29.12.2016

ZAKŁAD PODSTAW ELEKTRONIKI

Sterowanie elementami i układami optoelektronicznymi za pomocą mikrokontrolera na platformie Arduino

Wykonała:
Paulina Wątroba

Wstęp:

Projekt zakłada omówienie obsługi elementów i modułów optoelektronicznych w szczególności diody LED, fototranzystora, wyświetlacza 7-segmentowego LED, diody RGB LED z wbudowanym sterownikiem przez układ Arduino.

Projekt będzie miał formę ćwiczenia laboratoryjnego, które, po wcześniejszym przygotowaniu, można wykonać na zajęciach. Głównym celem tego ćwiczenia będzie zapoznanie się z mechanizmami działania, bibliotekami ułatwiającymi komunikację kontrolera i urządzeń oraz dodatkowo z podstawowym połączeniem i obsługą przerwań.

Należy zbudować zgodnie z dołączonym schematem połączeń układ zawierający mikrokontroler, czujnik szczelinowy, tranzystor z silnikiem, wyświetlacz i linijkę diodową z wykorzystan0iem płytki prototypowej,

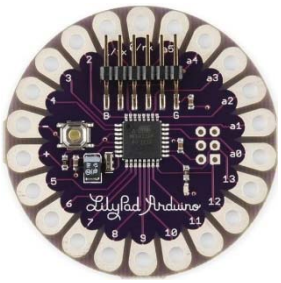
Elementy układu

- Płyta Arduino,
- Wyświetlacz LED 4-cyfrowy 7-segmentowy z układem TM1637
- Czujnik szczelinowy F-03 (dioda LED - fototranzystor z komparatorem LM393)
- Linijka diodowa LED RGB z kontrolerem WS2812B
- Płytka prototypowa (stykowa)
- Tranzystor TIP122
- Silnik prądu stałego
- Koło kodujące
- Rezystor
- Kondensator
- Zasilacz napięcia stałego 7,5V

Płyta Arduino

Arduino UNO i MEGA

Arduino Nano



Arduino LilyPad

"Arduino – platforma dla systemów wbudowanych oparta na prostym projekcie Open Hardware przeznaczonym dla mikrokontrolerów montowanych w pojedynczym obwodzie drukowanym, z wbudowaną obsługą wejścia/wyjścia oraz standaryzowanym językiem programowania. Język programowania Arduino jest oparty na środowisku Wiring i zasadniczo na języku C/C++. Celem projektu Arduino jest przygotowanie narzędzi – ogólnodostępnych, tanich, niewymagających dużych nakładów finansowych, elastycznych i łatwych w użyciu przez hobbystów. Częściowo, Arduino stanowi również alternatywę dla osób, które nie mają dostępu do bardziej zaawansowanych kontrolerów, wymagających bardziej skomplikowanych narzędzi." [pl.wikipedia.org]

Na rynku dostępne są różne układy oparte o ideę Arduino.

Najpopularniejszym z nich jest Arduino UNO zbudowany z wykorzystaniem mikrokontrolera ATMEGA328 taktowanego zegarem 16MHz.

Innym popularnym rozwiązaniem jest Arduino MEGA wyposażony w procesor ATMEGA1280. Od pierwowzoru różni się zastosowanym mikrokontrolerem, który posiada większą ilość wyprowadzeń oraz więcej pamięci FLASH dla programu.

Poniżej przedstawiono parametry typowych układów ARDUINO:

PARAMETR	**ARDUINO UNO**	**ARDUINO MEGA**
Mikrokontroler	ATMEGA328	ATMEGA1280
FLASH	32KB	128KB
RAM	1KB	8KB
EEPROM	1KB	4KB
PORTY We/Wy	23	86
Wejścia analogowe	6	16

Są również dostępne ograniczone funkcjonalnie zminiaturyzowane wersje do specjalnych zastosowań takie jak Arduino Mini i Nano do niewielkich urządzeń oraz LilyPad do sterowania elementami optoelektronicznymi wszywanymi w odzież.

