Gimnazjum nr 3 w Zabrzu

Projekt

Elektronika

Autorzy:
Bartek Mazur
Krzyś Dziembała

Klasa 3gb

20 kwietnia 2016

Spis treści

Wstęp

Ten projekt jest dedykowany ludziom na każdym poziomie znajomości tematu. Przez kolejne rozdziały będziemy uzupełniać podstawowe informacje potrzebne do zbudowania prostych układów elektronicznych, aby w kolejnej części zacząć przedstawiać podstawowe schematy i proste programy. Doświadczony w tej dziedzinie czytelnik może je po prostu pominąć (nie wszyscy chcą się uczyć programowania schematami blokowymi). *Nota dla zaawansowanych czytelników: Używamy procesorów "atmel", a do programowania środowiska Arduino. Cały kod używany w prezentacji będzie udostępniony na GitHub'ie. (Załącznik 1.)

1. Programowanie Arduino

Arduino aby działać musi zostać zaprogramowane. Programuje się je w języku Arduino, opartym na językach C/C++. Program jest kompilowany, czyli przetwarzany na język zrozumiały dla urządzenia, oraz na nie wgrywany za pomocą Arduino IDE, o którym więcej w sekcji "1.3. Środowisko".

1.1. Składnia języka używanego w środowisku C++

Język Arduino można podzielić na 3 główne części:

1. Struktura
2. Zmienne
3. Funkcje

1.1.1. Struktura

Aby program działał niezbędne są dwie funkcje:

- void setup() - wykonywana tylko raz na początku programu
- void loop() - wykonywana cały czas po wykonaniu funkcji "setup"

Wśród struktur możemy wyróżnić także:

- if() - wykonuje kod zawarty w nawiasach klamrowych, jeśli warunek w zwykłych nawiasach jest spełniony. Do porównania wartości stosuje się:
  - > - jest większe
  - < - jest mniejsze
  - == - jest równe
  - != - jest różne od
  - <= - mniejsze bądź równe

  - `>=` - większe bądź równe

- else - wykonuje kod zawarty w nawiasach klamrowych, jeśli warunek w if nie jest spełniony. **Używany tylko razem z "if".** Przykład:

```
if(x > y){
  Serial.println("x jest większe od y.");
} else {
  Serial.println("x nie jest większe od y.");
}
```

- for - jest to pętla, której zawartość w klamrach zostanie tyle razy wykonana, ile razy spełniony jest warunek podany na wejściu. Budowa:
  for(indeks, warunek dla którego funkcja ma się wykonywać, co zrobić z indeksem po wykonaniu)
  Przykład pętli for, która wypisze wartości od 0 do 2 włącznie:

```
for (int x=0;x<3;x++){
  Serial.println(x);
}
```

- while() - pętla, która wykonuje kod w nawiasach klamrowych, jeśli warunek w zwykłych nawiasach jest spełniony.
  Przykład pętli while wypisującej wartości zmiennej x i zwiększającej tę wartość jeśli jest ona mniejsza niż 3:

```
while(x<3){
  Serial.println(x);
  x++;
}
```

- do... while() - pętla podobna do pętli "while", ale kod zostanie wykonany co najmniej raz, nawet jeśli warunek nie jest spełniony.
  Przykład pętli do... while, która wypisze wartość x tylko raz (choć warunek **nie** jest spełniony):

```
int x = 3;
do{
  Serial.println(x);
}while(x<1);
```

Do tej kategorii możemy zaliczyć także niektóre symbole:

- ; - stawiany na końcu każdej (nie pustej) linii kodu. Nie jest niezbędny, jeśli linia jest zakończona znakiem }
- {} - w nawiasach klamrowych jest zawarty kod każdej funkcji, pętli i "if", oraz "else"
- // - stawia się przed komentarzem. Ten typ komentarzy zaczyna się od tych znaków i kończy się z końcem linijki. **Komentarze są ignorowane przez kompilator**
- /**/ - komentarz wieloliniowy. Komentarz wstawia się za "/*", a kończy się go "*/".
- = - przypisuje zmiennej określoną wartość. Przykład przypisania zmiennej "x" typu "string" wartości "wartość":

  ```
  char x[] = "wartość";
  ```

