Programowanie systemów autonomicznych

Blok obieralny: **Inteligentne Systemy Autonomiczne**

Instytut Informatyki Stosowanej
Stefanowskiego 18/22
al. Politechniki 11

Tomasz Jaworski
Piotr Duch

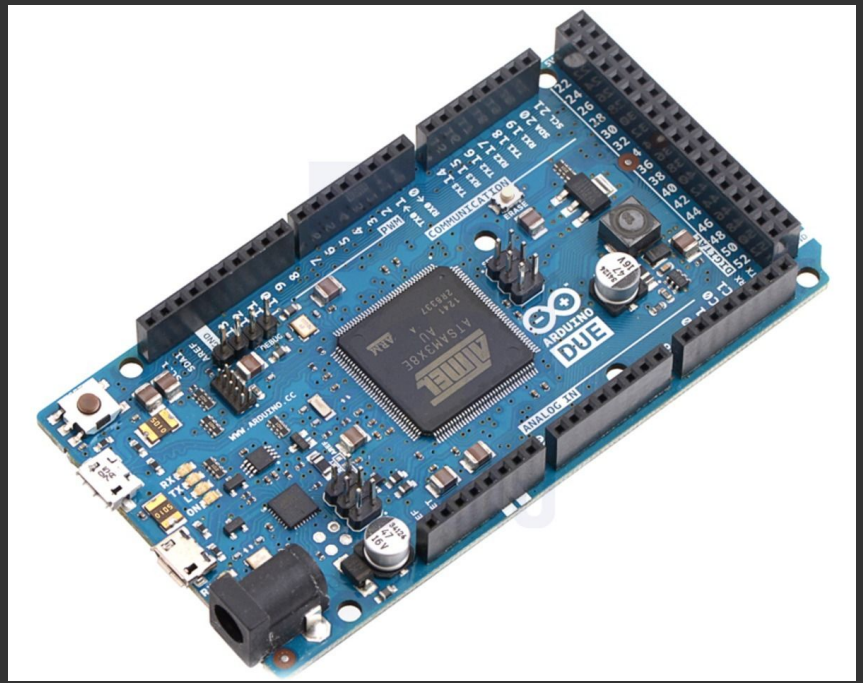
Programowanie systemów autonomicznych

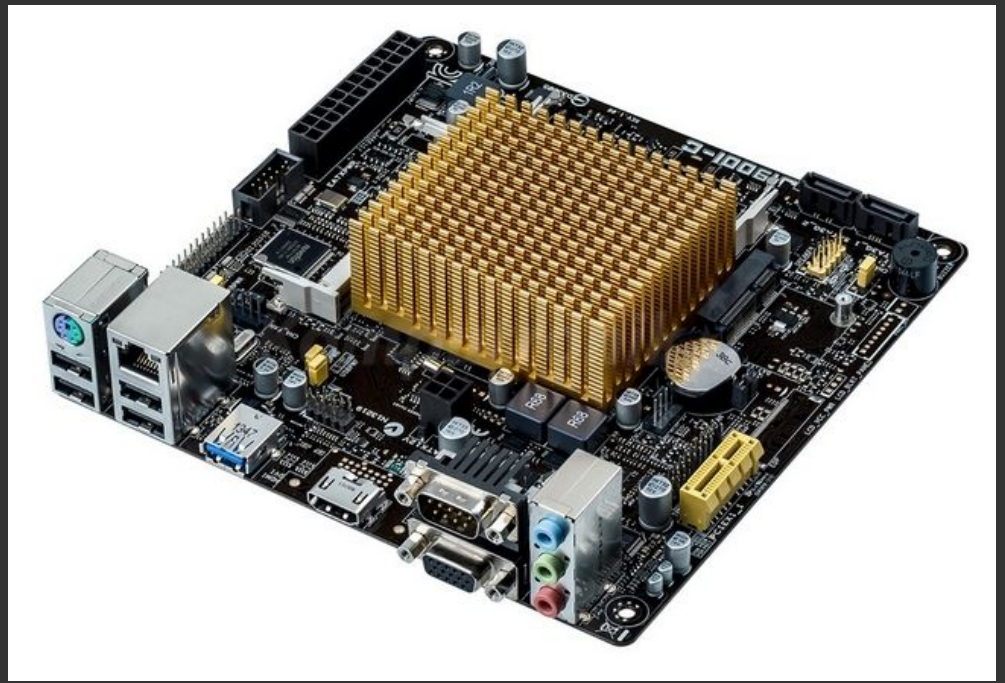
Wprowadzenie do programowania Arduino

Komputer (PC) + urządzenia wejścia/wyjścia

Komputer (Arduino) + urządzenia wejścia/wyjścia

Różnice? Podobieństwa?





Charakterystyka Arduino DUE

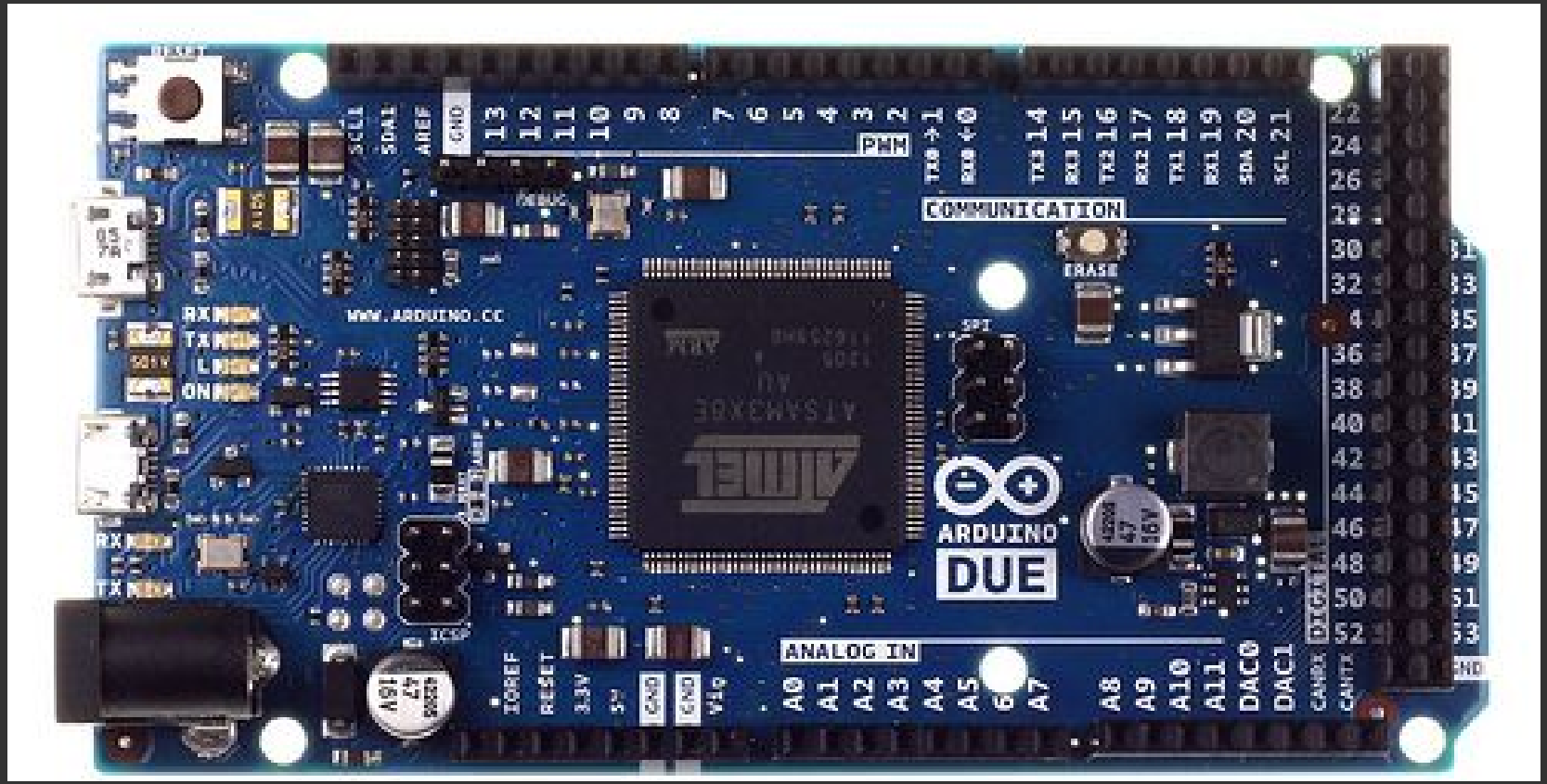
Mikrokontroler: **Atmel AT91SAM3X8E**

- Architektura: **32 bitowy ARM**
- Taktowanie: **82 MHz**
- Wielkość pamięci RAM: **96 kB**
- Wielkość pamięci Flash (pamięci programu): **512 kB**
- Napięcie zasilania (zalecane): 7-12 V (napięcie stałe)
- Napięcie zasilania (limit): 6-16 V (napięcie stałe)
- Napięcie wewnętrzne: 3,3 V
- Liczba wejść/wyjść cyfrowych: **54**
  - w tym **12** ma możliwość działania jako wyjścia PWM.
- Liczba wejść analogowych (ADC): 12
- Liczba wyjść analogowych (DAC): 2

- Brak RTC

- Obciążalność prądowa pojedynczego wyjścia logicznego: 6-9 mA (zależnie od wejścia)
- Obciążalność prądowa całkowita: **150 mA** - suma prądów pobieranych z wszystkich wyjść Arduino nie może przekroczyć tej wartości.

Arduino Due - omówienie interfejsów I/O



Środowisko programisty

Arduino IDE jest to rozbudowane środowisko programistyczne, przeznaczone do pisania kodu na **różne wersje** jednoukładowego komputera Arduino.
Systemy operacyjne: Windows, Mac OS oraz Linux.

Wbudowany kompilator architektury ARM - gcc, g++ (arm binutils)

Używany język:

- C z elementami C++
- głównie metody na rzecz obiektów,
- brak wyjątków (-fno-exceptions)
- dostępne malloc/free, new/delete

Własna instalacja - konfiguracja środowiska

Arduino IDE nie obsługuje natywnie wersji DUE, należy ją zainstalować:

- Menedżer płytek (**Narzędzia -> Płytka -> Menedżer płytek**), wybrać i zainstalować “**Arduino SAM Boards**”.

