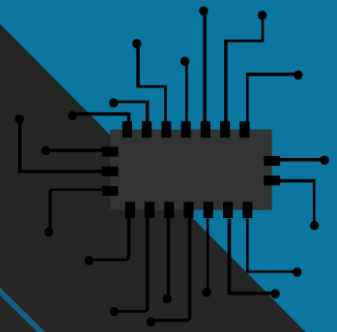
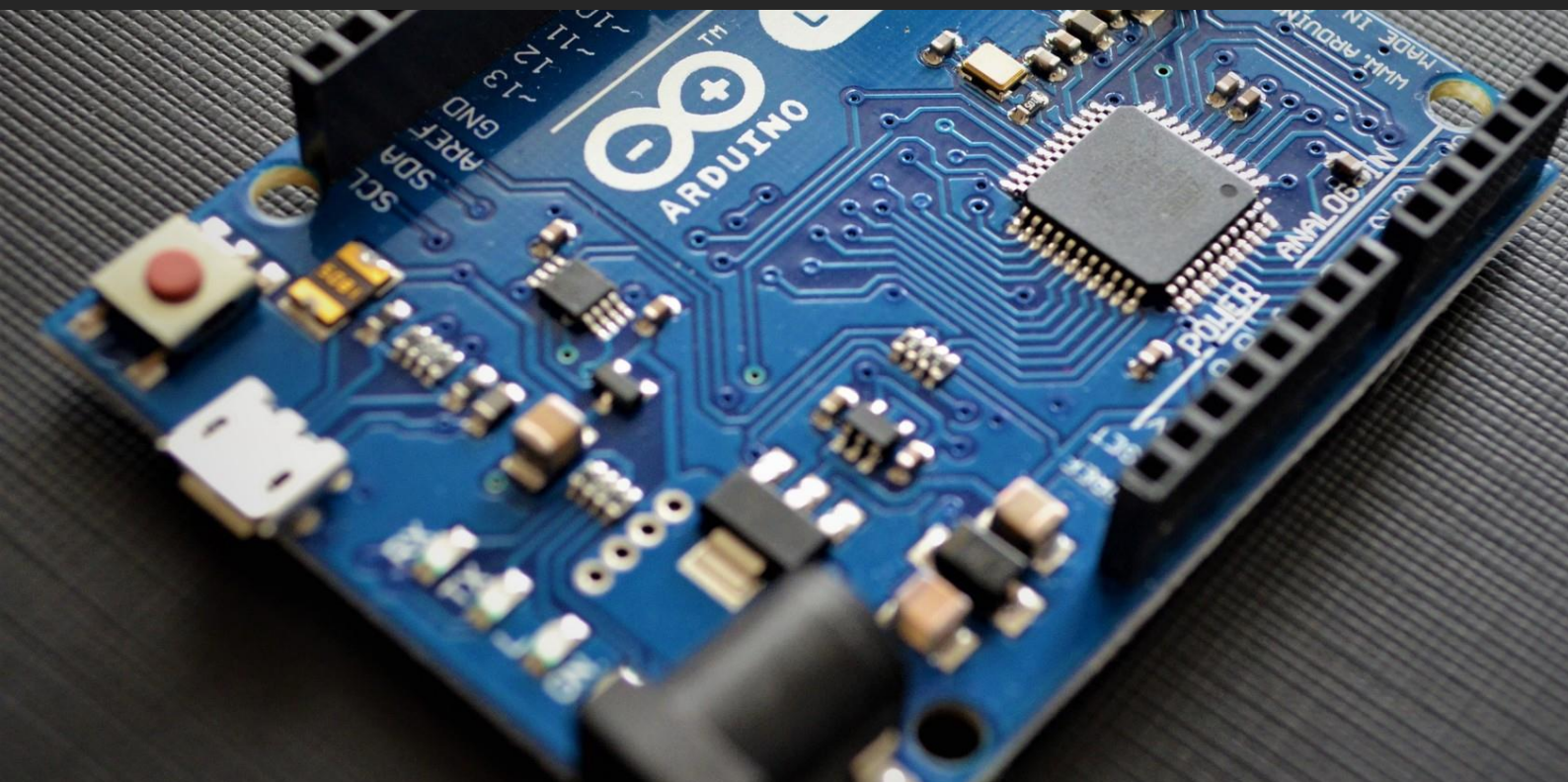


Mateusz Wiliński

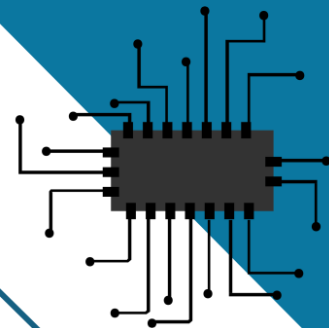


MIKROKONTROLERY PROGRAMOWANIE ARDUINO



-- 2026 10

Mateusz Wiliński



MIKROKONTROLERY PROGRAMOWANIE ARDUINO

Ebook Programowanie Arduino wraz z kursem wideo to drogowskaz, dzięki któremu nauczysz się programować mikrokontrolery w środowisku Arduino oraz poznasz zasadę ich działania.

KONTAKT

Mateusz Wiliński

Email: wilinski.mat@gmail.com

www: redwil.pl

Spis treści.....	3
Arduino Uno z ATmega328P	6
Programowanie.....	8
Symulatory Arduino.....	10
Programowanie Arduino w VS Code.....	11
Zmienne.....	12
Stałe	15
Typy danych w Arduino.....	17
Tablica ASCII	22
Funkcje	23
Instrukcja warunkowa.....	29
Enum.....	32
Switch	33
Pętla for	35
Pętla while	37
Tablica.....	38
Struct.....	40
Class.....	42
Pamięć SRAM	53
Wskaźniki.....	54
Pseudo-losowa liczba	57
Makra	58
Serial monitor	59
millis.....	60
Funkcje matematyczne	63
Systemy liczbowe	64
Operacje binarne.....	67
Modyfikacja bitów	70
Zarządzanie pinami Arduino	72
Przerwania.....	76
Schemat blokowy ATmega328P.....	77
Rejestry	79
Maski bitowe.....	80

Timery w Arduino.....	83
Przycisk chwilowy.....	87
Dioda LED RGB.....	90
Programowalne diody NeoPixel	91
Klawiatura membranowa 4x4.....	93
Joystick analogowy z przyciskiem	94
Brzęczyk piezoelektryczny.....	95
Czujnik PIR.....	98
Przesył danych w Arduino.....	99
Komunikacja USART (Serial Port).....	100
Komunikacja I2C.....	108
Komunikacja SPI.....	115
Wgrywanie programu za pomocą programatora ISP USBasp	123
Wgrywanie programu za pomocą innej płytki Arduino ISP	124
Komunikacja radiowa SPI NRF24L01.....	125
Komunikacja IR.....	135
Czytnik kart SD SPI.....	140
Zegar RTC DS3231.....	145
JSON	152
Wbudowana pamięć EEPROM.....	155
Zewnętrzna pamięć Eeprom FM24C256.....	156
Decade Counter CD4017.....	160
Shift Register 74HC595	161
Tworzenie biblioteki do obsługi 74HC595.....	166
Expander PCF8574	170
Wyświetlacz LCD 16x2.....	174
Wyświetlacz 7 segmentowy	179
Wyświetlacz 7 segmentowy 4-cyfrowy TM1637	183
Wyświetlacz 7 segmentowy 4-cyfrowy I2C HT16K33	187
SSD1306 OLED Display I2C	190
TFT SPI LCD 128*160 MSP1803	193
TFT SPI z PBC ILI9341 Touchscreen 320*240	195
Encoder	201
Sensor ultrasoniczny HC-SR04	206

Silniki DC.....	210
L293D H Bridge Motor Driver.....	212
Servo.....	214
Arduino z python	215

Arduino to platforma, która umożliwia programowanie mikrokontrolerów na płytkach Arduino, ESP8266, ESP32, STM32, itp. W każdej rodzinie płytek występują odmiany różniące się mikrokontrolerem, wielkością, ilością i rodzajem dodatkowych złączy.

W tej publikacji skupiamy się głównie na płytce Arduino Uno R3 / Nano wyposażone w 8 bitowy mikrokontroler ATmega328P. Znajomość Arduino sprowadza się do poznania zasad działania mikrokontrolera, w tym przypadku ATmega328P.

Ogólne zasady działania prostego mikrokontrolera MCU pozwoli łatwiej zrozumieć trudniejsze zagadnienia w znacznie bardziej rozbudowanych rozwiązaniach.

Mikrokontroler MCU ATmega328P to 8 bitowy mikrokomputer zawierający:

jednostkę centralną CPU,

pamięć RAM, FLASH, EEPROM

timery,

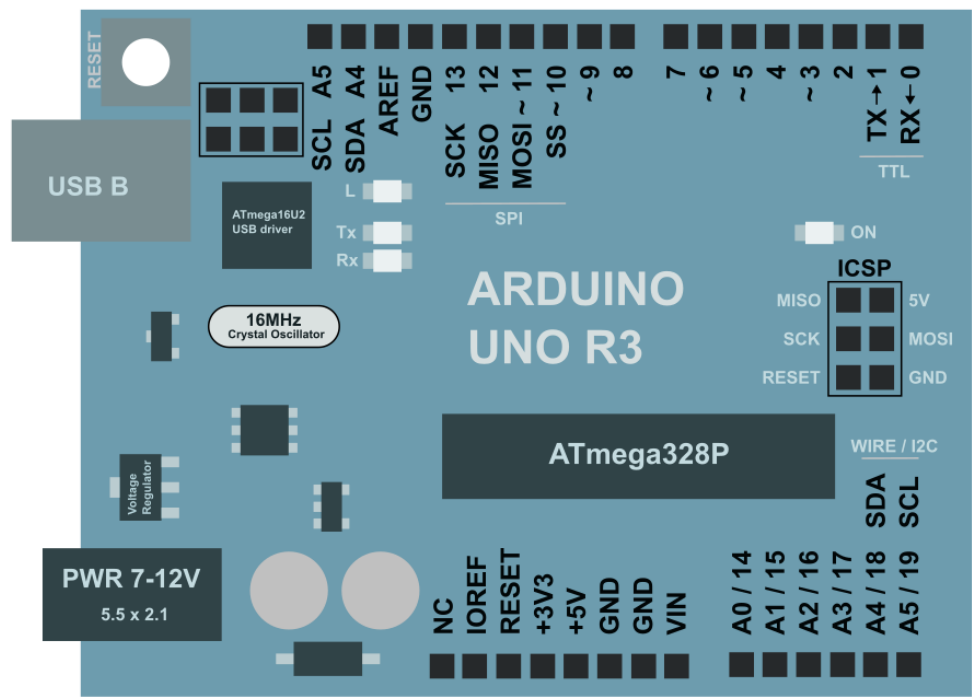
przetworniki analogowo-cyfrowe,

złącza wejścia/wyjścia (I/O)



Mikrokontroler:	ATmega328
Prędkość taktowania	16MHz
Napięcie zasilania	1.8 – 5.5V
Wydajność prądowa pinów I/O [optymalna / maksymalna]	20mA / 40mA

Podstawowe dane Arduino Uno



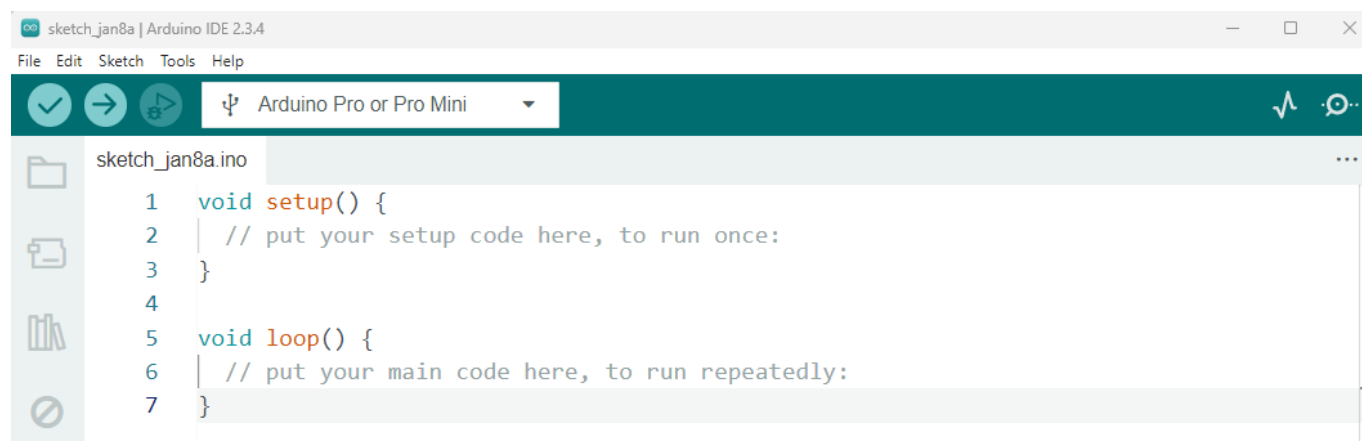
Digital Pins (0 – 13)(14 – 19)	Cyfrowe złącza wejściowe lub wyjściowe
Analog Pins (A0 - A5)	Piny analogowo-cyfrowe Złącza te działają domyślnie w trybie analogowym, odczytują napięcie za pomocą 10 bitowego przetwornika analogowo-cyfrowego (0 – 1023) Piny A0 – A5 mogą działać również jako cyfrowe wejścia lub wyjścia (14 - 19)
PWM pins (3, 5, 6, 9, 10, 11)	Wyjścia PWM o rozdzielczości 8 bit (0 – 255)
Interrupts (przerwania)	Pin2 - Int.0 Pin3 – Int.1

Pamięć:

FLASH: pamięć do zapisywania sketchu	32KB
SRAM: pamięć tymczasowa	2KB
EEPROM: pamięć informacji trwałych	1KB

Programowanie mikrokontrolerów na płytkach Arduino odbywa się za pomocą pochodnej języka C/C++ w oparciu o bibliotekę Arduino. Plik z kodem, zwany sketchedem, zazwyczaj tworzony jest w środowisku Arduino IDE i za jego pomocą wgrywany do mikrokontrolera.

[Link do programu Arduino IDE](#)



Rys. 1 Arduino IDE

Podpowiedzi kodu

Arduino IDE posiada podstawową funkcję podpowiedzi kodu.

Aktywacja tej opcji następuje poprzez zaznaczenie

Editor Quick Suggestions w File / Preferences.

Opcja podpowiedzi działa tylko wtedy, kiedy płytka jest podłączona do komputera.

Sketch Arduino wymaga dwóch metod: setup i loop.

setup

Metoda setup() wykonywana jest jeden raz, podczas włączenia płytki lub po jej resecie. Służy do jednorazowej inicjalizacji wartości początkowych.

loop

Pierwsze wywołanie metody loop() następuje tuż po zakończeniu metody setup. Loop powtarza się wiele razy w ciągu każdej sekundy.

Częstotliwość wywołań zależna jest od mocy mikroprocesora i złożoności programu.

Preferences

☒ Show files inside Sketches

Editor font size:

Interface scale:

Theme:

Language:

Show verbose output during

Compiler warnings

☐ Verify code after upload

☒ Auto save

☒ Editor Quick Suggestions

Additional boards manager UR

Link do sterownika dla płytek chińskich z chipem CH340: https://www.wch-ic.com/downloads/CH341SER_ZIP.html

Na początku przygody z programowaniem embedded, polecamy darmowe symulatory online, w których można tworzyć i programować układy.

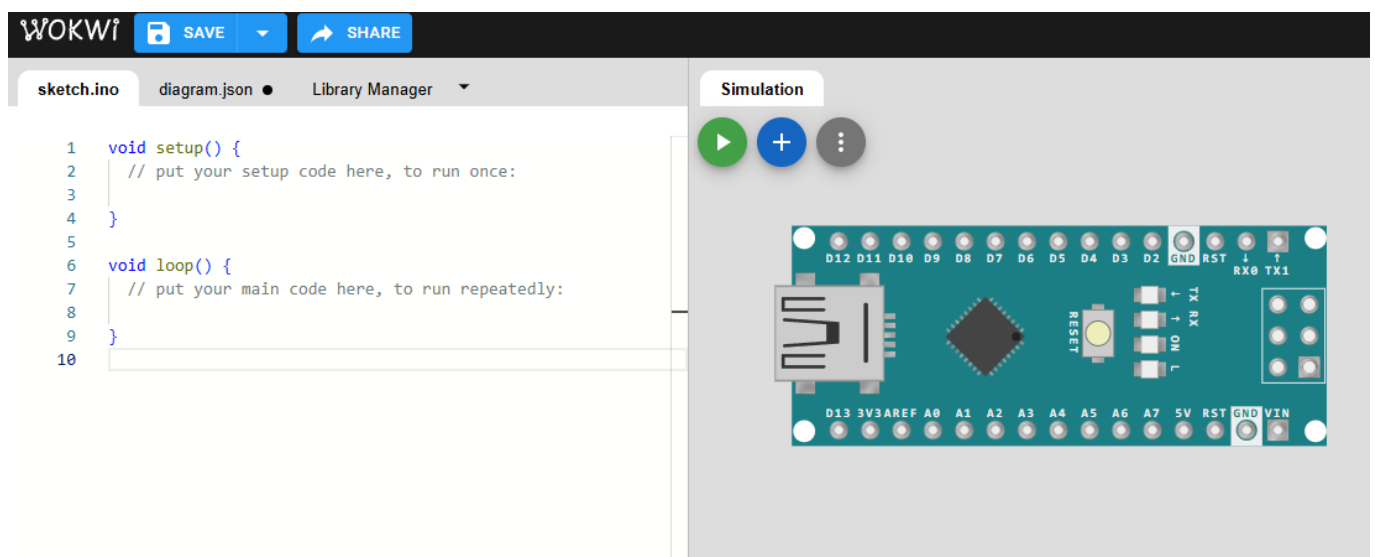
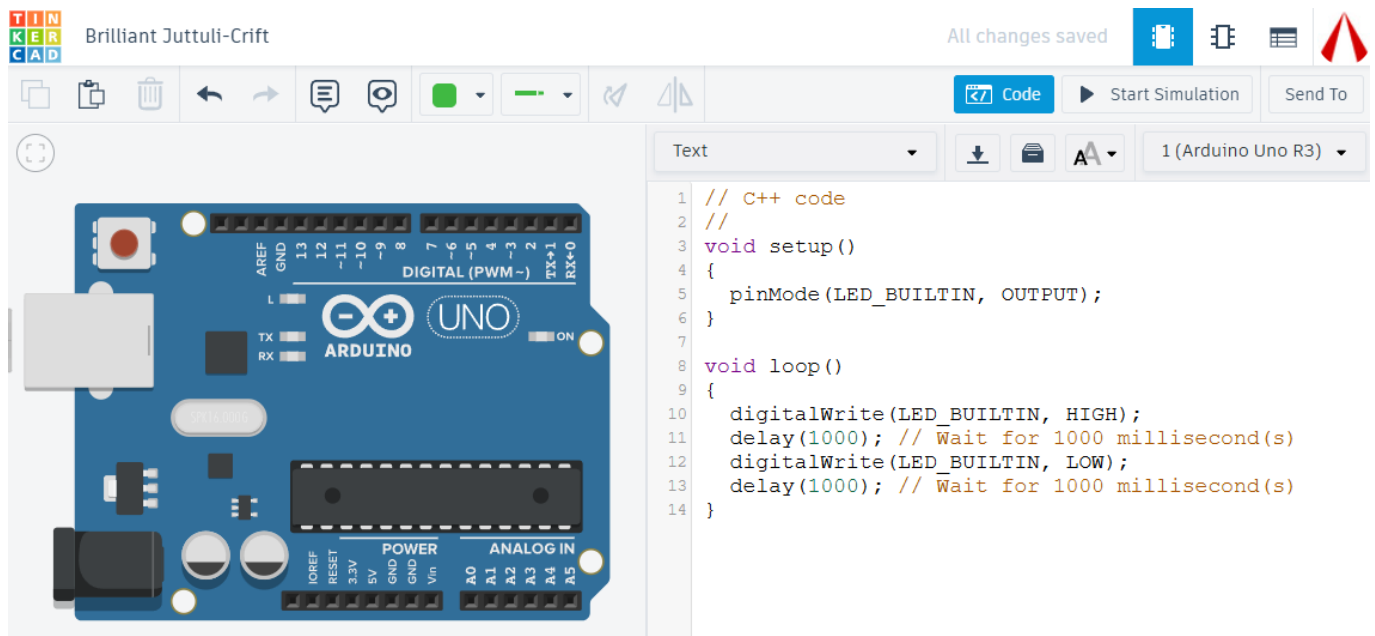
Lista polecanych symulatorów online:



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



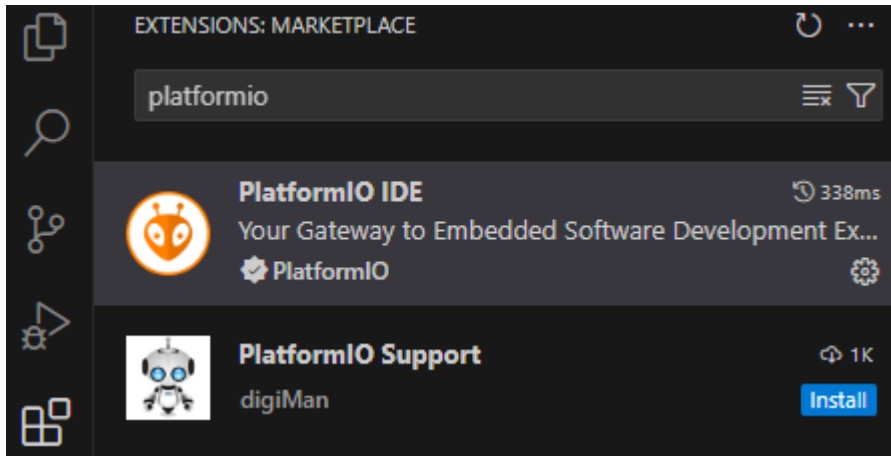
<https://wokwi.com/>



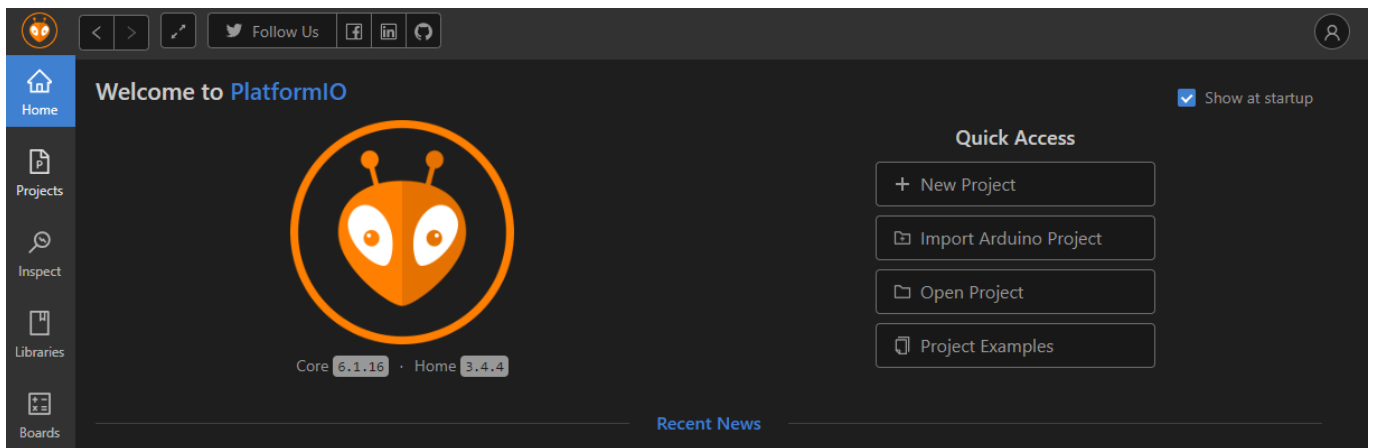
Płytki Arduino można programować na wiele sposobów, nie tylko w Arduino IDE.

Jedną z możliwości jest Visual Studio Code, do którego zostały napisane specjalne dodatki obsługujące Arduino. Jednym z nich jest dodatek **PlatformIO**.

Po zainstalowaniu VSC przejdź do zakładki Extensions, wpisz PlatformIO i zainstaluj.

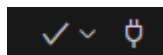


Po restarcie programu pojawi się dodatkowa ikona



Utwórz nowy projekt wybierając swoją płytkę Arduino.

Weryfikacja, wgrywanie programów i otwieranie Serial Monitora jest możliwe dzięki ikonom w górnej, prawej części ekranu:



Zmienne

Zmienna, to wirtualny kontener (obszar w pamięci podręcznej komputera), który pozwala przechowywać, odczytywać i zmieniać wartości danego typu w trakcie działania programu.

Zmienna składa się z:

1. typu danych - rodzaj informacji, np.: liczby całkowite, liczby ułamkowe, znaki
2. nazwy – unikalna etykieta odnosząca się do adresu w pamięci komputera.
Za pomocą nazwy odnosimy się do konkretnego miejsca w pamięci i pobieramy z niej zapisaną wartość lub tę wartość nadpisujemy nową
3. wartości - dane zapisane w określonym miejscu pamięci

Deklaracja zmiennej to podanie typu i nazwy, np:

```
int pinNumber;
```

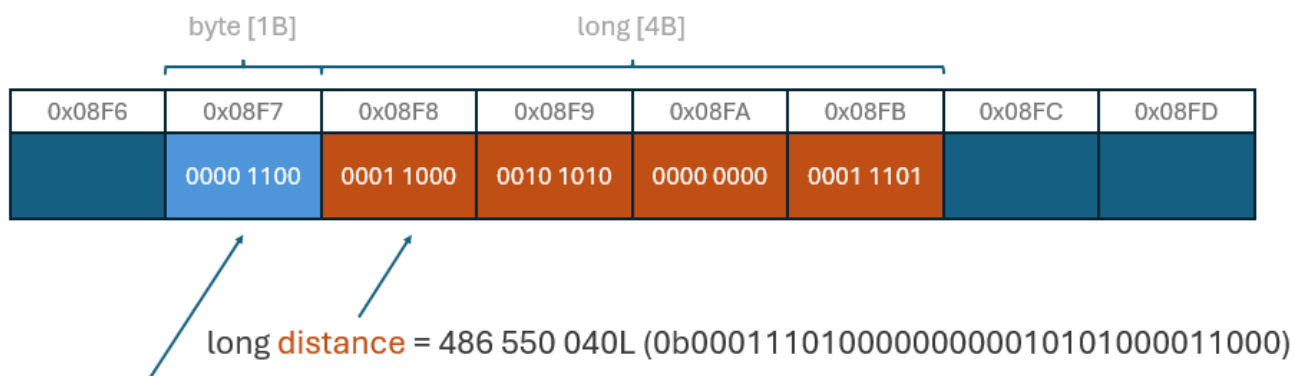
Do zadeklarowanej zmiennej w dowolnym momencie można przypisać wartość zgodną z podanym typem (bez tego zmienna będzie zainicjalizowana wartością domyślną, w przypadku int – 0).

W ten sposób, można nadpisywać wartości w zmiennych.

```
pinNumber = 9;
```

Tworzone zmienne można inicjalizować podczas tworzenia, np.:

```
int pinNumber = 6;
```



byte **houseNumber** = 12 (0b00001100)

Alokacja danych w systemie little-endian

(najmłodszy bajt zapisywany w pierwszej komórce pamięci)

Nazewnictwo zmiennych:

1. nazwy rozpoczynamy od małej litery, a kolejne słowo składowe zaczynamy z wielkiej litery (jest to tak zwana notacja camelCase, np: userID, newPassword);
2. nazwy muszą być opisowe (przykład złej nazwy to: zmienna2, cos, zm2);
3. należy stosować litery alfabetu angielskiego (bez polskich akcentów czy symboli);
4. można korzystać z liczb, ale nie można od nich zaczynać nazwy (np. player2, 2player)
5. istnieje kilka zarezerwowanych słów kluczowych (keywords), których nie można używać jako nazwy zmiennych (np. void, setup, return, delay i inne)
6. Język C/C++ jest CaseSensitive, co oznacza, że wielkość liter ma znaczenie.
Zmienne Number, number i NUMBER to trzy różne zmienne.

Zmienna statyczna

zawiera słowo kluczowe static, jest inicjalizowana tylko 1 raz podczas pierwszego wywołania funkcji. Używając zmiennej statycznej w metodzie loop, zostanie ona zainicjalizowana tylko podczas pierwszego wywołania.

```
void setup() {  
    Serial.begin(115200);  
}  
void loop() {  
    static int counter = 0; // Zmienna zachowuje wartość między wywołaniami loop()  
    Serial.print("Licznik: ");  
    Serial.println(counter);  
    counter++;  
    delay(1000);  
}
```

Zmienne lokalne

Zmienne tworzone wewnątrz bloku, czyli konstrukcji złożonej z nawiasów klamrowych stają się zmiennymi lokalnymi. Są dostępne w ramach tego bloku oraz bloków wewnętrznych.

Proste zmienne lokalne zapisywane są na stosie (stack) w RAM-ie tylko na czas wykonania funkcji i usuwane po zakończeniu funkcji, co oszczędza pamięć.

Zmienne statyczne lub obiekty tworzone za pomocą operatora `new`, nie są przechowywane na stosie.

Zmienne globalne

Zmienne tworzone w skrypcie głównym, stają się zmiennymi globalnymi.

Są dostępne w każdym miejscu kodu oraz istnieją przez cały czas działania programu.

Zmienne globalne są wygodniejsze w użyciu, ale mogą prowadzić do problemów wynikających z faktu, że można je nadpisywać z dowolnej funkcji, co może utrudniać debugowanie

Stałe

Stałe to zmienne, których wartości nie można nadpisywać. Wartość stałej musi zostać przypisana podczas deklaracji.

Nazwy stałych tworzy się z wielkich liter wg. zasady SNAKE_CASE.

```
const int SENSOR_PIN = 2; // Stała znana podczas kompilacji
const int ANALOG_PIN = A2;
const int SEED = analogRead(A0); // Stała nieznana podczas kompilacji (zostanie
przypisana dopiero podczas uruchomienia mikrokontrolera)
```

Zapis i odczyt stałej z pamięci Flash

Stałe, których wartości są znane podczas kompilacji mogą być zapisane w pamięci FLASH.

```
#include <avr/pgmspace.h>
const long CODE PROGMEM = 0x45CD00; // Stała przechowywana w pamięci Flash
void setup() {
    Serial.begin(115200);
    Serial.println("Odczyt z pamięci Flash:");
    long value = pgm_read_dword_near(&CODE); // Odczyt 32-bitowej wartości z Flash
    Serial.println(value, HEX);
}
void loop() {}
```

Zapis i odczyt tablicy z pamięci Flash

```
#include <avr/pgmspace.h>
// Tablica hexCodes przechowywana w pamięci Flash
const uint8_t hexCodes[] PROGMEM = {
    0xDE, 0xAD, 0xBE, 0xEF, 0x12, 0x34, 0x56, 0x78
};
void setup() {
    Serial.begin(115200);
    Serial.println("Odczyt z pamięci Flash:");
    for (int i = 0; i < sizeof(hexCodes); i++) {
        uint8_t value = pgm_read_byte_near(hexCodes + i); // Odczyt z pamięci Flash
        Serial.print(value, HEX);
        Serial.print(" ");
    }
    Serial.println();
}
void loop() {}
```

Makro

#define tworzy makro, które często jest stosowane jako stała.

```
#define LED_PIN 12 // pseudo stała (makro)
```

Podstawową wadą tego rozwiązania jest brak określenia typu wartości oraz problemy z debugowaniem.

Typ danych	Rozmiar	Właściwości
bool	1 byte	HIGH, 1, true lub LOW, 0, false <code>bool isReady = true;</code> Odwrócenie wartości typu bool: <code>isReady = !isReady;</code>
byte [uint8_t]	1 byte	Zakres: 0 do 255 <code>byte lightSeq = 0b00110101;</code> <code>uint8_t lightSeq = 0;</code>
int [int16_t]	2 byte	Zakres: -32768 do 32767 <code>int pinMode = 12;</code>
unsigned int [uint16_t]	2 byte	Zakres: 0 do 65535 <code>unsigned int counter = 0;</code> <code>uint16_t counter = 0;</code>
long [int32_t]	4 byte	Zakres: - 2147483648 do 2147483647 <code>long distance = 1234456L;</code>
unsigned long [uint32_t]	4 byte	Zakres: 0 do 4294967295 <code>unsigned long previousMillis = 0UL;</code>
float / double	4 byte	Zakres: -3.4028235E38 do +3.4028235E38 <code>float temp = 36.6;</code>
char	1 byte	Znaki z tablicy ASCII <code>char myChar = 'A';</code> <code>char myChar = 65;</code>
String		<code>String name = "Alex";</code> <code>char cname[5] = "Mark";</code> <code>char *cname = "Mark";</code> Zamiana wartości String na tablicę char: <code>name.toCharArray(cname, sizeof(cname));</code> <code>name.c_str();</code>

Boolean

W języku C++ typ boolean jest reprezentowany przez słowo kluczowe `bool`. Jest to typ logiczny, który służy do sterowania przepływem programu.

Może przyjmować tylko jedną z dwóch wartości:

true (prawda) 1 lub **false** (fałsz) 0

W Arduino, dla wartości równej 0 można stosować stałą `LOW`, a dla wartości 1 stałą `HIGH`.

Boolean może przyjmować wartość 0 lub 1, a przypisanie jakiejkolwiek liczby różnej od 0, przekonwertuje ją na liczbę 1.

Zmienna typu `bool` zajmuje 1B pamięci.

```
bool a = false; // 0
bool b = true;  // 1
bool c = 42;    // 42 zostanie zamieniona na 1
```

```
// Odwrócenie wartości typu bool
bool isReady = true;
isReady = !isReady; //isReady == false
```

Zmienna typu bool jako flaga

```
bool shouldBeExecuted = true;
void setup() {
    Serial.begin(115200);
}
void loop() {
    if (shouldBeExecuted) {          // Sprawdzenie, czy funkcja powinna być wykonana
        displayMsg();               // Wywołanie funkcji
        shouldBeExecuted = false;   // Ustawienie flagi, aby funkcja nie wykonała się
    }                               ponownie
    //... dalsza część kodu
}
void displayMsg() {
    Serial.println("Funkcja wykonana w loop, gdy isExecuted==true");
}
```

Liczby całkowite

Typy określające liczby całkowite mogą być:

signed (ze znakiem) - dodatnie i ujemne

unsigned (bez znaku) - tylko dodatnie

Domyślnie wszystkie liczby całkowite są traktowane jako typ: int

Dane liczbowe mogą być zapisywane w systemie dziesiętnym, binarnym lub szesnastkowym, np.:

```
uint16_t numberDEC = 45;
uint16_t numberBIN = 0b00101101;
uint16_t numberBIN2 = B00101101;
uint16_t numberHEX = 0x2D;
```

Zastosowanie typu `uint8_t` daje 5 - 28% wzrost wydajności w porównaniu do `uint16_t` czy `int`.

Kompilator rozpoznaje systemy liczbowe automatycznie. Wybór systemu liczbowego zależy od tego jaką liczbą najłatwiej się posłużyć w danym przypadku.

Liczby z różnych systemów można dowolnie łączyć:

```
int a = 3 + 0b0001 + 0xC;
Serial.println(a); //16
```

Liczby można wyświetlać w dowolnym systemie, np:

```
uint16_t number = 45;
void setup() {
    Serial.begin(115200);
    Serial.println(number);          // 45
    Serial.println(number, BIN);     // 101101
    Serial.println(number, HEX);     // 2D
}
void loop() {}
```

Liczby zmiennoprzecinkowe

Liczby ułamkowe mogą być zapisane w zmiennej typu: float lub double.

W przypadku mikrokontrolerów 8-bitowych, float i double są takie same, mają 32 bity. W mikrokontrolerach 32-bitowych, typ double ma 64 bity i podwójną precyzję.

Typy zmiennoprzecinkowe mogą zapisywać bardzo duże liczby, ale z ograniczoną precyzją. Wynika to z faktu, że nie każdą liczbę ułamkową można zapisać w postaci binarnej na określonej ilości bitów. To ograniczenie powoduje, że w pamięci nie zostanie zapisana podana liczba, ale najbliższa liczba, która może być zapisana w postaci 0 i 1.

Program przedstawiający różnicę zapisu liczb typu float i double

Chcąc wypisać większą ilość cyfr części dziesiętnej, dodaj drugi argument do funkcji print.

```
void setup() {  
  Serial.begin(115200);  
  float floatB = 12.12;  
  double doubleB = 12.12;  
  Serial.println(floatB, 10);    // 12.1199998856  
  Serial.println(doubleB, 10);   // 12.1200000000  
}  
void loop() {}
```

Znaki i wartości tekstowe

Znaki są zapisywane za pomocą typu char w postaci liczb, zgodnie z tablicą ASCII / Unicode. Znaki zapisuje się wewnątrz apostrofu.

```
char myChar = 'A';  
char myChar = 65;
```

Większą ilość znaków zapisuje się w postaci zmiennej typu String lub tablicy char. Ciągi znakowe w zmiennej typu String zapisuje się wewnątrz cudzysłowu.

```
String name = "Alex";  
char cname[5] = "Mark";  
char *cname = "Mark";
```

Zamiana znaków na liczby i na odwrót

```
void setup() {  
  Serial.begin(115200);  
  char a = 'a';  
  Serial.println((byte)a); // wyświetli 97  
  byte number = 64;  
  Serial.println((char)number); // wyświetli @  
}  
void loop() {}
```

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	NUL	44	2C	,	88	58	X
1	1	Start of heading SOH	45	2D	-	89	59	Y
2	2	Start of text STX	46	2E	.	90	5A	Z
3	3	End of text ETX	47	2F	/	91	5B	[
4	4	End of transmission EOT	48	30	0	92	5C	\
5	5	Enquiry ENQ	49	31	1	93	5D	]
6	6	Acknowledge ACK	50	32	2	94	5E	^
7	7	Bell BEL	51	33	3	95	5F	_
8	8	Backspace BS	52	34	4	96	60	`
9	9	Hor tab TAB	53	35	5	97	61	a
10	A	New line NL '\n'	54	36	6	98	62	b
11	B	Vert tab VT	55	37	7	99	63	c
12	C	Form feed FF	56	38	8	100	64	d
13	D	Carriage return CR '\r'	57	39	9	101	65	e
14	E	Shift out SO	58	3A	:	102	66	f
15	F	Shift in SI	59	3B	;	103	67	g
16	10	Data link escape DLE	60	3C	<	104	68	h
17	11	Device ctrl 1 DC1	61	3D	[=]	105	69	i
18	12	Device ctrl 2 DC2	62	3E	>	106	6A	j
19	13	Device ctrl 3 DC3	63	3F	?	107	6B	k
20	14	Device ctrl 4 DC4	64	40	@	108	6C	l
21	15	Negative ack NAK	65	41	A	109	6D	m
22	16	Synchronous idle SYN	66	42	B	110	6E	n
23	17	Eng of trans block ETB	67	43	C	111	6F	o
24	18	Cancel CAN	68	44	D	112	70	p
25	19	End of medium EM	69	45	E	113	71	q
26	1A	Subtitute SUB	70	46	F	114	72	r
27	1B	Escape ESC	71	47	G	115	73	s
28	1C	File separator FS	72	48	H	116	74	t
29	1D	Group separator GS	73	49	I	117	75	u
30	1E	Record separator RS	74	4A	J	118	76	v
31	1F	Unit separator US	75	4B	K	119	77	w
32	20	Space	76	4C	L	120	78	x
33	21	!	77	4D	M	121	79	y
34	22	"	78	4E	N	122	7A	z
35	23	#	79	4F	O	123	7B	{
36	24	\$	80	50	P	124	7C	
37	25	%	81	51	Q	125	7D	}
38	26	&	82	52	R	126	7E	~
39	27	'	83	53	S	127	7F	
40	28	(	84	54	T			
41	29	)	85	55	U			
42	2A	*	86	56	V			
43	2B	+	87	57	W			

Funkcja zawiera instrukcje, które mogą być wywoływane wielokrotnie w różnych miejscach programu.

Funkcja może przyjmować argumenty i zwracać wartość danego typu. Funkcja typu void nie zwraca żadnej wartości.

Każda funkcja ma określoną sygnaturę, na podstawie typów i ilości parametrów.

Funkcja musi zostać zadeklarowana i zdefiniowana przed wywołaniem.

Deklaracja określa sygnaturę funkcji, sygnalizuje jej istnienie (prototyp funkcji).

```
int multiplyBy2(int);  
int multiplyBy2(int num); //można również podać nazwę argumentu
```

Definicja zawiera deklarację wraz z blokiem kodu, w którym określone są instrukcje do wykonania. Definicja nie wymaga deklaracji, jeśli jest umieszczona w jednym pliku, powyżej miejsca wywołania.

```
int multiplyBy2(int num) {  
    int result = num * 2;  
    return result;  
};
```

Ponieważ kod jest czytany od góry w dół, deklaracje funkcji można dodać na górze skryptu, aby kompilator miał do niej dostęp, a definicję z ciałem funkcji na końcu pliku.

Arduino IDE samodzielnie generuje deklaracje funkcji przed kompilacją.

Funkcję wywołuje się podając jej nazwę z nawiasem okrągłym.

```
multiplyBy2(4); // zwróci wartość 8
```

Deklaracja i definicja funkcji wraz z wywołaniem

```
int multiplyBy2(int num); //deklaracja

void setup() {
  Serial.begin(115200);
  Serial.println(multiplyBy2(4)); //wywołanie
}

void loop() {}

int multiplyBy2(int num) { //definicja
  int result = num * 2;
  return result;
};
```

Funkcja z return

Funkcja, która zwraca wartość, np: int, float, char, int[], itp., musi zawierać instrukcję return.

```
int getRandomNumber() {
  randomSeed(analogRead(A0));
  return random(1, 101);
}
```

Wywołanie funkcji, która zwraca wartość zazwyczaj przypisuje się do zmiennej.

```
int rdn = getRandomNumber();
```

Często używa się instrukcji return do przerywania działania funkcji w danym miejscu po spełnieniu pewnych warunków.

Return kończy działanie funkcji, a kod umieszczony za return nie zostanie wykonany.

Funkcja z parametrami

Funkcja może posiadać parametry, dzięki czemu staje się bardziej uniwersalna.

```
int addNumbers(int num1, int num2){  
    return num1 + num2;  
}
```

Wywołując funkcję należy podać argumenty o określonym typie.

```
addNumbers(3,5);
```

Do funkcji można dodać parametry opcjonalne, które przyjmują domyślne wartości, jeśli nie zostaną określone podczas wywołania funkcji.

Parametry opcjonalne dodaje się jako ostatnie.

```
int multiply(int number1, int number2 = 1) {  
    return number1 * number2;  
}  
  
void setup() {  
    Serial.begin(9600);  
    Serial.println(multiply(3));    //zwróci 3x1  
    Serial.println(multiply(3, 2)); //zwróci 3x2  
}  
  
void loop() {}
```

Przeciążanie funkcji

Przeciążanie funkcji to tworzenie wielu funkcji o tej samej nazwie, ale różnych sygnaturach. Program rozpoznaje, którą z funkcji wywołać po typach wprowadzonych argumentów.

```
int addNumbers(int num1, int num2){
    return num1 + num2;
}
float addNumbers(float num1, float num2){
    return num1 + num2;
}
addNumbers(3, 5);
addNumbers(3.4f, 5.1f);
```

Zmienne przekazywane do funkcji przez referencję

Wartości zmiennych przekazywanych do funkcji stają się ich kopią. Modyfikacja tych wartości wewnątrz funkcji nie wpływa na przekazywane zmienne.

