Programowanie mikrokontrolerów w języku wysokiego poziomu 1

(TS1F1008)

Politechnika Białostocka - Wydział Elektryczny
Elektronika i telekomunikacja, sem. I, studia stacjonarne I stopnia
Rok akademicki 2024/2025

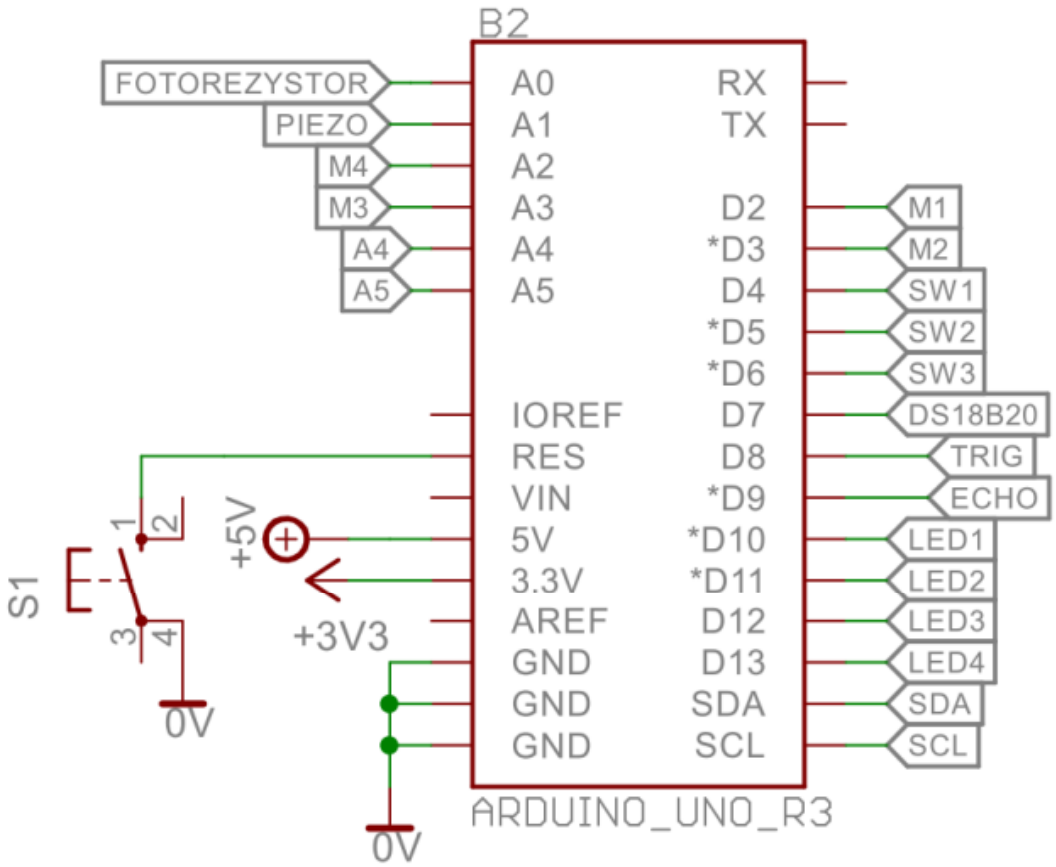
Wykład nr 6 (19.12.2024)

dr inż. Jarosław Forenc

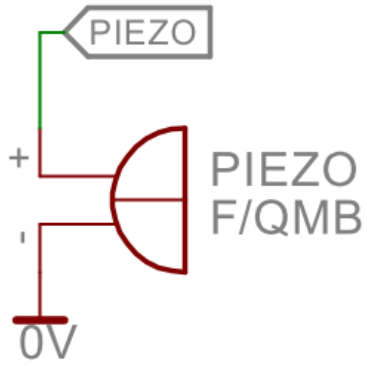
Plan wykładu nr 6

- Arduino
  - buzzer (brzęczyk piezoelektryczny)
  - fotorezystor
  - ultradźwiękowy czujnik odległości
  - czujnik natężenia światła
- Język C
  - deklaracja struktury i zmiennej strukturalnej
  - odwołania do pól struktury
  - inicjalizacja zmiennej strukturalnej
  - złożone deklaracje struktur
  - pola bitowe
  - unie

Arduino - buzzer (brzęczyk piezoelektryczny)



Schemat podłączenia
wyprowadzeń modułu Arduino



Schemat podłączenia
buzzera

Arduino - funkcja tone()

```
tone(pin, frequency, duration)
```

- Funkcja używana do generowania sygnału dźwiękowego o określonej częstotliwości na wskazanym pinie
- pin - numer pinu, na którym ma być generowany sygnał
- frequency - częstotliwość sygnału w hercach (Hz)
- duration - czas trwania sygnału w milisekundach (parametr opcjonalny), jeśli nie zostanie podany, to sygnał będzie generowany bez ograniczenia czasowego, aż do wywołania funkcji noTone()
- Tylko jeden sygnał dźwiękowy może być generowany w danym momencie
- Generuje falę kwadratową o określonej częstotliwości (i 50% współczynniku wypełnienia) na podanym pinie

Arduino - funkcja noTone()

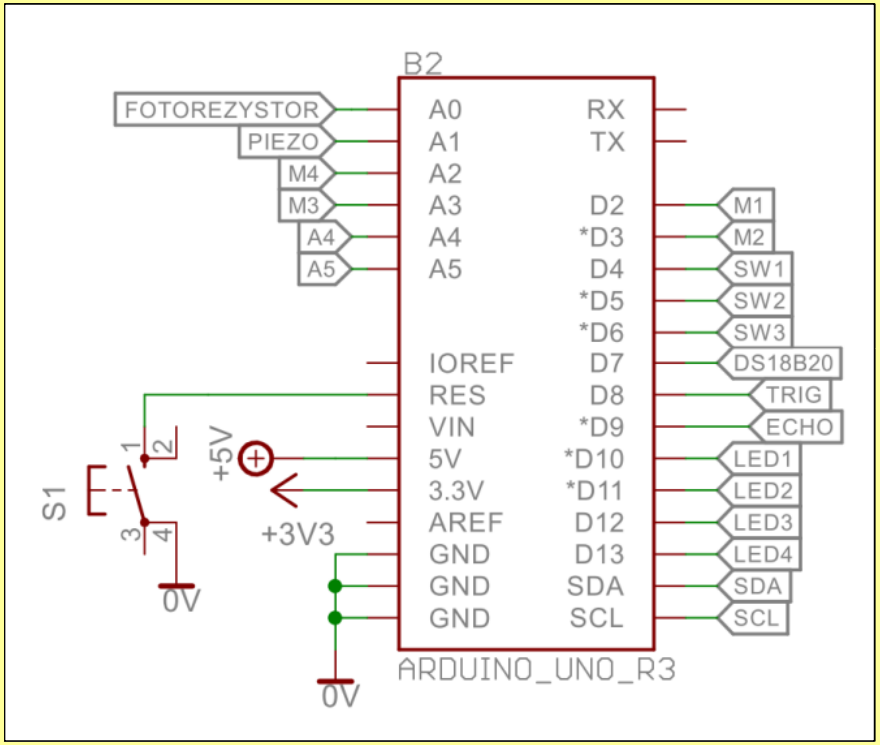
```
noTone(pin)
```

- Zatrzymuje sygnał dźwiękowy generowany przez funkcję tone() na określonym pinie
- pin - numer pinu, na którym ma być zatrzymany sygnał dźwiękowy
- Jeśli funkcja tone() nie była wcześniej wywołana dla podanego pinu, to noTone() nic nie robi

Arduino - buzzer (brzęczyk piezoelektryczny)

Program obsługujący buzzer

```
#include <Arduino.h>

#define BUZZER_PIN A1
#define SW1_PIN 4

void setup()
{
  pinMode(SW1_PIN,INPUT);
}
```