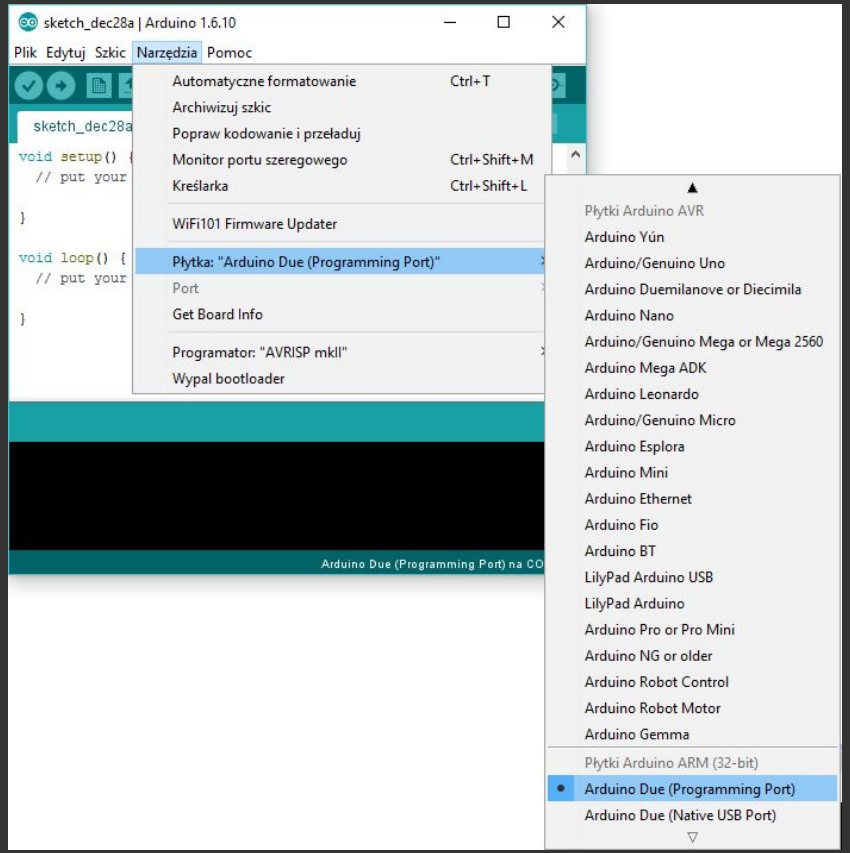
Własna instalacja - połączenie środowiska z modułem DUE

Zainstalowany dodatek do Arudino Due należu **uaktywnić**: **Narzędzia -> Płytka**, wybrać **Arduino Due (Programming Port)**.

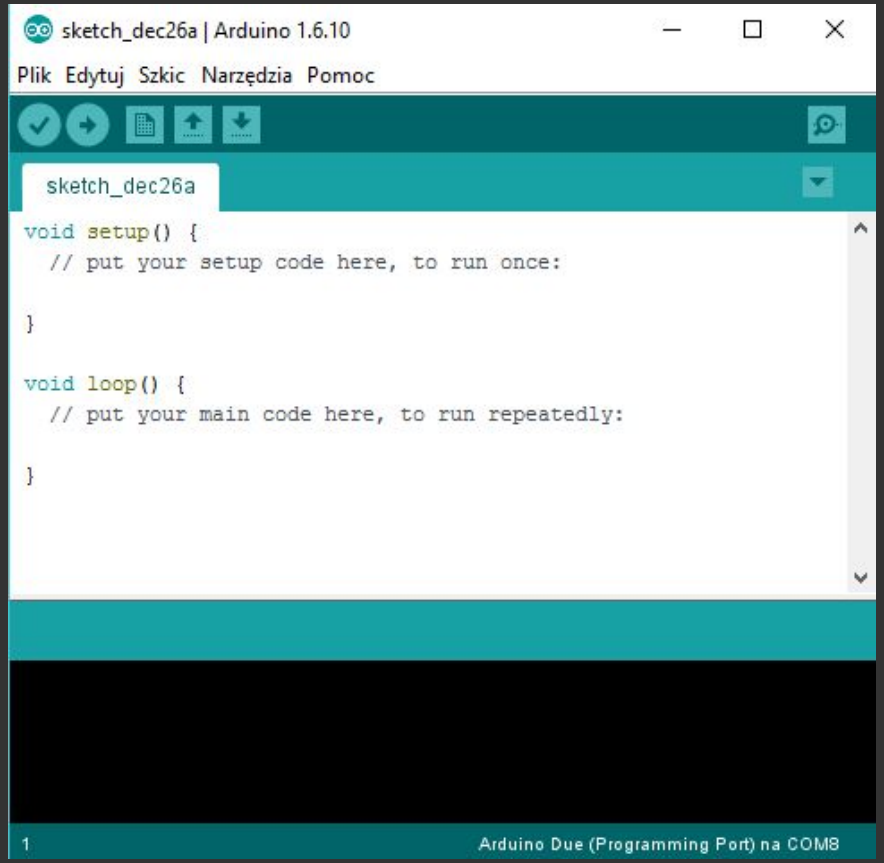
następnie

Z **Narzędzia -> Port** należy wybrać **port USB**, do którego podłączono Arduino DUE.

(system sam wykryje listę portów, łącznie z Arduino)



Pierwsze uruchomienie



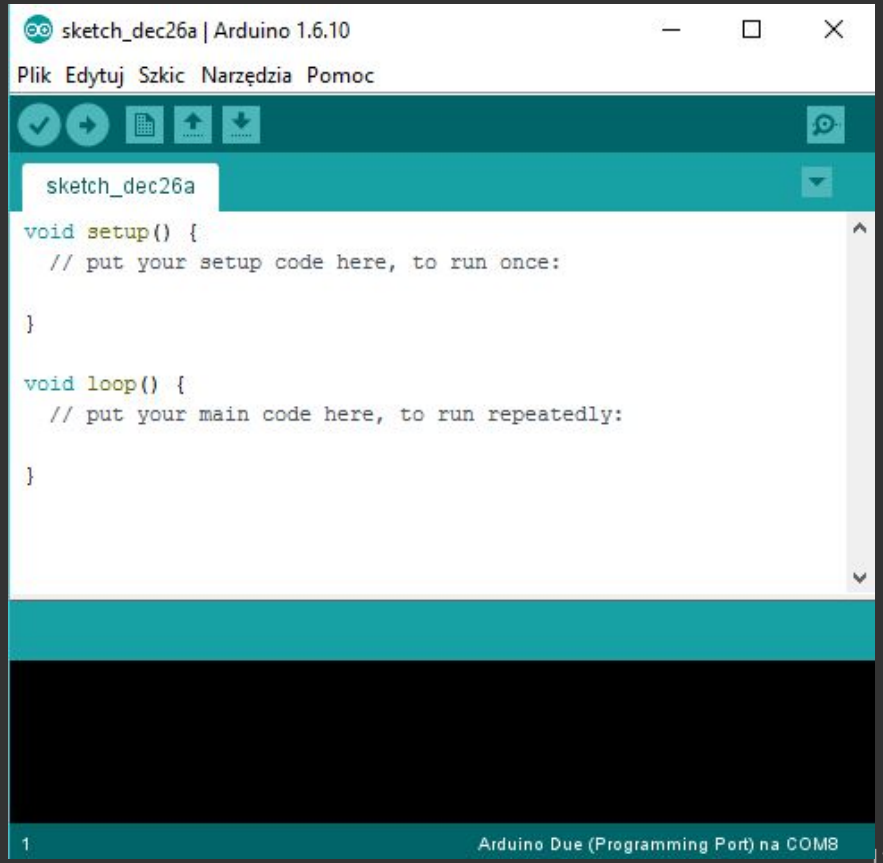
Przydatne narzędzia

Pasek narzędzi:

- Kompiluj/weryfikuj
- Wgraj i uruchom
- Nowy
- Otwórz
- Zapisz
- Monitor portu szeregowego

Przydatne ustawienia (Plik -> Ustawienia):

- Numerowanie linii
- Zwijanie kodu (code folding)



Arduino API

Arduino API - typy danych

int, unsigned int
16 bitów (2 bajty) - Arduino UNO
32 bity (4 bajty) - Arduino DUE

long, unsigned long
32 bity (4 bajty) - niezależnie od platformy

float
32 bity (4 bajty) - -3.4E+38 -- 3.4E+38 (pojedyncza precyzja)

double
64 bity (8 bajtów) - -1.7E+308 -- 1.7E+308 (podwójna precyzja)

Arduino API - typy danych

Ciągi znaków, identycznie z językiem C

```
char Str1[15];
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
char Str4[ ] = "arduino";
char Str5[8] = "arduino";
char Str6[15] = "arduino";
```

Arduino API - typy danych

Klasa String (https://www.arduino.cc/en/Reference/StringObject)

```
void loop() {
  // here's a String with empty spaces at the end (called white space):
  String stringOne = "Hello!       ";
  Serial.print(stringOne);
  Serial.print("<--- end of string. Length: ");
  Serial.println(stringOne.length());

  // trim the white space off the string:
  stringOne.trim();
  Serial.print(stringOne);
  Serial.print("<--- end of trimmed string. Length: ");
  Serial.println(stringOne.length());

  // do nothing while true:
  while (true);
}
```

Arduino API - typy danych

Tablice, jak w języku C:

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
char message[6] = "hello";
```

Arduino API - budowa programu

- Szkic (sketch)
- Wymaga napisania dwóch funkcji:
  - void setup(void) - wykonywana raz, po restarcie/podłączeniu zasilania
  - void loop(void) - wykonywana w pętli; zakończenie funkcji loop powoduje jej restart

Odniesienie do klasycznego schematu z języka C:

```
void main(void)
{
  setup();
  while(1)
    loop();
}
```

Arduino API - budowa programu -- różnice?

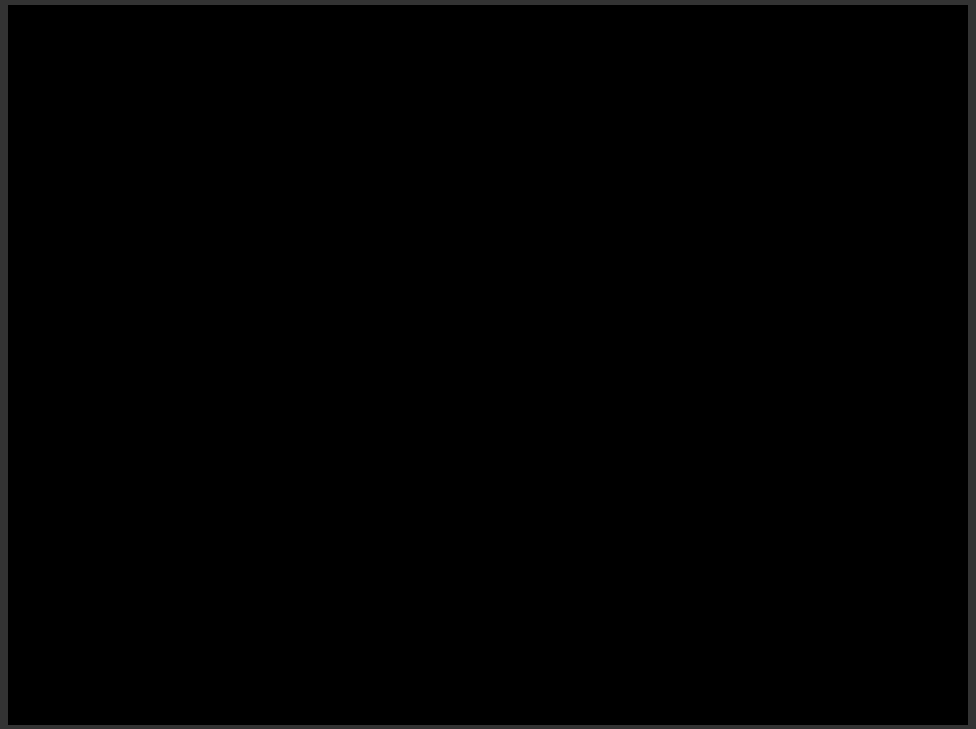
```
int licznik;
void setup(void)
{
  licznik = 0;
  Serial.begin(9600);
}

void loop(void)
{
  Serial.write(licznik);
  licznik++;
}
```

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

void loop(void)
{
  static int licznik = 0;
  Serial.write(licznik);
  licznik++;
}
```

```
int licznik = 0;
void setup(void)
{
  Serial.begin(9600);

  while(1)
  {
    Serial.write(licznik);
    licznik++;
  }
}

void loop(void)
{
}
```

Arduino API - funkcje/procedury

Arduino API - obiekt **LiquidCrystal** - wyświetlacz LCD



Arduino API - obiekt **LiquidCrystal** - wyświetlacz LCD

Biblioteka **LiquidCrystal** (jeśli nie ma, to trzeba dodać: **Sketch -> Include Library**)

Plik nagłówkowy:
#include <**LiquidCrystal.h**>

Konstruktor (notacja C++)
LiquidCrystal lcd(26, 28, 29, 30, 31, 32); // sygnały: RS, E, D4, D5, D6, D7

Konfiguracja:

```
void setup()
{
  lcd.begin(16,2);
}
```

Arduino API - obiekt **LiquidCrystal** - wyświetlacz LCD

Dostępne metody (https://www.arduino.cc/en/Reference/LiquidCrystal):

- **begin**(cols, rows) - inicjuj wyświetlacz (wejście: liczba wierszy i kolumn)
- **clear**() - czyść wyświetlacz
- **home**() - ustaw kursor w lewym górnym rogu (0, 0)
- **setCursor**(x, y) - ustaw kursor na współrzędnych *x* i *y* (licząc od 0)
- **print**(val [, format]) - wyświetl wartość (liczbę, tekst, znak) z formatowaniem
- **write**(ch) - wyświetl znak *ch*
- **noCursor**() - ukryj kursor
- **cursor**() - pokaż kursor
- **blink**() - włącz miganie kursora
- **noBlink**() - wyłącz miganie kursora
- **createChar**(ch, bmap) - utwórz własny znak *ch* z bitmapy *bmap* (5x8)

Arduino API - obiekt **LiquidCrystal** - wyświetlacz LCD

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(26, 28, 29, 30, 31, 32);

void setup()
{
  lcd.begin(16,2);
  lcd.setCursor(0,0); lcd.print("Hello, World!");
}
void loop() {
  static int i = 0;
  lcd.setCursor(0,1);
  lcd.print("i=");
  lcd.print(i++);
  delay(250);
}
```

Arduino API - obiekt **LiquidCrystal** - wyświetlacz LCD

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(26, 28, 29, 30, 31, 32);
void setup() {
  lcd.begin(16,2);
}
void loop() {
  delay(1000);
  lcd.setCursor(0,0); lcd.print("Programowanie   ");
  delay(3000);
  lcd.setCursor(0,1); lcd.print("urzadzen Arduino");
  delay(3000);
  lcd.setCursor(0,0); lcd.print("urzadzen Arduino");
  lcd.setCursor(0,1); lcd.print("jest BANALNIE ;)");
  delay(3000);
  lcd.setCursor(0,0); lcd.print("jest BANALNIE ;)");
  lcd.setCursor(0,1); lcd.print("proste.            ");
  delay(3000);
  lcd.clear();
}
```

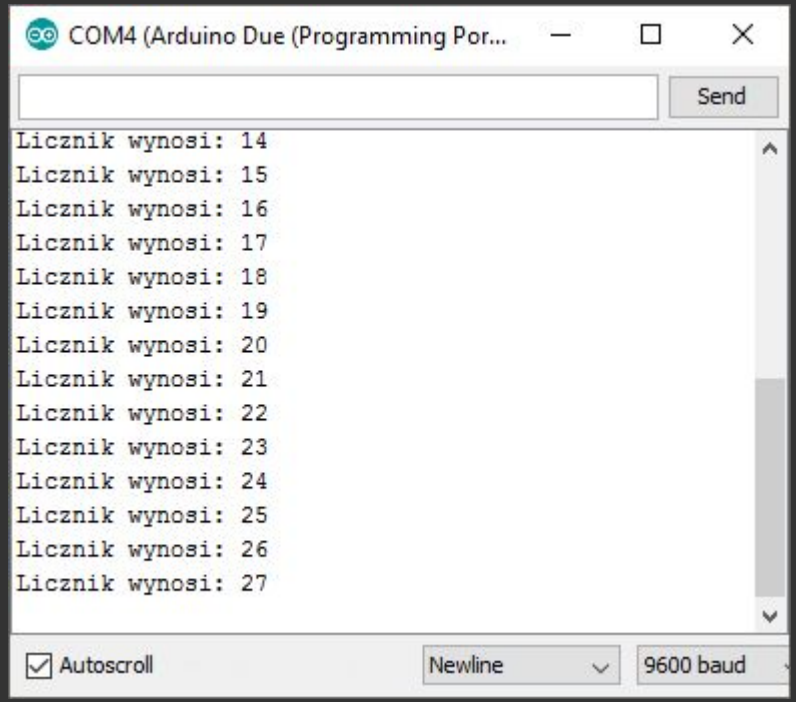
Arduino API - obiekt **Serial** - komunikacja/**konsola**

Port szeregowy: https://www.arduino.cc/en/reference/serial
Dostępne metody:

- **begin**(speed) - uruchom port szeregowy z prędkością speed
- **begin**(speed, config) - j/w + dodatkowa konfiguracja
- **print**(val [, format]) - wyślij wartość (tekst, liczbę, znak) do odbiornika
- **println**(val [, format]) - wyślij wartość + znak nowej linii do odbiornika
- int **available**() - pobierz liczbę danych w buforze wejściowym
- int **read**() - wczytaj i zwróć wartość bajta z bufora wejściowego lub -1
- String **readString**() - pobierz dane z bufora jako tekst (String)
- String **readStringUntil**(char) - wczytuj dane z bufora aż do napotkania *char* lub przekroczenia czasu
- **setTimeout**(ms) - ustaw limit czasu oczekiwania na dane [ms]
-

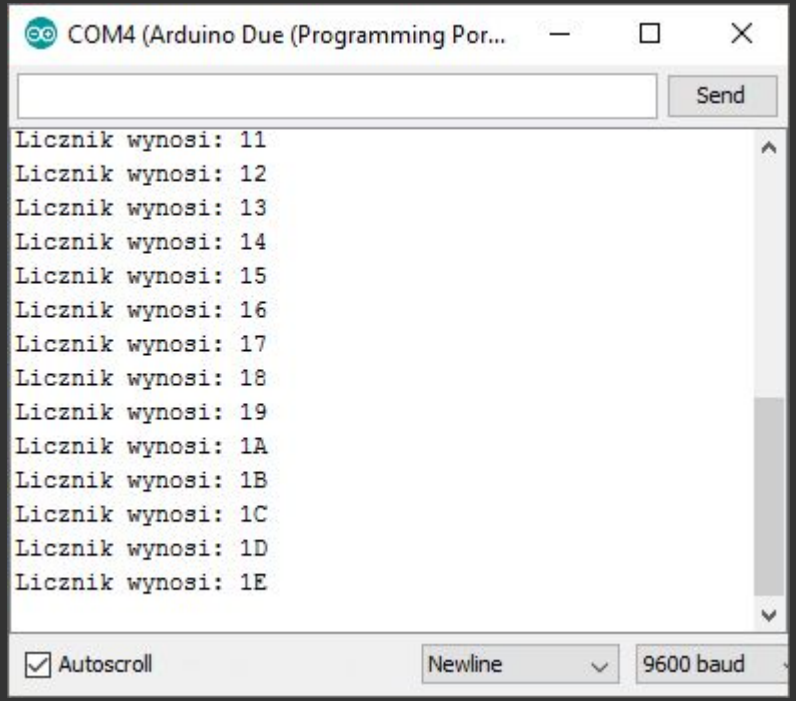
Arduino API - obiekt **Serial** - komunikacja/**konsola**

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

void loop() {
  static int licznik = 0;
  Serial.print("Licznik wynosi: ");
  Serial.println(licznik++);
  delay(500);
}
```



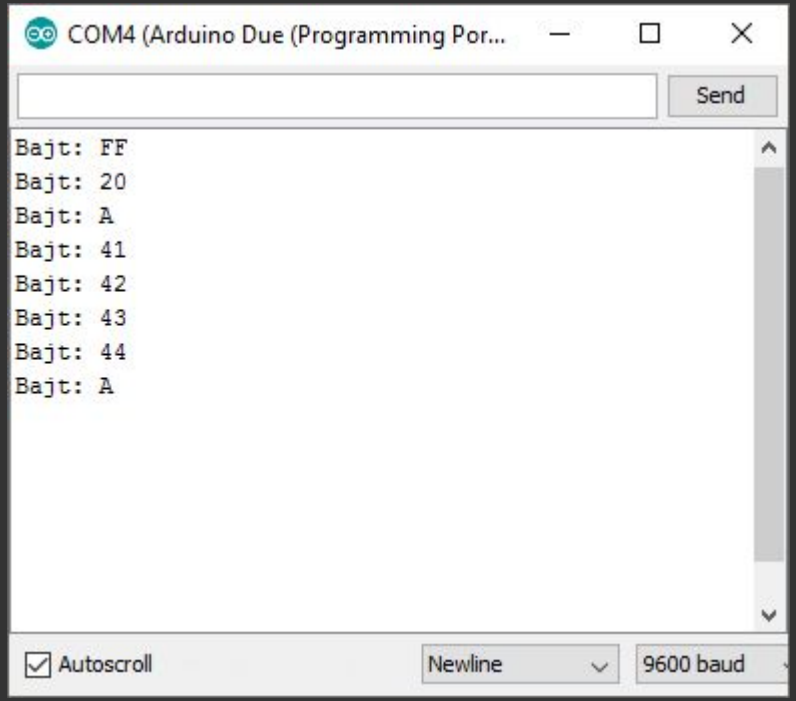
Arduino API - obiekt **Serial** - komunikacja/**konsola**

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

void loop() {
  static int licznik = 0;
  Serial.print("Licznik wynosi: ");
  Serial.println(licznik++, HEX);
  delay(500);
}
```