Zmienne można przekazać do funkcji przez referencję, dzięki czemu ich zmiana wewnątrz wpłynie na wartości zmiennych poza funkcją.

```
void swapNums(int &x, int &y) {
    int z = x;
    x = y;
    y = z;
}
void setup() {
    Serial.begin(115200);
    int firstNum = 10;
    int secondNum = 20;
    swapNums(firstNum, secondNum);
    Serial.println(firstNum);
    Serial.println(secondNum);
}
void loop() {}
```

Rekurencja

Rekurencja to wywołanie funkcji przez samą siebie.

Należy pamiętać, aby logika funkcji umożliwiała jej zakończenie.

```
// Funkcja obliczająca sumę kolejnych liczb naturalnych
int sum(int k) {
    if (k > 0) {
        return k + sum(k - 1);
    } else {
        return 0;
    }
}

void setup() {
    Serial.begin(9600);
    int result = sum(10);
    Serial.println(result);
}

void loop() {}
```

```
// Rekurencja do obliczania silni z danej liczby
int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}

void setup() {
    Serial.begin(9600);
    int number = 5;
    int result = factorial(number);
    Serial.print("Factorial of ");
    Serial.print(number);
    Serial.print(" is: ");
    Serial.println(result);
}

void loop() {}
```

Tworzenie funkcji w osobnych plikach .h i .cpp

Deklaracje funkcji dodaje się w plikach nagłówkowych .h, a definicje w plikach .cpp.

Wszystkie pliki muszą mieć odwołanie do pliku nagłówkowego .h

```
//plik nagłówkowy .h
#ifndef MATEMATYKA_H
#define MATEMATYKA_H
#include <Arduino.h>
extern const int BTN_PIN;
int addNumbers(int num1, int num2);
float addNumbers(float num1, float num2);
#endif
```

```
//plik .cpp
#include "Matematyka.h"

const int BTN_PIN = 2;
int addNumbers(int num1, int num2) {
    return num1 + num2;
}
float addNumbers(float num1, float num2) {
    return num1 + num2;
}
```

```
//plik projektowy .ino
#include "Matematyka.h"
void setup() {
    Serial.begin(115200);
    Serial.println(addNumbers(12, 6));
    Serial.println(BTN_PIN);
}
void loop() {}
```

Instrukcja warunkowa `if` służy do podejmowania decyzji w programie, pozwala na wykonanie różnych bloków kodu w zależności od spełnienia określonych warunków.

Warunki powinny przyjmować wartości typu `bool`.

```
if (warunek_jest_prawdziwy) {  
    // instrukcje wykonają się  
}  
  
if (warunek_nie_jest_prawdziwy) {  
    // instrukcje nie zostaną wykonane  
}
```

Operatory porównania

Warunki zazwyczaj, tworzy się je za pomocą operatorów porównania.

>	... jest większe od
>=	... jest większe lub równe niż
<	... jest mniejsze od
<=	... jest mniejsze lub równe niż
==	... jest równe
!=	... jest różne od

```
int tempSensor = 55;  
if (tempSensor > 50) {  
    // cooling system will turn on  
}
```

Operatory logiczne

Warunki można łączyć za pomocą operatorów logicznych.

&&	Operator AND zwraca true tylko wtedy, gdy oba warunki są prawdziwe.
	Operator OR zwraca true, jeśli przynajmniej jeden z warunków jest prawdziwy.
!	Operator NOT zwraca true, jeśli warunek jest fałszywy,

```
int tempSensor = 45;
bool isCoolingActive = false;
if (tempSensor > 50 && isCoolingActive == false) {
    // cooling system will turn on
}
```

```
int tempSensor1 = 45;
int tempSensor2 = 52;
if (tempSensor1 > 50 || tempSensor2 > 50) {
    // cooling system will turn on
}
```

Blok else

W instrukcji warunkowej można dołączyć blok else, który wykona się, jeśli żaden z bloków powyżej nie okaże się prawdziwy.

```
int tempSensor = 45;
if (tempSensor > 50) {
    // cooling system on
} else {
    // cooling system off
}
```

Blok else if

Instrukcja warunkowa może być stworzona z wielu bloków if else, które należy umieścić w odpowiedniej kolejności. W instrukcji warunkowej może być wykonany tylko jeden, pierwszy prawdziwy blok, reszta zostanie pominięta.

```
if (warunki) {  
    // instrukcje  
} else if (warunki) {  
    // instrukcje  
} else {  
    // instrukcje  
}
```

Przykład nieprawidłowo skonstruowanej instrukcji if

```
int temp = 40;  
  
if (temp > 12) {  
    //ten blok wykona się zawsze, gdy temp > 12  
}  
else if (temp > 34) {  
    //ten blok nie wykona się nigdy  
}  
else {  
    //wykona się jeśli żaden z powyższych nie będzie prawdziwy  
}
```

Enum to typ, który pozwala określić wartości liczbowe za pomocą nazw. Dzięki temu łatwiej i bardziej deskryptywnie można zarządzać kodem programu.

Elementy listy enum domyślnie przyjmują kolejne wartości naturalne rozpoczynające się od 0. Można to zmodyfikować przypisując do elementów konkretne wartości.

```
enum ledState{L_OFF, L_ON, L_BLINKING }; // 0, 1, 2
enum ledState{L_OFF = 2, L_ON, L_BLINKING }; // 2, 3, 4
enum ledState{L_OFF = 2, L_ON = 4, L_BLINKING }; // 2, 4, 5

ledState state = L_ON; // zmienna typu LED_STATE

void setup(){
  Serial.begin(115200);
  Serial.println(state); //1
}
void loop(){}
```

Traktowanie enum jako liczby i przełączanie stanów

```
enum ledState {L_OFF, L_ON, L_BLINK, L_BLINKFAST, L_LAST}; //dodanie L_LAST jako
elementu kontrolnego do zliczania ilości

int state = L_OFF;

void setup() {
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT);
  state = (state + 1) % L_LAST;
  Serial.println(state); //1
}
void loop() {}
```

Instrukcja switch działa podobnie do instrukcji warunkowej if. Za jej pomocą można sterować programem i dokonywać odpowiednich decyzji.

Instrukcja **break** przerywa działanie switch. Bez instrukcji break zostanie wykonany kolejny blok.

Instrukcja **default** działa podobnie jak else w instrukcji warunkowej, zostanie wywołany, gdy żaden z bloków powyżej nie zostanie obsłużony lub gdy nie zostanie wykonana instrukcja break.

W case'ach switcha nie można korzystać z wartości typu float.

Szablon instrukcji switch

```
int switchVar = 3;
switch (randNum) {
    case 3:
        // instrukcje
        break;
    case 4:
        // instrukcje
        break;
    default:
        // instrukcje
        break;
}
```

Przykład zarządzania instrukcjami na podstawie enum

```
enum ledState { L_OFF, L_ON, L_BLINK, L_LAST };
int state = L_ON;

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

void loop() {
    manageStates();
}

void manageStates() {
    switch (state) {
        case L_OFF:
            Serial.println("OFF");
            break;
        case L_ON:
            Serial.println("ON");
            break;
        case L_BLINK:
            Serial.println("BLINKING");
            break;
        default:
            state = L_OFF;
            break;
    }
}
```

Pętla służy do wielokrotnego wykonywania podobnych instrukcji.

W bloku for określa się ilość iteracji poprzez podanie wartości początkowej zmiennej iteracyjnej (zazwyczaj nazywanej i), końcowej oraz sposób jej zmiany.

Pętla będzie się wykonywać dopóki podany w środku warunek będzie prawdziwy.

W bloku pętli for określa się ilość iteracji poprzez podanie:

wartości początkowej zmiennej, np. `int i = 0;`

wartości warunku wykonania pętli, np.: `i <= 10;`

sposobu jej zmiany, np.: `i++;`

Pętla będzie się wykonywać dopóki podany w środku warunek będzie prawdziwy.

Szablon pętli for

```
for (int i = 0; i <= 9; i++) {  
    //instrukcje do wykonania w każdej iteracji pętli  
}
```

Pętlę for można zapisać również w inny sposób, np.:

```
int i = 0;  
for (; i <= 9;) {  
    //instrukcje do wykonania w każdej iteracji pętli  
    i++;  
}
```

Przykładowe programy z pętlą for

```
Serial.begin(9600);
```

```
// Program 1: Wyświetla liczby od 1 do 10
for (int i = 1; i <= 10; i++) {
    Serial.println(i);
}
```

```
// Program 2: Wyświetla liczby od 10 do 1
for (int i = 10; i >= 0; i--) {
    Serial.println(i);
}
```

```
// Program 3: Wyświetla liczby parzyste od 10 do 100
for (int i=10; i<=100; i+=2) {
    Serial.println(i);
}
```

```
// Program 4: Wyświetla kwadraty liczb od 1 do 10
for (int i=1; i<=10; i++) {
    Serial.print(i);
    Serial.print("^2 = ");
    Serial.println(i*i);
}
```

Pętla while może być stosowana zamiennie z pętlą for, ale głównie stosuje się ją w przypadku, gdy nie wiadomo ile razy powinna się ona wykonać. Np. podczas oczekiwania na połączenie z internetem lub pobierania danych.

Pętla będzie się wykonywać dopóki warunek będzie prawdziwy.

Szablon pętli while

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

```
//Program 1: Wyświetla liczby parzyste od 0 do 20
int i = 0;
while (i <= 20) {
    Serial.println(i);
    i+=2;
}
```

Tablica jednowymiarowa

Tablica może zawierać wiele danych tego samego typu. Każda wartość zapisana w tablicy ma swój numer porządkowy (indeks) dzięki czemu można na nich przeprowadzać wiele operacji za pomocą pętli. Indeksowanie tablicy zaczyna się od 0.

A	S	T	R	A	J	A
0	1	2	3	4	5	6

Długość tablicy (ilość elementów n) musi być określona podczas jej tworzenia.

Tablica kończy się niewidzialnym znakiem \0.

Sposoby tworzenia tablicy

```
// Sposób 1
// tablica tworzona na stosie
int numbers[] = { 12, 2, 9, 22 };
```

```
// Sposób 2
// Utworzenie tablicy z 4 elementami, które mają wartość domyślną 0
// Przypisanie wartości do elementów tablicy
int numbers[4];
numbers[0] = 12;
numbers[1] = 2;
numbers[2] = 9;
numbers[3] = 22;
```

```
// Sposób 3 (dla zaawansowanych)
// Tablica tworzona na stercie (heap)
int* numbers = new int[4];
```

```
// Zwolnienie zaalokowanej pamięci
delete[] numbers;
```

Tablica stringów i char

```
char* imiona[] = { "Mark", "Anna", "John"};  
char znaki[] = {'@', '!', 'z', 0}; //na koniec należy dodać 0
```

Nadpisywanie lub pobieranie wartości z tablicy następuje poprzez odwołanie do tablicy i podanie numeru indeksu.

```
numbers[0] = 32;  
Serial.print(numbers[0]); //32
```

Obliczanie długości tablicy

Długość tablicy oblicza się dzieląc rozmiar całej tablicy przez rozmiar pojedynczego elementu

```
int len = sizeof(numbers)/sizeof(numbers[0]);
```

lub

```
char name[] = {"Leonard"};  
int dlugoscTabChar = strlen(name);
```

Przy tablicach tworzonych na stacku obliczanie długości tablicy w ten sposób nie zadziała, ponieważ nazwa tablicy jest wskaźnikiem do miejsca w pamięci.

Tablica dwuwymiarowa (matrix)

```
int array1[wiersze][kolumny];
```

Przykład tablicy dwuwymiarowej

```
int array1[2][3] = {{4, 5, 2},  
                    {9, 1, 0}};
```

array1[2][3]	Kolumna0	Kolumna1	Kolumna2
Wiersz0	array1 [0][0] = 4	array1 [0][1] = 5	array1 [0][2] = 2
Wiersz1	array1 [1][0] = 9	array1 [1][1] = 1	array1 [1][2] = 0

```
Serial.print(array1[1][0]); //9 ----Wiersz 1, Kolumna 0
```

Struct pozwala stworzyć nowy obiekt (typ danych), który pozwala przechować wiele danych o różnych typach. Elementy Structa są domyślnie publiczne.

Struct book z metodami

```
struct book {
private:
    String _title;
    double _price;
    int _publishYear;
public:
    book(String title, double price, int year) {
        _title = title;
        _price = price;
        _publishYear = year;
    }
    void updatePrice(double newPrice) {
        _price = newPrice;
    }
    void displayInfo() {
        Serial.println(_title);
        Serial.print("Price: ");
        Serial.print(_price);
        Serial.println("zł.");
        Serial.print("Published in: ");
        Serial.println(_publishYear);
    }
};

void setup() {
    Serial.begin(115200);
    book b1 = { "Arduino programming", 49.99, 2025 };
    b1.updatePrice(39.99);
    b1.displayInfo();
    book b2 = book("Arduino mastery", 199.99, 2026);
    b2.displayInfo();
}

void loop(){}
}
```

Tablica struktur

```
void setup() {
    Serial.begin(115200);
    book books[3] = {
        book("Arduino programming", 99.99, 2025),
        book("Arduino mastery", 199.99, 2026),
        book("Electronics", 59.99, 2022)
    };
    for (int i = 0; i < sizeof(books) / sizeof(books[0]); i++) {
        books[i].displayInfo();
    }
}
void loop() {}
```

Metody na zewnątrz struktur

```
// Metody na zewnątrz struct
struct person {
    String name;
    int age;
};
void displayPersonInfo(person a);
void setup() {
    Serial.begin(9600);
    person jan = { "Jan", 28 };
    person anna = { "Anna", 22 };
    anna.age = 21;
    displayPersonInfo(anna);
}
void displayPersonInfo(person a) {
    String txt = a.name;
    txt += " is ";
    txt += a.age;
    Serial.println(txt);
}
```

Klasa jest instrukcją tworzenia nowego obiektu. Tworzone instancje klasy mają wspólne cechy określone w tej klasie.

Zmienne tworzone wewnątrz klasy nazywamy polami (atributami), a funkcje metodami.

Modyfikatory dostępu

Klasa, jej atrybuty i metody mogą być ustawione z odpowiednim modyfikatorem dostępu:

- `public` - atrybuty i metody są dostępne z zewnątrz klasy
- `private` - atrybuty i metody są dostępne tylko z poziomu danej klasy
- `protected` - atrybuty i metody są dostępne tylko z poziomu danej klasy oraz klas podrzędnych

Domyślnie wszystkie elementy klasy są ustawione jako prywatne.

Dobłą praktyką jest ustawianie pól klasy jako prywatne, a dostęp do nich umożliwić za pomocą publicznych metod.

Obiekty klasy mają typ referencyjny.

Przykład Klasy Player

```
class Player {
public:
    int attack;
    int speed;
};

void setup() {
    Serial.begin(9600);
    Player player;
    player.attack = 5;
    Serial.print(player.attack);
}
```

Funkcja modyfikująca pola obiektu

```
class Player {
public:
    int attack;
    int speed;
};

void addPower(Player& player, int power);

void addPower(Player& player, int power) {
    player.attack += power;
}

void setup() {
    Serial.begin(9600);
    Player player;
    player.attack = 5;
    addPower(player, 2);
    Serial.print(player.attack);
}
```

Statyczna metoda klasy

Wewnątrz definicji klasy można tworzyć metody statyczne, które odnoszą się do wszystkich instancji klasy. Wewnątrz takiej metody nie można odnosić się do pól niestatycznych obiektu.

```
static void DisplayPlayersCount(){ }
```

```
// Wywołanie funkcji:
```

```
Player:: DisplayPlayersCount();
```

Konstruktor

```
class Player {  
public:  
    int attack;  
    int speed;  
  
    Player(){  
        attack = 0;  
        speed = 0;  
    }  
};
```

C++ wywołuje domyślny konstruktor jeśli nie zostanie podany. Jest możliwość zablokowania tworzenia obiektów z domyślnym konstruktorem w poniższy sposób:

```
Player() = delete;
```

Klasa Book

```
class Book {
private:
    String _title;
    double _price;
    int _publishYear;
    static int _bookCount;
public:
    Book(String t, double p, int y) {
        _title = t;
        _price = p;
        _publishYear = y;
        _bookCount--;
    }
    double getPrice() {
        return _price;
    }
    void updatePrice(double newPrice) {
        _price = newPrice;
    }
    void displayInfo() {
        Serial.println("==== Book Info =====");
        Serial.print("Title: ");
        Serial.println(_title);
        Serial.print("Price: ");
        Serial.print(_price);
        Serial.println(" zł.");
        Serial.print("Published in: ");
        Serial.println(_publishYear);
    }
    static void displayBooksCount() {
        Serial.print("Books count: ");
        Serial.println(_bookCount);
    }
};
```

```
int Book::_bookCount = 10;

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

    Book myBook("Arduino Mastery", 59.99, 2025);
    myBook.updatePrice(39.99);
    myBook.displayInfo();

    Book books[] = {
        Book("Arduino Mastery", 59.99, 2025),
        Book("Arduino Mastery 2", 49.99, 2026),
        Book("Arduino Mastery 3", 29.99, 2027)
    };

    for (int i = 0; i < sizeof(books) / sizeof(books[0]); i++) {
        Serial.println(books[i].getPrice());
    }
}

void loop() {
}
```

Rozdzielenie klasy Book na pliki .h i .cpp

```
// plik projektu
#include "Book.h"
int Book::_bookCount = 10;

void setup() {
    Serial.begin(115200);
    Book books[] = {
        Book("Arduino Mastery", 59.99, 2025),
        Book("Arduino Mastery 2", 49.99, 2026),
        Book("Arduino Mastery 3", 29.99, 2027)
    };

    for (int i = 0; i < sizeof(books) / sizeof(books[0]); i++) {
        books[i].displayInfo();
    }
}

void loop() {}
```

```
// plik nagłówekowy .h
#ifndef BOOK_H
#define BOOK_H
#include <arduino.h>

class Book {
private:
    String _title;
    double _price;
    int _publishYear;
    static int _bookCount;

public:
    Book(String t, double p, int y);
    double getPrice();
    void updatePrice(double newPrice);
    void displayInfo();
    static void displayBooksCount();
};

#endif
```

```

// plik .cpp
#include "Book.h"

String _title;
double _price;
int _publishYear;
static int _bookCount;

Book::Book(String t, double p, int y) {
    _title = t;
    _price = p;
    _publishYear = y;
    _bookCount--;
}

double Book::getPrice() {
    return _price;
}

void Book::updatePrice(double newPrice) {
    _price = newPrice;
}

void Book::displayInfo() {
    Serial.println("==== Book Info =====");
    Serial.print("Title: ");
    Serial.println(_title);
    Serial.print("Price: ");
    Serial.print(_price);
    Serial.println(" zł.");
    Serial.print("Published in: ");
    Serial.println(_publishYear);
}

static void Book::displayBooksCount() {
    Serial.print("Books count: ");
    Serial.println(_bookCount);
}

```

Dziedziczenie

Od klasy głównej (bazowej) można tworzyć klasy pochodne za pomocą znaku : [dwukropek]

Klasy pochodne otrzymują atrybuty klasy bazowej, a dodatkowo mogą posiadać własne, specyficzne tylko dla niej.

Przykład klasy bazowej Vehicle i klas pochodnych

```
class Vehicle {
public:
    int maxSpeed;
    Vehicle(int speed)
        : maxSpeed(speed) {}
    void drive() {
        Serial.print("Driving at speed: ");
        Serial.println(maxSpeed);
    }
};

class Car : public Vehicle {
public:
    Car(int speed)
        : Vehicle(speed) {}
};

class Truck : public Vehicle {
public:
    int maxWeight;
    Truck(int speed, int weight)
        : Vehicle(speed), maxWeight(weight) {}
    void load(int weight) {
        if (weight <= maxWeight) {
            Serial.println("Truck loaded successfully.");
        } else {
            Serial.println("Exceeded maximum weight limit!");
        }
    }
};
```

```
void setup() {  
  Serial.begin(115200);  
  Car myCar(180);  
  myCar.drive();  
  Truck myTruck(120, 5000);  
  myTruck.drive();  
  myTruck.load(4500);  
  myTruck.load(5500);  
}  
void loop() {}
```

Obiekty klas pochodnych mogą implementować własne instrukcje wewnątrz metod, określonych w klasie głównej.

```
class Vehicle {
public:
    virtual void drive() {
        Serial.println("The vehicle is moving");
    }
};

class Car : public Vehicle {
public:
    void drive() override {
        Serial.println("The car is driving on the road");
    }
};

class Motorcycle : public Vehicle {
public:
    void drive() override {
        Serial.println("The motorcycle is speeding down the highway");
    }
};

void setup() {
    Serial.begin(115200);
    Vehicle v;
    Car c;
    Motorcycle m;
    v.drive();
    c.drive();
    m.drive();
}

void loop() {}
```

Pamięć SRAM to pamięć operacyjna, która zmienia się dynamicznie w trakcie trwania programu. Arduino Uno posiada 2kB, czyli 2048 bajtów pamięci. Daje to 2048 adresów od 0 do 2047, które zazwyczaj są podawane w postaci szesnastkowej.

Pierwsze 256 bajtów (0 – 255) jest zarezerwowanych dla rejestrów i prawidłowego działania mikrokontrolera.

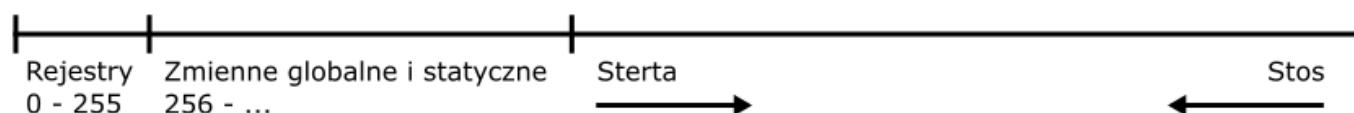
Reszta pamięci jest alokowana dynamicznie w zależności od potrzeb i specyfiki programu.

Zaczynając od adresu 0x0100 (256) rozpoczyna się obszar dla zmiennych globalnych i statycznych, przypisanie adresów następuje podczas kompilacji, a obszar ten zostanie niezmienny w trakcie wykonywania programu.

Po alokacji obszaru zmiennych globalnych i statycznych, reszta adresów pozostaje na alokację sterty i stosu.

Sztery pozwalają na alokację dynamicznych struktur danych, np. obiekty. Pierwsze adresy sterty rozpoczynają się tuż po ostatnim adresie obszaru zmiennych globalnych i statycznych i rosną w górę.

Na stosie alokowane są zmienne lokalne, parametry funkcji, adresy powrotu, podstawowe typy danych o stałej wielkości. Adresy stosu zaczynają się od końca i biegną w dół.



W bardzo rozbudowanych lub niepoprawnie napisanych programach dane zapisywane na sztercie i stosie, mogą się nałożyć, co stanie się źródłem problemów.

Sprawdzanie adresu zmiennej:

```
int zmienna = 10;
void setup() {
    Serial.begin(9600);
    Serial.println((uintptr_t)&zmienna, HEX);
}
void loop() {}
```

Wskaźniki

Każda zmienna / obiekt tworzony w czasie działania programu ma pewną wartość i zajmuje określoną ilość pamięci podręcznej. Wartość takiej zmiennej jest zapisywana w określonym miejscu pamięci, którego zazwyczaj nie musimy znać, ponieważ wartości te wywołujemy za pomocą nazwy zmiennej. Przekazywanie zmiennych do funkcji pozwala operować na jej kopii. Chcąc operować na konkretnej wartości (zapisanej w konkretnym miejscu pamięci), a nie kopii należy korzystać ze wskaźników (lub referencji).

Wskaźnik tworzy się podobnie jak zmienną dodając znak *.

Pobieranie adresu w pamięci

Adres pobiera się za pomocą znaku & określając (rzutując) jego typ jako uintptr_t, który przyjmuje pełną wartość adresu niezależnie od platformy (8bit, 32bit, 64bit)
Zmienne globalne i lokalne są przechowywane w innych obszarach pamięci

```
int a = 12;

void setup() {
    Serial.begin(115200);
    Serial.println((uintptr_t)&a, HEX); //0x100
}

void loop() {}
```

Ponieważ zmienna typu int zajmuje 2 bajty, kolejna zmienna zostanie zapisana w adresie zwiększonym o 2. Kolejność lokowania zmiennych jest ustalana przez kompilator.

```

int a = 12;
int b = 12;

void setup() {
  Serial.begin(115200);
  Serial.println((uintptr_t)&a, HEX); //0x102
  Serial.println((uintptr_t)&b, HEX); //0x100
}
void loop() {}

```

Tworzenie wskaźników do zmiennych

```

int a = 12;
int *ptr_a = &a;
void setup() {
  Serial.begin(115200);
  Serial.println((uintptr_t)ptr_a, HEX); //adres na który wskazuje wskaźnik np.
0x100
  Serial.println(*ptr_a); //wartość pod adresem wskaźnika, 12
  (*ptr_a)++; //inkrementacja wartości do której odnosi się wskaźnik (12)
  ptr_a++; //inkrementacja adresu wskaźnika o 1 pozycję (ilość bajtów
odpowiednia dla danego typu)
  *ptr_a++; //inkrementacja adresu wskaźnika o 1 pozycję (ilość bajtów
odpowiednia dla danego typu), a następnie zwrócenie wartości z nowego adresu
}
void loop() {}

```

Tworzenie wskaźników do tablic

```

int arr[] = {1, 2, 3, 4};
int *ptr = arr; // Wskaźnik wskazuje na pierwszy element tablicy
int *ptr = &arr[2]; //Wskazuje na element o indeksie 2
Serial.println(*ptr); // Pierwszy element tablicy (1)
Serial.println(*(ptr+1)); // Drugi element tablicy (2)

```

Przykłady przekazywania adresów / wskaźników do funkcji

```
// Za pomocą wskaźników *
void increment(int* v) {
    (*v)++;
}
void setup() {
    Serial.begin(9600);
    int a = 5;
    increment(&a);
    Serial.print(a);
}
void loop() {}
```

```
// Za pomocą referencji &
void increment( int& v){
    v++;
}
void setup(){
    Serial.begin(9600);
    int a = 5;
    increment(a);
    Serial.print(a);
}
void loop() {}
```

```
int increment( int v){
    return ++v;
}
void setup() {
    Serial.begin(115200);
    int a = 5;
    Serial.print(increment(a));
}
void loop() {}
```

W arduino generowanie losowych liczb jest skomplikowanym procesem. W prosty sposób można generować pseudo-losowe liczby. Oznacza to, że sekwencja kolejnych liczb będzie jednakowa w zależności od wartości początkowej ustalonego ziarna. W celu zróżnicowania losowości sekwencji liczb, ziarno można ustalić na podstawie nieużywanego analogowego pinu wejściowego, który przyjmuje wartości od 0-1023.

```
randomSeed(analogRead(A0)); // Deklaracja ziarna
```

Przypisanie pseudo-losowej wartości do zmiennej:

```
byte randNum = random(255); // [0 - 254]  
byte randNum = random(10, 100); // [10 - 99]
```

Makro to instrukcja preprocesora, która może np. zmieniać wyrażenia na inne, przypisywać proste funkcje do stałych i inne.

Makra tworzy się za pomocą `#define`

Przykłady makra:

```
#define LED_PIN 13 // makro, które reprezentuje pin nr 13
#define struct class // zamienia struct na class
#define SQUARE(x) ((x) * (x)) // Makro do obliczania kwadratu liczby
#define ARRAYLEN(x) (sizeof(x)/sizeof(x[0])) // Makro do obliczania długości tablicy
#define TOGGLE_LED(x) digitalWrite(x, !digitalRead(x)) //przełącz stan pinu x
```

Użycie makra:

```
int wynik = SQUARE(liczba);
int tabLength = ARRAYLEN(numbers);
```

Makro z warunkiem

```
#define DEBUG_MODE 1 // Zmienna kontrolująca tryb debugowania

void setup() {
    Serial.begin(9600);
    #if DEBUG_MODE
        Serial.println("Tryb debugowania włączony");
    #else
        Serial.println("Tryb produkcyjny");
    #endif
}

void loop() {}
```

Makra można używać w miejsce stałych czy prostych funkcji, co z reguły przyspiesza działanie programu. Wiąże się to jednak z kilkoma wadami, m.in.

Brak kontroli nad typami danych

Utrudnione debuggowanie i śledzenie błędów

Serial monitor to Monitor Portu Szeregowego, który pozwala na komunikację mikrokontrolera z komputerem za pomocą magistrali USART. Więcej w rozdziale Komunikacja USART.

Inicjalizacja komunikacji:

```
Serial.begin(9600);
```

Wysyłanie danych

Metoda print / println wysyła dane skonwertowane na znaki (ASCII)

```
Serial.print(12);  
Serial.println("Hej"); //println dodaje znak przejścia do nowej linii
```

```
Serial.write(65);
```

Metoda write wysyła dane liczbowe bez konwersji na znaki ASCII

Odbieranie danych

Wysłane dane przechowywane są w buforze, które można pobrać za pomocą wielu dostępnych metod:

Metoda read() pobiera znak ASCII (lub jego wartość liczbową).

Metoda parseInt() pobiera z bufora pierwszą napotkaną wartość liczbową.

Przed pobraniem danych warto sprawdzić czy jakiekolwiek dane są dostępne w buforze za pomocą metody `available()`, która określa liczbę danych w bajtach.

```
// Przykład: Wyświetla wartość z bufora (np. wpisaną na klawiaturze)
void setup() {
    Serial.begin(9600);
}
void loop() {
    while(Serial.available() > 0) {
        byte recData = Serial.read(); // zwraca wartość liczbową znaku ASCII
        //char recChar = Serial.read(); // zwraca znak ASCII
        Serial.print(recData);
    }
}
```

millis

Funkcje czasowe tworzy się w oparciu o aktualny czas oraz zapisany czas wystąpienia ostatniego (lub kolejnego) wywołania.

W programie, który wymaga wielu funkcji wywoływanych w innych odstępach czasu, należy stworzyć osobne zmienne `countDelay` i `previousMillis` lub stworzyć osobną klasę.

```
// Przykład: Wywołuje funkcje cyklicznie z uwzględnieniem ostatniego wywołania
const int interval = 1000;
unsigned long previousMillis = 0;
int counter = 0;

void setup() {
    Serial.begin(9600);
}
void countWithDelay() {
    if (millis() - previousMillis >= interval) {
        previousMillis = millis();
        counter++;
        Serial.println(counter);
    }
}
void loop() {
    countWithDelay();
}
```

```
// Przykład: Funkcje cykliczne z uwzględnieniem kolejnego wywołania
const int interval = 1000;
unsigned long nextCountInterval;
int counter = 0;

void setup() {
    nextCountInterval = interval;
    Serial.begin(9600);
}
void countWithDelay() {
    if (millis() >= nextCountInterval) {
        nextCountInterval += interval;
        counter++;
        Serial.println(counter);
    }
}
void loop() {
    countWithDelay();
}
```

Utworzenie dedykowanej klasy do obsługi cyklicznych wywołań o różnym interwale.

```
class Astratimer {
private:
    int _interval;
    unsigned long _previousMillis;
    int _counter;

public:
    Astratimer(int interval) {
        _interval = interval;
        _previousMillis = 0;
        _counter = 0;
    }
    bool canExecute() {
        if (millis() - _previousMillis >= _interval) {
            _previousMillis = millis();
            return true;
        } else {
            return false;
        }
    };
};
```

```
// Zastosowanie klasy Astratimer w programie
Astratimer tim1(1000);
Astratimer tim2(3000);

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

void loop() {
    if (tim1.canExecute()) {
        Serial.println(millis());
    }
    if (tim2.canExecute()) {
        Serial.println("Hej");
    }
}
```

Nazwa funkcji	Opis działania	Przykład
abs (x)	Zwraca wartość bezwzględną	abs(-5) -> 5
constrain(x, min, max)	Ogranicza liczbę x do przedziału	constrain(x, 1, 10)
map (x, min1, max1, min2, max2)	Mapuje liczbę x z jednego zakresu na drugi	map(x, 0, 255, 0, 100)
max (x, y)	Zwraca większy z argumentów	max(3, 12) -> 12
min (x, y)	Zwraca mniejszy z argumentów	min(3, 12) -> 3
pow (x, y)	Potęgowanie x do y	pow(3, 2) -> 3 ²
sqrt (x)	Pierwiastek z liczby x	sqrt(16) -> 4
sin, cos (x)	Funkcje trygonometryczne x w radianach Stopnie z -> radiany x (z * PI / 180)	
log(x)	Logarytm naturalny z x	
log10(x)	Logarytm przy podstawie 10 z x	
exp(x)	Funkcja wykładnicza e ^x (e = 2.71828)	
a % b	Modulo (reszta z dzielenia)	5%2 -> 1
PI	Stała liczby Pi = 3.1416	
E	Stała Eulera E = 2.7183	
DEG_TO_RAD	Konwersja stopni(z) na radiany(x)	PI / 180 x = z * DEG_TO_RAD
RAD_TO_DEG	Konwersja radianów(x) na stopnie(z)	180 / PI z = x * RAD_TO_DEG

System dziesiętny

Na co dzień posługujemy się liczbami tworzonymi w systemie dziesiętnym (DEC, Decimal), za pomocą 10 cyfr od 0 - 9.

Jest to system pozycyjny, który posiada rzędy jedności, dziesiątek, setek, tysięcy, itd.... Każdy rząd ma odpowiednią wagę, którą oblicza się za pomocą potęgi liczby 10. Liczba 10 jest podstawą systemu, ponieważ używa się w nim dziesięciu cyfr.

Wykładnik potęgi wynosi 0 dla rzędu jedności i zwiększa się o 1 dla każdego kolejnego rzędu, idąc od prawej strony do lewej.

Rząd jedności jest najmniej znaczący.

Przykłady:

$$23_{10} = 2 \cdot 10^1 + 3 \cdot 10^0$$

$$145_{10} = 1 \cdot 10^2 + 4 \cdot 10^1 + 5 \cdot 10^0$$

$$2034_{10} = 2 \cdot 10^3 + 0 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0$$

System binarny

System dwójkowy (BIN, binarny) tworzony jest z dwóch cyfr 0 i 1 (stąd nazwa). Jest to system pozycyjny, podobnie jak system dziesiętny, z tą różnicą, że wagę rzędu(bitu) oblicza się na podstawie potęgi liczby 2.

$$11_2 = 1 \cdot 2^1 + 1 \cdot 2^0 = (\text{w systemie dziesiętnym } 2 + 1 = 3)$$

$$110_2 = 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = (\text{w systemie dziesiętnym } 4 + 2 + 0 = 6)$$

$$00101101_2 = 0 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 32 + 8 + 4 + 1 = 45_{10}$$

Bit to najmniejsza porcja informacji, która może przyjąć jeden z dwóch stanów, 0 lub 1. W arduino będziemy bardzo często operować na bitach, ale będziemy je grupować i zapisywać w innych jednostkach, np. Bajtach.

1 Bajt składa się z 8 bitów więc może określić 256 różnych konfiguracji - $2^8 = 256$

W programowaniu indeksowanie / liczenie zaczyna się od 0, dlatego za pomocą 1 Bajta można zapisać liczby od 0 do 255.

Liczby binarne można podawać w postaci pełnego bajta 0b/B, np:
0b00101101 lub B00101101

lub kilku bajtów, np. 0011001101010101

1 byte to najmniejsza jednostka informacji, która może być zapisana w pamięci. Chcąc zapisać tylko 1bit w pamięci, system i tak zarezerwuje pełny 1 byte.

System szesnastkowy

System szesnastkowy(HEX, hexadecimal) - system, którego podstawa wynosi 16.

Składa się z cyfr od 0 do 9 i liter od A do F, co daje w sumie 16 znaków:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A(10), B(11), C(12), D(13), E(14), F(15)

$$1A_{16} = 1 \cdot 16^1 + 10 \cdot 16^0 = 16 + 10 = 26_{10}$$

$$B5D_{16} = 11 \cdot 16^2 + 5 \cdot 16^1 + 13 \cdot 16^0 = 2816 + 80 + 13 = 2909_{10}$$

$$FF_{16} = 15 \cdot 16^1 + 15 \cdot 16^0 = 240 + 15 = 255_{10}$$

Liczby szesnastkowe podaje się za pomocą zapisu 0x, np: 0xA6C

System szesnastkowy jest często używany do adresowania, np. 0x32 oraz określania kolorów RGB z użyciem symbolu hash, np. #FF93A2

Zapis #FF93A2 definiuje kolor na podstawie kanałów RGB:

$$R: FF_{16} = 255_{10}$$

$$G: 93_{16} = 147_{10}$$

$$B: A2_{16} = 162_{10}$$

Dane liczbowe można przedstawić za pomocą różnych systemów. W Arduino najczęściej korzysta się z systemu dwójkowego BIN, dziesiętnego DEC i szesnastkowego HEX. Reprezentacja liczby 45 w 3 systemach liczbowych:

```
byte num = 0b00101101; //BIN
```

```
byte num = 45; //DEC
```

```
byte num = 0x2D; //HEX
```

Zamiana liczb dziesiętnych na binarne i szesnastkowe

Zamiana liczby dziesiętnej na liczbę w innym systemie polega na jej dzieleniu przez podstawę docelowego systemu (np. 2 lub 16).

Z powstałych reszt dzielenia, tworzy się nową liczbę od dołu w górę.

Zamiana wartości dziesiętnych na binarne:

$$11_{10} = 1011_2 \text{ (0b1011)}$$

$$127_{10} = 0111\ 1111_2 \text{ (0b01111111)}$$

2	reszta z dzielenia	
11	1	
5	1	
2	0	
1	1	

2	reszta z dzielenia	
127	1	
63	1	
31	1	
15	1	
7	1	
3	1	
1	1	

Zamiana wartości dziesiętnych na szesnastkowe

$$139_{10} = 8B_{16} \text{ (0x8B)}$$

$$2306_{10} = 902_{16} \text{ (0x902)}$$

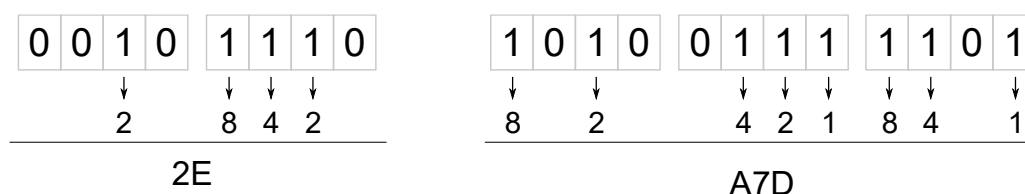
16	reszta z dzielenia	
139	B	
8	8	

16	reszta z dzielenia	
2306	2	
144	0	
9	9	

Zamiana wartości binarnych na szesnastkowe

Zamiana liczb binarnych na szesnastkowe jest bardzo prosta, ponieważ każde 4 bity reprezentują zakres od 0 do 15, czyli dokładnie jeden znak liczby szesnastkowej.

W odwrotny sposób można zamieniać liczby szesnastkowe na binarne.



Operatory binarne działają na podstawowej, najmniejszej jednostce informacji cyfrowej, czyli pojedynczym bicie.

Tabela prawdy dla operacji bitowych

a	b	AND a&b	OR a b	XOR a^b	NOT ~a	NAND ~(a&b)
0	0	0	0	0	1	1
0	1	0	1	1	1	1
1	0	0	1	1	0	1
1	1	1	1	0	0	0

Operator binarny AND &

Porównując dwa łańcuchy znaków binarnych o takiej samej długości, operator binarny AND zwraca prawdę (wartość 1) dla bitów, które w obu łańcuchach miały wartość 1.

```
byte val1 = 0b0101;
byte val2 = 0b0011;
byte result = val1 & val2; // wynik 0b1
```

	0	1	0	1
& AND	0	0	1	1
	0	0	0	1

Sprawdzanie czy liczba jest parzysta / nieparzysta

W liczbie parzystej bit nr 0 wynosi 0, a w nieparzystej 1.

Na liczbie wykonuje się operację bitową AND z liczbą 1. Jeśli wynikiem jest 1 to liczba jest nieparzysta, a jeśli jest 0 to jest parzysta.

Operator binarny OR |

Zwraca wartość 1 wtedy, gdy choć jeden z porównywanych bitów w łańcuchach binarnych ma wartość 1.

```
byte val1 = 0b0101;
byte val2 = 0b0011;
byte result = val1 | val2; // wynik 0b111
```

	0	1	0	1
 OR	0	0	1	1
	0	1	1	1

Operator binarny XOR ^

zwraca wartość 1 wtedy, gdy wartości porównywanych bitów w łańcuchach binarnych są różne. [^ kareta]

```
byte val1 = 0b0101;  
byte val2 = 0b0011;  
byte result = val1 ^ val2; // wynik 0b110
```

	0	1	0	1
XOR ^	0	0	1	1
	0	1	1	0

Operator binarny NOT~

Odwraca wartość pojedynczego łańcucha bitów. [~ tylda]

```
byte val1 = 0b0011;  
byte result = ~val1; // wynik 0b1100
```

NOT ~	0	0	1	1
	1	1	0	0

Operator binarny NAND ~& (Not AND)

```
byte val1 = B11001001;  
byte val2 = B01100011;  
byte result = ~(val1 & val2); // wynik 0b10111110
```

SHIFT << >>

Operator Shift (w uproszczeniu) służy do przesuwania bitów.

Operator Shift zachowuje się inaczej w przypadku liczb dodatnich i ujemnych. Tutaj będziemy się zajmować liczbami dodatnimi, dlatego uznajmy, że podczas przesuwania bitów, dokładane bity będą wypełniane zerami.

MSB (most significant bit) to bit reprezentujący największą wartość (z lewej strony),
LSB (least significant bit) to bit reprezentujący najmniejszą wartość (z prawej strony).

Przykłady przesunięć:

0	0	1	1
---	---	---	---

 << 1

0	1	1	0
---	---	---	---

0	0	1	1
---	---	---	---

 << 2

1	1	0	0
---	---	---	---

```
byte val1 = 0b00000011;  
Serial.println(val1 << 2,BIN); // wynik: 0b1100
```

1	0	0	1
---	---	---	---

 >> 1

0	1	0	0
---	---	---	---

1	0	0	1
---	---	---	---

 >> 3

0	0	0	1
---	---	---	---

```
byte val1 = 0b00001001;  
Serial.println(val1 >> 3,BIN); // wynik: 0b00000001
```

Wzór na wartość przesunięcia bitowego

$$n \ll k = n \times 2^k$$

$$5 \ll 1 \rightarrow 0101 \ll 1 \rightarrow 1010 = 10$$

$$5 \ll 2 \rightarrow 0101 \ll 2 \rightarrow 10100 = 20$$

```
void setup() {  
  Serial.begin(9600);  
  byte binary1 = 0b1011;  
  byte binary2 = 0b1011;  
  byte nowy = (binary1<<4)+binary2;  
  Serial.print(nowy,BIN);  
}
```

Modyfikacja bitów

bitSet(x, n)

Metoda bitSet() ustawia wartość 1 dla podanego numeru bitu n i zwraca nową wartość do zmiennej x.

x: zmienna w której ma nastąpić zmiana bitu

n: numer bitu do zmiany na 1

```
void setup() {
```

```
void setup() {  
  Serial.begin(9600);  
  byte x = 0b10000000;  
  Serial.print("BEFORE: ");  
  Serial.println(x, BIN);  
  bitSet(x, 2); // zamienia 3 bit od prawej strony na 1  
  Serial.print("AFTER: ");  
  Serial.println(x, BIN); // 10000100  
}
```

bitClear(x, n)

Metoda bitClear() ustawia wartość 0 dla podanego numeru bitu n i zwraca nową wartość do zmiennej x.

x: zmienna w której ma nastąpić zmiana bitu

n: numer bitu do zmiany na 0

bitRead(x, n)

Metoda bitRead() zwraca wartość danego bitu n z liczby x.

x: liczba (zmienna) z której dane są zaczytywane

n: numer bitu do zaczytania

```
byte x = 0b10001100;  
Serial.println(bitRead(x,2)); //zwraca wartość 3 bitu od prawej strony: 1
```

bitWrite(x, n, b)

Metoda bitWrite() zmienia wartość bitu n w liczbie x

x: liczba (zmienna) w której dane mają być zmienione

n: numer bitu do zaczytania

b: wartość 0 lub 1, która ma zostać nadpisana do podanego wcześniej bitu

```
byte x = 0b10000000;  
bitWrite(x, 0, 1);  
Serial.println(x, BIN); // 0b10000001
```

pinMode

Płytką Arduino posiada szereg pinów, każdy z nich może być pinem wyjściowym – OUTPUT lub wejściowym – INPUT.