Arduino to kompletny system umożliwiający sterowanie różnymi układami oraz odczyt danych z różnych źródeł. Poważnym atutem układów Arduino jest ustandaryzowany rozkład wyprowadzeń, co umożliwia stosowanie gotowych rozwiązań rozszerzających możliwości układu. Specjalne płytki zwane shieldami mogą uzupełnić nasz układ o kartę sieciową, wyjścia do sterowania silnikami krokowymi, czy czujnik odległości. Od strony programu każdy pin układu jest jednoznacznie określony, co ułatwia tworzenie własnych.

Przycisk **RESET**. Umożliwia on powrót do pierwotnego stanu układu bezpośrednio po włączeniu zasilania. Po naciśnięciu przycisku Arduino rozpoczyna wykonywanie programu od początku, dane w pamięci RAM mikrokontrolera zostaną wyzerowane.

Interfejs **USB** umożliwia za pomocą komputera programowanie układu Arduino oraz komunikację z układem wykonującym program. Poprzez USB można zasilać gotowy projektowany układ, jednak USB ma małą wydajność prądową i nie można z niego zasilać układów wymagających większych mocy np. silników, krokowych, czy serwomechanizmów.

Każdy układ ARDUINO wyposażony jest w gniazdo **DC** umożliwiające zasilanie. Napięcie zasilania powinno być z przedziału **7 - 12V**. Zasilanie poprzez gniazdo DC ma sens w przypadku,

gdy część układu wymaga napięcia zasilania większego niż 5V oraz w sytuacji, gdy jest wymagany większy prąd zasilania układu, albo gdy budowany układ docelowo nie jest podłączony do komputera.

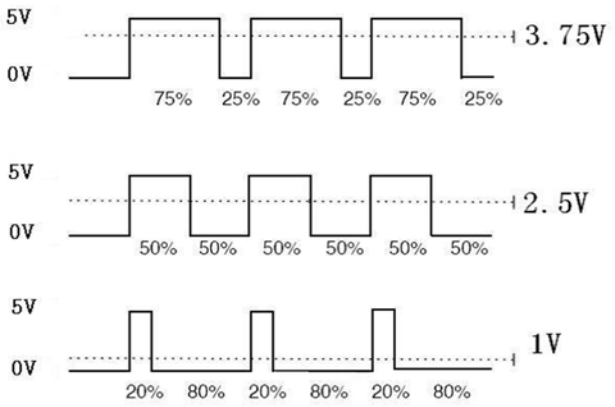
Na płycie Arduino znajduje się jedno lub dwa złącza sześcio-pinowe wykorzystywane do programowania zainstalowanych mikrokontrolerów. Złącza oznaczone są jako **ICSP1 i ICSP2**. ARDUINO wyposażone jest w przynajmniej **4 diody LED**. Dwie z nich oznaczone jako **RX i TX**. Sygnalizują one transmisję szeregową pomiędzy komputerem a sterownikiem. Kolejna dioda oznaczona jako **PWR** to kontrolka zasilania układu poprzez złącze DC lub USB. Ostatnia typowo umieszczona dioda to dioda LED podłączona do wyprowadzenia pin13. Można ją programowo zapalać i gasić w zależności od potrzeb. Dioda podłączona jest do masy układu, więc podanie logicznej jedynki na pin13 spowoduje jej zapalenie, natomiast podanie logicznego zera spowoduje jej zgaszenie.

Najważniejszym elementem układu Arduino są listwy PIN na górze i na dole. Ich rozmieszczenie jest ustandaryzowane, co ułatwia wykorzystywanie przykładowych programów oraz dodawanie shieldów.