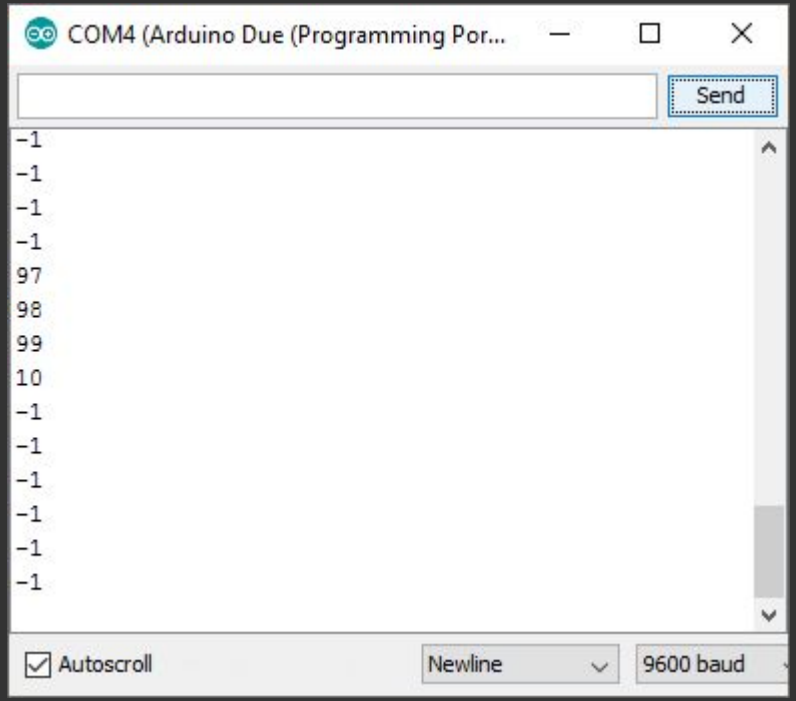
Arduino API - obiekt **Serial** - komunikacja/**konsola**

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

void loop() {
  if (Serial.available() > 0) {
      byte b = Serial.read();

      Serial.print("Bajt: ");
      Serial.println(b, HEX);
  }
}
```



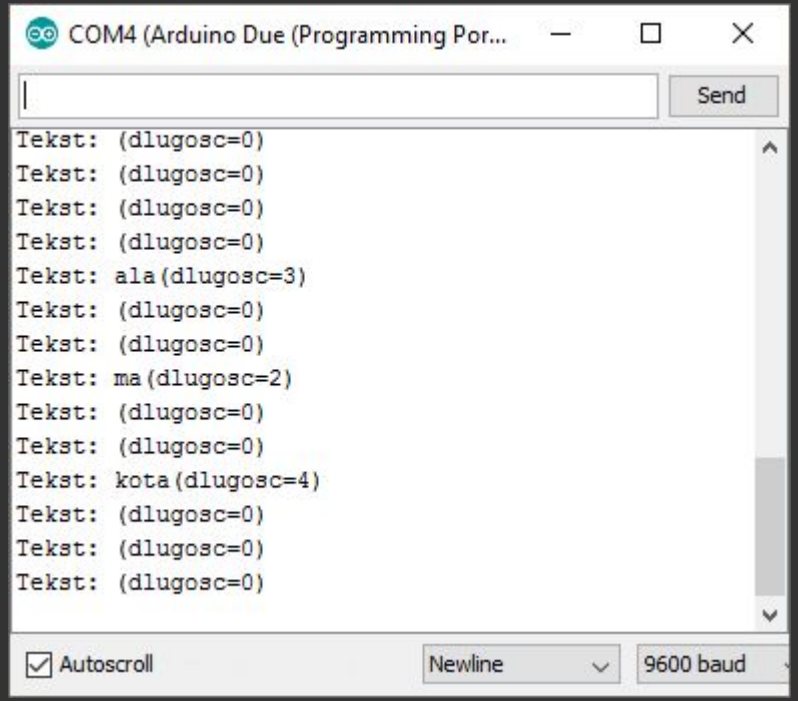
Arduino API - obiekt **Serial** - komunikacja/**konsola**

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

void loop() {
  int data = Serial.read();
  Serial.print(data);
  Serial.println();
  delay(100);
}
```



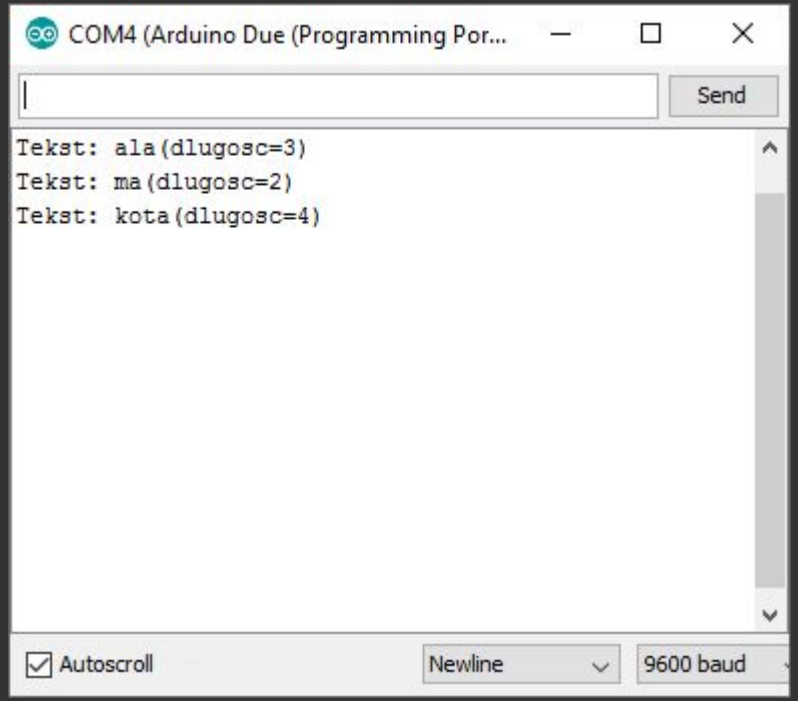
Arduino API - obiekt **Serial** - komunikacja/**konsola**

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

void loop() {
  String text = Serial.readStringUntil('\n');
  Serial.print("Tekst: ");
  Serial.print(text);
  Serial.print("(dlugosc=");
  Serial.print(text.length());
  Serial.println(")");
}
```



Arduino API - obiekt **Serial** - komunikacja/**konsola**

```
void setup() {
  Serial.begin(9600);
  Serial.setTimeout(-1);
}

void loop() {
  String text = Serial.readStringUntil('\n');
  Serial.print("Tekst: ");
  Serial.print(text);
  Serial.print("(dlugosc=");
  Serial.print(text.length());
  Serial.println(")");
}
```



Arduino API - Funkcje do obsługi czasu

Opóźnienie czasowe:

```
void delay(unsigned long ms);
void delayMicroseconds(unsigned int mc);
```

Wstrzymuje działanie programu na *ms* milisekund lub *mc* mikrosekund.
1 sekunda = 1,000 ms = 1,000,000 us

Punkt czasowy:

```
unsigned long millis(void);
unsigned long micros(void);
```

Pobiera wartość czasu od startu/restartu urządzenia (brak RTC!)

Arduino API - Funkcje matematyczne

Arduino/hardware/tools/avr/avr/include/math.h

Makra:

- min(a, b), max(a, b), abs(x), constraint(x, low, high)

Funkcje:

- **float/float**: sqrtf
- **double/double**: pow, sqrt, sin, cos, tan, floor, ceil, exp, log, log10
- **int/float**: isnan, isinf
- **long/long**: map(value, sLow, sHigh, dLow, dHigh)

Arduino API - Funkcje matematyczne

Makra - uwaga na efekty uboczne

```
int a = 10, b = 20;
int z = min(a++, b++);
```
a = ?, b = ?, z = ?

```
int a = 10, b = 20;
int z = (a++) > (b++) ? (a++) : (b++)
```
a = ?, b = ?, z = ?

Arduino API - Generator liczb pseudolosowych

Losuj wartość z zakresu 0 - high:

```
long random(long high);
```

Losuj wartość z zakresu low - high:

```
long random(long low, long high);
```

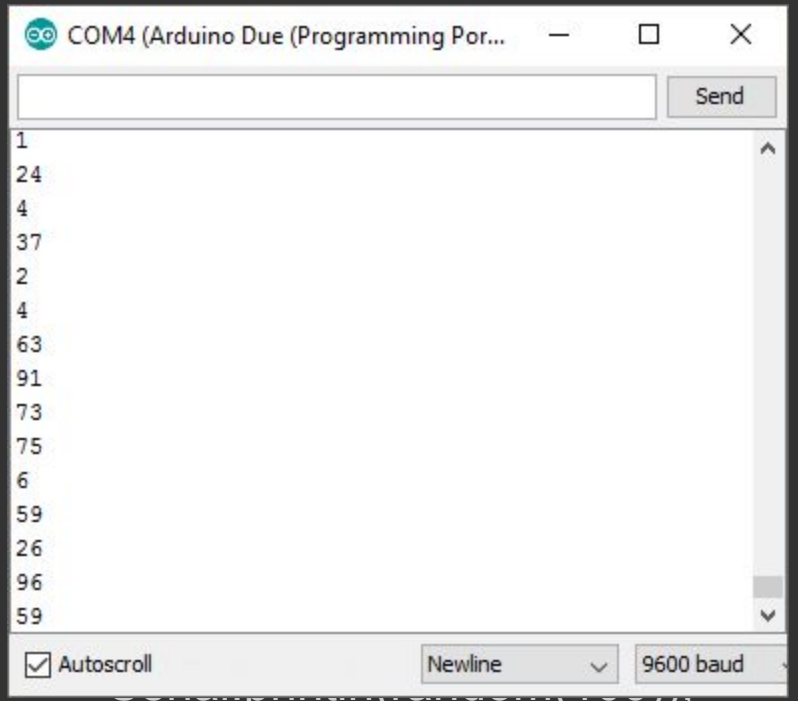
Ustaw wartość początkową generatora:

```
void randomSeed(unsigned long);
```

Odpowiedniki w CRT: `rand()`, `srand()`.

Arduino API - Generator liczb pseudolosowych

```
void setup(void)
{
	randomSeed(0);
	Serial.begin(9600);
}

void loop(void)
{
	Serial.print(random(100));
}
```



Arduino API - operacje na bitach (makra!)

Odczytaj wartość bitu (x=0/1):

```
x = bitRead(number, bit);
```

Ustaw bit:

```
bitSet(number, bit);
```

Zeruj bit:

```
bitClear(number, bit);
```

Zapisz nową wartość bitu:

```
bitWrite(number, bit, newvalue);
```

Arduino API - IO

Wejście/wyjście cyfrowe (binarne)

- **Wyjście**: Pozwala na wystawienie wartości logicznej (sygnału cyfrowego), do wykorzystania przez inne urządzenia (np. włącz/wyłącz silnik).
- **Wejście**: Pozwala na pobranie wartości logicznej od zewnętrznego urządzenia (czujnik: czy okno zamknięte?)
- Możliwe stany logiczne: 0 (LOW, fałsz) oraz 1 (HIGH, prawda)

Wartości napięciowe sygnałów:
Arduino DUE: LOW = 0V, HIGH = 3V3
Arduino UNO: LOW = 0V, HIGH = 5V

Uwaga na wydajność prądową wyjść cyfrowych!

Arduino API - IO - wejścia/wyjścia binarne

Ustawienie kierunku przepływu informacji dla danego pinu:

`pinMode(pin, mode)`

`pin` - numer pinu 0-53

`mode` - tryb: INPUT, OUTPUT, INPUT_PULLUP

Pobranie stanu logicznego (wartości sygnału) danego pinu:

`bool digitalRead(pin)`

Ustawienie stanu logicznego (wartości sygnału) danego pinu:

`void digitalWrite(pin, value)`

value - wartość logiczne: LOW, HIGH

Arduino API - IO - wejścia/wyjścia binarne

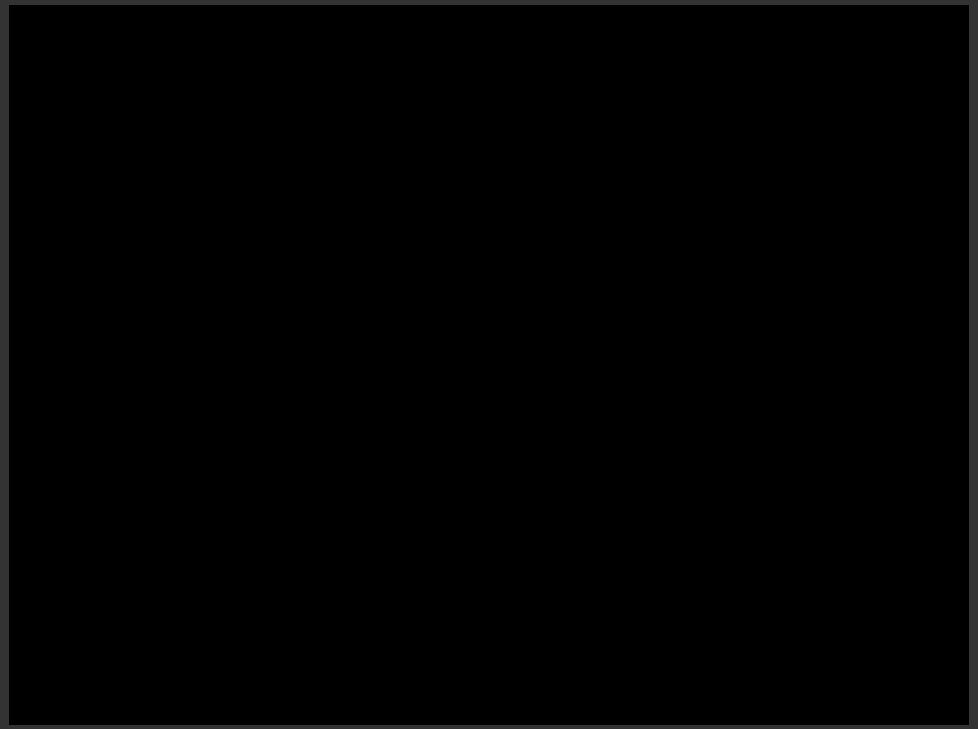
Obciążalność prądowa pinów Arduino DUE

Prąd	Kierunek=**WYJŚCIE**; Numer pinu - parametr pin w funkcji pinMode												
3 mA	0		2		13			16-20		43		52	
15 mA		1		3-12		14	15		23-42		44-51		53

Prąd	Kierunek=**WEJŚCIE**; Numer pinu - parametr pin w funkcji pinMode												
6 mA	0		2		13			16-20		43		52	
9 mA		1		3-12		14	15		23-42		44-51		53

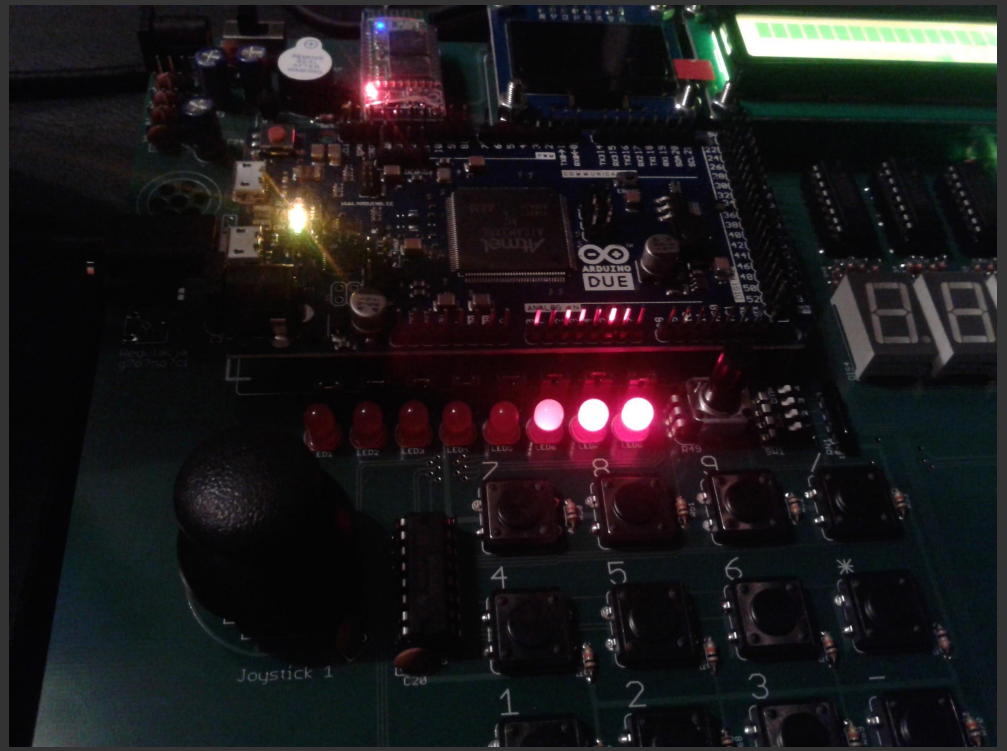
Arduino API - IO - Przykład

```
int ledPin = 5;
void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```



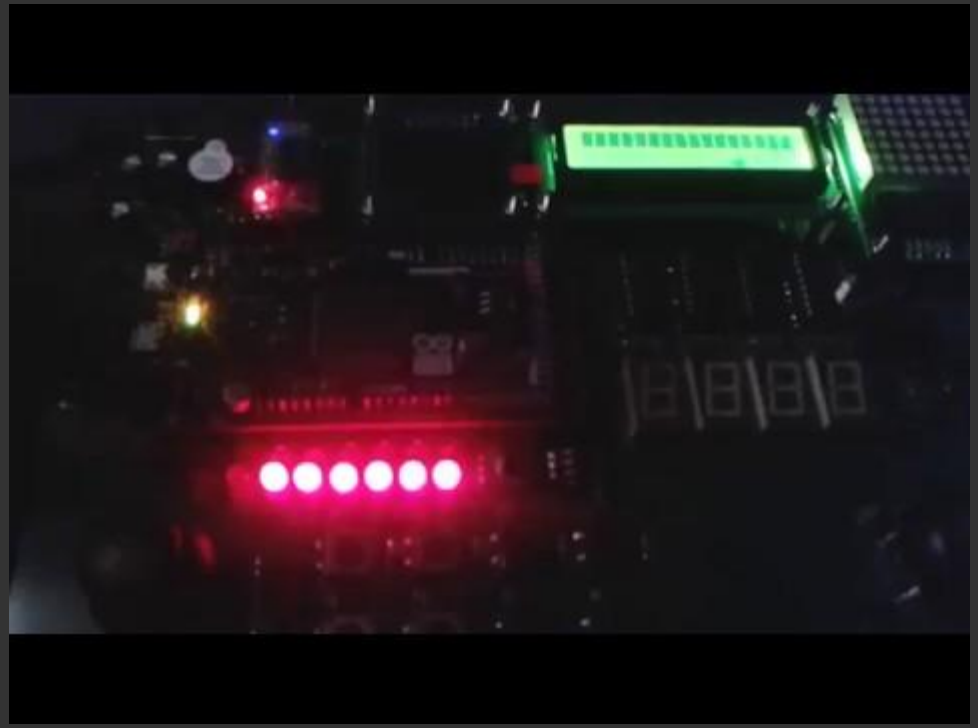
Arduino API - IO - Przykład

```
void setup() {
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
}
void loop() {
  digitalWrite(2, 1);
  digitalWrite(3, 1);
  digitalWrite(4, 1);

  delayMicroseconds(20);
  digitalWrite(4, 0);

  delayMicroseconds(60);
  digitalWrite(3, 0);

  delayMicroseconds(150);
  digitalWrite(2, 0);

  delayMicroseconds(770);
}
```



Arduino API - IO - Przykład

```
void setup() {
  for (int i = 0; i < 8; i++)
      pinMode(2 + i, OUTPUT);
}

void loop() {
  static byte value = 0;
  for (int i = 0; i < 8; i++)
      digitalWrite(2 + i, bitRead(
            value, i));

  value++;
  delay(100);
}
```



