

Arduino dla każdego małego i dużego

Wstęp do świata elektroniki, robotyki i programowania

Licence/Licencja

I share these files under non-commercial licence. / Udostępniam plik tylko do wykorzystania niekomercyjnego.



Autor: Zbigniew „HardeN” Brożbar www.robomaniak.pl

SPIS TREŚCI

1. Słów kilka o Arduino	2
2. Arduino IDE	2
2.1. Instalacja Arduino IDE	3
2.2. Konfiguracja Arduino IDE	4
2.3. Pierwszy program	6
2.4. Ciekawostki o Arduino IDE	7
3. Podstawy programowania płytek Arduino	8
3.1. Podstawowy język C++	8
3.1.1. Typy danych	9
3.1.2. Zmienne	9
3.1.3. Operatory	11
3.1.4. Instrukcje warunkowe	12
3.1.5. Funkcje	15
3.1.6. Pętle	16
3.1.7. Tablice	20
3.1.8. Struktura	20
3.1.9. Słowa kluczowe	21
3.1.10. Struktura szkicu	22
3.1.11. Klasy	24
3.2. Podstawy języka „arduino”	25
4. Podstawy elektroniki	34
4.1. Podstawowe pojęcia	34
4.1.1. Prawo Ohma	36
4.1.2. Prawa Kirchhoffa	37
4.2. Najczęściej spotykane komponenty elektroniczne	38
4.2.1. Rezystor	40
4.2.2. Kondensator	43
4.2.3. Półprzewodniki	47
4.2.4. Dioda	48
4.2.5. Tranzystor	50
4.2.6. Układ scalony	53
4.2.7. Stabilizator	54
4.3. Płytki stykowe	56
4.4. Schematy elektroniczne	57
4.4.1. Podstawowe symbole elementów elektronicznych	58
4.4.2. Schemat blokowy	59
4.4.3. Schemat ideowy	62
4.4.4. Schemat „wizualny”	67
4.4.5. Oprogramowanie	68
5. Arduino NANO	69
5.1. Informacje podstawowe	70
5.2. Porty wejścia/wyjścia	71
5.2.1. Porty cyfrowe	71
5.2.2. Porty PWM	71
6. Bibliografia	73
6.1. Książki	73
6.2. Strony internetowe	73

1. Słów kilka o Arduino

Arduino jest platformą programistyczną przeznaczoną do tworzenia układów elektronicznych. Co to oznacza? Dzięki arduino możesz nie tylko pisać programy, lecz także modyfikować sprzęt, budować wymyślone przez siebie urządzenia, pojazdy, roboty. Mówiąc o Arduino często mamy na myśli płytkę Arduino. Płytki mają wiele odmian. W tej książce skupię się na wersji Arduino Nano.

Programowanie odbywa się w języku C/C++. Na początku swojej przygody z programowaniem możesz korzystać z gotowych programów, nazywanych w szkicami. Dzięki korzystaniu z gotowych szkiców możemy szybciej zrozumieć podstawowe elementy języka programowania, takie jak zmienne, instrukcje, funkcje, pętle lub biblioteki.

Na koniec tego krótkiego wstępu definicja Arduino z wikipedia.

„Arduino – platforma programistyczna dla systemów wbudowanych oparta na prostym projekcie Open Hardware przeznaczonym dla mikrokontrolerów montowanych w pojedynczym obwodzie drukowanym, z wbudowaną obsługą układów wejścia/wyjścia oraz standaryzowanym językiem programowania. Język programowania Arduino jest oparty na środowisku Wiring i zasadniczo na języku C/C++ (kilka prostych przekształceń kodu wykonywane przed przejściem do avr-gcc). 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.”

2. Arduino IDE

Arduino IDE to specjalna aplikacja stworzona przez twórców platformy arduino. IDE czyli z ang. Integrated Development Environment czyli po polsku zintegrowane środowisko programistyczne. Arduino IDE to nie tylko edytor kodu programu ale też na swój sposób rozbudowana aplikacja posiadająca w sobie wszystko co potrzebujemy do pracy z wszystkimi płytkami arduino. Aplikacja jest edytorem, kompilatorem szkiców, programatorem, posiada mechanizm zarządzania bibliotekami i inne. W tym rozdziale będę starał się przekazać Tobie podstawowe informacje jak poruszać się w tej aplikacji.



2.1. Instalacja Arduino IDE

Zacznijmy od instalacji. Aplikację możemy pobrać z oficjalnej strony arduino.cc, z sekcji Software, Downloads. Aplikacja dostępna jest na system operacyjny Windows, Linux i Mac OS X. W tej książce będę korzystał z wersji na system Windows 10.

Downloads



Arduino IDE 1.8.13

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any Arduino board.

Refer to the [Getting Started](#) page for Installation instructions.

SOURCE CODE

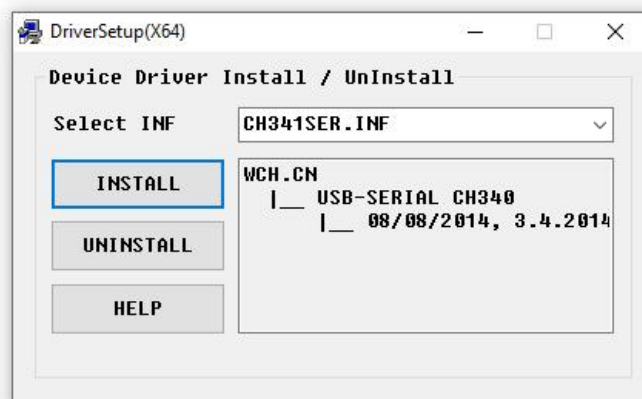
Active development of the Arduino software is [hosted by GitHub](#). See the instructions for [building the code](#). Latest release source code archives are available [here](#). The archives are PGP-signed so they can be verified using [this](#) gpg key.

DOWNLOAD OPTIONS

- Windows** Win 7 and newer
- Windows** ZIP file
- Windows app** Win 8.1 or 10 [Get](#)
- Linux** 32 bits
- Linux** 64 bits
- Linux** ARM 32 bits
- Linux** ARM 64 bits
- Mac OS X** 10.10 or newer

[Release Notes](#)
[Checksums \(sha512\)](#)

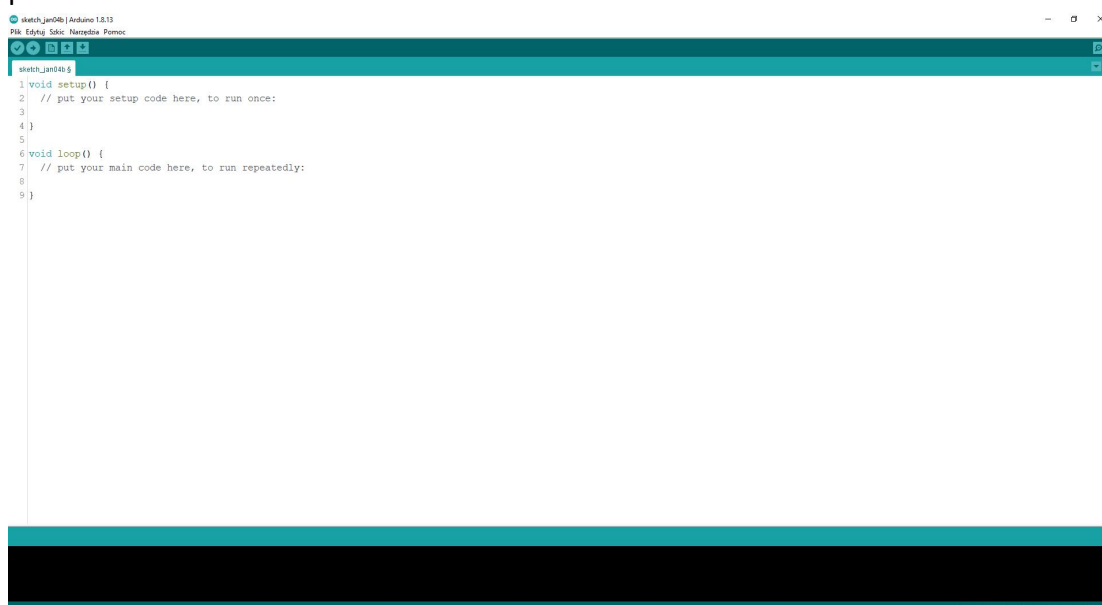
Pobieramy aplikację naciskając kursorem myszki na „Windows Win 7 and newer”. Przechodzimy do podstrony gdzie możemy wesprzeć twórców darowizną ale możemy też tylko pobrać plik naciskając na „JUST DOWNLOAD”. Po pobraniu pliku na swój komputer uruchamiamy instalator z prawami administratora. W celu instalacji nowej aplikacji na naszym komputerze musimy być zalogowani jako administrator lub uruchomić plik z prawami administratora. W pierwszym kroku instalacji zgadzamy się na licencję, klikając „I Agree”. W kolejnym kroku wybieramy komponenty do instalacji, sugeruje tutaj przejść dalej zostawiając ustawienia domyśle, klikamy „Next”. Kolejny krok to wybranie folderu gdzie chcemy zainstalować aplikację, klikamy „Install”, później „Close”. Po drodze jeszcze poproszony zostaniesz o możliwość zainstalowania sterowników do różnych płytek arduino, możesz się zgodzić choć my będziemy potrzebować osobnych sterowników dostępnych np. na stronie <http://robomaniak.pl/hardedu> , plik „ch341ser.zip”. Pobieramy plik, rozpakowujemy i instalujemy poprzez plik setup.exe.



Po uruchomieniu z prawami administratora powinniśmy widzieć powyższe okno. Instalacja jest banalna, wystarczy kliknąć w przycisk „INSTALL”.

2.2. Konfiguracja Arduino IDE

Jak już przygotowaliśmy Arduino IDE do pracy przydałoby się wgrać pierwszy program. Uruchamiamy aplikację ze skrótu na pulpicie i powinniśmy zobaczyć poniższe okno.



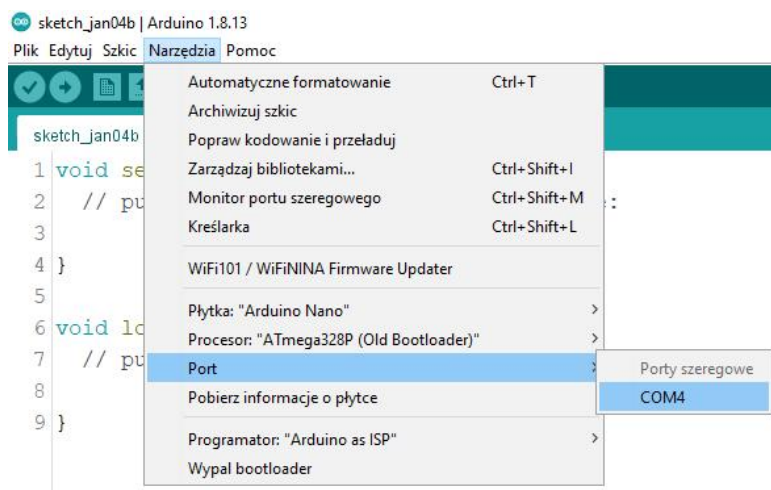
Zacznijmy od podstawowych informacji. Na poniższym zdjęciu przedstawiłem zbliżenie na którym możemy zauważyć sekcje menu aplikacji.





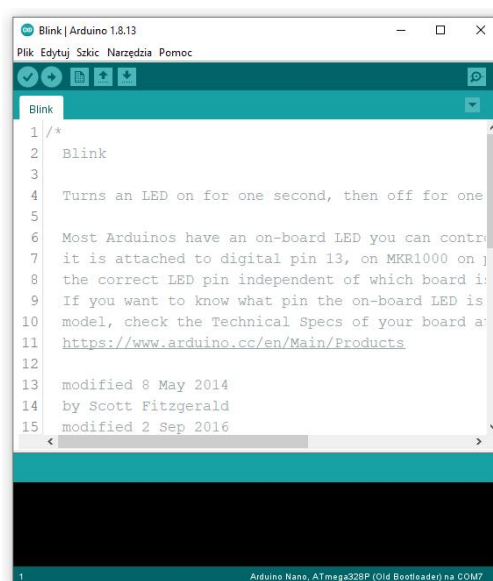
```
sketch_jan04b $
1 void setup() {
2   // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }
```

Skupimy się na podstawowych opcjach, sekcja „Narzędzia”, następnie „Płytką”, „Arduino AVR Boards”, wybieramy Arduino Nano. Kolejno sekcja „Narzędzia”, opcja „Procesor”, wybieramy „ATmega328P” lub „ATmega328P (Old Bootloader)”. W tym miejscu wyjaśnię pokrótce czym jest bootloader, jest to aplikacja znajdująca na płytce arduino która służy do uruchomienia twojego programu. Mając to wszystko ustawione podłączmy nasze arduino nano do komputera i wybierzmy odpowiedni port COM. Znowu wracamy do sekcji „Narzędzia”, „Port” i wybieramy dostępny COM. W moim przypadku jest to COM4 ale u Ciebie może być to inny, ważne że nie będzie to raczej COM1, a jakiś powyżej COM2.

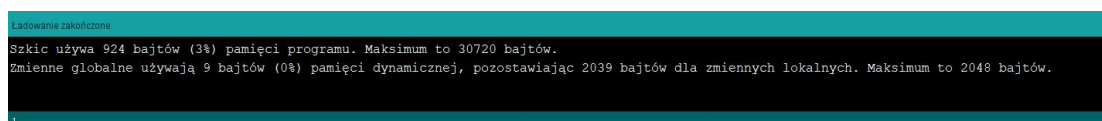


2.3. Pierwszy program

Przyszła czas na sprawdzenie czy wszystkie powyższe kroki wykonaliśmy prawidłowo i jesteśmy w stanie wgrać na naszą płytkę arduino pierwszy program. Sugeruję aby sprawdzić to jednym z dostępnych przykładów jakie są dostępne w aplikacji Arduino IDE. W celu wgrania szkicu należy w menu wybrać „Plik”, następnie „Przykłady”, kolejno „01. Basic” i „Blink”. Powinno otworzyć się nam nowe okno programu z gotowym szkicem jak na poniższym zdjęciu.



W tym momencie powinniśmy być już gotowi do wgrania pierwszego szkicu na nasze arduino. W celu wgrania programu należy wybrać przycisk oznaczony poziomą strzałką skierowaną w prawo lub użyć skrótu klawiszy **CTRL + U**. Po wgraniu szkicu na arduino powinniśmy uzyskać poniższą wiadomość.



Jeśli otrzymamy inny komunikat, sprawdź czy masz wybraną prawidłową płytkę arduino, procesor oraz port. Jeśli nie masz żadnego portu do wyboru sprawdź czy płytkę arduino jest dobrze podłączona przez przewód USB lub ponownie wgraj sterownika do CH341. Odłącz i podłącz arduino do komputera.

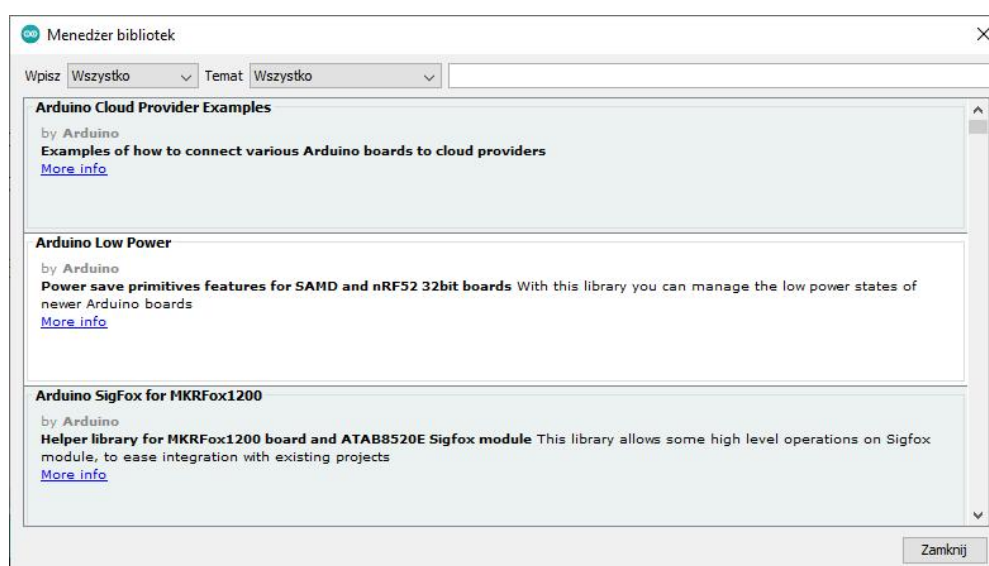
2.4. Ciekawostki o Arduino IDE

Jedną z bardzo przydatnych rzeczy przy programowaniu to odpowiednie formatowanie tego co piszemy. Estetycznie napisany program, bardzo dobrze się czyta. Arduino IDE posiada narzędzie do auto formatowania naszego tekstu. Wystarczy w dowolnym momencie pisania szkicu naciskając skrót klawiszowy **CTRL + T** aby nasz program ustawił się zgodnie nawiasami klamrowymi, odpowiednio wstał gdzie trzeba spację. Nasz kod zacznie wyglądać bardzo estetycznie, a my na pewno lepiej będziemy mogli się w nim odnaleźć.

Kolejnym udogodnieniem w trakcie pisania jest wstawienie z lewej strony okna numeracji wierszy. W celu dodania numerów należy wejść *“Preferencje”*, możemy je znaleźć w menu, sekcja *“Plik”*. Następnie w karcie *“Ustawienia”* zaznaczyć opcję *“Wyświetl numery linii”*.

Arduino IDE jak już pokazałem w poprzednim rozdziale to nie tylko edytor kodu ale też narzędzie posiadające wbudowane przykłady wykorzystania poszczególnych bibliotek. Czym są biblioteki wspomnę jeszcze w jednym z kolejnych rozdziałów. Na ten moment dowiedzmy się że wchodząc w sekcję *“Plik”*, następnie *“Przykłady”*, mamy dostęp do przykładowych szkiców z danych bibliotek gdzie możemy podejrzeć jak autor wykorzystał daną funkcjonalność.

Jak już wspominałem o bibliotekach to należy dodać jeszcze że z poziomu Arduino IDE możemy zarządzać nimi. Nie trzeba w większości przypadków pobierać ręcznie ich z internetu i wklejać do folderu *“libraries”* znajdującego się w *“Dokumentach”*, folder *“Arduino”*. W celu łatwiejszego zarządzania twórcy przygotowali nam mechanizm który możemy wywołać skrótem klawiszy CTRL + SHIFT + I lub poprzez menu *“Szkic”*, *“Dołącz bibliotekę”*, *“Zarządzaj bibliotekami”*.



Na górze po prawej okna wpisujemy nazwę interesującej nas biblioteki, następnie wybieramy ją i instalujemy.

3. Podstawy programowania płytek Arduino

Jak wspomniałem w jednym z pierwszych rozdziałów płytki arduino mogą być programowane w języku C/C++ ale nie tylko. W internecie możemy znaleźć aplikacje które pozwalają pisać programy w języku JAVA, Python czy Scratch. W tej książce chcę używać Arduino IDE i w oficjalnej wersji jest to edytor kodu C/C++.

W niniejszym rozdziale postaram się Tobie wyjaśnić podstawową składnię języka C++ oraz przedstawić kilka przykładowych szkiców. Więcej o programowaniu postaram się przedstawić w formie video na platformie youtube pod adresem <https://robomaniak.pl/programowanie/>.

3.1. Podstawowy język C++

Język C++ wywodzi się z języka C stworzonego w roku 1972. W okolicy roku 1983 oficjalnie został użyta rozbudowana wersja C czyli C++. Za twórcę języka C++ uznaje się duńskiego profesora Bjarna Stroustrupa („Bjarne Stroustrup”). Ogólnie rzecz ujmując język C++ wprowadza wiele rozszerzeń w tym programowanie obiektowe ale powstaje maksymalnie zgodny z językiem C. Język C++ to połączenie języka strukturalnego i obiektowego. Język strukturalny czyli taki w którym króluje zasada „zrób najpierw to, a potem tamto”, mamy hierarchiczne dzielenie kodu na bloki, struktury. Język obiektowy czyli taki w którym program jest zbiorem komunikujących się ze sobą obiektów, jednostek zawierających określone dane i potrafiących wykonywać na nich określone operacje. Posiadający mechanizmy dziedziczenia i wiele innych. Reasumując C++ jest językiem hybrydowym, umożliwia programowanie strukturalne, jak i programowanie orientowane obiektowo.

Zajmijmy się teraz składnią języka C++ czyli jego semantyką która definiuje precyzyjnie znaczenie poszczególnych symboli oraz ich funkcji.

Podstawowe elementy języka programowania to typy danych, zmienne, operatory, instrukcje, funkcje, pętle, tablice, struktury, klasy.