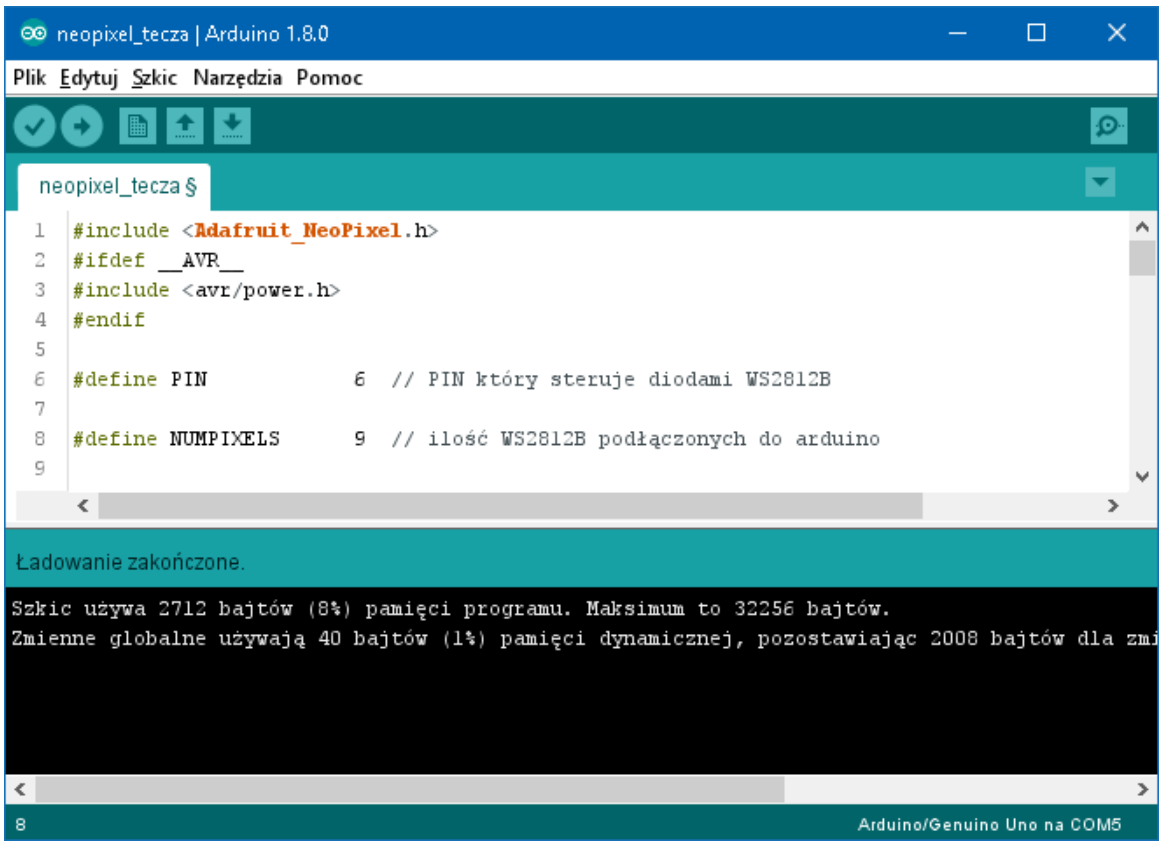
- **IOREF** - wskazuje jakim napięciem zasilany jest procesor w Arduino (ważne dla niektórych shieldów),
- **RESET** - reset układu Arduino, lub reset układów shield,
- **3V3** - zasilanie układów wymagających napięcia 3,3V,
- **5V** - zasilanie układów TTL,
- **GND** - masa układu,
- **VIN** - napięcie ze złącza DC
- **AREF** służące do podania napięcia odniesienia dla przetwornika analogowo-cyfrowego.

Piny **ANALOG IN** umożliwiające odczyt wartości analogowych. Przetwornik analogowo-cyfrowy układu umożliwia odczyt wartości napięcia z przedziału **0-AREF** lub **0-5V**. Odczytana wartość może być 8-mio lub 10-cio bitowa. Wejścia analogowe opisane są jako **A0** do **A5**. Piny te mogą być wykorzystane jako wejścia lub wyjścia cyfrowe.