- + - oznacza dodawanie
- - - oznacza operację odejmowania
- * - mnożenie
- / - dzielenie
- % - modulo (reszta z dzielenia). Działanie: x % y - reszta z dzielenia x przez y
- ++ - powiększ o 1. Przykład kodu zwiększającego zmienną x o 1:

```
x++;
```

- – - zmniejsz o 1. Wykorzystanie w ten sam sposób co –
- += - powiększ o. Przykład kodu zwiększającego zmienną x o 7:

  ```
  x+=7;
  ```

  Równoważne z kodem

  ```
  x = x + 7;
  ```

- -= - pomniejsz o. "x -= y" jest równoważne z "x = x - y"
- *= - przemnóż przez. "x *= y" równoważne do "x = x*y"
- /= - podziel przez. "x /= y" = "x = x/y"
- %= - zapisz jako resztę z dzielenia przez. "x %= y" jest równoważne z "x = x%y".

Należy pamiętać również o:

- #include - używany na początku kodu przed jakąkolwiek funkcją, aby dołączyć biblioteki. **Nie stawiamy po nim średnika!** Przykład:

  ```
  \#include <Keyboard.h> //dodaje bibliotekę wprowadzającą
    obsługę klawiatury
  ```

- #define - używany na samym początku kodu prze jakąkolwiek funkcją. Za jego pomocą możemy zdefiniować jaką wartość będzie mieć fragment tekstu. Działa tak jakby zamiast ciągu znaków zaraz po słowie "define" były te oddzielone od nich spacją. **Nie stawiamy po nim średnika!** Przykład kodu wypisującego co 1s tekst zdefiniowany na początku.

```
#define tekst "Hej!"
#define przerwa 1000

void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.println(tekst);
  delay(przerwa);
}
```

1.1.2. Zmienne

Dane przetwarzane przez program przechowywane są jako zmienne. W zależności od rodzaju zmiennej może ona przechowywać różne wartości.
Wyróżniamy następujące typy zmiennych:

- void - używany tylko do tworzenia funkcji, które nie zwracają żadnej wartości.

- boolean - zmienne typu boolean przechowują tylko wartości **true** i **false**. Zamiast "false" można użyć 0, a każda niezerowa liczba zostanie uznana za "true".

- char - typ danych zawierający bajt pamięci. Przechowuje wartość znaku. Podczas nadawania zmiennej wartości należy zapisać ją w pojedynczym cudzysłowie. Jednak znaki przechowywane są jako odpowiadające im liczby zgodnie z kodowaniem ASCII. Zmienna "char znak = 'A';" może być zapisana również tak: "char znak = 65;". Korzystając z tej własności możemy np pisząc "znak += 32;" zmienić wartość tej zmiennej z 'A' na 'a', czyli zmienić dużą literą na małą (i odwrotnie).

- byte oraz unsigned char - przechowuje 8 bitową liczbą od 0 do 255. Zalecane jest używanie "byte" zamiast "unsigned char".

- int - przechowuje wartości liczbowe. Zakres wartości zależy od procesora. Dla Arduino Uno (i innych urządzeniach opartych na procesorze ATMega) jest 16-bitowa i ma zakres od -32768 do 32767.
- word oraz unsigned int - od zwykłego różni się tym, że przechowuje tylko wartości nieujemne. Jego zakres dla Arduino Uno i innych Arduino opartych na procesorach ATMega jest od 0 do 65535.
- long - przechowuje wartości liczbowe. Ma rozmiar 32 bitów. Przyjmuje wartości od -2147483648 to 2147483647. Po liczbie należy dać literę "L" (long liczba = 51523487L).
- unsigned long przechowuje 32 bitową liczbę. Ma zakres od 0 do 4294967295.
- short - dla wszystkich Arduino jest taki sam jak int na procesorach ATMega.
- float - typ zmiennej dla liczb zmiennoprzecinkowych. Zajmują 32 bity pamięci. Mogą przechowywać wartości od -34028235E+38 do 3.4028235E+38. Liczby zmiennoprzecinkowe **nie są dokładne**! Na przykład "6.0 / 3.0" może nie być równe "2.0". Mają dokładność **tylko 6-7 cyfr znaczących**! Operacje na zmiennych zmiennoprzecinkowych **trwają dłużej** niż na liczbach całkowitych. Należy również pamiętać, że nie umieszczenie kropki przy podawaniu wartości spowoduje, że zmienna traktowana będzie jako "int".
- double - zmienna zmiennoprzecinkowa o podwójnej dokładności względem "float". Na procesorach ATMega zajmuje 32 bity pamięci, a na Arduino Due 64 bity. **W przypadku Arduino NIE jest dokładniejsza od zmiennej typu "float".**
- string - tablica znaków. Najprościej tworzy się ją tak:

```
char nazwa[] = "Zawartość zmiennej nazwa";
```

- String - obiekt przechowujący ciągi znaków. Umożliwia więcej niż tablica znaków. Posiada tzw. metody, które ułatwiają m.in. łączenie ich, wyszukiwanie znaków i podmienianie ich.