3.1.1. Typy danych

Ogólnie rzecz ujmując w języku C++ musimy zdefiniować typ danych jakie chcemy używać w celu zarezerwowania pamięci, miejsca na przechowywanie informacji, wartości. Wyróżniamy następujące typy danych:

- char - typ znakowy,
- bool - typ logiczny,
- int - liczba całkowita,
- short - liczby całkowite krótkie,
- long - liczby całkowite długie,
- float - liczba zmiennoprzecinkowa (rzeczywista),
- double - liczby zmiennoprzecinkowe podwójnej precyzji,

W poniższej tabelce podstawowe typy danych:

Typ	Wielkość w bajtach	Zakres liczbowy
bool	1	true lub false
char	1	typ znakowy
int	2 lub 4	liczby całkowite o takim samym zakresie jak short albo long
short	2	małe liczby całkowite z zakresu -32 768 do 32 767 ze znakiem albo od 0 do 65 535 bez znaku
long	4	duże liczby całkowite z zakresu -2 147 483 648 do 2 147 483 647 albo od 0 do 4 294 967 295
float	4	liczby zmiennoprzecinkowe
unsigned long	8	liczby zmiennoprzecinkowe podwójnej precyzji
double	8	liczby zmiennoprzecinkowe podwójnej precyzji
long double	8	liczby zmiennoprzecinkowe rozszerzonej podwójnej precyzji

3.1.2. Zmienne

Zmienna, jak sama nazwa wskazuje będzie się zmieniać w trakcie programu. Zmienna to pewien stosunkowo mały obszar w pamięci, w którym możemy przechowywać dane różnego typu np. liczby całkowite, liczby rzeczywiste (zmiennoprzecinkowe), znak, tekst oraz kilka innych wartości, które będą nas w



przyszłości interesowały. Nie można jednak wszystkiego zapisywać do jednej zmiennej. Każda zmienna ma swoje przeznaczenie, wielkość i właściwości. Na zmiennych liczbowych możemy wykonywać operacje matematyczne, w innych z kolei możemy przechowywać tekst.

Aby użyć zmiennej należy najpierw ją zadeklarować, czyli podać jej typ np. chcąc zadeklarować zmienną o nazwie „*styk*” typu całkowitego użyjemy składni „*int styk;*”. Dobrą praktyką jest deklarowanie nazwa w zapisie wielbłądźm. Polega to na zapisywaniu nazwa składających się z kilku wyrazów bez spacji i z każdym wyrazem zaczynamy od dużej litery prócz zapisu pierwszego wyrazu np. „*int liczbaDuza;*”. Nazwy zmiennych powinny opisywać przeznaczenie zmiennej w programie. Nazwa zmiennej może zawierać jedynie litery, cyfry, oraz znak podkreślenia, nie może zawierać spacji i nie może zaczynać się od cyfry. Pisząc program nie używamy polskich znaków diakrytycznych przy nazwach zmiennych, funkcji np. zamiast *ć* używamy *c*. W nazwach rozróżniana jest wielkość liter i nie mogą one pokrywać się ze słowami kluczowymi języka c++ np. *return*, *main*, *loop* itp.

Zmienną możemy zadeklarować czyli tylko nadać jej typ i nazwę lub zadeklarować i inicjować czyli podać typ nazwę i wartość np. „*int zmiennaA = 5;*”. Zmienne dzielimy ze względu na ich zasięg. Zmienna może być lokalna lub globalna.

```
1 int zmiennaA;
2 void setup() {
3     int zmiennaB;
4 }
5
6 void loop() {
7     int zmiennaC;
8 }
```

Na powyższym rysunku zadeklarowaliśmy trzy zmienne o nazwie: *zmiennaA*, *zmiennaB*, *zmiennaC*. Wszystkie są typu *int*, liczby całkowite. Zmienna o nazwie *zmiennaA* jest zmienną globalną. Można korzystać z tej zmiennej w każdym miejscu naszego programu. Zmienna o nazwie *zmiennaB* jest zmienną lokalną i można z niej korzystać tylko w funkcji o nazwie *setup*. Zmienna o nazwie *zmiennaC* jest zmienną lokalną i można z niej korzystać tylko w funkcji o nazwie *loop*. Więcej o zasięgach zmiennych powiedziałem w materiale video w serwisie youtube http://robomaniak.pl/ebook_zmienne/.

3.1.3. Operatory

Operatory możemy podzielić na trzy rodzaje: arytmetyczne, relacyjne, logiczne. Operatory arytmetyczne to najprościej mówiąc takie jakie wykorzystujemy w operacjach matematycznych tj. dodawanie, odejmowanie, dzielenie itd.

Operator	Wykorzystanie	Wynik	Opis
+	1+1	2	dodawanie
-	1-1	0	odejmowanie
*	2*3	6	mnożenie
/	4/2	2	Dzielenie
%	6%4	2	Dzielenie z resztą

Następnie operatory relacyjne, zapewne też znane wam z lekcji matematyki ale są lekkie zmiany oraz różnice:

Operator	Przykład	Opis
==	X == Y	porównanie, sprawdź czy X jest równe Y
!=	X != Y	zaprzeczenie porównania, sprawdź czy X jest różne od Y
<	X < Y	mniejszość, sprawdź czy X jest mniejsze od Y
>	X > Y	większość, sprawdź czy X jest większe od Y
<=	X <= Y	mniejsze lub równe, sprawdź czy X jest mniejsze lub równe Y
>=	X >= Y	większe lub równe, sprawdź czy X jest większe lub równe Y
++	X++	inkrementacja, zwiększ wartość X o 1
--	X--	dekrementacja, zmniejsz wartość o 1
+=	X += 2	zwiększenie, dodaj 2 do wartości X
-=	X -= 2	zmniejszenie, odejmij 2 od wartości X
*=	X *= 2	mnożnik, pomnóż przez 2 wartość X
/=	X /= 2	dzielnik, podziel przez 2 wartość X

X i Y to nazwy zmiennych.

Na końcu powiedzmy sobie o dwóch operatorach logicznych. Będziemy często używać ich przy instrukcjach czy pętlach o czym później.

Operator	Alternatywny operator	Przykład	Opis
&&	and	X < 1 && Y > 2 X < 1 and Y > 2	logiczne „i”, następującą instrukcję należy przetłumaczyć następująco: X jest mniejsze od 1 i Y jest większe od 2
	or	X < 1 Y > 2 X < 1 or Y > 2	logiczne „lub”, następującą instrukcję należy przetłumaczyć następująco: X jest mniejszy od 1 lub Y jest większe od 2

O operatorach logicznych dowiesz się więcej w kolejnym rozdziale.

3.1.4. Instrukcje warunkowe

Podczas pisania naszego kodu często musimy dokonywać wyborów np. porównać pewne wartości aby wykonać różne operacje. Do tego celu wykorzystujemy instrukcję warunkową *if*.

```
1 void setup() {
2   pinMode(2, OUTPUT);
3 }
4
5 void loop() {
6   int nowy = 1;
7   int stary = 2;
8   if (nowy > stary) {
9     digitalWrite(2, HIGH);
10  }
11 }
```

W powyższym przykładzie użyliśmy instrukcji warunkowej która porównuje wartości dwóch zmiennych. Zmienna *nowy* o wartości 1 oraz zmienna o nazwie *stary* o wartości 2. Wewnątrz instrukcji warunkowej mamy wykonanie funkcji *digitalWrite*. Niniejsza funkcja może wykonać się tylko wtedy jak wartość zmiennej *nowy* będzie większa od wartości zmiennej *stary*. Przeanalizujmy kod, na początku zamiast zmiennych podajmy wartości: *if (1 > 2) { digitalWrite(2, HIGH); }*, *jeśli 1 jest większe od 2 to wykonaj funkcję digitalWrite(2, HIGH);* . Niestety wykonując powyższą instrukcję warunkową otrzymaliśmy tzw. nieprawdę, funkcja *digitalWrite* nie wykona się. Poniżej poprawimy kod w taki sposób aby mogła wykonać się funkcja *digitalWrite*.

```
1 void setup() {
2   pinMode(2, OUTPUT);
3 }
4
5 void loop() {
6   int nowy = 4;
7   int stary = 2;
8   if (nowy > stary) {
9     digitalWrite(2, HIGH);
10  }
11 }
```

Instrukcja warunkowa *if* może zostać użyta na kilka sposobów.

Instrukcja warunkowa	Opis
<code>if(warunek){ instrukcja1; instrukcja2; }</code>	Jeśli <i>warunek</i> jest prawdziwy to wykonaj <i>instrukcja1</i> , <i>instrukcja2</i> .
<code>if(warunek){ instrukcja1; } else { instrukcja2; }</code>	Jeśli <i>warunek</i> jest prawdziwy to wykonaj <i>instrukcja1</i> w przeciwnym wypadku wykonaj <i>instrukcja2</i> .
<code>if(warunek1){ instrukcja1; } else if(warunek2){ instrukcja2; } else { instrukcja3; }</code>	Jeśli <i>warunek1</i> jest prawdziwy wykonaj <i>instrukcja1</i> w przeciwnym wypadku sprawdź <i>warunek2</i> . Jeśli <i>warunek2</i> jest prawdziwy wykonaj <i>instrukcja2</i> w przeciwnym wypadku wykonaj <i>instrukcja3</i> . Instrukcje <i>else if</i> można dodawać w „nieskończoność”. Wystąpienie słowa kluczowego <i>else</i> może wystąpić tylko na końcu stworzonej instrukcji warunkowej.

Instrukcja warunkowa jest jednym z miejsc gdzie możemy wykorzystać operatory logiczne. Przeanalizujemy poniższy szkic.

```

1 void setup() {
2   pinMode(2, OUTPUT);
3 }
4
5 void loop() {
6   int nowy = 4;
7   int stary = 2;
8   if (nowy > 3 && stary <= 2) {
9     digitalWrite(2, HIGH);
10  }
11 }

```

W linii 8 mamy instrukcję warunkową w której mamy następujący warunek: *nowy > 3 && stary <= 2*, jeśli wartość zmiennej *nowy* jest większa od 2 i wartość zmiennej *stary* jest mniejsza lub równa 2 to wykonaj *digitalWrite(2, HIGH)*. W powyższym warunku musimy uzyskać dwa porównania prawdziwe, czyli wartość zmiennej *nowy* musi być większa oraz wartość zmiennej *stary* musi być mniejsze bądź równe. Zakładając że wartości są takie jak inicjowaliśmy w linii 6 oraz 7 to cały warunek jest prawdziwy i funkcja *digitalWrite* się wykona.

Jeśli zmienimy operator na logiczne „lub” uzyskamy następujący warunek: *nowy > 3 || stary <= 2*, jeśli wartość zmiennej *nowy* jest większa od 2 lub wartość

zmiennej *stary* jest mniejsza lub równa 2 to wykonaj `digitalWrite(2, HIGH)`. W powyższym warunku wystarczy uzyskać jedno porównania prawdziwe, czyli wartość zmiennej *nowy* musi być większa lub wartość zmiennej *stary* musi być mniejsze bądź równe. Zakładając że wartości są takie jak inicjowaliśmy w linii 6 oraz 7 to cały warunek jest prawdziwy i funkcja `digitalWrite` się wykona. Jeśli zmienimy wartość zmiennej *nowy* na 1, fragment warunku ze zmienną *nowy* jest nieprawdziwy ale nie zmieni to wyniku całego warunku ponieważ wystarczy tylko aby jeden z fragmentów został spełniony aby cały warunek był prawdziwy.

Za pomocą instrukcji warunkowej `if` możemy określić dokładnie co ma się wydarzyć w zależności od stanu jednej lub kilku zmiennych. Instrukcja `if` daje nam pełną kontrolę nad przebiegiem programu. Kolejną instrukcją warunkową jest tzw. instrukcja wielokrotnego wyboru `switch`. Działanie instrukcji `switch` jest zupełnie inne. W przypadku niej możemy wykonywać decyzje tylko i wyłącznie na podstawie wartości jednej zmiennej. Możliwości instrukcji `switch` są nieporównywalnie mniejsze, jednak używanie jej w niektórych przypadkach jest znacznie korzystniejsze dla szybkości działania programu i estetyki kodu niż użycie instrukcji `if`. Stąd też warto przyrzeć się bliżej niniejszej instrukcji. Zaczniemy od poznania składni instrukcji `switch` w tzw. pseudokodzie:

```
1  switch ( zmienna )
2  {
3      case wartosc1:
4          instrukcja1();
5          break;
6
7      case wartosc2:
8          instrukcja2();
9          break;
10
11     case wartoscN:
12         instrukcjaN();
13         break;
14
15     default:
16         instrukcjaDomyslna();
17         break;
18 }
```

Instrukcje zaczynamy od użycia słowa kluczowego `switch` w nawiasach półokrągłym podajemy nazwę zmiennej na której chcemy pracować. Kolejnym krokiem jest użycie słowa kluczowego `case` oraz podajemy wartość, musi być to liczba całkowita. W kolejnym kroku między znakiem „:”, a słowem kluczowym `break`; wpisujemy instrukcje które mają się wykonać jeśli wartość zmiennej *zmienna* będzie równać się wartości. Przejdźmy do faktycznego przykładu.

```

1 void setup() {
2   pinMode(2, OUTPUT);
3   pinMode(3, OUTPUT);
4   pinMode(4, OUTPUT);
5   pinMode(5, OUTPUT);
6 }
7 void loop() {
8   int losowa = random(1, 10);
9   switch ( losowa )
10  {
11    case 1:
12      digitalWrite(2, HIGH);
13      break;
14
15    case 2:
16      digitalWrite(3, HIGH);
17      break;
18
19    case 3:
20      digitalWrite(4, HIGH);
21      break;
22
23    default:
24      digitalWrite(5, HIGH);
25      break;
26  }
27 }

```

W powyższym programie w linii 8 stworzyliśmy zmienną o nazwie *losowa* do której wpisana zostanie wartość losowa z przedziału od 1 do 10. W instrukcji *switch* sprawdzamy wartość zmiennej *losowa*. Jeśli wartość zmiennej *losowa* równa się 1 wykona się funkcja *digitalWrite(2, HIGH)*; W przypadku jeśli wartość równa się 2 to wykona się funkcja *digitalWrite(3, HIGH)*; itd. Jeśli wartość zmiennej będzie większa od 3 to wykona się funkcja *digitalWrite(5, HIGH)*; Używając instrukcji *switch* należy pamiętać aby użyć słowa kluczowego *break*. Słowo to oznacza wykonanie przerwania obecnej instrukcji i musi być użyte zanim wywołamy kolejny *case*.

Więcej o instrukcja warunkowej *if* powiedziałem w materiale video w serwisie youtube http://robomaniak.pl/ebook_if/ , natomiast o instrukcji *switch* w video http://robomaniak.pl/ebook_switch/ .

3.1.5. Funkcje

Temat funkcji jest bardzo rozległy dlatego na ten moment będę chciał przedstawić tobie podstawowe informacje na temat. Funkcja jest to fragment programu, któremu nadajemy nazwę i który możemy wykonać poprzez podanie jego nazwy. Funkcja musi mieć swój typ, podobnie jak było to ze zmiennymi. Najbardziej popularnym typem funkcji jest *void* i *int*. Typ *void* oznacza funkcję która nie zwraca wartości. Funkcje innego typu np. *int* oznacza funkcje zwracającą wartość typu *int*. Możliwe że na ten moment jest to trochę niezrozumiałe ale za chwilę postaram się to wyjaśnić na przykładach. Budowa funkcji w pseudokodzie:



```

1 typ nazwa() {
2     instrukcja1;
3     instrukcja2;
4 }

```

Każda funkcja musi posiadać nawiasy półokrągłe oraz nawiasy klamrowe określające zakres funkcji. Przejdźmy do przykładu:

```

1 void setup() {
2     pinMode(2, OUTPUT);
3     pinMode(3, OUTPUT);
4 }
5 void naszaFunkcja(int a, int b) {
6     digitalWrite(a, HIGH);
7     digitalWrite(b, HIGH);
8 }
9 void loop() {
10    naszaFunkcja(2, 3);
11 }

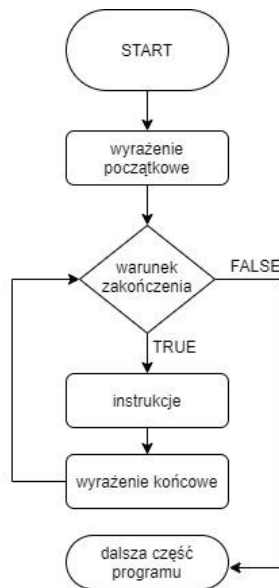
```

W powyższym kodzie stworzyliśmy funkcję o nazwie *naszaFunkcja*, typ funkcji to *void* czyli taki który nie zwraca żadnej wartości. W nawiasach półokrągłych zdefiniowaliśmy dwa argumenty, czyli takie zmienne lokalne które otrzymają wartość w trakcie jej inicjacji funkcji. W momencie inicjacji funkcji zostaną wykonane dwie instrukcje *digitalWrite(a, HIGH);* oraz *digitalWrite(b, HIGH);*. W linii 10 inicjujemy funkcję, wywołujemy ją poprzez podanie jej nazwy. W nawiasach półokrągłych podajemy wartości naszych atrybutów, zmiennych. Funkcja która zdefiniowaliśmy wykona się dopiero w naszym szkicu w miejscu kiedy ją wywołamy, możemy funkcję wywoływać wielokrotnie z różnymi atrybutami.

Więcej o funkcjach powiedziałem w materiale video w serwisie youtube http://robomaniak.pl/ebook_funkcje/

3.1.6. Pętle

Pętla to fragment kodu programu ograniczony zasięgiem, powtarzany zgodnie z warunkiem działania pętli. Pętle pozwalają cyklicznie wykonywać ciąg instrukcji. W języku C++ wyróżniamy trzy rodzaje pętli: *for*, *while*, *do ... while*. Zaczniemy od poznania pętli *for*. Składnia niniejszej pętli wygląda następująco:

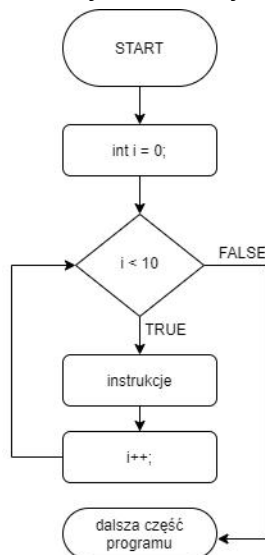


```

for( wyrażenie początkowe; warunek zakończenia; wyrażenie końcowe){
    instrukcje;
}

```

Wyrażenie początkowe czyli deklaracja i inicjacja zmiennej na której będzie pracowała pętla. Warunek zakończenia pętli, przeważnie porównanie wartości zapisanej w zmiennej utworzonej w wyrażeniu początkowym do jakiejś wartości liczbowej. Wyrażenie końcowe czyli to co wykona się na końcu cyklu pętli przeważnie jest to inkrementacja lub dekrementacja zmiennej na której pracuje pętla. Przykład:



```

for(int i = 0; i < 10; i++){
    instrukcje;
}

```

Niniejsza pętla *for* na początku swojego działania stworzyła zmienną o nazwie „i” i przypisanie wartości początkowej 0. W kolej części mamy warunek który sprawdza czy zmienna „i” jest mniejsza od 10. Jeśli zmienna „i” jest mniejsza od 10 wtedy wykonujemy instrukcje znajdujące się w pętli. Ostatnim etapem działania pętli jest zwiększenie o 1 zmiennej „i”. W nowym cyklu pętli sprawdzany jest warunek, jeśli został spełniony, wykonywane są instrukcje. Pętla zakończy swoje działanie kiedy zmienna „i” będzie równa 10, ponieważ liczba 10 nie jest mniejsza od 10, a jest równa.

Kolejną pętlą jaką poznamy jest *while*. Składnia:

```
while( warunek){  
    instrukcje;  
}
```

Pętle deklarujemy zaczynając od słowa kluczowego *while*, w nawiasie półokrągłych ustawiamy warunek działania pętli, a w nawiasach klamrowych instrukcje które chcemy wykonać. Przykład:

```
int i = 0;  
while( i < 10 ){  
    instrukcje;  
    i++;  
}
```

Na początku musimy mieć jakąś zmienną lub inną instrukcję w celu przygotowania warunku działania pętli. W powyższym przykładzie stworzona została zmienna „i” z wartością początkową 0. W kolejnym kroku w nawiasach półokrągłych stworzony został warunek działania pętli, wartość zmiennej „i” musi być mniejsza od 10. Jeśli ten warunek jest spełniony to może wykonać się cykl naszej pętli. Jako że jedna z instrukcji w pętli jest to inkrementacja zmiennej „i” to w kolejnym cyklu wartość zmiennej będzie zwiększona o 1. Pętla zakończy się kiedy zmienna „i” będzie większa, bądź równa 10.

Pętla *for* i *while* zanim wykonają z pierwszy cykl działania najpierw sprawdzają warunek, jeśli warunek nie został spełniony pętla nie rozpocznie się i nie zostanie wykonana żadna instrukcja znajdująca się w niej.

Wyjątkową pętlą jest *do ... while*. Składnia:

```
int i = 0;
do{
    instrukcje;
    i++;
}while( i < 10 );
```

Zapis podobny do pętli *while* ale działanie już nie do końca takie same ale to już opiszemy sobie na przykładzie.

```
int i = 0;
do{
    instrukcje;
    i++;
}while( i < 10 );
```

Pętla *do ... while* najpierw wykonuje instrukcje znajdujące się wewnątrz, a dopiero później sprawdza czy warunek jej działania pozwala jej na to. W niniejszym przykładzie pętla działa identycznie jak pętla *while* ale zmodyfikujmy go lekko.

```
int i = 10;
do{
    instrukcje;
    i++;
}while( i < 10 );
```

Jak możemy zobaczyć zmianie uległa wartość zmiennej „i”, teraz wartość początkowa równa się 10. Czy w takim razie wykona się choć jeden cykl pętli? Tak, ponieważ pętla *do ... while* najpierw wykona instrukcje znajdujące się w jej zasięgu a dopiero później sprawdzi warunek. Pętla *while* w takim przypadku nie została by wywołana.