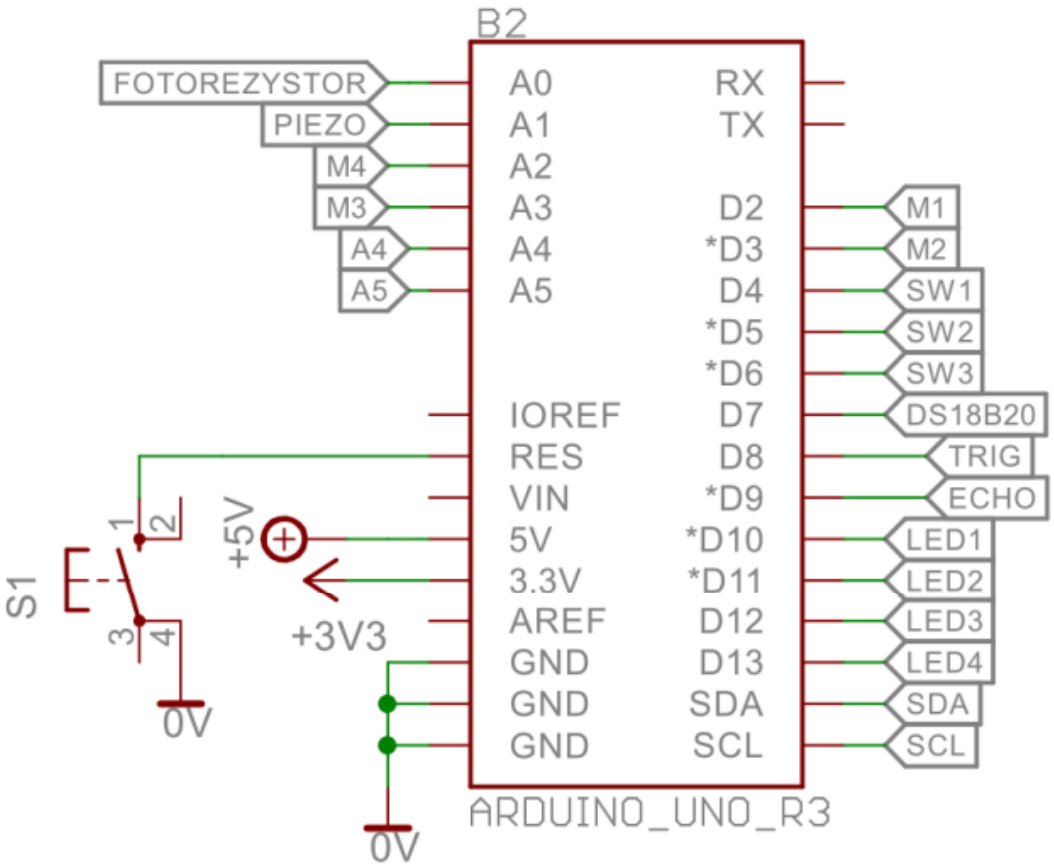


Arduino - buzzer (brzęczyk piezoelektryczny)

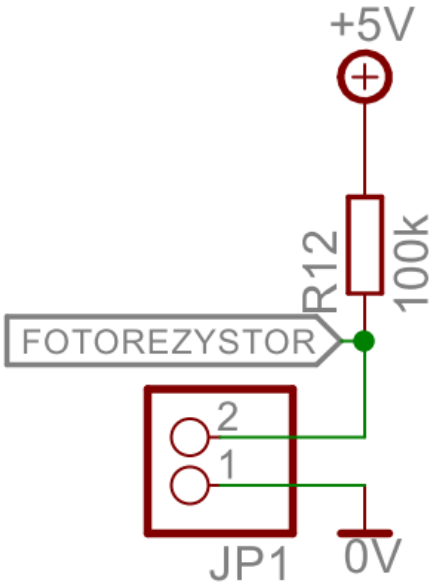
Program obsługujący buzzer

```
void loop()
{
  if(digitalRead(SW1_PIN) == LOW)
  {
    delay(200);
    tone(BUZZER_PIN,1000);
    delay(200);
    tone(BUZZER_PIN,500);
    delay(200);
    tone(BUZZER_PIN,750);
    delay(200);
    noTone(BUZZER_PIN);
  }
}
```

Arduino - fotorezystor



Schemat podłączenia
wyprowadzeń modułu Arduino



Schemat podłączenia
fotorezystora

Arduino - fotorezystor

- Fotorezystor - element półprzewodnikowy, którego rezystancja zmienia się w zależności od natężenia padającego na niego światła
- Wraz ze wzrostem natężenia oświetlenia rezystancja fotorezystora maleje
- Fotorezystor jest podłączony do analogowego wyjścia A0
- Arduino posiada 10-bitowy przetwornik analogowo-cyfrowy
- Napięcie w zakresie od 0 do 5 V doprowadzone do wejścia analogowego Arduino jest przetwarzane przez przetwornik na liczbę całkowitą z przedziału od 0 do 1023
- Liczbę tę można odczytać, wywołując funkcję analogRead()
- Do wyświetlenia jej wartości wykorzystane zostało okno monitora portu szeregowego

Arduino - funkcja analogRead()

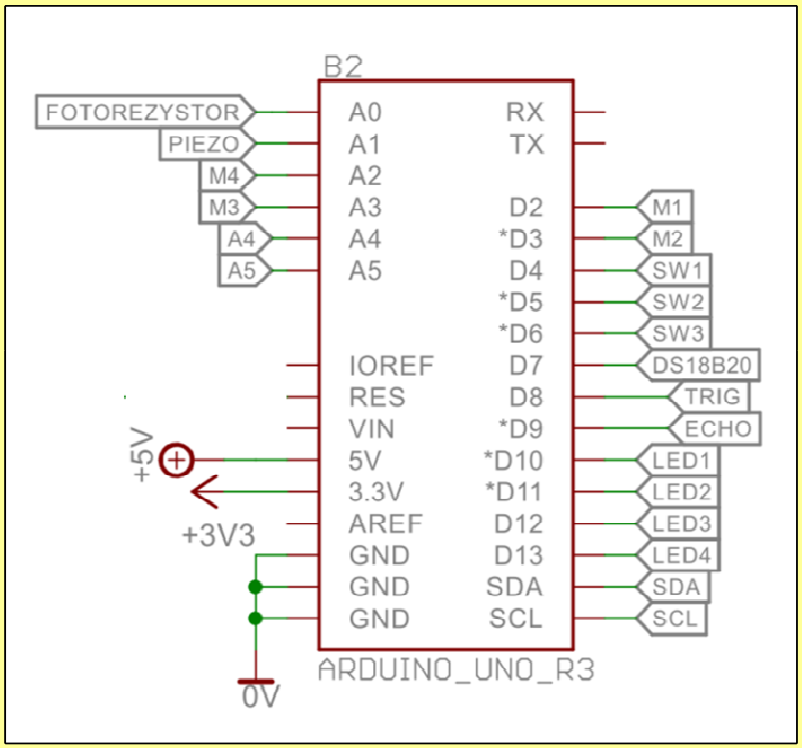
```
analogRead(pin)
```

- Odczytuje wartość analogową z jednego z pinów analogowych (A0-A5)
- pin - numer pinu analogowego, z którego odczytujemy wartość
- Zwraca wartość typu int z przedziału od 0 do 1023
- Zwrócona wartość jest proporcjonalna do zmierzonego napięcia
- Wartość 0 odpowiada 0 V, a 1023 odpowiada 5 V