Domyślnie piny ustawione są jako piny wejściowe - INPUT.

Za pomocą metody pinMode() można określić piny jako wyjściowe - OUTPUT lub aktywować wewnętrzny pullup rezystor dla trybu wejściowego - INPUT_PULLUP.

```
pinMode(13, OUTPUT); //Ustawienie pinu nr 13 na wyjściowy
```

digitalWrite

Metoda digitalWrite() ustawia stan pinu wyjściowego (1 / HIGH / true lub 0 / LOW / false)

```
digitalWrite(13, 1); //Stan wysoki 5V  
digitalWrite(13, 0); //Stan niski 0V
```

```
// Przykład: Ustawienie pinów wyjściowych 2-6 na stan wysoki  
for (int i = 2; i <= 6; i++) {  
    pinMode(i, OUTPUT);  
    digitalWrite(i, 1);  
}
```

```
// Przykład: Ustawienie pinów określonych w tablicy  
byte pins[] = { 3, 5, 6 };  
for (byte i = 0; i < sizeof(pins) / sizeof(pins[0]); i++) {  
    pinMode(pins[i], OUTPUT);  
    digitalWrite(pins[i], 1);  
}
```

```
// Przykład: Przełączanie LED za pomocą wartości 0 / 1 wpisanych z Serial Monitor
void setup() {
  pinMode(13, OUTPUT);
  Serial.begin(9600);
}
void loop() {
  if (Serial.available() > 0) {
    int recData = Serial.parseInt();
    bool state;
    if (recData == 1) state = HIGH;
    else if (recData == 0) state = LOW;
    else return;
    digitalWrite(13, state);
  }
}
```

digitalRead

Metoda digitalRead odczytuje wartości cyfrowe napięć wejściowych.

Interpretacja zakresów napięć na pinach cyfrowych w trybie INPUT:

0V – 1V stan niski LOW

3V - 5V stan wysoki HIGH.

Przedział 1V – 3V to stan zabroniony, w którym zmiana stanu logicznego nie następuje.

```
int sensorVal = digitalRead(2); //Wczytuje stan logiczny z pinu nr 2
```

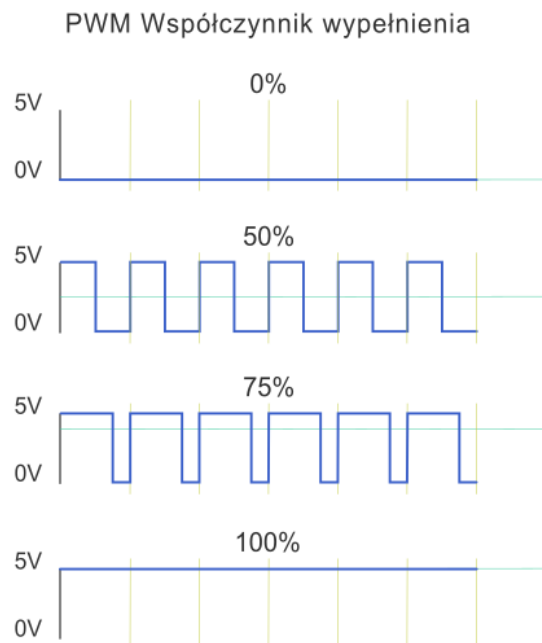
W zależności od potrzeb, odczytywanie danych z urządzeń analogowych(przycisk, potencjometr, itp.) zazwyczaj przeprowadza się z pomocą wbudowanego pullup rezystora lub zewnętrznego rezystora pullup lub pulldown, likwidując zjawisko szumów i fałszywych odczytów.

analogWrite

Piny cyfrowe 3, 5, 6, 9, 10, 11 mogą być wykorzystane do generowania przebiegu PWM (przebieg prostokątny o zmiennym współczynniku wypełnienia) z częstotliwością ok. 490Hz dla pinów 3, 9, 10, 11 i ok. 976Hz dla pinów 5 i 6.

Metoda `analogWrite()`, za pomocą 8-bitowego przetwornika generuje przebieg sygnału PWM w zakresie od 0 do 255.

```
analogWrite(11, 127); //generuje przebieg PWM  
127/255 na pinie 11
```



analogRead

Piny analogowe A0 – A5 przyjmują dane analogowe za pomocą 10-bitowego przetwornika analogowo-cyfrowego (ADC). Pozwala to na zamianę napięć wejściowych sygnału analogowego z przedziału 0V-5V na postać cyfrową wyrażaną liczbowo w zakresie od 0 do 1023.

```
int sensorVal = analogRead(A0);
```

Piny analogowe mogą pracować również jako cyfrowe piny wyjściowe/wejściowe przypisując im numery 14 – 19.

```
// Przykład: Wyświetlanie wartości potencjometru w procentach  
void setup() {  
    Serial.begin(9600);  
}  
void loop() {  
    int sensorValue = analogRead(A0);  
    int percentage = map(sensorValue, 0, 1023, 0, 100); // %  
    Serial.println(percentage);  
    delay(10);  
}
```

delay / delayMicroseconds

Metoda delay() wstrzymuje działanie programu na określony czas, podany w milisekundach, delayMicroseconds w mikrosekundach.

Ponieważ delay wstrzymuje większość procesów mikrokontrolera, jej stosowanie jest bardzo ograniczone. W wielu przypadkach, w miejsce delay, stosuje się rozwiązania oparte na metodzie millis().

```
delay(1000); //Wstrzymanie programu na 1000ms (1s)  
delayMicroseconds(500); //Wstrzymanie programu na 500us (0,5ms)
```

Obsługa przerwań polega na zatrzymaniu aktualnie wykonywanego kodu, wykonaniu szybkiej akcji opisanej w przerwaniu i powrocie do właściwego programu.

Układ Arduino Uno pozwala na obsługę przerwań wewnętrznych(software) i dwóch przerwań zewnętrznych(hardware) za pomocą napięć na pinach 2 (int.0) i 3 (int.1).

Przerwania dodaje się za pomocą funkcji `attachInterrupt()`

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);
```

ISR – nazwa funkcji, która ma się wywołać w czasie wywołania przerwania

mode – jeden z 3 czynników powodujących wywołanie przerwania na podanym pinie:

FALLING - gdy stan pinu zmieni się z wysokiego na niski

RISING - gdy stan pinu zmieni się z niskiego na wysoki

CHANGE - gdy stan pinu zmieni się

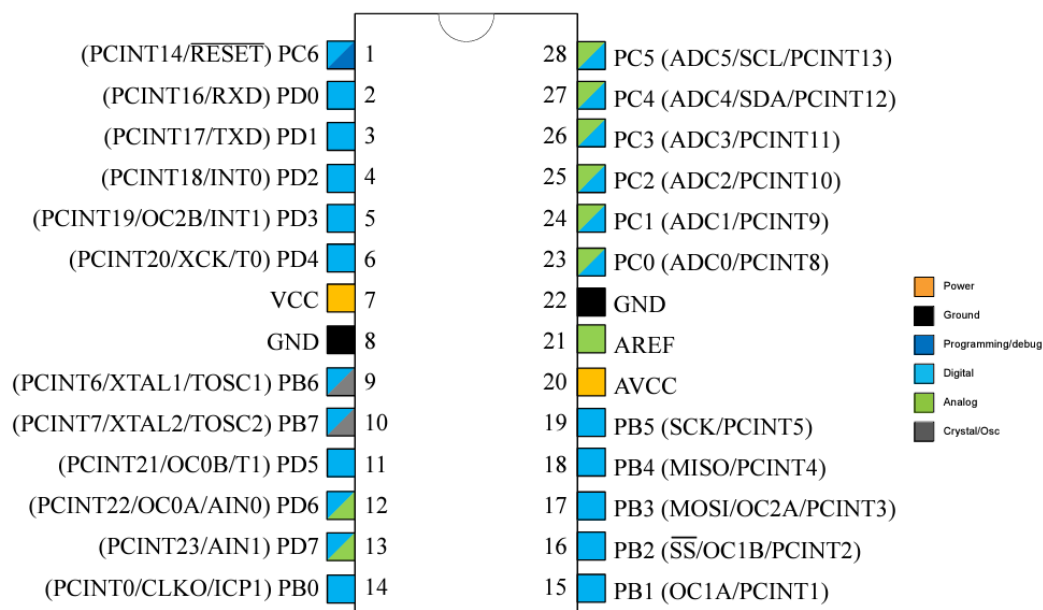
Przerwania są domyślnie włączone, ale można je wyłączać i włączać za pomocą metod:

```
noInterrupt();  
interrupt();
```

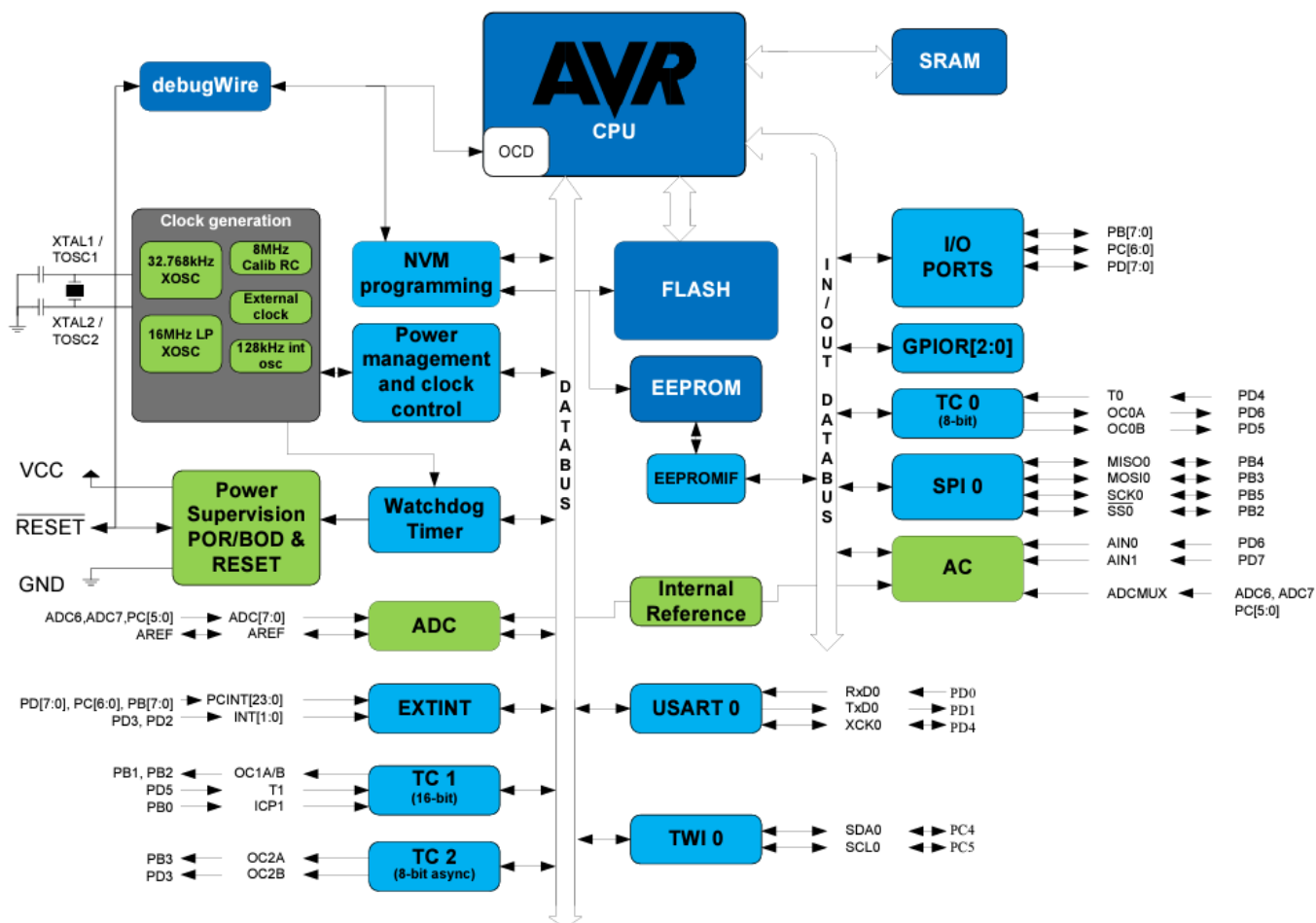
Za pomocą metody `detachInterrupt(digitalPinToInterrupt(pin))` można rozłączyć zdefiniowaną funkcję i przypisać inną.

Wewnątrz ISR (Interrupt Service Routine) nie można używać `delay / millis / micros / print, etc.`

Pinout mikrokontrolera ATmega328P 28P3



Schemat blokowy ATmega328P



Porty rejestrów pozwalają na niskopoziomowe manipulowanie pinami I/O. Rejestry są wyznaczonymi adresami w pamięci podręcznej. Dla łatwiejszego programowania rejestry mają swoje stałe nazwy, dzięki czemu nie trzeba posługiwać się adresami pamięci.

Arduino posiada 3 porty związane z pinami I/O:

B (piny 8 – 13)

C (analogowe piny wejściowe)

D (piny 0 – 7)

Każdy port jest kontrolowany przez 3 rejestry:
x to znak portu [B,C lub D]

DDRx – określa kierunek danych

0 – INPUT

1 – OUTPUT

PORTx – określa stan portu

0 – LOW

1 – HIGH

PINx – odczytuje stan wejściowy portu

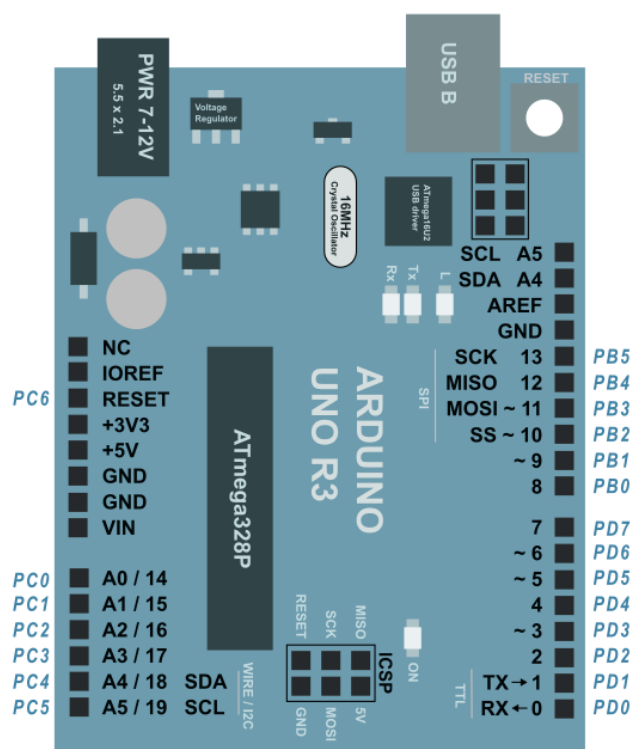


Tabela ustawień pinów na podstawie rejestrów DDR i PORT

DDRx	PORTx	I/O State	PULLUP
0	0	INPUT	NO
0	1	INPUT	YES
1	0	OUTPUT	-
1	1	OUTPUT	-

```
DDRD = B11111100; // ustawia kierunek pinów dla portu D (0-7), pin 0 i 1 - INPUT,
piny 2-7 OUTPUT
```

```
PORTD = B11100000; // ustawia sygnał dla pinów portu D, pin 0-4 LOW, pin 5-7 HIGH
```

DDB0 - to bit nr 0 w rejestrze DDRB (czyli pierwszy pin w sekcji B)

Maski bitowe |=, &= i inne są często spotykane przy operacjach z rejestrami. Pozwalają zmieniać tylko poszczególne bity w całej sekwencji rejestru.

PBx/PCx/PDx to stałe z wartościami x = 0,1,2,3,4,....

|= Maska ustawiająca

```
DDRB |= (1 << PB5); // Ustawia pin PB5 (pin 5 w porcie B) (ogólnie pin 13) na  
OUTPUT (reszta pinów zostaje niezmienniona)
```

&= Maska zerująca / sprawdzająca

```
PORTB &= ~(1 << PB2); // Ustaw bit PB2 na LOW
```

```
// Sprawdzenie czy pin PB2 jest w stanie HIGH  
if (PINB & (1 << PB2)) {  
    Serial.println("PB2 jest HIGH");  
}
```

^= Maska zmieniająca

```
PORTB ^= (1 << PB5); // Zmień stan bitu PB5(pin 13) na przeciwny
```

Ustawianie pinów za pomocą odwołań do adresu rejestrów

```
void setup() {
    // Bezpośrednia manipulacja rejestrami poprzez adresy
    (*(volatile uint8_t*)0x24) |= (1 << 5); // DDRB: Ustawienie bitu 5 (pin 13 jako
wyjście)
}
void loop() {
    // PORTB: Ustawienie bitu 5 (włączenie diody LED)
    (*(volatile uint8_t*)0x25) |= (1 << 5);
    delay(500);
    // PORTB: Wyzerowanie bitu 5 (wyłączenie diody LED)
    (*(volatile uint8_t*)0x25) &= ~(1 << 5);
    delay(500);
}
```

`volatile uint8_t*)0x24` Rzutowanie adresu 0x24 (rejestr DDRB) na wskaźnik do zmiennej typu `uint8_t`

`(*...)` odwołanie do wartości pod rzutowanym adresem

Praca na rejestrach to wzrost wydajności!

Zamiana funkcji `digitalWrite` na instrukcje na rejestrach zwiększają częstotliwość wywołań z 180kHz na 2MHz, czyli wzrost ponad 10x!!!

```
void setup() {
    pinMode(13, OUTPUT);
}

void loop() {
    //digitalWrite(13,1);
    //digitalWrite(13,0);
    PORTB |= B00100000; //PORTB |= (1 << PB5);
    PORTB &= ~B00100000; //PORTB &= ~(1 << PB5);
}
```

Instrukcja przełączająca pin działa odrobinę wolniej z częstotliwością 1,14MHz ponieważ musi odczytać wartość bitu nr 5, a następnie go odwrócić

```
PORTB ^= (1<< PB5);
```

Każda iteracja pętli `loop()` wiąże się z pewnymi operacjami działającymi w tle, np. sprawdzeniem eventu Serial. Operacje te dodają ok 250ns (4 x 62,5ns).

Funkcję `main` można nadpisać, aby zyskać delikatny wzrost wydajności, ale często nie jest to dobry pomysł.

Dla dociekliwych, poniżej treść funkcji `main`, którą można stworzyć na nowo we własnym skrypcie.

```
int main() {
    init();           // Initializes timers, ADC, and more
    setup();          // Calls the user-defined setup
    while (true) {
        loop();       // Calls the user-defined loop
        if (serialEventRun) serialEventRun();
    }
    return 0;
}
```

Przykładowy skrypt po nadpisaniu głównej funkcji `main` (taki skrypt przełącza piny z częstotliwością 2,66MHz).

```
#include <avr/io.h>
int main() {
    DDRB |= (1 << PB5);
    while (true) {
        PORTB |= (1 << PB5);
        PORTB &= ~(1 << PB5);
    }
    return 0;
}
```

Arduino posiada clock (zegar kwarcowy / oscylator) o częstotliwości 16Mhz. Każdy cykl zegara odbywa się co 1/16000000s czyli co 62.5ns. Jest to przybliżona wartość, zegar kwarcowy ma pewne odchylenie.

ATmega328P posiada 3 timery[0, 1, 2], które zliczają cykle zegara.

Timer0: 8-bit (0-255), używany do delay, millis, micros, itp.

Timer1: 16-bit (0-65535), używany w bibliotekach (np. servo)

Timer2: 8-bit (0-255), używany w funkcji tone()

Timery działają na rejestrach 8 i 16 bitowych, co oznacza, że mogą zliczać maksymalnie do liczby 255 lub 65535. Po przekroczeniu tej liczby counter się resetuje i generuje przerwanie.

Aby uzyskać dłuższe czasy można zmienić multiplikator (prescaler), który będzie zliczał cykle w pewnych odstępach np. x1, x8, x64, x256, x1024. Multiplikator znajduje się pomiędzy clockiem, a timerem. Domyślnie wszystkie timery zliczają cykle z prescalarem x64.

Timer0 z prescalarem x64 generuje przerwanie co ok.1ms, dzięki czemu stosuje się go w metodzie millis().

$256 [\text{ilość cykli}] \times 64 [\text{prescaler}] \times 62.5\text{ns} = 1024000\text{ns} = 1,024\text{ms}$

Prescalery mogą być ustawiane za pomocą rejestrów dla każdego z timerów:

Timer0 - TCCR0B

Timer1 - TCCR1B

Timer2 - TCCR2B

Rejestr **TCNT1** zwraca aktualną ilość cykli. Można go łatwo zresetować manualnie przypisując do niego 0.

Rejestr **OCR1A** przechowuje wartość, która jest porównywana z TCNT1, po spełnieniu równości rejestr TCNT generuje przerwanie.

```
ISR(TIMER1_COMPA_vect) {}
```

Rejestr **TIMSK1** służy do włączania i wyłączania przerwań związanych z Timer1. Włączenie odpowiednich bitów w tym rejestrze pozwala na wywoływanie przerwań przy określonych

zdarzeniach, takich jak osiągnięcie wartości porównawczej lub przepełnienie licznika. Przerwania te są użyteczne do wykonywania działań w precyzyjnych odstępach czasowych lub w odpowiedzi na zmiany wartości licznika.

Zmiana prescalera może wpływać np. na częstotliwość generatora sygnałów PWM, częstotliwość przerwań czy zmniejszenie poboru mocy.

Wzór na obliczanie wartości prescalera dla timera1

$$\text{Prescaler Value} > \frac{16 \times 10^6 \times t[s]}{65535}$$

$$\text{OCRn} = 65535 - \frac{16 \times 10^6 \times \text{Time}[s]}{\text{Prescaler Value}}$$

Zmiana prescalera odbywa się na 3 pierwszych bitach rejestru TCCR1B (CS10, CS11, CS12)

TCCR1B	7	6	5	4	3	2	1	0
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10

CS12	CS11	CS10	USE
0	0	0	No Clock Timer STOP
0	0	1	1 (No Prescaling)
0	1	0	8 (From Prescaler)
0	1	1	64 (From Prescaler)
1	0	0	256 (From Prescaler)
1	0	1	1024 (From Prescaler)
1	1	0	External clock source on T1 Pin. Clock on falling edge
1	1	1	External Clock source on T1 pin. Clock on rising edge.

Zmiana prescalera

```
TCCR1B = 0; // reset rejestru
TCCR1B |= B00000100; // prescaler 256
```

Blink LED co 1s

```
#include <avr/io.h>

void setup() {
    TCCR1A = 0; //reset timera1
    TCCR1B = 0; //reset prescalera
    TCCR1B |= B00000100; // prescaler x256
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
    if (TCNT1 > 62500) {
        TCNT1 = 0; //reset licznika
        digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
    }
}
```

Blink LED co 0.5s z użyciem przerwania

```
int led = LED_BUILTIN;
bool LED_STATE = true;
void setup() {
    pinMode(led, OUTPUT);
    cli();                //stop interrupts for till we make the settings
    TCCR1A = 0;           // Reset entire TCCR1A to 0
    TCCR1B = 0;           // Reset entire TCCR1B to 0
    TCCR1B |= B00000100; //Set CS12 to 1 so we set prescaler to 256
    TIMSK1 |= B00000010;
    OCR1A = 31250;        //set compare register A to this value
    sei();                //Enable back the interrupts
}

void loop() {}

ISR(TIMER1_COMPA_vect) {
    TCNT1 = 0;
    LED_STATE = !LED_STATE;
    digitalWrite(led, LED_STATE);
}
```

Obsługa timera1 do zmierzenia czasu wywołania funkcji.

```
void setup() {
  Serial.begin(115200);
  TCCR1B = bit(CS10); // ustawia mnożnik timera1 na x1
  TCNT1 = 0; // resetuje licznik timera na 0
  // Funkcja do zmierzenia
  DDRB = B00111000; // zamiennik pinMode dla wielu pinów 8-13
  PORTB = B00111000; // zamiennik digitalWrite() dla wielu pinów jednocześnie
  // Koniec funkcji do zmierzenia
  unsigned long value = TCNT1; // zczytuje wartość licznika timera
  Serial.print("Cycles: ");
  Serial.println(value - 1);
  Serial.print("Microseconds: ");
  Serial.println((float)(value - 1) / 16);
}
void loop() {}
```

Biblioteka TimerOne

[Link do biblioteki TimerOne na github](#)

Za pomocą biblioteki TimerOne można stworzyć przerwania wewnętrzne do wykonywania operacji cyklicznych.

```
#include <TimerOne.h>
int ledPin = 10;

void ledBlink() {
  digitalWrite(ledPin, !digitalRead(ledPin));
}

void setup() {
  pinMode(ledPin, OUTPUT);
  Timer1.initialize(1000000); //in microseconds
  Timer1.attachInterrupt(ledBlink);
}

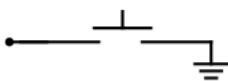
void loop() {}
```

Przycisk chwilowy to jeden z podstawowych komponentów analogowych stosowany w robotyce i automatyce. Przyciski chwilowe występują w wersji NO - Normally Open oraz NC – Normally Closed. W arduino zazwyczaj ustawia się go w trybie pulldown lub pullup za pomocą rezystora 10k. Pulldown ściąga stan na niski (pin jest połączony z GND przez rezystor), a pullup podnosi stan na wysoki (pin jest połączony z 5V).

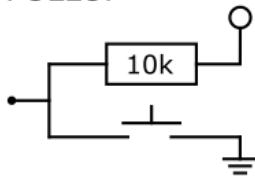


Zastosowanie kondensatora o małej pojemności, pozwoli zminimalizować zjawisko drgania styków przycisku (debouncing).

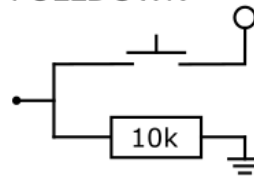
INPUT_PULLUP



PULLUP



PULLDOWN



Wykrywanie pojedynczych kliknięć wymaga dodatkowych zabiegów, np. poprzez sprawdzanie poprzedniego i aktualnego stanu przycisku.

```
// Zliczanie kliknięć przycisku chwilowego
bool btnPrevState = false;
int counter = 0;
int btnPin = 4;
void setup() {
  btnPrevState = digitalRead(btnPin);
  Serial.begin(9600);
}
void onClick() {
  bool btnNowState = digitalRead(btnPin);
  if (btnNowState == 1 && btnPrevState == 0) {
    counter++;
    Serial.println(counter);
  }
  btnPrevState = btnNowState;
}
void loop() {
  onClick();
}
```

```
// Przycisk chwilowy z millis
const int bounce = 200;
unsigned long lastClickedTime = 0;
int counter = 0;
int btnPin = 4;

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

void checkButton() {
  if (digitalRead(btnPin) == HIGH) {
    if (millis() - lastClickedTime > bounce) {
      lastClickedTime = millis();
      counter++;
      Serial.println(counter);
    }
  }
}

void loop() {
  checkButton();
}
```

Przycisk chwilowy z przerwaniami

```
// Przykład: Push button z interrupt
volatile int counter = 0;

void setup(){
  Serial.begin(9600);
  attachInterrupt(digitalPinToInterrupt(2), myISR, RISING);
}

void loop(){}

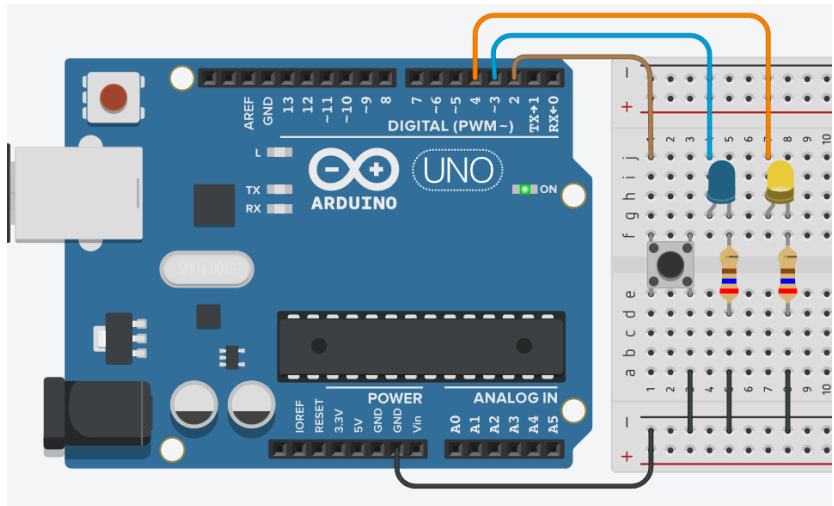
void myISR(){
  counter++;
  Serial.println(counter);
}
```

Wielofunkcyjny przycisk chwilowy z millis

```
int pressLength = 0;
unsigned long lastDebounceTime = 0;
int debounceDelay = 50;
int opt1delay = 1000;
int opt2delay = 2000;
int buttonPin = 2;
int out1 = 3;
int out2 = 4;

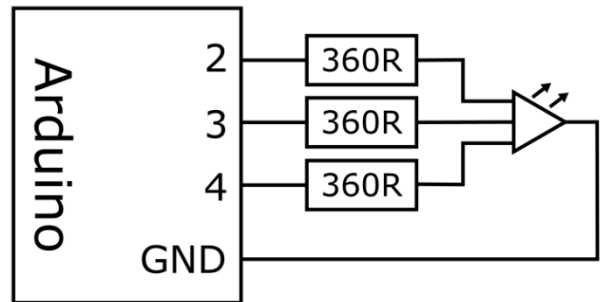
void setup() {
  pinMode(buttonPin, INPUT_PULLUP);
  pinMode(out1, OUTPUT);
  pinMode(out2, OUTPUT);
}

void loop() {
  unsigned long currentTime = millis();
  int buttonState = digitalRead(buttonPin);
  if (buttonState == LOW && (currentTime - lastDebounceTime) > debounceDelay) {
    pressLength += debounceDelay;
    lastDebounceTime = currentTime;
  } else if (buttonState == HIGH) {
    pressLength = 0;
  }
  if (pressLength >= opt2delay) {
    digitalWrite(out2, HIGH);
  } else if (pressLength >= opt1delay) {
    digitalWrite(out1, HIGH);
  } else if (pressLength >= 100) {
    digitalWrite(out1, LOW);
    digitalWrite(out2, LOW);
  }
}
```



Dioda RGB to element zawierający wewnątrz 3 diody: czerwoną, zieloną i niebieską. Sterując jasnością każdej z nich, uzyskuje się kolor końcowy diody RGB.

Na rynku dostępne są diody ze wspólną katodą(-) lub anodą(+).



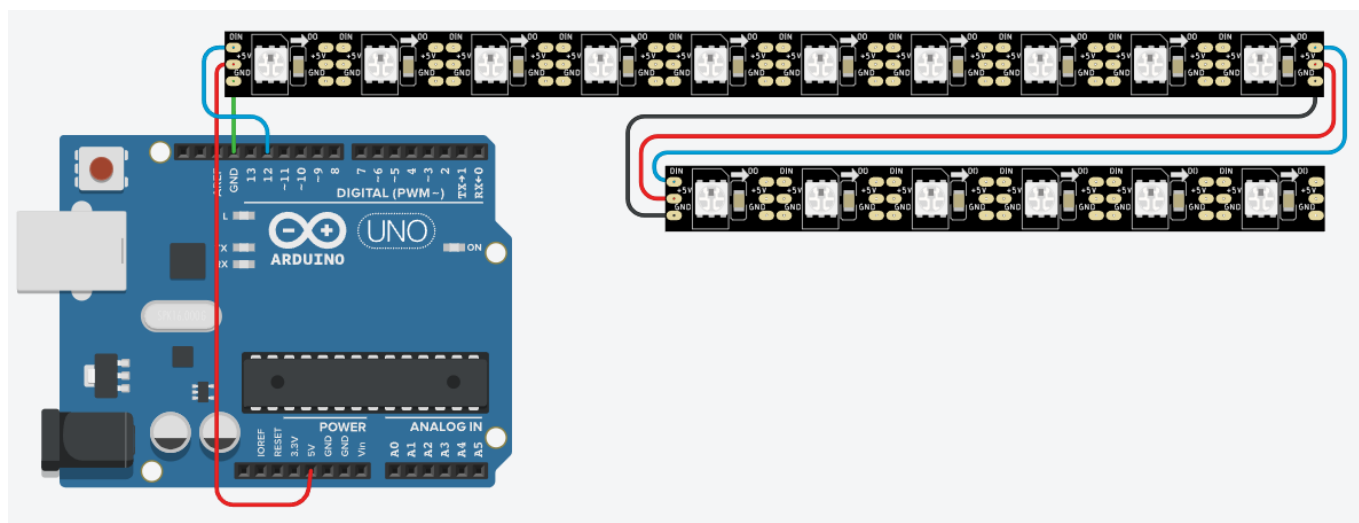
```
// Przykład: Losowy kolor diody RGB
int interval = 500;
long lastTime = 0;
int pins[] = {3, 5, 6};
int pinsLength;

void setup(){
    randomSeed(analogRead(A0));
    pinsLength = sizeof(pins)/sizeof(pins[0]);
    for (int i = 0; i < pinsLength; i++){
        pinMode(pins[i], OUTPUT);
    }
}

void RandomizeRGBLed(){
    for (int i = 0; i < pinsLength; i++) {
        int rand = random(0, 256);
        analogWrite(pins[i], rand);
    }
}

void loop(){
    long timeNow = millis();
    if(timeNow - interval >= lastTime){
        lastTime = timeNow;
        RandomizeRGBLed();
    }
}
```

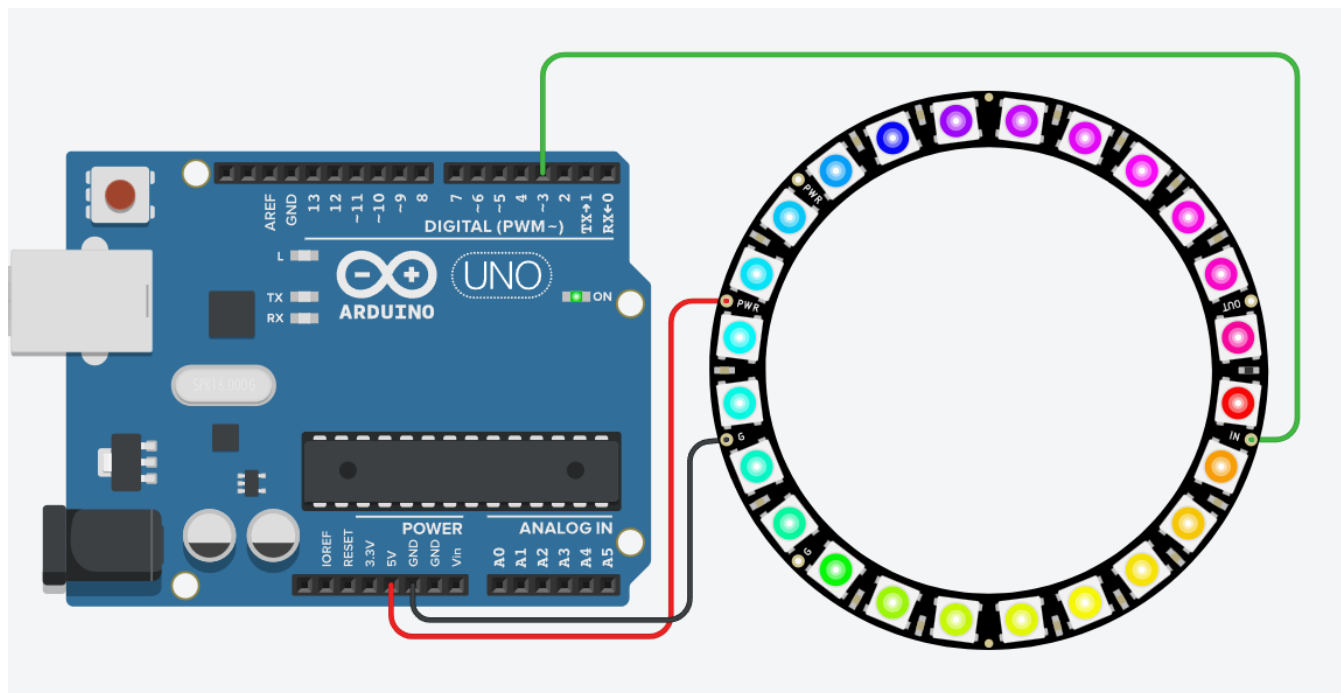
Programowalne diody NeoPixel



Paski NeoPixel można dowolnie łączyć.

Utworzenie obiektu Adafruit_NeoPixel o nazwie strip	Adafruit_NeoPixel strip(ilość diod, nrPinu);
Inicjalizacja obiektu o nazwie strip	strip.begin();
Ustalenie koloru rgb diody o numerze n	strip.setPixelColor(n, red, green, blue);
Inicjalizacja podanych wcześniej kolorów	strip.show();
Dodanie koloru do wielu diod jednocześnie	strip.fill(color, first, count); ostatnią diodę można oznaczyć jako: strip.numPixels() - 1

Animacja kolorów z koła wartości HSV



```
#include <Adafruit_NeoPixel.h>
#define LED_PIN 3
#define LED_COUNT 24
#define HUE_RING 65536
#define HSV_MAX 255
// uint32_t rgbcolor = strip.ColorHSV(hue, saturation, brightness);
Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRB);
void setup() {
    strip.begin();
}
void loop() {
    updateLeds();
}
void updateLeds() {
    for (int i = 0; i < LED_COUNT; i++) {
        strip.setPixelColor(i, strip.ColorHSV(i * HUE_RING / LED_COUNT, HSV_MAX,
HSV_MAX));
        strip.show();
        delay(100);
    }
    strip.clear();
    strip.show();
}
```

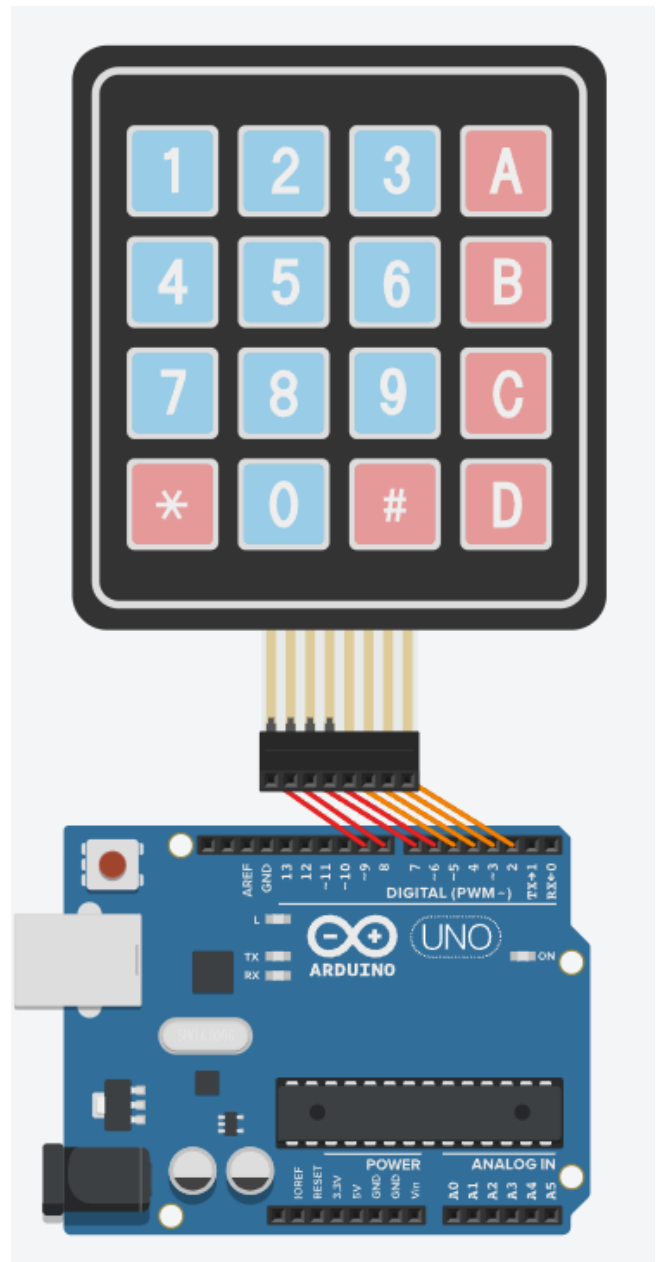
```
#include <Keypad.h>
const byte ROWS = 4;
const byte COLS = 4;

//mapowanie klawiatury
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'.','0','#','D'}
};

// przypisanie pinów
byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};

Keypad keypad = Keypad(
  makeKeymap(keys), rowPins, colPins,
  ROWS, COLS );

void setup(){
  Serial.begin(9600);
}
void loop(){
  char key = keypad.getKey();
  if(key){
    Serial.print("Key Pressed : ");
    Serial.println(key);
  }
}
```

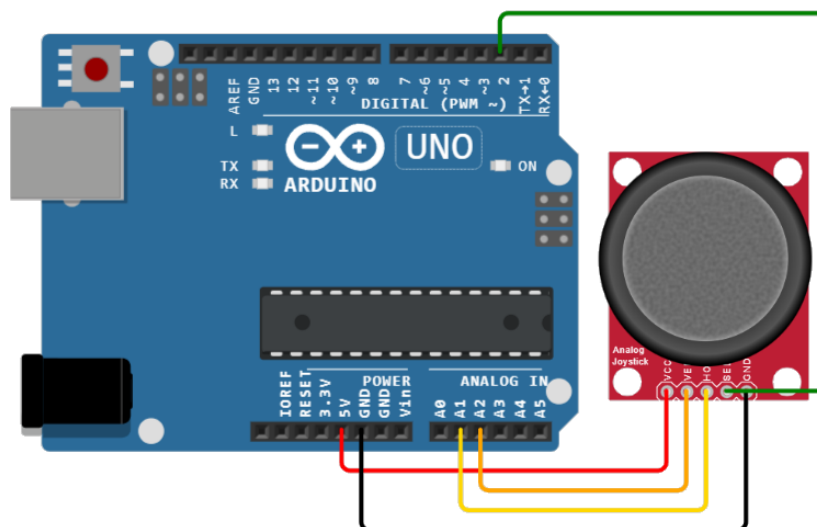


Klawiatura 4x4 wymaga podłączenia 8 pinów cyfrowych co znacznie ogranicza dalsze wykorzystanie mikrokontrolera. W takich sytuacjach można skorzystać z expanderów, np.: PCF8574

Joystick analogowy z przyciskiem

Analogowy joystick wyposażony jest w przycisk oraz pozwala na wykrywanie ruchu gałki w dwóch osiach x i y. Zwraca wartości od 0 do 1023 dla maksymalnych wychyleń oraz połowę dla pozycji centralnej.

Gałka w prawym-dolnym rogu zwraca wartość (0, 0), a w lewym-górnym (1023, 1023).



Pobrane wartości można mapować na bardziej użyteczny zakres, np.: od -100 do 100.