Arduino API - IO

Wejście/wyjście analogowe

- **Wyjście**: Pozwala na wygenerowanie sygnału **napięciowego**, do wykorzystania przez inne urządzenia (np. sterowanie poziomem gazu w samochodzie)
- **Wejście**: Pozwala na zmierzenie wartości napięcia, podanego na wejście przez urządzenie zewnętrzne (czujnik jasności).
- Możliwy zakres wartości: 0 - 1023 (10 bitów) lub 4095 (12 bitów)
- Liczba wejść: **12** (A0-A11), wyjść: **2** (DAC0, DAC1)
- Wyjścia i wejścia to **różne** piny

Wartości napięciowe sygnałów w Arduino DUE:
0 = 0V; 512 = 1,65V; 1023 = 3,3V (dla 10 bitów)
0 = 0V; 2048 = 1,65V; 4095 = 3,3V (dla 12 bitów)

Arduino API - wejście analogowe

Odczytanie wartości napięcia w jednostkach przetwornika:

```
int analogRead(int pin);
```

Ustawienie liczby bitów przetwornika (10 lub 12)

```
int analogReadResolution(int bits);
```

- `bits` = **10** -> max=**1024**; **12** -> max=**4096**

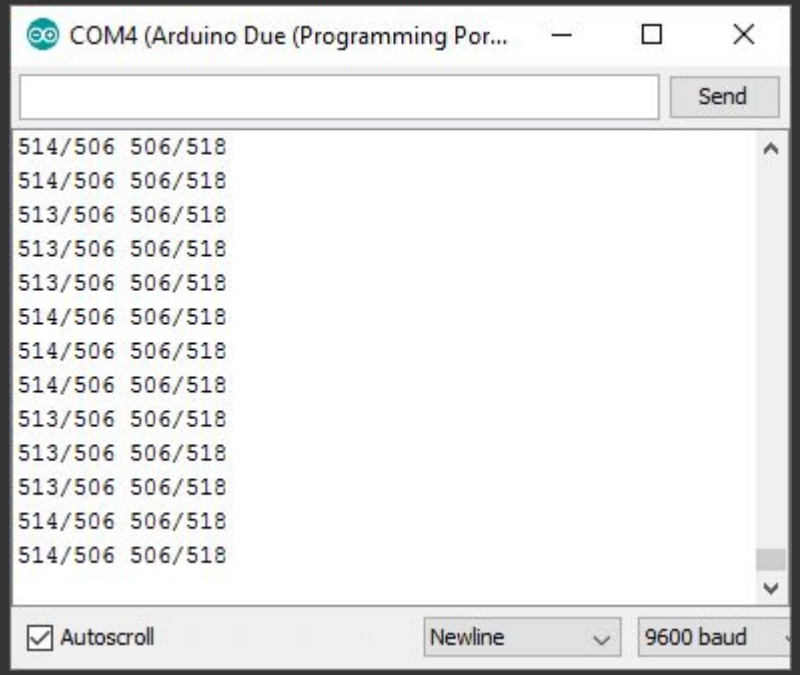
Dostępne wejścia analogowe (piny): **A0 - A11 (12 pinów)**

Przykład:

```
int value = analogRead(A3);
float voltage = 3.3f * (float)value / 1024.0f;
```

Arduino API - wejście analogowe (napięciowe)

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


void loop() {


  int x1 = analogRead(A1);
  int y1 = analogRead(A0);
  int x2 = analogRead(A11);
  int y2 = analogRead(A10);

  Serial.print(x1); Serial.print('/'); Serial.print(y1);
  Serial.print(' ');
  Serial.print(x2); Serial.print('/'); Serial.print(y2);
  Serial.print('\n');
}
```



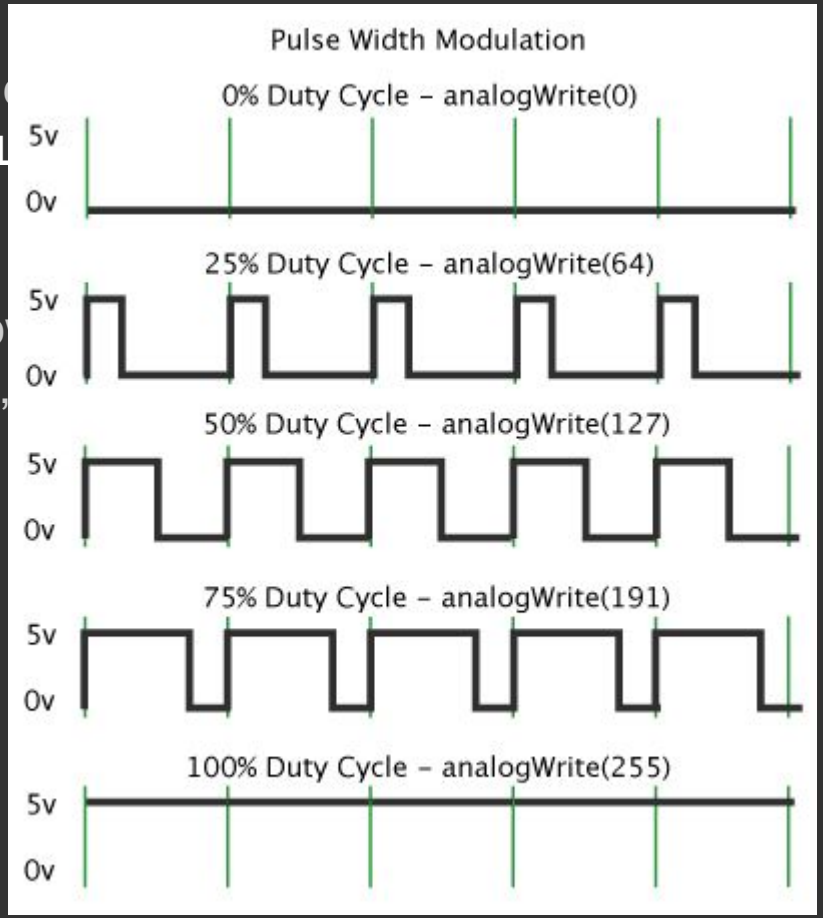
Arduino API - wyjście analogowe

Ustawienie wartości analogowej val na pini

```
int analogWrite(int pin, int val
```

Dostępne wyjścia analogowe:

- 0 (DAC0), 1 (DAC1) - wyjścia napięcio
- 2 … 13 - generatory PWM, max = 255,



Arduino API - wyjście analogowe (PWM)

```
void setup() {
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);

  analogWrite(4, 5);
  analogWrite(3, 20);
  analogWrite(2, 59);
}

void loop() {
}
```

```
void setup() {
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
}
void loop() {
  digitalWrite(2, 1);
  digitalWrite(3, 1);
  digitalWrite(4, 1);

  delayMicroseconds(20);
  digitalWrite(4, 0);

  delayMicroseconds(60);
  digitalWrite(3, 0);

  delayMicroseconds(150);
  digitalWrite(2, 0);

  delayMicroseconds(770);
}
```

Arduino API - Przerwania

Wystąpienie przerwania powoduje **wstrzymanie** aktualnie wykonywanego kodu oraz wykonanie **procedury obsługi przerwania** (**ISR** - Interrupt Service Routine).

Podstawowe źródła przerwań:

- Wejściowe piny cyfrowe
- Timery/liczniki

Przerwania są domyślnie **aktywne**.

Przerwania nie są od wykonywania **długich operacji** (np. mnożenie wartości typu double) a jedynie do **zmiany stanu programu**.

Pamiętaj o `volatile`!

Arduino API - Przerwania

Aktywacja/deaktywacja

`interrupts();`
Aktywuje kontroler przerwań

`noInterrupts();`
Deaktywuje kontroler przerwań. Niektóre funkcje (np. komunikacja) może przestać działać.

Przykład:
noInterrupts();
    // tutaj kod, który nie może być przerwany, precyzyjny pomiar czasu, itd…
interrupts();

Arduino API - Przerwania WE/WY

Dodanie obsługi przerwania

`attachInterrupt(intno, ISR, mode);`
Ustaw funkcję obsługi ***ISR*** przerwania o wektorze ***intno***. Funkcja ***ISR*** będzie uruchamiana w zależności od trybu ***mode***.

Wektor przerwania na podstawie numeru pinu:
`digitalPinToInterrupt(pin)`

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

Konwerter **digitalPinToInterrupt** nie jest wymagany dla DUE ale zalecany, ze względu na możliwe problemy z kompatybilnością.

Arduino API - Przerwania WE/WY

Dostępne wartości parametru ***mode***:

- **LOW** - gdy pin jest w stanie niskim (LOW, 0)
- **HIGH** - gdy pin jest w stanie wysokim (HIGH, 1)
- **CHANGE** - gdy na pinie wykryto zmianę stanu (0->1 lub 1->)
- **RISING** - gdy wykryto zbocze narastające (0->1)
- **FALLING** - gdy wykryto zbocze opadające (1->0)

Arduino API - Przerwania WE/WY

Usunięcie obsługi przerwania

`detachInterrupt(intno);`
Wyłącz obsługę przerwania o wektorze ***intno***.

Wektor przerwania na podstawie numeru pinu:
`digitalPinToInterrupt(pin)`

Wykorzystanie:
`detachInterrupt(digitalPinToInterrupt(pin));`

Arduino API - Przerwania WE/WY

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

  for (int i = 46; i <= 49; i++)
        pinMode(i, INPUT);

  attachInterrupt(digitalPinToInterrupt(46), key_right, FALLING);
  attachInterrupt(digitalPinToInterrupt(47), key_up, FALLING);
  attachInterrupt(digitalPinToInterrupt(48), key_left, FALLING);
  attachInterrupt(digitalPinToInterrupt(49), key_down, FALLING);
}

void loop() {
}
```

```
void key_right(void) {
  Serial.println("right");
}

void key_up(void) {
  Serial.println("up");
}

void key_left(void) {
  Serial.println("left");
}

void key_down(void) {
  Serial.println("down");
}
```

Arduino API - Przerwania WE/WY

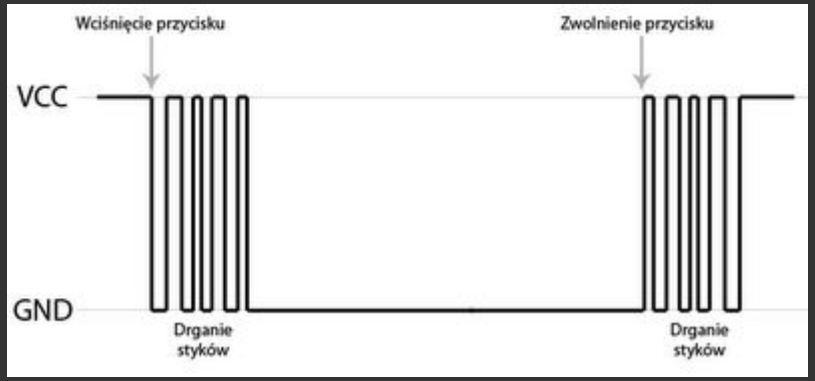
```
volatile byte state = 0;
void blink(void);

void setup(void) {
  pinMode(47, INPUT);                 // strzałka “w górę”
  pinMode(3, OUTPUT);                 // dioda LED
  attachInterrupt(digitalPinToInterrupt(47), blink, CHANGE);
}

void loop(void) {
  digitalWrite(3, state);
}

void blink(void) {
  state = !state;
}
```

A czy **digitalWrite** można użyć w **blink**?