Piny **DIGITAL** oznaczone **D0** do **D13 -** wejścia/wyjścia cyfrowe. Piny **0 i 1** to piny specjalne, na których wyprowadzono linie portu szeregowego. Można go wykorzystać do transmisji szeregowej z innym układem. Piny **3, 5, 6, 9, 10, 11** oznaczono jako ~ lub **PWM**. Mogą one pracować w trybie PWM umożliwiającym sterowanie szerokością impulsu, co często wykorzystywane jest do sterowania, przebieg o określonym wypełnieniu odpowiada napięciu stałemu o poziomie pośrednim między stanem niskim 0V i wysokim 5V (przykład na rysunku poniżej).



Oprogramowanie ArduinoIDE



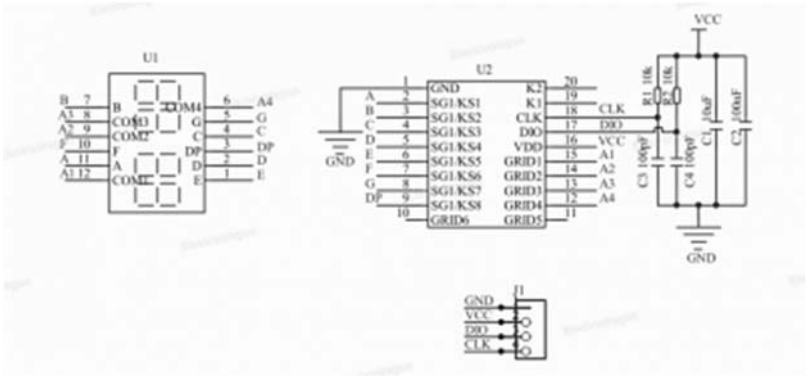
Arduino IDE jest wieloplatformową aplikacją napisaną w języku Java. Środowisko IDE zawiera edytor kodu z takimi funkcjami jak kolorowanie składni, automatyczne wcięcia w kodzie oraz pozwala na kompilację i załadowanie programu do płyty Arduino.

Programy dla Arduino są napisane głównie w języku C/C++, do prawidłowego działania układu użytkownik musi zdefiniować co najmniej dwie funkcje, aby otrzymać gotowy do uruchomienia program:

- setup() – funkcja wykonywana raz, umożliwia definicję parametrów początkowych (startowych),
- loop() – funkcja w której znajduje się kod programu wykonywanego cyklicznie do czasu odłączenia zasilania od układu

Opis elementów i modułów

Wyświetlacz LED 4 cyfrowy 7-segmentowy z kontrolerem TM1637



Wyświetlacz składa się z 4 cyfr LED rozdzielonych znakiem „:”. Każda z cyfr składa się z 7 segmentów. Moduł sterowany jest przez kontroler TM1637 umożliwiający podłączenie za pomocą 4 przewodów:

- CLK: wyjście zegarowe- podłączamy do pinu cyfrowego Arduino, np D4
- DIO: dane - podłączamy do pinu cyfrowego Arduino, np D3,
- VCC: zasilanie - podłączamy do 5V Arduino
- GND: masa - podłączamy do GND Arduino

Do sterowania układem przez Arduino wykorzystujemy bibliotekę: <TM1637Display.h>, która pozwala ustalić poziom jasności świecenia cyfr, wyświetlać 4-cyfrowe liczby, lub dowolnie ustawiać dla każdej z pozycji wyświetlaną wartość.

Definicja pinów:

```
const int CLK = 3;
const int DIO = 4;
```

Utworzenie obiektu klasy:

```
TM1637Display display(CLK, DIO);
```

Do sterowania wyświetlaczem wykorzystujemy metody klasy.

Metoda	Opis
setBrightness(brightness)	Ustawia jasność **brightness** wyświetlacza.
showNumberDec(num,leading_zero, length,pos);	Wyświetla na ekranie numer **num**. Parametr **leading_zero** może przyjmować wartość false lub true; jeżeli true: liczba **num** jest dopełniana zerami do ilości **length**. **pos** oznacza pozycję, od której wyświetlana jest **num** (0: lewy segment wyświetlacza, tutaj: 3: najbardziej prawy).
setSegments(segments[],length, pos);	Wyświetla kombinację określoną przez **segments** od pozycji **pos**. Tabela **segments** ma **length** elementów.
encodeDigit(digit);	Zamienia podaną liczbę **digit** (0-15) na wartość odpowiadającą kodom segmentów; liczby powyżej 9 traktowane są jako heksadecymalne (A, B, C, D, E, F); zwróconą wartość możecie użyć z funkcją **setSegmants()**.