1.1.3. Funkcje

Funkcje to segmenty kodu, które można wywołać wiele razy i z różnymi parametrami. Zapisuje się je w ten sposób:

```
typ nazwa(typP parametr1, typP parametr2, ...){
  kod;
  return(zwracanaWartosc);
}
```

gdzie typ oznacza typ wartości zwracanej przez funkcję (np. void, int, boolean, itd.). W zwykłych nawiasach umieszcza się nazwy parametrów, które mogą zostać funkcji przekazane i ich typ (np. int, boolean, itd.). Na końcu funkcji zazwyczaj umieszcza się return(wartosc). Nie dotyczy to jednak funkcji typu "void", które nie zwracają żadnych wartości.

W sekcji "1.1.1. Struktura" wymieniliśmy już funkcje "setup" i "loop". Jednak Arduino ma więcej predefiniowanych funkcji, z których wymienimy kilka podstawowych.

- pinMode(x, y) - funkcja określająca jaką rolę (y) ma pełnić pin cyfrowy (digital pin) o numerze "x". "y" może przyjmować następujące wartości:
  - OUTPUT - wyjście
  - INPUT - wejście
  - INPUT_PULLUP - wejście z wewn. opornikiem
- digitalWrite(x, y) - funkcja określająca jaki stan (y) ma przyjąć pin cyfrowy (zdefiniowany jako "OUTPUT") o numerze "x". "y" może przyjmować następujące wartości:
  - LOW - masa (-)
  - HIGH - zasilanie (+)
- digitalRead(x) - funkcja zwracająca stan wejściowego pinu cyfrowego "x". Zwraca tylko wartości HIGH i LOW

- milis() - podaje czas w milisekundach
- delay(x) - zatrzymuje wykonywanie programu na "x" milisekund (1000ms=1s).
- attachInterrupt(digitalPinToInterrupt(x), func, mode) - dodaje przerwanie wykonujące funkcję "func", jeśli z cyfrowym pinem wejściowym "x" stanie się to co określono jako "mode". Piny obsługujące przerwania są różne w zależności od Arduino. Przed użyciem attachInterrupt() radzę zobaczyć załącznik 3 Funkcja func **nie może pobierać parametrów, ani zwracać żadnych wartości** (definiowana jako void). "mode" może przyjmować następujące wartości:
  - LOW - jeśli stan pinu to "LOW".
  - CHANGE - jeśli zmieni się wartość (LOW->HIGH lub odwrotnie).
  - RISING - jeśli wartość zmieni się z LOW na HIGH.
  - FALLING - jeśli wartość zmieni się z HIGH na LOW.
- detachInterrupt(digitalPinToInterrupt(x)) - usuwa przerwanie z pinu "x".
- Serial - umożliwia komunikację przez port szeregowy. Korzystamy do tego z "monitora portu szeregowego" w Arduino IDE. Używa się jako Serial.func(), gdzie funkcji "func()" jest naprawdę wiele. Wymienię jednak kilka podstawowych:
  - begin(x) - ustala częstotliwość przesyłania danych w bitach na sekundę (baud) jako "x". "x" może przyjmować wartości 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 lub 115200.
  - print(x) - przesyła wartość x przez port szeregowy.
  - println(x) - działa jak Serial.print(), ale kolejny komunikat będzie w nowej linijce.

  Jeśli chcecie zobaczyć wszystkie funkcje Serial.func() zobaczcie załącznik 4.

Jak wcześniej napisaliśmy istnieje wiele predefiniowanych funkcji, których tu nie wymieniliśmy, ale jeśli chcecie je wszystkie poznać zobaczcie załącznik 5 i kolumnę "Functions".