Arduino - fotorezystor

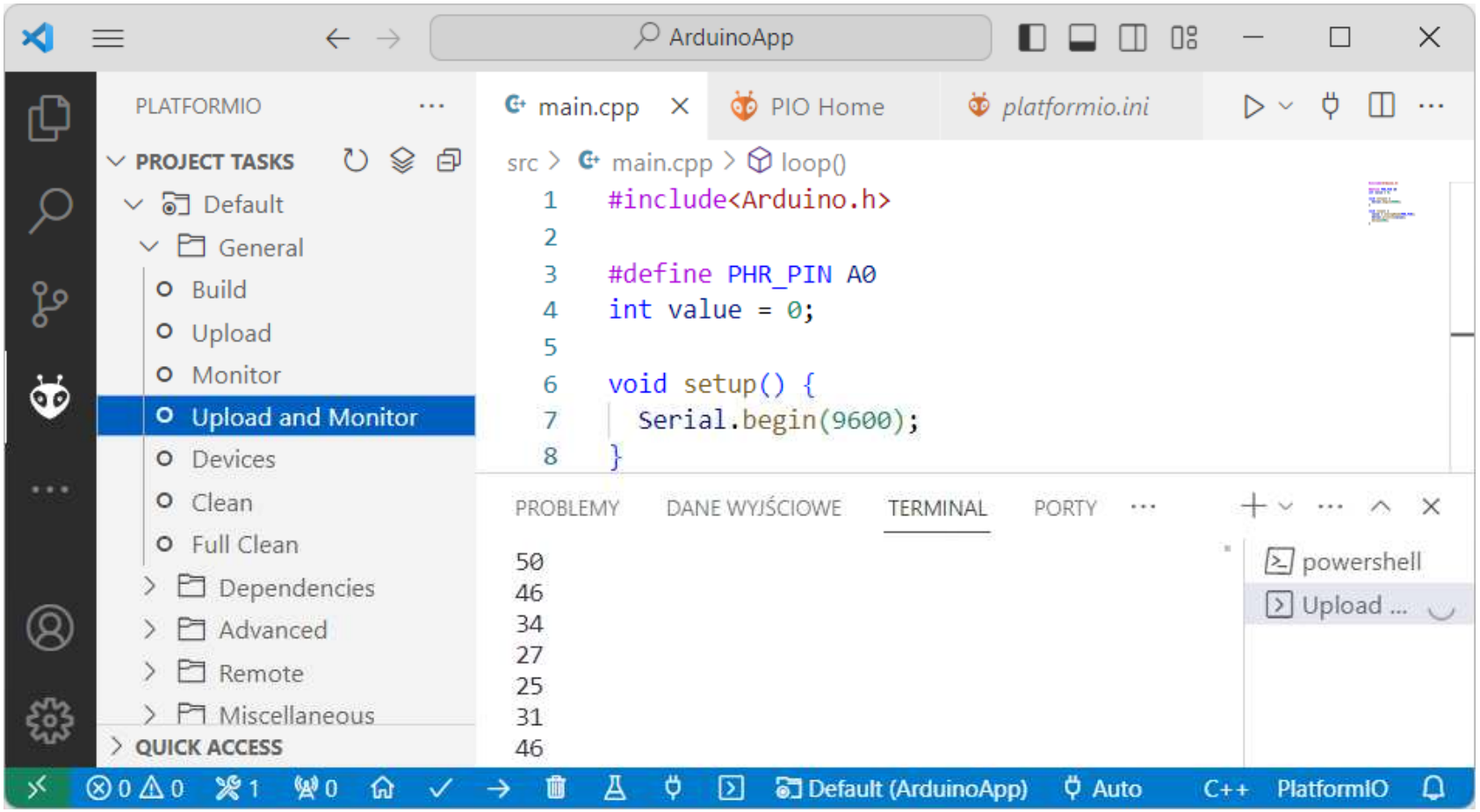
Program obsługujący fotorezystor

```
#include<Arduino.h>

#define PHR_PIN A0
int value = 0;

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

void loop()
{
  value = analogRead(PHR_PIN);
  Serial.println(value);
  delay(250);
}
```

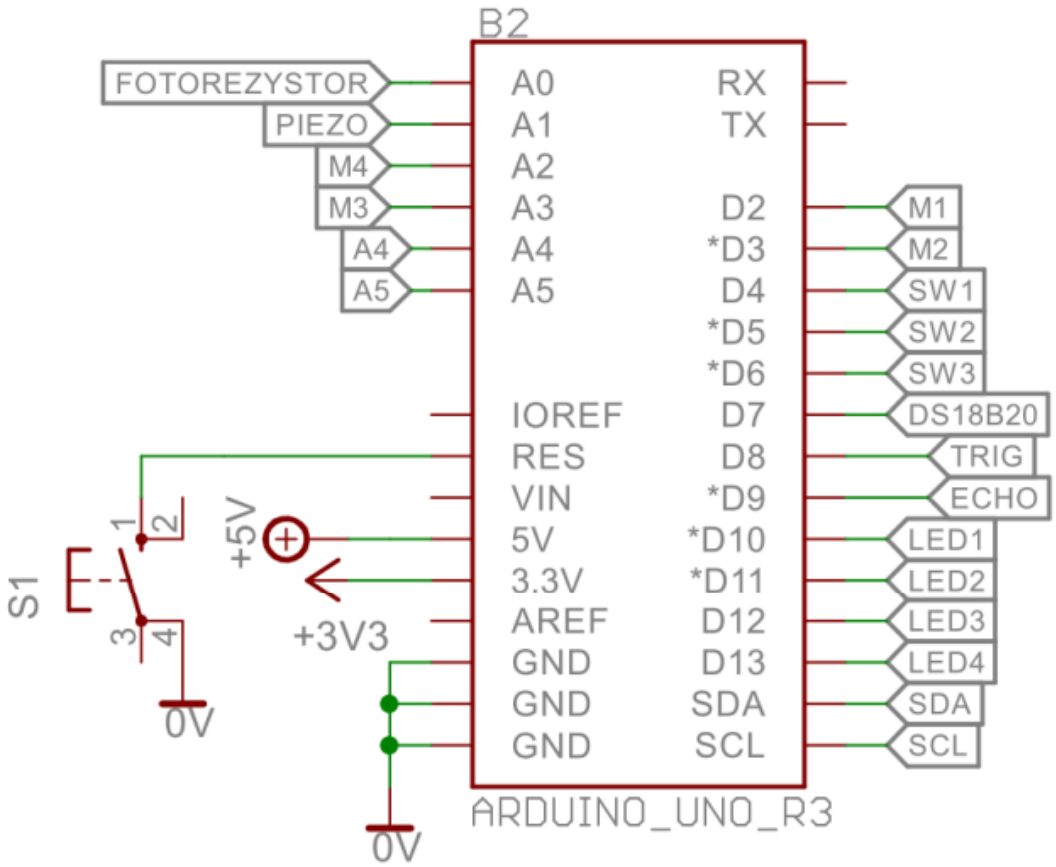


Arduino - fotorezystor

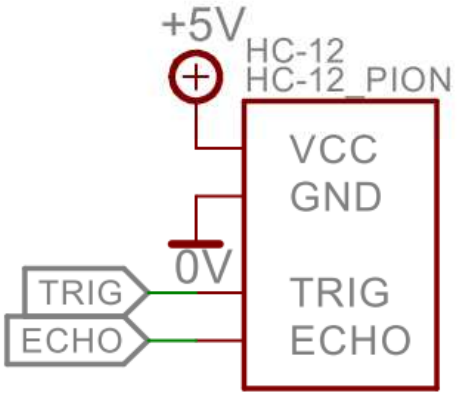
- Uruchomienie programu: PlatformIO → PROJECT TASK → Default → General → Upload and Monitor



Arduino - czujnik odległości (HC-SR04)