Więcej o pętlach powiedziałem w materiale video w serwisie youtube http://robomaniak.pl/ebook_petle/

3.1.7. Tablice

Poznaliśmy już zmienne czyli takie pojedyncze komórki o typie danych które posiadały swoje indywidualne nazwy i własne wartości. Teraz wyobraźmy sobie że możemy połączyć te wszystkie komórki danego typu danych pod jedną nazwą. To są właśnie tablice. Tablice dzielimy na jednowymiarowe i wielowymiarowe. Zaczniemy od tablic jednowymiarowych oraz ich składni:

```
int szkielet[10];
```

Powyższy fragment oznacza że zadeklarowaliśmy tablice o nazwie *szkielet* jest to tablica składająca się z 10 elementów . Jeśli chcemy zadeklarować i inicjalizować tablice należy wpisać do niej poszczególne elementy np.

```
int szkielet[5] = {1,2,3,4,5};
```

W celu wywołania zawartości elementu tablicy należy odnieść się do numeru komórki np. *szkielet[1]* ma wartość 2, *szkielet[4]* ma wartość 5. Dlaczego tak? Elementy tablicy liczone są od komórki 0 do komórki n-1 czyli w naszym przykładzie od 0 do 4. Kolejnym rodzajem są tablice wielowymiarowe, najczęściej dwuwymiarowe. Inicjalizacja tablicy dwuwymiarowej wygląda następująco:

```
int glowa[2][3];
```

Deklaracja niniejszej tablicy może wyglądać następująco:

```
int glowa[2][3] = {{1,2,3},{4,5,6}};
```

Tworząc tablicę możemy ją najpierw deklarować, a przypisanie wartości do komórek możemy zrobić w następujący sposób:

```
glowa[1][2] = 5;  
szkielet[3] = 4;
```

Pamiętajcie tylko że wpisanie wartości jak powyżej może nastąpić tylko wewnątrz jakieś funkcji nawet jak tablica ma zasięg globalny.

Więcej o tablicach powiedziałem w materiale video w serwisie youtube http://robomaniak.pl/ebook_tablice/

3.1.8. Struktura

Struktura to taki obiekt, typ złożony, który może składać się z zmiennych i tablic. Struktura to takie przejście między zmienną, a klasą. O klasach powiemy już w kolejnym rozdziale.



Składnia struktury w języku C++:

```
struct nazwa{  
    zmienne;  
};
```

Strukturę rozpoczynamy słowem kluczowym *struct*, następnie podajemy jej nazwę własną, a w nawiasach klamrowych jej typy danych. Przykład:

```
struct led{  
    int port;  
    int delay;  
    bool stan = HIGH;  
};
```

Powyższy zapis stworzył w naszym programie nowy typ danych o nazwie *led*, jest to typ dla struktur. W celu stworzenia struktury należy wykonać następującą instrukcję: *struct led lampki[5]*; Oznacza to że została stworzona struktura o nazwie *lampki* typu *led*, składająca się z 5 elementów. Jeśli chcemy przypisać wartości do elementów tej struktury wykonujemy przykładowe instrukcje:

```
lampki[1].port = 2;  
lampki[1].delay = 500;
```

Taki zapis pozwolił nam przypisać do elementu 1 struktury *lampki* przypisać wartość do zmiennej *port* liczbę 2, zmiennej *delay* wartość 500. Zmienna *stan* domyślnie została ustawiona na HIGH.

Więcej o strukturach powiedziałem w materiale video w serwisie youtube http://robomaniak.pl/ebook_struktura/

3.1.9. Słowa kluczowe

Zacznijmy od podstawowej definicji, słowami kluczowymi nazywamy wszystkie słowa w semantyce języka C++ które mają spełniać pewne funkcje języka. Nie mamy tutaj na myśli funkcji w rozumieniu fragmentu programu ale operacji zaproponowanych przez twórcę języka C++.

Wspomnijmy o słowach kluczowych jakie poznaliśmy do tej pory np. *if*, *switch*, *break*, *case*, *while*, *for* itd. Jak widzimy są to różne słowa które mają jakieś znaczenie w naszym języku programowania. Omówmy sobie jeszcze kilka słów o jakich nie napisałem, a jakie są ważne dla nas.



Słowo kluczowe	Opis
return	Słowo używane w funkcjach typu innego niż void, na końcu funkcji w celu zwrócenia wartości. Przykład: <pre>int oblicz(int a, int b){ int wynik = a + b; return wynik; } void loop(){ int liczba = oblicz(1,2); }</pre> Wartość liczby wynik równa się 3.
break	Słowo znane z instrukcji <i>switch</i> ale można go użyć w celu zakończenia pętli przed jej warunkiem końca. Przykład: <pre>for(int i = 0; i < 10; i++){ if(i == 5) break; }</pre> Pętla <i>for</i> zakończy się jeśli wartość zmiennej „i” będzie równa 5.
continue	Słowo używane w pętlach oznaczające zakończenie danego cyklu pętli i przejście do kolejnego. Oznacza to że dany cykl pętli kończy się w momencie wywołania instrukcji <i>continue</i> i nie jest wykonywana żadna inna instrukcja poniżej. Przykład: <pre>for(int i = 0; i < 10; i++){ if(i == 5) continue; Serial.println(i); }</pre> W powyższej pętli za każdym razem prócz momentu kiedy wartość „i” równa się 5 zostanie wykonana metoda <i>println</i> na obiekcie <i>Serial</i>.

3.1.10. Struktura szkicu

Myślę że jesteśmy już gotowi aby przejść do napisania swojego pierwszego programu. W tym rozdziale mówimy sobie strukturę naszego szkicu. Podstawowy szkic wygląda następująco:



```

1 void setup() {
2   // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }

```

Prześledzimy sobie go linia po linii. W 1 linii mamy definicję funkcji *setup* typu *void*. Wcześniej omawialiśmy sobie że funkcje się definiuje i inicjuje. W tym wypadku nie mamy inicjacji funkcji. Więc jak to działa? Myśmy funkcje tylko definiowaliśmy. Wspominałem w jednym z pierwszych rozdziałów o czymś takim jak *bootloader*. Przenieśmy się na chwilę w realia komputera i systemu operacyjnego. Mając komputer mówimy o sprzęcie zamkniętym w obudowie, który dla nas użytkowników działa jedynie jeśli posiada system operacyjny i jesteśmy w stanie w tym systemie uruchamiać jakieś programy, pliki wykonawcze *exe*. Wróćmy teraz na poziom arduino. Arduino jest takim komputerem którego systemem operacyjnym jest *bootloader* a naszym programem *exe* jest szkic jaki piszemy i wgrywamy na naszą płytkę. Inaczej mówiąc funkcję *setup* jak i *loop* inicjalizuje nasz *bootloader*, a my przygotowujemy tylko ich definicję.

Inicjalizacja funkcji *setup* występuje tylko jeden raz w momencie kiedy uruchamiana jest płytka arduino, „wstaje system”. Funkcja *loop* wykonywana jest w formie pętli nieskończonej, nasz *bootloader* ciągle inicjuje funkcję *loop* co pozwala jej działać bez przerwy w pojedynczych cyklach.

W linii 2 i 7 mamy komentarze które od strony programu nie mają żadnego znaczenia jest to tylko informacja umieszczona dla programisty.

Znamy strukturę szkicu napiszmy program i go omówmy.

```

1 void setup() {
2   pinMode(2, OUTPUT);
3   pinMode(3, OUTPUT);
4 }
5 void naszaFunkcja(int a, int b) {
6   digitalWrite(a, HIGH);
7   digitalWrite(b, HIGH);
8 }
9 void loop() {
10  naszaFunkcja(2, 3);
11 }

```

Szkic ten był już widoczny wcześniej ale opiszmy go krok po kroku. W funkcji *setup* którą nazywamy funkcją konfiguracyjną wywołujemy dwie metody *pinMode*, których zadaniem jest ustawienie styków 2 i 3 na płytce arduino na tryb pracy wysyłania sygnału. W linii 5 definiujemy funkcję o nazwie *naszaFunkcja*, która przyjmuje dwa atrybuty, zmienne typu liczba całkowita o nazwie „a” i „b”. W funkcji *loop* wywołujemy, inicjalizujemy funkcję *naszaFunkcja* z atrybutem pierwszym

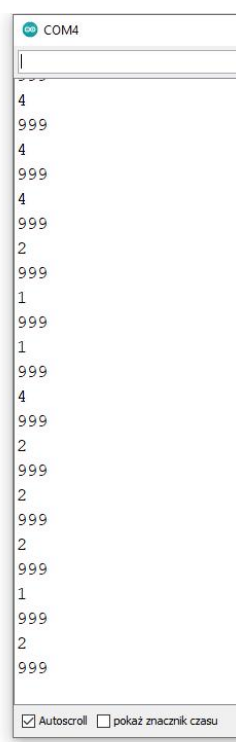
równym 2 oraz drugim równym 3. W tym momencie inicjalizujemy metody *digitalWrite* o atrybutach 2 i 3 oraz *HIGH*. W ten sposób na porcie 2 i 3 płytki arduino mamy napięcie 5V. Ważna kwestia która do tej pory nie została poruszona ale możliwe że już to zauważyłeś, zawsze na końcu instrukcji musisz zakończyć znakiem średnik, ; .

3.1.11. Klasy

Powoli wchodzimy w świat programowania obiektowego. Wszystko co do tej pory zostało omówione jest wiedzą podstawową. Ten rozdział należy do jednych z trudniejszych i musisz być pewnym że to co zostało poruszone do tej pory z języka programowania jest dla Ciebie zrozumiałe. Sugeruje poćwiczyć pisząc jakieś proste programy zanim zaczniesz czytać ten rozdział lub wrócić do niego w późniejszym etapie zapoznawania się z książką.

Postaram się pokazać klasę w podstawowej konfiguracji ale takiej aby pozwoliło tobie użyć jej w swoich programach. Przykład:

```
1 class bohater {
2   public:
3     int wiek;
4     int wyswietlSile();
5     bohater(int w );
6   private:
7     int sila = random(1, 5);
8 };
9
10 int bohater::wyswietlSile() {
11   return sila;
12 }
13
14 bohater::bohater( int w) {
15   wiek = w;
16 }
17
18 void setup() {
19   Serial.begin(9600);
20 }
21
22 void loop() {
23   bohater marian(999);
24   Serial.println(marian.wyswietlSile());
25   Serial.println(marian.wiek);
26   delay(200);
27 }
28
29
30
31
```



Powyższy fragment szkicu pokazuje w jaki sposób można zadeklarować klasę oraz utworzyć z niej obiekt. Linia 1 to definicja nazwy naszej klasy czyli *bohater*. Następnie wyróżniamy dwa słowa kluczowe *public* oraz *private*. Oznaczają one prawa dostępu z zewnątrz do danych zapisanych w klasie. Wyróżniamy jeszcze jedno słowo kluczowe *protected* które blokuje zmianę wartości np. Zmiennych bo ich

inicjalizacji. Słowo *public* oznacza że do danych poniżej mamy pełny dostęp odwołując się do obiektu, słowo *private* oznacza że dane mogą być zmieniane tylko przez funkcje wewnątrz klasy. W tym momencie nawiążmy do wcześniej poznanej struktury, *struct*, klasa jest taką strukturą która prócz danych może przechowywać też funkcje.

Linia 4 to definicja funkcji typu *int* o nazwie *wyswietlSila*. Pełna definicja tej funkcji znajduje się w linii od 10 do 12. W celu stworzenia funkcji należy jak w linii 10, określić jej typ, odnieść się do nazwy klasy oraz nazwy funkcji. Następnie można postępować jak ze zwykłą funkcją. Kolejną nowością jest tzw. Konstruktor obiektu, klasy. W linii 5 znajduje się nazwa „funkcji” identyczna jak nazwa klasy. Jest to właśnie konstruktor. Konstruktor pozwala nam stworzyć obiekt klasy już z wartościami. W naszym przypadku jest to jedynie podanie wartości dla zmiennej *wiek* co widzimy w linii od 14 do 16. Kolejna linia 7 to definicja i inicjalizacja zmiennej *sila* której wartość jest losowa z przedziału od 1 do 5. W tym momencie mamy zakończone omówienie klasy, przejdźmy do jej inicjalizacji, czyli stworzenia obiektu. W funkcji *loop*, która jak pamiętamy jest interpretowana jako pętla nieskończona w linii 23 tworzymy obiekt o nazwie *marian* i podajemy atrybut 999 co oznacza wartość zmiennej *wiek* dla tego obiektu. Cykl życia tego obiektu jest to jedno przejście instrukcji w funkcji *loop*. Jeden cykl tej pętli, kiedy rozpocznie się odnowa funkcja *loop* obiekt jest nadpisywany losowana jest np. nowa wartość zmiennej *sila*. W linii 24 i 25 odwołujemy się do obiektu *Serial* który pozwala mi wyświetlić w tzw. Monitorze portu szeregowego wartości zgodnie z podanymi atrybutami metody *println*. W oknie COM4 możemy zaobserwować wartości tych atrybutów. Najpierw wyświetlana jest wartość losowa zmiennej *sila*, następnie wartość zmiennej *wiek* która jest stała i wynosi 999.

3.2. Podstawy języka „arduino”

To co teraz napiszę w tym akapicie jest moim osobistymi przemyśleniem. Język programowania który de facto mamy do czynienia programując płytki arduino jest rozszerzeniem języka C++, dla mnie jest to framework oparty na języku C++. Framework to tzw. platforma programistyczna czyli środowisko z gotowymi bibliotekami, zawierające swoje funkcje, obiekty i mechanizmy. Od razu na myśl mi przychodzi właśnie aplikacja Arduino IDE. Wszystko w jednym miejscu, dodatkowo możemy rozbudować je o kolejne biblioteki. Czym są te biblioteki? Jest to zestaw funkcji i obiektów które pozwalają nam w prosty sposób obsłużyć pewne mechanizmy na płytce arduino jak i dodatkowe elementy elektroniczne które możemy podłączyć.



W tym rozdziale będę chciał opisać podstawowe funkcje i mechanizmy z którymi spotkamy się programując płytki arduino. Pisząc w przyszłości szkice na arduino proszę zapamiętać ten adres i skorzystaj z niego zawsze kiedy będziesz szukać pomocy. Programista powinien mieć w pamięci różne miejsca gdzie znajduje się opis funkcji, struktury języka <https://www.arduino.cc/reference/en/>

Nazwa	Przykład	Opis
STAŁE		
<i>HIGH</i>	<i>digitalWrite(2, HIGH);</i>	Wartość logiczna oznaczająca stan prawdy dla arduino jest to stan określający napięcie na styku większe niż 3V
<i>LOW</i>	<i>digitalWrite(2, LOW);</i>	Wartość logiczna oznaczająca stan nieprawdy dla arduino jest to stan określający napięcie na styku większe niż mniejsze niż 1.5V
<i>TRUE</i>	<i>if(a == TRUE)</i>	Wartość logiczna typu bool oznaczający stan prawdy.
<i>FALSE</i>	<i>if(a == FALSE)</i>	Wartość logiczna typu bool oznaczający stan nieprawdy.

Nazwa	Przykład	Opis
<i>Serial</i>	Serial jest nazwą obiektu który służy komunikacja szeregową między arduino a komputerem lub innymi urządzeniami. Do komunikacji szeregową służą piny TX/RX. W płytce Arduino Nano posiada tylko jeden port szeregowy. Obiekt Serial posiada szereg gotowych funkcji który pozwoli nam w prosty sposób skomunikować się z naszą płytka arduino. Poniżej najczęściej używane:	
	<i>Serial.begin(9600);</i>	Funkcja <i>begin()</i> ustawia szybkość transmisji danych w bitach na sekundę, przeważnie jest to 9600. Pamiętaj aby taka sama prędkość była ustawione w Arduino IDE w oknie „Monitor portu szeregowego”.
	<i>Serial.println(„Test”);</i>	Funkcja <i>println()</i> wysyła na port szeregowy wartość atrybutu, w naszym przykładzie płytka arduino wyśle na port szeregowy słowo <i>Test</i> w wierszu. Po słowie <i>Test</i> nastąpi znak końca wiersza.
	<i>Serial.print(„Test”);</i>	Funkcja <i>println()</i> wysyła na port szeregowy wartość atrybutu, w naszym przykładzie płytka arduino wyśle na port szeregowy słowo <i>Test</i> .
	<i>Serial.read();</i>	Funkcja <i>read()</i> odczytuje wartość wysłaną przez port szeregowy do płytki arduino. Wysyłkę możemy zrobić m.in. Arduino IDE w oknie „Monitor portu szeregowego”.

Nazwa	Przykład	Opis
TYPY DANYCH		
<i>String</i>	<i>String text = "Hello World"</i>	<i>String</i> jest klasą dzięki której możemy stworzyć obiekt. Nazwa text jest obiektem typu <i>String</i> . Wielkość liter w słowie <i>String</i> ma znaczenie. Klasa <i>String</i> posiada bardzo dużo funkcji oraz swoich operatorów. Wszystko znajdziesz w dokumentacji. W tej książce na klasie <i>String</i> będziemy jeszcze wykonywać ćwiczenia.
<i>word</i>	<i>word var = 65535;</i>	Typ danych który może przechować wartość od 0 do 65535.
FUNKCJE		
<i>pinMode(pin, trybPracy);</i>	<i>pinMode(2, OUTPUT);</i>	Funkcja która ustawia tryb pracy pojedynczego pinu. Numer pinu podajemy jako pierwszy argument, drugi to tryb pracy. Wyróżniamy trzy tryby pracy: OUTPUT - pin pracuje jako wyjście, INPUT - pin pracuje jako wejście, INPUT_PULLUP - pin pracuje jako wejście, a jego stan początkowy ustawiony jest na wysoki.
<i>digitalWrite(pin, stan);</i>	<i>digitalWrite(2, HIGH);</i>	Funkcja umożliwia ustawienie stanu pinu na HIGH (5V) lub LOW (0V).

Nazwa	Przykład	Opis
FUNKCJE		
<i>digitalRead(pin);</i>	<i>digitalRead(2);</i>	Funkcja umożliwia odczytanie wartości na pinie cyfrowym. Wartość można zapisać do zmiennej typu <i>bool</i> lub porównać w instrukcji warunkowej.
<i>analogWrite(pin, wartość);</i>	<i>analogWrite(200);</i>	Funkcja umożliwia zapis stanu na pinie analogowym lub PWM, o pinach PWM jeszcze sobie powiemy później. Zapis liczbowy z zakresu od 0 do 255 co odpowiada napięciu od 0 do 5V. Wartość 1 odpowiada około 20mV.
<i>analogRead(pin);</i>	<i>analogRead(A0);</i>	Funkcja umożliwia odczyt na pinie analogowym. Odczytana wartość może zostać zapisana do zmiennej lub użyta w instrukcji warunkowej. Odczytujemy zakres liczbowy od 0 do 1023. Odczytana wartość 1 odpowiada około 5mV.
<i>pulseIn(pin, stan);</i>	<i>pulseIn(7, HIGH);</i>	Funkcja umożliwia pomiar czasu trwania impulsu na pinie <i>INPUT</i> . Dzięki czemu możemy zwrócić czas w jakim stan danego pinu osiągnął zadaną wielkość. np. czas jaki na pinie 7 utrzymał się HIGH.

Nazwa	Przykład	Opis
FUNKCJE		
<i>millis();</i>	<i>unsigned long time = millis();</i>	Funkcja umożliwia odczytanie czasu w milisekundach od momentu włączenia arduino do „prądu”. Przy użyciu typu danych <i>unsigned long</i> możemy w teorii odczytać wartość do 50 dni, później czas liczy się od 0.
<i>micros();</i>	<i>unsigned long time = micros();</i>	Funkcja zwraca liczbę w mikrosekundach od momentu uruchomienia płytki arduino. Wartość przepełni się po około 70 minutach i zacznie liczyć od 0. Funkcja ta pozwala dokładniej liczyć czas.
<i>delay(czas);</i>	<i>delay(1000);</i>	Funkcja umożliwia zatrzymanie działania naszego programu przez zadany przez nas czas. Czas jaki podajemy w funkcji liczymy w milisekundach. np. 1000 oznacza 1 sekundę.
<i>delayMicroseconds(czas);</i>	<i>delayMicroseconds(50);</i>	Funkcja umożliwia zatrzymanie działania naszego programu przez czas wyrażony w mikrosekundach, np. 50 oznacza 0.00005 sekundy.
<i>min(a,b);</i>	<i>min(10,20);</i>	Funkcja zwraca mniejszą wartość.
<i>max(a,b);</i>	<i>max(10,20);</i>	Funkcja zwraca większą wartość.
<i>abs(wartość);</i>	<i>abs(-20);</i>	Funkcja zwraca wartość bezwzględną np. <i>abs(-20);</i> zwraca 20.

Nazwa	Przykład	Opis
FUNKCJE		
<i>constrain(x, a, b);</i>	<i>constrain(val, 10, 20);</i>	Funkcja zwróci wartość jeśli zawartość przykładowej zmiennej <i>val</i> mieści się w zakresie od 10 do 20.
<i>map(val, fL, fH, tL, tH);</i>	<i>map(x, 1, 50, 1, 200);</i>	Funkcja konwertuje zakres liczbowy. Wartość zmiennej <i>x</i> znajduje się w zakresie od 1 do 50, po użyciu funkcji wartość zostanie przeliczona do nowego zakresu od 1 do 200. Funkcja zwraca przekonwertowaną liczbę np. zmienna <i>x</i> równa się 25, jest z zakresu od 1 do 50, po konwersji w zakres od 1 do 200 będzie równać się 100.
<i>pow(x,p);</i>	<i>pow(2,3);</i>	Funkcja zwraca wartość podniesioną do potęgi np. liczba 2 do potęgi 3 równa się 8.
<i>sqr(x);</i>	<i>sqr(3);</i>	Funkcja zwraca wartość podniesioną do kwadratu np. liczba 3 do kwadratu równa się 9.
<i>randomSeed(x);</i>	<i>randomSeed(123);</i>	Funkcja konfiguracyjna, inicjująca generator liczb pseudolosowej.
<i>random(x);</i> <i>random(min, max);</i>	<i>random(300);</i> <i>random(10, 30);</i>	Funkcja zwracająca liczbę pseudolosową, np. z zakresu od 0 do 300 lub od 10 do 30.
<i>sizeof();</i>	<i>sizeof(tablica);</i>	Funkcja zwraca liczbę bajtów zajmowaną przez zmienną.

