        Serial.print(alarm_time[i]);
        Serial.print(" ");
  }
  Serial.println();
```

```
  // ZADANIE: Napiszcie tutaj wyświetlenie tego samego na
//wyświetlaczu 7-segmentowym
  // PODPOWIEDŹ: Wpiszcie poniżej polecenie tm1637.display(0, 6);
//i zobaczcie co to zmieni na wyświetlaczu
  // Jeżeli udało wam się zrobić pokazywanie godziny na
//wyświetlaczu, to możecie spróbować przenieść to do powyższej
//pętli for
}
```

Następnym krokiem jest stworzenie funkcji *SetClock()* , która odpowiadała za ustawianie alarmu od zegara. Najpierw wykonamy reset alarmu poprzez już poprzednio zdefiniowaną funkcję *ResetAlarm()*, która ustawi wszystkie wartości na 0 . Następnie w pętli *while* robimy wczytywanie numerów za pomocą funkcji *GetNumber()* do momentu kiedy nie wypełnimy całej tablicy *alarm_time[ ]*.

```
void SetClock()
{
  Serial.println();
  Serial.println("Ustaw godzine:");
  ResetAlarm();
  int index = 0;

  while(index < CLOCK_SIZE)
  {
          if (irrecv.decode(&results))
          {
          int number = GetNumber();
          if(number != -1)
          {
          alarm_time[index] = number;
          index++;
          }

          ShowAlarmTime();
          irrecv.resume();
          }
```

```
  }

  // ZADANIE: Przypiszcie do zmiennej a_hours godzinę a do
//zmiennej a_minutes minuty od naszego alarmu. W tym celu
//posłużcie się tablicą alarm_time
  // PODPOWIEDŹ: Swoje wyniki możecie sprawdzać klikając w lupę
//w programie Arduino IDE

  can_run = true;

  Serial.print("Ustawiona godzina: ");
  Serial.print(a_hours);
  Serial.print(":");
  Serial.println(a_minutes);
}
```

Poniżej znajduje się nasza główna pętla *loop()*. W tej pętli wywołujemy funkcje *ReadPilotKeys()* w celu ciągłego odczytywania klawiszy naciskanych na naszym pilocie.
Niżej zczytujemy godzinę oraz minuty z modułu RTC za pomocą funkcji *getHour()* i *getMinute()*. Jak zauważymy do funkcji *getHour()* podajemy dwie zmienne do nawiasów: *h12* i *PM*. Za pomocą tych zmiennych możemy sprawdzać w jakim formacie wyświetla się godzinę (12 czy 24 godziny). W tym projekcie nie będziemy korzystać z *h12* i *PM*, ale niezbędne jest aby znalazły się w nawiasach od funkcji *getHour().*
Cztery linijki zawierające *Serial.print()* służą do wypisywania godziny w konsoli Arduino IDE, którą możemy otworzyć klikając w ikonkę lupy.
Parę linijek niżej możemy zauważyć instrukcję sterująca *if*. W prostym objaśnieniu ta funkcja włącza alarm w momencie, kiedy obecna godzina i minuta jest taka sama jak ta ustawiona przez nas dla alarmu. Jeżeli warunek jest spełniony to *alarm_switch* przyjmuje wartość *true,* czyli prawdę i na jej podstawie powinien uruchomić buzzer. Dodatkowo widzimy zmienną zabezpieczającą o nazwie *can_run*, która sprawia, że alarm uruchamia się tylko raz. Bez niej w momencie kiedy alarm zostałby wyłączony to automatycznie uruchomi się ponownie.

```
void loop()
{
  ReadPilotKeys();

  hours = rtc.getHour(h12, PM);
  minutes = rtc.getMinute();

  Serial.print("Obecna godzina: ");
  Serial.print(hours);
  Serial.print(":");
  Serial.println(minutes);

  // ZADANIE: Tutaj należy wyświetlać obecną godzinę na
//wyświetlaczu 7-segmentowym
  // PODPOWIEDŹ: Korzystajcie tylko ze zmiennych hours i minutes
//oraz z operacji dzielenia (/) i reszty z dzielenia (%)

  // Uruchamianie alarmu
  if(hours == a_hours && minutes == a_minutes && can_run == true)
  {
        alarm_switch = true;
        can_run = false;
  }

  // ZADANIE: Napiszcie własny kod, który będzie włączał buzzer w
//momencie kiedy zmienna alarm_switch jest prawdą (true)
// PODPOWIEDŹ: Popatrzcie w kod od setup jak ustawiamy, żeby
//buzzer nie był włączony w momencie uruchamiania Arduino
  // Jeżeli uda wam się to zrobić to spróbujcie zrobić pikanie z
//odstępami czasowymi za pomocą polecenia delay(<czas>)

  delay(100);
}
```

Poniżej zamieszczamy napisane przez nas rozwiązanie do zaawansowanego zadania. Zalecamy wpierw spróbować samemu rozwiązać podane zadania przed spojrzeniem na (przykładowe) rozwiązanie.