```
map(analogRead(XJOY), 0, 1023, -100, 100)
```

```
//Przykład: Pobieranie danych z joysticka analogowego
const int XJOY = A1;
const int YJOY = A2;
#define SEL_PIN 2

void setup() {
    pinMode(SEL_PIN, INPUT_PULLUP);
    Serial.begin(9600);
}

void loop() {
    int joyX = map(analogRead(XJOY), 0, 1023, -100, 100);
    int joyY = map(analogRead(YJOY), 0, 1023, -100, 100);
    bool selPressed = digitalRead(SEL_PIN) == LOW;
    Serial.print(joyX);
    Serial.print(", ");
    Serial.println(joyY);
    Serial.println(selPressed);
}
```

Brzęczyk piezoelektryczny

Głośnik piezoelektryczny przekształca sygnały elektryczne w dźwięk na zasadzie deformacji mechanicznej materiału piezoelektrycznego (najczęściej ceramiki), pod wpływem przyłożonego napięcia.

Na rynku dostępne są głośniki piezoelektryczne: pasywne i aktywne.



Głośnik aktywny ma wbudowany oscylator o stałej częstotliwości (najczęściej 2kHz), działa bezpośrednio po podłączeniu do zasilania DC.



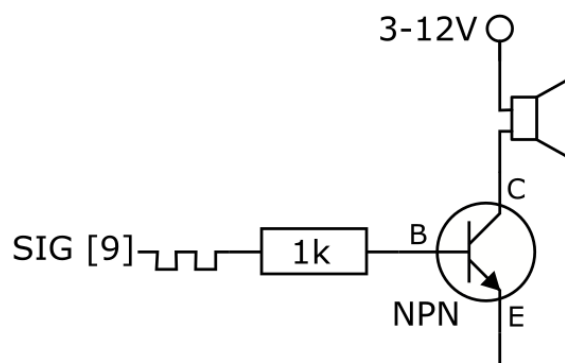
Głośnik pasywny wymaga zewnętrznego sygnału zmiennego, dzięki któremu można regulować częstotliwość i amplitudę dźwięku.

Odgrywanie dźwięków w Arduino jest możliwe dzięki metodzie `tone`, która generuje sygnał prostokątny z częstotliwością w zakresie od ok. 31Hz – 80kHz. Spektrum dźwięków słyszalnych dla człowieka to przedział od 20Hz – 20kHz

```
tone (pinPWM, częstotliwość dźwięku, [długość odtwarzania]);
```

Pełna biblioteka i dokumentacja: <https://github.com/bhagman/Tone#ugly-details>

```
const int speakerPin = 9;
void setup(){
    pinMode(speakerPin, OUTPUT);
}
void loop(){
    tone(speakerPin, 500, 500);
    delay(1500);
}
```



Rys. 2 Schemat połączenia głośnika z tranzystorem

Odgrywanie melodii za pomocą pasywnego głośnika piezoelektrycznego

```
int speakerPin = 9;
// pitches for C scale
int c4 = 262;
int d4 = 294;
int e4 = 330;
int f4 = 349;
int g4 = 392;
int a4 = 440;
int b4 = 494;
int c5 = 523;

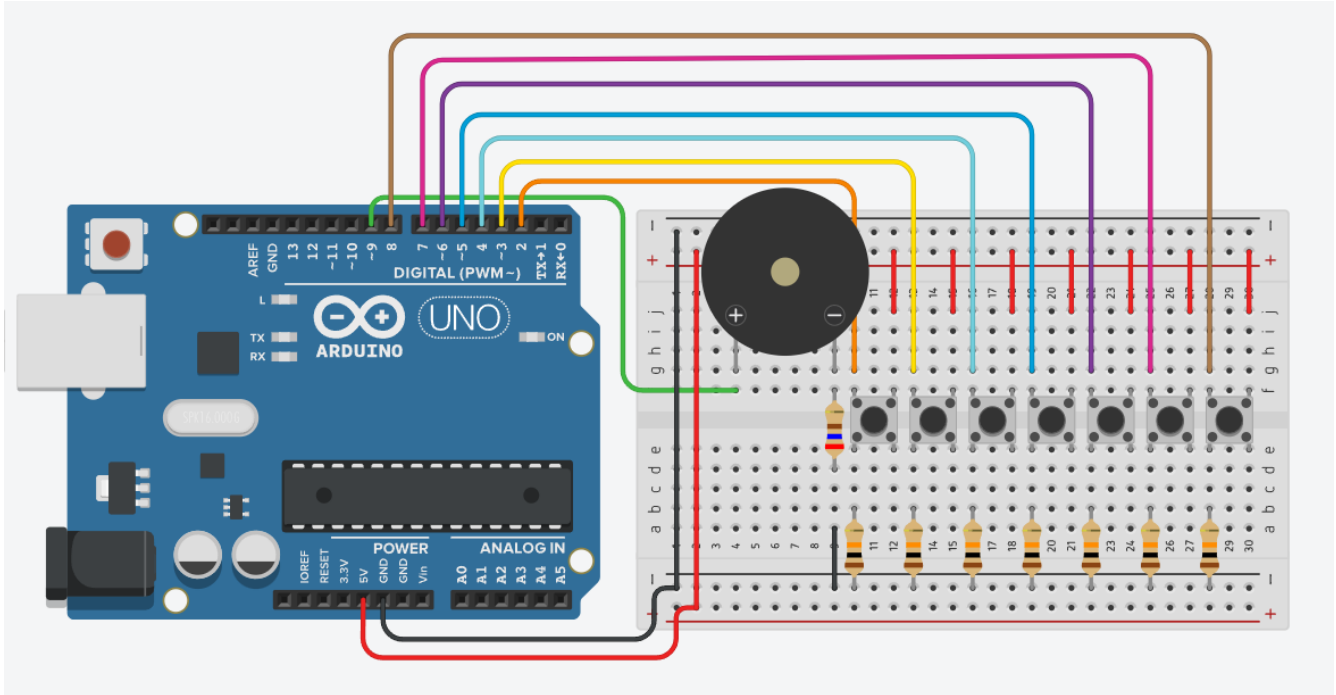
// rhythm
int tempo = 1000;
int w = tempo; //whole note - 4 beats
int h = tempo/2; // half note - 2 beats
int q = tempo/4; // quarter note - 1 beat
int e = tempo/8; // eight note - 1/2 beat
int s = tempo/16; // sixteen note - 1/4 beat
int melody[] = {a4,c4,d4,d4,d4,e4,f4,f4,f4,g4,e4,e4,d4,c4,d4};
int rhythm[] = {e,e,q,q,e,e,q,q,e,e,q,q,e,e,h};

void playMelody(){
    int len = sizeof(melody)/sizeof(melody[0]);
    for(int i=0; i<len; i++){
        tone(speakerPin, melody[i], rhythm[i]);
        delay(rhythm[i]*2);
    }
}

void setup(){
    pinMode(speakerPin, OUTPUT);
    playMelody();
}

void loop(){}
}
```

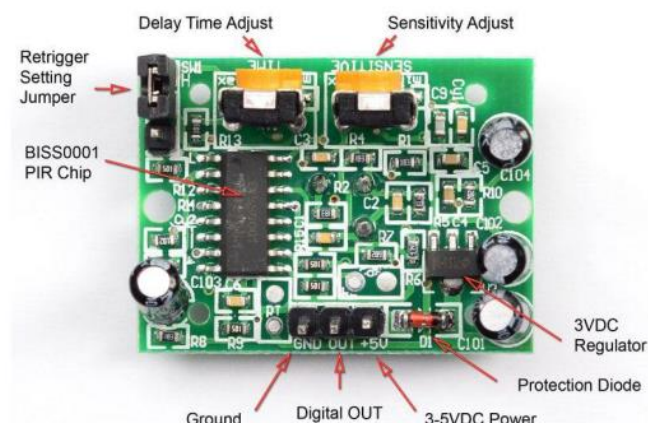
Pianino z użyciem piezo i przycisków chwilowych



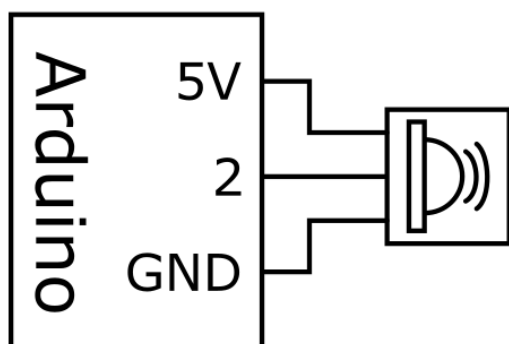
```
// Przykład: Pianino z użyciem piezo i przycisków chwilowych
int tones[] = {261,293,330,350,392,440,494};
int keys[] = {2,3,4,5,6,7,8};
int speakerPin = 9;

void setup(){
  pinMode(speakerPin, OUTPUT);
}
void loop(){
  for(int i=0; i<7; i++){
    if(digitalRead(keys[i])==true){
      tone(speakerPin, tones[i],50);
    }
  }
}
```

PIR sensor służy do wykrywania ruchu na podstawie emitowanych przez obiekty fali podczerwonych. W nowych wersjach sensora, można ustawiać np. czułość lub opóźnienie sygnału OUT.



Zasada działania PIR



Sensor podzielony jest na 2 części, które rejestrują otoczenie. W momencie wykrycia ciepłego ciała powstaje dodatnia zmiana napięć pomiędzy półówkami, a w momencie opuszczenia strefy czujnika, tworzy się ujemna różnica. Ta zmiana generuje puls napięcia na pinie OUT.

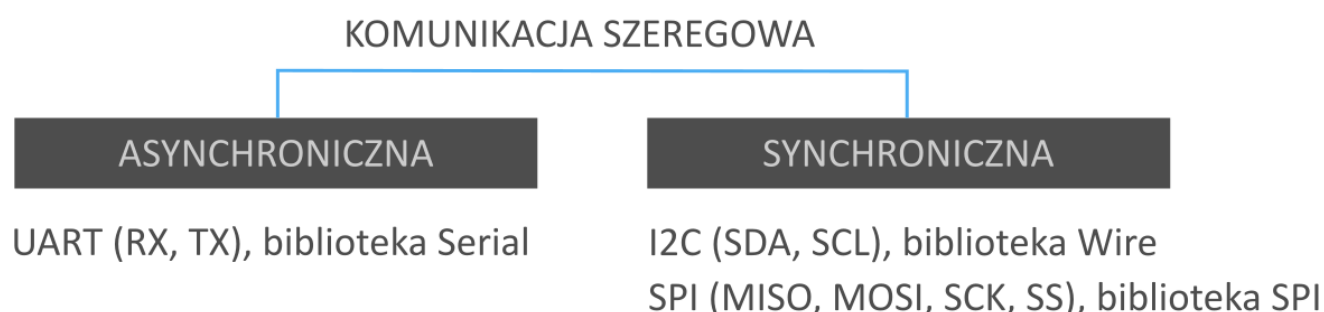
```
// Przykład: Wykrycie ruchu za pomocą sensora PIR
void setup(){
  Serial.begin(9600);
}
void loop(){
  int pir = digitalRead(A0);
  if(pir == 1){
    Serial.println("Ruch wykryty");
  }else{
    Serial.println("Brak ruchu");
  }
  delay(10);
}
```

Komunikacja szeregowa i równoległa to dwa podstawowe sposoby przesyłania danych między urządzeniami.

W komunikacji równoległej jednocześnie przesyłanych jest kilka bitów, co zwiększa szybkość transmisji, ale wymusza użycie większej ilości przewodów.

W przypadku komunikacji szeregowej informacje przesyła się sekwencyjnie, bit po bicie. Transmisja danych następuje z każdym cyklem zegara. Dane wysyłane do danego urządzenia są przechowywane w buforze, który może pomieścić 64 Bajty danych.

W arduino korzysta się głównie z komunikacji szeregowej.



Komunikacja szeregowa może występować w 3 trybach:

Simplex Mode - Komunikacja jednokierunkowa, aktywne jest tylko jedno urządzenie: nadające lub odbierające

Half Duplex Mode - Aktywne są obydwa urządzenia, ale nie w tej samej chwili. Wysyłanie i odbieranie danych przebiega naprzemiennie, np. Internet, Walkie-talkie

Full Duplex Mode - Obydwa urządzenia mogą jednocześnie wysyłać i odbierać dane

Komunikacja USART (Serial Port)

USART (Universal Synchronous / Asynchronous Receiver-Transmitter) to sprzętowy moduł, który umożliwia komunikację szeregową między płytką Arduino, a innymi urządzeniami (np. komputer, moduły czujników, Bluetooth, inne mikrokontrolery) za pomocą pinów Rx i Tx.

UART realizuje transmisję i odbiór danych w sposób asynchroniczny, co oznacza, że nie wymaga dodatkowego sygnału zegarowego do synchronizacji.

Arduino posiada wbudowany moduł CP2102 (lub CH340 w przypadku chińskich zamienników) do komunikacji seryjnej z komputerem za pomocą kabla USB.

TTL (Transistor-Transistor Logic) to standard logicznego poziomu napięcia używany w wielu urządzeniach elektronicznych. W arduino korzysta się często z poziomu napięć 3.3V i 5V.

Transmisja może nastąpić tylko wtedy, kiedy obydwa urządzenia dzielą te same ustawienia:

- Szybkość transmisji (baud rate – częstotliwość pulsów zegara / ilość bitów na sekundę)
- Format ramki danych
- Warunek rozpoczęcia / zakończenia transmisji (start/stop)

Inicjalizacja portu szeregowego

Inicjalizacja portu szeregowego ze standardową prędkością 9600 i domyślnym ustawieniem ramki

```
Serial.begin(9600);
```

Format ramki

Format ramki jest parametrem opcjonalnym, w większości przypadków zastosowanie domyślnego formatu (SERIAL_8N1) jest wystarczające. Domyślny parametr określa 8 bitów danych, bez kontroli bitu parzystości, 1 bit stop. Urządzenie odbierające dane musi spodziewać się takiego właśnie ustawienia.

1 bit – Start bit (rozpoczynający transmisję, gdy jego stan zmienia się z 1 na 0)

5 – 9 bitów informacji (bity wysyłane są od najmniej znaczącego bitu - LSBF)

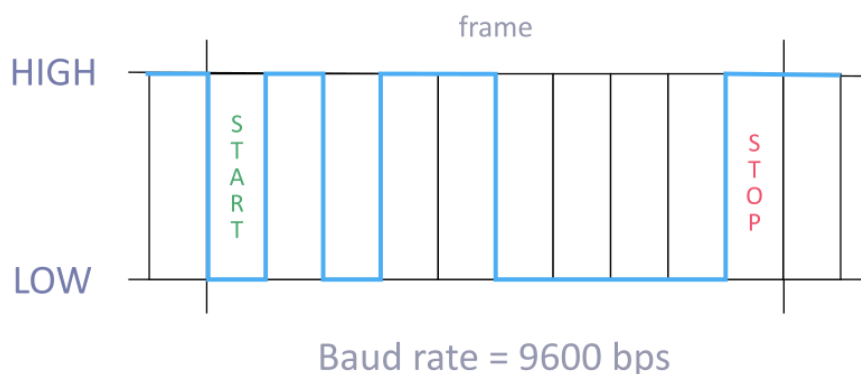
1 bit kontrolny parzystości (opcjonalny) brak - N, parzyste – E, nieparzyste – O)

1 lub 2 bity – Stop bit (1 lub 2)



Przykład przesłania wartości 13.

$$13_{10} = 0000\ 1101_2$$



W Arduino IDE można przesyłać i odbierać dane za pomocą Serial Monitor.

Wysyłanie danych

Dane można wysyłać za pomocą metody print lub write:

print / println wysyła dane skonwertowane na znaki (ASCII)

```
Serial.print(12);  
Serial.println("Hej");
```

Funkcja println wyświetla argumenty i dodaje znak przejścia do nowej linii ('\r\n')

```
Serial.print("Hej");  
Serial.print('\r');  
Serial.print('\n');
```

write wysyła dane bez konwersji na znaki ASCII

```
Serial.write(65);
```

Pobieranie danych

Wysyłanie danych do komputera powoduje ich wyświetlenie w Serial.Monitorze. W przypadku wysyłania danych między płytkami lub z komputera do płytki, należy te dane odebrać i przetworzyć.

Serial.available

Dane wysyłane do portu szeregowego są przechowywane w 64 (lub 128) bajtowym buforze, które są usuwane zaraz po pobraniu. Pobierając dane, warto sprawdzić czy są one dostępne. Można to zrobić za pomocą metody Serial.available(), która zwraca ilość bajtów danych do odczytania z Serial port (wraz z Enterem). Jeśli bufor jest pusty, funkcja zwróci wartość 0.

```
if (Serial.available() > 0) { }
```

Program odczyta dane z bufora w określonym domyślnie czasie (1s).

Czas oczekiwania można zmienić za pomocą metody:

```
Serial.setTimeout(1000); // czas w ms
```

Dane z portu Serial można pobierać za pomocą kilku dostępnych metod:

Serial.read()

```
// zwraca wartości numeryczne znaków lub znak ASCII w zależności od typu danych
byte recData = Serial.read(); // zwraca wartość liczbową znaku ASCII
char recChar = Serial.read(); // zwraca znak ASCII

void setup() {
    pinMode(13, OUTPUT);
    Serial.begin(9600);
}
void loop() {
    if (Serial.available() > 0) {
        char a = Serial.read();
        if (a == '1') {
            digitalWrite(13, 1);
        } else if (a == '0') {
            digitalWrite(13, 0);
        }
    }
}
```

Serial.readString()

```
// zwraca wartość typu string
String recData = Serial.readString();
```

Serial.readStringUntil()

```
// zwraca wartość typu string bez znaku Enter
String recData = Serial.readStringUntil('\n');
```

Serial.readBytes()

```
// zwraca liczbę otrzymanych bajtów (bytesLength), które można przypisać do tablicy
o określonej długości (BUF_SIZE)
const int BUF_SIZE = 50;
char bytesReceived[BUF_SIZE];

void setup() {
    Serial.begin(9600);
}
void loop() {
    if (Serial.available()) {
        int bytesLength = Serial.readBytes(bytesReceived, BUF_SIZE);
        Serial.print("Liczba bajtów: ");
        Serial.println(bytesLength);
        Serial.print("Otrzymane bajty: ");
        for(int i = 0; i < bytesLength; i++)
            Serial.print(bytesReceived[i]);
    }
}
```

Serial.parseInt()

parseInt zwraca z bufora pierwszą napotkaną wartość liczbową

```
// zwraca kolejną liczbę całkowitą (lub 0 w przypadku przekroczenia czasu)
if(Serial.available()){
    int a = Serial.parseInt();
    Serial.print(a);
}
```

parseInt przyjmuje argumenty: SKIP_NONE, SKIP_ALL, SKIP_WHITESPACE

```
Serial.parseInt(SKIP_ALL, ',');
```

Serial.parseFloat()

```
// zwraca kolejną liczbę rzeczywistą (lub 0 w przypadku przekroczenia czasu)
if(Serial.available()){
    float b = Serial.parseFloat();
    Serial.print(b);
}
```

serialEvent() to metoda/event, który wywoła się za każdym razem, kiedy w buforze danych pojawi się informacja do pobrania. Ważne by pobrać dane z bufora, a tym samym go wyzerować, aby event nie wykonywał się ciągle.

```
void serialEvent(){
    if(Serial.available()){
        Serial.println(Serial.read());
    }
}
```

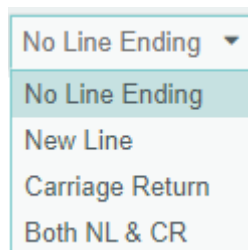
Line ending

Wysyłane dane mają swój początek i koniec. Istnieją specjalne znaki, które to określają.

Carriage Return '`\r`'

New line '`\n`' (w systemie windows `\n` zawiera obydwa znaki: `\r\n`)

W zależności od projektu i sposobu komunikacji, należy dostosować opcję zakończenia linii w Arduino IDE.



Moduł USB to TTL

Komunikacja TTL z komputerem może przebiegać za pomocą specjalnego modułu USB to TTL.



Połączenie pinów z Arduino

USB to TTL	3V3	TXD	RXD	GND	+5V
ARDUINO		RXD	TXD	GND	5V

Wgrywanie programu za pomocą USB to TTL

W Arduino IDE należy wybrać: Tools / Programmer / AVRISP mkII

Tuż po kliknięciu przycisku Upload, na płytce należy wcisnąć przycisk Reset

Software Serial

<https://docs.arduino.cc/learn/built-in-libraries/software-serial/>

Biblioteka SoftwareSerial pozwala na komunikację szeregową za pomocą innych pinów bez użycia sprzętowego interfejsu USART.

Wysyłanie i otrzymywanie danych w tym samym czasie nie jest możliwe.

W przypadku użycia kilku portów, tylko jeden może otrzymywać dane w danej chwili.

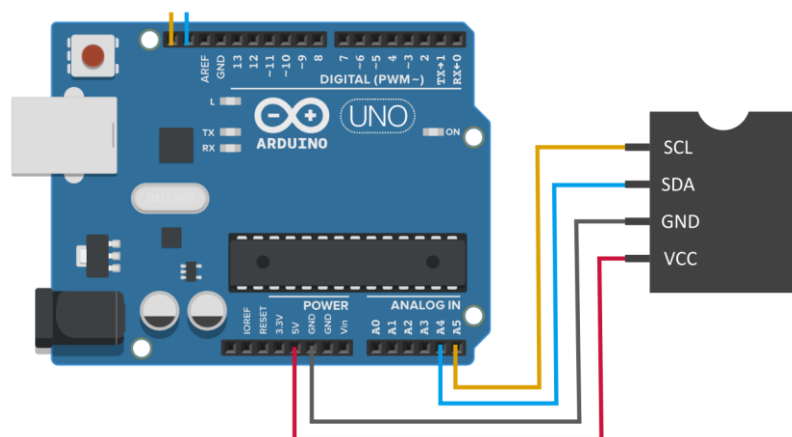
Połączenie pinów z Arduino

USB to TTL	3V3	TXD	RXD	GND	+5V
ARDUINO		2	3	GND	5V

```
#include <SoftwareSerial.h>
#define rxPin 2
#define txPin 3
SoftwareSerial mySerial = SoftwareSerial(rxPin, txPin);
int counter = 0;

void setup() {
    pinMode(rxPin, INPUT);
    pinMode(txPin, OUTPUT);
    mySerial.begin(9600);
}
void loop() {
    mySerial.println(counter);
    counter++;
    delay(1000);
}
```

Magistrala I2C (inter-integrated circuit), zwana również TWI (two wire interface), I²C, IIC to urządzenie do przesyłania danych na jednej linii (half duplex)



Interfejs I2C (Two-Wire) jest przydatny do łączenia wielu urządzeń, ponieważ wszystkie mogą dzielić te same dwa piny (oraz wspólny pin uziemienia).

Urządzenia w sieci I2C są „adresowalne”. Adresy są przekazywane za pomocą 7 bitów, stąd zakres od 0 do 127. Każde urządzenie musi mieć unikalny adres z zakresu od 8 do 119. Adres 0 jest zarezerwowany jako adres „rozgłoszeniowy”, adresy od 1 do 7 są zarezerwowane do innych celów, a adresy od 120 do 127 są zarezerwowane na przyszłość.

Domyślny rozmiar bufora transmisji w bibliotece Wire wynosi 32 bajty. Oznacza to, że całkowity rozmiar danych, które można przesłać w jednej transmisji I2C (między `Wire.beginTransmission()` a `Wire.endTransmission()`), nie może przekroczyć 32 bajtów.

Połączenie I2C

I2C jest połączeniem open drain, to znaczy, że linie są normalnie w stanie wysokim, a komunikacja rozpoczyna się po zmianie stanu na niski.

Analog port 4 (A4) = SDA (serial data)

Analog port 5 (A5) = SCL (serial clock)

Piny SDA i SCL wymagają pull-up rezystora, aby utrzymywać je w stanie wysokim, zazwyczaj jest to 4.7K. Atmega328 jest skonfigurowany z wewnętrznym pull-up rezystorem więc dla niewielkich sieci, na krótkich odległościach dodatkowe rezystory nie są niezbędne.

Interfejs I2C pozwala na komunikację synchroniczną, ponieważ jeden z przewodów SCL dyktuje taktowanie zegara.

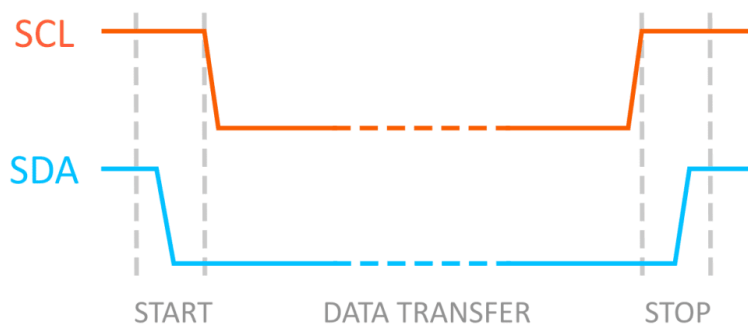
Przesyłanie danych za pomocą interfejsu I2C

Komunikacja I2C polega na:

Ustawieniu warunku START

Przesyłu danych(adres urządzenia podrzędnego, byte danych)

Ustawieniu warunku STOP



START

Linia SCL jest w stanie wysokim, następnie linia SDA przyjmuje stan niski, a tuż po tym również linia SCL przyjmuje stan niski.

STOP

Transmisję kończy warunek STOP. Linia SCL jest w stanie wysokim, a następnie linia SDA zmienia stan na stan wysoki.

W czasie transmisji będą zmieniały się stany obu linii.

Napięcie na linii SDA będzie się zmieniać ze względu na przesyłane dane (LOW – 0, HIGH – 1).

W momencie pulsu zegara SCL rejestrowany jest stan na linii SDA.

Adres urządzenia podrzędnego 7 bitów + R/W + ACK

Jako pierwszy transmitowany jest 7-bitowy adres docelowego urządzenia podrzędnego, zaczynając od najważniejszego bitu.

Następnie transmitowany jest 1 bit odczytu/zapisu. 1 oznacza odczyt, a 0 zapis.

Master nasłuchuje odpowiedzi urządzenia docelowego, stan niski na linii SDA jest potwierdzeniem (ACK) istnienia urządzenia o tym adresie oraz gotowości do odbioru danych. Jeśli nie ma podłączonego i włączonego urządzenia o tym adresie, lub nie ma on wymaganego adresu, to urządzenie podrzędne zignoruje ten adres, pozostawiając linię SDA wysoką (dzięki rezystorowi podciągającemu). To traktowane jest jako odmowa (NAK).

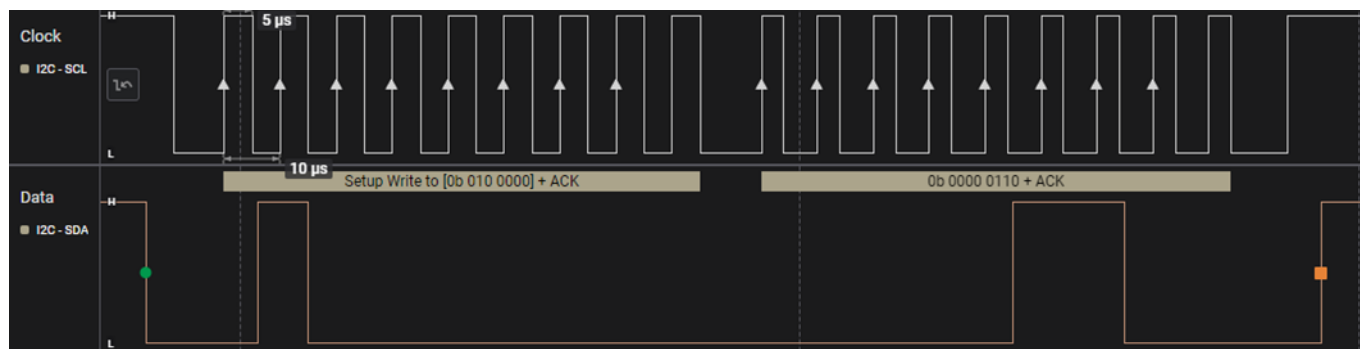
Adres rejestru 8 bitów + ACK

Następnie nadawany jest adres rejestru urządzenia (8 bitów) potwierdzony ponownie bitem ACK.

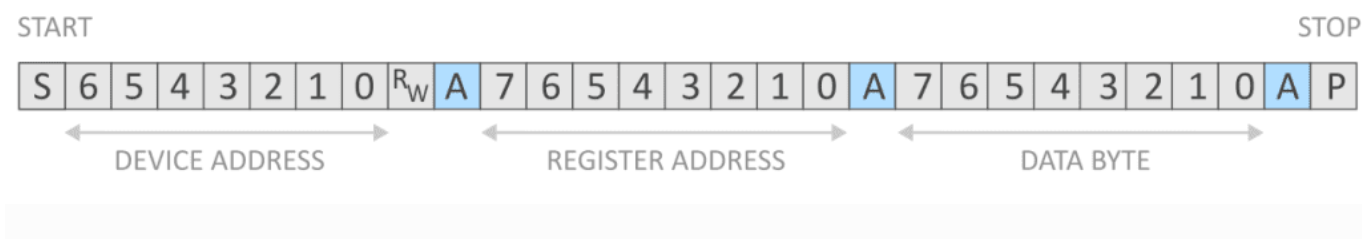
Przesył danych 8 bitów + ACK

Na koniec transmitowany jest bajt danych, zaczynając od najważniejszego bitu, również potwierdzany bitem ACK.

Wizualizacja stanów logicznych linii CLK i DATA podczas wysyłania wartości 0x06 na adres 0x20 z częstotliwością 100kHz



Ramka danych w I2C



Przesył danych za pomocą magistrali I2C

```
#include <Wire.h>
void setup() {
    Wire.begin(); //Aktywacja magistrali
    Wire.beginTransmission(0x30); //Inicjalizacja komunikacji z podanym adresem
    Wire.write(data); // Zapisanie 1 bajta danych do bufora
    Wire.endTransmission(); // Wysłanie danych i zakończenie komunikacji
}
void loop() {}
```

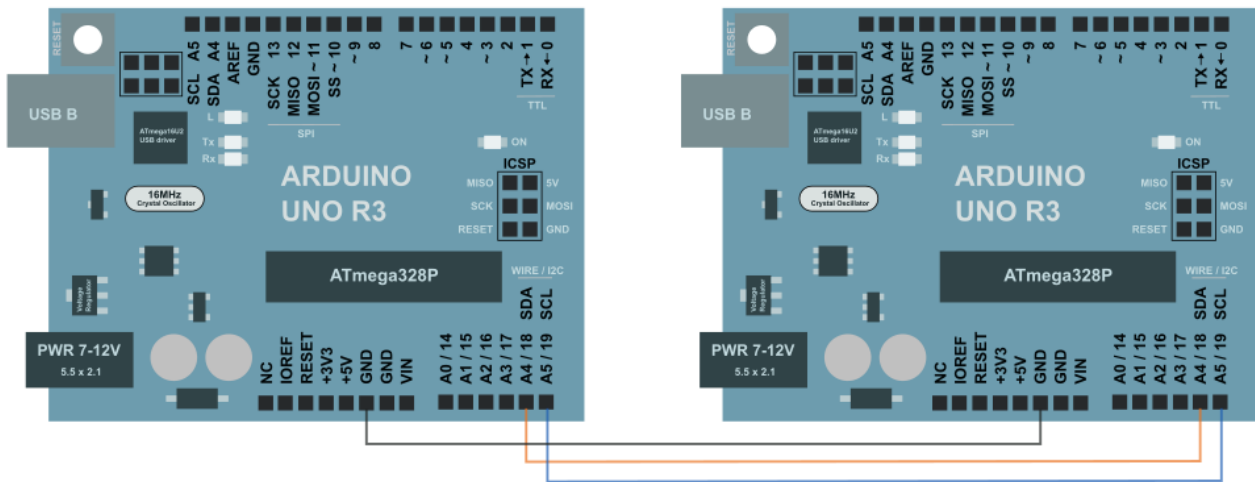
Dane odbiera się za pomocą eventu, który powinien być prosty i szybki w realizacji. W funkcji `receiveEvent` można pobierać ilość przesłanych bajtów.

```
Wire.onReceive(receiveEvent);
void receiveEvent(int howMany){
    myData = Wire.read();
    bool a = bitRead(myData, 0);
    digitalWrite(13, a);
}
```

Przykład przesłania tablicy danych:

```
#include <Wire.h>
void setup() {
    Wire.begin();
}
void loop() {
    Wire.beginTransmission(0x50);
    byte data[] = {1, 2, 3, 4, 5};
    Wire.write(data, sizeof(data));
    Wire.endTransmission();
    delay(1000);
}
```

Przesyłanie danych I2C pomiędzy 2 płytkami Arduino



//Master Arduino

```
#include <Wire.h>
const int SLAVE_ADDR = 0x20;
// 0x20 - 0b00100000
int counter = 0;

void setup() {
  delay(500);
}

void SendData(byte data){
  Wire.begin();
  Wire.beginTransmission(SLAVE_ADDR);
  Wire.write(data);
  Wire.endTransmission();
  delay(500);
}

void loop() {
  SendData(counter);
  counter++;
}
```

//Slave Arduino

```
#include <Wire.h>
const byte MY_ADDRESS = 0x20;
const int LED = 2;

void setup() {
  Wire.begin(MY_ADDRESS);
  pinMode(LED, OUTPUT);
  Wire.onReceive(receiveEvent);
  Serial.begin(9600);
}

void loop() {}

void receiveEvent() {
  while(Wire.available()){
    byte incData = Wire.read();
    Serial.println(incData);
    digitalWrite(LED, incData%2);
  }
}
```

Skaner urządzeń I2C

Funkcja `Wire.endTransmission()` zwraca kod statusu, który może być jednym z poniższych:

- 0: Sukces (success)
- 1: Przepełnienie bufora transmitera (data too long to fit in transmit buffer)
- 2: Adres NACK (received NACK on transmit of address)
- 3: Dane NACK (received NACK on transmit of data)
- 4: Inny błąd (other error)

```

#include <Wire.h>
#define WIRE Wire

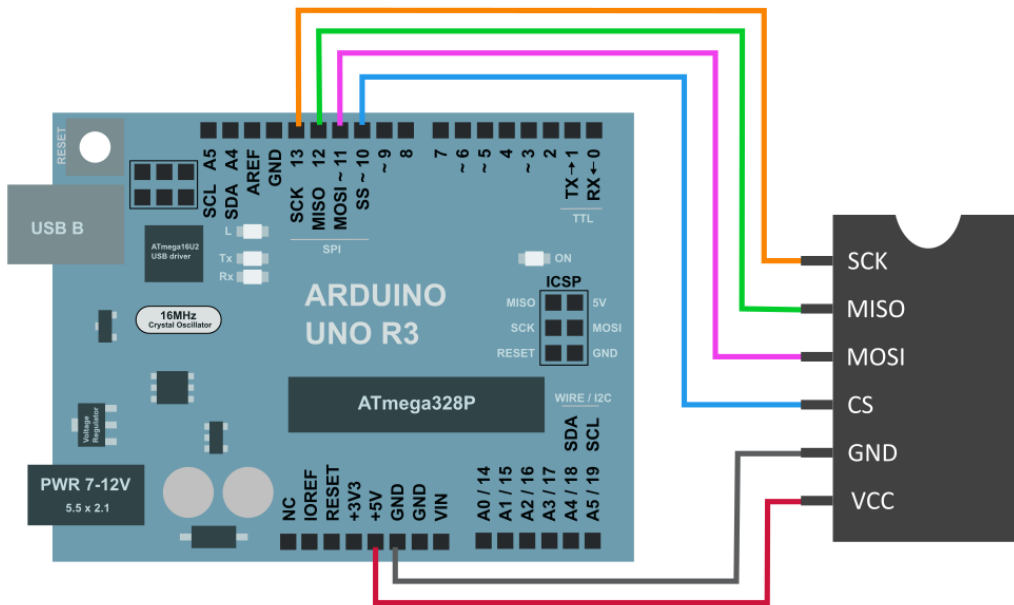
void setup() {
    WIRE.begin();
    Serial.begin(9600);
}

void loop() {
    byte error, address;
    int nDevices;
    Serial.println("Scanning...");
    nDevices = 0;
    for (address = 1; address < 127; address++) {
        WIRE.beginTransmission(address);
        error = WIRE.endTransmission();
        if (error == 0) {
            Serial.print("I2C device found at address 0x");
            if (address < 16)
                Serial.print("0");
            Serial.print(address, HEX);
            Serial.println(" !");
            nDevices++;
        } else if (error == 4) {
            Serial.print("Unknown error at address 0x");
            if (address < 16)
                Serial.print("0");
            Serial.println(address, HEX);
        }
    }
    if (nDevices == 0)
        Serial.println("No I2C devices found\n");
    else
        Serial.println("done\n");

    delay(5000); // wait 5 seconds for next scan
}

```

Magistrala SPI pozwala na jednoczesne wysyłanie i odbieranie danych z różnymi prędkościami. Arduino pełni rolę urządzenia nadrzędnego i może się komunikować z wieloma urządzeniami podrzędnymi w trybie Full Duplex.



Komunikacja SPI przebiega na podstawie 4 przewodów:

SCK (clock) – pin 13

MISO (main in sub out) – pin 12

MOSI (main out sub in) – pin 11

SS / CS (chip select) – pin 10 (domyślnie)

Linie MOSI, MISO i SCK są wspólne dla wszystkich urządzeń przyłączonych do magistrali SPI.

Każde urządzenie ma osobną linię SS, która decyduje o tym, z którym urządzeniem następuje przesył danych w danym momencie.

Domyślnie na pinie SS utrzymywany jest stan HIGH, który uniemożliwia przesył danych. Przełączenie stanu na LOW jest sygnałem do przesyłu danych do danego urządzenia. Po transmisji danych stan pinu ponownie ustawiany jest na HIGH, co pozwala zwolnić magistralę dla innych urządzeń.

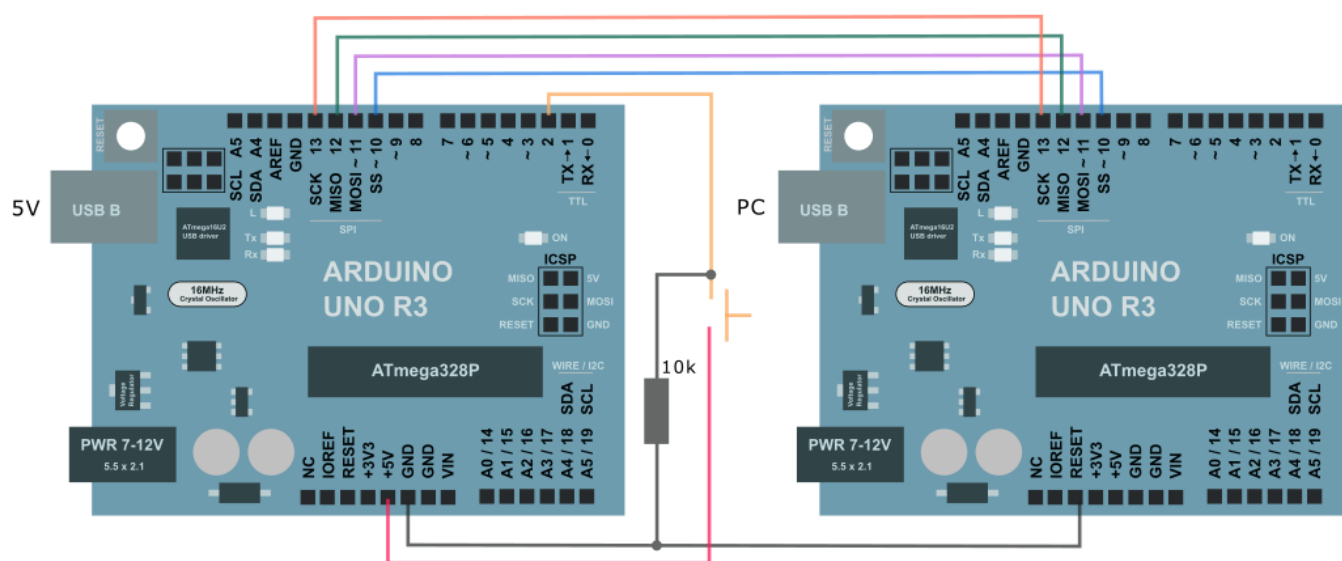
Przesył danych za pomocą magistrali SPI

Komunikacja SPI jest możliwy za pomocą biblioteki SPI.