Nazwa	Przykład	Opis
FUNKCJE		
<code>int();</code>	<code>int(x);</code>	Jedna z funkcji konwertujących wartość zmiennej. Funkcja <code>int();</code> zwraca wartość przekonwertowaną na liczbę całkowitą.
<code>attachInterrupt();</code>	Funkcja umożliwiająca skorzystanie z przerwania. Arduino Nano posiada dwa piny do używania przerwań, pin:2 i 3. W skrócie przerwania pozwalają na automatyzację pewnych działań niezależną od tego co aktualnie wykonuje program. Więcej o przerwaniach napiszę później.	
<code>tone(pin, częstotliwość);</code> <code>tone(pin, częstotliwość, czas);</code>	<code>tone(3, 1000);</code> <code>tone(3, 1000, 30);</code>	Funkcja generuje falę prostokątną o określonej częstotliwości, jeśli nie określimy czasu generowania fali to musimy użyć funkcji <code>noTone(pin);</code> . Przykład: na pinie 3 generujemy falę o częstotliwości 1000Hz przez okres 30 milisekund.
Słowa kluczowe, elementy języka		
<code>#define</code>	<code>#define liczba 5</code>	Dyrektywa pre-procesorowa umożliwia zdefiniowanie nazwy, która znaleziona w szkicu zostanie przez kompilator zamieniona na odpowiednią wartość. Wartości dyrektyw nie mogą być zmieniane w trakcie pisania kodu podobnie jak wartości zmiennych. Dzięki dyrektywom możemy wykonywać makra oraz instrukcje warunkowe. Przykład: pod nazwą <i>liczba</i> kryje się wartość 5

Nazwa	Przykład	Opis
Słowa kluczowe, elementy języka		
<code>#include</code>	<code>#include <biblioteka.h></code>	Element pozwala dołączyć do naszego kodu plik zewnętrzny np. bibliotekę. Dzięki takiemu dołączeniu możemy korzystać z większej ilości funkcji, zmiennych, pozwala to też uporządkować nasz kod i podzielić go na części.
<code>/* */</code>	<code>/* komentarz 1 2 */</code>	Komentarz blokowy, wszystko co znajduje się między znacznikami <code>/*</code> i <code>*/</code> zostanie zignorowane przez kompilator.
<code>//</code>	<code>// komentarz</code>	Znacznik komentarza jednowierszowego, wszystko w wierszu naszego programu co zostanie zapisane po <code>//</code> zostanie zignorowane przez kompilator.
<code>!</code>	<code>bool a = TRUE; a = !a;</code>	Operator zaprzeczenia, zmienia wartość na przeciwność.
<code>goto</code>	<code>for(int i =0;i<5;i++){ int a = random(2, 6); If(a > 4){ goto skok; } } skok: digitalWrite(2,HIGH);</code>	Słowo <code>goto</code> pozwala przenieść przepływ naszego programu do oznaczonego punktu. np. jeśli w pętli zmienna <code>a</code> będzie większa od 5 to nasz wykonywanie naszego programu zostanie przeniesione do punktu oznaczonego <code>skok</code> :

4. Podstawy elektroniki

4.1. Podstawowe pojęcia

Elektronika swoje podstawy ma w fizyce, bez podstawowych pojęć trudno się odnaleźć. W tym rozdziale powiemy sobie o podstawowych pojęciach tj. napięcie, prąd, rezystancja, moc i masa.

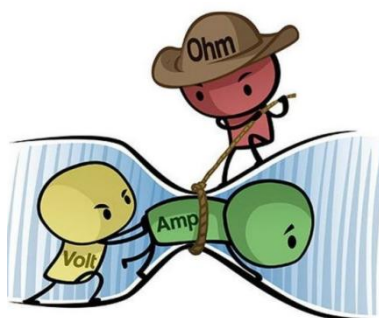
Na początek weźmy prąd czyli natężenie prądu. Teoretycznie każdy z nas słyszał o takim słowie jak prąd, na pewno jest one często używane. Definiując prąd mamy na myśli uporządkowany ruch ładunków elektrycznych, dodatnich i ujemnych. Nośnikami ładunków mogą być elektrony lub jony. Prąd zacznie płynąć, gdy umożliwimy nośnikom przepływ z jednego punktu do drugiego, pod warunkiem że mamy napięcie. Im więcej będzie nośników, które przepłyną przez połączenie w jednostce czasu, tym większy będzie prąd. Przepływ prądu musi być uporządkowany i przyjmuje się że kierunek płynięcia prądu jest od bieguna dodatniego do ujemnego. Jednostką natężenia prądu jest amper, oznaczony dużą literą A, natomiast skrótem jest duża litera I. Do pomiaru prądu służy amperomierz. Dla wyobrażenia sobie jak płynie prąd pomyślmy o rzece która ma swoje podziemne źródło, a woda spływa do morza. Przepływ tej wody to prąd, czyli ilość wody w jednostce czasu jaki przepłynie do morza.

Powiedzmy teraz o napięciu czyli pewnej wielkości, która powoduje w ogóle przepływ prądu elektrycznego. Napięcie to różnica potencjałów elektrycznych, która występuje pomiędzy dwoma punktami pola elektrycznego lub obwodu elektrycznego. Napięcie elektryczne, które zostało wytworzone, sprawia, że ładunek elektryczny jest przenoszony przez przewodnik. Upraszczając, można powiedzieć, że napięcie określa siłę, z jaką chcą się do siebie zbliżyć ładunki. Czyli jeśli nie ma napięcia, to nie ma też siły. Natomiast wraz ze wzrostem napięcia, wzrasta i siła. Dla zobrazowania tej sytuacji możemy wrócić do naszego źródła i morza. Jeśli źródło będzie w stromych górach, więc woda wydostająca się z źródła będzie szybciej płynąć w dół. Napięciem w tym przypadku jest nachylenie ziemi oraz ciśnienie z jakim woda wydostaje się z źródła. Dzięki temu więcej wody i szybciej ma możliwość spłynąć do morza. Jednostką napięcia jest volt, który oznaczony jest literą V, natomiast skrótem jest litera U.

Prąd dzielimy na zmienny i stały. Zmienny czyli taki dla którego natężenie zmienia się w czasie w dowolny sposób. Stały czyli taki w którym kierunek przepływu się nie zmienia. Mówiąc o prądzie zmiennym mamy na myśli m.in. cykliczny, okresowy, przemienny np. sinusoidalny, prostokątny. Głównie w elektronice używamy prądu stałego z różnego rodzaju baterii i akumulatorów. Natomiast prąd w sieci elektrycznej otrzymujemy zmienny, przemienny. Następnie zmieniamy go z pomocą transformatora i prostujemy np. z pomocą mostka Graetza.



Kolejnym pojęciem ważnym w elektronice jest rezystancja czyli opór jaki stawiają wszystkie elementy układu elektronicznego. Rezystancja jest to pewna relacja pomiędzy natężeniem, a napięciem. W idealnym przewodniku czyli elemencie który może przewodzić prąd, rezystancja jest równa 0. Dzięki temu nie ma żadnego spadku napięcia na nim. Niestety taki przewodnik nie istnieje. Wróćmy znowu do wody, źródła i morza. Woda wypływa i płynie, jest to prąd, napięcie to nachylenie terenu, ciśnienie wody i dodajmy jeszcze szerokość koryta rzeki. Jeśli rzeka jest prosta, bez zanieczyszczeń, kamieni i tamy wybudowanej przez ludzi lub bobry to woda szybko spłynie od źródła do morza. Takich rzek nie ma, te wszystkie przeszkody jakie na swojej drodze spotka woda to właśnie opór, rezystancja. Każdy element elektroniczny daje pewien opór płynącemu prądowi np. z powodu swojej budowy, właściwości fizycznych, jeśli prąd się zmniejszy jednocześnie spadnie napięcie. Reasumując przewodniki, które mają duży opór, gorzej przewodzą prąd od tych, które mają mały opór. Jednostką, która określa opór, jest om, czyli symbol Ω , a skrótem literka R.



Lei de Ohm
Fonte: Build Electronic Circuits
źródło: instagram

Podsumowaniem tych informacji niech będzie ten humorystyczny rysunek. Widzimy tutaj Ohma który z pomocą lasa blokuje drogę Voltowi i Amperowi. Volt siłą swoją przepycha Ampera i dzięki temu prąd może płynąć. Idealnie obrazujemy tutaj rolę napięcia w stosunku do natężenia.

Ostatnim pojęciem jaki chcę poruszyć w tym rozdziale jest moc w elektronice. Moc elementów elektronicznych wyrażana jest iloczynem przepływającego prądu do napięcia. Ogólnie rzecz ujmując moc to wielkość fizyczna określająca pracę jaką musi wykonać element elektryczny. Często poprzez podanie mocy danego elementu stwierdza się w jakich warunkach może on pracować aby nie został uszkodzony. Przy niektórych elementach nie podaje się wartości napięcia i prądu pracy ale podaje się moc. Teraz my musimy obliczyć czy dany element może pracować w naszym układzie.

Moc wyrażamy w jednostce wat, skrót duża litera W. Skrót mocy to duża litera P.
Wzór do obliczenia mocy:

$$P = U * I.$$

4.1.1. Prawo Ohma

Napięcie, natężenie prądu i rezystancję wiąże ze sobą prawo Ohma. Prawo fizyki zaprezentowane przez Georga Simona Ohma w latach 1825-1825. Prawo te opisuje stosunek napięcia przyłożonego do elementu o danej rezystancji, w wyniku czego płynie prąd. Podstawowym wzorem jest:

$$U = I * R$$

U to napięcie, **I** to natężenie prądu, **R** to rezystancja (opór)

Dzięki powyższemu wzorowi, przy odpowiednich przekształceniach możemy obliczyć napięcie, natężenie, opór jeśli posiadamy dwie wartości, a brakuje nam trzeciej.

$$I = U / R$$

$$R = U / I$$

Na przykład posiadamy napięcie równe 12 V, opór 200 Ω i musimy obliczyć jaki prąd będzie płynąć.

$$I = 12 \text{ V} / 200 \Omega = 0,06 \text{ A}$$

Musimy teraz zrozumieć to prawo aby przejść dalej. Postaram się to wyjaśnić najprościej, poszczególne zależności. Jak już zapewne zauważyłem lubię porównywać układ elektroniczny do przepływu wody. Wyobraźmy sobie jezioro na którym jest tama i wypływa z niej woda do rzeki. W jeziorze jest mało wody i odpływ w takie jest ciągle tak samo otwarty to ciśnienie wody wypływającej z niej jest niskie. Mówimy tutaj o sytuacji gdzie mamy stały opór, zawór w tamie jest tak samo otwarty. Jeśli w jeziorze będzie nagle więcej wody to będzie woda pod większym ciśnieniem zacznie wydostawać się przez zawór i więcej wody będzie trafiać do rzeki. Przekładając to na napięcie i prąd, jeśli zwiększymy napięcie to i prąd zacznie większy płynąć. Przykład:

$$I = 24 \text{ V} / 200 \Omega = 0,12 \text{ A}$$

Widzimy w powyższym przykładzie że jeśli dwukrotnie zwiększyliśmy napięcie to prąd też wzrósł dwukrotnie. W kolejnym kroku ustalmy sobie że w jeziorze mamy cały czas tyle samo wody, natomiast będziemy przykręcać zawór, przez co do rzeki zacznie trafiać mniej wody. Przykład:



$$I = 24 \text{ V} / 400 \Omega = 0,06 \text{ A}$$

Nam nadzieję że w tych prostych przykładach zrozumiałeś jak działa prawo Ohma. Pamiętaj że jest to jedno z trzech najważniejszych praw w elektronice i warto je pamiętać.

4.1.2. Prawa Kirchhoffa

W roku 1845 niemiecki fizyk Gustaw Kirchhoff sformułował pierwsze ze swoich praw dotyczące przepływu prądu w rozgałęzionym obwodzie elektrycznym. Prawo może wyrazić następująco:

Suma natężeń prądów wpływających do węzła jest równa sumie natężeń prądów wypływających z tego węzła.

Może to zabrzmieć trochę niezrozumiale ale już spieszę z porównaniem i tak znowu będzie to woda. Założmy że do jednego pojemnika szczelnie zamkniętego doprowadzamy wodę dwoma węzami ogrodowymi podpiętymi pod kran. Z pojemnika wychodzą natomiast 3 węże ogrodowe które doprowadzają wodę do podlania trzech drzew owocowych. Ilość wody jaka trafi do pojemnika równa się ilości wody dostarczonej do podlania drzew, ponieważ w żaden cudowny sposób nie zniknie, jeśli wszystko jest szczelnie zrobione. Tak właśnie działa to w układzie elektrycznym, jeśli dostarczymy prąd to musimy go wykorzystać. Suma prądu dostarczonego i zużytego musi równać się 0. W przeciwnym wypadku układ nie będzie działać prawidłowo.

Drugie prawo Kirchhoffa mówi natomiast o napięciu. Najczęściej prawo to jest formułowane w postaci:

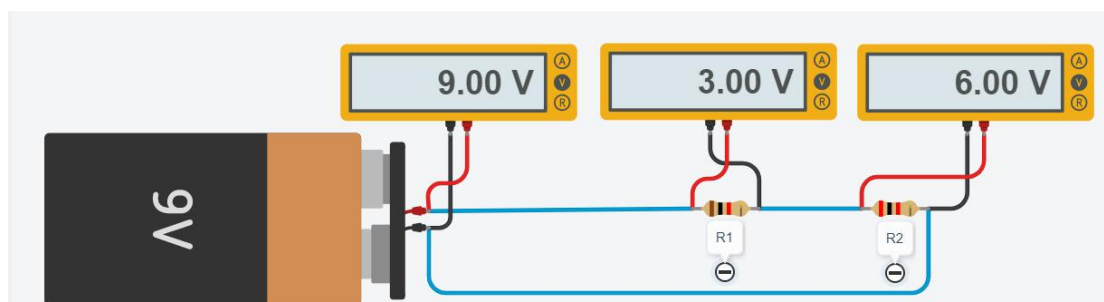
W zamkniętym obwodzie suma spadków napięć na oporach równa jest sumie sił elektromotorycznych występujących w tym obwodzie.

Prawo to mówi, że jeżeli w obwodzie lub wybranej jego części wybierzemy drogę zamkniętą, potocznie mówiona od plusa do minusa, to suma napięć na poszczególnych elementach jest równa napięciu źródła. Najprościej wyjaśnić to będzie prawdopodobnie na pomiarach z pomocą voltomierza.

Wspomnę tylko że voltomierz to urządzenie do pomiaru napięcia, przeważnie jest on częścią tzw. multimetru.



Na powyższym zdjęciu prezentuję przykładowe urządzenie. W celu rozpoczęcia pomiaru należy ustawić go napięcie stałe V_{DC} w zakresie do 20V.

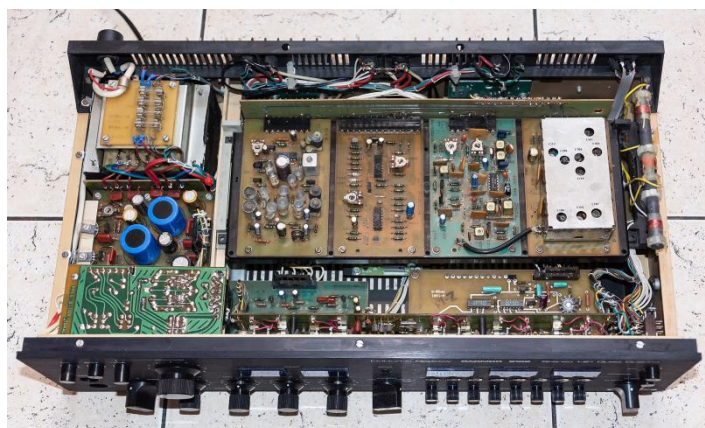


Na powyższym zdjęciu prezentuję prosty układ elektroniczny składający się z zasilania, bateria 9V, dwóch rezystorów $R1 = 1000\Omega$ i $R2 = 2000\Omega$. Dodatkowo w symulatorze dodałem trzy voltomierze. Jak widzimy voltomierz pierwszy z lewej pokazuje napięcia źródła i kolejno voltomierze prezentują spadki napięć na poszczególnych elementach. Suma spadków równa się napięciu źródła, $3V + 6V = 9V$.

4.2. Najczęściej spotykane komponenty elektroniczne

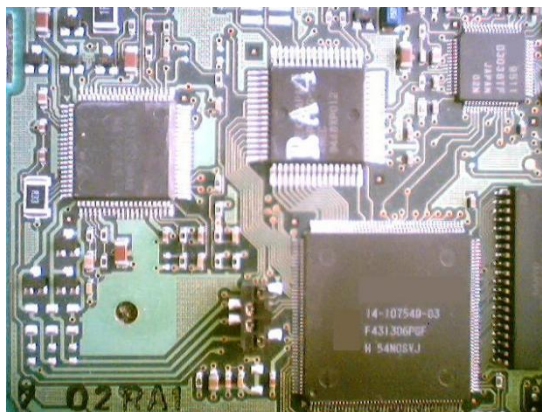
Poznaliśmy podstawowe pojęcia w świecie elektroniki, przyszedł czas na poznanie podstawowych komponentów. Omówimy sobie tylko niektóre z nich w tym rozdziale ale więcej poznanie w czasie lektury dalszej części tego dzieła prawie literackiego.

Na początek musimy sobie powiedzieć o rodzajach elementów pod względem montażu. Będę starał pokazać różne faktyczne elementy i o nich opowiedzieć. Rozróżniamy elementy typu THT i SMD. Elementy THT to takie które montuje się na płytce drukowanej poprzez jej przewleknięcie. Płytką drukowaną, tzw PCB, to najprościej rzecz ujmując platforma z połączeniami elektronicznymi i punktami lutowniczymi na której w sposób stały z pomocą spoiwa lutowniczego montujemy elementy elektroniczne.



Źródło: https://pl.wikipedia.org/wiki/P%C5%82ytka_drukowana

Powyższe zdjęcie prezentuje kilka PCB z elementami THT. Elementy przewlekane są dość duże, dzięki czemu łatwo się je lutuje. Elementy elektroniczne w tej technologii posiadają wyprowadzenia pozwalające na montaż z jednej strony PCB, a montaż z drugiej. Elementy THT nie pozwalają na dużą gęstość upakowania na płycie drukowanej. Poza tym w montażu automatycznym są kłopotliwe, dlatego też powstała technologia SMD, przystosowana do montażu powierzchniowego. Elementy te charakteryzują się tym że są dużo mniejsze od elementów THT przy zachowaniu podobnych właściwości. Elementy SMD przeważnie nie posiadają długich wyprowadzeń i montowane są z tej samej strony PCB z której zostały ułożone.



Źródło: https://pl.wikipedia.org/wiki/Monta%C5%BC_powierzchniowy

4.2.1. Rezystor

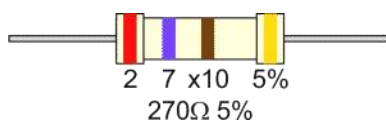


źródło: <https://eduinf.waw.pl/>

Rezystory, nazywane też opornikami są elementami biernymi, czyli takimi na którym występuje tylko strata energii. Zadaniem rezystora jest zamiana energii elektrycznej w ciepło. Celem tego jest spadek energii. Idealny rezystor posiada tylko jedną wielkość która go charakteryzuje czyli rezystancję wyrażoną w jednostce Ohm [Ω]. Natomiast jako że nie ma ideałów, w praktyce występuje jeszcze pojemność wewnętrzna oraz wewnętrzna indukcyjność. Dla nas ważnymi parametrami pojedynczego rezystora jest jego rezystancja nominalna czyli określona przez producenta, tolerancja czyli dokładność tej rezystancji, moc znamionowa czyli ile ciepła przez dłuższy czas może wydzielić opornik zanim zostanie uszkodzony. Wartości rezystorów THT są kodowane z pomocą pasków odpowiadającym: dwa pierwsze paski cyfry, trzeci to mnożnik oraz kolejny to tolerancja.

Kolor	Cyfra	Mnożnik	Tolerancja
Brak			20% (E6)
Srebrny		x 0,01	10% (E12)
Żółty		x 0,1	5% (E24)
Czarny	0	x 1	
Brązowy	1	x 10	
Czerwony	2	x 100	
Pomarańczowy	3	x 1000	
Żółty	4	x 10 000	
Zielony	5	x 100 000	
Niebieski	6	x 1000 000	
Fioletowy	7	x 10 000 000	
Szary	8	x 100 000 000	
Biały	9	x 1000 000 000	

Wykonajmy proste ćwiczenie które mam nadzieję pozwoli tobie zrozumieć sposób odczytywania kodów z rezystorów.



źródło: <https://eduinf.waw.pl/>

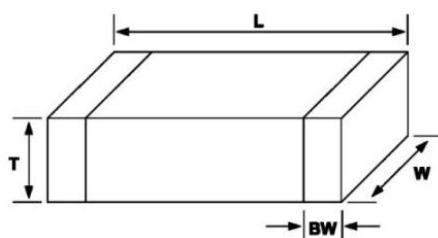
W powyższym zdjęciu chyba wszystko zostało wyjaśnione ale może opiszmy to sobie. Rezystor odczytujemy od lewej do prawej, sprawdzając aby z prawej strony wystąpił kolor złoty, srebrny lub brak paska. Zaczynamy odczytywanie kolor czerwony to liczba 2, kolor fioletowy to liczba 7. Wyszła nam liczba 27 i zaczynamy mnożyć przez wartość 10 zakodowaną przez pasek w kolorze brązowym. Wyszło nam z obliczeń że rezystor ma wartość rezystancji nominalnej 270 Ω ale jego tolerancja to 5%. Czyli wartość faktyczna rezystancji może wahać się od 256,6 Ω do 283,5 Ω.

