Systemy Wbudowane

Arduino C++

Wersja 2026

dr inż. Marek Wilkus
Wydział Inżynierii Metali i Informatyki Przemysłowej
AGH Kraków
http://home.agh.edu.pl/~mwilkus

Arduino C - stałe

- Unikać redefiniowania istniejących stałych.
- Stosowane dla polepszenia zrozumiałości kodu.
  - Lepiej **`HIGH`** niż `0x01`
  - Lepiej **`INPUT_PULLUP`** niż `0x2`
- Uwzględniają wielkość liter.

Arduino C - Stałe

- true, false
- HIGH
  - Stan wysoki (wyjście)
  - Włączenie 20K rezystorów podciągających (wejście).
- LOW
  - Stan niski (wyjście)
  - Wyłączenie 20K rezystorów podciągających (wejście).
- INPUT, OUTPUT, INPUT_PULLUP

Arduino C - Stałe

- INPUT_PULLUP:

```
pinMode(12, INPUT);
digitalWrite(12,HIGH);
```

- LED_BUILTIN – wyprowadzenie, do którego podłączony jest LED diagnostyczny, w Arduino Uno pin 13.

Arduino C - Stałe

- Matematyczne:

```
#define PI          3.14159265358979323846264338327795
#define HALF_PI     1.57079632679489661923132169163975
#define TWO_PI      6.28318530717958647692528676655901
#define DEG_TO_RAD  0.01745329251994329576923690768488
#define RAD_TO_DEG  57.29577951308232087679815481410
#define EULER       2.71828182845904523536028747135266
```


Funkcje matematyczne

- min(x,y), max(x,y)
- abs(x)
- constrain(x,a,b)
  - x – gdy x ∈ [a..b]
  - a – gdy x<a
  - b – gdy x>b
- map(liczba,fromLow,fromHigh,toLow,toHigh)
  - Mapuje wartość z jednego przedziału na inny
  - **Nie obsługuje liczb zmiennoprzecinkowych! (przycina)**
- pow(podstawa, wykladnik), sqrt(x)
- radians(deg), degrees(rad)
- sq(x) = x*x

Funkcja delay i jej konsekwencje

- Istnieją dwie metody projektowania programu wykonywanego okresowo:
  - Korzystając z delay – w przyszłości trudny do rozbudowy,
  - Korzystając z licznika – wymaga odpowiedniego zaprojektowania (jako maszyna stanowa), lecz umożliwia łatwą rozbudowę,

```
void loop()
{
   digitalWrite(ledPin, HIGH);
   delay(500);
   digitalWrite(ledPin, LOW);
   delay(500);
}
```

Wersja z delay:
- W trakcie 500ms nie możemy nic zrobić (poza przerwaniami)

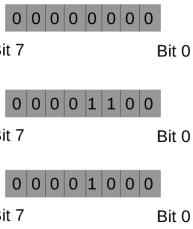
```
unsigned long previousMillis=0;
void loop()
{

  if (millis() - previousMillis >= 500)
  {
    previousMillis = millis();


      digitalWrite(ledPin, !digitalRead(ledPin));
  }
//!!!

}
```

Wersja z millis – kod otwarty na rozbudowę

Operacje na bitach

- bitSet(x,n) – zapisuje 1 na bit n zmiennej x.
- bitClear(x,n) – zapisuje 0.

```
byte k=0;

bitSet(k,2);
bitSet(k,3);

bitClear(k,2);
```



Operacje na bitach

- bitWrite(x,n,b) – zapisuje n-ty bit zmiennej x wartością b (1/0 lub HIGH/LOW).
- bitRead(x,n) – odczytuje wartość n-tego bitu zmiennej x.