```
#define RECV_PIN 11
#define DISPLAY_CLK 2
#define DISPLAY_DIO 3
#define BUZZER 4

#define NUMBERS_SIZE 10
#define CLOCK_SIZE 4

#include <Wire.h>
#include <IRremote.h>
#include <DS3231.h>
#include "TM1637.h"

IRrecv irrecv(RECV_PIN);
decode_results results;
String option;

String numbers[NUMBERS_SIZE] = { "ff6897", "ff30cf", "ff18e7",
"ff7a85", "ff10ef", "ff38c7", "ff5aa5", "ff42bd", "ff4ab5", "ff52ad" };
byte alarm_time[CLOCK_SIZE] = { 0, 0, 0, 0 };

TM1637 tm1637(DISPLAY_CLK, DISPLAY_DIO);

DS3231 rtc;
bool h12;
bool PM;

byte hours, minutes;
byte a_hours, a_minutes;
bool alarm_switch = false;
bool can_run = false;

int GetNumber()
{
  option = String(results.value, HEX);

  for(int i = 0; i < NUMBERS_SIZE; i++)
```

```
  {
        if(numbers[i] == option)
        {
        return i;
        }
  }

  return -1;
}

void ReadPilotKeys()
{
  if (irrecv.decode(&results))
  {
        option = String(results.value, HEX);
        Serial.print("Odbiornik: ");
        Serial.println(option);

        // ZADANIE: W tym miejscu piszemy obsługę naciśnięcia
//poszczególnych klawiszy. Należy zrobić ustawianie naszego alarmu
//(SetClock()) oraz wyłączenie budzika (alarm_switch)
        // PODPOWIEDŹ: Kliknijcie w ikonkę lupy w programie
//Arduino IDE, a potem naciskajcie klawisze od pilota w celu
//sprawdzenia jakie kody są przez niego wysyłane
        if(option == "ff906f")
        {
        SetClock();
        }

        if(option == "ffc23d")
        {
        alarm_switch = false;
        }

        irrecv.resume();
  }
}

void ResetAlarm()
```

```
{
  alarm_time[0] = 0;
  alarm_time[1] = 0;
  alarm_time[2] = 0;
  alarm_time[3] = 0;
}

void ShowAlarmTime()
{
  for(int i = 0; i < CLOCK_SIZE; i++)
  {
        Serial.print(alarm_time[i]);
        Serial.print(" ");
  }
  Serial.println();

  // ZADANIE: Napiszcie tutaj wyświetlenie tego samego na
//wyświetlaczu 7-segmentowym
  // PODPOWIEDŹ: Wpiszcie poniżej polecenie tm1637.display(0, 6);
//i zobaczcie co to zmieni na wyświetlaczu
  // Jeżeli udało wam się zrobić pokazywanie godziny na
//wyświetlaczu, to możecie spróbować przenieść to do powyższej
//pętli for
  tm1637.display(0, alarm_time[0]);
  tm1637.display(1, alarm_time[1]);
  tm1637.display(2, alarm_time[2]);
  tm1637.display(3, alarm_time[3]);
}

void SetClock()
{
  Serial.println();
  Serial.println("Ustaw godzine:");
  ResetAlarm();
  int index = 0;

  while(index < CLOCK_SIZE)
  {
        if (irrecv.decode(&results))
```

```
        {
        int number = GetNumber();
        if(number != -1)
        {
        alarm_time[index] = number;
        index++;
        }

        ShowAlarmTime();
        irrecv.resume();
        }
  }

  // ZADANIE: Przypiszcie do zmiennej a_hours godzinę a do
//zmiennej a_minutes minuty od naszego alarmu. W tym celu
//posłużcie się tablicą alarm_time
  // PODPOWIEDŹ: Swoje wyniki możecie sprawdzać klikając w lupę
//w programie Arduino IDE
  a_hours = (10 * alarm_time[0]) + alarm_time[1];
  a_minutes = (10 * alarm_time[2]) + alarm_time[3];
  can_run = true;

  Serial.print("Ustawiona godzina: ");
  Serial.print(a_hours);
  Serial.print(":");
  Serial.println(a_minutes);
}

void setup()
{
  Serial.begin(9600);

  Serial.println("IR: wlaczanie...");
  irrecv.enableIRIn();
  Serial.println("IR: uruchomiony");
  Serial.println();

  Serial.println("Wyswietlacz: wlaczanie...");
  tm1637.init();
```

```
  tm1637.set(BRIGHT_TYPICAL);   //BRIGHT_TYPICAL =
//2,BRIGHT_DARKEST = 0,BRIGHTEST = 7;
  Serial.println("Wyswietlacz: uruchomiony");
  Serial.println();

  Serial.println("Buzzer: wlaczanie...");
  pinMode(BUZZER, OUTPUT);
  digitalWrite(BUZZER, HIGH);
  Serial.println("Buzzer: uruchomiony");
  Serial.println();

  Wire.begin();
}

void loop()
{
  ReadPilotKeys();

  hours = rtc.getHour(h12, PM);
  minutes = rtc.getMinute();

  Serial.print("Obecna godzina: ");
  Serial.print(hours);
  Serial.print(":");
  Serial.println(minutes);

  // ZADANIE: Tutaj należy wyświetlać obecną godzinę na
//wyświetlaczu 7-segmentowym
  // PODPOWIEDŹ: Korzystajcie tylko ze zmiennych hours i minutes
//oraz z operacji dzielenia (/) i reszty z dzielenia (%)
  tm1637.display(0, hours / 10);
  tm1637.display(1, hours % 10);
  tm1637.display(2, minutes / 10);
  tm1637.display(3, minutes % 10);

  // Uruchamianie alarmu
  if(hours == a_hours && minutes == a_minutes && can_run == true)
  {
        alarm_switch = true;
```

```
        can_run = false;
  }

  // ZADANIE: Napiszcie własny kod, który będzie włączał buzzer w
//momencie kiedy zmienna alarm_switch jest prawdą (true)
  // PODPOWIEDŹ: Popatrzcie w kod od setup jak ustawiamy, żeby
//buzzer nie był włączony w momencie uruchamiania Arduino
  // Jeżeli uda wam się to zrobić, to spróbujcie zrobić pikanie z
//odstępami czasowymi za pomocą polecenia delay(<czas>)
  // Alarm - BUZZER
  if(alarm_switch == true)
  {
        digitalWrite(BUZZER, LOW);
        delay(500);
        digitalWrite(BUZZER, HIGH);
  }

  delay(100);
}
```