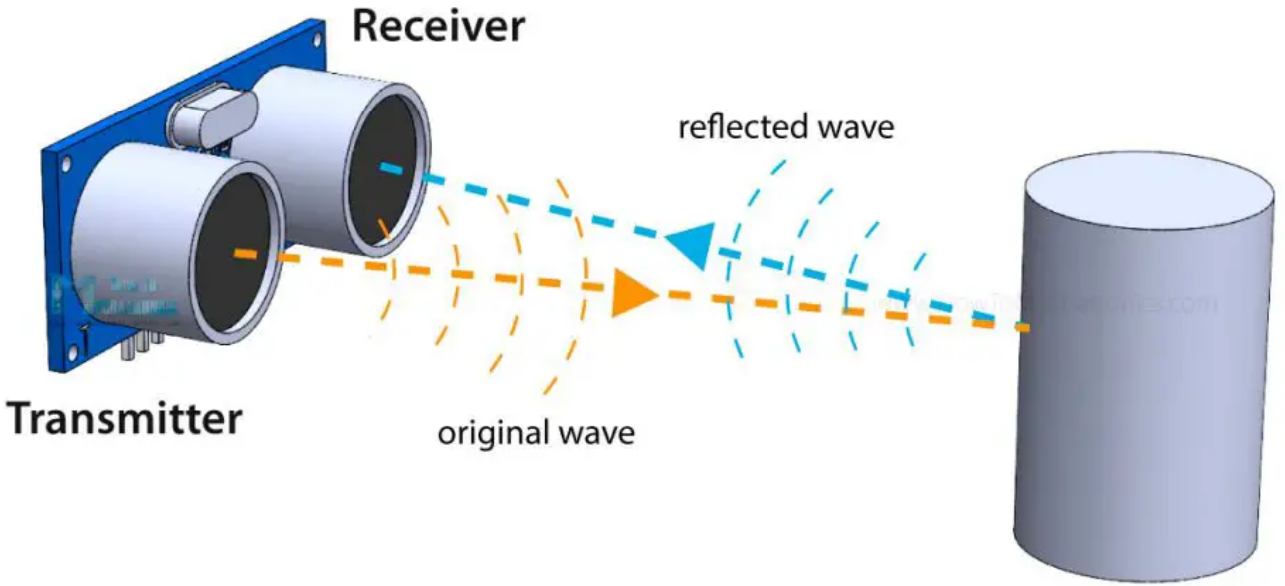


Schemat podłączenia
wyprowadzeń modułu Arduino



Schemat podłączenia
czujnika HC-SR04

Arduino - czujnik odległości (HC-SR04)

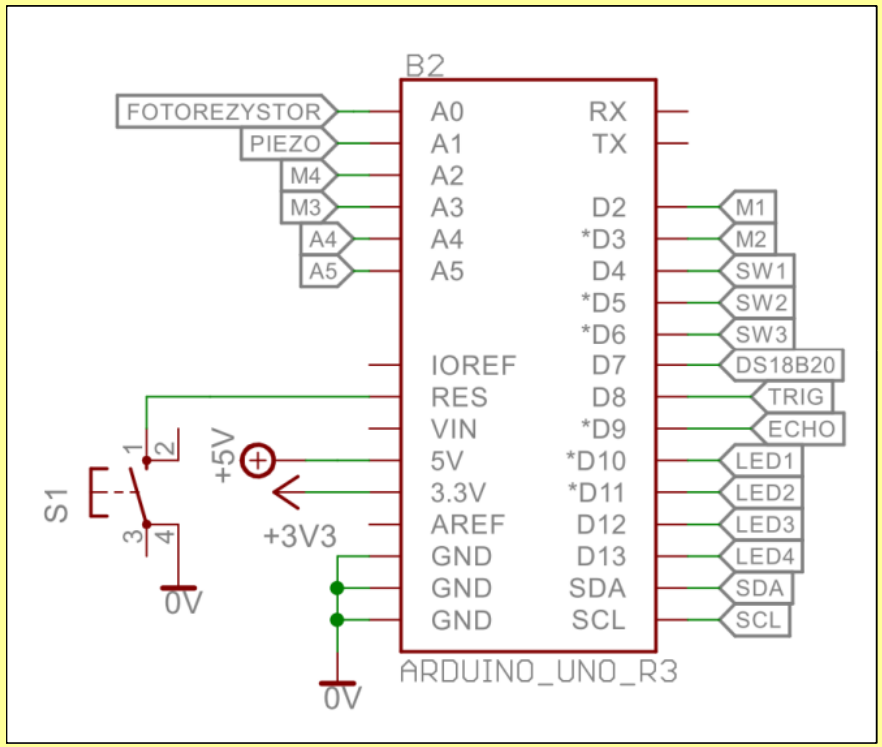


- Czujnik składa się z nadajnika i odbiornika, układu sterowania oraz generatora sygnału pomiarowego
- Nadajnik wysyła sygnał ultradźwiękowy, który odbija się od obiektu i wraca do odbiornika
- Odległość jest obliczana na podstawie różnicy czasu, który upływa od chwili wysłania sygnału pomiarowego, do chwili jego odbioru

Arduino - czujnik odległości (HC-SR04)

Czujnik odległości

```
#include<Arduino.h>

#define TRIG_PIN 8
#define ECHO_PIN 9

long duration;
int distance;

void setup()
{
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  Serial.begin(9600);
}
```



Arduino - czujnik odległości (HC-SR04)

Czujnik odległości

```
void loop()
{
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  duration = pulseIn(ECHO_PIN, HIGH);
  distance = duration * 0.034 / 2;
  Serial.print("Odlegosc: ");
  Serial.print(distance);
  Serial.println(" cm");
  delay(250);
}
```

Arduino - czujnik odległości (HC-SR04)

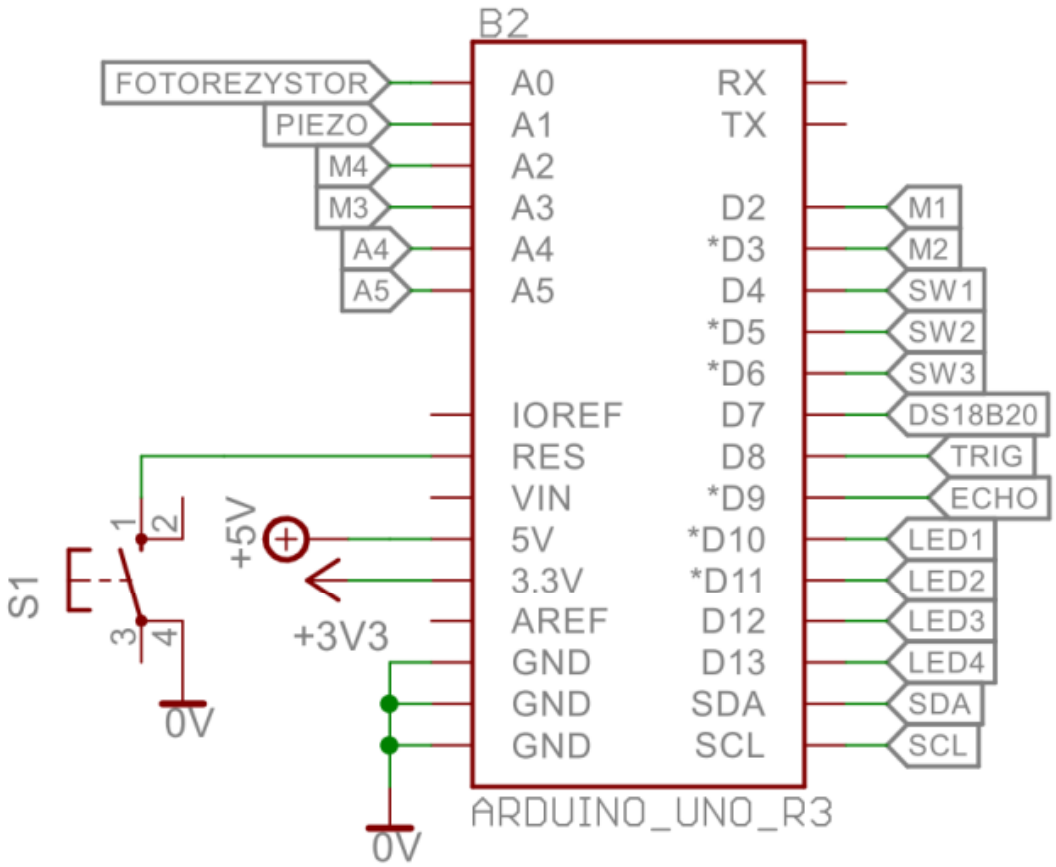
- Zasada pomiaru:
  - po podaniu na wejście Trig stanu wysokiego o czasie trwania 10 µs, wbudowany generator wysyła na nadajnik czujnika sygnał ultradźwiękowy o częstotliwości 40 kHz przez czas 200 µs
  - po zakończeniu wysyłania tego sygnału pin Echo zmienia stan na wysoki, rozpoczynając pomiar
  - wyemitowana fala ultradźwiękowa jest wysyłana i gdy napotka na drodze obiekt, zostaje odbita w stronę odbiornika
  - zmierzony czas trwania stanu wysokiego na pinie Echo (funkcja pulseIn()) jest proporcjonalny do odległości czujnika od przeszkody
  - odległość d (w cm) jest obliczana na podstawie wzoru:

$$d = \frac{t\,[\mu s] \cdot v\,\left[\frac{cm}{\mu s}\right]}{2}$$

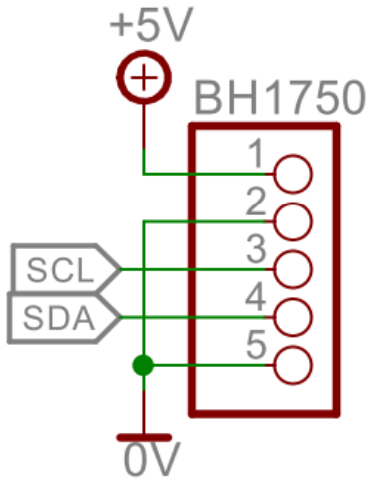
t - zmierzony czas w µs

v - prędkość dźwięku w powietrzu (v = 340 m/s = 0,034 cm/µs)

Arduino - czujnik natężenia światła (BH1750)



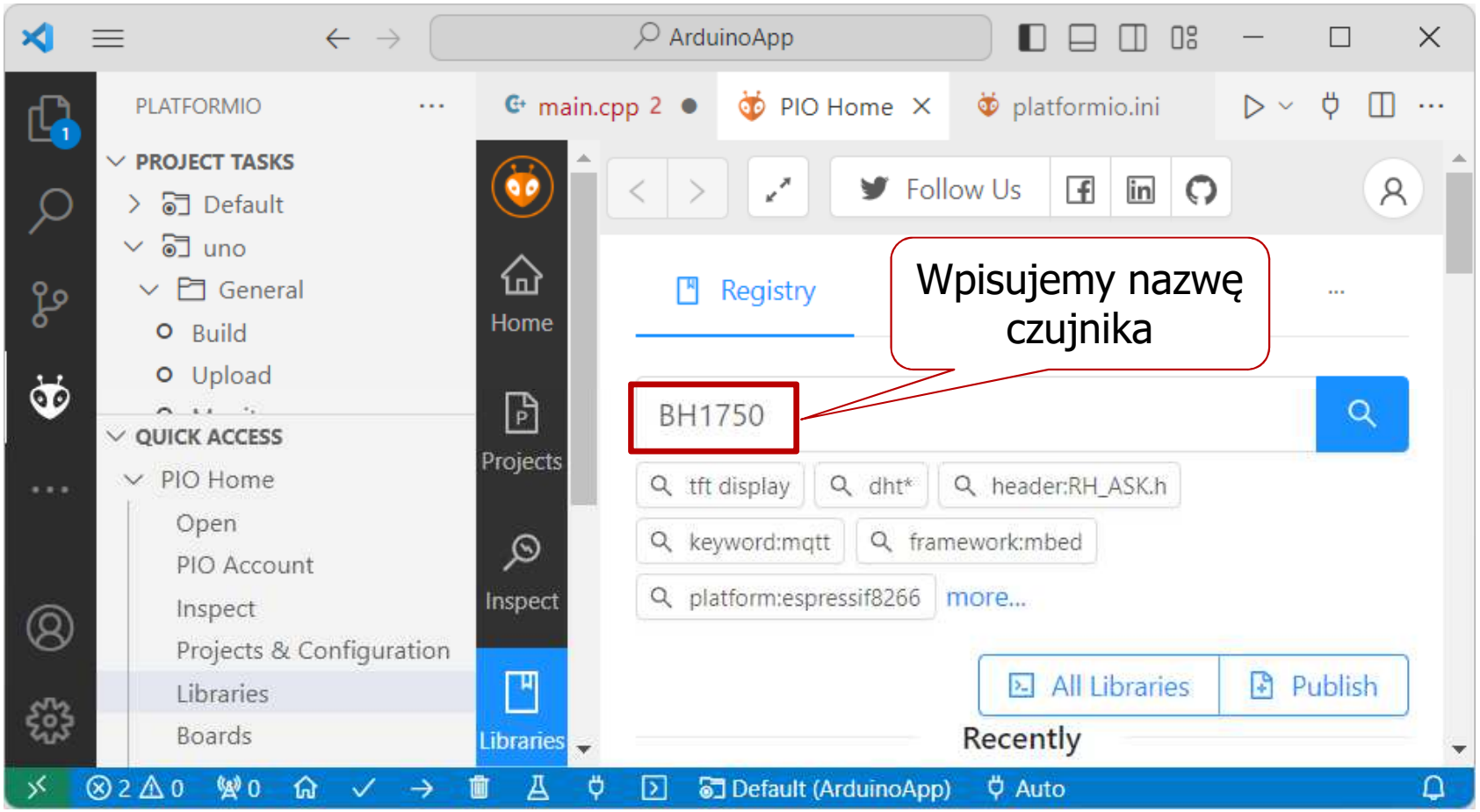
Schemat podłączenia
wyprowadzeń modułu Arduino



Schemat podłączenia
czujnika BH1750

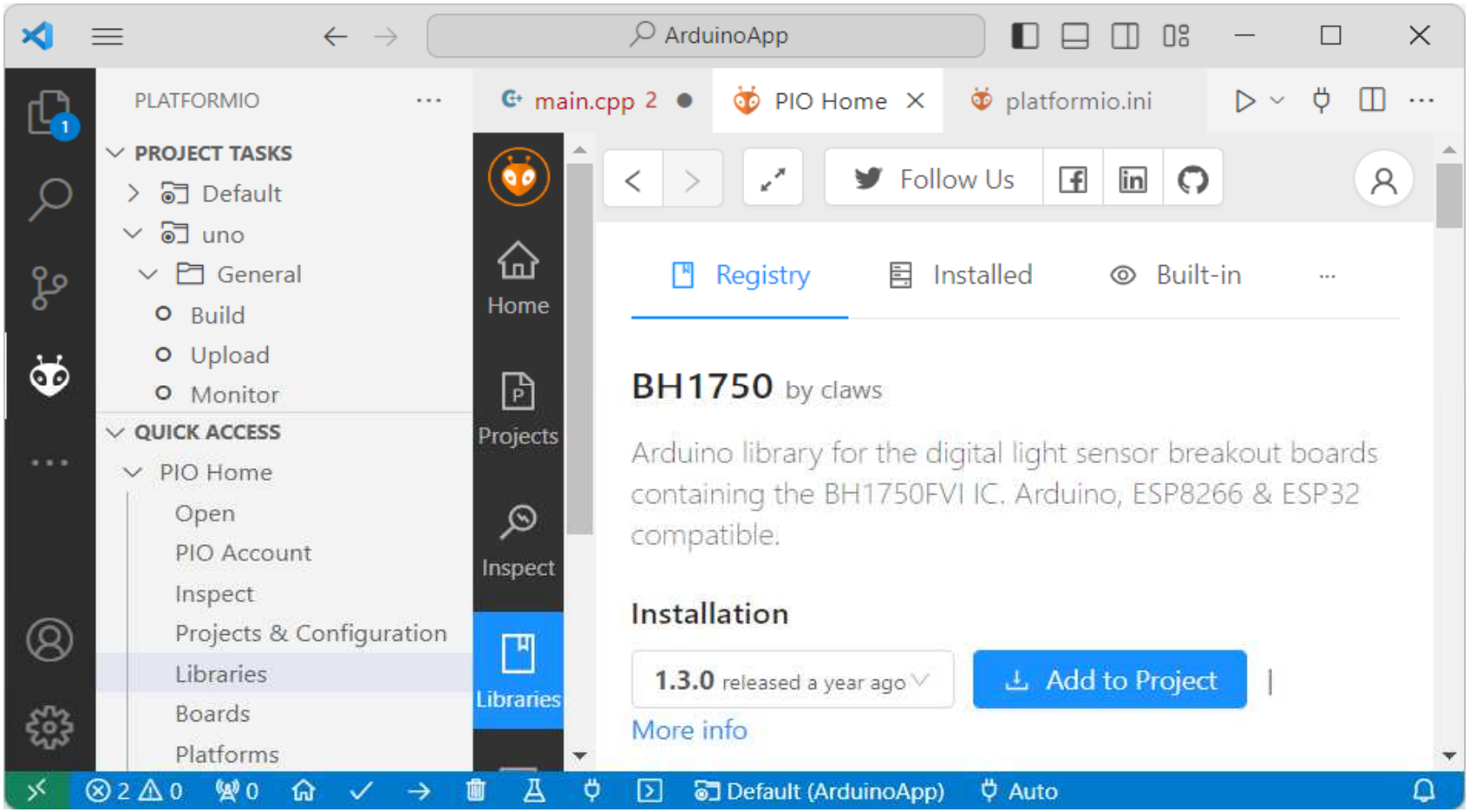
Arduino - czujnik natężenia światła (BH1750)