Ogólnie rzecz ujmując rezystory THT są zbudowane w różnej technologii m.in. węglowe, metalizowane, drutowy w ceramice. Najbardziej popularne są rezystory węglowe o mocy 1/4W do 2W. Natomiast jeśli potrzebujemy rezystorów o dużej mocy rozproszenia ciepła powinniśmy sprawdzić drutowe w ceramice m.in. 5W, 10 W.

Rezystory SMD kodujemy inaczej. Ogólnie rzecz ujmując elementy SMD są mniejsze temu też stworzono inny sposób znakowania. Obudowa rezystorów SMD ma kształt niedużego prostopadłościanu z czego dwa krańcowe boki są pokryte metalem w celu ich przylutowania. Na obudowie rezystora która jest czarna naniesione są cyfry. Przeważnie są to 3 cyfry ale nie zawsze. Jeśli nasz rezystor ma trzy cyfry na obudowie np. 104 to rozkodowujemy to następująco:

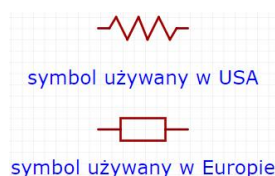
$$10 \times 10^4 = 100\,000\, \Omega$$

Oznacza to że dwie pierwsze cyfry mnożymy przez 10 do potęgi liczby z pozycji trzeciej. Możemy też spotkać oznaczenia czterocyfrowe, bardziej dokładne. Przykład: 1822, rozkodowana wartość to $182 \times 10^2 = 18\,200\, \Omega$. Jeszcze inny sposób kodowania to oznaczenia z literą R, np. 35R7 co równa się 35,7 Ω. Wspomnę jeszcze że bardzo małe rezystory są oznaczane według specjalnego kodu EIA-96, składającego się z dwóch cyfr i litery np. 38C co równa się $243 \times 100 = 24\,300\, \Omega$. Pełna tabelka kodów dostępna w internecie po wpisaniu frazy „kodowanie EIA-96”. Mówiąc o rezystorach SMD należy też wspomnieć że jest ich szereg wielkości, tzw. obudowy.



Obudowa	Długość (L)	Szerokość (W)	Wysokość (T)	Szczerość pola lutowniczego (BW)	Moc
0402	1 mm	0,5 mm	0,3 mm	0,2 mm	1/16W
0603	1,6 mm	0,8 mm	0,45 mm	0,25 mm	1/10W
0805	2 mm	1,25 mm	0,5 mm	0,35 mm	1/8W
1206	3,1 mm	1,6 mm	0,55 mm	0,45 mm	1/8W - 1/4W
1210	3,1 mm	2,6 mm	0,555 mm	0,5 mm	1/4W - 1/3W
2010	5 mm	2,5 mm	0,55 mm	0,6 mm	1/2 - 3/4W
2512	6,35 mm	3,2 mm	0,6 mm	0,6 mm	1w - 2W

Mówiąc o rezystorach należy też wspomnieć o sposobie ich łączenia. Jest to bardzo ważna kwestia ponieważ nie ma rezystorów o wszelkiej możliwej wartości rezystancji. Wartość rezystorów określa tzw. szereg wartości. Przeważnie spotykane rezystory są szeregu głównego E12 czyli liczby: 10, 12, 15, 18, 22, 27, 33, 39, 47, 56, 68, 82. Innym popularnym szeregiem jest dokładniejszy E24 czyli liczby: 10, 11, 12, 13, 15, 16, 18, 20, 22, 24, 27, 30, 33, 36, 39, 43, 47, 51, 56, 62, 68, 75, 82, 91. Reasumując w szeregu E12 możemy znaleźć rezystor o wartości np. 4.7Ω ale nie znajdziemy 3.3Ω . Natomiast w szeregu E24 mamy większy wybór. Dzięki odpowiedniemu łączeniu rezystorów możemy uzyskać opór o interesującej nas wartości. Zanim przejdziemy do łączenia powiedzmy sobie jaki symbol mają rezystory na schematach. O symbolach więcej powiemy w rozdziale o schematach ale już teraz poznamy przynajmniej podstawowe symbole.

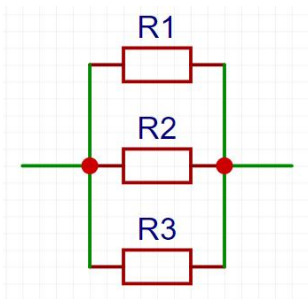


Rozróżniamy połączenie szeregowe i równoległe rezystorów. W połączeniu rezystancją zastępczą czyli rezystancją sumy użytych rezystorów jest wynik wzoru użytego na przykładzie z poniższego rysunku:



$$R_1 + R_2 + R_3 = R_z$$

Gdzie R_1 , R_2 , R_3 jest wartością poszczególnych rezystorów, a R_z jest rezystancją zastępczą. Przykład: wartość $R_1 = 10 \Omega$, $R_2 = 22 \Omega$, $R_3 = 47 \Omega$, wartość $R_z = 10 + 22 + 47 = 79 \Omega$. W ten sposób udało nam się uzyskać rezystor o rezystancji zastępczej której nie ma w szeregu E12 i E24.



Przyszła pora na połączenie równoległe. Powyższy rysunek pokazuje połączenie trzech rezystorów w sposób równoległy. W celu obliczenia rezystancji zastępczej musimy wykonać obliczenia z następującego wzoru:

$$\begin{aligned} \frac{1}{R_z} &= \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} \\ \frac{1}{R_z} &= \frac{(R_2 * R_3) + (R_1 * R_3) + (R_1 * R_2)}{(R_2 * R_3) + (R_1 * R_3) + (R_1 * R_2)} \\ R_z &= (R_1 * R_2 * R_3) / ((R_2 * R_3) + (R_1 * R_3) + (R_1 * R_2)) \end{aligned}$$

Przykład $R_1 = 10 \Omega$, $R_2 = 22 \Omega$, $R_3 = 47 \Omega$, wartość $R_z = (10 * 22 * 47) / (22 * 47 + 10 * 47 + 10 * 22) = 10340 / (1034 + 470 + 220) = 10340 / 1724 = 5,99 \Omega$

4.2.2. Kondensator

Przyszła pora na kondensator czyli element elektroniczny który gromadzi ładunek elektryczny. Zbudowany jest z dwóch półprzewodników tzw. okładek rozdzielonych dielektrykiem. Kondensatory możemy podzielić na kilka grup w zależności od konstrukcji:

- kondensatory elektrolityczne
- kondensatory ceramiczne
- kondensatory foliowe
- kondensatory tantalowe



- kondensatory powietrzne

Kondensatory możemy podzielić również na biegunowe (spolaryzowane) i bezbiegunowe (niespolaryzowane). Do kondensatorów biegunowych zaliczamy m.in. Kondensatory elektrolityczne.

Kondensatory używa się głównie w celu przeciw działaniu zakłóceń, układach filtrujących.

Podstawowymi kondensatorami są elektrolityczne i ceramiczne. Kondensator elektrolityczny zbudowany jest z elektrody metalowej i elektrolitowej, które podłączone są do wyprowadzeń i rozdzielone są warstwą dielektryka. Elektroda metalowa wykonana jest zazwyczaj z aluminium lub tantalu, a rolę dielektryka pełni cienka warstwa tlenku metalu (np. tlenku glinu).

Przy stosowaniu kondensatorów elektrolitycznych pamiętajmy jak szukać elektrody dodatniej i ujemnej. Przy nowych kondensatorach przewlekanych plusem jest dłuższa nóżka, a minusem krótsza. Drugim sposobem jest sprawdzenie na obudowie kondensatora nadruku, często na czarnym lub srebrnym pasku jest napisany znak „+”, minus. Oznacza to że nóżka z tej strony jest ujemna.



Symbole kondensatorów:



Kondensatory cechują dwa główne parametry: pojemność wyrażona w jednostce Farad [F] oraz napięcie pracy. Farad jest bardzo dużą jednostką dlatego w praktyce często spotkasz kondensatory o pojemności:

- mikrofarad [μF] ($1 \mu\text{F} = 0,000\,001 \text{ F}$),
- nanofarad [nF] ($1 \text{ nF} = 0,000\,000\,001 \text{ F}$),
- pikofarad [pF] ($1 \text{ pF} = 0,000\,000\,000\,001 \text{ F}$).

Drugim parametrem było napięcie pracy informujące nas jakie maksymalnie może być podłączone napięcie do kondensatora aby nie wystąpiło ryzyko uszkodzenia jego warstw. Najpopularniejszymi wartościami napięć pracy jest: 10V, 16V, 25V, 35V, 50V, 63V, 100V. Oczywiście kondensatory występują w wersji przewlekanej (THT) i na płytkowej (SMD).

Podobnie jak przy rezystorach kondensatory występują w pojemnościach pogrupowanych w tzw. szeregi tolerancji:

Nazwa	Tolerancja	Szereg
E6	20%	10, 15, 22, 33, 47, 68
E12	10%	10, 12, 15, 18, 22, 27, 33, 39, 47, 56, 68, 82
E24	5%	10, 11, 12, 13, 15, 16, 18, 20, 22, 24, 27, 30, 33, 36, 39, 43, 47, 51, 56, 62, 68, 75, 82, 91

Pojemność kondensatorów w przypadku większych obudów pisze się normalnie, natomiast w przypadku innych koduje się. Dla przykładu w kondensatorze ceramicznym używa się kodu składającego się z trzech cyfr. Pierwsze dwie cyfry oznaczają wartość w szeregu w pF, trzecia cyfra jest mnożnikiem, potęgi liczby 10.



$$103: 10 * 10^3 \text{ pF} = 10000 \text{ pF} = 10\text{nF}$$



$$104: 10 * 10^4 \text{ pF} = 100000 \text{ pF} = 100\text{nF}$$

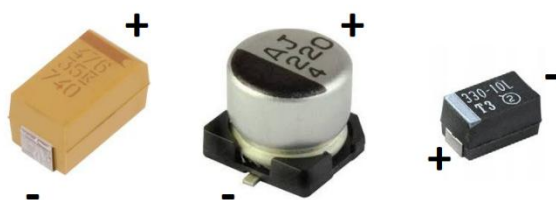
Jeśli na kondensatorze pojawi się duża litera przy oznaczeniu pojemności, to oznacza ona tolerancję:

E (0,005%), L (0,01%), P (0,02%), W (0,05%), B (0,1%), C (0,25%), D (0,5%), F (1%), G (2%), H (2,5%), J (5%), K (10%), M (20%), N (30%), Q (-10% do +30%), T (-10% do +50%), S (-20% do +50%), Z (-20% do +80%). Natomiast jeśli pojawi się mała litera (z wyjątkiem p i n) informuje ona o dopuszczalnym napięciu pracy: m (25V), l (40V), a (63V), b (100V), c (160V), d (250V), e (400V), f (630V), h (1000V), i (1600V). Zamiast

liter napięcie też może być podane też bezpośrednio np. 82nJ63: 82nF, tolerancja 5%, napięcie 63V lub 33nK100V: 33nF, tolerancja 10%, napięcie 100V.

Kondensatory SMD mają inne oznaczenia i często na obudowie nic nie jest napisane. Spowodowane może być to tym że elementy oznaczone wymagają dodatkowego etapu produkcji i są droższe. Często elementy SMD są montowane na taśmie produkcyjnej przez robota, dla niego nie ma znaczenia napis na elemencie, on dostaje odpowiednią taśmę i montuje poszczególne elementy. Jeśli kupujemy nie oznaczone kondensatory trzymajmy je w oznaczonym klaserze, pudełku lub zapiszmy ich parametry na taśmie. Oczywiście możemy spotkać kondensatory oznaczone np. kodem E.I.A. gdzie litera oznacza pojemność w pF, a liczba mnożnik, potęgi liczby 10: A (1,0), B (1,1), C (1,2), D (1,3), E (1,5), F (1,6), G (1,8), H (2,0), J (2,2), K (2,4), a (2,5), L(2,7), M (3,0), N (3,3), b (3,5), P (3,6), Q (3,9), d (4,0), R (4,3), e (4,5), S (4,7), f (5,0), T (5,1), U (5,6), m (6,0), V (6,2), W (6,8), n (7,0), X (7,5), t (8,0), Y (8,2), g (9,0), Z (9,1). Przykład: D3: $1.3 * 10^3 = 1300 = 1.3 \text{ nF}$ lub S2: $4.7 * 10^2 = 470\text{pF}$. Czasem też na początku dodawana jest litera oznaczająca producenta np. KA2: $1.0 * 10^2 = 100\text{pF}$ wyprodukowany przez firmę Kemet. Wartości pojemności mniejsze od 1pF używają cyfry "9", która oznacza podział przez 10 (cyfra 8 oznacza podział przez 100). Na przykład: f9 = $5.0 / 10 = 0.5 \text{ pF}$ lub n9 = $7.0 / 10 = 0.7\text{pF}$

Jeśli miejsca jest więcej, stosuje się kodowanie trzycyfrowe. Pierwsze dwie cyfry oznaczają pojemność w pF, a trzecia cyfra jest mnożnikiem: 104: = $10 * 10^4 = 100000\text{pF} = 100\text{nF} = 0.1\mu\text{F}$ lub 473: = $47 * 10^3 = 47000\text{pF} = 47\text{nF}$. Przy kondensatorach elektrolitycznych podaje się często napięcie pracy oraz pojemność. Kody składają się z litery (kodowanie napięcia) oraz 3 cyfr (2 pierwsze oznaczają pojemność w pF, trzecia jest mnożnikiem). Napięcia kodowane jest zgodne z oznaczeniem: e(2,5V), G(4V), J(6,3V), A(10V), C(16V), D(20V), E(25V), V(35V), H(50V). Na przykład A475: $47 * 10^5 = 4700000\text{pF} = 4.7\mu\text{F}$ i o napięciu pracy do 10V. Mówiąc o kondensatorach elektrolitycznych musimy pamiętać o ich biegunowości, tutaj musimy być ostrożni i sprawdzić oznaczenia w sklepie gdzie kupiliśmy element lub w datasheet:



4.2.3. Półprzewodniki

W celu przejścia dalej musimy poruszyć kwestię półprzewodników ale w sposób bardzo podstawowy. W przyrodzie istnieją trzy kategorie materiałów: przewodniki - czyli materiały dobrze przewodzące prąd, izolatory lub dielektryki - materiały źle przewodzące prąd oraz półprzewodniki - materiały które w pewnych warunkach przewodzą lub wykazują cechy izolatorów. Pamiętamy z poprzednich rozdziałów że przepływ prądu polega na ruchu nośników ładunków elektrycznych, jeśli materiał nie posiada swobodnych ładunków, to prąd nie będzie płynął, wtedy mówimy do izolatorze. Przewodniki natomiast posiadają dużo swobodnych elektronów które pochodzą z tzw. powłoki walencyjnej atomów. Elektrony w tej powłoce wykorzystywane są do wiązań ze sobą atomów metalu w siatkę krystaliczną. Elektrony swobodne są nośnikami ładunku elektrycznego w metalach. Zatem po przyłożeniu napięcia popłynie prąd elektronowy, którego natężenie określa znane ci już prawo Ohma.

Przechodząc do sedna czyli półprzewodników, ich atomy wykorzystują wszystkie elektrony walencyjne do utrzymania siatki krystalicznej. Nie ma zatem swobodnych elektronów. Choć w pewnych warunkach może dojść do wybicia elektronu walencyjnego i stanie się on swobodnym elektronem. Jednakże atom półprzewodnika, który utracił elektron walencyjny, staje się jonem dodatnim, czyli otrzymuje ładunek dodatni. Fizycznie jon ten nie może się przemieszczać w półprzewodniku, ponieważ jest uwięziony w siatce kryształu. Jednakże może przechwycić elektron walencyjny, który został wybity z innego atomu siatki. Dojdzie do tzw. rekombinacji ładunków. Schwytyany elektron zneutralizuje ładunek dodatni atomu. W efekcie ładunek dodatni zmieni położenie w strukturze półprzewodnika – przemieści się do atomu, który poprzednio stracił elektron walencyjny. Efekt będzie taki sam, jakby ten ładunek się swobodnie przemieszczał. Taki brak elektronu walencyjnego nazywamy dziurą, a ruch ładunków dodatnich nazywamy prądem dziurowym. W półprzewodniku mogą zatem przemieszczać się jednocześnie dwa rodzaje ładunków: elektrony (ujemne) oraz dziury (dodatnie). W czystym kryształ półprzewodnika np. krzemu liczba dziur odpowiada liczbie elektronów swobodnych i zwykle nie jest duża, dlatego półprzewodniki posiadają względnie dużą oporność. Z pomocą tzw. domieszkowania możemy zmienić liczbę elektronów walencyjnych, dodając je tworząc półprzewodnik typu n lub zabierając, tworząc nadmiar dziur czyli typu p. Powstałe w ten sposób nośniki nazywa się nośnikami większościowymi: w półprzewodniku p nośnikami większościowymi są dziury, a w półprzewodniku n elektrony.

W celu uzyskania półprzewodnika typu N do kryształu krzemu wprowadza się domieszki donorowe czyli pierwiastki tj. fosfor (P), antymon (Sb), arsen (As), bizmut (Bi). Atomy domieszki donorowej mają pięć elektronów na ostatniej powłoce walencyjnej i do utworzenia wiązań z sąsiadującymi atomami krzemu są potrzebne tylko cztery elektrony. Piąty elektron nie bierze udziału w wiązaniu kowalentnym, jest słabo związany z jądrem atomu i dzięki temu dostarczenie niewielkiej energii pozwala na zerwanie wiązania. Elektron ten po zerwaniu przechodzi do pasma przewodnictwa oraz pozostawia dodatni jon domieszki donorowej. W



półprzewodniku typu N znajduje się więcej elektronów w paśmie przewodnictwa niż dziur w paśmie podstawowym i dlatego elektrony są nośnikami większościowymi.

Półprzewodniki typu P otrzymujemy poprzez dodanie domieszki akceptorowej do półprzewodnika samoistnego. Przykładem niech będzie krzem do którego możemy dodać pierwiastki tj. bor (B), glin (Al), gal (Ga), ind (In). Atomy domieszki akceptorowej mają na swojej ostatniej powłoce walencyjnej trzy elektrony. Oznacza to że do stworzenia wiązania kowalentnego czyli takiego posiadającego parę brakuje jednego elektronu. Elektron ten można zastąpić elektronem z innego sąsiedniego wiązania. Elektron przechodzi z pasma podstawowego na dodatkowy poziom energetyczny akceptorowy. Atom domieszki staje się ujemnym jonem domieszki akceptorowej. W półprzewodniku typu P dziury są nośnikami większościowymi w paśmie podstawowym, a elektrony nośnikami mniejszościowymi w paśmie przewodzenia.

Złącze PN nazywamy styk dwóch kryształów ciała stałego, które tworzą ścisły kontakt i jest to warstwa przejściowa między obszarem półprzewodnika typu P, a obszarem półprzewodnika typu N. Pamiętajmy że w półprzewodniku typu P koncentracja dziur jest większa niż elektronów, natomiast w półprzewodniku N koncentracja elektronów jest większa niż dziur. W momencie połączenia dwóch półprzewodników następuje dyfuzja nośników i następuje zjonizowanie atomów domieszek. Obszar złącza PN nazywamy obszarem ładunku przestrzennego, warstwą zaporową lub obszarem zubożonym o dużej rezystancji. Obszary przyłączeniowe są odpowiednio naelektryzowane, dodatnio w półprzewodniku typu N i ujemnie w półprzewodniku typu P. W wyniku odmiennej polaryzacji obszarów przyłączeniowych powstaje różnica potencjałów czyli napięcie dyfuzyjne.

Jeśli przyłożymy do półprzewodnika p plus napięcia zasilającego, a do półprzewodnika n minus, to dziury w p i elektrony w n będą się przemieszczały w kierunku złącza, co spowoduje zmniejszenie bariery potencjału. Jeśli napięcie zasilające będzie większe od tej bariery (np. ponad 0,6V dla półprzewodnika z krzemu), to nośniki większościowe zaczną dyfundować przez barierę i w obwodzie popłynie prąd elektryczny – półprzewodnik staje się przewodnikiem. Złącze p-n będzie spolaryzowane w kierunku przewodzenia. Jeśli zmienimy biegunowość napięcia zasilającego (do półprzewodnika p przyłożymy minus, a do n plus), to spowoduje ono odpływ nośników większościowych z obszaru złącza i w efekcie powiększenie bariery potencjału. W obwodzie będzie płynął bardzo mały prąd dyfuzyjny – półprzewodnik staje się praktycznie izolatorem. Złącze p-n będzie spolaryzowane zaporowo.

4.2.4. Dioda

Jeśli już mamy podstawowe informacje na temat półprzewodników możemy przejść do diody. Dioda jest zbudowana z półprzewodnika typu p i n, które tworzą ze sobą złącze p-n.