Przykłady:

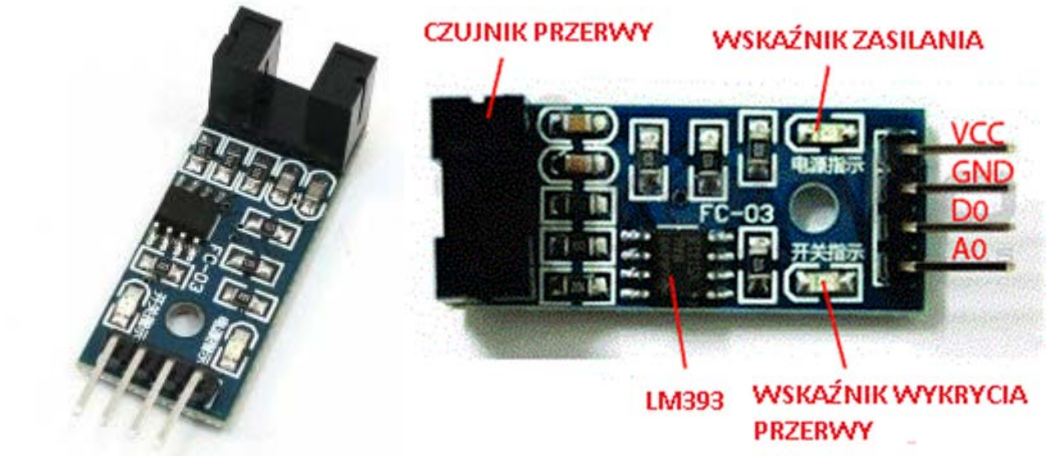
Ustawienie jasności wyświetlacza:

```
display.setBrightness(0x0a);
```

Wyświetlanie liczby:

```
display.showNumberDec(liczba);
```

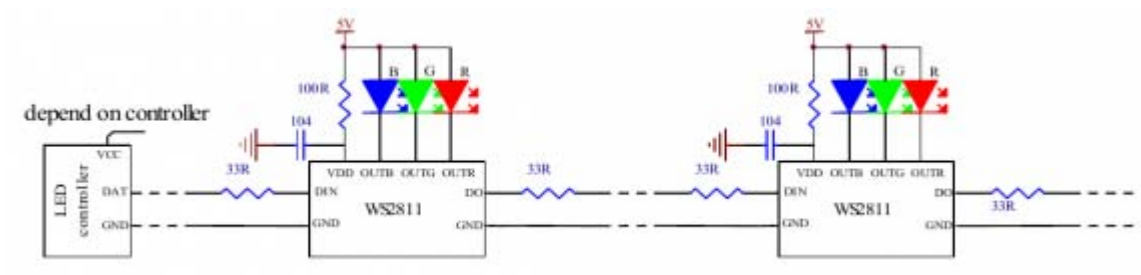
Czujnik szczelinowy F-03 (dioda LED - fototranzystor) z komparatorem LM393



Czujnik szczelinowy zbudowany jest z układu dioda LED–fototranzystor do detekcji przerwy, komparatora LM393 oraz diod LED wskazujące zasilanie i wykrycie zmiany stanu czujnika, posiada wyjście analogowe i cyfrowe.

Wykrycie przerwy objawia się zmianą stanu na wyjściach, zmianę tą można wykrywać po podłączeniu wyjścia cyfrowego do pinu D2 Arduino (obsługującego sprzętowe przerwanie) co pozwoli na przykład podejmować działania zależne od zmiany.

Linijka diodowa LED RGB z kontrolerem WS2812B



WS2812 jest inteligentnym sterownikiem **RGB LED** z wbudowanym źródłem światła posiadający możliwość szeregowego łączenia, a także sterowania praktycznie dowolną ilością diod RGB tylko za pomocą pojedynczego pinu cyfrowego mikrokontrolera za pośrednictwem magistrali 1-Wire.