- Zastosowanie czujnika BH1750 wymaga zainstalowania biblioteki:
  PlatformIO → QUICK ACCESS → PIO Home → Libraries



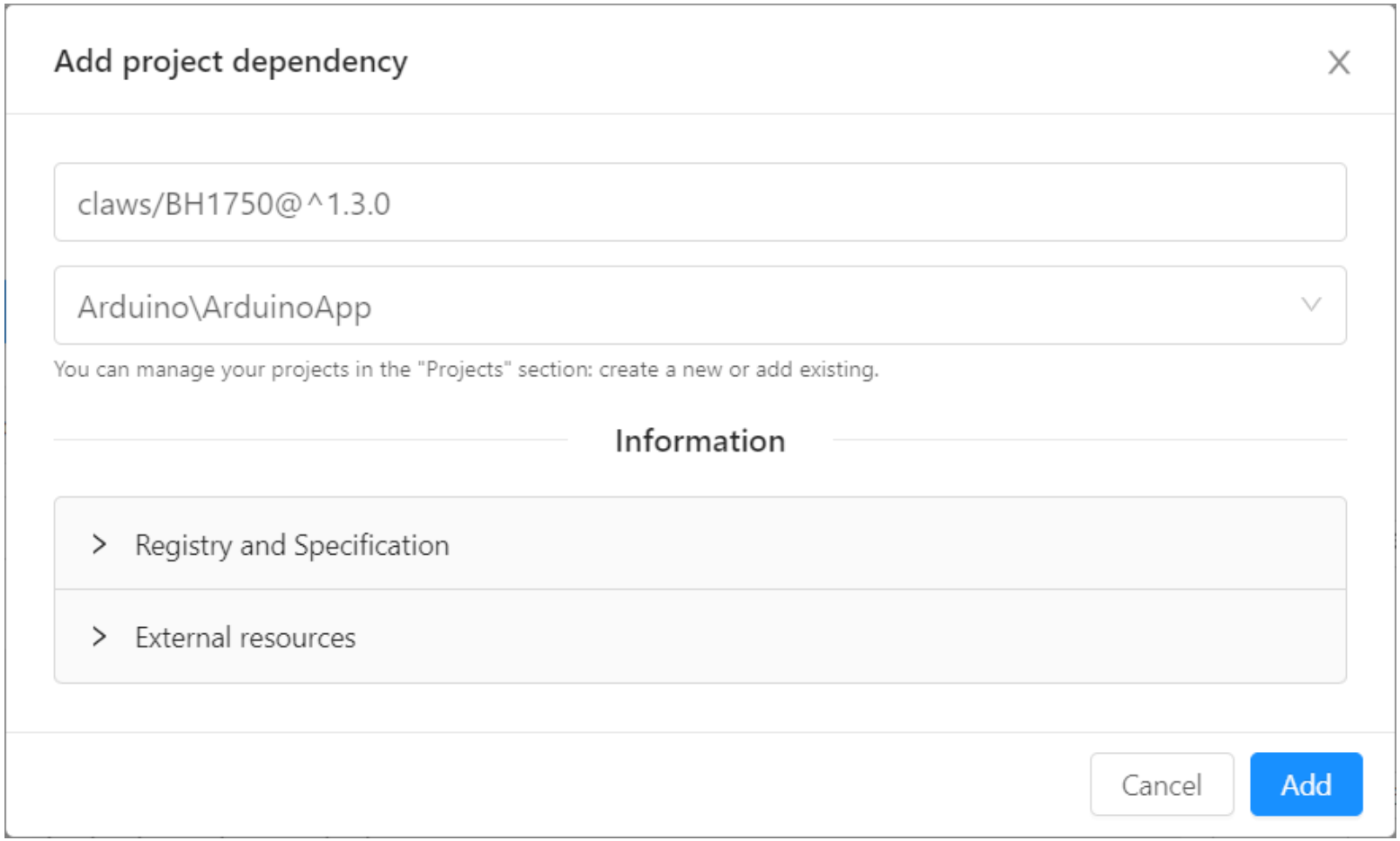
Arduino - czujnik natężenia światła (BH1750)

- Wybieramy bibliotekę o nazwie BH1750 by claws i instalujemy ją, klikając przycisk Add to Project



Arduino - czujnik natężenia światła (BH1750)

- Wybieramy projekt, do którego ma być dodana biblioteka BH1750 by claws



Arduino - czujnik natężenia światła (BH1750)

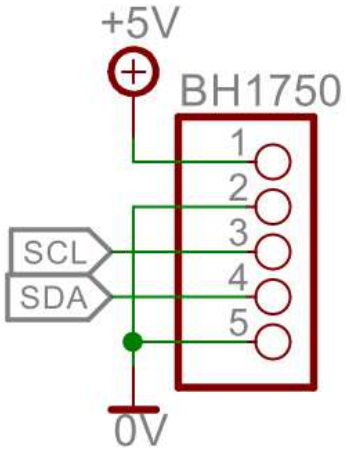
- W pliku konfiguracyjnym platformio.ini zostanie dodany wiersz:

```
lib_deps = claws/BH1750@^1.3.0
```

- W pliku .cpp z kodem źródłowym zostaną dodane dwie biblioteki

```
#include <BH1750.h>
#include <Wire.h>
```

- Czujnik komunikuje się z modułem Arduino za pomocą interfejsu I2C, wykorzystując dwie linie:
  - danych - SDA (Serial Data Line)
  - zegarową - SCL (Serial Clock Line)



Arduino - czujnik natężenia światła (BH1750)

Czujnik natężenia światła

```
#include <Arduino.h>
#include <BH1750.h>
#include <Wire.h>

BH1750 LightMeter(0x23);

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

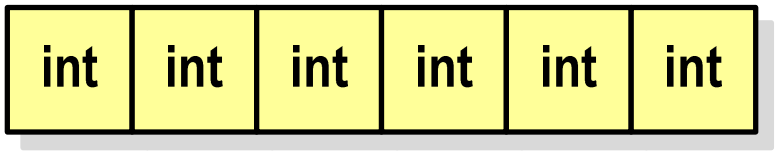
Arduino - czujnik natężenia światła (BH1750)

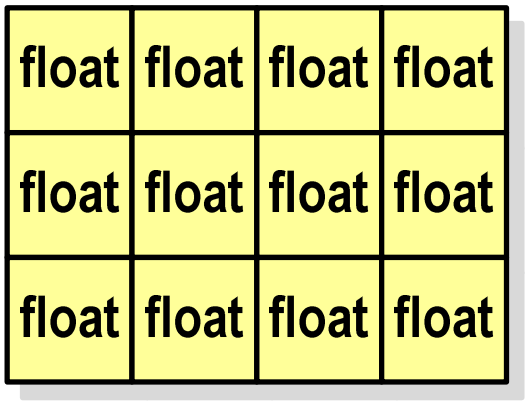
Czujnik natężenia światła