Arduino API - Przerwania WE/WY - **Problem z przyciskami**

Źródła:

- drganie mechaniczne zestyków,
- zaburzenie parametrów elektrycznych podczas ruchu powierzchni kontaktowych

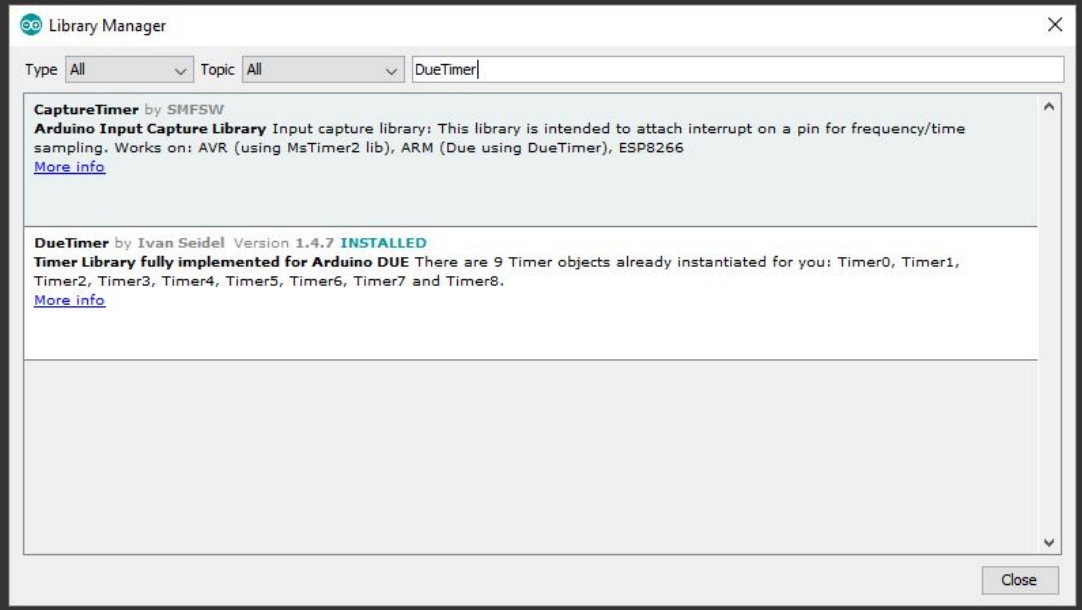


Arduino API - Przerwania timerów - biblioteka **DueTimer**

GitHub: https://github.com/ivanseidel/DueTimer

Instalacja biblioteki **DueTimer**:

1. **Sketch -> Include Library -> Manage libraries**
2. **Sketch -> Include Library -> DueTimer**
   Wynik:
   #include <DueTimer.h>



Arduino API - DueTimer

- Biblioteka zewnętrzna, dedykowana wyłącznie dla Arduino DUE (nieportowalna ze względu na sprzęt)
- Na starcie udostępnia 9 timerów: **Timer0** … **Timer8**, będących obiektami klasy **DueTimer**
- W przypadku korzystania z biblioteki **Servo**, niedostępne są timery 0-5 (należy aktywować dyrektywę #define USING_SERVO_LIB true w pliku nagłówkowym DueTimer.h)
- Wszystkie metody klasy DueTimer zwracają DueTimer& (za wyjątkiem getFrequency i getPeriod)
- Podobnie jak dla przerwań WE/WY, timery mogą mieć tylko jedną funkcję ISR

Arduino API - DueTimer

Metody klasy DueTimer:

- `DueTimer& getAvailable()` - Zwraca ref. do pierwszego wolnego timera
- `attachInterrupt(void (*isr)())` - Podłącza obsługę **isr** przerwania
- `detachInterrupt()` - Usuwa obsługę przerwania
- `start(long microseconds = -1)` - Uruchom timer; opcjonalnie ustaw okres w mikrosekundach.
- `stop()` - Zatrzymaj timer
- `setFrequency(double frequency)` - Ustaw częstotliwość timera [Hz]
- `double getFrequency()` - Pobierz częstotliwość timera
- `setPeriod(long microseconds)` - Ustaw okres timera [us]
- `long getPeriod()` - Pobierz okres timera [us]

Arduino API - DueTimer

```
#include <DueTimer.h>
void setup() {
  Serial.begin(9600);
  Timer4.attachInterrupt(handler).start();
}

void loop() {
}

void handler(void) {
  Serial.print("test");
}
```

```
#include <DueTimer.h>
void setup() {
  Serial.begin(9600);
  Timer4.attachInterrupt(handler);
  Timer4.setFrequency(1);
  Timer4.start();
}

void loop() {
}

void handler(void) {
  Serial.print("test");
}
```

Płytka edukacyjna - dostępne zasoby 1/2

Diody LED (L):
- L8=p2, L7=p3, L6=p4, L5=p5, L4=p6, L3=p7, L2=p8, L1=p9

Serwonapędy (S):
- S1=p10, S2=p11, S3=p12, S4=p13

Dźwięk:
- Brzęczyk=p24, Audio=dac0

Joysticki:
- J1X=a11, J1Y=a10, J2X=a1, J2Y=a0

Potencjometr
- POT=a9

Klawiatura - strzałki:
- Prawo=p46, Góra=p47, Lewo=p48, Dół=p49

Przełączniki konfiguracyjne
- SW1=p50, SW2=p51, SW3=p52, SW4=p53

Czujnik ultradźwiękowy:
- Ping=p45, Echo=p44

Płytka edukacyjna - dostępne zasoby 2/2

Czujnik ultradźwiękowy:
- Ping=p45, Echo=p44

Wyświetlacz 7SEG:
- Dane=p38, Zegar=p39, Strob1=p33, Strob2=p34, Strob3=p35, Strob4=p36, OE=p37

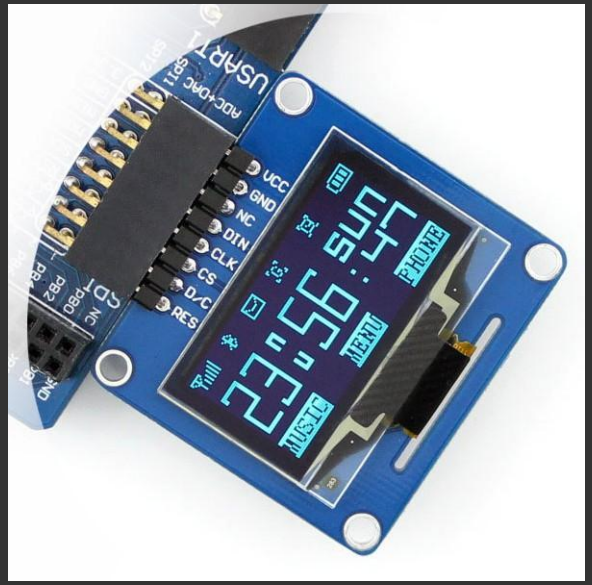
Wyświetlacz alfanumeryczny LCD (2x16):
- RS=p26, E=p28, D4=p29, D5=p30, D6=p31, D7=p32

Klawiatura 4x4:
- Zegar=p41, DaneWe=p42, Zatrzask=p40

Wyświetlacz matrycowy 8x8:
- CS=43, standardowy interfejs SPI

Wyświetlacz OLED

Rozdzielczość: **128x64** (8192 punkty)
Technologia: **OLED**
Sterowanie z Arduino: klasa **SH1106_SPI_FB**
Biblioteka: https://github.com/InteligentneSystemyAutonomiczneIIS/Fast-SH1106-library-for-Arduino-Due



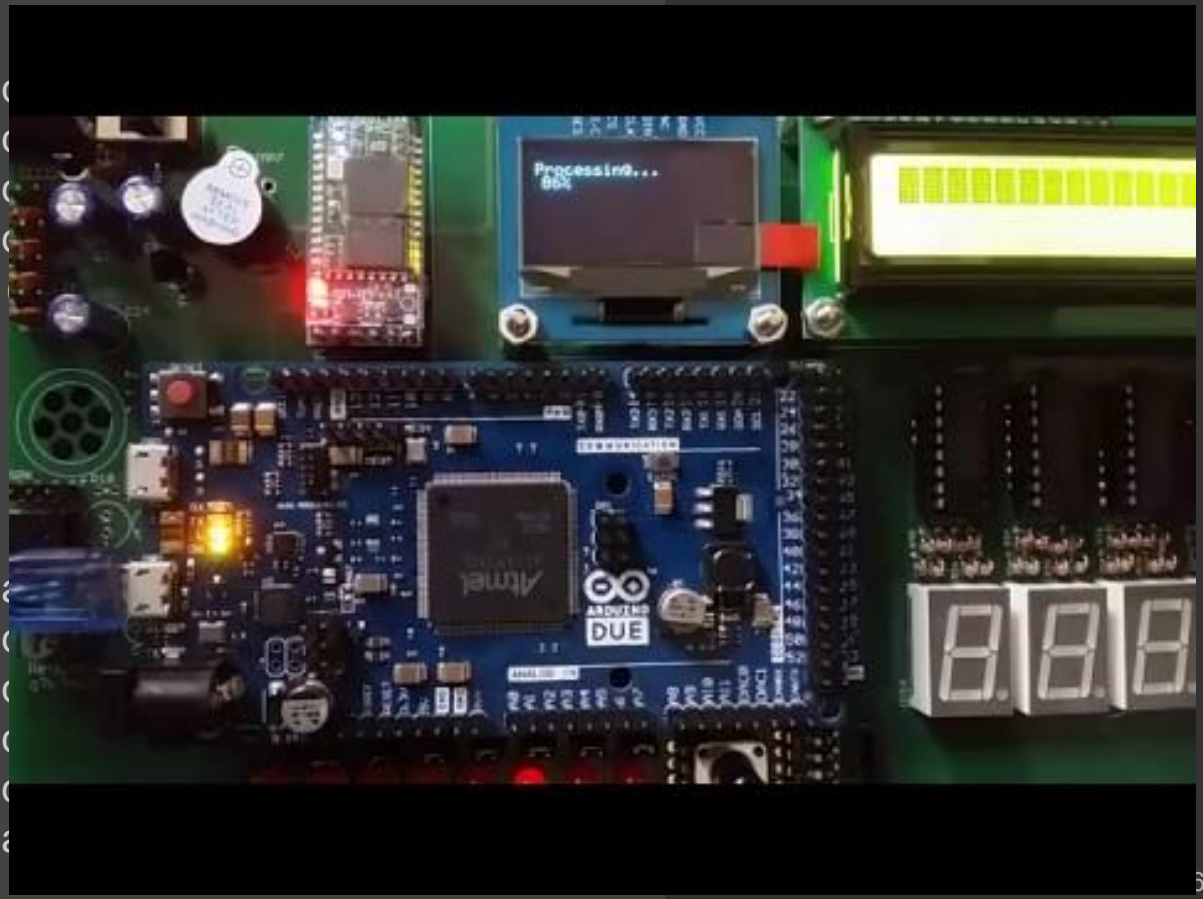
Obsługa OLED - Klasa **SH1106_SPI_FB**

```
#include <SH1106_SPI.h>

SH1106_SPI_FB oled;

void setup(void) {
  oled.begin();
  oled.print("Start");
  oled.renderAll();
  delay(1000);
}

void loop() {
  oled.clear();
  oled.gotoXY(0, 1);
  oled.print("Processing...");
  oled.renderAll();
  for (int i = 0; i <= 100; i++) {
        oled.gotoXY(4, 2);
        oled.print(i); oled.print("%  ");
        oled.renderAll();
        delay(100);
  }
  delay(1000);
  oled.clear();
  oled.gotoXY(10, 3);
  oled.print("READY!");
  oled.renderAll();
  delay(1000);
}
```