```
#include "SPI.h"

pinMode(SS, OUTPUT);
digitalWrite(SS, HIGH);
SPI.begin(); //rozpoczęcie komunikacji SPI
SPI.beginTransaction(SPISettings(clockSpeed, bitOrder, dataMode));
    //ustawienia komunikacji
    // clockSpeed: Prędkość zegara SPI, np. 500000 dla 500 kHz.
    // bitOrder: Kolejność przesyłania bitów. MSBFIRST lub LSBFIRST
    // dataMode: Tryb SPI (kombinacja CPOL i CPHA). Dostępne tryby to:
    // SPI_MODE0: CPOL = 0, CPHA = 0,
    // SPI_MODE1: CPOL = 0, CPHA = 1,
    // SPI_MODE2: CPOL = 1, CPHA = 0,
    // SPI_MODE3: CPOL = 1, CPHA = 1.
digitalWrite(SS, LOW);
SPI.transfer(byte);          //przesyłanie danych, linia SS w stanie LOW
digitalWrite(SS, HIGH);
SPI.endTransaction();        // zakończenie transakcji, zwolnienie magistrali dla
innych urządzeń
```

Komunikacja SPI między dwoma płytkami Arduino



```

//MASTER ARDUINO

#include<SPI.h>
int counter = 0;
int btnPin = 2;
bool btnPrevState = false;

void setup(){
    Serial.begin(9600);
    SPI.begin();
    SPI.beginTransaction(SPISettings(125000, MSBFIRST , SPI_MODE0));
    pinMode(SS, OUTPUT);
}

void onClick() {
    bool btnNowState = digitalRead(btnPin);
    if (btnNowState == 1 && btnPrevState == 0) {
        digitalWrite(SS, LOW);
        SPI.transfer(counter);
        digitalWrite(SS, HIGH);
        SPI.endTransaction();
        counter++;
    }
    btnPrevState = btnNowState;
}

void loop(){
    onClick();
}

```

```

//SLAVE ARDUINO

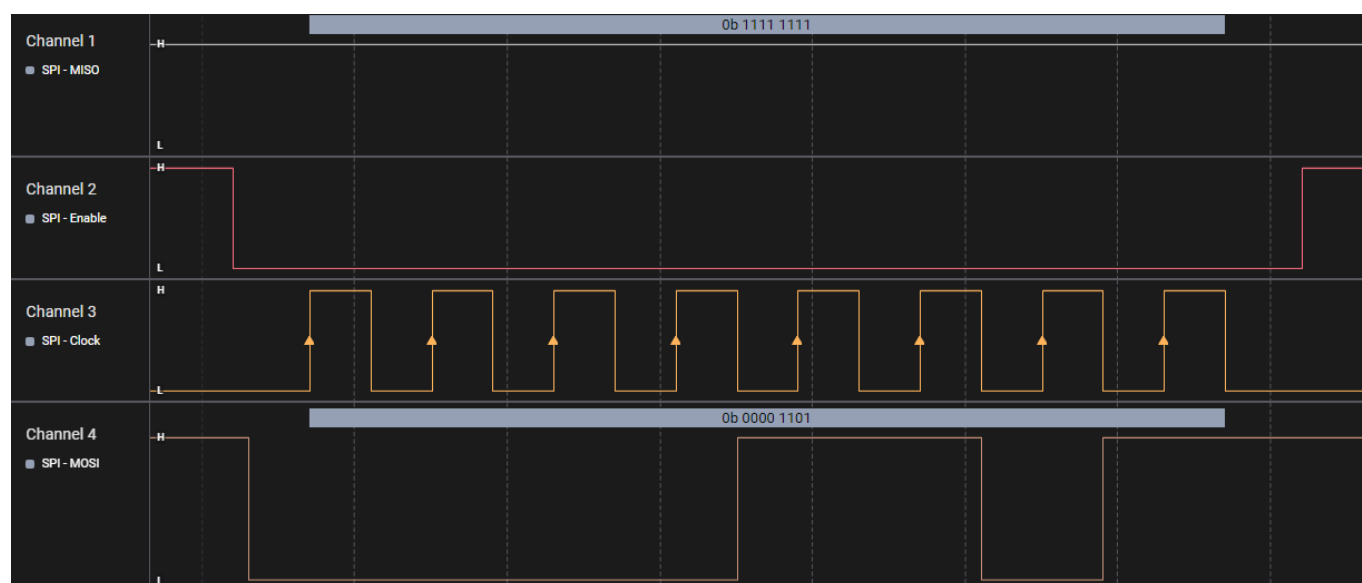
#include<SPI.h>
volatile bool flag1 = false;

void setup(){
    Serial.begin(9600);
    pinMode(MOSI, OUTPUT);    //should be made output to send data to Master
    SPCR |= _BV(SPE);        //SPI Port is enabled or SPCR |= (1 << SPE);
    SPI.attachInterrupt();
    pinMode(SS, INPUT_PULLUP);
}

void loop(){
    if (flag1 == true){
        Serial.println(SPDR);
        flag1 = false;
    }
}

ISR (SPI_STC_vect){
    flag1 = true;
}

```



Przesyłanie danych tekstowych SPI

```
//master
#include<SPI.h>
char txMsg[] = "Arduino!";
char rxMasg[20] = "";

void setup(){
    Serial.begin(9600);
    SPI.begin();
    SPI.setClockDivider(SPI_CLOCK_DIV128); //16 MHz/128
    pinMode(SS, OUTPUT);    //SS is DPin-10
    digitalWrite(SS, LOW);  //Slave is selected
}

void loop(){
    SPI.transfer('<'); //start mark
    for(int i=0; i< sizeof(txMsg); i++) {
        SPI.transfer(txMsg[i]); //SPI transfer is byte-by-byte
    }
    SPI.transfer('>'); //end mark
    delay(1000);      //test interval
}
```

```

//slave
#include<SPI.h>
char rxMsg[20] = "";
volatile bool flag1 = false;
bool flag2 = false;
int i = 0;

void setup(){
  Serial.begin(9600);
  pinMode(SS, INPUT_PULLUP); //SPI.h helps to get the meaning of SS
  pinMode(MOSI, OUTPUT);      //should be made output to send data to Master
  SPCR |= _BV(SPE);           //SPI Port is enabled
  SPI.attachInterrupt();      //SPI interrupt is enabled
}
void loop(){
  if (flag1 == true) { //SPDR has data
    if (flag2 == false) //start mark of message not yet received
    {
      char q = SPDR;
      if (q == '<')
      {
        flag2 = true; //start mark is detected
        flag1 = false; //must be made false for the next cycle
      }
    }
  }
  else {
    if (SPDR == '>') //checking if end mark has arrived
    {
      flag1 = false;
      flag2 = false;
      i = 0;
      Serial.print(rxMsg); //end mark has arrived; show the received message
      Serial.println();    //insert newline
    }
    else
    {
      rxMsg[i] = SPDR; //save received character/data byte in array
      i++;             //adjust array pointer
      flag1 = false;   //must be false for next cycle
    }
  }
}
}
}

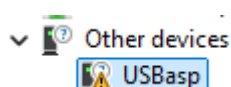
```

```
ISR (SPI_STC_vect) { //MCU comes here when there is a data byte in SPDR
    flag1 = true;    //flag to indicate there is data in the SPDR register
}
```

Wgrywanie programu za pomocą programatora ISP USBasp

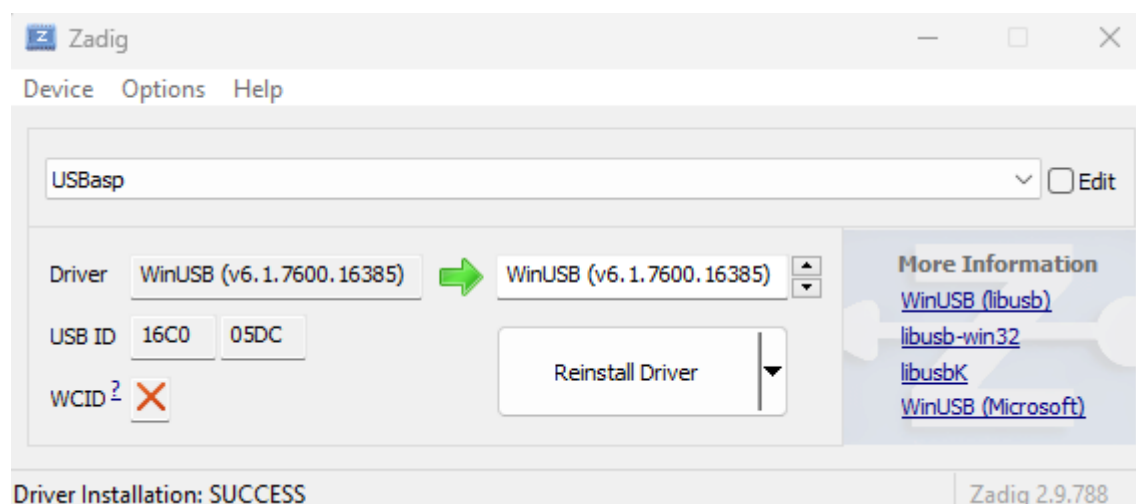
ISP - In System Programmer

ICSP - In Circuit Serial Programmer



Pobierz i zainstaluj sterowniki do programatora za pomocą programu Zadig

<https://zadig.akeo.ie/>



Wybierz płytkę, bez określania portu USB

W menu Tools / Programmer wybierz USBasp

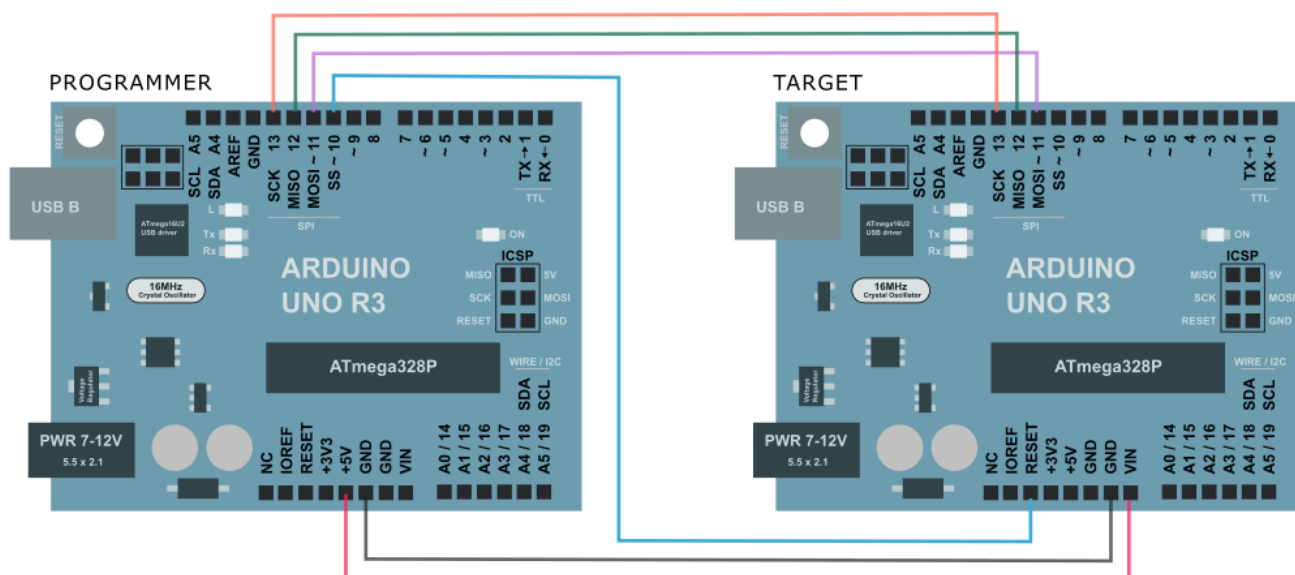
Wgraj sketch za pomocą menu Sketch / Upload Using Programmer

Wgrywanie programów za pomocą ISP usuwa bootloader, co uniemożliwi późniejsze wgrywanie programów za pomocą USB.

Bootloader można wgrać za pomocą ISP w menu Tools / Burn Bootloader

Wgrywanie programu za pomocą innej płytki Arduino ISP

1. Połącz 2 płytki w konfiguracji SPI (z tą różnicą, że pin 10 z programatora idzie do pinu Reset na urządzeniu docelowym).
2. Uruchom sketch ArduinoISP.ino z File / Examples
3. Wgraj sketch na działającą płytkę, dzięki czemu będzie działać jako programator
4. Uruchom sketch, który chcesz wgrać na płytkę docelową
5. W menu Tools / Programmer wybierz Arduino as ISP
6. Wgraj sketch za pomocą menu Sketch / Upload Using Programmer



Wgrywanie programów za pomocą ISP usuwa bootloader co uniemożliwi późniejsze wgrywanie programów za pomocą USB.

Komunikacja radiowa SPI NRF24L01

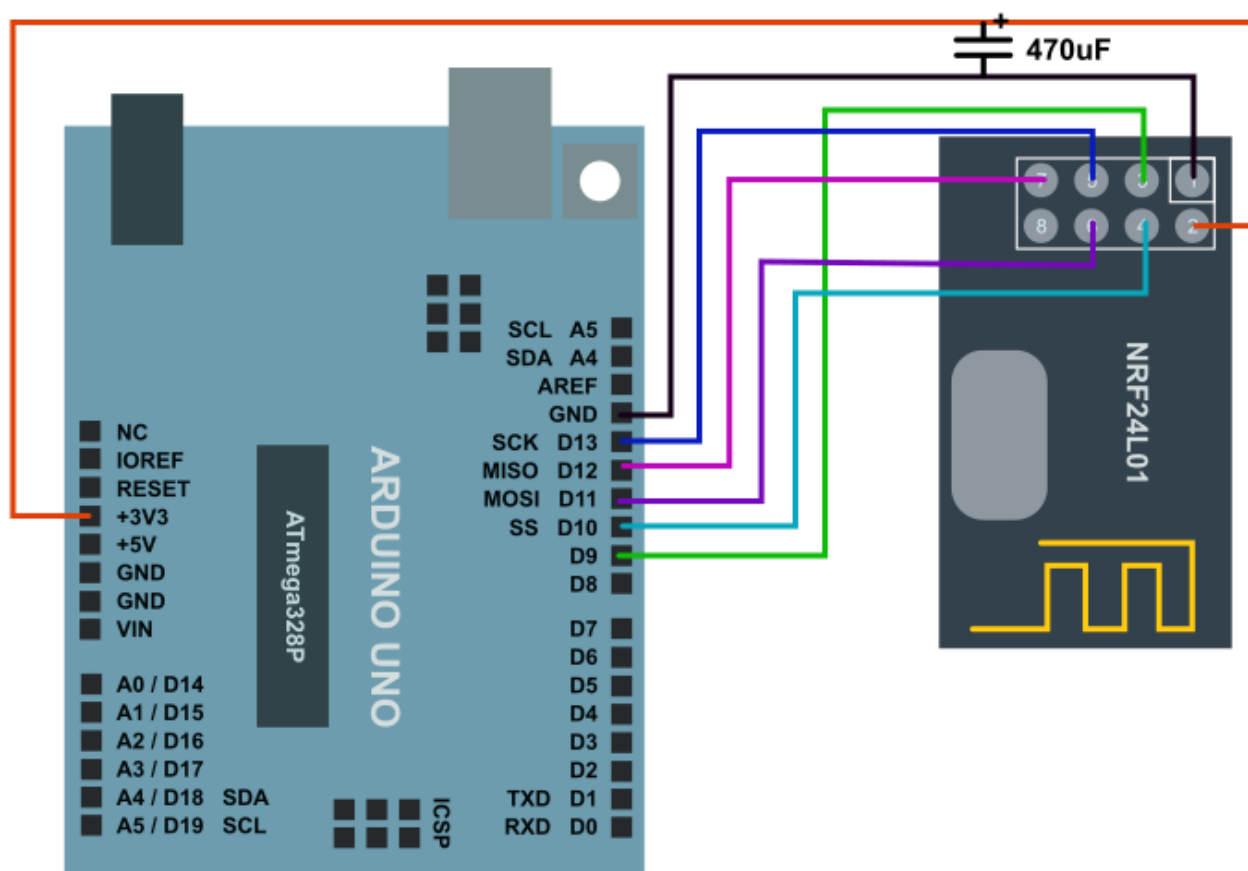
Napięcie zasilania : 1,9V - 3,6V (Pozostałe piny tolerują 5V)

Pobór prądu : 11mA

Częstotliwość pracy : 2,4GHz

Prędkość transmisji 250kbps, 1Mbps lub 2Mbps

Dystans: 100m



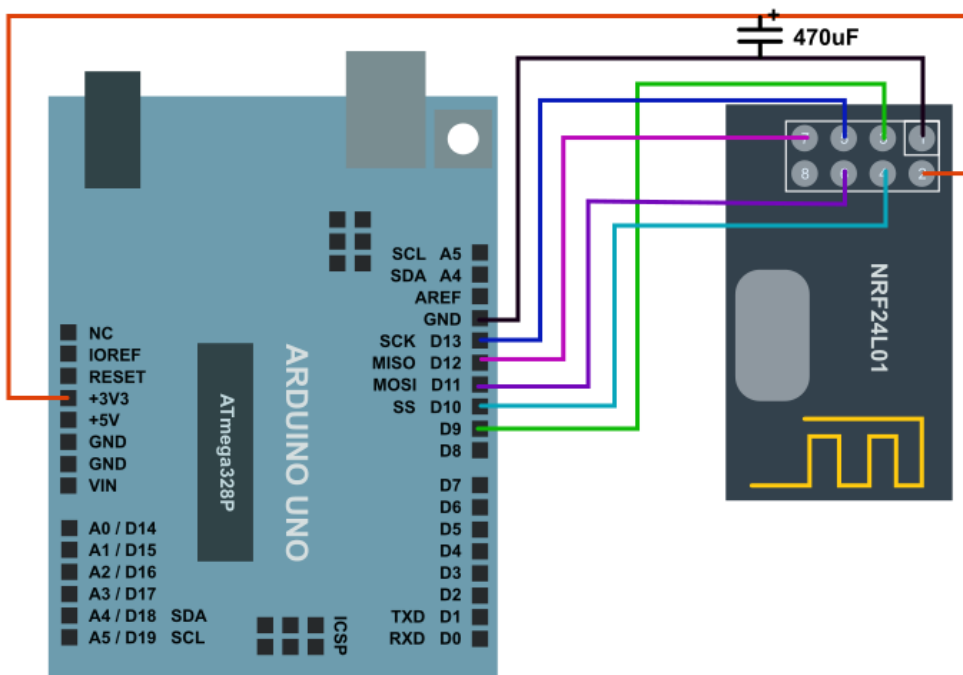
Przesyłanie danych NRF24L01 0 i 1 do przełączania LED

```
//nadajnik
#include <SPI.h>
#include "RF24.h"
RF24 radio(9, 10);
const byte adres[6] = "00001";
int msg[1];
bool isActive = true;
const int countDelay = 1000;
unsigned long previousMillis = 0;

void setup(void) {
  Serial.begin(9600);
  radio.begin();
  radio.openWritingPipe(adres);
  radio.stopListening();
}

void executeEvery1() {
  if (millis() - previousMillis >= countDelay) {
    previousMillis = millis();
    msg[0] = isActive ? 1 : 0;
    radio.write(msg, 1);
    Serial.println(msg[0]);
    isActive = !isActive;
  }
}

void loop() {
  executeEvery1();
}
```

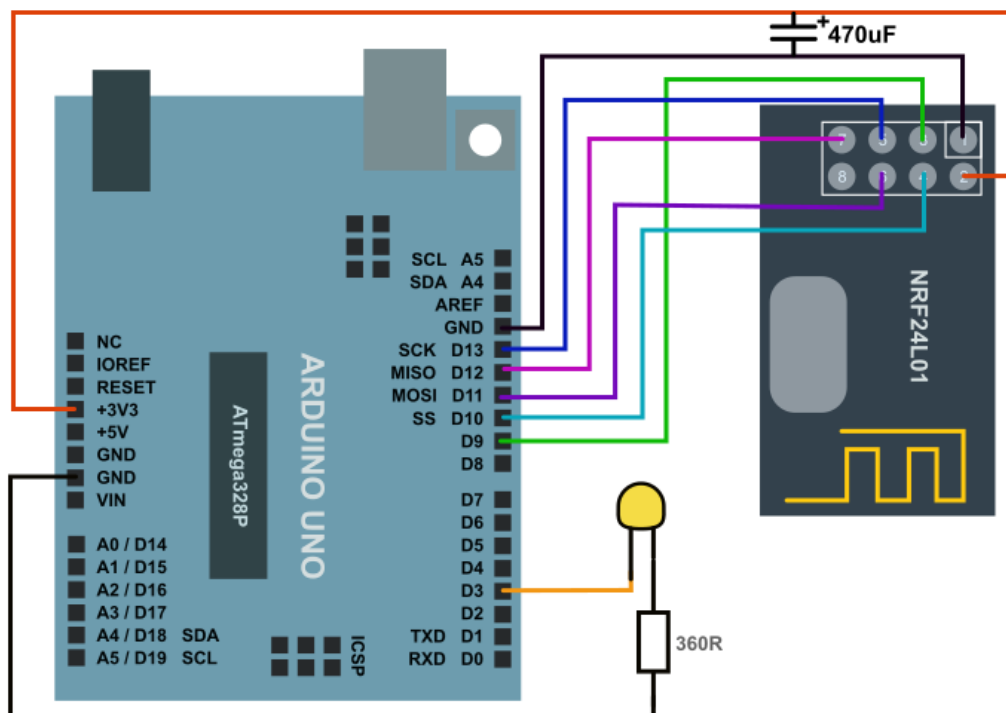


```

//odbiornik
#include <SPI.h>
#include "RF24.h"
RF24 radio(9,10);
const int LED_PIN = 3;
const byte adres[6] = "00001";
int msg[1];

void setup(){
  pinMode(LED_PIN, OUTPUT);
  radio.begin();
  radio.openReadingPipe(0,adres);
  radio.startListening();
}
void loop(){
  if (radio.available()){
    radio.read(msg, 1);
    digitalWrite(LED_PIN, msg[0]);
  }
}

```



Przesyłanie tekstu NRF24L01 i wyświetlanie na I2C Oled

```
//nadajnik
#include <SPI.h>
#include <RF24.h>
RF24 radio(9, 10);
const byte adres[6] = "00001";

void setup() {
  Serial.begin(9600);
  radio.begin();
  radio.openWritingPipe(adres);
  radio.setPALevel(RF24_PA_HIGH); // Moc sygnału
  radio.stopListening(); // Tryb nadawania
}

void loop() {
  if (Serial.available() > 0) {
    String dane = Serial.readString();
    Serial.println("Wysyłam: " + dane);
    char dane_do_wyslania[32];
    dane.toCharArray(dane_do_wyslania, 32);
    radio.write(&dane_do_wyslania, sizeof(dane_do_wyslania));
  }
}
```

```
//odbiornik
// do odbiornika dołączony jest wyświetlacz i2c oled1306
#include <SPI.h>
#include <RF24.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

RF24 radio(9, 10); // CE, CSN
const byte adres[6] = "00001"; // Ustalony adres komunikacji

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1 // Reset nie jest podłączony, używamy -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
```

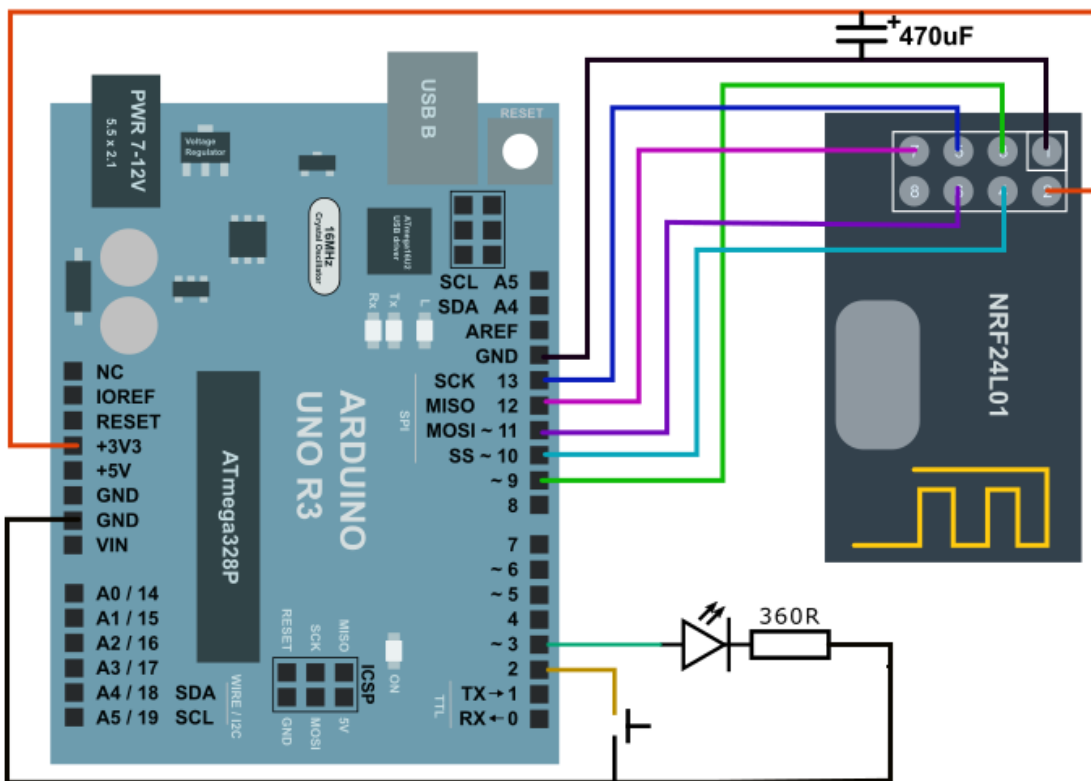
```

void setup() {
    Serial.begin(9600);
    radio.begin();
    radio.openReadingPipe(0, adres);
    radio.setPALevel(RF24_PA_HIGH);
    radio.startListening();
    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Adres 0x3C dla OLED I2C
        Serial.println(F("Nie można zainicjalizować wyświetlacza OLED"));
        for(;;);
    }
    display.clearDisplay();
    display.display();
    display.setTextSize(1);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0, 0);
    display.println("Oczekiwanie na dane...");
    display.display();
}

void loop() {
    if (radio.available()) {
        char odbior_danych[32] = "";
        radio.read(&odbior_danych, sizeof(odbior_danych));
        Serial.println("Otrzymano: " + String(odbior_danych));
        display.clearDisplay();
        display.setCursor(0, 0);
        display.setTextSize(2);
        display.setTextColor(SSD1306_WHITE);
        display.println(odbior_danych);
        display.display();
    }
    delay(20);
}

```

Komunikacja NRF24L01 w 2 kierunkach



```
//NRF24 nr1 i nr2
#include <SPI.h>
#include <RF24.h>
const int LED = 3;
const int BTN = 2;
RF24 radio(9, 10);
const byte addresses[][6] = {"00001", "00002"};
bool inData = 0;
bool btnPrevState = false;
bool state = 0;

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(BTN, INPUT_PULLUP);
  radio.begin();
  radio.openWritingPipe(addresses[0]); //zamiana 0 na 1 dla 2 urządzenia
  radio.openReadingPipe(1, addresses[1]); //zamiana 1 na 0 dla 2 urządzenia
  radio.setPALevel(RF24_PA_HIGH);
}
```

```

void onClick() {
    bool btnNowState = digitalRead(BTN);
    if (btnNowState == 1 && btnPrevState == 0) {
        state = !state;
        radio.stopListening();
        radio.write(&state, sizeof(state));
    }
    btnPrevState = btnNowState;
}

void loop() {
    delay(10);
    radio.startListening();
    if(radio.available()){
        radio.read(&inData, sizeof(inData)); // Przekazanie wskaźnika na zmienną
        digitalWrite(LED, inData);
    }
    onClick();
}

```

Wysyłanie pakietu danych (struct)

Nadajnik

```
#include <SPI.h>
#include <RF24.h>
RF24 radio(9, 10);
const byte address[6] = "00001";
int counter = 0;
float temp = 0.0;

// Max size of this struct is 32 bytes - NRF24L01 buffer limit
struct Data_Package {
    int a;
    float b;
};
Data_Package data = {0, 0.0};

void setup() {
    radio.begin();
    radio.openWritingPipe(address);
    radio.setPALevel(RF24_PA_MIN);
    radio.stopListening();
}

void loop() {
    radio.write(&data, sizeof(Data_Package));
    counter += 1;
    data.a = counter;
    temp += 0.1;
    data.b = temp;
    delay(500);
}
```

```
#include <SPI.h>
#include <RF24.h>
RF24 radio(9, 10);
const byte address[6] = "00001";

struct Data_Package {
    int a;
    float b;
};
Data_Package data;

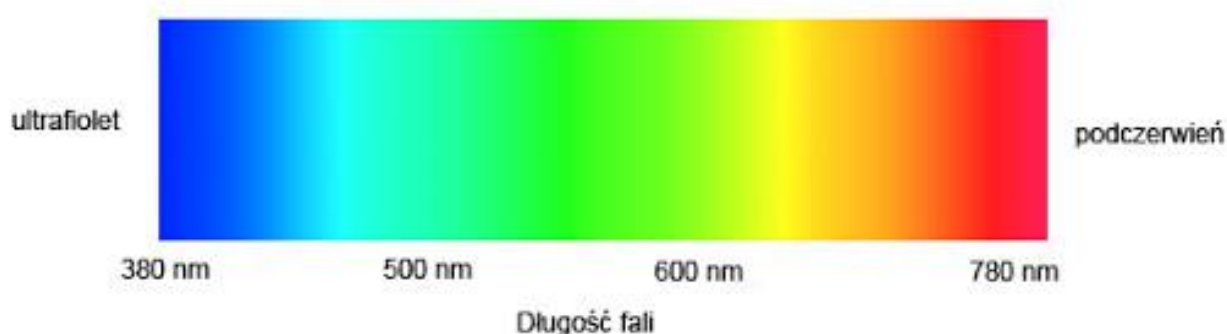
void setup() {
    Serial.begin(9600);
    radio.begin();
    radio.openReadingPipe(0, address);
    radio.setPALevel(RF24_PA_MIN);
    radio.startListening();
}

void loop() {
    if (radio.available()) {
        radio.read(&data, sizeof(Data_Package));
    }
    Serial.print("a: ");
    Serial.print(data.a);
    Serial.print(" b: ");
    Serial.println(data.b);
}
```

Promieniowanie podczerwone jest wszędzie wokół nas, dociera ze słońca, a także każdy przedmiot o temperaturze powyżej zera absolutnego (0K lub -273,15C) promieniuje fale podczerwone.

Światło podczerwone nie jest widzialne gołym okiem, ale można je odczuć w postaci ciepła.

Im dłuższa fala, tym mniejsza energia.

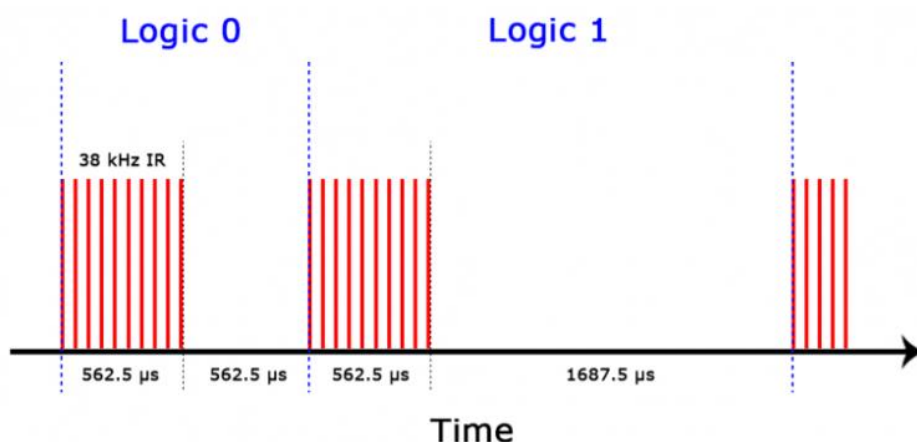


Przesył danych IR przebiega na podstawie modulowanych sygnałów świetlnych z częstotliwością 38k razy na sekundę (38kHz), co pozwala na filtrowanie zakłóceń świetlnych z otoczenia.

Dane zawierają m.in. informacje o protokole, adres urządzenia oraz kod instrukcji, np.

`Protocol=NEC Address=0xEF00 Command=0xA Raw-Data=0xF50AEF00`

Dla protokołu NEC stan niski i wysoki wykrywany jest za pomocą długości modulowanych impulsów.



Odbiornik podczerwieni to fotodioda ze wzmacniaczem (preamplifier), który zamienia otrzymywane światło podczerwone z częstotliwością 38kHz na sygnał elektryczny (bity).

Odbiornik IR pracuje w trybie aktywnego niskiego sygnału, co oznacza, że linia domyślnie utrzymuje stan wysoki, a podczas odbierania danych, bity będą odbierane w stanie niskim.

Początek i koniec transmisji wyznaczają stany niskie o czasie 10ms.

Na oscyloskopie lub analizatorze logicznym, nie widać impulsów 38kHz, ponieważ odbiornik zamienia je na bity danych.

Logiczne "0" to impuls ok. 700 μ s + przerwa ok. 480 μ s.

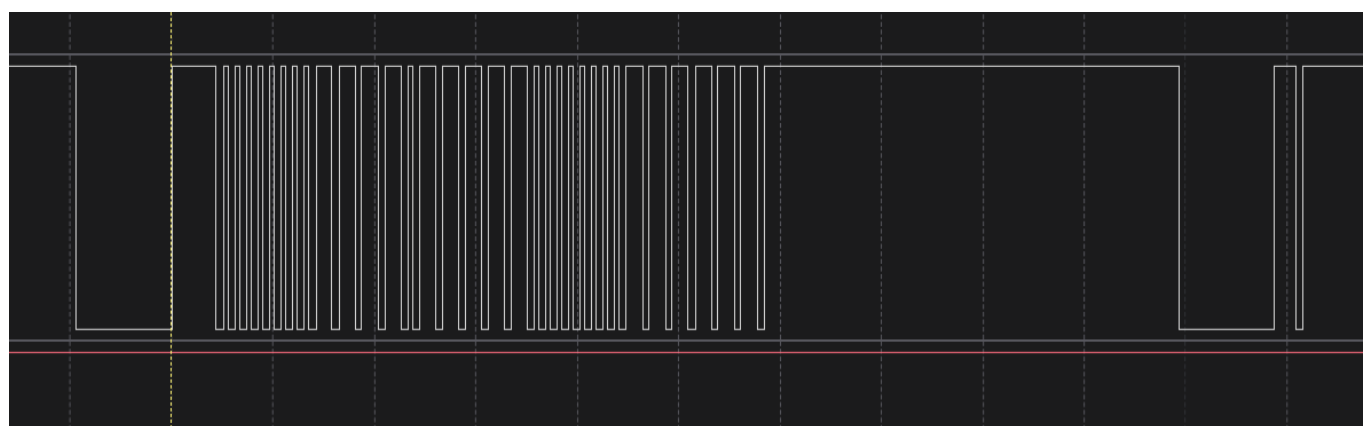
Logiczne "1" to impuls ok. 700 μ s + dłuższa przerwa ok. 1560 μ s.



Protocol=NEC Address=0xEF00 Command=0x3 Raw-Data=0xFC03EF00 (LSB first)

0xFC03EF00 = 0b1111 1100 0000 0011 1110 1111 0000 0000

LSB: 0000 0000 1111 0111 1100 0000 0011 1111



Biblioteka IRremote do komunikacji IR z Arduino

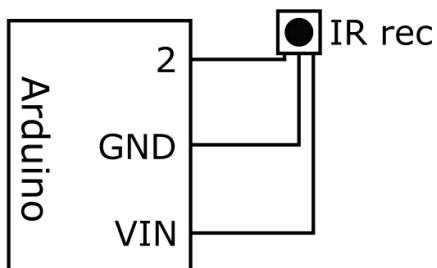
IRremote by shirriff, z3t0, ArminJo

4.4.1 installed

Send and receive infrared signals with multiple protocols. Currently included protocols: Denon / Sharp, JVC, LG / ...
[More info](#)

4.4.1 ▼ **REMOVE**

Schemat połączenia odbiornika IR z Arduino



Wyświetlenie informacji z pilota IR

```
#include <IRremote.hpp>
const int IR_RECEIVE_PIN = 2;

void setup() {
  Serial.begin(9600);
  IrReceiver.begin(IR_RECEIVE_PIN, DISABLE_LED_FEEDBACK);
}

void loop() {
  // Sprawdź czy odbiornik odebrał sygnał
  if (IrReceiver.decode()) {
    // Wyświetl kod HEX wciśniętego przycisku
    Serial.println(IrReceiver.decodedIRData.decodedRawData, HEX);
    IrReceiver.printIRResultShort(&Serial); // Drukuje kompletne dane
    IrReceiver.resume(); // Zresetuj odbiornik do odebrania nowego sygnału
  }
}
```

Przycisk 1 włącza LED, przycisk 0 wyłącza

```
#include <IRremote.hpp>
const int IR_RECEIVE_PIN = 2;
int on = 3, off = 2;

void setup() {
  Serial.begin(9600);
  IrReceiver.begin(IR_RECEIVE_PIN, DISABLE_LED_FEEDBACK);
  pinMode(13, OUTPUT);
}

void loop() {
  if (IrReceiver.decode()) {
    Serial.println(IrReceiver.decodedIRData.command);
    IrReceiver.printIRResultShort(&Serial); // Drukuje kompletne dane
    IrReceiver.resume();                    // Zresetuj odbiornik do odebrania
nowego sygnału
    if (IrReceiver.decodedIRData.command == on) {
      digitalWrite(13, HIGH);
    } else if (IrReceiver.decodedIRData.command == off) {
      digitalWrite(13, LOW);
    }
  }
}
```

```
#define IR_RECEIVE_PIN 2
#define IR_SEND_PIN 3
#include <IRremote.hpp> // include the library
const int btnUp = 4;      // vol up
const int btnDown = 5;    // vol down
uint32_t sCommand[] = { 0xF8070707, 0xF40B0707 };

void setup() {
    IrSender.begin();
    disableLEDFeedback();
    pinMode(btnUp, INPUT_PULLUP);
    pinMode(btnDown, INPUT_PULLUP);
}

void SendCode(uint32_t cmd) {
    uint8_t sRepeats = 0;
    IrSender.sendNECRaw(cmd, sRepeats);
    sRepeats++;
    if (sRepeats > 3) {
        sRepeats = 3;
    }
    delay(100);
}

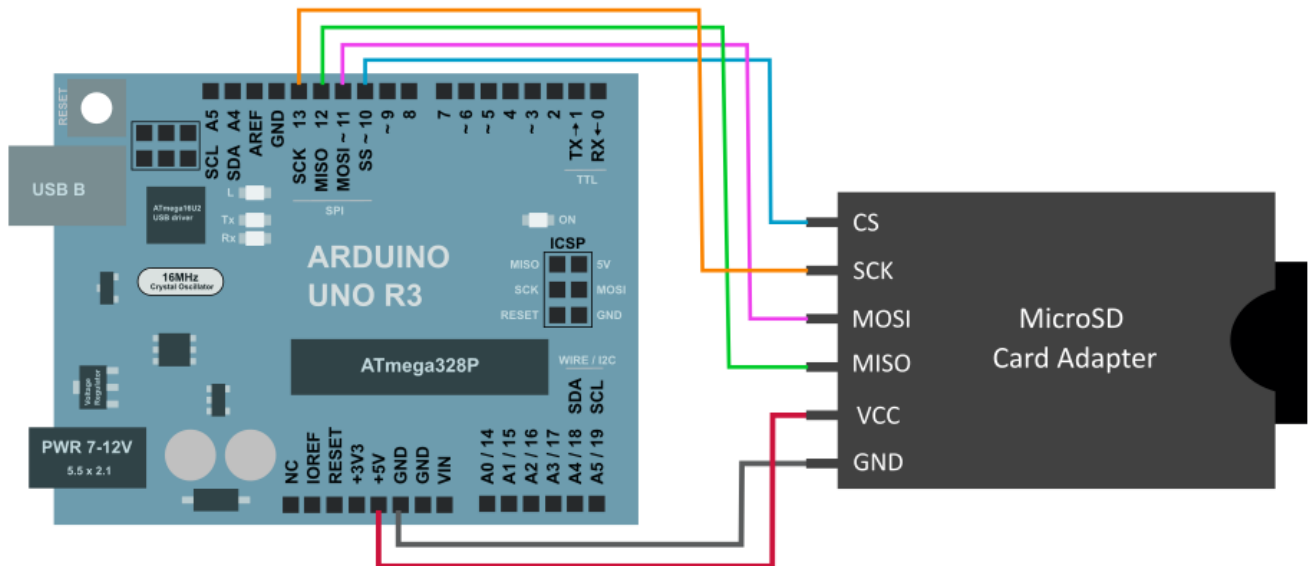
void loop() {
    if (digitalRead(btnUp) == LOW) {
        SendCode(sCommand[0]);
    }
    if (digitalRead(btnDown) == LOW) {
        SendCode(sCommand[1]);
    }
}
```

Moduł do obsługi kart SD komunikują się przez magistralę SPI.



Karty SD powinny mieć rozmiar mniejszy niż 2GB, sformatowane do systemu plików FAT32 (lub FAT).

Istnieje kilka różnych bibliotek do obsługi kart SD, w poniższym przykładzie posłużymy się biblioteką SD.h i SPI.h



Zapisywanie i odczytywanie informacji z pliku txt na karcie SD

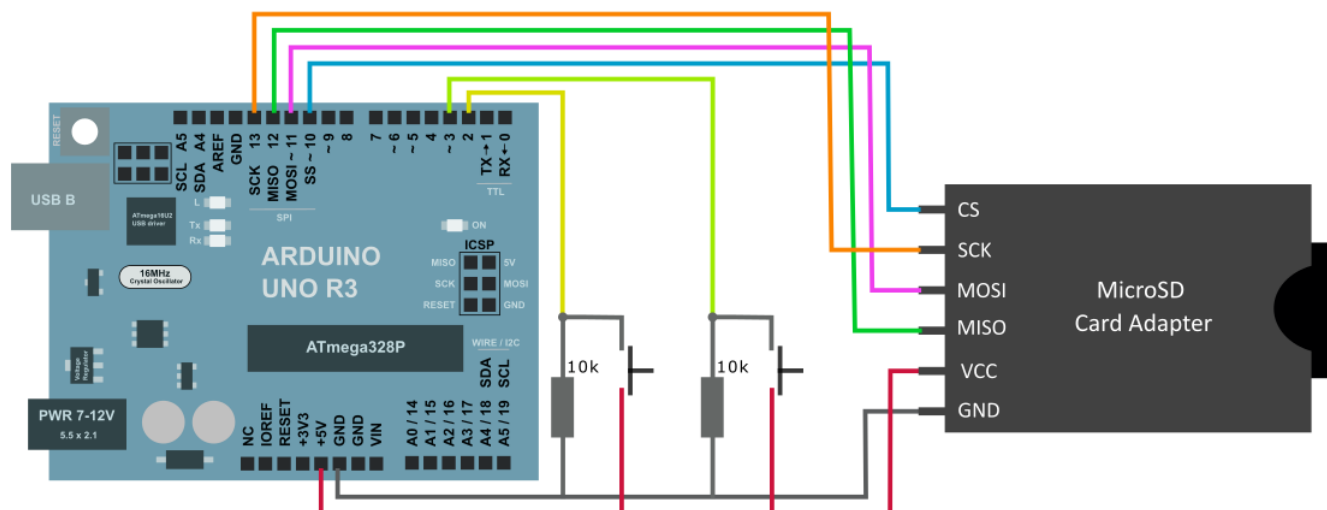
```
#include <SPI.h>
#include <SD.h>
File myFile;
const char* fileName = "test5.txt";

void setup() {
    Serial.begin(9600);
    Serial.print("Initializing SD card...");
    if (!SD.begin(10)) {
        Serial.println("initialization failed!");
        while (1)
            ;
    }
    Serial.println("initialization done.");
    myFile = SD.open(fileName, FILE_WRITE);
    if (myFile) {
        Serial.println("Writing to file ...");
        myFile.println("testing 1, 2, 3.");
        myFile.close();
    } else {
        Serial.println("error opening file for writing");
    }

    myFile = SD.open(fileName);
    if (myFile) {
        while (myFile.available()) {
            Serial.write(myFile.read());
        }
        myFile.close();
    } else {
        Serial.println("error opening file for reading");
    }
}

void loop() {}
```

Zapisywanie liczby kliknięć na karcie SD



```

#include <SPI.h>
#include <SD.h>
const int chipSelect = 10;
const int btnPin = 2;
const int resetBtn = 3;
File myFile;
const char* fileName = "test10.txt";
int counter = 0;
bool btnPrevState = false;
bool btnPrevReset = false;

void setup() {
    Serial.begin(9600);
    Serial.print("Initializing SD card...");
    if (!SD.begin(chipSelect)) {
        Serial.println("initialization failed!");
        while (1)
            ;
    }
    Serial.println("initialization done.");
    if(SD.exists(fileName)){
        counter = load();
    }
    Serial.println(counter);
}

void save(int data){
    myFile = SD.open(fileName, O_TRUNC | FILE_WRITE);
    if (myFile) {
        myFile.println(data);
        myFile.close();
    } else {
        Serial.println("error opening file for saving");
    }
}

```

```

int load(){
  myFile = SD.open(fileName, O_READ);
  if (myFile) {
    String numberStr = "";
    while (myFile.available()) {
      char c = myFile.read();
      if (isdigit(c)) {
        numberStr += c;
      }
    }
    myFile.close();
    if (numberStr.length() > 0) {
      return numberStr.toInt();
    } else {
      Serial.println("No valid number found in file.");
      return 0;
    }
  } else {
    Serial.println("Error opening file for reading.");
    return 0;
  }
}