```
void loop()
{
  float lux = LightMeter.readLightLevel();
  Serial.print("Natezenie swiatla: ");
  Serial.print(lux);
  Serial.println(" lx");
  delay(250);
}
```

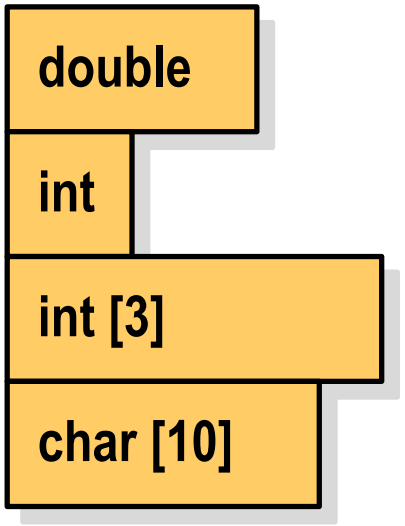
Struktury w języku C

- Tablica - ciągły obszar pamięci zawierający elementy tego samego typu





- Struktura - zestaw elementów różnych typów, zgrupowanych pod jedną nazwą



Deklaracja struktury

```
struct nazwa
{
    opis_pola_1;
    opis_pola_2;
    ...
    opis pola_n;
};
```

```
struct punkt
{
    int x;
    int y;
};
```

- Elementy struktury to pola (dane, komponenty, składowe) struktury
- Deklaracje pól mają taką samą postać jak deklaracje zmiennych
- Deklarując strukturę tworzymy nowy typ danych (struct punkt), którym można posługiwać się tak samo jak każdym innym typem standardowym

Deklaracja struktury

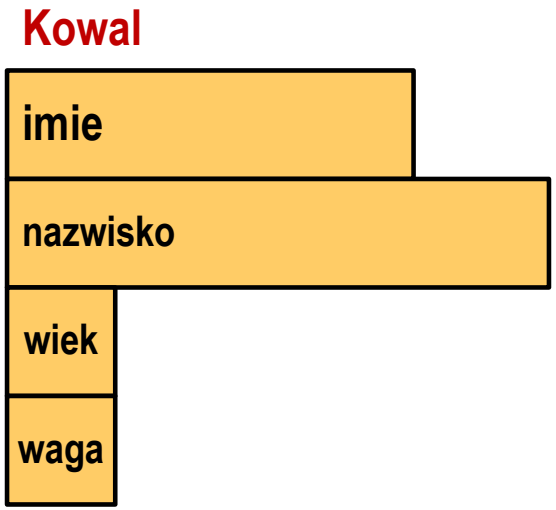
```
struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
};
```

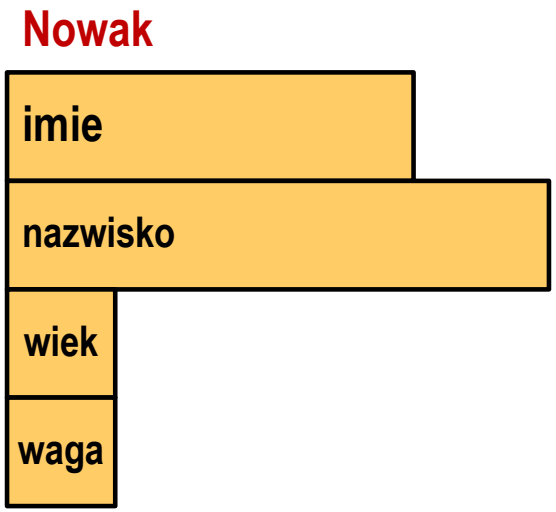
```
struct zesp
{
    float Re, Im;
};
```

- Deklaracja struktury nie tworzy obiektu (nie przydziela pamięci na pola struktury)
- Zapisanie danych do struktury wymaga zdefiniowania zmiennej strukturalnej

Deklaracja zmiennej strukturalnej

```
#include <stdio.h>

struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
} Kowal ;

int main(void)
{
    struct osoba Nowak ;
    ...
}
```





- Kowal, Nowak - zmienne typu struct osoba

Odwołania do pól struktury

- Dostęp do pól struktury możliwy jest dzięki konstrukcji typu:

```
nazwa_zmiennej_strukturalnej.nazwa_pola
```

- Operator . nazywany jest operatorem bezpośredniego wyboru pola
- Zapisanie wartości do pól zmiennej Nowak ma postać

```
Nowak.wiek = 25;
strcpy(Nowak.imie,"Jan");
```

- Wyrażenie Nowak.wiek traktowane jest jak zmienna typu int, zaś wyrażenie Nowak.imie traktowane jest jak łańcuch znaków

```
printf("%s - wiek %d\n", Nowak.imie, Nowak.wiek);
scanf("%d", &Nowak.wiek);
gets(Nowak.imie);
```

Struktury - przykład (osoba)

```
#include <stdio.h>

struct osoba
{
    char imie[15];
    char nazwisko[20];
    int  wiek;
};

int main(void)
{
     struct osoba Nowak;
```

Struktury - przykład (osoba)

```
    printf("Imie:      ");
    gets(Nowak.imie);

    printf("Nazwisko: ");
    gets(Nowak.nazwisko);

    printf("Wiek:      ");
    scanf("%d",&Nowak.wiek);

    printf("%s %s, wiek: %d\n", Nowak.imie,
           Nowak.nazwisko, Nowak.wiek);

    return 0;
}
```

```
Imie:      Jan
Nazwisko: Nowak
Wiek:      22
Jan Nowak, wiek: 22
```

Struktury w języku C

- Inicjalizacja może dotyczyć tylko zmiennych strukturalnych, nie można inicjalizować pól w deklaracji struktury

```
struct osoba
{
    char imie[15], nazwisko[20];
    int wiek, waga;
};
```

```
struct osoba Nowak = {"Jan","Nowak",25,74};
```

- Do zmiennych strukturalnych można stosować operator =

```
struct osoba Kowal = {"Ewa","Kowal",21,54};
struct osoba Kowal1;
Kowal1 = Kowal;
```

Struktury w języku C

```
#include <stdio.h>

struct date
{
    int day;
    int month;
    int year;
} day1 = {19,11,2018};

int main(void)
{
    struct date day2;
```

day1

day	19
month	11
year	2018

day2

day	?
month	?
year	?

Struktury w języku C

```
    day2.day = 1;
    day2.month = 9;
    day2.year = 2018;

    printf("Date1: %02d-%02d-%4d\n",
        day1.day,day1.month,day1.year);
    printf("Date2: %02d-%02d-%4d\n",
        day2.day,day2.month,day2.year);

    return 0;
}
```

```
Date1: 19-11-2018
Date2: 01-09-2018
```

day1

day	19
month	11
year	2018

day2

day	1
month	9
year	2018

Struktury - przykład (miernik)

```
#include <stdio.h>

struct miernik
{
    double k;    // klasa dokładności
    int d;       // liczba działek podziałki
    double Zp;   // zakres pomiarowy
};

int main(void)
{
    // Amperomierz LE-3P
    struct miernik LE3P = {0.5,60,12};
    double Dpm, p;
```

Struktury - przykład (miernik)

```
    printf("Amperomierz analogowy LE-3P\n");
    printf("Zakres pomiarowy: %g A\n",LE3P.Zp);
    printf("Liczba dzialek podzialki: %d\n",LE3P.d);
    printf("Klasa dokladnosci: %g\n",LE3P.k);
    printf("-------------------------------\n");

    printf("Bezwzgledny maksymalny blad pomiaru:\n");
    p = 0.2;
    Dpm = LE3P.Zp*(LE3P.k/100+p/LE3P.d);
    printf("* dla p = %g, Dpm = %g A\n",p,Dpm);


    p = 0.5;
    Dpm = LE3P.Zp*(LE3P.k/100+p/LE3P.d);
    printf("* dla p = %g, Dpm = %g A\n",p,Dpm);

    return 0;
}
```

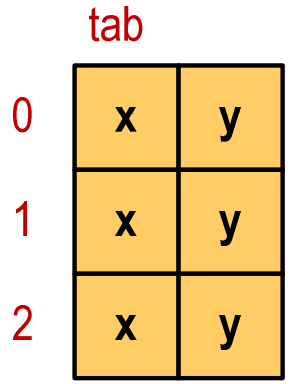
Struktury - przykład (miernik)