1.2. Styl pisania

Nie mówimy tutaj czysto o algorytmice (przeszukiwanie danych, analiza obrazu itp.) jednak jest pewien schemat postępowania. Arduino składa się z czujników (sensorów) oraz odbiorników.
Możemy na bieżąco analizować i przetwarzać dane. Na samym początku może się to wydawać niezrozumiałe ale przy odrobinie wprawy i obycia z urządzeniem zmienią podejście.

Podstawą jest umiejętność skonstruowania algorytmu, czyli ciągu poleceń do wykonania. Można zapisać to tak:
Czytaj wartość czujnika `->` analiza `->` akcja zależna od wyniku analizy (np. zapal światło, włącz silnik).

1.2.1. Schemat ogólny

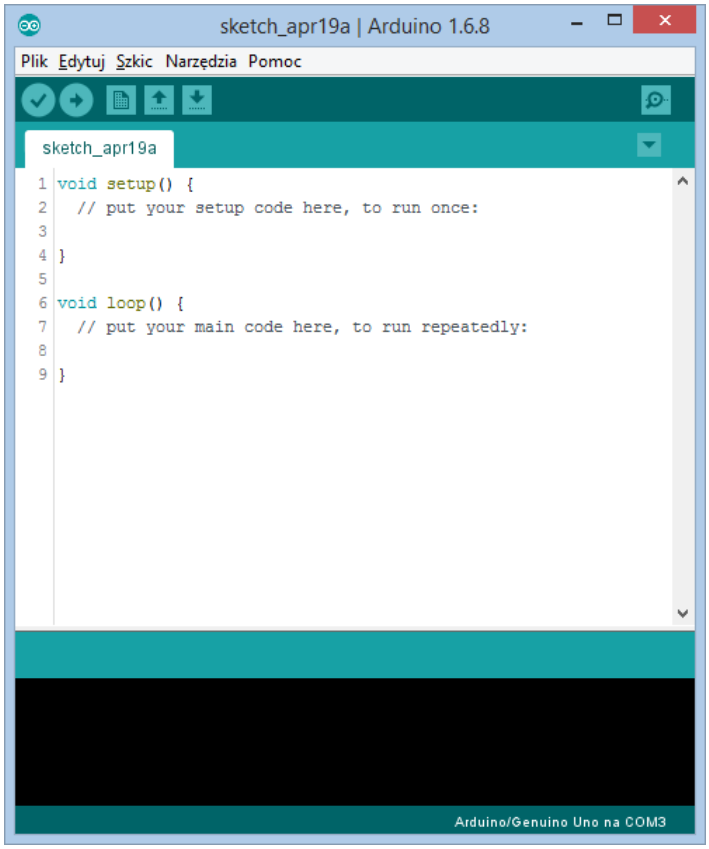
Przyjrzymy się ogólnej postaci programów. Zostaną one wytłumaczone na przykładzie schematu blokowego aby nawet niewprawiony czytelnik był w stanie cokolwiek zrozumieć.

1.2.2. Przykłady z życia wzięte

W tej części przeanalizujemy kod "normalnie" używany

1.3. Środowisko

Środowisko - miejsce w którym dzieją się cuda, a mianowicie powstają nasze programy. Jak każdy język, Arduino również ma własne środowisko (Arduino IDE). Środowisko umożliwia nam edytowanie programów oraz kompilację i wgrywanie ich na urządzenie.
Wygląda ono w ten sposób: Ilustracja 1.



Ilustracja 1: Arduino IDE

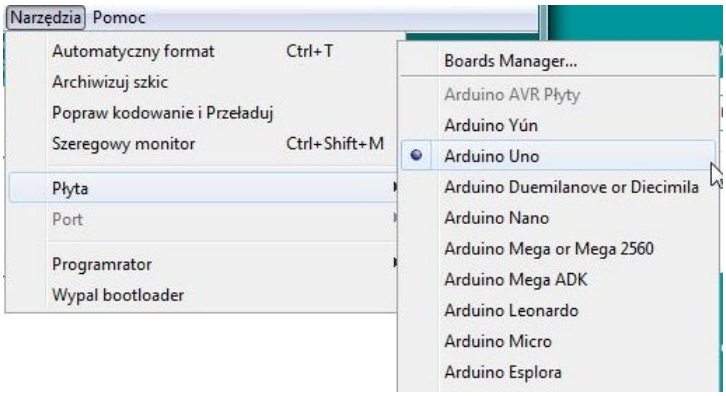
1.3.1. Skąd pobrać

Pobieramy je ze strony (Załącznik 2) zgodne z wersją naszego systemu operacyjnego. Następnie instalujemy, wykonując polecenia instalatora ew. (w przypadku Windowsa) rozpakowujemy zip'a do dowolnej lokalizacji i z niej uruchamiamy arduino.exe.