Do sterowania układem przez Arduino wykorzystujemy bibliotekę <Adafruit_NeoPixel.h>, wykorzystujemy następujące funkcje:

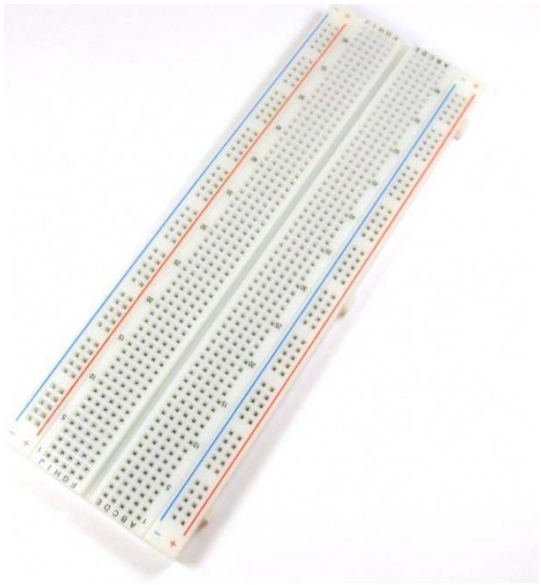
begin()	inicjalizacja biblioteki NeoPixel
setPixelColor()	ustawienia koloru świecenia wybranej diody w łańcuchu
show()	wysłanie danych do naszego łańcucha z diodami
setPin()	zmiana pinu sterującego
setBrightness()	ustawienie jasności świecenia
clear()	wygaszenie całego łańcucha
getPixels()	pobranie wskaźnika do danych, po 3 bajty dla każdej diody
getBrightness()	pobranie ustawionej jasności świecenia
numPixels()	pobranie ilości diod w łańcuchu
getPixelColor()	pobranie koloru wybranej diody
canShow()	zwraca wartość true, jeśli możliwe jest wysłanie danych

Przykład:
Utworzenie obiektu
```
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(ILOSC_DIOD, PIN, NEO_GRB
+ NEO_KHZ800);
```
Inicjalizacja:
```
pixels.begin();
Ustawienie koloru czerwonego dla diody nr 3
pixels.setPixelColor(3, 255, 0, 0);
```
Wysłanie ustawień do łańcucha diodowego
```
pixels.show();
```

Tranzystor TIP122

Tranzystor NPN Darlingtona służy w układzie do sterowania napięciem zasilającym silnik, który obraca dyskiem z nacięciem do wywoływania zmian w czujniku szczelinowy. Sterowanie odbywa się przez podanie napięcia z wyjścia cyfrowego PWM o odpowiednim wypełnieniu przez rezystor ograniczający na bazę tranzystora. W obwodzie kolektor-emiter, do którego podłączono silnik z dodatkowym źródłem zasilania następuje wysterowanie napięcia wpływające na zmianę prędkości obrotowej silnika.