Czas

- delay(x), delayMicroseconds(x)
- micros() - liczba µs od uruchomienia Arduino (lub ostatniego przeładowania tego licznika).
- millis() - jak micros(), w milisekundach.

```
byte ledState = LOW;
long previousMillis = 0;
long interval = 500;

...

void loop()
{
  unsigned long currentMillis = millis();
  if(currentMillis - previousMillis > interval)
{
    previousMillis = currentMillis;
    if (ledState == LOW)
      ledState = HIGH;
    else
      ledState = LOW;

    digitalWrite(ledPin, ledState);
  }
}
```


Jeszcze raz przerwania

- Tymczasowe wyłączenie przerwań na wskazanym fragmencie kodu:

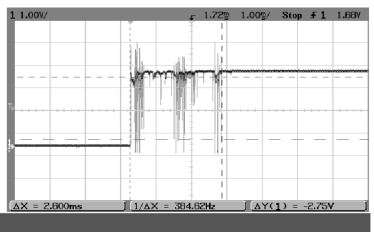
```
noInterrupts();
. . .
interrupts();
```

- Wyłączenie obsługi przerwania:

```
detachInterrupt(nrPrzerwania);
```


Debouncing

- Wciśnięcie przycisku nie skutkuje z reguły pojedynczą zmianą stanu.
- Styki przycisku uderzają o siebie, odbijając się i przez niewielki czas (0.1-1ms) dają zmienne stany,
- Działanie oparte o ilość wciśnięć przycisku działa więc w sposób niekontrolowany.
- Niezbędna jest eliminacja drgań.
- Można zrealizować ją sprzętowo (kondensator), lub programowo.



Programowy debouncing

- Założenie: Sprawdzamy okresowo stan linii.
- Jeżeli linia pozostaje zmieniona przez konkretną ilość sprawdzeń → przycisk rzeczywiście został wciśnięty/puszczony.
- Wymagania:
  - Zapisanie dotychczasowego stanu przycisku,
  - Licznik ilości sprawdzeń.
  - Zewnętrzna zmienna do zapisu i końcowego stanu przycisku
- W implementacji możemy zmieścić się w byte przez użycie pól bitowych!

Programowy debouncing

- ALGORYTM:
  - jeżeli (aktualny stan przycisku != zapisany stan)
    - licznik++;
    - jeżeli (licznik >= 4)
      - zapisany stan = aktualny stan przycisku
      - ustaw odpowiednio zewnętrzną zmienną
      - licznik = 0
  - W przeciwnym wypadku
    - licznik = 0

Programowy debouncing

- ALGORYTM:
  - jeżeli (aktualny stan przycisku != zapisany stan)
    - licznik++;
    - jeżeli (licznik >= 4)
      - zapisany stan = aktualny stan przycisku
      - **ustaw odpowiednio zewnętrzną zmienną**
      - licznik = 0
  - W przeciwnym wypadku
    - licznik = 0

```
state=digitalRead(7);
if (!state)
{
  button1=1;
}
count=0;
```

Zmieniamy stan na „aktywny" gdy przycisk został wciśnięty.
Brak efektu powtarzania.

```
state=digitalRead(7);
button1=state;
count=0;
```

Cały czas „publikujemy" stan. Wciśnięcie powoduje b. częste powtarzanie.

Programowy debouncing

```
//Kod dla debouncingu jednego przycisku
//Funkcja wywołuje się co iteracja głównej pętli.
//wynik sprawdzamy w zmiennej globalnej button1.
void debounce()
{
  static byte count=0;       //ilość sprawdzeń
  static byte state=0;       //Stan przycisku. Zmienne STATYCZNE zachowują wartości!
  if (digitalRead(7)!=state) //Jeżeli stan na wejściu zmienił się
  {
    count++;                 //ilość++
    if (count>=4)            //przez >=4 sprawdzenia przycisk nie zmianił swojego stanu
    {
        state=digitalRead(7); //zapisz nowy stan jako trwały
        if (!state)
        {             //Prosze nie używac button1=!state, bo go wyzeruje!
          button1=1;  //zerujemy my gdy uznamy, ze wtkonaliśmy to co jest
        }             //niezbędne przy przyciśniętym przycisku.
        count=0;          //zliczanie możliwe od nowa
    }
  }
  else  //stan pozostaje cały czas taki sam, nic nie zliczamy
  {
    count=0;
  }
}
```


Programowy debouncing

- Użycie:

```
void loop() {

  debounce();
  if (button1)
  {
    Serial.println("BUTTON1 DOWN!");
    button1=0;
  }

  |
```

Wywołanie funkcji debounce, ustalenie stanu przycisku do button1.