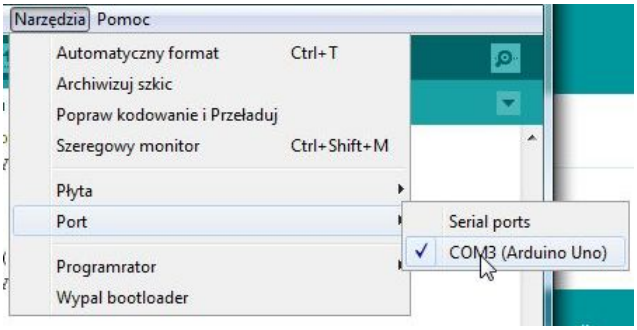
1.3.2. Pierwsze kroki

Programowanie jest intuicyjne ale musimy pamiętać o kilku rzeczach. A mianowicie należy ustawić płytkę (Ilustracja 2), oraz port szeregowy (Ilustracja 3)

[1, 2]



Ilustracja 2: Ustawianie płytki



Ilustracja 3: Ustawianie portu szeregowego

2. Podstawy elektroniki

2.1. Fizyka w elektronice

Fizyka w elektronice nie różni się od tej poznanej w gimnazjum. Był cały dział 'prąd'. Ten dział właśnie nam się przyda. Nie zagłębiając się we wszystkie szczegóły, powtórzymy prawo Ohma:

$$I = \frac{U}{R}$$

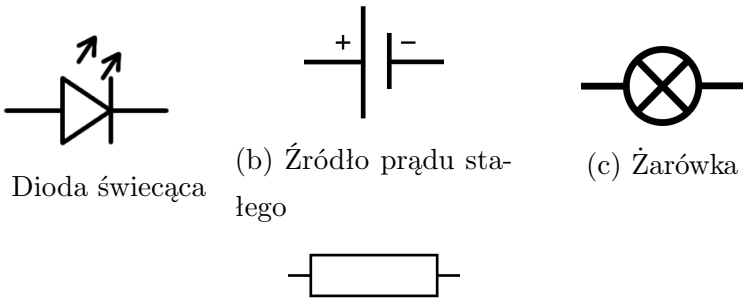
Zadanie z tym związane (pojęcia będą wytłumaczone w dalszej części):
Mamy diodę oraz źródło zasilania 5V. Jaki rezystor musimy podłączyć aby prąd płynący przez diodę był ok 20mA

2.2. Elementy elektroniczne i ich symbole

2.2.1. Symbole

Elektronika ma swój własny język. Przypomnijmy oznaczenia niektórych elementów z elektroniki:

- Ilustracja 4a-dioda świecąca
- Ilustracja 4b-źródło prądu stałego
- Ilustracja 4c-żarówka
- Ilustracja 4d-rezystor



(a) Dioda świecąca

(b) Źródło prądu stałego

(c) Żarówka

(d) Rezystor

Ilustracja 4: Symbole

2.2.2. Definicje tych elementów

Żarówka - jest to źródło światła (i ciepła) elektrycznego, poprzez żarzenie się trudno topliwego materiału (często wykorzystywany jest drucik wolframowy przez który płynie duży prąd). Sprawność ok. 8-10 lumenów/wat.

Źródło prądu stałego - jak nazwa wskazuje jest to miejsce, w którym "powstaje prąd" jednostką jest napięcie wolt[V]. W elektronice powszechnie stosuje się napięcie 3,3V i 5V. Dla porównania w gniazdku jest 220-230V.

Rezystor - jest wykorzystywany do ograniczenia płynącego prądu w obwodzie. Zamienia energię elektryczną w ciepło. Rezystory posiadają również własny kod o którym więcej w sekcji "2.3. Jak czytać opór"

Dioda świecąca - element elektryczny który przewodzi prąd tylko w jedną stronę. Stosowane w wyświetlaczach LED jak również w pilotach (na światło podczerwone) Charakteryzuje się dosyć dużą sprawnością 26-300 lumenów/W.