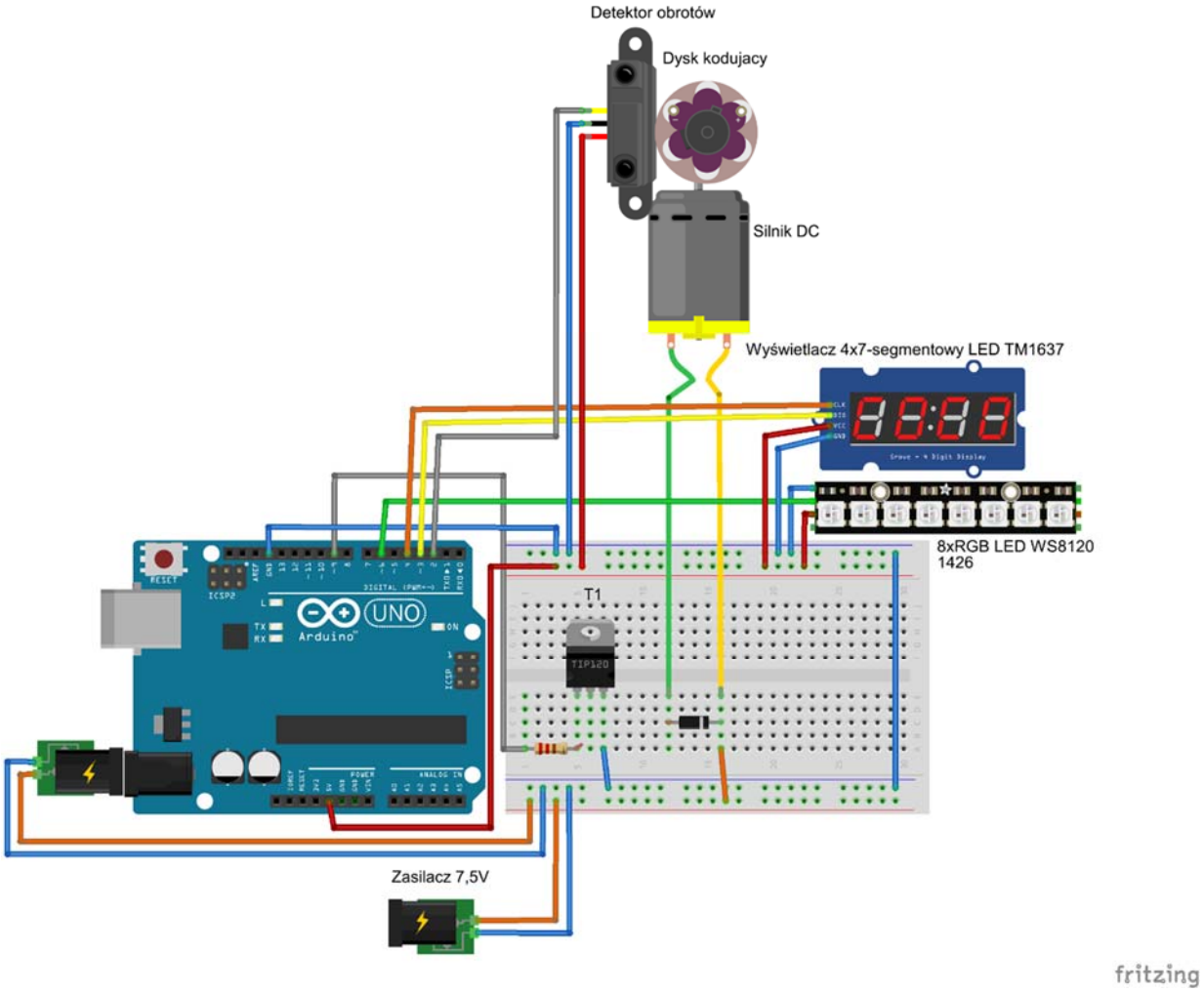
Płytka prototypowa



Narzędzie umożliwiające szybkie tworzenie, podłączanie układów obwodów elektronicznych bez konieczności ich każdorazowego lutowania i przygotowywania/wytrawiania dedykowanych płytek połączeniowych.

Montaż układu:

Należy zgodnie z załączonym schematem połączyć elementy układu i sprawdzić poprawność połączeń.



Schematy wykonane za pomocą programu Fritzing.

Programowanie układu

Przed wykonaniem tej części zadania należy upewnić się, że układ został poprawnie zmontowany!

Przygotowany układ pozwala przetestować rejestrowanie zmian czujnika optycznego i sterowanie wyświetlaczem 4-pozycyjnym 7-segmentowym LED z wykorzystaniem układu wbudowanego kontrolera, a także sterowanie szeregiem diodowym RGB LED z wbudowanym kontrolerem pozwalającym dowolnie sterować stanem i przeznaczeniem każdej diody z osobna. Pozwala to na ograniczenie potrzebnych połączeń, a jednocześnie zwiększa możliwości wykorzystania tego typu elementów w różnych urządzeniach i układach.