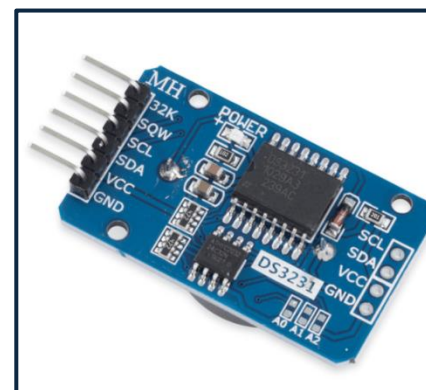
void onClickCount() {
  bool btnNowState = digitalRead(btnPin);
  if (btnNowState == 1 && btnPrevState == 0) {
    counter++;
    save(counter);
    Serial.println(counter);
  }
  btnPrevState = btnNowState;
}

void onClickReset() {
  bool btnNowState = digitalRead(resetBtn);
  if (btnNowState == 1 && btnPrevReset == 0) {
    counter = 0;
    save(counter);
    Serial.println(counter);
  }
  btnPrevReset = btnNowState;
}

void loop() {
  onClickCount();
  onClickReset();
}

```

Moduł DS3231 to precyzyjny RTC (Real-Time Clock – zegar czasu rzeczywistego), który pozwala na śledzenie czasu nawet po odłączeniu zasilania, dzięki wbudowanej baterii.



DS3231 zawiera oscylator kwarcowy z kompensacją temperaturową, co minimalizuje wpływ wahań temperatury na dokładność czasu.

Oscylator pracuje z częstotliwością 32.768 kHz, co zapewnia wysoką precyzję (± 2 ppm (parts per million), co oznacza odchylenie rzędu kilku minut w ciągu roku).

Typowy prąd zużywany przez DS3231 to około 3.5 μ A w trybie podtrzymywania baterijnego, co pozwala na bardzo długą żywotność baterii.

Wyjście SQR pozwala generować przebieg fali prostokątnej o częstotliwościach 1Hz, 4kHz, 8kHz i 32kHz.

Wyjście 32kHz to bezpośredni sygnał z TCXO.

Rejestry DS3231

Domyślny adres DS3231 to 0x68 (7-bit address)

DS3231 przechowuje czas w rejestrach, które zawierają wartości w formacie BCD (Binary Coded Decimal). Aby odczytać lub zapisać czas, musisz konwertować liczby pomiędzy formatem dziesiętnym, a BCD.

Format BCD zapisuje każdą cyfrę dziesiętną w postaci binarnej (4 bity na cyfrę), czyli 4 najbardziej znaczące bity wyznaczają część dziesiętną liczby, a 4 najmniej znaczące wyznaczają rząd jedności, np. liczba 45 w zapisie binarnym BCD to 0100 0101

Rejestry przechowywania czasu

Sekundy: [0x00]

Minuty: [0x01]

Godziny: [0x02]

Dzień tygodnia: [0x03]

Dzień miesiąca: [0x04]

Miesiąc: [0x05]

Rok: [0x06]

// Chcąc ustawić godzinę 14:56:13 należy użyć poniższego kodu:

```
#include <Wire.h>
#define DS3231_ADDRESS 0x68 // Adres I2C modułu DS3231

uint8_t decToBcd(uint8_t val) {
    return ((val / 10) << 4) | (val % 10);
}

void setTime(uint8_t hour, uint8_t min, uint8_t sec) {
    uint8_t hoursBCD = decToBcd(hour);
    uint8_t minBCD = decToBcd(min);
    uint8_t secBCD = decToBcd(sec);
    Wire.beginTransmission(0x68);
    Wire.write(0x00);
    Wire.write(secBCD);
    Wire.write(minBCD);
    Wire.write(hoursBCD);
    Wire.endTransmission();
}

void setup() {
    Serial.begin(9600);
    Wire.begin();
    Wire.beginTransmission(DS3231_ADDRESS);
    if (Wire.endTransmission() != 0) {
        Serial.println("Brak połączenia z modułem DS3231!");
        while (1);
    }
    setTime(14,56,13);
}

void loop() {
    delay(1000); // Poczekać 1 sekundę
}
```

// Pobieranie czasu z DS3231

```
#include <Wire.h>
#define DS3231_ADDRESS 0x68 // Adres I2C modułu DS3231

uint8_t bcdToDec(uint8_t val) {
    return (val >> 4) * 10 + (val & 0b00001111);
}

void getDateTime(uint8_t &sec, uint8_t &min, uint8_t &hour) {
    Wire.beginTransmission(DS3231_ADDRESS);
    Wire.write(0x00); // Adres początkowy rejestru
    Wire.endTransmission();
    Wire.requestFrom(DS3231_ADDRESS, 3);
    sec = bcdToDec(Wire.read()); // Sekundy
    min = bcdToDec(Wire.read()); // Minuty
    hour = bcdToDec(Wire.read()); // Godziny
}

void setup() {
    Serial.begin(9600);
    Wire.begin();
    Wire.beginTransmission(DS3231_ADDRESS);
    if (Wire.endTransmission() != 0) {
        Serial.println("Brak połączenia z modułem DS3231!");
        while (1);
    }
}

void loop() {
    uint8_t sec, min, hour;
    getDateTime(sec, min, hour);
    Serial.print("Godzina: ");
    Serial.print(hour);
    Serial.print(':');
    Serial.print(min);
    Serial.print(':');
    Serial.println(sec);
    delay(1000); // Poczekać 1 sekundę
}
```

Alarmy

DS3231 ma dwa niezależne alarmy, które mogą być skonfigurowane do wyzwalania o określonej godzinie, minucie, dniu tygodnia, itp. Alarmy mogą być wykorzystywane do budzenia mikrokontrolera lub wyzwalania zdarzeń w określonych interwałach czasu.

Pomiar temperatury

DS3231 ma wbudowany czujnik temperatury, który mierzy temperaturę wewnętrzną co 64 sekundy. Wynik tego pomiaru jest wykorzystywany do kompensacji temperaturowej oscylatora, co zapewnia stabilność zegara.

Temperatura w DS3231 jest przechowywana w dwóch rejestrach:

[0x11] (MSB) – wyższy bajt (8 bitów) temperatury, który zawiera całkowitą wartość stopni Celsjusza.

[0x12] (LSB) – niższy bajt (2 najstarsze bity), który zawiera część ułamkową temperatury w krokach co 0.25°C.

Aby odczytać temperaturę, musimy pobrać te dwa bajty, połączyć je i przekształcić na wartość temperatury w stopniach Celsjusza.

// Pobranie temperatury

```
#include <Wire.h>
#define DS3231_ADDRESS 0x68 // Adres I2C modułu DS3231

float getTemperature() {
    Wire.beginTransmission(DS3231_ADDRESS);
    Wire.write(0x11); // Adres rejestru temperatury
    Wire.endTransmission();
    Wire.requestFrom(DS3231_ADDRESS, 2);
    int8_t tempMSB = Wire.read(); // Część całkowita temperatury
    uint8_t tempLSB = Wire.read(); // Część ułamkowa temperatury
    return tempMSB + ((tempLSB >> 6) * 0.25); // Połączenie części całkowitej i ułamkowej
}

void setup() {
    Serial.begin(9600);
    Wire.begin();
    Wire.beginTransmission(DS3231_ADDRESS);
    if (Wire.endTransmission() != 0) {
        Serial.println("Brak połączenia z modułem DS3231!");
        while (1);
    }
}

void loop() {
    float temperature = getTemperature();
    Serial.print("Temperatura: ");
    Serial.print(temperature);
    Serial.println(" °C");
    delay(1000);
}
```

// Pobieranie i wyświetlanie daty, godziny i temperatury z biblioteką RTCLib

```
#include <Wire.h>
#include <RTCLib.h>
RTC_DS3231 rtc;

void setup() {
  Serial.begin(9600);
  Wire.begin();
  if (!rtc.begin()) {
    Serial.println("Nie można znaleźć modułu DS3231!");
    while (1);
  }
  if (rtc.lostPower()) {
    Serial.println("RTC stracił zasilanie, ustawiam czas...");
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }
}

void loop() {
  DateTime now = rtc.now(); // Pobranie bieżącej daty i godziny
  Serial.print("Data: ");
  Serial.print(now.year());
  Serial.print('/');
  Serial.print(now.month());
  Serial.print('/');
  Serial.println(now.day());

  Serial.print("Godzina: ");
  Serial.print(now.hour());
  Serial.print(':');
  Serial.print(now.minute());
  Serial.print(':');
  Serial.println(now.second());

  float temperature = rtc.getTemperature();
  Serial.print("Temperatura: ");
  Serial.print(temperature);
  Serial.println(" °C");

  delay(1000);
}
```

// Pobieranie i wyświetlanie daty i godziny bez bibliotek

```
#include <Wire.h>
#define DS3231_ADDRESS 0x68 // Adres I2C modułu DS3231

// Funkcja konwertująca liczbę BCD na format dziesiętny
uint8_t bcdToDec(uint8_t val) {
    return (val >> 4) * 10 + (val & 0b00001111);
}
// Funkcja konwertująca liczbę dziesiętną na format BCD
uint8_t decToBcd(uint8_t val) {
    return ((val / 10) << 4) | (val % 10);
}

void setTime() {
    char date[] = __DATE__; // Format: "Jan 25 2025"
    char time[] = __TIME__; // Format: "15:30:45"
    const char *months[] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
"Sep", "Oct", "Nov", "Dec" };
    uint8_t month = 0;
    for (uint8_t i = 0; i < 12; i++) {
        if (strncmp(&date[0], months[i], 3) == 0) {
            month = i + 1;
            break;
        }
    }
    uint8_t day = atoi(&date[4]); // Dzień
    uint16_t year = atoi(&date[7]); // Rok
    uint8_t hour = atoi(&time[0]); // Godzina
    uint8_t minute = atoi(&time[3]); // Minuty
    uint8_t second = atoi(&time[6]); // Sekundy

    Wire.beginTransmission(DS3231_ADDRESS);
    Wire.write(0x00); // Adres rejestru sekund
    Wire.write(decToBcd(second));
    Wire.write(decToBcd(minute));
    Wire.write(decToBcd(hour));
    Wire.write(decToBcd(0)); // Dzień tygodnia (nie używany)
    Wire.write(decToBcd(day));
    Wire.write(decToBcd(month));
    Wire.write(decToBcd(year - 2000)); // Rok (ostatnie dwie cyfry)
    Wire.endTransmission();
}

// Funkcja odczytująca datę i godzinę
void getDateTime(uint8_t &sec, uint8_t &min, uint8_t &hour, uint8_t &day, uint8_t &date,
uint8_t &month, uint16_t &year) {
    Wire.beginTransmission(DS3231_ADDRESS);
    Wire.write(0x00); // Adres początkowy rejestru
    Wire.endTransmission();
}
```

```

Wire.requestFrom(DS3231_ADDRESS, 7);
sec = bcdToDec(Wire.read()); // Sekundy
min = bcdToDec(Wire.read()); // Minuty
hour = bcdToDec(Wire.read()); // Godziny
day = bcdToDec(Wire.read()); // Dzień tygodnia
date = bcdToDec(Wire.read()); // Dzień miesiąca
month = bcdToDec(Wire.read()); // Miesiąc
year = bcdToDec(Wire.read()) + 2000; // Rok
}

void setup() {
  Serial.begin(9600);
  Wire.begin();
  Wire.beginTransmission(DS3231_ADDRESS);
  if (Wire.endTransmission() != 0) {
    Serial.println("Brak połączenia z modułem DS3231!");
    while (1)
      ;
  }
  setTime();
}

void loop() {
  uint8_t sec, min, hour, day, date, month;
  uint16_t year;
  getDateTime(sec, min, hour, day, date, month, year);

  Serial.print("Data: ");
  Serial.print(year);
  Serial.print('/');
  Serial.print(month);
  Serial.print('/');
  Serial.println(date);

  Serial.print("Godzina: ");
  Serial.print(hour);
  Serial.print(':');
  Serial.print(min);
  Serial.print(':');
  Serial.println(sec);

  delay(1000); // Poczekaj 1 sekundę
}

```

Obiekty można zapisywać w postaci JSON (słownik). Do jego obsługi można skorzystać z biblioteki ArduinoJson: <https://arduinojson.org/>

```
{  
  "imie": "Mateusz",  
  "numer": 1234  
}
```

```
// Przykład: Serializacja na obiekt Json  
#include <ArduinoJson.h>  
struct JData{  
  const char* imie;  
  long numer;  
};  
JData jdata;  
  
void setup() {  
  Serial.begin(9600);  
  const char* input = "{ \"imie\": \"Mateusz\", \"numer\":1234 }";  
  JsonDocument doc;  
  DeserializationError error = deserializeJson(doc, input);  
  if(error){  
    Serial.print("Error: ");  
    Serial.println(error.c_str());  
    return;  
  }  
  jdata.imie = doc["imie"];  
  jdata.numer = doc["numer"];  
  Serial.println(jdata.imie);  
  Serial.println(jdata.numer);  
}  
void loop() {}
```

```

// Przykład: Pobieranie obiektu JSON z pliku na karcie SD
#include <SPI.h>
#include <SD.h>
#include <ArduinoJson.h>
const char* fileName = "test.txt";
File myFile;
struct Config {
    char hostname[64];
    int port;
};
Config config;
void setup() {
    Serial.begin(9600);
    while (!Serial) {
        ; // wait for serial port to connect. Needed for native USB port only
    }
    Serial.print("Initializing SD card...");
    if (!SD.begin(4)) {
        Serial.println("initialization failed!");
        while (1);
    }
    Serial.println("initialization done.");
    myFile = SD.open(fileName, FILE_WRITE); //O_APPEND, O_CREATE
    if (myFile) {
        myFile.close();
    } else {
        Serial.println("error opening file");
    }
    myFile = SD.open(fileName); // re-open the file for reading:
    if (myFile) {
        JsonDocument doc;
        DeserializationError error = deserializeJson(doc, myFile);
        if(error){
            Serial.print("Error: ");
            Serial.println(error.c_str());
            return;
        }
        config.port = doc["port"];
        strncpy(config.hostname, doc["hostname"], sizeof(config.hostname));
        Serial.print("Host: ");
        Serial.println(config.hostname);
        Serial.print("Port: ");
        Serial.println(config.port);
    }
}

```

```
    myFile.close();  
  } else {  
    Serial.println("error opening file");  
  }  
}  
void loop() { }
```

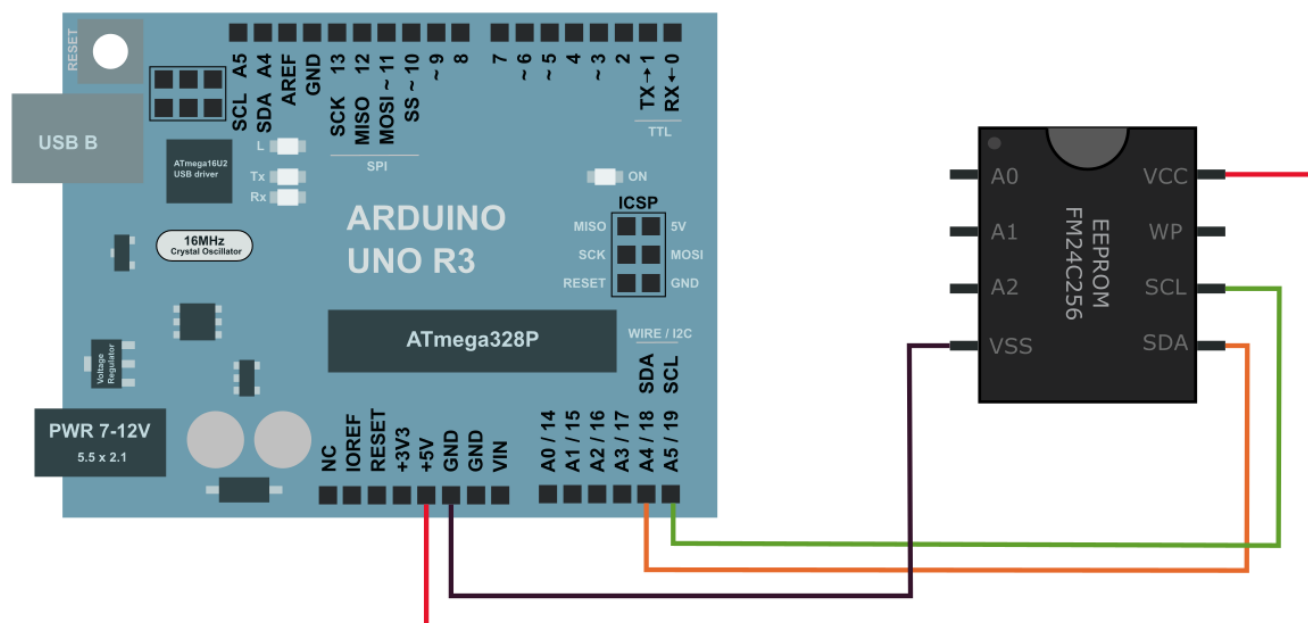
Wbudowana pamięć EEPROM wynosi 1024 Bajty.

```
// Przykład: Zapisywanie i odczytywanie danych z wbudowanej pamięci EEPROM
#include <EEPROM.h>
int addr = 0;
byte val = 12;
void setup() {
    Serial.begin(9600);
    EEPROM.write(addr, val);
    delay(5);
    int readVal = EEPROM.read(addr);
    Serial.print("Value read from EEPROM: ");
    Serial.println(readVal);
}
void loop() {
}
```

Przykład: Zapisywanie i odczytywanie danych większych od 1B

```
#include <EEPROM.h>
const int EEPROM_ADDRESS = 0;
void setup() {
    Serial.begin(9600);
}
void loop() {
    if (Serial.available()) {
        //int dataToWrite = Serial.parseInt();
        String dataToWrite = Serial.readString();
        EEPROM.put(EEPROM_ADDRESS, dataToWrite);
        delay(10);
        //int valueRead;
        String valueRead;
        EEPROM.get(EEPROM_ADDRESS, valueRead);
        Serial.print("Odczytana wartość: ");
        Serial.println(valueRead);
    }
}
```

Zewnętrzna pamięć Eeprom FM24C256



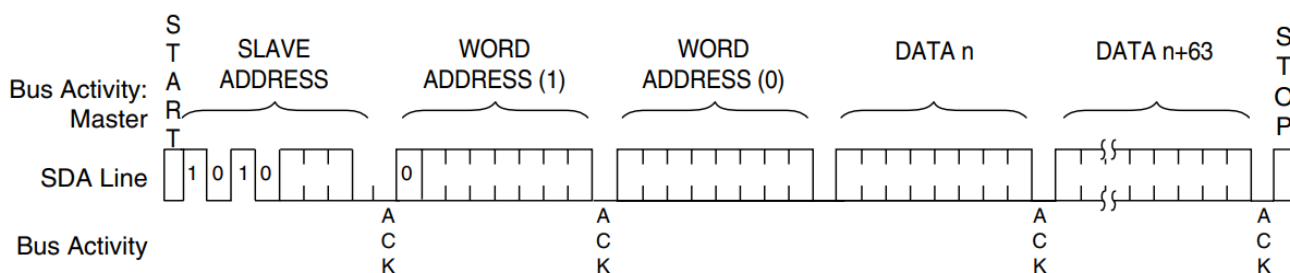
Pamięć eeprom posiada domyślny adres I2C: 1010000, który można modyfikować za pomocą pinów A0, A1 i A2.

1	0	1	0	A2	A1	A0	R/W
---	---	---	---	----	----	----	-----

Ilość komórek pamięci jest znacznie większa od 255, dlatego ich adresowanie odbywa się za pomocą 16bitowego adresu. Dane zapisywane w buforze mogą zajmować maksymalnie 1B, dlatego adres komórek należy rozdzielić na 2 składowe 8 bitowe.

Pamięć FM24C256 jest podzielona na 64 Bajtowe strony, co umożliwia szybsze i łatwiejsze zapisywanie / odczytywanie danych w kolejnych komórkach.

Mając na uwadze maksymalny rozmiar bufora Arduino, tj. 32 Bajty, w jednej operacji można przesłać maksymalnie 2 Bajty adresu i 30 Bajtów danych.



Zapisywanie 1 Bajta danych do Eeprom

```
#include <Wire.h>
#define EEPROM_I2C_ADDRESS 0x50
unsigned int eepromAddress = 0;

void setup() {
  Wire.begin();
  Serial.begin(9600);
  int readData = readEEPROM(eepromAddress);
  Serial.println(readData);
}

void loop() {
  if (Serial.available()) {
    int dataToWrite = Serial.parseInt();
    writeEEPROM(eepromAddress, dataToWrite);
    delay(10);

    int readData = readEEPROM(eepromAddress);
    Serial.print("Odczytana wartość: ");
    Serial.println(readData);
  }
}

void writeEEPROM(unsigned int address, byte data) {
  Wire.beginTransaction(EEPROM_I2C_ADDRESS);
  Wire.write((int)(address >> 8)); // Wyślij wyższy bajt adresu
  Wire.write((int)(address & 0xFF)); // Wyślij niższy bajt adresu
  Wire.write(data);
  Wire.endTransmission();
  delay(5);
}

byte readEEPROM(unsigned int address) {
  byte rData = 0xFF;
  Wire.beginTransaction(EEPROM_I2C_ADDRESS);
  Wire.write((int)(address >> 8)); // Wyślij wyższy bajt adresu
  Wire.write((int)(address & 0xFF)); // Wyślij niższy bajt adresu
  Wire.endTransmission();
  // Wire.requestFrom(address, quantity, stop);
  Wire.requestFrom(EEPROM_I2C_ADDRESS, 1);
  if (Wire.available()) rData = Wire.read();
  return rData;
}
```

Zapisywanie wartości tekstowych do Eeprom

```
#include <Wire.h>
#define EEPROM_ADDR 0x50    // Adres I2C pamięci EEPROM (7-bitowy adres)
#define BUFFER_SIZE 32     // Maksymalny rozmiar bufora dla I2C

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

void loop() {
    if (Serial.available()) {
        String sentence = Serial.readStringUntil('\n');
        sentence.trim();
        writeStringToEEPROM(0x0000, sentence.c_str());
        char buffer[BUFFER_SIZE];
        readStringFromEEPROM(0x0000, buffer, BUFFER_SIZE);
        Serial.println(buffer);
    }
}

void writeStringToEEPROM(uint16_t address, const char* str) {
    uint16_t length = strlen(str);
    uint16_t offset = 0;
    while (offset < length) {
        Wire.beginTransmission(EEPROM_ADDR);
        Wire.write((uint8_t)(address >> 8)); // Wysłanie starszego bajtu adresu
        Wire.write((uint8_t)(address & 0xFF)); // Wysłanie młodszego bajtu adresu
        //uint8_t bytesToWrite = min(BUFFER_SIZE - 2, length - offset);
        uint8_t bytesToWrite = BUFFER_SIZE - 2; // Odejmujemy 2 bajty dla adresu
        for (uint8_t i = 0; i < bytesToWrite; i++) {
            Wire.write(str[offset + i]);
        }
        Wire.endTransmission();
        delay(5);
        address += bytesToWrite;
        offset += bytesToWrite;
    }
}

void readStringFromEEPROM(uint16_t address, char* buffer, uint16_t bufferLength) {
    uint16_t offset = 0;
    while (offset < bufferLength - 1) {
        Wire.beginTransmission(EEPROM_ADDR);
        Wire.write((uint8_t)(address >> 8)); // Wysłanie starszego bajtu adresu
```

```

Wire.write((uint8_t)(address & 0xFF)); // Wysłanie młodszego bajtu adresu
Wire.endTransmission();
uint8_t bytesToRead = min(BUFFER_SIZE, bufferLength - 1 - offset);
Wire.requestFrom(EEPROM_ADDR, bytesToRead);
while (Wire.available()) {
    buffer[offset] = Wire.read();
    offset++;
    if (offset >= bufferLength - 1) {
        break; // Zabezpieczenie przed przepełnieniem bufora
    }
}
address += bytesToRead;
}
buffer[offset] = '\0'; // Dodanie null terminatora na końcu odczytanego ciągu
}

```

CD4017 to układ scalony będący licznikiem dziesiętnym Johnsona z dekodowaniem wyjść.

Układ ma 10 wyjść, które są aktywowane kolejno jeden po drugim.

Piny układu CD4017

Pin 15 (Reset): Resetuje licznik, czyli ustawia wszystkie wyjścia w stan niski. Wysoki stan na tym pinie spowoduje reset.

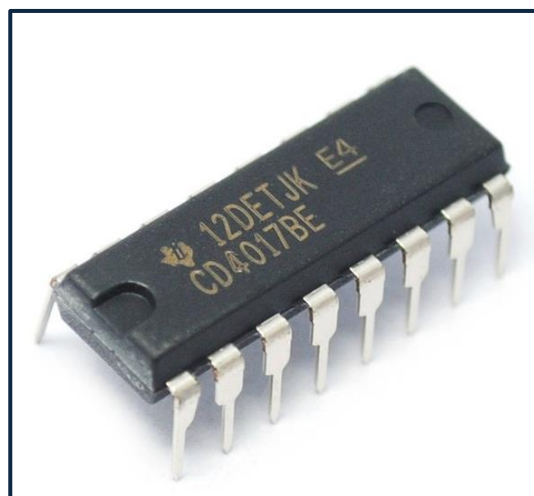
Pin 14 (Clock Input): Wejście zegarowe. Na ten pin podajesz sygnał taktujący, który decyduje o przełączaniu wyjść.

Pin 13 (Clock Enable): Gdy jest w stanie wysokim, blokuje licznik, więc należy go podłączyć do GND, jeśli chcesz, by licznik działał.

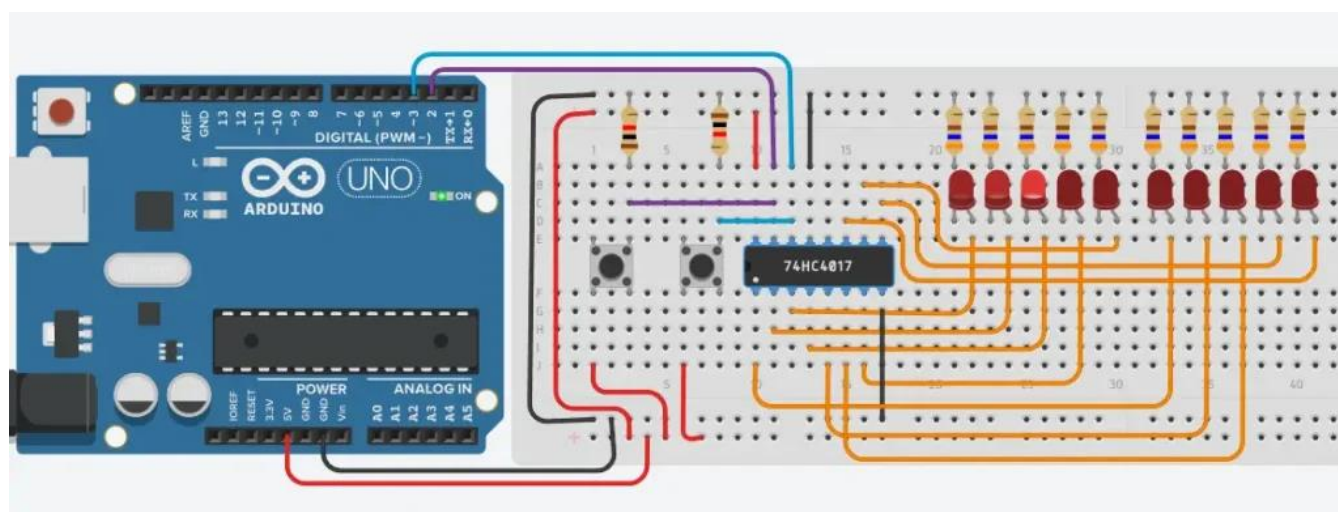
Piny 1-7, 9-11 (Wyjścia Q0 – Q9): Każde z tych wyjść staje się aktywne w zależności od sygnału zegarowego.

Pin 16 (Vcc): Zasilanie (zazwyczaj 5V).

Pin 8 (GND): Masa.



Połączenie 74HC4017 Johnson Decade Counter z Arduino



```
// Przełączanie wyjść na liczniku CD4017
```

```
int clockPin = 3;
```

```
int resetPin = 2;
```

```
void setup(){
```

```
  pinMode(clockPin, OUTPUT);
```

```
  pinMode(resetPin, OUTPUT);
```

```
}
```

```
void loop(){
```

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

```
    digitalWrite(clockPin, HIGH);
```

```
    digitalWrite(clockPin, LOW);
```

```
    delay(200);
```

```
  }
```

```
  digitalWrite(resetPin, HIGH);
```

```
  digitalWrite(resetPin, LOW);
```

```
}
```

Shift Register 74HC595

Shift Register 74HC595 to 8-bitowy chip zawierający 16 pinów, dzięki któremu można zarządzać 8 wyjściami cyfrowymi. Pozwala to rozszerzyć liczbę dostępnych pinów cyfrowych.

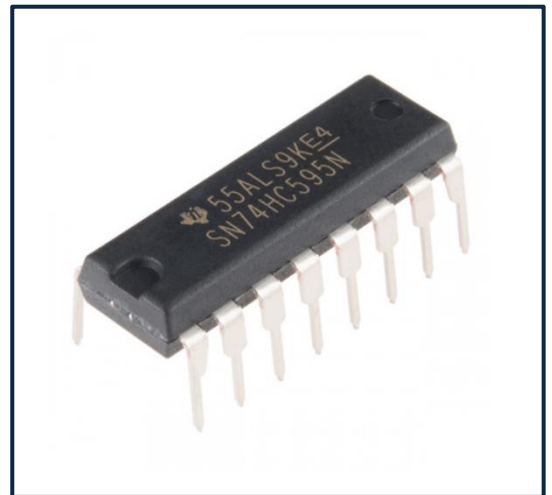
Jak działa rejestr przesuwający?

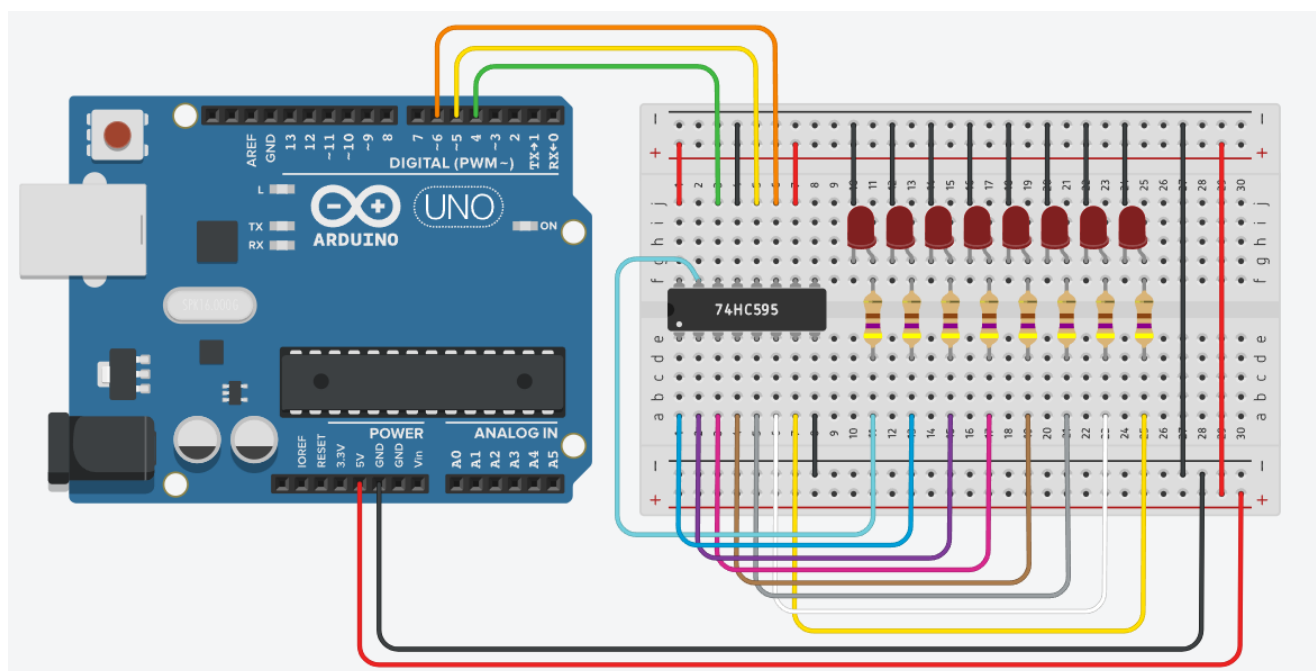
Moduł składa się z rejestru przesuwającego (Shift Register) i rejestru pamięci (storage register).

Każdy z nich posiada osobny zegar wewnętrzny.

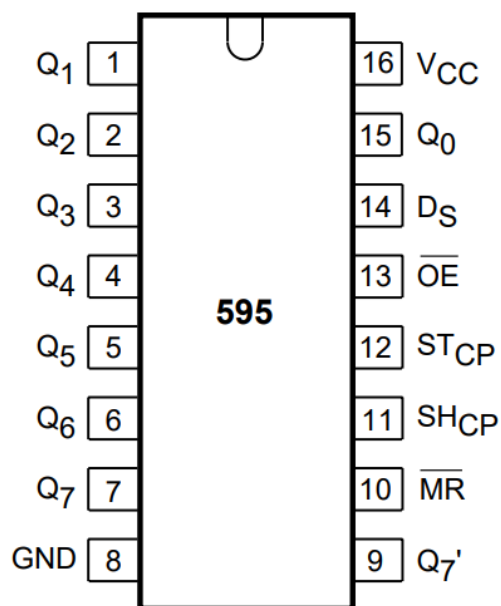
Kiedy clockPin zmienia stan z Low na High, shift register czytuje stan z dataPin. Dane zostają przetransferowane do rejestru pamięci. Kiedy latchPin zmienia stan z Low na High, wtedy równoległe sygnały zostają wysłane do wyjść Q0-Q7.

Rejestr 74HC595 podaje prąd na wyjścia, dlatego nazywamy go sourcing current, ale istnieją również rejestry, tzw. sinking current, w przypadku których należy podłączyć odbiorniki na pinach w odwrotny sposób.





SYMBOL	PIN	DESCRIPTION
Q0 to Q7	15, 1 to 7	parallel data output
GND	8	ground (0 V)
Q7'	9	serial data output
MR	10	master reset (active LOW)
SHCP	11	shift register clock input
STCP	12	storage register clock input
OE	13	output enable (active LOW)
DS	14	serial data input
VCC	16	positive supply voltage



shiftOut()

Metoda shiftOut przesuwa byte danych, bit po bicie. Przesuwanie może rozpocząć się od MSB lub LSB.

shiftOut(dataPin, clockPin, bitOrder, value)

dataPin – pin przesyłający dane (int)

clockPin – pin zegara (int)

bitOrder – kierunek przesuwu (MSBFIRST lub LSBFIRST)

value – wartość (byte)

```
// Przykład: Wyświetlenie 1 bajta za pomocą diod LED i shiftOut
int dataPin = 4;
int latchPin = 5;
int clockPin = 6;
byte databyte = 0b10100101;

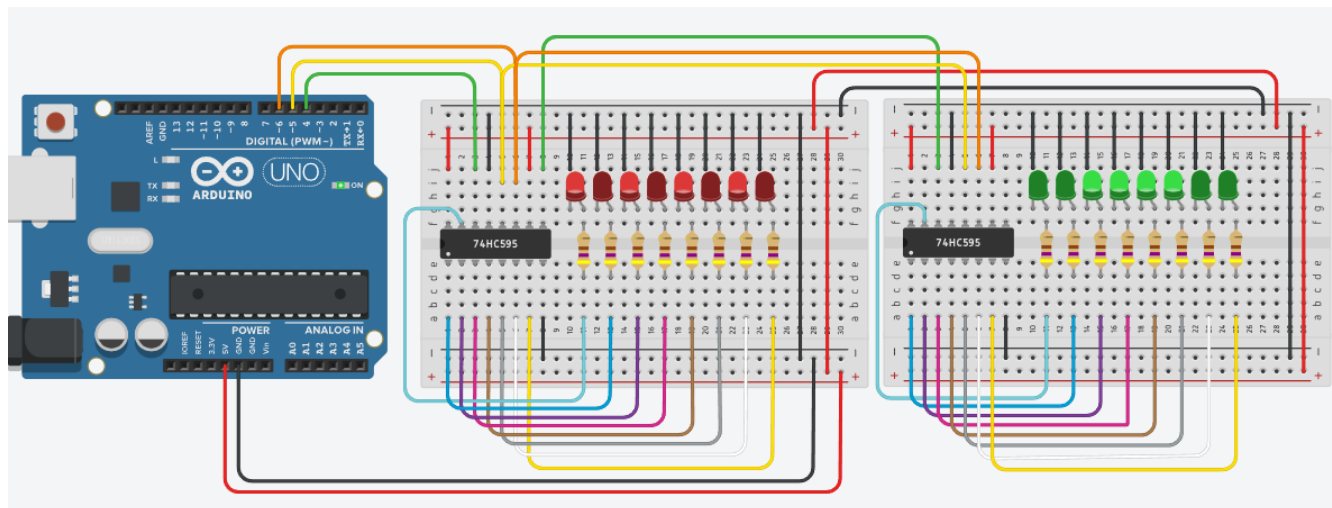
void setup() {
  pinMode(dataPin, OUTPUT);
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  digitalWrite(latchPin, LOW);
  //shiftOut(dataPin, clockPin, LSBFIRST, databyte);
  for (int i = 0; i <= 7; i++) {
    digitalWrite(clockPin, LOW);
    digitalWrite(dataPin, bitRead(databyte, i));
    digitalWrite(clockPin, HIGH);
  }
  digitalWrite(latchPin, HIGH);
}
void loop() {}
```

```
// Przykład: Shift Register bez shiftOut

int dataPin = 4;
int latchPin = 5;
int clockPin = 6;

byte databyte = 0b10000101;

void setup() {
    pinMode(dataPin, OUTPUT);
    pinMode(latchPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
    digitalWrite(latchPin, LOW);
    //shiftOut(dataPin, clockPin, LSBFIRST, databyte);
    for (int i = 0; i <= 7; i++) {
        digitalWrite(clockPin, LOW);
        digitalWrite(dataPin, bitRead(databyte, i));
        digitalWrite(clockPin, HIGH);
    }
    digitalWrite(latchPin, HIGH);
}
void loop() {}
```



//Przykład: Połączenie dwóch rejestrów przesuwnych

```
int latchPin = 5;
int clockPin = 6;
int dataPin = 4;
unsigned int leds1, leds2 = 0;

void setup() {
    pinMode(latchPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
    leds1 = 0b10101010;
    leds2 = 0b00111100;
    updateShiftRegisterL(leds1);
    updateShiftRegisterL(leds2);
}

void updateShiftRegisterL(byte databyte) {
    digitalWrite(latchPin, LOW);
    shiftOut(dataPin, clockPin, LSBFIRST, databyte);
    digitalWrite(latchPin, HIGH);
}

void loop() {}
```

Tworzenie biblioteki do obsługi 74HC595

Wytyczne tworzenia bibliotek w Arduino

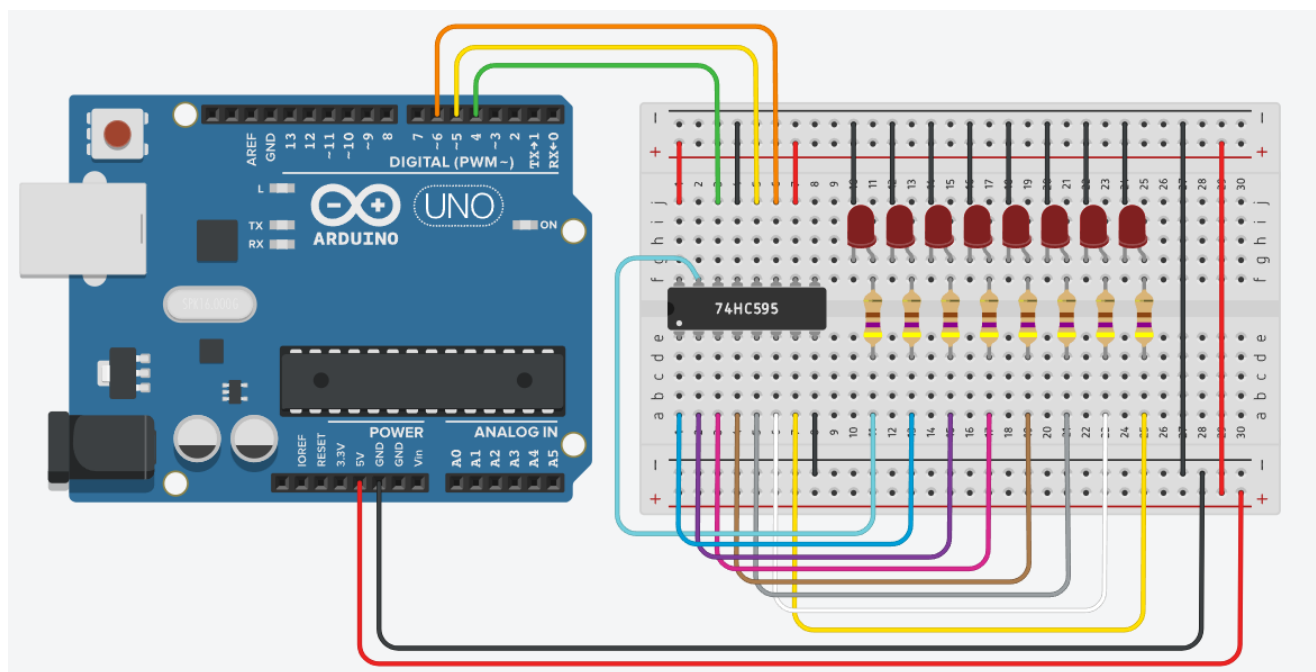
<https://docs.arduino.cc/learn/contributions/arduino-creating-library-guide>

<https://arduino.github.io/arduino-cli/0.35/library-specification>

Tworzenie własnej biblioteki Arduino polega na przygotowaniu kilku plików. Na koniec należy je wszystkie spakować i w taki sposób importować do Arduino IDE.

Biblioteki są zapisywane w lokalizacji: C:\Users\wilin\Documents\Arduino\libraries\NaszaBib

Instrukcja `#include` dodaje biblioteki do projektu, ścieżka podana w cudzysłowie "lib.h" świadczy o tym, że plik znajduje się w tym samym folderze, a adres podany w nawiasach ostrych <lib.h>, świadczy o tym, że biblioteka znajduje się w folderze głównym.



Plik implementacyjny funkcji deklarowanych w pliku nagłówkowym

```
#include "ShiftReg.h"

int _dataPin;
int _latchPin;
int _clockPin;

ShiftReg::ShiftReg(int dataPin, int latchPin, int clockPin) {
    _dataPin = dataPin;
    _latchPin = latchPin;
    _clockPin = clockPin;
}

void ShiftReg::begin() {
    pinMode(_dataPin, OUTPUT);
    pinMode(_latchPin, OUTPUT);
    pinMode(_clockPin, OUTPUT);
}

void ShiftReg::sendData(byte data) {
    digitalWrite(_latchPin, LOW);
    for (int i = 0; i <= 7; i++) {
        digitalWrite(_clockPin, LOW);
        digitalWrite(_dataPin, bitRead(data, i));
        digitalWrite(_clockPin, HIGH);
    }
    digitalWrite(_latchPin, HIGH);
}
```

Plik .h (header)

Główny plik nagłówkowy biblioteki Arduino, zawiera podstawowe definicje i deklaracje funkcji