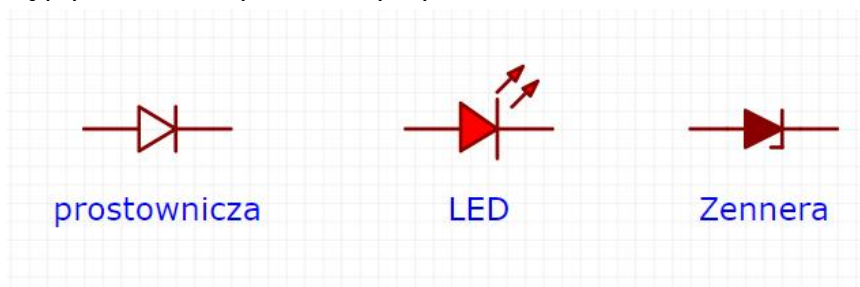
Istnieje wiele rodzajów diod ale my podzielmy je na trzy typy:

- diody prostownicze - służą do przepuszczania prądu tylko w jednym kierunku,
- diody stabilizacyjne Zenera - służą do stabilizacji napięcia w zasilaczach,

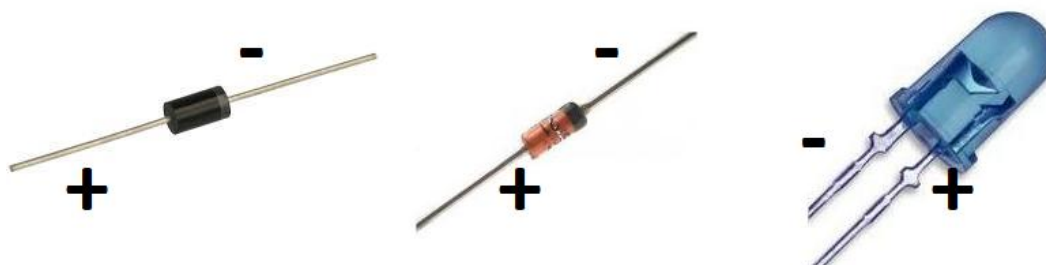


- LED - czyli diody emitujące światło.

Jeśli chodzi o budowę wyróżniamy dwie końcówki: anodę - połączoną z półprzewodnikiem typu p i katodę - połączoną z półprzewodnikiem typu n. Prąd przez diodę płynie od anody do katody. Symbole:



Kierunek zwrócenia trójkąta wskazuje jak płynie prąd, na powyższych symbolach prąd płynie od lewa do prawa.

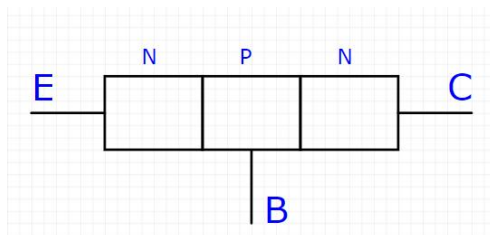


Co należy jeszcze wiedzieć o diodach? Musimy pamiętać że jeśli połączymy diodę w kierunku zaporowym (w którym nie przewodzi prądu), to zawsze popłynie przez nią niewielki prąd. Spowodowane jest to tym, że nośniki przenikają przez barierę potencjału. Jednakże prąd ten jest tak mały, że praktycznie możemy go pominąć w większości przypadków. Diody występują w technologii przewlekanej jak i na płytkowej. Kodowanie paramentów jest różne, przeważnie określane poprzez nazwę, model diody i informacje o jej parametrach należy szukać w dokumentacji. Mówiąc o diodach musimy na chwilę zatrzymać się prze tzw. LEDach. LED czyli dioda świecąca. Jak to jest że LEDy świecą? Wewnątrz diody znajduje się półprzewodnik który w stanie przewodzenia wydziela światło. Używając diod świecących musimy pamiętać że w zależności jaki kolor użyjemy to mamy do czynienia z innym napięciem pracy. I tak na przykład kolor czerwony (1,6-2,2V), żółty (2-2,3V), zielony (2-3,7V), niebieski (2,9-4V), biały (3-3,6V) i IR (1,1-1,7V). Diody świecące mogą występować jako przewlekane, smd, paski czy wyświetlacze. Jeśli mówimy o paskach czy wyświetlaczach przeważnie mają one wspólną katodę lub anodę, a prąd przepływa jeśli podłączymy w układ zamknięty odpowiednie wyprowadzenia.

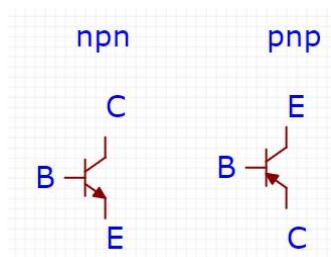
4.2.5. Tranzystor

Element półprzewodnikowy składający się z więcej niż dwóch warstw półprzewodnika. Tranzystory dzielimy na bipolarne i unipolarne.

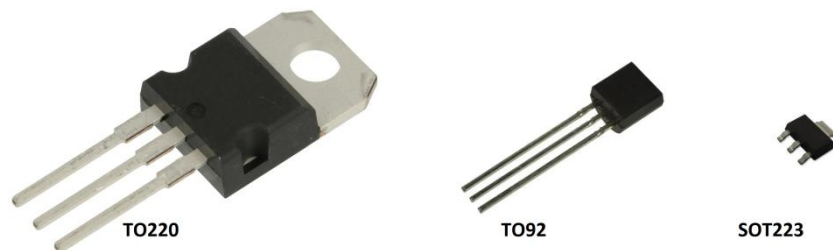
Zacznijmy od tranzystorów bipolarnych które składają się z trzech warstw półprzewodników tj. NPN i PNP. Do każdej z warstw podłączona jest osobna elektroda. Zasadę działania omówimy sobie na przykładzie tranzystora NPN, w PNP jest podobnie, lecz posiada on odwróconą polaryzację.



Elektrody tranzystora nazywamy następująco: E - emiter (wysyła ładunki), B - baza (steruje przepływem ładunków i C - kolektor (zbiera ładunki). Na styku warstw półprzewodnikowych znajdują się dwa złącza p-n i tworzy się na nich bariera potencjału. Jeśli spolaryzujemy złącze baza-emiter w kierunku przewodzenia (baza ma wyższy potencjał od emitera) to spowodujemy że elektrony z półprzewodnika typu **n** emitera, przejdą przez barierę potencjałów B-E (p-n) i znajdą się w półprzewodniku **p** bazy. Jeśli teraz spolaryzujemy złącze baza-kolektor zaporowo, (kolektor ma wyższy potencjał od bazy), to kolektor zacznie przyciągać ładunki ujemne z półprzewodnika typu **n** emitera do **p** bazy. W wyniku czego w obwodzie kolektor-emiter popłynie dużo większy prąd niż w obwodzie baza-emiter. Podsumowując małe zmiany prądu bazy wywołują duże zmiany prądu kolektora. Jeśli prąd bazy jest niewielki, prądu kolektora prawie nie ma, zwiększając prąd bazy, zwiększamy prąd emitera. W tranzystorze pnp powstają identyczne zjawiska, należy jedynie odwrócić biegunowość napięć baza-emiter oraz kolektor-baza. Współczynnik wzmocnienia prądowego tranzystora ale zwykle wynosi ponad 100, czyli przy prądzie bazy 1 mA, prąd kolektora wynosi 100mA. Na schematach elektronicznych tranzystory NPN i PNP oznaczamy następująco:



Tranzystory wstępują w ogromnej ilości obudów, jest ich ponad 50. Najczęściej spotykanymi jest TO220, SOT223 i TO92.

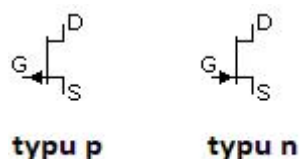


Przyszedł czas na zajęcie się zastosowaniem tranzystorów w elektronice. Najprościej rzecz ujmując tranzystory sterują przepływem prądu lub wzmacniają go, można powiedzieć że są takim elektronicznym przełącznikiem. Z pomocą sygnału na bazie jesteśmy w stanie sterować przepływem prądu między kolektorem, a emiterym. Postarajmy sobie wyobrazić taki układ gdzie z pomocą płytki arduino i portowi cyfrowemu numer 2 podłączonemu do tranzystora jesteśmy w stanie sterować silnikiem 2A. Normalnie taki prąd nie jesteśmy w stanie przepościć przez porty arduino ale z pomocą natężenia 20mA możemy w teorii sterować prądem nawet 2A, gdzie silnik podpięty jest do emitery tranzystora, a kolektor do źródła napięcia.

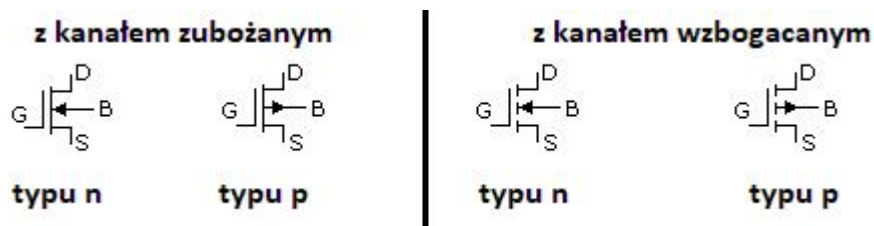
Powiedzmy sobie teraz kilka słów na temat tranzystorów unipolarnych. Tranzystory unipolarne, zwane też polowymi są podstawowym elementem współczesnej techniki cyfrowej. Sterowanie w nich odbywa się za pomocą pola elektrycznego. Zasadniczą częścią tranzystora polowego jest kryształ odpowiednio domieszkowanego półprzewodnika z dwiema elektrodami: źródłem (symbol S, od ang. source, odpowiednik emitery w tranzystorze bipolarnym) i drenem (D, drain, odpowiednik kolektora). Pomiędzy nimi tworzy się tzw. kanał, którym płynie prąd. Wzdłuż kanału umieszczona jest trzecia elektroda, zwana bramką (G, gate, odpowiednik bazy). Tranzystory polowe dzielimy na dwa główne typy: JFET oraz MOSFET.

Tranzystor JFET czyli tzw. Tranzystor polowy złączowy. Tranzystor taki składa się z warstwy półprzewodnika typu n (tranzystor z kanałem typu n) lub typu p (tranzystor z kanałem typu p) oraz wbudowanej w nią, silnie domieszkowanej warstwy półprzewodnika przeciwnego typu (odpowiednio p+ i n+). W rezultacie w tranzystorze jest złącze p-n. Na zewnątrz obudowy wyprowadzone są trzy końcówki: dren (D, drain); źródło (S, source) oraz bramka (G, gate). Jeśli do źródła S i drenu D przyłożymy napięcie U_{DS} , to popłynie prąd elektryczny I_{DS} . W takiej konfiguracji nie ma znaczenia polaryzacja tego napięcia – tranzystor JFET przewodzi w obu kierunkach na linii S-D. Jeśli jednak złącze G-S spolaryzujemy ujemnie, tzn. do bramki G przyłożymy niższy potencjał od potencjału źródła S, to złącze p-n zostanie spolaryzowane zaporowo. Wynika z tego, że tranzystor JFET działa jak zawór, a regulatorem przepływu jest napięcie na bramce G. Ponieważ w trakcie pracy złącze G-S spolaryzowane jest zaporowo (przy polaryzacji w kierunku przewodzenia tranzystor JFET traci swoje właściwości), to w obwodzie bramki tranzystora polowego

prąd praktycznie nie płynie. Oporność wejściowa ma wartość kilku megaomów. Sterowanie odbywa się za pomocą napięcia U_{GS} . Symbol tranzystora JFET:

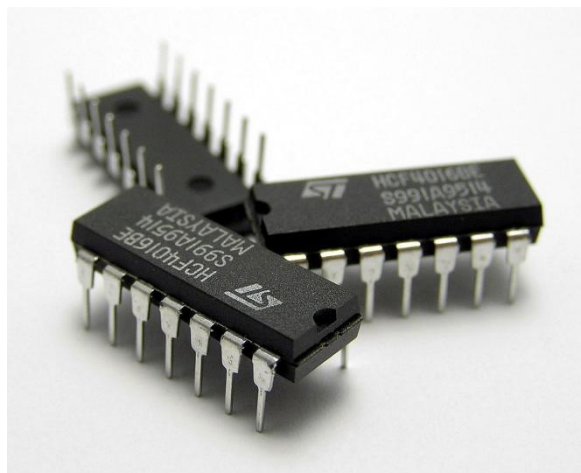


Tranzystor MOSFET jest zwany również tranzystorem polowym z izolowaną bramką. Zbudowany jest inaczej niż tranzystor JFET i posiada inne parametry. Dzisiaj jest to najpowszechniej stosowany tranzystor polowy. Znajdziesz go praktycznie w każdym urządzeniu cyfrowym: w komputerach, telefonach, pamięciach, itp. Tranzystor MOSFET posiada cztery elektrody choć tylko trzy są widoczne: Źródło S (ang. Source), Bramkę G (ang. Gate), Dren D (ang. Drain), Podłoże B (ang. Bulk). Podłoże B jest zwykle wewnętrznie połączone ze źródłem S, dlatego na zewnątrz wyprowadzone są tylko trzy elektrody: S, G i D. Zasada działania tranzystora MOSFET opiera się na sterowaniu przepływem prądu od źródła S do drenu D za pomocą bramki G. Jeśli do bramki G nie jest przyłożone żadne napięcie, to prąd nie będzie płynął pomiędzy S i D, ponieważ pomiędzy tymi elektrodami nie ma swobodnych nośników ładunku. Na styku półprzewodników p i n tworzą się dwa złącza p-n, z których jedno jest zawsze spolaryzowane zaporowo bez względu na kierunek napięcia przyłożonego do S i D. Jeśli jednak przyłożymy do bramki G napięcie dodatnie w stosunku do S i B, to spowoduje ono przyciągnięcie elektronów z półprzewodnika p i jednocześnie odepchnięcie dziur. Ponieważ bramka G jest izolowana od półprzewodnika p, to nie popłynie tutaj praktycznie żaden prąd. Wszystko odbywa się na zasadzie oddziaływań elektrostatycznych. Tranzystory MOSFET możemy spotkać w schematach pod symbolami:



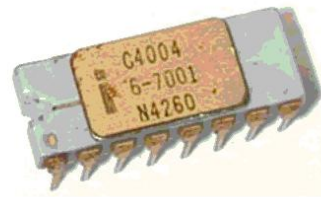
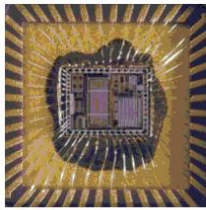
Tranzystory unipolarne są dzisiaj dźwignią przemysłu elektronicznego. Stosuje się je praktycznie w każdym bardziej zaawansowanym urządzeniu elektronicznym. Mogą one pełnić funkcje wzmacniaczy i przełączników sterowanych napięciem elektrycznym (tranzystor MOSFET posiada bardzo dużą oporność wejściową).

4.2.6. Układ scalony



Zminiaturyzowany układ elektroniczny to zminiaturyzowany układ elektroniczny zawierający w swym wnętrzu od kilku do setek milionów podstawowych elementów elektronicznych, takich jak tranzystory, diody, rezystory, kondensatory połączonych w pewną całość funkcjonalną. Układ scalony jest zwykle zamknięty w hermetycznej obudowie szklanej, metalowej, ceramicznej lub z tworzywa. Pierwsze układy scalone powstały w 1958 roku. Pierwszy układ stworzył Jack Kilby, pracownik Texas Instruments. Zalety scalania: duże upakowanie elementów funkcjonalnych w objętości (miniaturyzacja), duża niezawodność, niska cena, niskie czasy propagacji sygnału (duża częstotliwość pracy), małe zużycie energii. Układy scalone możemy podzielić m.in. ze względu na typ sygnału: analogowe, cyfrowe i mieszane. Układy analogowe są przystosowane do przetwarzania sygnałów (napięć lub prądów), których wartości zmieniają się w sposób ciągły w pewnym przedziale wartości. W układach cyfrowych sygnałami są dwa poziomy napięć: wysoki i niski. Drugim podziałem można zrobić ze względu na skalę scalenia (integracji). Miarą skali scalenia jest liczba podstawowych jednostek funkcjonalnych (bramek w układach cyfrowych lub tranzystorów w analogowych) zawartych w jednym układzie. Wyróżnia się:

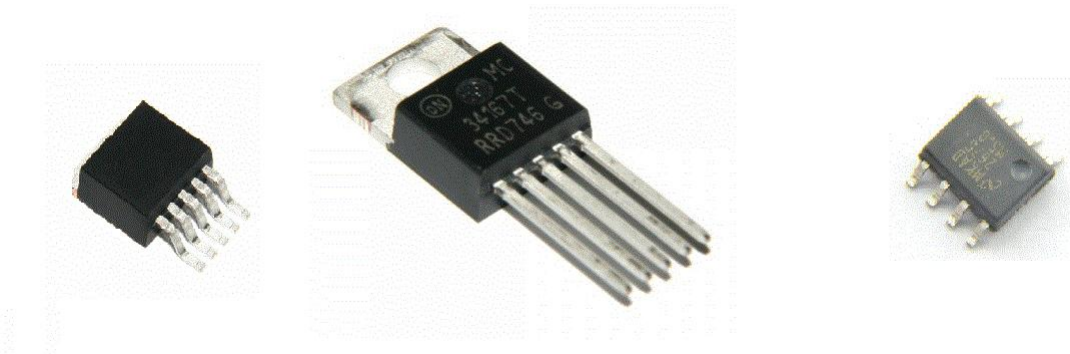
- układy o małej skali scalenia SSI (ang. Small Scale Integration), zawierające do 100 elementów
- układy o średniej skali scalenia MSI (ang. Medium Scale Integration), zawierające 10^2 - 10^3 elementów
- układy o wielkiej skali scalenia LSI (ang. Large Scale Integration), zawierające 10^3 - 10^5 elementów
- układy o bardzo wielkiej skali scalenia VLSI (ang. Very Large Scale Integration), zawierające ponad 10^5 elementów.



Źródło: <http://www.mjf.pg.gda.pl/homepages/jasiu/stud/ECS/wykl-13-US-wst-hyb.pdf>

4.2.7. Stabilizator

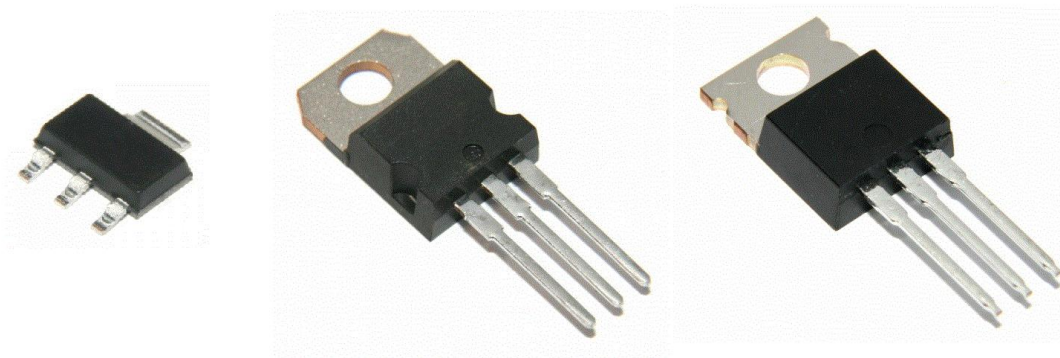
Układ elektroniczny, którego zadaniem jest utrzymywanie na wyjściu stałego napięcia (stabilizator napięcia) lub prądu (stabilizator prądu) niezależnie od obciążenia układu i wahań napięcia zasilającego. Powinien działać niezależnie od obciążenia układu oraz wahań napięcia, chroniąc tym samym wszelkie dalsze części elektroniki. Proces ochrony zawdzięczamy ujemnemu sprzężeniu zwrotnemu. Stabilne napięcie charakteryzuje się niezmiennością, niezależnie od panującej temperatury czy wartości pobieranego prądu. Warto wiedzieć, że każde urządzenie elektryczne dostarcza napięcie w dość nieprzewidywalny sposób, a jego wahania są typowe, dlatego napięcie określa się głównie w pewnych granicach i zakresie. Jednym z głównych parametrów stabilizatorów napięcia jest napięcie gwarantowanej stabilizacji. Jest to napięcie wejściowe w zakresie którego dane urządzenie zapewnia gwarantowany poziom napięcia na wyjściu. Równie istotnym parametrem jest napięcie pracy stabilizatora. Stabilizatory dzielą się głównie na dwie grupy: stabilizatory liniowe (o regulacji ciągłej) i stabilizatory impulsowe.



źródło: google.com grafika

Stabilizatory impulsowe, inaczej zwane przetwornicami, umożliwiają wytworzenie wyższego stabilnego napięcia wyjściowego niż napięcie dostarczane ze źródła zasilania. Takie działanie odbywa się na podstawie zjawiska samoindukcji (indukcji własnej). Jest to rodzaj zjawiska elektromagnetycznego, które występuje w przypadku, gdy siła elektromotoryczna wytwarzana jest w tym samym obwodzie co prąd, który przepływając przez układ powoduje indukcję. W ten sposób powstaje siła elektromotoryczna, która przeciwstawia się zmianom natężenia prądu elektrycznego.

Dzięki temu można uzyskać element podwyższający lub obniżający napięcie – w zależności od potrzeb.