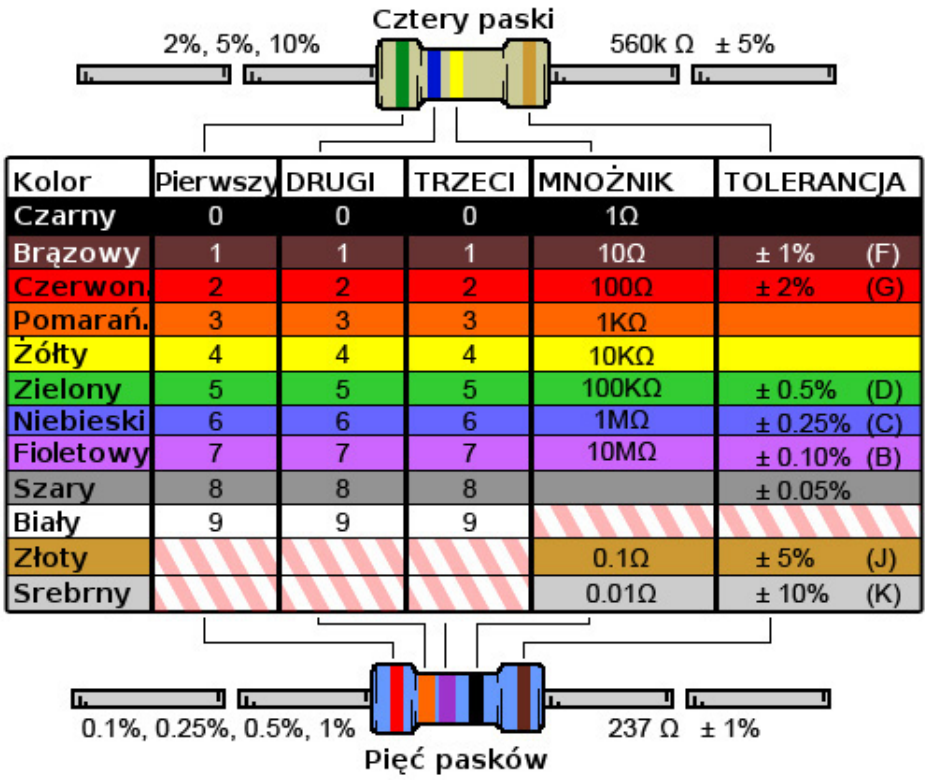
2.2.3. Zastosowanie symboli

Do rysowania schematów używamy tych właśnie symboli. Symbole te są znane na całym świecie więc gdy narysujemy to co jest na ilustracji 4a każdy będzie wiedział, że to dioda. Na początku nie będziemy rysować bardzo skomplikowanych układów ale należy wiedzieć że one istnieją. Przydają się przy czytaniu niektórych instrukcji (np. do jakiegoś czujnika).

2.3. Jak czytać opór

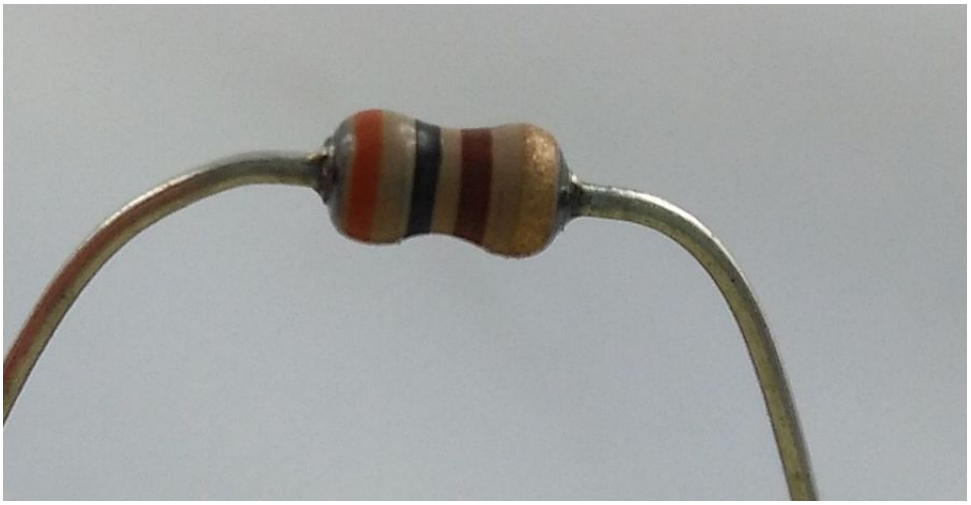
Nie ma tutaj żadnej filozofii. Musimy po prostu 'podstawić' nasz opornik i przeczytać. Spróbujcie sami - ilustracja 6. Rozwiązanie na końcu.