```
/*
  ShiftReg.h - Library for Shift Register 74HC595
  Created by Mateusz Wilinski, May 19, 2024.
  Released into the public domain.
*/
#ifndef ShiftReg_h
#define ShiftReg_h
#include <Arduino.h>

class ShiftReg {
public:
  ///! Tworzy obiekt ShiftReg
  ///!
  ///! @param dataPin nr pinu data połączony z pinem Input (14) 74HC595
  ///! @param latchPin nr pinu latch połączony z pinem ORC (12) 74HC595
  ///! @param clockPin nr pinu clock połączony z pinem SRC (11) 74HC595

  ShiftReg(int dataPin, int latchPin, int clockPin);

  ///! inicjalizuje obiekt ShiftReg
  void begin();

  ///! wysyła bajt danych do rejestru
  void sendData(byte data);
private:
};

#endif
```

Plik keywords.txt

```
# -----  
# ShiftReg library syntax  
# -----  
  
# Datatypes  
ShiftReg KEYWORD1  
  
# Methods  
begin KEYWORD2  
sendData KEYWORD2  
  
# Constants (LITERAL1)
```

Plik library.properties

Metadane biblioteki (autor, wersja, zależności)

```
name=ShiftReg  
version=1.0.0  
author=Mateusz Wilinski, astraja.pro@gmail.com  
maintainer=Mateusz Wilinski, astraja.pro@gmail.com  
sentence=Interact with 74HC595 Shift Register  
paragraph=ShiftReg makes interaction with Shift Register easy  
category=Device Control  
url=https://github.com/astraja/ShiftReg.git  
architectures=avr,arm,megaavr,esp32  
includes=ShiftReg.h
```

Podstawowy kod Arduino

```
#include <ShiftReg.h>  
ShiftReg shiftReg(4,5,6); // ShiftReg(int dataPin, int latchPin, int clockPin);  
byte databyte = 0b10000101;  
  
void setup(){  
    shiftReg.begin();  
    shiftReg.sendData(databyte);  
}  
void loop(){}
```

Expander PCF8574

PCF8574 służy do rozszerzania liczby portów wejścia/wyjścia poprzez interfejs I2C.

Każdy z 8 pinów PCF8574 może być ustawiony jako wejście lub wyjście.

Po uruchomieniu modułu piny przyjmują stan HIGH

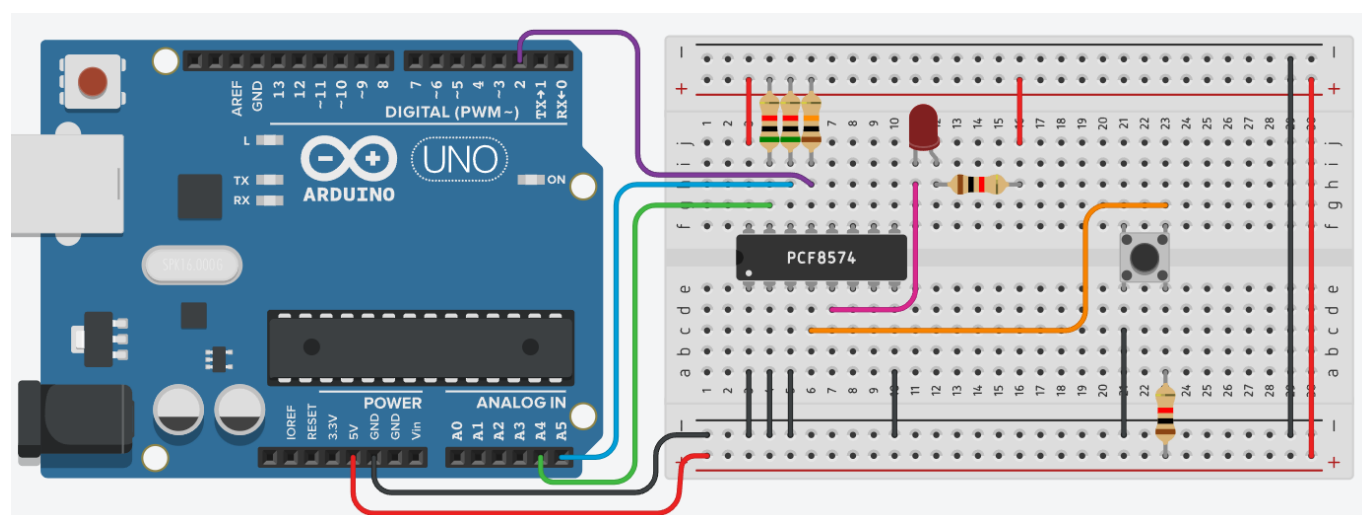
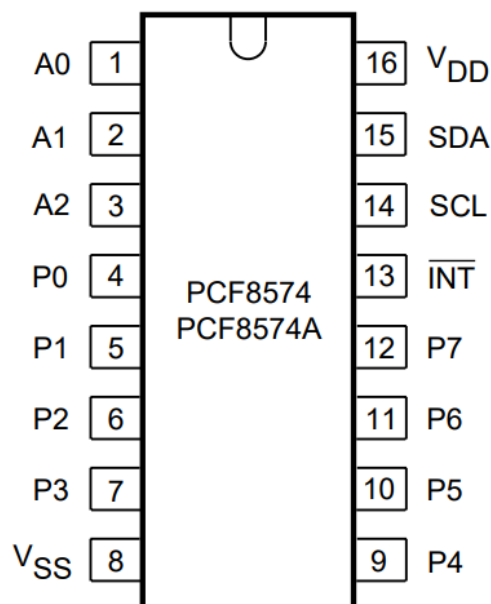
Max. Prąd (sinking current) wynosi 25mA.

Dla stanu wysokiego prąd płynący z pinów wynosi maks. 300uA, a dla stanu niskiego 25mA. W związku z tym, elementy sterowane podłączane są anodą do zasilania 5V, a katodą do PCF8574 lub za pomocą tranzystora NPN.

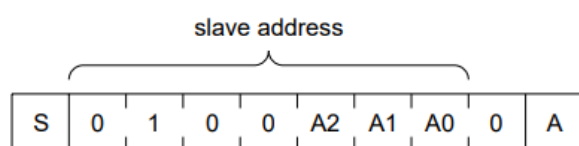
0 ustawia piny jako wyjściowe, a 1 jako piny wejściowe.

Expanderem można sterować za pomocą biblioteki <PCF8574.h> lub standardowego połączenia I2C.

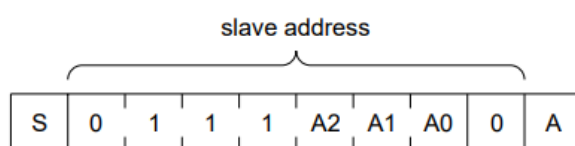
Pin nr 13 (Interrupt) zmienia stan z wysokiego na niski przy zmianie stanu któregośkolwiek z pinów cyfrowych. Powinien być podłączony z pullup rezystorem.



Ustawienie adresu I2C PCF8574(a) / PCF8574A(b)



a.



MBD973

b.

Zarządzanie PCF8574 za pomocą biblioteki

PCF8574 library by Renzo

Mischianti...

2.3.7 installed

Most starred PCF8574 library for
Arduino (standard and SAMD),
Raspberry Pi Pico and rp2040 boards...
[More info](#)

2.3.7

REMOVE

https://github.com/xreef/PCF8574_library

```
#include <Wire.h>
#include <PCF8574.h>
const int addr = 0x20; // Adres domyślny dla PCF8574
PCF8574 pcf8574(addr);

void setup() {
  Wire.begin();
  pcf8574.begin();
  pcf8574.pinMode(0, OUTPUT); // Ustawienie pierwszego pinu jako wyjście
}

void loop() {
  pcf8574.digitalWrite(0, HIGH);
  delay(1000);
  pcf8574.digitalWrite(0, LOW);
  delay(1000);
}
```

Zarządzanie PCF8574 przez I2C

```
// Pobieranie stanów pinów PCF8574
#include <Wire.h>
const int PCF8574_ADDR = 0x20;

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

void loop() {
    Wire.requestFrom(PCF8574_ADDR, 1);
    if (Wire.available()) {
        byte data = Wire.read();
        Serial.println(data, BIN);
    }
    delay(500);
}
```

```
// Pobieranie stanów pinów po ich zmianie
#include <Wire.h>
const int PCF8574_ADDR = 0x20;

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

void loop() {
    if (digitalRead(2) == 0) {
        Wire.requestFrom(PCF8574_ADDR, 1);
        if (Wire.available()) {
            byte data = Wire.read();
            Serial.println(data, BIN);
            delay(200);
        }
    }
}
```

```

// Pobieranie stanów pinów z interrupt
#include <Wire.h>
#define PCF8574_ADDR 0x20
volatile bool canGetData = false;

void setup() {
    Serial.begin(9600);
    Wire.begin();
    attachInterrupt(digitalPinToInterrupt(2), handleInterrupt, FALLING);
}

void loop() {
    if (canGetData) {
        canGetData = false;
        Wire.requestFrom(PCF8574_ADDR, 1);
        if (Wire.available()) {
            byte newData = Wire.read();
            Serial.println(newData, BIN);
        }
    }
}

void handleInterrupt() {
    canGetData = true;
}

```

```

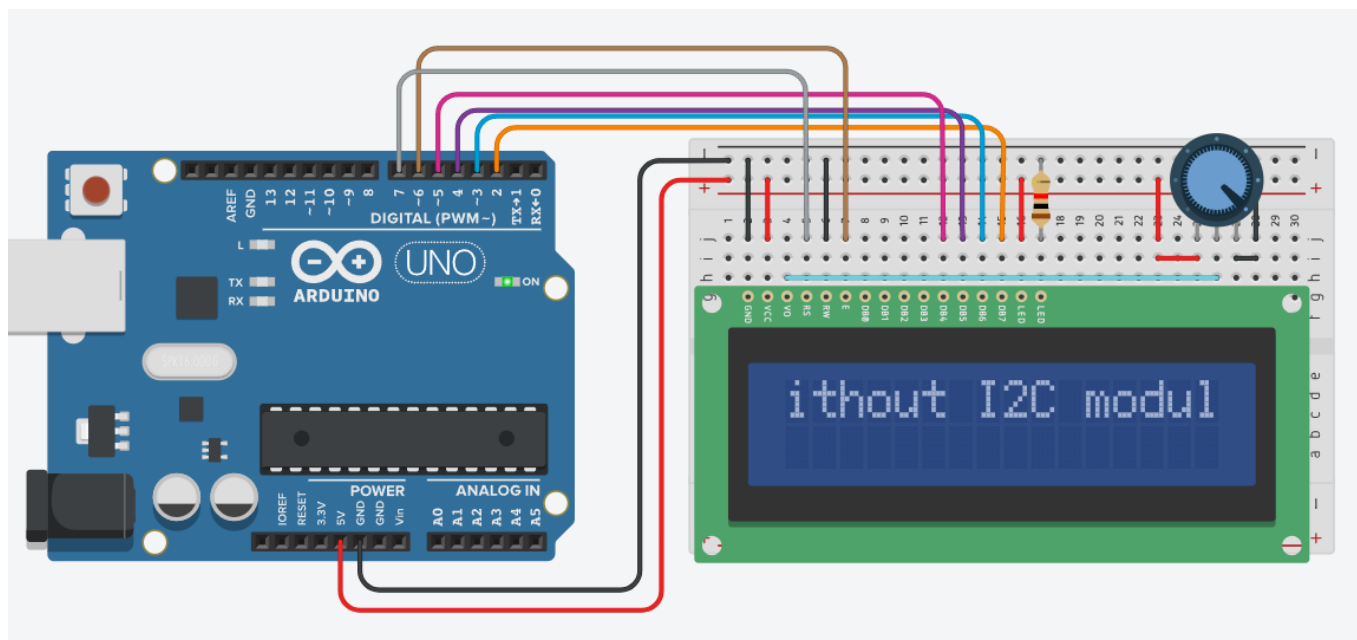
// Wysyłanie danych do PCF8574
#include <Wire.h>
const int PCF8574_ADDR = 0x20;

void setup() {
    delay(1000);
    Wire.begin();
    Wire.beginTransmission(PCF8574_ADDR);
    Wire.write(0b01111111); // Ustawienie ostatniego bitu na 0 (output)
    Wire.endTransmission();
}

void loop() {}

```

Na rynku dostępne są wyświetlacze LCD z lub bez dodatkowego modułu i2c. Moduł i2c pozwala na dużo łatwiejsze połączenie wyświetlacza z płytką (4 przewody). Wyświetlacze bez modułu i2c wymagają użycia aż 6 przewodów, rezystora i potencjometru do sterowania jasnością wyświetlacza.



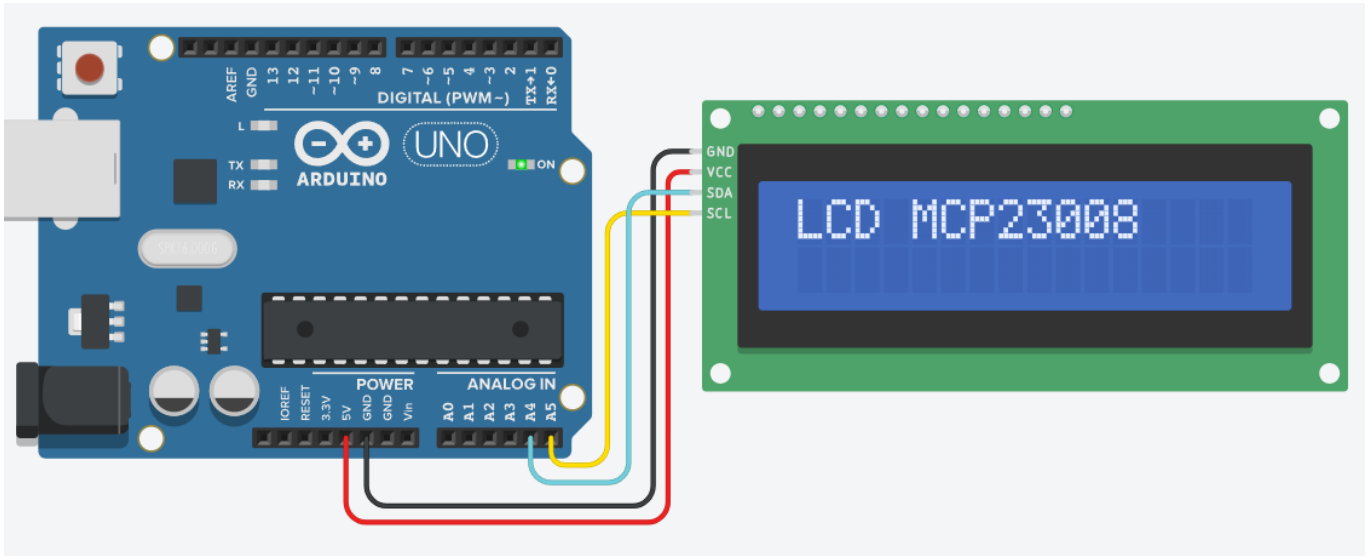
Rys. 3 Połączenie wyświetlacza 16x2 z Arduino

LCD 16x2 LiquidCrystal

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
void setup() {
    lcd.begin(16, 2);
}
void loop(){
    displayText();
}
void displayText(){
    lcd.setCursor(0,0); //kolumna, wiersz
    lcd.print("LCD without I2C module");
    delay(1000);
    for(int i=0;i<23;i++){
        lcd.scrollDisplayLeft();
        delay(100);
    }
    lcd.home();
}
```

LCD 16x2 I2C PCF8574 Adafruit_LiquidCrystal

Moduł PCF8574 to dostawka I2C (expander) do wyświetlacza MCP23008. Domyślnie wyświetlacze z dostawką mają adres 0x27.



Rys. 4 Połączenie wyświetlacza 16x2 I2C z Arduino

Przykład: Podstawowy program z użyciem biblioteki Adafruit_LiquidCrystal

```
#include <Adafruit_LiquidCrystal.h>
Adafruit_LiquidCrystal lcd(0);

void setup(){
  lcd.begin(16, 2);
  lcd.print("LCD MCP23008");
}
void loop(){}

```

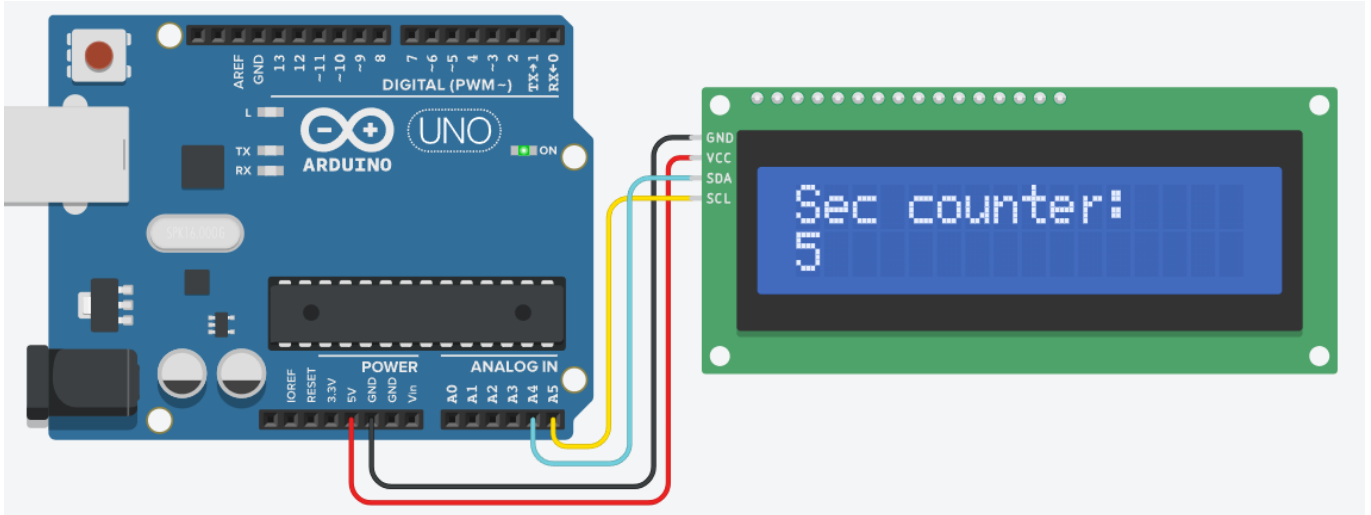
LCD 16x2 I2C LiquidCrystal_I2C

```
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
void setup(){
  lcd.begin();
  lcd.setCursor(0, 0);
  lcd.print("LCD I2C Testing");
}
void loop(){}

```

W tinkercad należy wybrać LCD PCF8574 i adres 32

```
LiquidCrystal_I2C lcd(32, 16, 2);
```



Przykład: LCD 16x2 I2C LiquidCrystal_I2C Licznik sekund

```
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2); //tinkercad
int interval = 1000;
long lastTime = 0;
int sec = 0;

void setup(){
  lcd.init();
  lcd.backlight();
  lcd.print("Sec counter:");
  DisplayInfo();
}

void DisplayInfo(){
  lcd.setCursor(0,1);
  lcd.print(sec);
  sec++;
}

void loop(){
  long timeNow = millis();
  if(timeNow >= lastTime + interval){
    DisplayInfo();
    lastTime = timeNow;
  }
}
```

Tworzenie znaku

Na wyświetlaczu można wyświetlić własne znaki, które można stworzyć dzięki tablicy byte. Każde pole zawiera 5 x 8 pixeli, które można włączyć lub wyłączyć.



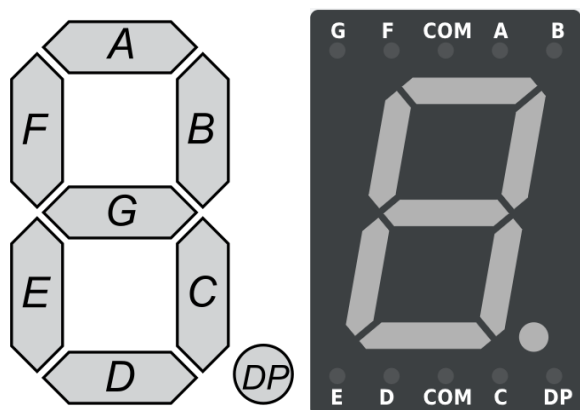
```
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(32, 16, 2);

byte minka[] = {
  B00000,
  B00000,
  B01010,
  B00000,
  B10001,
  B01110,
  B00000,
  B00000
};

void setup() {
  lcd.init();
  lcd.backlight();
  lcd.createChar(0,minka);
  lcd.setCursor(6,1);
  lcd.print("Minka");
}

void loop() {
  lcd.setCursor(8, 0);
  lcd.write(byte(0));
  delay(100);
}
```

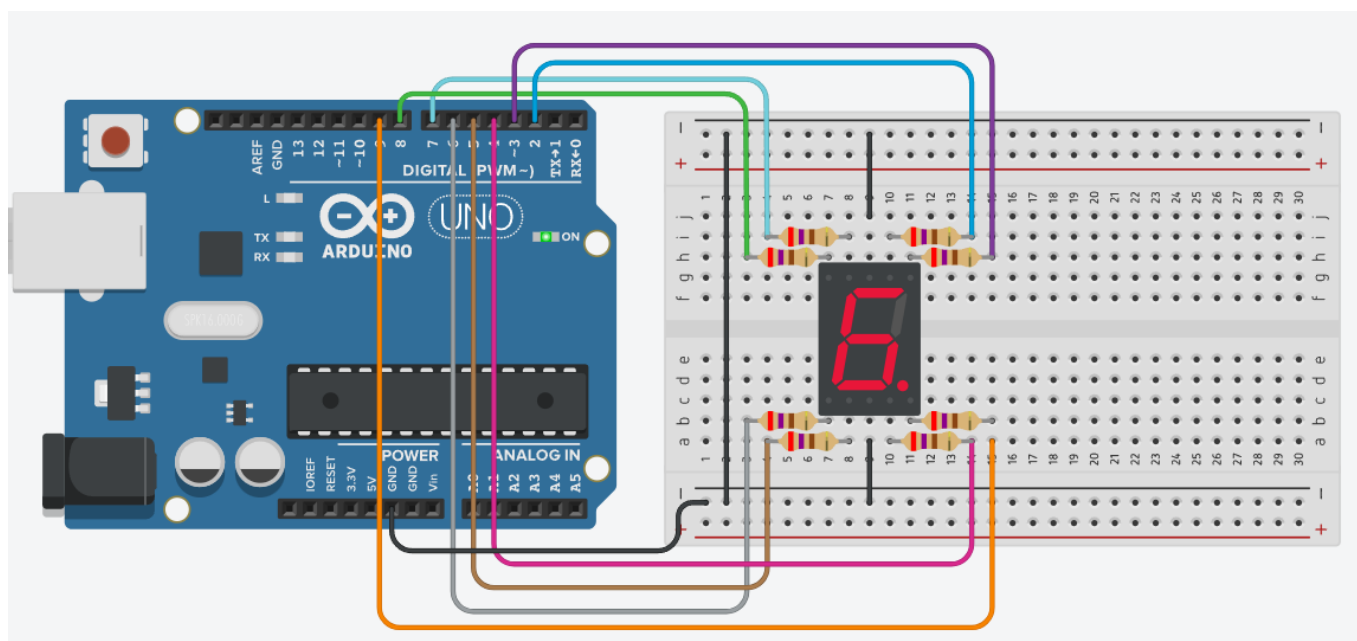
Wyświetlacz 7 segmentowy



Wyświetlacz występuje w 2 wersjach:

- wspólna katoda połączona z masą
- wspólna anoda połączona ze źródłem zasilania

Wyświetlacz posiada 10 pinów, 8 z nich służy do włączenia odpowiedniego segmentu LED (A, B, C, D, E, F, G, DP).



Rys. 5. Połączenie wyświetlacza ze wspólną katodą

Przykład: Wyświetlenie kolejnych cyfr na wyświetlaczu

```
byte pins[] = { 2, 3, 4, 5, 6, 7, 8, 9 };
byte len = sizeof(pins) / sizeof(pins[0]);
byte nums[10][8] = {
    { 1, 1, 1, 1, 1, 1, 0, 0 }, // 0
    { 0, 1, 1, 0, 0, 0, 0, 0 }, // 1
    { 1, 1, 0, 1, 1, 0, 1, 0 }, // 2
    { 1, 1, 1, 1, 0, 0, 1, 0 }, // 3
    { 0, 1, 1, 0, 0, 1, 1, 0 }, // 4
    { 1, 0, 1, 1, 0, 1, 1, 0 }, // 5
    { 1, 0, 1, 1, 1, 1, 1, 0 }, // 6
    { 1, 1, 1, 0, 0, 0, 0, 0 }, // 7
    { 1, 1, 1, 1, 1, 1, 1, 0 }, // 8
    { 1, 1, 1, 1, 0, 1, 1, 0 }  // 9
};

void setup() {
    setPins();
    displayCounter(9);
}

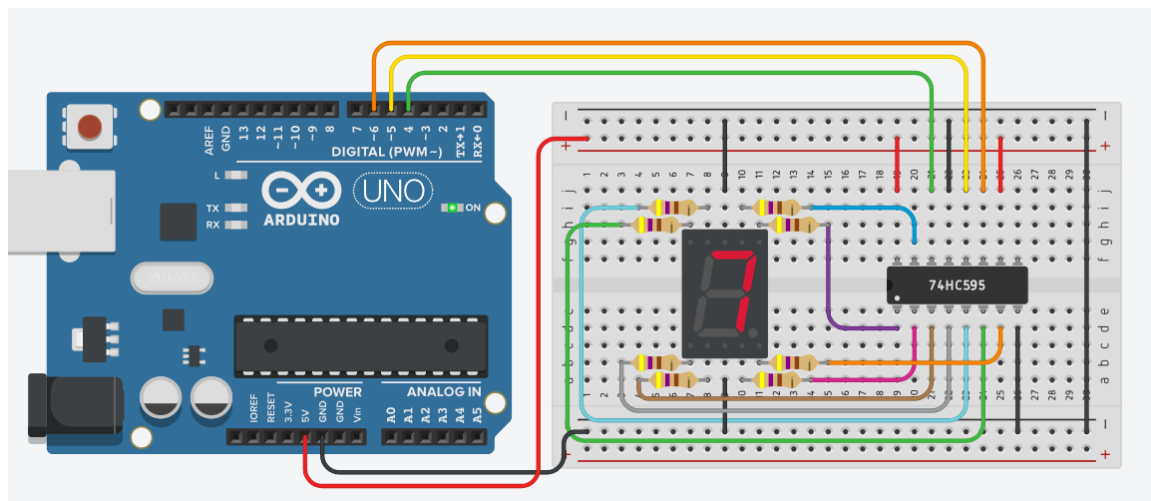
void loop() {}

void displayNumber(int number) {
    for (byte i = 0; i < 7; i++) {
        digitalWrite(pins[i], nums[number][i]);
        digitalWrite(pins[7], 1); //dot switch
    }
}

void setPins() {
    for (byte i = 0; i < len; i++) {
        pinMode(pins[i], OUTPUT);
    }
}

void displayCounter(byte number) {
    for (byte i = 0; i <= number; i++) {
        displayNumber(i);
        delay(500);
    }
}
```

Wyświetlacz 7 segmentowy z Shift Register



Rys. 6 Połączenie wyświetlacza z rejestrem przesuwным

```

byte latchPin = 5, clockPin = 6, dataPin = 4, digit;
byte nums[10][8] = {
    { 1, 1, 1, 1, 1, 1, 0, 0 }, // 0
    { 0, 1, 1, 0, 0, 0, 0, 0 }, // 1
    { 1, 1, 0, 1, 1, 0, 1, 0 }, // 2
    { 1, 1, 1, 1, 0, 0, 1, 0 }, // 3
    { 0, 1, 1, 0, 0, 1, 1, 0 }, // 4
    { 1, 0, 1, 1, 0, 1, 1, 0 }, // 5
    { 1, 0, 1, 1, 1, 1, 1, 0 }, // 6
    { 1, 1, 1, 0, 0, 0, 0, 0 }, // 7
    { 1, 1, 1, 1, 1, 1, 1, 0 }, // 8
    { 1, 1, 1, 1, 0, 1, 1, 0 } // 9
};

void setup() {
    pinMode(latchPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
    displayDigit(7);
}

void displayDigit(byte d) {
    digit = 0;
    for (int i = 0; i <= 7; i++) {
        if (nums[d][i] == 1) bitSet(digit, i);
    }
    digitalWrite(latchPin, LOW);
    shiftOut(dataPin, clockPin, MSBFIRST, digit);
    digitalWrite(latchPin, HIGH);
}

void loop() {}

```

Wyświetlacz 7 segmentowy 4-cyfrowy TM1637



Biblioteka TM1637 do obsługi wyświetlacza z Arduino.

TM1637 może być podłączony do dowolnych pinów mikrokontrolera.

TM1637 by Avishay
Orpaz...

1.2.0 installed

Driver for 4 digit 7-segment display modules, based on the TM1637 chip. These chips can...
[More info](#)

Połączenie pinów z Arduino

TM1637	GND	VCC	DIO	CLK
ARDUINO	GND	5V	3	2

Inicjalizacja kodu

```
#include <TM1637Display.h>
#define CLK 2
#define DIO 3
TM1637Display display(CLK, DIO);
```

Podstawowe instrukcje

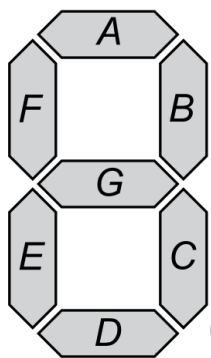
```
display.setBrightness(1); // 0-7 jasność wyświetlacza
display.clear(); // czyszczenie ekranu
display.showNumberDec(0, false); // __0
display.showNumberDec(1, true); // 0001
display.showNumberHexEx(0xf1af); // f1Af
display.showNumberDecEx(1234, 64); //12:34
```

Tablica z danymi do wyświetlenia

```
uint8_t data[] = { 0xff, 0xff, 0xff, 0xff };  
data[0] = display.encodeDigit(10); //A  
data[1] = display.encodeDigit(12); //C  
data[2] = display.encodeDigit(0); //0  
data[3] = display.encodeDigit(4); //4  
display.setSegments(data);
```

Tworzenie własnego segmentu za pomocą zdefiniowanych w bibliotece stałych.

Wiersze odpowiadają kolejnym cyfrom wyświetlacza.



```
uint8_t customSegment[] = {  
    SEG_A | SEG_D,  
    SEG_A | SEG_G | SEG_D,  
    SEG_B | SEG_C | SEG_E | SEG_F,  
    SEG_A | SEG_B | SEG_C | SEG_D  
};  
  
display.setSegments(customSegment);
```

Komunikacja z TM6137 bez biblioteki

```
#define CLK 2           // Pin zegara
#define DIO 3           // Pin danych
const int pulseDelay = 5; // Czas opóźnienia w mikrosekundach
int counter = 0;

void start() {
    pinMode(CLK, OUTPUT);
    pinMode(DIO, OUTPUT);
    digitalWrite(CLK, HIGH);
    digitalWrite(DIO, HIGH);
    delayMicroseconds(pulseDelay);
    digitalWrite(DIO, LOW);
}

void stop() {
    pinMode(CLK, INPUT);
    pinMode(DIO, INPUT);
}

void writeByte(byte data) {
    for (byte i = 0; i < 8; i++) {
        digitalWrite(CLK, LOW);
        digitalWrite(DIO, data & 0x01);
        delayMicroseconds(pulseDelay);
        digitalWrite(CLK, HIGH);
        delayMicroseconds(pulseDelay);
        data >>= 1;
    }
    // Odczytanie bitu ACK
    pinMode(DIO, INPUT);
    digitalWrite(CLK, LOW);
    delayMicroseconds(pulseDelay);
    digitalWrite(CLK, HIGH);
    delayMicroseconds(pulseDelay);
    pinMode(DIO, OUTPUT);
}

void sendCommand(byte command) {
    start();
    writeByte(command);
    stop();
}
```

```

void displayByte(byte address, byte data) {
    sendCommand(0x44); // Ustawienie adresu automatycznego
    start();
    writeByte(0xC0 | address); // Ustawienie adresu startowego
    writeByte(data);
    stop();
}

void setBrightness(byte brightness) {
    sendCommand(0x88 | brightness); // Ustawienie jasności
}

const byte digitToSegment[] = {
    0b00111111, // 0
    0b00000110, // 1
    0b01011011, // 2
    0b01001111, // 3
    0b01100110, // 4
    0b01101101, // 5
    0b01111101, // 6
    0b00000111, // 7
    0b01111111, // 8
    0b01101111 // 9
};

void displayNumber(int num) {
    displayByte(0, digitToSegment[num / 1000 % 10]);
    displayByte(1, digitToSegment[num / 100 % 10]);
    displayByte(2, digitToSegment[num / 10 % 10]);
    displayByte(3, digitToSegment[num % 10]);
}

void setup() {
    setBrightness(0x0F); // Maksymalna jasność
}

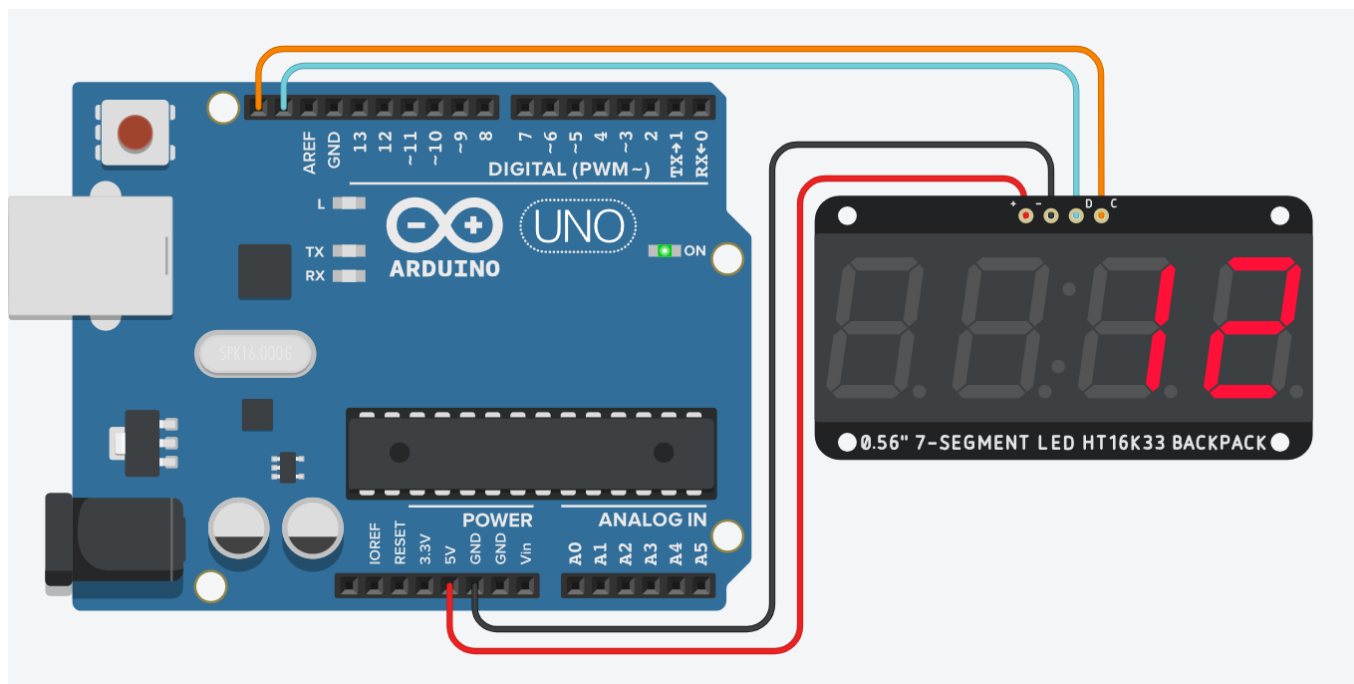
void loop() {
    displayNumber(counter);
    delay(100);
    counter++;
}

```

Wyświetlacz 7 segmentowy 4-cyfrowy I2C HT16K33

Połączenie pinów z Arduino I2C

LED	GND	VCC	SDA	SCL
ARDUINO	GND	5V	A4	A5



Rys. 7 Połączenie wyświetlacza z Arduino

Wyświetlacz współpracuje z biblioteką Adafruit_LEDBackpack

```
#include <Adafruit_LEDBackpack.h>
Adafruit_7segment matrix = Adafruit_7segment();
void setup(){
  matrix.begin(112); //Adres urządzenia
}
```

Adafruit LED
Backpack Library by...

1.5.1 installed

Adafruit LED Backpack Library
for our 8x8 matrix and 7-
segment LED backpacks...
[More info](#)

Podstawowe instrukcje

```
matrix.print(9999, DEC);  
matrix.print(0xB4EF, HEX); //hex  
matrix.print(1.234); //float  
matrix.print("7SEA"); //string  
matrix.writeDigitNum(4, 6, false); //position, number, dot  
matrix.writeDisplay();
```

Komunikacja z HT16K33 I2C bez biblioteki

```
#include <Wire.h>
#define HT16K33_ADDRESS 0x70

void sendCommand(uint8_t command) {
    Wire.beginTransmission(HT16K33_ADDRESS);
    Wire.write(command);
    Wire.endTransmission();
}

void sendData(uint8_t address, uint8_t data) {
    Wire.beginTransmission(HT16K33_ADDRESS);
    Wire.write(address);
    Wire.write(data);
    Wire.endTransmission();
}

const byte digitToSegment[] = {
    //pierwszy bit to kropka
    0b00111111, // 0
    0b00000110, // 1
    0b01011011, // 2
    0b01001111, // 3
    0b01100110, // 4
    0b01101101, // 5
    0b01111101, // 6
    0b00000111, // 7
    0b01111111, // 8
    0b01101111 // 9
};

void setup() {
    Wire.begin(); // Inicjalizacja interfejsu I2C
    delay(100);   // Oczekiwanie na stabilizację
    sendCommand(0x21); // Włącz oscylator
    sendCommand(0x81); // Wyłącz migotanie
    sendCommand(0xEF); // Ustaw jasność E0 - EF

    //rejestry: 0x00, 0x02, 0x04, 0x06, 0x08
    sendData(0x00, digitToSegment[2]);
    sendData(0x02, digitToSegment[1]);
    sendData(0x04, 0b11111111); //włącz dwukropek
    //sendData(0x04, 0b0); //wyłącz dwukropek
    sendData(0x06, digitToSegment[4]);
    sendData(0x08, digitToSegment[5]);
}
```

```
}
```

```
void loop() {}
```

SSD1306 OLED Display I2C



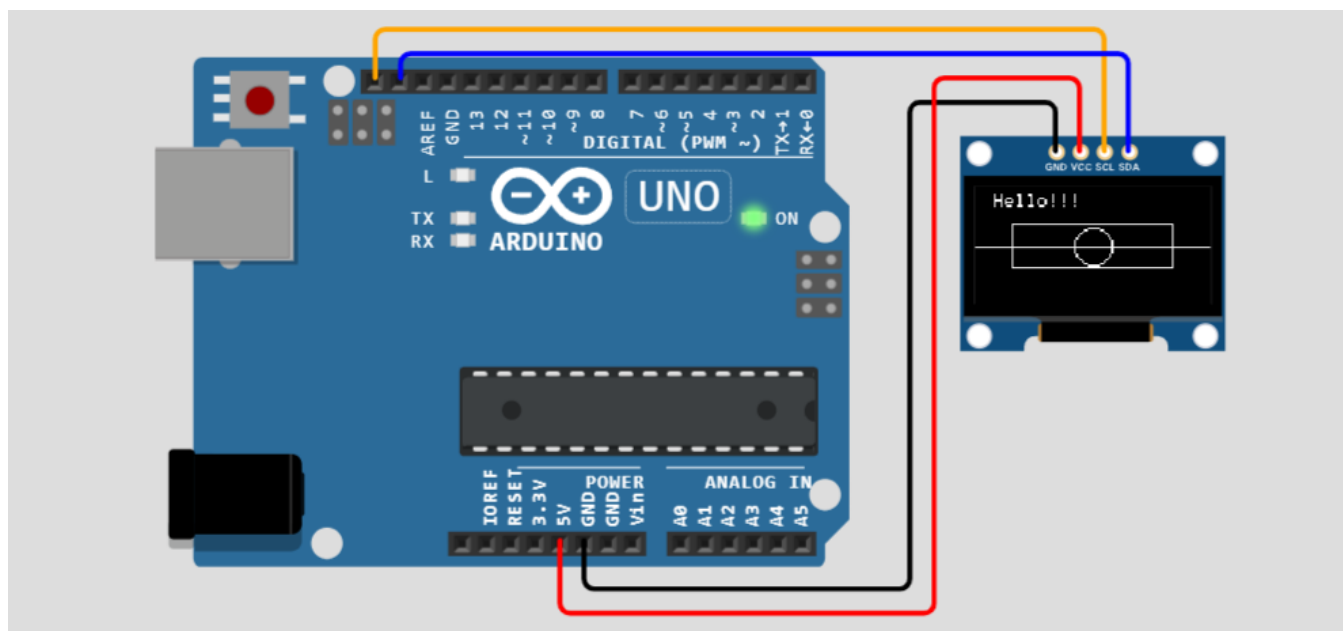
Monochromatyczny 128x64 OLED wyświetlacz z interfejsem I2C

Wyświetlaczem można sterować za pomocą kilku bibliotek, np. Adafruit_SSD1306

Układ współrzędny ekranu ma swój początek (0,0) w lewym, górnym rogu.