Źródłem zmian czujnika będzie sterowany z Arduino silnik prądu stałego, do czego wykorzystamy przebieg o modulowanym wypełnieniu kontrolującym napięcie na tranzystorze zasilającym silnik.

W celu realizacji powyższych zadań musimy do naszego programu dołączyć następujące biblioteki:

- sterowanie wyświetlaczem

  ```
  #include <TM1637Display.h>
  ```

- sterowanie diodami LED RGB

  ```
  #include <Adafruit_NeoPixel.h>
  ```

- opcjonalne wykorzystanie timera mikrokontrolera

  ```
  #include <TimerOne.h
  ```

Potrzebne będą nam odpowiednie parametry opisujące sposób podłączenia modułów do układu:

```
#define NEOPIXELPIN   6  // PIN który steruje diodami WS2812B
#define NUMPIXELS     8  // ilość WS2812B podłączonych do Arduino
#define motorPin      9  //PWM sterowanie silnikiem
const int CLK = 3;       //wejście CLK wyświetlacza
const int DIO = 4;       //wejście DIO wyświetlacza
```

Obiekty klas:

```
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN,
NEO_GRB + NEO_KHZ800);
TM1637Display display(CLK, DIO);
```

Programowanie funkcji setup();

Należy ustawić jasność świecenia wyświetlacza:

```
display.setBrightness(0x0a);
```

Korzystając funkcji *pinMode(pin.mode)* musimy ustawić odpowiednie tryby pracy poszczególnych pinów, np.:

```
pinMode(motorPin, OUTPUT);
```

Konieczne będzie zdefiniowanie odpowiednich zmiennych, ich wartości początkowych oraz ustawienie przerwania sprzętowego na wejściu D2 i przygotowanie funkcji counter(); obsługującej przerwanie:

```
attachInterrupt(0, counter, RISING);
```

Należy zdecydować czy do kontroli czasu wykorzystywać będziemy funkcję *millis()*, czy też bibliotekę TimerOne i zależnie od tego w odpowiedni sposób przechowywać informację o czasie działania programu. Przyjmujemy, że czas pomiaru obrotów to 1 sekunda

Programowanie funkcji *loop();*

W tej funkcji musimy zaprogramować:

- sterowanie prędkością silnika (można zaprogramować wartość stałą lub zmieniającą się w czasie):

  ```
  analogWrite(motorPin, i);
  ```

- sprawdzamy, czy minął czas 1 sekundy poprzez porównanie czasu bieżącego z poprzednio zapisanym, jeżeli tak to wykonujemy poniższe komendy
- zatrzymanie obsługi przerwań na czas prezentacji danych

  ```
  noInterrupts();
  ```

- przeliczenie i wysłanie aktualnej informacji o obrotach na wyświetlacz

  ```
  rpm = pulses * 60; //przeliczanie obroty na minutę

  display.showNumberDec(rpm);
  ```

- wyświetlenie na linijce diodowej informacji o wartości napięcia sterującego silnikiem w wybrany sposób

  ```
  pixels.setPixelColor(7, pixels.Color(255, 0, 0));
  pixels.show();
  ```

- Zerujemy licznik czasu
- Przywracamy obsługę przerwań

  ```
  interrupts();
  ```

Programowanie funkcji *counter()*

Funkcja wywoływana podczas obsługi przerwania służąca do zliczania obrotów

```
void counter(){
  if(digitalRead (encoder_pin) {
        pulses++;}
```

Należy zaobserwować stan informacji na wyświetlaczu i linijce diodowej czy są zgodne z zaprogramowanymi parametrami.

Zaprogramować narastanie napięcia sterującego obrotami silnika, uzależnić od niego ilość zapalanych diod. W celu zapewnienia zatrzymania się koła, za pomocą funkcji delay() zaprogramować 10 sekundowe opóźnienie.

Zaobserwować przy ilu zapalonych diodach koło zaczyna się kręcić.

Można dodatkowo zaprogramować zapalanie się, gaszenie oraz zmianę koloru poszczególnych diod.