W systemie znakowania paskowym 2 pierwsze paski oznaczają wartość rezystancji którą czytamy jako jedną liczbę, a 3 pasek mnożnik przez który należy pomnożyć te dwie pierwsze liczby. Czwarty pasek to dopuszczalna tolerancja. - tolerancja to zakres błędu. Np dla opornika 10[ohm] +- 10% znaczy, że opór będzie nam się wahał od 9-11[ohm]. Użyjcie do tego ilustracji 5.



Kolor	Pierwszy	DRUGI	TRZECI	MNOŻNIK	TOLERANCJA
Czarny	0	0	0	1Ω	
Brązowy	1	1	1	10Ω	± 1% (F)
Czerwon.	2	2	2	100Ω	± 2% (G)
Pomarań.	3	3	3	1KΩ	
Żółty	4	4	4	10KΩ	
Zielony	5	5	5	100KΩ	± 0.5% (D)
Niebieski	6	6	6	1MΩ	± 0.25% (C)
Fioletowy	7	7	7	10MΩ	± 0.10% (B)
Szary	8	8	8		± 0.05%
Biały	9	9	9		
Złoty				0.1Ω	± 5% (J)
Srebrny				0.01Ω	± 10% (K)

Ilustracja 5: Kod paskowy rezystorów



Ilustracja 6: Przykładowy rezystor

2.4. Domowy warsztat

2.4.1. O co chodzi

W tym rozdziale czytelnik dowie się jak zacząć przygodę z programowaniem jeśli nigdy nie miał styczności z tym. Po wybraniu swojego poziomu są umieszczone tabelki z zadaniami oraz z listą zakupów .

2.4.2. Podstawy podstaw - poziom 1

Jest to idealne połączenie dla czytelników, którzy zaczynają przygodę z elektroniką oraz z programowaniem jednocześnie. Dzięki temu nauczy się podstaw z tych dziedzin. To, co czytelnik ma zrobić aby posiąść tę wiedzę będzie omówione w dalszej części dokumentu. Koszyk odpowiedni w na tym poziomie znajdziemy w tab: Zawartość koszyka na poszczególnych poziomach. Poziom 1: -któreś Arduino -płytka stykowa -kabelki (damsko-męskie i męsko-męskie) czujnik, oporniki (do diod)

2.4.3. Advanced beginner - poziom 2

Czytelnik: -zna podstawową składnię używaną w Arduino, -zna różnice między Arduino Nano oraz Arduino Uno, -potrafi uruchomić diodę wciskając przycisk, -spędził co najmniej 5h w środowisku Arduino

Gdy czytelnik posiądzie wiedzę minimalną, może przejść do kolejnego poziomu wtajemniczenia w elektronikę, prócz poszerzenia naszego warsztatu, zmiany trudności zadań, czytelnik musi wykazać się swoją kreatywnością, ponieważ w tym momencie pojawiają się schody.

2.4.4. Expert - poziom 3

Jako, że jest to poradnik dla początkujących i średnio zaawansowanych, wyróżnimy tylko 3 poziomy wtajemniczenia. W tym poziomie nie ma już stricte list zakupów, ponieważ Expert sam dobrze wie co chce robić.

2.4.5. Lista zakupów

Jak już wcześniej zostało napisane, w tym rozdziale zostanie poruszony temat listy zakupów oraz jak kupować

Najlepszą stroną do kupowania elektroniki jest allegro, wszystkie dostępne (mniej lub bardziej podstawowe) czujniki znajdziemy w kategorii 'Arduino' (opcjonalnie nazwa czujnika np. magnetometr).

Jeżeli mamy czas i potrzebujemy więcej czujników, najlepiej zamówić je w Chinach poprzez Aliexpress. Nie należy się obawiać, że nie przyjdą jednak musimy uzbroić się w cierpliwość (najdłużej czekałem ok 70 dni).