Połączenie pinów z Arduino

LED	GND	VCC	SDA	SCL
ARDUINO	GND	5V	A4	A5



Wielkość czcionki w pixelach:

Szerokość: 5 pikseli (plus 1 piksel odstępu) * fontsize

Wysokość: 7 pikseli (plus 1 piksel odstępu) * fontsize

SSD1306 OLED Display z biblioteką Adafruit

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SCREEN_WIDTH 128    // Szerokość wyświetlacza w pikselach
#define SCREEN_HEIGHT 64    // Wysokość wyświetlacza w pikselach
#define OLED_RESET -1       // Reset pin nie jest używany dla I2C
#define SCREEN_ADDRESS 0x3C // Adres I2C wyświetlacza SSD1306
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

void setup() {
  Wire.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;); // Zatrzymanie programu, jeśli inicjalizacja się nie powiedzie
  }
  display.clearDisplay(); // Wyczyszczenie bufora wyświetlacza
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.println(F("Hello, world!"));
  display.drawLine(0, SCREEN_HEIGHT/2, SCREEN_WIDTH, SCREEN_HEIGHT/2,
SSD1306_WHITE); //x0,y0,x1,y1,color
  display.drawRect(20, 20, SCREEN_WIDTH-40, SCREEN_HEIGHT-40,
SSD1306_WHITE); //x0,y0,w,h,color
  display.drawCircle(SCREEN_WIDTH/2, SCREEN_HEIGHT/2, 10, SSD1306_WHITE);
//x0,y0,r,color
  display.setCursor(0, 56); // position to display
  display.print("Bye, bye!"); // text to display
  display.display();
}

void loop() {}
```

UTFT lib. <http://www.rinkydinkelectronics.com/library.php?id=79>

To kolejna biblioteka, za pomocą której można sterować ekranem I2C OLED.

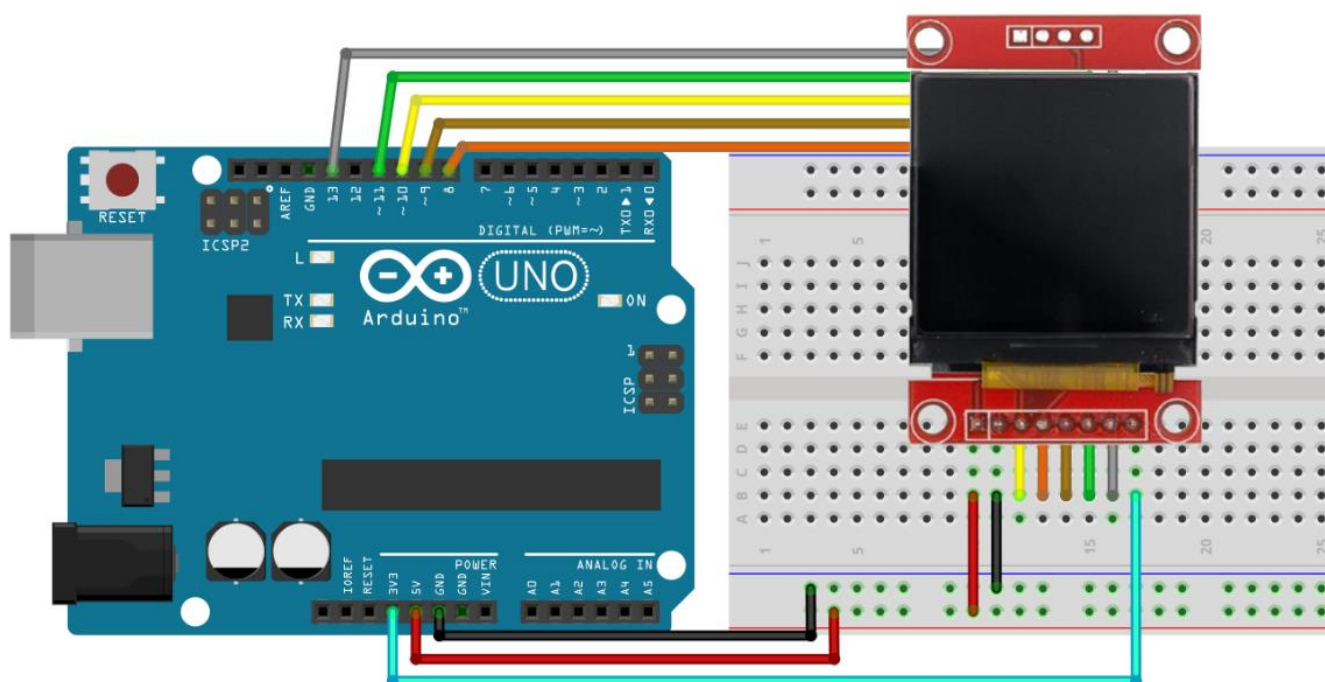
TFT SPI LCD 128*160 MSP1803

Screen: 1.8" 65K Color

Driver IC: ST7735S

Połączenie pinów z Arduino

TFT	VCC	GND	CS	RESET	A0	SDA	SCK	LED
ARDUINO	5V	GND	10	8	9	11	13	3.3V



Inicjalizacja kodu

```
#include <TFT.h>
#include <SPI.h>
#define cs 10
#define dc 9
#define rst 8

TFT TFTscreen = TFT(cs, dc, rst);
char sensorPrintout[4];

void setup() {
    TFTscreen.begin();
    TFTscreen.background(0, 0, 0);
    TFTscreen.stroke(0, 122, 255);
    TFTscreen.setTextSize(2);
    TFTscreen.text("Sensor Value :\n ", 0, 0);
    TFTscreen.setTextSize(5);
}
```

Podstawowe instrukcje

```
void loop() {
    int potVal = map(analogRead(A5),0, 1023, 0, 100);
    String sensorVal = String(potVal);
    sensorVal.toCharArray(sensorPrintout, 4);
    TFTscreen.stroke(0, 0, 255);
    TFTscreen.text(sensorPrintout, 0, 20);
    delay(500);
    TFTscreen.stroke(0, 0, 0);
    TFTscreen.text(sensorPrintout, 0, 20);
}
```

TFT SPI z PBC ILI9341 Touchscreen 320*240



Screen: 2.8" 65K Color

Driver IC: ILI9341 / ST7789V

IO logic: 3.3V

Połączenie LCD z Arduino

TFT	VCC	GND	CS	RESET	A0[DC]	MOSI	SCK	LED	MISO
ARDUINO	5V	GND	10	8	9	11	13	3.3V	12

```

#include <Adafruit_GFX.h>
#include "Adafruit_ILI9341.h"
#define TFT_DC 9
#define TFT_CS 10
#define TFT_RST 8
#define TFT_MISO 12
#define TFT_MOSI 11
#define TFT_CLK 13

Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_MOSI, TFT_CLK, TFT_RST,
TFT_MISO);
#include <Fonts/FreeSerif24pt7b.h> // Add a custom font
int Variable1; // Create a variable to have something dynamic
to show on the display

void setup() {
  tft.begin();
  tft.fillScreen(0x0000); // Fill screen with black
  tft.setTextWrap(false);
  tft.setCursor(0, 30); // Set position (x,y)
  tft.setTextColor(0xFFFF); // Set color of text. First is the color of text and
after is color of background
  tft.setTextSize(4);
  tft.println("ASTRAJA");
  tft.drawRect(0, 110, 110, 60, 0x07FF); // Draw rectangle (x,y,width,height,color)
}

```

```

void loop() {
  Variable1++;
  if (Variable1 > 150) {
    Variable1 = 0;
  }
  char string[10]; // Create a character array of 10 characters
  // Convert float to a string:
  dtostrf(Variable1, 3, 0, string); // (<variable>,<amount of digits we are going
to use>,<amount of decimal digits>,<string name>)
  tft.setCursor(21, 125);           // Set position (x,y)
  tft.setTextColor(0xFFE0, 0x0000); // Set color of text. First is the color of
text and after is color of background
  tft.setTextSize(4);               // Set text size. Goes from 0 (the smallest) to
20 (very big)
  tft.println(Variable1);           // Print a text or value
  if (Variable1 < 10) // If Variable1 is less than 10...
  {
    // Fill the other digit with background color:
    tft.fillRect(44, 124, 24, 34, 0x0000); // Draw filled rectangle
(x,y,width,height,color)
  }
  if (Variable1 < 100) // If Variable1 is less than 100...
  {
    // Fill the other digit with background color:
    tft.fillRect(69, 124, 24, 34, 0x0000); // Draw filled rectangle
(x,y,width,height,color)
  }
}
}

```

```

#include "Adafruit_ILI9341.h"
#define TFT_DC 9
#define TFT_CS 10
#define TFT_RST 8
#define TFT_MISO 12
#define TFT_MOSI 11
#define TFT_CLK 13
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_MOSI, TFT_CLK, TFT_RST,
TFT_MISO);
int counter = 0;

void setup() {
    tft.begin();
    tft.setRotation(0);
}
void display() {
    tft.fillScreen(ILI9341_BLACK);
    tft.setTextColor(ILI9341_RED);
    tft.setTextSize(2);
    tft.setCursor(1, 20);
    tft.print("ASTRAJA");

    tft.setTextColor(ILI9341_GREEN);
    tft.setTextSize(3);
    tft.setCursor(1, 120);
    tft.print("COUNTER:");

    tft.setTextColor(ILI9341_BLUE);
    tft.setTextSize(6);
    tft.setCursor(1, 180);
    tft.print(counter);
    counter++;
    delay(100);
}
void loop() {
    display();
}

```

Połączenie touch screen:

TFT	T_CLK	T_CS	T_DIN	T_DO	T_IRQ
ARDUINO	3	4	5	6	7

Biblioteka URTouch:

http://www.rinkydinkelectronics.com/library.php?id=92#google_vignette

```
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9341.h"
#include "URTouch.h"
#define TFT_DC 9
#define TFT_CS 10
#define TFT_RST 8
#define TFT_MISO 12
#define TFT_MOSI 11
#define TFT_CLK 13
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_MOSI, TFT_CLK, TFT_RST,
TFT_MISO);

#define t_SCK 3
#define t_CS 4
#define t_MOSI 5
#define t_MISO 6
#define t_IRQ 7
URTouch ts(t_SCK, t_CS, t_MOSI, t_MISO, t_IRQ);
```

```

void setup(){
    tft.begin();
    tft.setRotation(3);
    ts.InitTouch();
    ts.setPrecision(PREC_EXTREME);
    tft.fillScreen(ILI9341_BLACK);
    tft.setTextColor(ILI9341_RED);
    tft.setTextSize(2);
    tft.setCursor(85,5);
    tft.print("Touch Demo");
}

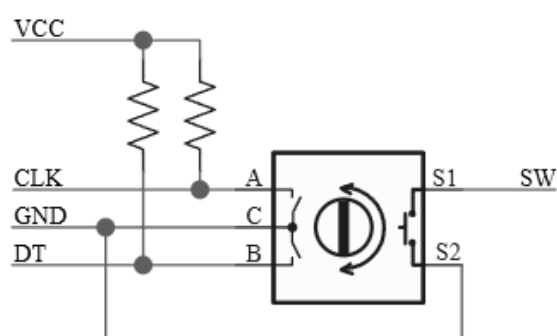
void loop(){
    int x, y;
    while(ts.dataAvailable())
    {
        ts.read();
        x = ts.getX();
        y = ts.getY();
        if((x!=-1) && (y!=-1))
        {
            x += 13;
            y += 4;
            int radius = 4;
            tft.fillCircle(x, y, radius, ILI9341_YELLOW);
        }
    }
}

```



Encoder przypomina wyglądem potencjometr, podczas obrotu wysyłany jest sygnał (CLK, DT) identyfikujący zmiany położenia (kąt i kierunek obrotu).

Encodery cechują się rozdzielczością, np. 100 impulsów na obrót, oznacza to, że w czasie jednego obrotu zliczy 100 impulsów. Dzięki temu można obliczyć kąt obrotu, tj. $360^\circ / 100 = 3,6^\circ$. Encodery mogą posiadać przycisk (SW).



Rys. 8 Połączenie enkodera z rezystorami pullup

Rodzaje encoderów:

Encodery przyrostowe (incremental encoders):

Wydają impulsy za każdym razem, gdy obracane jest pokrętko.

Nie podają bezpośrednio aktualnej pozycji – do określenia pozycji wymagają liczenia impulsów.

Posiadają dwa wyjścia: A i B, które generują sygnały przesunięte w fazie o 90° .

Encodery absolutne (absolute encoders):

Podają dokładną pozycję wału w jednym obrocie, co pozwala określić położenie bez konieczności zliczania impulsów.

Często są bardziej złożone i droższe od enkoderów przyrostowych.

```

#define CLK 3
#define DT 4
#define SW 2
int counter = 0;

void setup() {
    pinMode(CLK, INPUT);
    pinMode(DT, INPUT);
    pinMode(SW, INPUT_PULLUP);
    Serial.begin(9600);
    attachInterrupt(digitalPinToInterrupt(SW), resetCounter, FALLING);
}

void loop() {
    readEncoder();
}

void resetCounter(){
    counter = 0;
    Serial.println(counter);
}

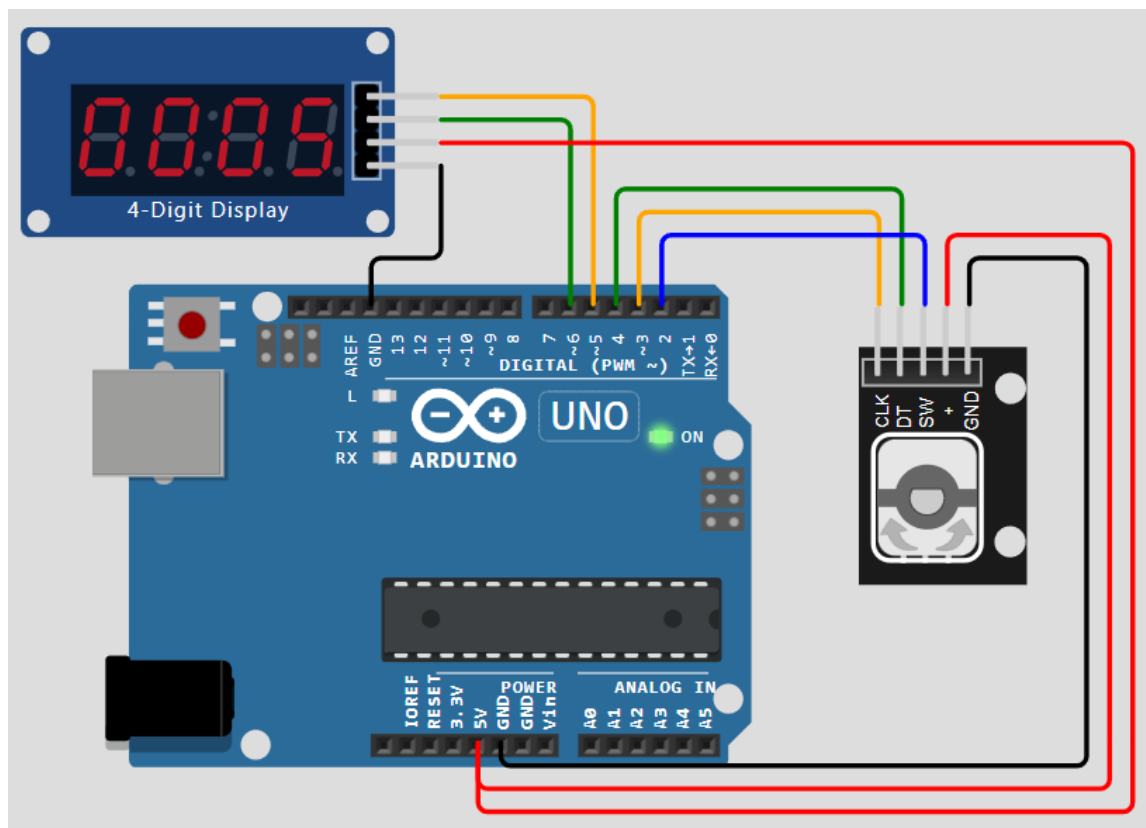
void readEncoder(){
    static int lastStateCLK = digitalRead(CLK);
    int currentStateCLK = digitalRead(CLK);
    if (currentStateCLK != lastStateCLK && currentStateCLK == 1) {
        if (digitalRead(DT) != currentStateCLK) {
            counter++;
        } else {
            counter--;
        }
        Serial.println(counter);
    }
    lastStateCLK = currentStateCLK;
}

```

Timer z encoderem i wyświetlaczem

Program ustawiający timer za pomocą encodera i odliczający czas wyświetlany na wyświetlaczu TM1637

<https://wokwi.com/projects/419161032688634881>



```
class Astratimer {
private:
    int _countDelay;
    unsigned long _previousMillis;
    int _counter;

public:
    Astratimer(int countDelay) {
        _countDelay = countDelay;
        _previousMillis = 0;
        _counter = 0;
    }
    bool canExecute() {
        if (millis() - _previousMillis >= _countDelay) {
            _previousMillis = millis();
            return true;
        } else {
            return false;
        }
    };
};
```

```

#define CLK 3
#define DT 4
#define SW 2
#include <TM1637Display.h>
#define CLK_DISPLAY 5
#define DIO 6
TM1637Display display(CLK_DISPLAY, DIO);
Astratimer tim1(1000);
int counter = 0;
int interval = 20;
volatile int menuState = -1;

void displayNumber(int num){
    display.showNumberDec(num, true);
}

void setup() {
    pinMode(CLK, INPUT);
    pinMode(DT, INPUT);
    pinMode(SW, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(SW), toggleMenu, FALLING);

    display.setBrightness(1);
    display.clear();
}

void toggleMenu(){
    menuState *= -1;
}

void countDown(){
    if(counter > 0){
        counter--;
    }else{
        counter = interval;
    }
    displayNumber(counter);
}

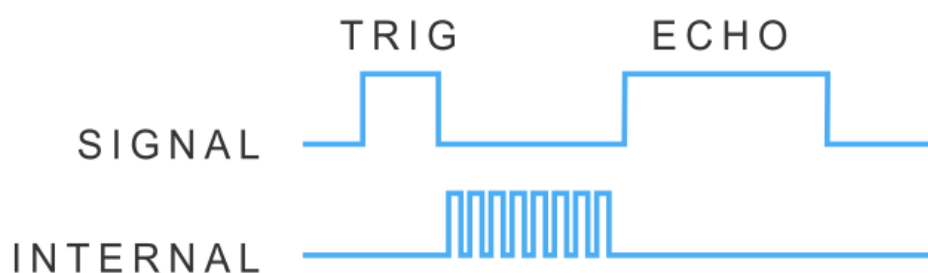
void loop() {}

```



Sensor ultrasoniczny używa fali ultradźwiękowej o częstotliwości 40kHz do pomiaru odległości w zakresie 2 – 400cm.

Szerokość pulsu wynosi 10us.



Zasada działania sensora:

TRIG otrzymuje stan wysoki przez 10us, to inicjuje wysłanie 8 cykli fali ultradźwiękowej z częstotliwością 40kHz.

Po otrzymaniu fali powrotnej, ECHO ustawiony zostaje na stan wysoki i zaczyna zliczać czas za pomocą funkcji pulseIn, co docelowo przełoży się na odległość.

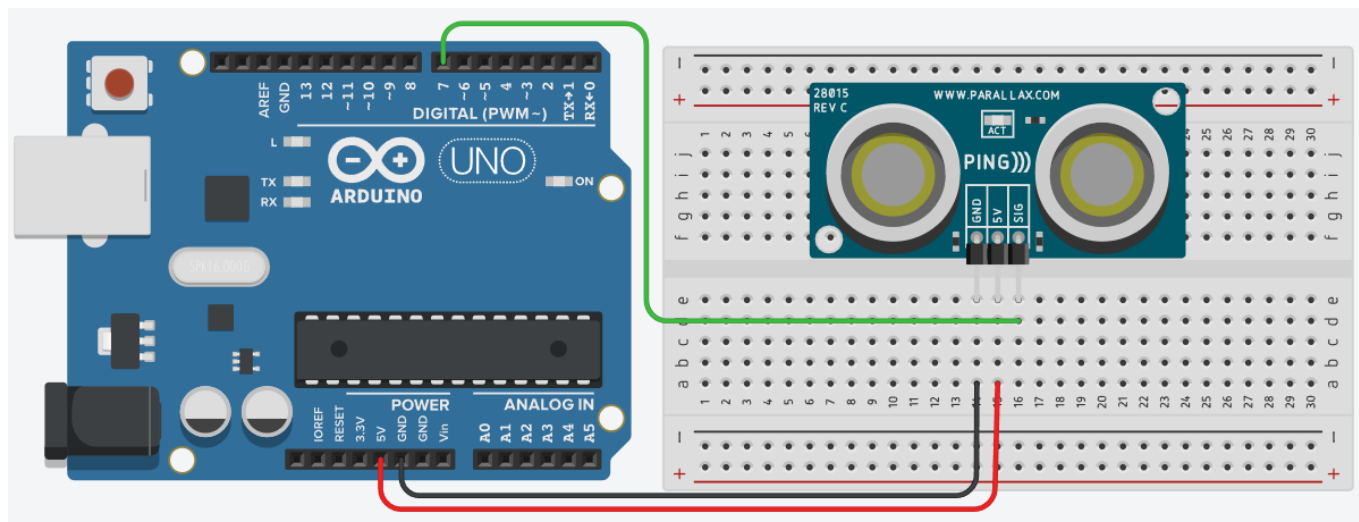
W przypadku, gdy fala nie wróci w ciągu 38ms, oznacza to brak przeszkód, co powoduje przełączenie pinu ECHO na stan LOW.

Prędkość dźwięku = 340m/s = 0.0343 cm/us

odległość = (czas echo * prędkość dźwięku) / 2

```
duration = pulseIn(echoPin, HIGH);  
distance = (duration * 0.0343)/2; //fala porusza się w 2 strony
```

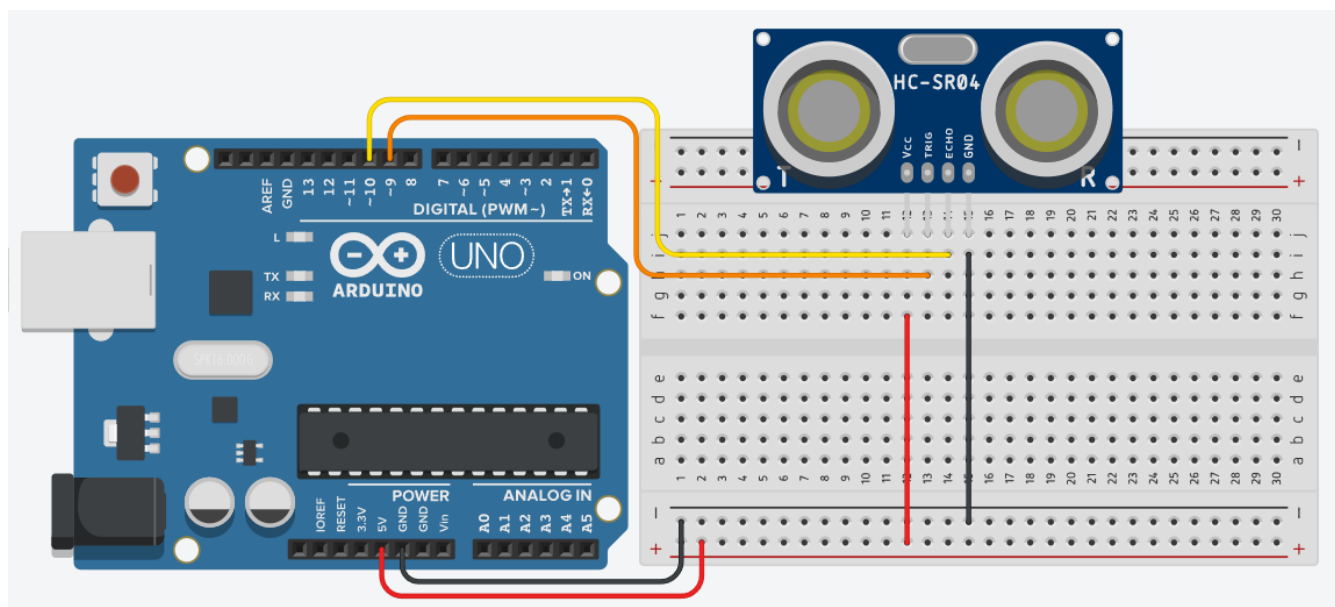
Czujnik ultradźwiękowy z 3 pinami



```
int cm = 0;
```

```
long readUltrasonicDistance(int triggerPin, int echoPin) {  
    pinMode(triggerPin, OUTPUT);  
    digitalWrite(triggerPin, LOW);  
    delayMicroseconds(2);  
    digitalWrite(triggerPin, HIGH);  
    delayMicroseconds(10);  
    digitalWrite(triggerPin, LOW);  
    pinMode(echoPin, INPUT);  
    return pulseIn(echoPin, HIGH);  
}  
void setup() {  
    Serial.begin(9600);  
}  
void loop() {  
    cm = 0.01723 * readUltrasonicDistance(7, 7);  
    Serial.print(cm);  
    Serial.println("cm");  
    delay(100);  
}
```

Czujnik ultradźwiękowy HC-SR04 z 4 pinami



```
int trigPin = 9, echoPin = 10;
float duration, distance;

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Serial.begin(9600);
}

void loop() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distance = (duration * 0.0343)/2;
  Serial.print("Distance: ");
  Serial.println(distance);
  delay(100);
}
```

Wyświetlenie odległości z biblioteką NewPing

```
#include <NewPing.h>
#define TRIGGER_PIN_0 8
#define ECHO_PIN_0 8
#define MAX_DISTANCE 400
NewPing sensor0(TRIGGER_PIN_0, ECHO_PIN_0, MAX_DISTANCE);
float distance0;

void setup() {
  Serial.begin(9600);
}
void loop() {
  distance0 = sensor0.ping_cm();
  delay(20);
  Serial.print("Distance 0 = ");
  if (distance0 >= 400 || distance0 <= 2) {
    Serial.println("Out of range");
  } else {
    Serial.print(distance0);
    Serial.println(" cm");
    delay(500);
  }
  delay(500);
}
```



Budowa i działanie silnika DC

Stojan: Nieruchoma część silnika, która tworzy pole magnetyczne. Może być wykonany z magnesów trwałych (w silnikach szczotkowych) lub cewek (w silnikach bezszczotkowych).

Wirnik (rotor): Obracająca się część silnika, zawierająca uzwojenia przewodzące prąd.

Komutator: Mechanizm, który przełącza kierunek prądu w uzwojeniach wirnika, znajdujący się w silnikach szczotkowych. W silnikach bezszczotkowych jest to realizowane elektronicznie.

Szczotki: Elementy w silnikach szczotkowych, które dostarczają prąd do wirnika poprzez komutator.

Parametry silnika

Moment obrotowy to siła obrotowa generowana przez silnik wyrażana w Nm lub $\text{kg}\cdot\text{cm}$.

Np. silnik z momentem $4\text{kg}\cdot\text{cm}$ będzie w stanie obrócić 4kg w odległości 1cm od środka wirnika lub

2kg w odległości 2cm,

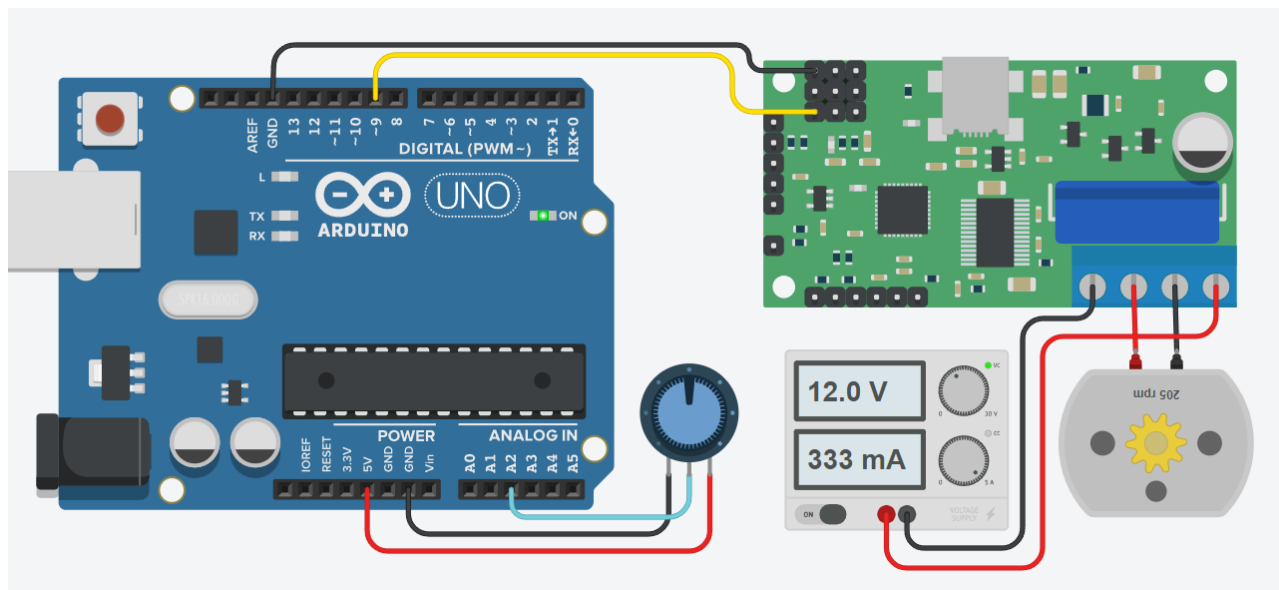
1kg w odległości 4cm, itd...

Prędkość obrotowa (RPM) to liczba obrotów na minutę.

Moc wyjściowa (W) to moc mechaniczna generowana przez silnik (iloczyn momentu obrotowego i prędkości obrotowej).

Sterowanie silnikiem z kontrolerem Pololu

Potencjometr steruje kierunkiem i prędkością silnika DC



```
#include <Servo.h>
Servo motor;
#define POT_PIN 2
#define PWM_PIN 9

void setup(){
    motor.attach(PWM_PIN);
}

void loop(){
    int potValue = analogRead(POT_PIN);
    int speed = map(potValue, 0, 1023, 0, 180);
    if(speed > 88 && speed < 92){
        motor.write(LOW);
    }
    motor.write(speed);
    delay(200);
}
```

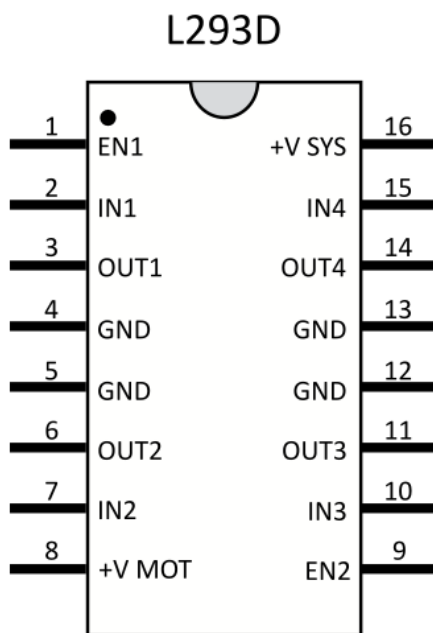
L293D H Bridge Motor Driver

L293D to układ IC do sterowania silnikami DC.

Liczba kanałów: 2

Napięcie zasilania: < 36V

Prąd na kanał (chwilowy): 600mA (1.2A)



Do układu można podłączyć 4 silniki w trybie half-bridge lub 2 silniki w trybie full-bridge.

Tryb half-bridge (jedno wyjście OUT) daje możliwość ruchu w jednym kierunku z prędkością ustaloną przez pin wspólny EN1.

Tryb full-bridge daje pełną kontrolę w dwóch kierunkach. przy natężeniu ciągłym 600mA i chwilowym 1.2A.

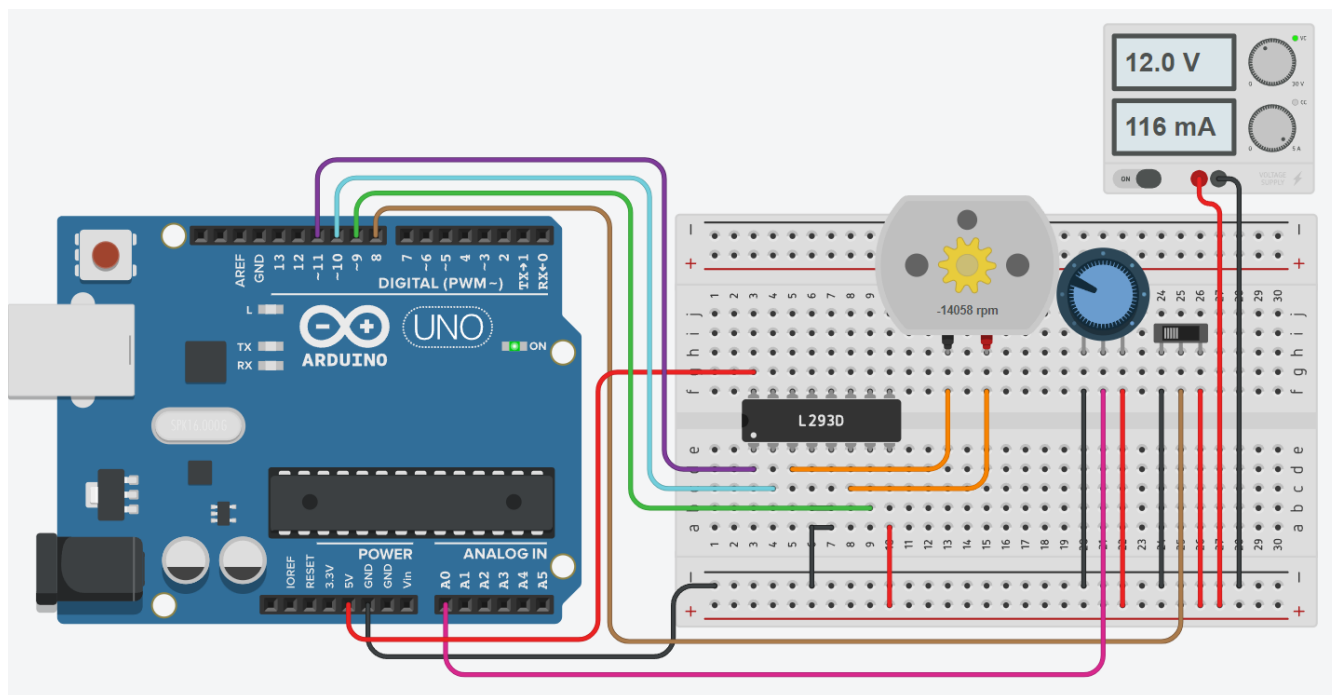
Napięcie pinu 16 (zasilania części logicznej) powinno wynosić 5V.

```
// Przykład: Sterowanie silnikiem za pomocą L293D
int enablePin = 11;
int in1Pin = 10;
int in2Pin = 9;
int switchPin = 8;
int potPin = A0;

void setup(){
    pinMode(in1Pin, OUTPUT);
    pinMode(in2Pin, OUTPUT);
    pinMode(enablePin, OUTPUT);
    pinMode(switchPin, INPUT_PULLUP);
}

void SetMotor(int speed, boolean dir){
    analogWrite(enablePin, speed);
    digitalWrite(in1Pin, !dir);
    digitalWrite(in2Pin, dir);
}

void loop(){
    int speed = map(analogRead(potPin), 0, 1023, 0, 255);
    boolean dir = digitalRead(switchPin);
    SetMotor(speed, dir);
}
```

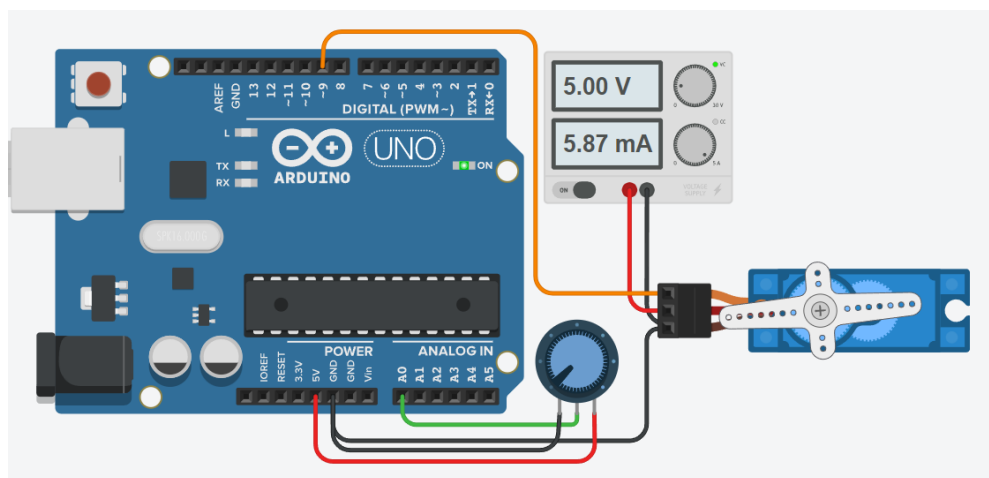




Silniki serwo steruje się za pomocą sygnału PWM.
Współczynnik wypełnienia decyduje o kącie obrotu serwa.

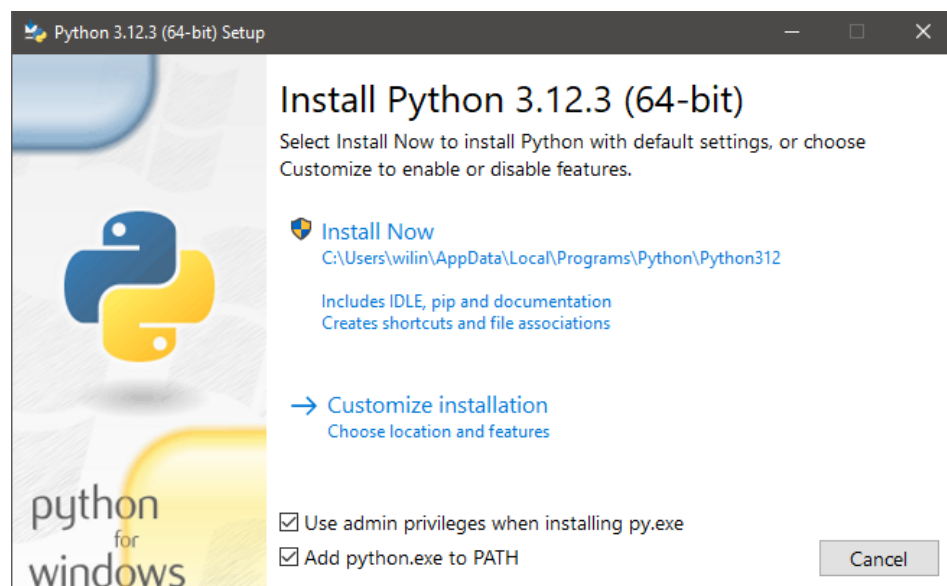
```
// Przykład: Zmiana obrotu serwa za pomocą potencjometru
#include <Servo.h>
int servoPin = 9;
int pos;
Servo servo1;

void setup(){
    servo1.attach(servoPin);
    Serial.begin(9600);
}
void loop(){
    int pot = analogRead(A0);
    pos = map(pot, 0, 1023, 0, 180);
    servo1.write(pos);
    Serial.println(servo1.read());
}
```



Przygotowanie środowiska python

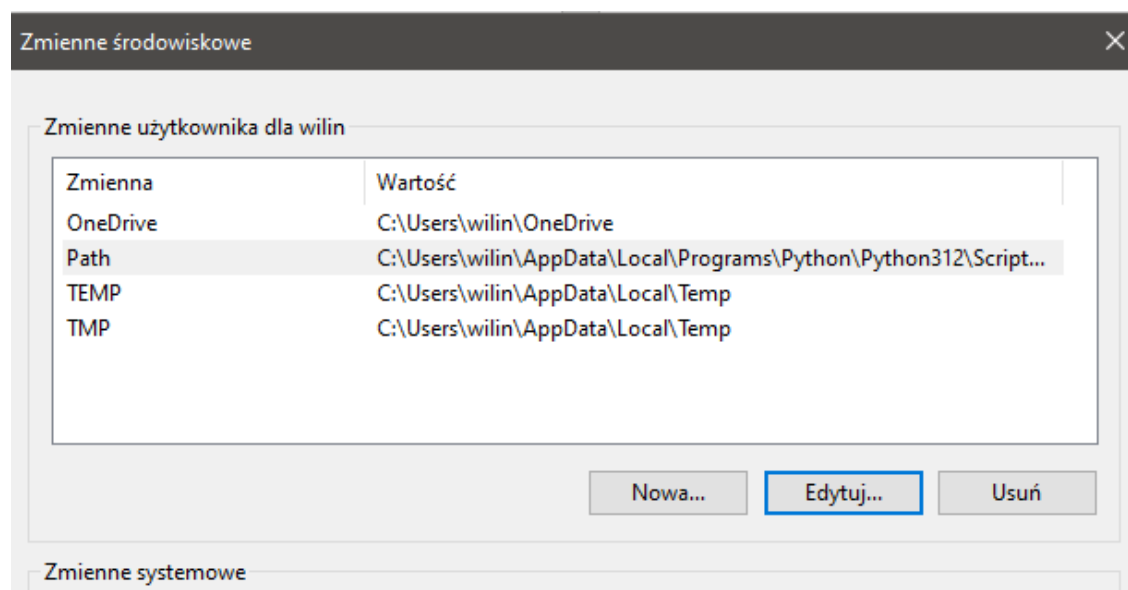
Pobierz i zainstaluj [python](#) (zaznacz Add python.exe to PATH) oraz edytor [Visual Studio Code](#)



W ustawieniach Zmienne Środowiskowe powinny być dodane poniższe ścieżki (uwzględniając zainstalowaną wersję):

C:\Users\wilin\AppData\Local\Programs\Python\Python312\Scripts\

C:\Users\wilin\AppData\Local\Programs\Python\Python312\



Zainstaluj bibliotekę serial przez pip: **pip install pyserial**

Wysyłanie danych z komputera do Arduino

Arduino

```
const int ledPin = 13;
char inData;

void setup() {
    Serial.begin(9600);
    pinMode(ledPin, OUTPUT);
}

void loop() {
    if (Serial.available() > 0) {
        inData = Serial.read();
        if (inData == 'H') {
            digitalWrite(ledPin, HIGH);
        }
        if (inData == 'L') {
            digitalWrite(ledPin, LOW);
        }
    }
}
```

Python console

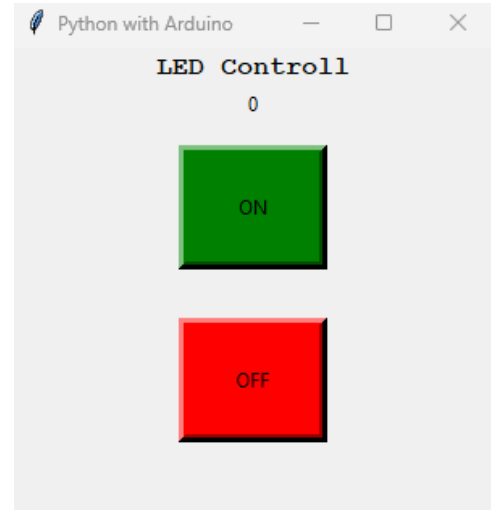
```
import serial
import time
ser = serial.Serial('com5', 9600) #podaj odpowiedni com
time.sleep(3)
def sendCommand(c):
    ser.write(c.encode())
while(True):
    my_command = input("0-off, 1-on: ")
    # my_command = input("L-off, H-on: ")
    sendCommand(my_command)
```

Wysyłanie danych za pomocą programu graficznego python tkinter

```
import serial
import time
import tkinter

def set_button1_state():
    btn_state_label['text'] = "LED ON"
    ser.write(bytes('H', 'UTF-8'))
def set_button2_state():
    btn_state_label['text'] = "LED OFF"
    ser.write(bytes('L', 'UTF-8'))

ser = serial.Serial('com3', 9600)
time.sleep(3)
root = tkinter.Tk()
root.geometry('300x300')
root.title("Python with Arduino")
label3 = tkinter.Label(text = 'LED Controll',font=("Courier", 12,'bold')).pack()
btn_state_label = tkinter.Label(text = "___")
btn_state_label.pack()
button1state = tkinter.Button(root,
    text="ON",
    command=set_button1_state,
    height = 4,
    fg = "black",
    width = 8,
    bd = 5,
    background = "green"
)
button1state.pack(side='top', ipadx=10, padx=10, pady=15)
button2state = tkinter.Button(root,
    text="OFF",
    command=set_button2_state,
    height = 4,
    fg = "black",
    width = 8,
    bd = 5,
    background='red'
)
button2state.pack(side='top', ipadx=10, padx=10, pady=15)
tkinter.mainloop()
```



Wysyłanie danych z arduino do komputera

Arduino

```
const int countDelay = 1000;
unsigned long previousMillis = 0;
int counter = 0;
void setup() {
    Serial.begin(9600);
}
void countWithDelay() {
    if (millis() - previousMillis >= countDelay) {
        previousMillis = millis();
        Serial.println(counter);
        counter++;
    }
}
void loop() {
    countWithDelay();
}
```

Python

```
import serial
import time
port = 'COM5'
baud_rate = 9600
try:
    ser = serial.Serial(port, baud_rate, timeout=1)
    time.sleep(2)
    print("Connected to:", port)
    while True:
        if ser.in_waiting > 0:
            line = ser.readline().decode('utf-8', errors='ignore').rstrip()
            print("Received:", line)
except serial.SerialException as e:
    print("Error:", e)
finally:
    ser.close()
    print("Serial connection closed")
```