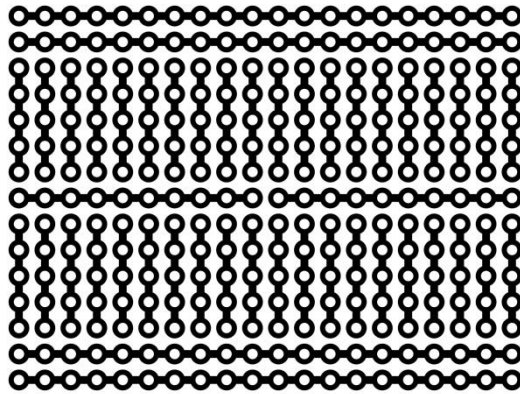


źródło: google.com grafika

Stabilizatory liniowe odpowiadają za dostarczenie stabilizowanego napięcia do odbiornika ze źródła zasilania. Początkowo stosowane były stabilizatory liniowe oparte na elementach dyskretnych (lampy, diody Zenera, tranzystory). Z czasem wyparły je stabilizatory w postaci układów scalonych, charakteryzujące się znacznie lepszymi parametrami. Większość stabilizatorów liniowych jest budowana w formie układów z trzema elektrodami. Stabilizatory liniowe można podzielić na cztery grupy: stabilizatory napięć dodatnich o ustalonym napięciu, stabilizatory napięć ujemnych o ustalonym napięciu, stabilizatory napięć dodatnich o napięciu ustalonym przez użytkownika, stabilizatory napięć ujemnych o napięciu ustalonym przez użytkownika. Najpopularniejsze stabilizatory napięć dodatnich pochodzą z serii 78XX, a napięć ujemnych – 79XX, gdzie dwie ostatnie cyfry określają napięcie wyjściowe. W środku oznaczenia może pojawić się dodatkowa litera informująca o maksymalnym prądzie pracy: L – 0,1 A, M – 0,5 A, bez litery – 1 A lub 1,5 A, S – 2 A, T – 3 A [3]. Przykładowo LM78L05 ma napięcie wyjściowe 5 V i prąd do 0,1 A, a LM79M15 – napięcie -15 V i prąd 0,5 A. Stabilizatory liniowe możemy również podzielić na nieregulowane i regulowane. Model regulowany da się bez problemu regulować, co ma duże znaczenie dla osób poszukujących bardzo skonkretyzowanych rozwiązań z dużą możliwością wprowadzenia zmian. W tym celu wykorzystuje się dodatkowe potencjometry. Najpopularniejsze stabilizatory napięcia to m.in. LM317 i LM7805

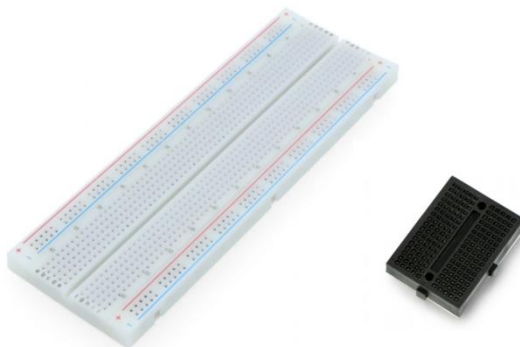
4.3. Płytki stykowa

Często nazywana też płytką prototypową lub uniwersalną. Takie narzędzie pozwala na szybkie tworzenie prototypów obwodów drukowanych bez konieczności ich każdorazowego przygotowywania, wytrawiania i lutowania.



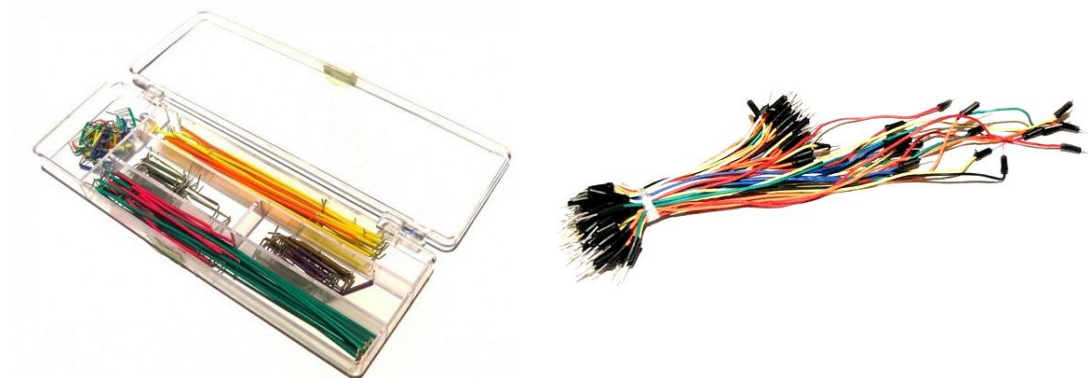
Źródło: https://pl.wikipedia.org/wiki/P%C5%82ytka_prototypowa

Powyżej prezentuję typowy układ ścieżek na drukowanej płytce prototypowej. W sprzedaży jest wiele rodzajów płytek, najczęściej używanymi są z 830 lub 170 otworami.



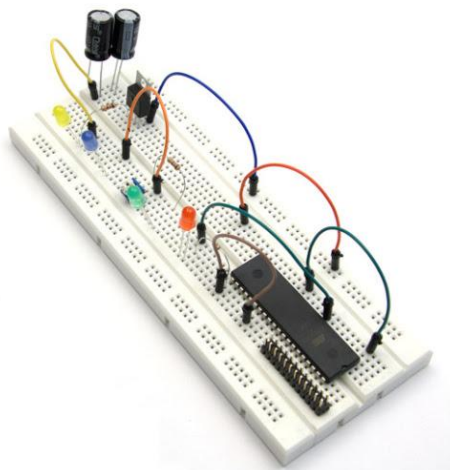
Źródło: <https://abc-rc.pl> i <https://botland.com.pl/>

Do tworzenia układów na płytce stykowej używamy przewodów połączeniowych lub zwrotek.



Źródło: <https://abc-rc.pl>

Tworząc układ na płytce stykowej staraj się zachować porządek. Im bardziej popłączesz przewody tym trudniej będzie tobie w późniejszym etapie znaleźć ewentualny błąd.



Źródło: <http://electropark.pl/>

4.4. Schematy elektroniczne

Zaczynając swoją przygodę z elektroniką mało co rozumiałem „rysunki” tych wszystkich układów. Znałem jakieś podstawowe symbole ale reszta była dla mnie czymś magicznym. Wtedy uświadomiłem sobie że muszę poszerzyć swoją wiedzę dotyczącą symboliki występującej na schemacie. W tym rozdziale będę chciał Tobie pokazać jak czytać i poniekąd tworzyć swoje własne schematy. Uważam że taka wiedza przyda się Tobie szybciej niż myślisz. Znajomość symboli i zasad panujących w schematach elektronicznych sprawi, że nie będziesz czuł się zakłopotania ani strachu podczas pracy z układami. Nauka elektroniki to wbrew pozorom ciągła praca nad poszerzaniem swojej wiedzy. Schematy elektroniczne można porównać do mapy z siecią dróg, ulic łączących ze sobą ważne elementy, domu. Dzięki takiej mapie możemy odkryć co tak naprawdę dzieje się w Naszym układzie. Gdy opanujesz symbole i oznaczenia, nauczysz się czytać schematy zobaczysz jakie to wszystko wydaje się proste. Oczywiście sama umiejętność czytanie nie stworzy z Ciebie eksperta, ponieważ musisz też czytać ze zrozumieniem, prawie jak na lekcji polskiego.

W elektronice wyróżniamy trzy główne rodzaje schematów choć ja poruszę jeszcze czwarty, który sam nazwałem schematem wizualnym lub połączenia elementów. Trzy główne rodzaje to:

- schemat blokowy - pokazuje w sposób ogólnikowy połączenia pomiędzy obwodami dyskretnymi urządzeniami. Każdy obwód jest reprezentowany przez „blok” o kształcie

kwadratu lub innej figury. Linie poprowadzone pomiędzy tymi elementami (mogą być zakończone strzałkami) symbolizującej zależności pomiędzy obwodami.

- schemat ideowy (często nazywany po prostu schematem) zawiera symboliczne oznaczenia wszystkich podzespołów znajdujących się w danym obwodzie.

- schemat wykonawczy, zwany również schematem poglądowym, ilustruje elementy obwodu zainstalowane w płytce lub podstawie montażowej. Pozwala on na szybkie znalezienie i zidentyfikowanie podzespołów, które chcesz sprawdzić lub wymienić.

Źródłem do tego rozdziału jest książka „Schematy elektroniczne i elektryczne. Przewodnik dla początkujących” autorstwa Stana Gibilisco wydawnictwa Helion.

4.4.1. Podstawowe symbole elementów elektronicznych

Chciałbym tutaj pokazać Wam tylko część z używanych symboli, jest ich bardzo dużo ale podrozdział ten miałby wtedy sam kilka stron.

Nazwa elementu	Symbol
amperomierz	
antena	
bezpiecznik	
bramka AND	
bramka NAND	
bramka NOR	
bramka NOT	
bramka OR	
bramka XOR	
cewka powietrzna	
dioda - symbol ogólny	
dioda elektroluminescencyjna (LED)	

Nazwa elementu	Symbol
potencjometr	
punkt kontrolny	
rezystor nastawny	
rezystor stały	
tranzystor bipolarny npn	
tranzystor bipolarny pnp	
tranzystor jednozłączowy	
tranzystor polowy MOSFET z kanałem typu n	
tranzystor polowy MOSFET z kanałem typu p	
tranzystor polowy z kanałem typu n	
tranzystor polowy z kanałem typu p	
tranzystor świetłoczuły npn	

Nazwa elementu	Symbol
dioda polowa	
dioda Schottky'ego	
dioda tunelowa	
dioda Zenera	
fotodioda	
generator sygnału	
głośnik	
kondensator stały spolaryzowany	
kondensator stały niespolaryzowany	
kondensator zmienny	
Miernik (symbol ogólny)	
mikrofon	
optoizolator	

Nazwa elementu	Symbol
tranzystor światłoczuły pnp	
triak	
tyrystor diodowy obukierunkowy (driak)	
układ scalony	
uziemienie	
uziemienie	
woltomierz	
wzmacniacz	
wzmacniacz operacyjny	
źródło napięcia	
źródło prądowe	
Źródło zasilania	
żarówka	

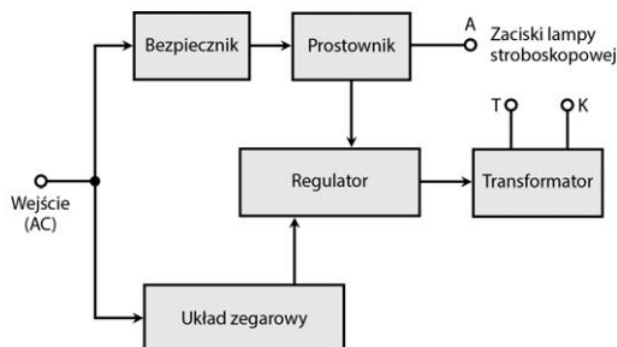
Źródło: książka „Schematy elektroniczne i elektryczne. Przewodnik dla początkujących” autorstwa Stana Gibilisco wydawnictwa Helion

4.4.2. Schemat blokowy

Schemat blokowy opisuje ogólną budowę urządzenia lub systemu elektronicznego. Schemat taki może posłużyć do uproszczonej prezentacji obwodu. Schemat blokowy możemy podzielić na przynajmniej dwa rodzaje: schemat blokowy funkcyjny oraz schemat blokowy procesu.

Schemat blokowy funkcyjny to taki który w sposób graficzny wyjaśnia jak działają poszczególne elementy urządzenia. Tworząc układ od podstaw możemy na początek stworzyć schemat zawierający tylko ogólne elementy, funkcje naszego obwodu. Z czasem sprecyzujemy jak dane fragmenty będą działać. My na początku potrzebujemy tylko informacji co dana rzecz ma robić i jak działać. Schematy blokowe funkcyjne opisują nasz układ. W późniejszym etapie będziemy rozwijać jak faktycznie dany blok ma wyglądać tworząc do tego inny schemat. Taki schemat blokowy funkcyjny można porównać do etapowania pracy nad układem. Mamy blok który ma coś robić i wiemy ile takich bloków potrzebujemy aby urządzenie mogło działać. Omówmy sobie coś prostego. Weźmy układ zasilacza lampy stroboskopowej.

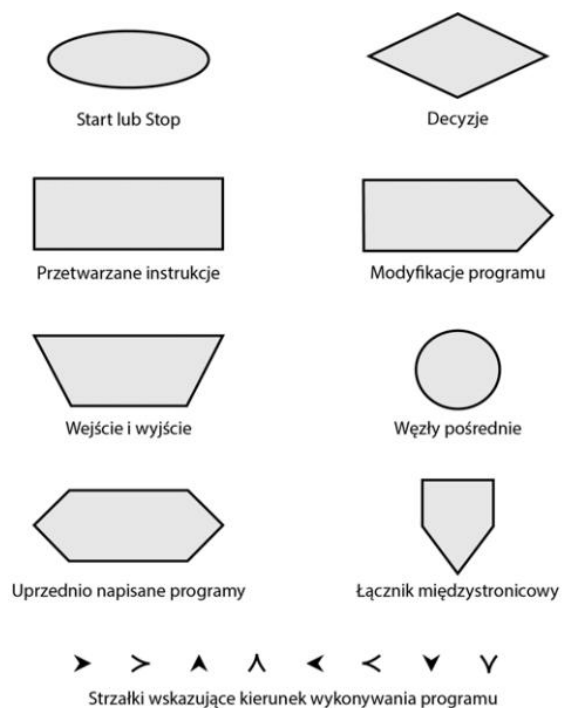




Źródło: książka „Schematy elektroniczne i elektryczne. Przewodnik dla początkujących” autorstwa Stana Gibilisco wydawnictwa Helion

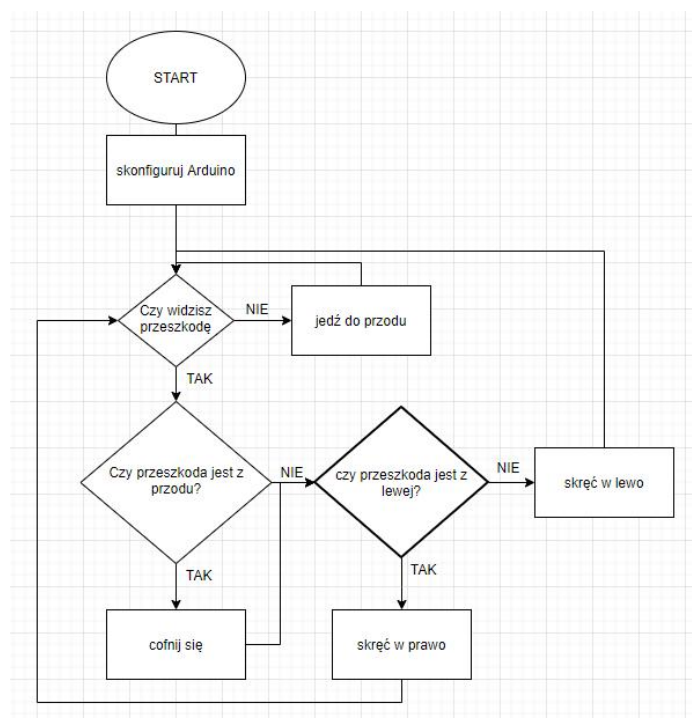
Przyjrzyjmy się powyższemu schematowi, czytanie w tym przypadku rozpoczniemy od lewej strony. Wejście (AC) oznacza że z tej strony podłączeni jesteśmy do prądu przemiennego w Polsce o napięciu 230V i częstotliwości 50Hz. Prąd przechodzi przez bezpiecznik ale też układ zegarowy. Z bezpiecznika prąd musi przejść przez prostownik, oparty zapewne na diodach. Idąc dalej prostownik podłączony jest do jednego z zacisków lampy oraz regulatora. Układ zegarowy generuje sygnał zegarowy dla regulatora zapewne do sterowania regulatorem. Regulator natomiast wysyła prąd zgodnie z sygnałem układu zegarowego do transformatora który zmienia napięcie i łączy dwa pozostałe zaciski lampy stroboskopowej. Widzimy teraz że powyższy układ opisał nam funkcje poszczególnych elementów urządzenia ale dopiero tworząc schemat ideowy będziemy mogli faktycznie sprawdzić z jakich elementów urządzenie się składa i jak są one połączone.

Schematem blokowym procesu nazywamy taki schemat który opisuje działanie programu. Często taki schemat możemy spotkać jako opis programu komputerowego. Dlaczego poruszam temat tego schematu tutaj? Uważam że tworząc programy na Arduino czy same mikro kontrolery musimy znać ten rodzaj schematu blokowego. Powyższy schemat blokowy poruszał kwestie opisowe elementów układu ale tworząc układy elektroniczne z programowalnymi częściami często będziemy musieli posłużyć się schematem procesu. Na początek symbolika takich schematów.



Źródło: książka „Schematy elektroniczne i elektryczne. Przewodnik dla początkujących” autorstwa Stana Gibilisco wydawnictwa Helion

Jako przykład przygotowałem dla Ciebie schemat procesu dla algorytmu omijania przeszkód chyba w najprostszej wersji. Za chwilę postaram się Tobie go wyjaśnić i mam nadzieję że to pozwoli Tobie zrozumieć jak taki schemat można odczytywać.



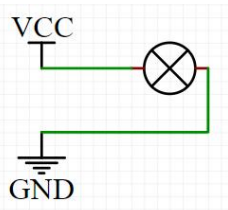
Nasz program startuje i na początku ustawiamy odpowiednio styki naszej płytki Arduino. Część styków pewnie jest ustawione jako wejście, a część jako wyjście. Po konfiguracji przechodzimy do pierwszego bloku decyzyjnego. Gdzie z pomocą odczytów z czujników chcemy stwierdzić czy przed naszym urządzeniem znajduje się jakaś przeszkoda. Jeśli nie została ona wykryta, pojazd jedzie do przodu i wraca do ponownego sprawdzenia. Jeśli natomiast przeszkoda znajduje się przed nami, pragniemy uzyskać odpowiedź czy jest z przodu. W przypadku kiedy przeszkoda jest przed nami, pojazd cofa się i przechodzi do kolejnego bloku decyzyjnego. Można było do tego bloku też przejść prędzej jeśli w wcześniejszym bloku decyzyjnym odpowiedź była negatywna. Kolejnym pytaniem jest czy przeszkoda jest z lewej, jeśli jest skręcamy w prawo i wracamy do pierwszego bloku decyzyjnego który wykrywał czy w ogóle jest jakaś przeszkoda. Jeśli natomiast przeszkody nie ma z lewej oznacza to że musi być z prawej więc skręcamy w lewo. Następnie wracamy na początek. Mam nadzieję że już wiesz czym są takie schematy i w jaki sposób mógłbym je wykorzystać.

4.4.3. Schemat ideowy

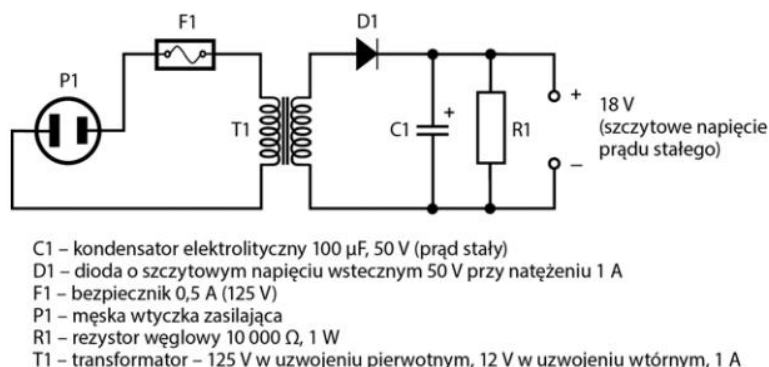
Schemat ideowy, często nazywany w skrócie schematem, przedstawia wszystkie elementy obwodu, a także połączenia pomiędzy nimi. Oznacza to że na schemacie takim znajdziemy wszystkie elementy elektroniczne, wyszczególnienie które znajdują się w obwodzie oraz połączenia między nimi. Często schematy zawierają też dokładne wartości tych elementów i jesteśmy w stanie sporządzić listę tych podzespołów występujących w obwodzie np. rezystor 100 Ω , rezystor 120 Ω , tranzystor npn BC547B, kondensator nF, LED itd. Czyli taki schemat musi nieść ze sobą spis elementów, połączeń między nimi. Często w takich schematach używamy skrótów np. R1, C1, U1, Q1 itd. Dopiero w przypisach możemy dowiedzieć się jaką wartość miał rezystor R1, kondensator C1 czy jaki był to układ U1, tranzystor Q1.

Poruszymy tutaj podstawowe informacje jak tworzyć i czytać schematy. Jeszcze raz zaznaczę że jeśli chciałbym poszerzyć swoją wiedzę polecam książkę na której głównie oparłem ten rozdział czyli „Schematy elektroniczne i elektryczne. Przewodnik dla początkujących” autorstwa Stana Gibilisco wydawnictwa Helion.

Ważną kwestią też tutaj jest z jakich schematów będziemy korzystać przy odczytywaniu lub w jakim programie będziemy je tworzyć. Pozwól że oprę się czy odczytywaniu na wybranych prostych schematach, a omawiając ich tworzenie pokażę Tobie kilka swoich opartych o Arduino.



Zacniemy od czegoś banalnego, na powyższym schemacie widzimy żarówkę podłączoną do źródła napięcia, złącze VCC i GND. Przy tak banalnym przykładzie nie ma co dużo się rozpisywać. Weźmy coś trudniejszego a przy okazji poznajmy podstawowe etykietowanie elementów, czyli umieszczanie skrótów nazw, typów na schemacie.



Źródło: książka „Schematy elektroniczne i elektryczne. Przewodnik dla początkujących” autorstwa Stana Gibilisco wydawnictwa Helion

Powyższe zdjęcie prezentuje schemat zasilacza zawierającego specyfikację komponentów oraz ich zaznaczenie na rysunku. Dzięki takiemu oznaczeniu możemy w prosty sposób uniknąć dodatkowego kłopotu nachodzenia na siebie opisów komponentów zapisanych na rysunku. Ja na przykład często chcę zawrzeć wszystkie opisy na swoich schematach co czasem sprawia że sam schemat staje się mało czytelny.