Obsługa OLED - Klasa **SH1106_SPI_FB**

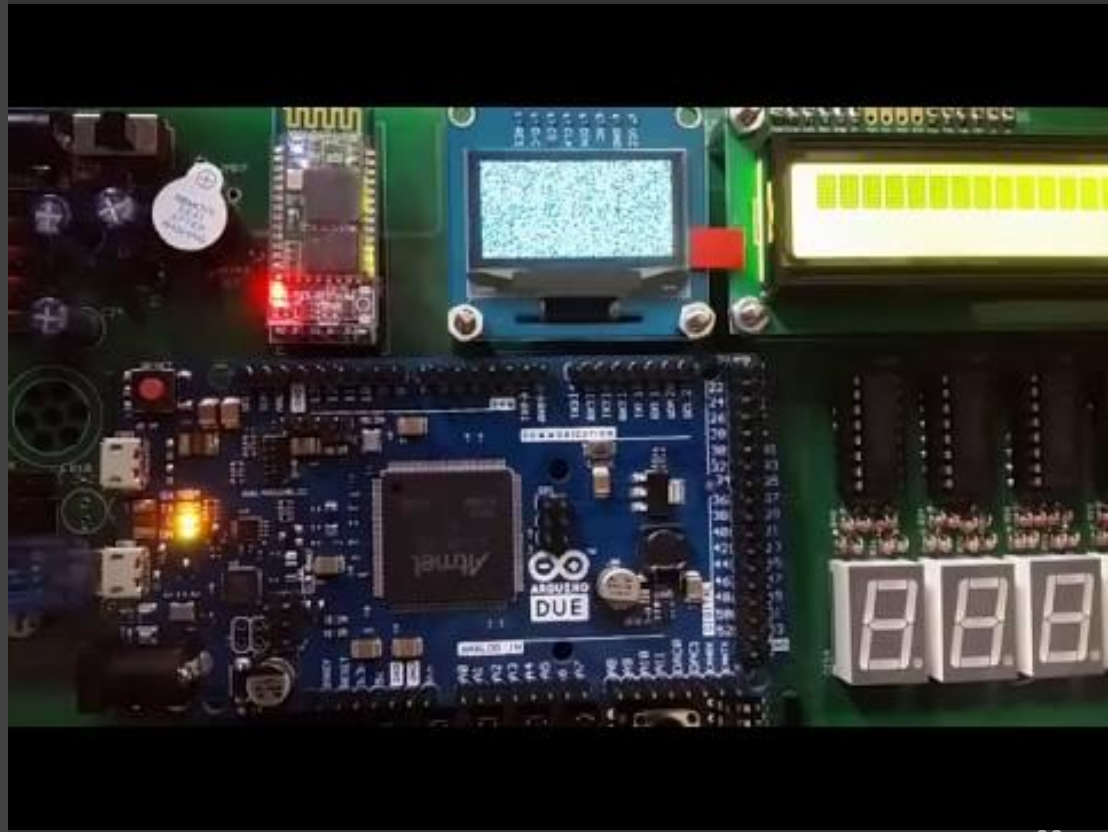
```
#include <SH1106_SPI.h>

SH1106_SPI_FB oled;

void setup(void) {
  oled.begin();
  oled.print("Start");
  oled.renderAll();
  delay(1000);
}
```

```
void
  oled
  oled
  oled
  oled
  for
  }
  dela
  oled
  oled
  oled
  oled
  dela
}
```



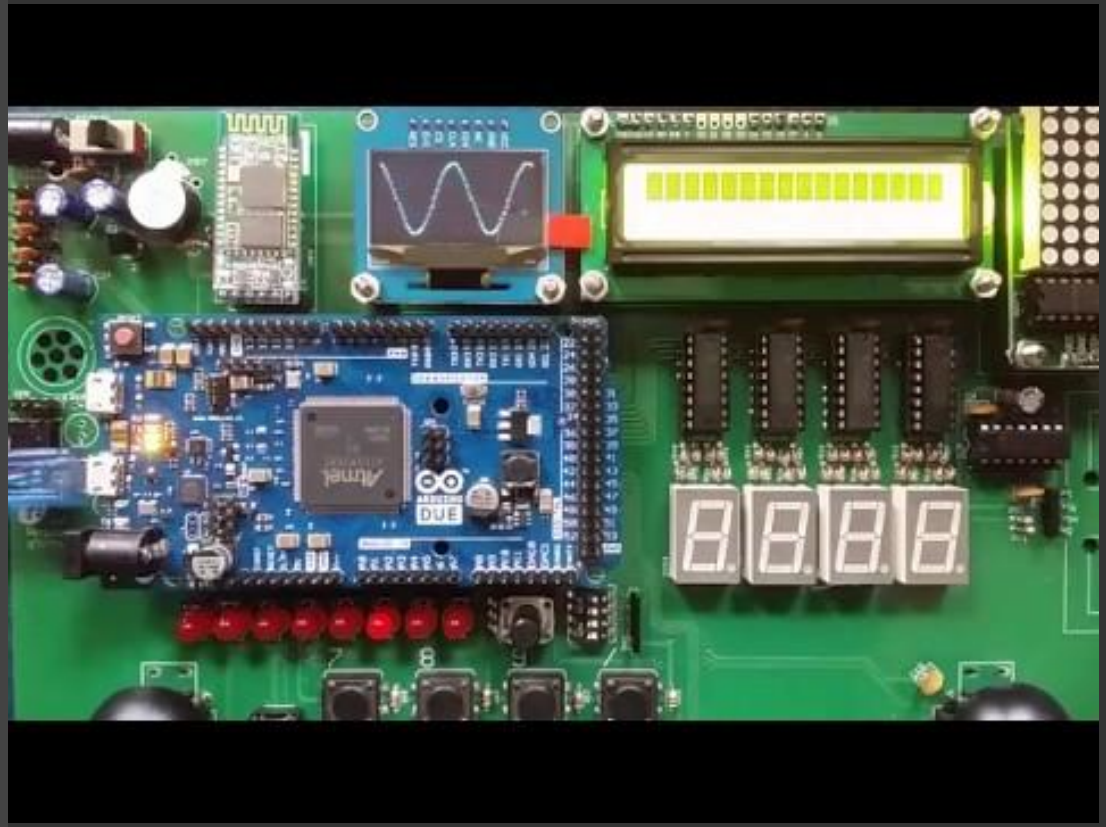
Obsługa OLED - Klasa **SH1106_SPI_FB**

```
#include <SH1106_SPI.h>

SH1106_SPI_FB oled;

void setup(void) {
  oled.begin();
}

void loop() {
  for (int x = 0; x < 128; x++)
        for (int y = 0; y < 64; y++)
        oled.setPixel(x, y, random(2));

  oled.renderAll();
}
```



Obsługa OLED - Klasa **SH1106_SPI_FB**

```
#include <SH1106_SPI.h>
SH1106_SPI_FB oled;
int y[128] = {}, dx = 0;
void setup(void) {
  oled.begin();
}
void draw(bool s) {
  for (int i = 0; i < 128; i++)
        oled.setPixel(i, 32 + y[i], s);
}
void loop() {
  draw(false);
  for (int i = 0; i < 128; i++)
        y[i] = 30*sin((float)(i+dx) / 10.0f);
  dx++;
  draw(true);
  oled.renderAll();
}
```



Obsługa OLED - Klasa **SH1106_SPI_FB**

Dostępne API:

- **begin**() - Inicjuje wyświetlacz (automatyczne czyszczenie zawartości)
- **write**(data) - Wyświetla znak na podstawie kodu data; znaki nie-ascii nie są wyświetlane.
- **clear**() - Czyści zawartość ekranu.
- **gotoXY**(cx,cy) - Ustawia pozycję wyświetlania tekstu na pozycję *x,y*. Uwaga! *cx* i *cy* to nie współrzędne pikselowe, tylko znakowe.
- **setPixel**(x, y, v) - Ustawia piksel (*x, y*) wartością *v*.
- **writeLine**(x1, y1, x2, y2) - Rysuje linię (tylko pionowe/poziome)
- **writeRect**(x, y, w, h, fill) - rysuje/wypełnia prostokąt
- **print**(value) - wyświetla wartość (tekst, liczbę, itd) w aktualnej pozycji
- **println**(value) - działa jak print(); znak \n nie jest obsługiwany

Płytka edukacyjna - zasoby sprzętowe

Definicje słowne w języku C (#define):
https://github.com/InteligentneSystemyAutonomiczneIIS/ISADefinitions

Dziękuję za uwagę