```
    printf("Amperomi
    printf("Zakres p
    printf("Liczba d
    printf("Klasa do
    printf("--------

    printf("Bezwzgle
    p = 0.2;
    Dpm = LE3P.Zp*(L
    printf("* dla p = %g, Dpm = %g A\n",p,Dpm);

    p = 0.5;
    Dpm = LE3P.Zp*(LE3P.k/100+p/LE3P.d);
    printf("* dla p = %g, Dpm = %g A\n",p,Dpm);

    return 0;
}
```

```
Amperomierz analogowy LE-3P
Zakres pomiarowy: 12 A
Liczba dzialek podzialki: 60
Klasa dokladnosci: 0.5
-------------------------------
Bezwzgledny maksymalny blad pomiaru:
* dla p = 0.2, Dpm = 0.1 A
* dla p = 0.5, Dpm = 0.16 A
```

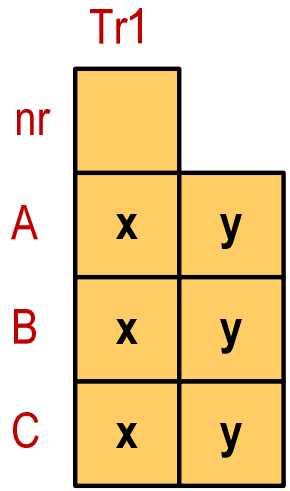
Złożone deklaracje struktur

```
struct punkt
{
    int x;
    int y;
} tab[3];
```




```
tab[0].x = 10;
tab[0].y = 20;
tab[1].x = 15;
…
```

```
struct trojkat
{
    int nr;
    struct punkt A, B, C;
} Tr1;
```

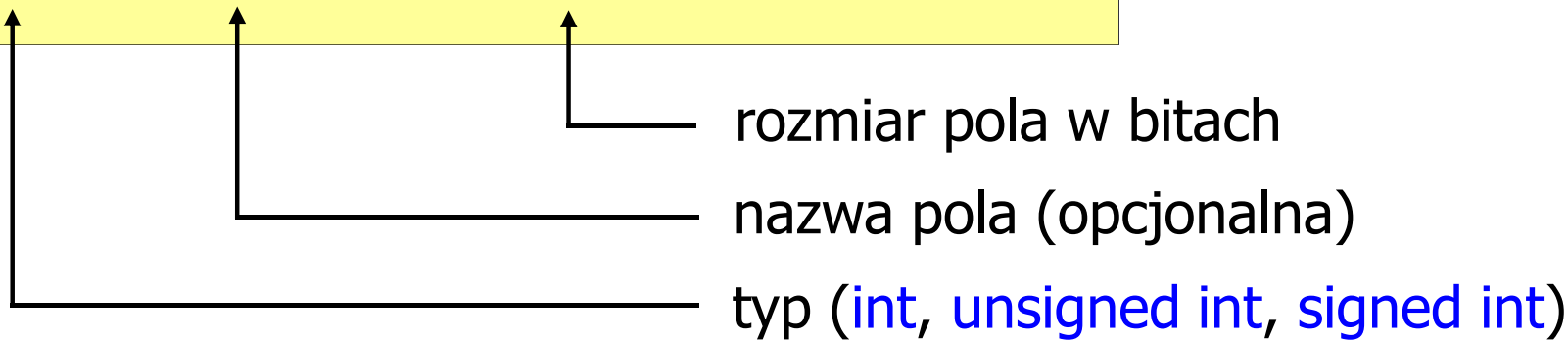



```
Tr1.nr = 1;
Tr1.A.x = 10;
Tr1.A.y = 20;
Tr1.B.x = 15;
…
```

Pola bitowe

- Umożliwiają dostęp do pojedynczych bitów oraz przechowywanie małych wartości zajmujących pojedyncze bity
- Pola bitowe deklarowane są wewnątrz struktur

```
typ id_pola : wielkość_pola;
```



- Wartości zapisane w polach traktowane są jak liczby całkowite
- Zakres wartości pól wynika z wielkości_pola

Pola bitowe

```
struct Bits
{
    unsigned int a : 4;     /* zakres: 0...15 */
    unsigned int b : 2;     /* zakres: 0...3 */
    unsigned int   : 4;
    unsigned int c : 6;     /* zakres: 0...63 */
};
```

- Dostęp do pól bitowych odbywa się na takiej samej zasadzie jak do normalnych pól struktury

```
struct Bits dane;

dane.a = 10;
dane.b = 3;
```

Pola bitowe

```
struct Bits
{
    unsigned int a : 4;     /* zakres: 0...15 */
    unsigned int b : 2;     /* zakres: 0...3 */
    unsigned int   : 4;
    unsigned int c : 6;     /* zakres: 0...63 */
};
```

- Jeśli pole nie ma nazwy, to nie można się do niego odwoływać
- Pola bitowe nie mają adresów
  - nie można wobec pola bitowego stosować operatora & (adres)
  - nie można polu bitowemu nadać wartości funkcją scanf()

Pola bitowe - przykład

```
struct Flags_8086
{
    unsigned int CF : 1;    /* Carry Flag */
    unsigned int    : 1;
    unsigned int PF : 1;    /* Parity Flag */
    unsigned int    : 1;
    unsigned int AF : 1;    /* Auxiliary - Carry Flag */
    unsigned int    : 1;
    unsigned int ZF : 1;    /* Zero Flag */
    unsigned int SF : 1;    /* Signum Flag */
    unsigned int TF : 1;    /* Trap Flag */
    unsigned int IF : 1;    /* Interrupt Flag */
    unsigned int DF : 1;    /* Direction Flag */
    unsigned int OF : 1;    /* Overflow Flag */
};
```

Unie

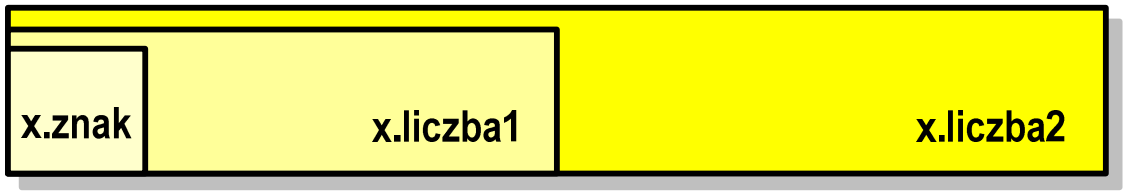
- Specjalny rodzaj struktury umożliwiający przechowywanie danych różnych typów w tym samym obszarze pamięci

```
union zbior
{
    char   znak;
    int    liczba1;
    double liczba2;
};
```

- Do przechowywania wartości w unii należy zadeklarować zmienną

```
union zbior x;
```

- Zmienna x może przechowywać wartość typu char lub typu int lub typu double, ale tylko jedną z nich w danym momencie



- Rozmiar unii wyznaczany jest przez rozmiar największego jej pola

Unie

- Dostęp do pól unii jest taki sam jak do pól struktury

```
union zbior x;
x.znak = 'a';
x.liczba2 = 12.15;
```

```
union zbior
{
    char   znak;
    int    liczba1;
    double liczba2;
};
```

- Unię można zainicjować jedynie wartością o typie jej pierwszej składowej
- Unie tego samego typu można sobie przypisywać

```
union zbior x = {'a'};
union zbior y;
y = x;
```

Koniec wykładu nr 6

Dziękuję za uwagę!