2.4.6. Zadania

Tak jak w przypadku zakupów, problemy zadaniowe mają swoje własne serwisy, które ułatwiają pracę. Jest kilka ważnych zasad związanych z tym: szukamy po angielsku, problem wpisujemy w google i szukamy na Stackoverflow, do Arduino polecam forum Arduino też (przykładowe problemy)

2.4.7. Reszta śmieci do zedytowania

Zadania mają charakter edukacyjny i wszystkie uczą różnych knifów w programowaniu Arduino. Czasami jednak czytelnik będzie musiał wykazać się umiejętnością szperania w sieci. Podpowiedzi do zadań będą umieszczone w internecie. Jak zacząć: Gdy kupiliśmy Arduino Nano wpinamy je jak na zdjęciu (zdj płytka stykowa-Arduino Nano) teraz możemy bez problemu wpiąć inne elementy (zdj serwa i np naszego sygnalizatora) Gdy kupimy Arduino uno nie potrzebujemy nawet płytki stykowej ponieważ jest ono tak skonstruowane że możemy od razu wpinać urządzenie w płytkę (znowu zdjęcia ale z uno) To jest tak jakby wersja testowa. Na płytce stykowej łączymy wszytko kabelkami. Załącznik lista zakupów: lv1, lv2, lv3

3. Pierwszy mikrokontroler

Jest to dość podstawowy element: co trzeba mieć aby zacząć przygodę z elektroniką. Odpowiedz: praktycznie nic. W kolejnych częściach spróbujemy ustalić poziom zaawansowania w czytelnika w elektronice oraz w programowaniu.

3.1. Od czego zacząć

Wielu z nas nie ma nawet miernika w domu, myślę że to jest dobry moment na zakupienie takiego urządzenia. Będziemy wyglądać bardziej profesjonalnie. Poza tak oczywistym zakupem musimy mieć jeszcze kilka rzeczy o których dowiecie się w kolejnych częściach.

3.2. budowa

Przy pomocy Arduino jesteśmy w stanie tworzyć bardzo skomplikowane rzeczy ale ono nie jest do tego przystosowane. Głównym zastosowaniem jest tj inteligenty dom.

3.3. budowa

4. Zastosowanie elektroniki w życiu codziennym

5. Więcej

5.1. Jak wiedzieć więcej

Internet jest pełen informacji ale nie zawsze łatwo jest je znaleźć. W tym celu powstały różne fora, poradniki i blogi, które pomagają nam. Polecam szukać w języku angielskim (nasze problemy wpisywać po w Google po angielsku np. "How to include new library to Arduino"). Gdy to zrobimy najlepiej szukać odpowiedzi na forum Stackoverflow. Serwis ma ponad 10 milionów artykułów, dzięki czemu jest duża szansa, że ktoś kiedyś miał taki sam problem jak wy. Stackoverflow można nazwać Wikipedią dla informatyków.

5.2. System kontroli wersji

System kontroli wersji to oprogramowanie, które pomaga śledzić zmiany w kodzie źródłowym. Dzięki temu mamy dostęp do kodu który kiedyś napisaliśmy (i został zmieniony/usunięty). Możemy również całkowicie zmieniać kod bez obawy, że przestanie on działać (w każdym momencie możemy przywrócić poprzednią wersję). Jest stosowany w dużych korporacjach, ponieważ ułatwia wspólną pracę. Jest kilka systemów kontroli wersji na rynku ale najbardziej popularnym (i polecanym przez nas) jest GitHub. Jako przykład możemy dodać, że został on wykorzystany do pisania tej pracy. W załącznikach na końcu będzie link do serwisu i do naszych repozytoriów (nie ważne, że nie macie pojęcia co to oznacza. Musicie po prostu wejść). [3, 4]

5.3.

Załączniki

1. https://github.com/Szarp/makieta_skrzyzowania
2. https://www.arduino.cc/en/Main/Software (ang.)
3. https://www.arduino.cc/en/Reference/AttachInterrupt (ang.)
4. https://www.arduino.cc/en/Reference/Serial (ang.)
5. https://www.arduino.cc/en/Reference/HomePage (ang.)

Bibliografia

[1] *https://www.arduino.cc/en/Guide/Introduction.*

[2] *https://www.arduino.cc/en/Reference/HomePage.*

[3] *https://pl.wikipedia.org/wiki/System_kontroli_wersji.*

[4] *https://pl.wikipedia.org/wiki/Git_(oprogramowanie).*