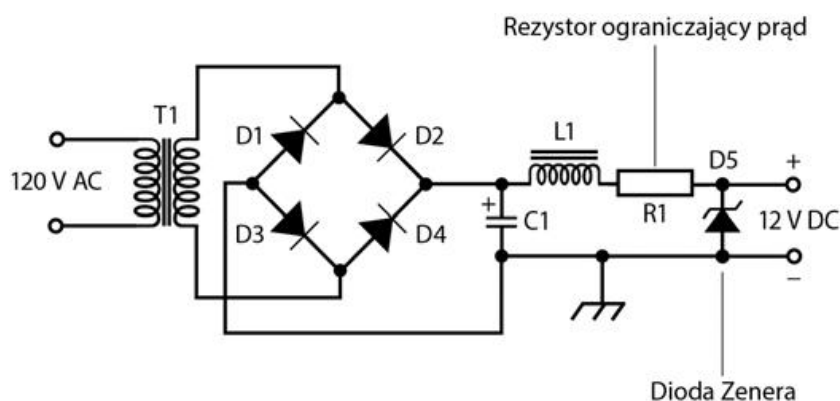
Skrót	Nazwa komponentu
ANT	antena
B	bateria, ogniwo
C	kondensator
CB	płytki obwodu
D	dioda
EP	słuchawka
F	bezpiecznik
GND	uziemia

Skrót	Nazwa komponentu
P	wtyczka
PC	komórka fotoelektryczna
PH	słuchawka
Q	tranzystor
R	rezystor
RFC	dławik przystosowany do pracy z sygnałem o częstotliwości pasma radiowego
RY	przełącznik
S	włącznik



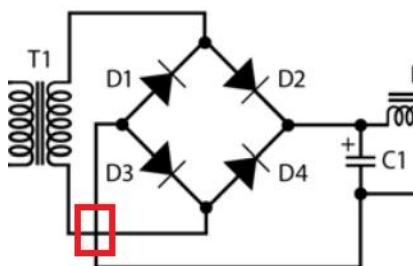
Skrót	Nazwa komponentu
I	żarówka
IC	układ scalony
J	gniazdo lub listwa zaciskowa
K	przekaźnik
L	cewka
LED	dioda LED
M	miernik
NE	neonówka

Skrót	Nazwa komponentu
SCR	krzemowy prostownik sterowany
T	transformator
TP	zacisk lub punkt pomiarowy
U	układ scalony
V	lampa próżniowa
Y	kryształ kwarcu
Z	obwód prefabrykowany

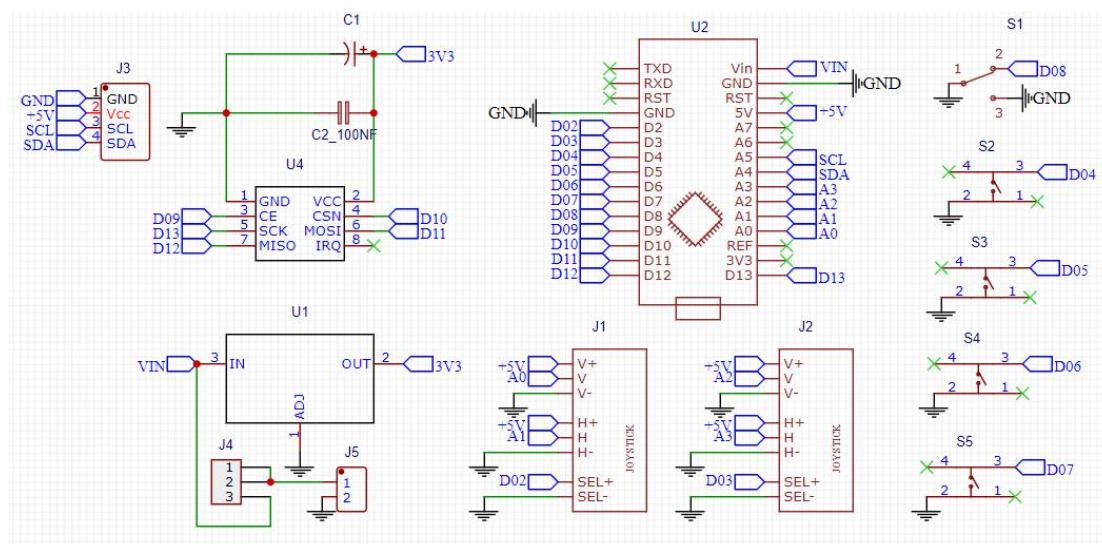


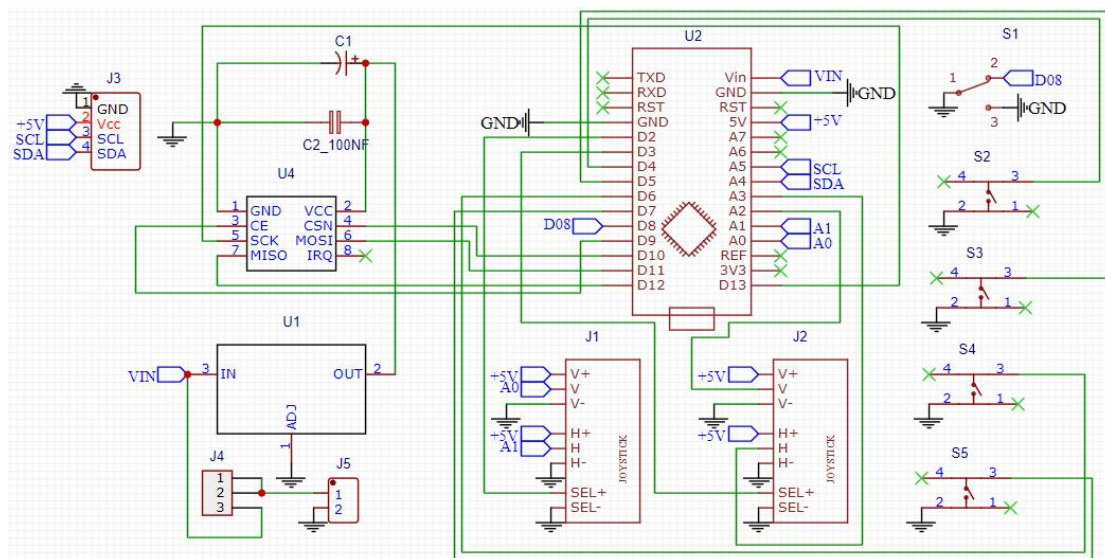
Źródło: książka „Schematy elektroniczne i elektryczne. Przewodnik dla początkujących” autorstwa Stana Gibilisco wydawnictwa Helion

Przyjrzyjmy się innemu schematowi i opiszmy sobie tym razem połączenia między elementami. Z powyższą tabelką i symbolami jakie Tobie pokazałem myślę że jesteśmy w stanie nazwać poszczególne elementy schematu ale czy wiemy jak są one połączone? Zaczniemy od takich kropek na przewodach połączeniowych. Nazywamy je węzłami, oznacza to że w tym miejscu łączą się połączenia, ścieżki. Jeśli połączenia przecinają się ale nie ma kropki oznacza to że nie są one ze sobą połączone. W powyższym schemacie widzimy połączenie z uzwojenia transformatora T1 idące do diody D3 i D4, jednocześnie mamy połączenie z diody D1 i D3 idące w kierunku kondensatora C1.



Jeśli chodzi o nauczenie się czytania schematów przyjdzie to z czasem, jeśli znasz symbolikę i zasadę łączenia komponentów przyjdzie to z czasem, jak będziesz zdobywał doświadczenie. Chciałbym jeszcze Tobie pokazać jeden z moich schematów gdzie zastosowałem Arduino i podstawowe komponenty. Ja ten schemat zrobiłem w aplikacji EasyEDA o której powiem Tobie jeszcze w jednym z kolejnych rozdziałów.

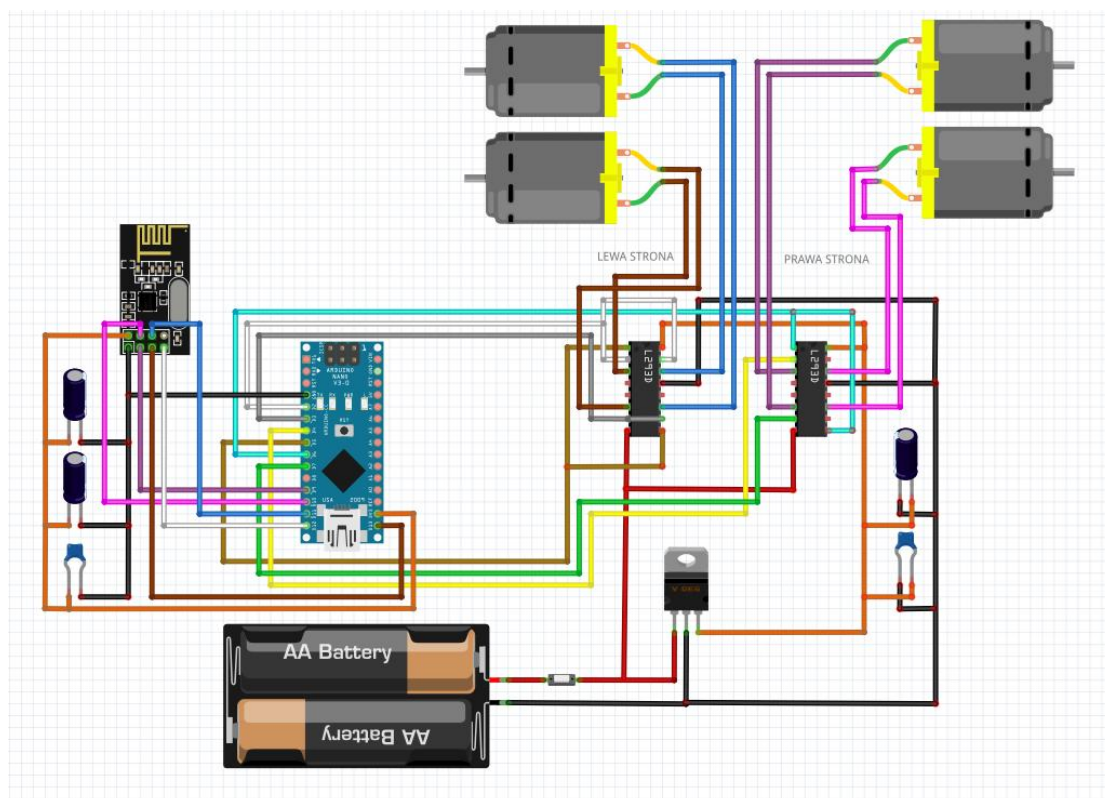




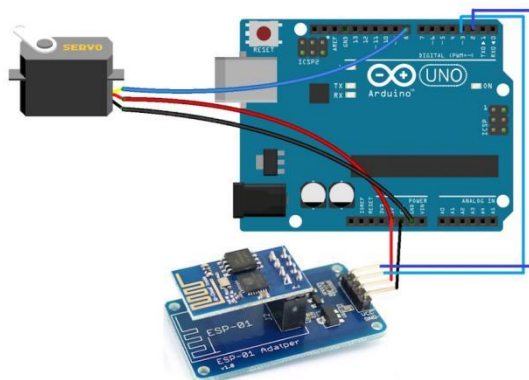
Możemy zauważyć że teraz schemat jest mniej czytelny ale oczywiście bardzo często właśnie takie schematy spotkacie. Kolejnym symbolem jest  , jest to znam informujący Nas i program że ten styk jest bez połączenia z innym elementem. Ponieważ część programów z naszego schematu tworzy projekt płytki PCB temu kompilatorowi jest potrzebna taka informacja. Podsumowując co to jest za schemat? Jest to jeden z moich projektów pilota RC opartego na Arduino Nano, wyświetlaczu OLED SSD1306, komunikacja z pomocą NRF24L01, a sterowanie to dwa joysticki analogowe oraz kilka przycisków.

4.4.4. Schemat „wizualny”

Schemat połączeń, wizualny to mój pomysł na nazwanie tego co bardzo często spotkamy szukając informacji na temat Arduino i połączeń z jakimś modulem. Schematem wizualnym nazywam takie które prezentują połączenia różnych komponentów gdzie widzimy rysunek lub zdjęcie elementu.



Na powyższym schemacie widzimy wszystkie elementy obwodu jako obrazy, rysunki połączone ze sobą za pomocą kolorowych przewodów. Jeśli obwód jest prosty to w prosty sposób możemy zrozumieć panujące tam połączenia ale jeśli jest w nim dużo połączeń trzeba być uważnym. Bardzo łatwo pomylić miejsca gdzie połączenia się łączą lub przenikają przez siebie.



Źródło: <https://electronnicproject.blogspot.com/2018/01/make-web-controlled-servo-with-arduino.html>

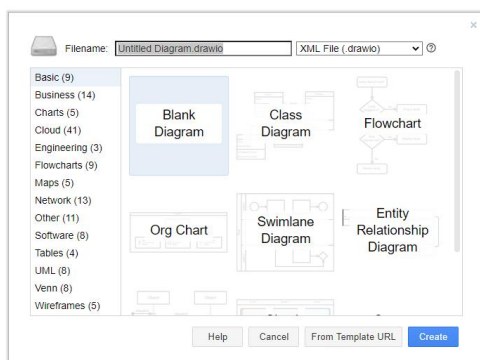


Autor: Zbigniew „HardeN” Brożbar www.robomaniak.pl

W internecie znajdziemy również takie schematy gdzie widzimy fizyczne zdjęcia elementu. Arduino posiada tak dużo modułów kompatybilnych ze sobą że często zdarza się że oprogramowanie jakie używamy może nie posiadać go w swojej bazie rysunków.

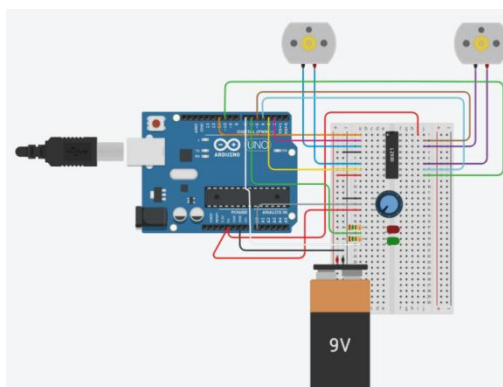
4.4.5. Oprogramowanie

Szybkim krokiem przechodzimy do oprogramowania. Pokażę Tobie kilka fajnych programów które na pewno ułatwią Tobie tworzenie schematów.



- **diagrams.net** - aplikacja internetowa do tworzenia schematów blokowych. Znajdziemy tutaj całkowicie darmowe narzędzie które bardzo ułatwi Tobie tworzenie w/w schematów. Schematy możemy zapisywać na swoim komputerze lub w chmurze.

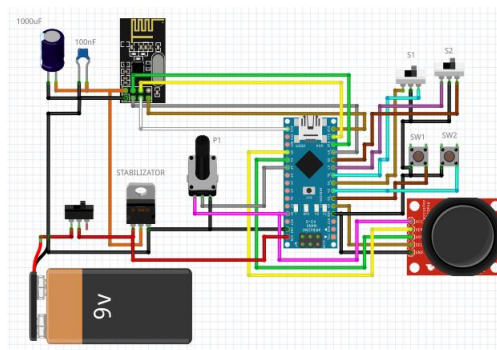
adres strony: <https://app.diagrams.net/>



- **TINKERCAD** - aplikacja internetowa która początkowo była narzędziem do modelowania 3D ale od jakiegoś czasu posiada bardzo dobrze działający symulator elektroniki. Możecie tutaj stworzyć układ i sprawdzić jego działanie ale ja często używam go jako aplikację do tworzenia schematów wizualnych.

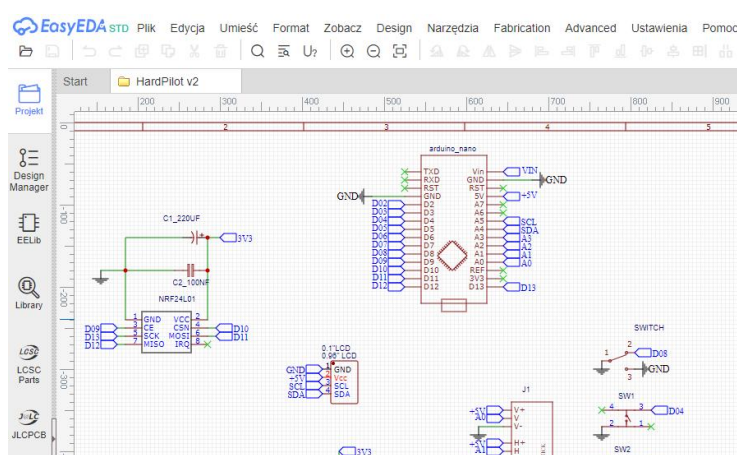
adres strony: <https://www.tinkercad.com/>





- **Fritzing** - darmowa aplikacja do tworzenia schematów wizualnych ale również schematów ideowych i innych. Aplikacja posiada bardzo dużą społeczność która tworzy do niej różnego rodzaju paczki m.in. rysunkami komponentów, modułów do Arduino.

adres strony: <https://fritzing.org/>



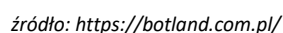
EasyEDA - aplikacja internetowa do tworzenia schematów ideowych oraz płytek PCB. Zintegrowana z firmą JLCPCB która tworzy płytki PCB. Aplikacja działa w chmurze, projekty zapisywane są na serwerach EasyEDA.

adres strony: <https://easyeda.com/>

5. Arduino NANO

Arduino Nano to jedna z mniejszych wersji płytki arduino. Według mnie jest to jedna z lepszych płytek do rozpoczęcia swojej przygody w świecie elektroniki. Płytką opartą jest na układzie ATmega328 o rdzeniu 8-bitowym. Ilość funkcji jakie oferuje nam Nano jest podobna do najpopularniejszej wersji Arduino UNO. Brakuje kilka funkcji jak zasilanie z gniazda DC czy też układu ATmega16u2 ale w zamian do komunikacji USB mamy CH341.





5.2. Porty wejścia/wyjścia

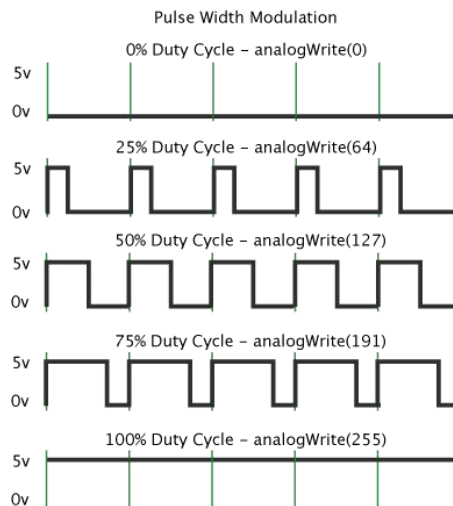
W sumie na płytce Arduino mamy 22 porty wejścia/wyjścia. Z czego te porty dzielimy na trzy rodzaje: porty cyfrowe, porty analogowe i porty PWM. Portem wejścia jest VIN czyli zasilanie naszego Arduino oraz dwa wyjścia z regulatora napięcia czyli 5V oraz 3.3V.

5.2.1. Porty cyfrowe

Na płytce posiadamy 14 portów cyfrowych od 0 do 13 ale ja proponuję używać od 2 do 13. Porty cyfrowe charakteryzują się tym że przyjmują dwa stany LOW czyli napięcie 0-2V lub HIGH czyli napięcie 3V - 5V. Każdy z tych portów może być wejściem lub wyjściem. Dodatkowo możemy ustawić port jako wejście pullup czyli podbite w górę. Oznacza to że w programie ustawiając funkcję pinMode z atrybutem INPUT_PULLUP stan pinu jako wejście ustawia się na HIGH. Działa to dlatego bo mikrokontroler ATmega ma wewnętrzne rezystory podciągające do zasilania. Piny cyfrowe skonfigurowane jako wyjście mogą dostarczyć maksymalnie 40mA ale suma wszystkich wyjść nie może być większa niż 500 mA. Dodatkowo pin 13 jest podłączony do wbudowanej diody LED.

5.2.2. Porty PWM

Porty PWM modulują szerokość impulsu HIGH. Modulacja szerokości impulsu to technika uzyskania wyników analogowych za pomocą portów cyfrowych. Sterowanie cyfrowe służy do tworzenia impulsu prostokątnego, jest to sygnał przełączenia między włączeniem (HIGH) i wyłączeniem (LOW). Czas trwania sygnału HIGH w czasie nazywamy szerokością impulsu. Arduino Nano posiada 6 pinów PWM, są to 3, 5, 6, 9, 10, 11. Piny PWM są pinami cyfrowymi wyjścia sterowane z pomocą funkcji *analogWrite()*.



źródło: <https://www.arduino.cc/en/Tutorial/Foundations/PWM>

Powyższy rysunek obrazuje jak wygląda sygnał PWM o różnym wypełnieniu, szerokości impulsu. Minimalna wartość wypełnienia to 0, a maksymalna 255. Sterując wykorzystujemy funkcję `analogWrite(pin, wypełnienie)`. Jak działa w końcu ten impuls PWM ? Wyobraźmy sobie diodę LED którą sterujemy przez pin PWM. Jeśli wypełnienie ustawimy na 50% czyli wartość 127 dioda będzie świeciła na połowie mocy. Działa to w ten sposób że sygnał HIGH przerywany jest co równy czas. Identyczny czas mamy stan HIGH oraz LOW.

6. Bibliografia

6.1. Książki

- „Arduino dla zaawansowanych” autor Rick Anderson i Dan Cervo
- „Arduino dla początkujących. Kolejny krok. Poznaj tajniki Arduino!” autor Simon Monk
- „Elektronika od praktyki do teorii.” autor Charles Platt - wydanie II
- „Podstawy elektroniki w praktyce” część 1 i 2 autor Anna Tąpolska
- „Schematy elektroniczne i elektryczne. Przewodnik dla początkujących” autor Stan Gibilisco

6.2. Strony internetowe

- <https://pl.wikipedia.org/>
- <https://edufinf.waw.pl/>
- <https://forbot.pl/>
- <https://www.arduino.cc/>
- <https://ucgosu.pl/>
- <https://plociennik.info/>
- <https://botland.com.pl/>
- <https://cpp0x.pl/>
- <http://www.mif.pg.gda.pl>