Obsługa zdarzenia w button1

Skasowanie „znacznika" po obsłużeniu zdarzenia.

Reszta kodu.

Oszczędzanie RAMu

- Zastosowanie pól bitowych:
  - -2B z zajętego RAMu (mamy go 2kB)
  - +15-17B zajętej pamięci programu (mamy jej jednak aż 32kB)

```
struct button_db
{
  unsigned int count   : 4;
  unsigned int state   : 1;
  unsigned int pressed : 1;
} debounced;
```

Licznik: 4 bity, 0..15

Zapamiętany stan: 1 bit, 0..1

Ostateczny stan przycisku: 1 bit, 0..1

Biblioteki

- Wbudowane pozwalają na obsługę:
  - EEPROMu,
  - Zaawansowanych protokołów szeregowych,
  - Wyświetlacza LCD,
  - Kart SD,
  - Interfejsu SPI,
  - Portu szeregowego na dowolnym wyprowadzeniu,
  - Specjalistycznych urządzeń zewnętrznych.

Biblioteka EEPROM

```
#include <EEPROM.h>

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

int a=0;
int v;
int poprzednie=0;


void serialEvent()
{
  if (Serial.available()>0)
  {
    int k=Serial.parseInt();
    EEPROM.write(a,k);
  }
}
```

Adres w EEPROM (0)

Adres w EEPROM (0)

Wartość (bajt!)

Adres w EEPROM (0)

```
void loop()
{
  v=EEPROM.read(a);
  if (poprzednie!=v)
  {
    Serial.println(v);
    poprzednie=v;
  }

}
```

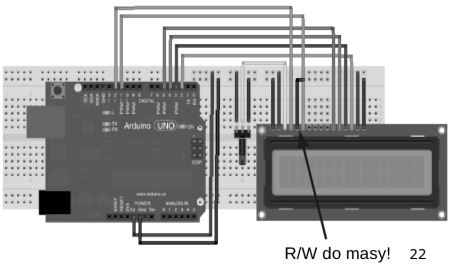

EEPROM – inne polecenia

- EEPROM.get(adres, dane)
  jak EEPROM.read, ale pozyskuje odpowiednią ilość bajtów dla danego typu.
- EEPROM.put(adres, dane)
  - jak EEPROM.write, ale zapisuje odpowiednią ilość danych dla danego typu.

- UWAGA:
  Nie ma żadnych zabezpieczeń przed nadpisaniem wartości "od środka" przez podanie niewłaściwego adresu. **Programista musi wiedzieć jakiej długości jest zapisywana zmienna i gdzie ustawić następny adres.**

Podłączanie wyświetlacza LCD

- Wymagania:
  - +5V, GND
  - Zasilanie LED podswietlenia (opcjonalne)
  - Regulacja kontrastu (potencjometr)
  - Enable
  - Register Select
  - R/W (LOW)
  - 4 bity szyny danych



R/W do masy!

Podłączanie wyświetlacza LCD

- Zapis wartości do rejestru danych + zapis rozkazu do rejestru rozkazów (linia RS)
- Dla wielu wyświetlaczy zgodnych z Hitachi – wbudowana biblioteka.

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  lcd.begin(16, 2);
  lcd.print("hello, world!");
  lcd.setCursor(0, 1);
  lcd.print("Dolna linia");
}
```


Podłączanie wyświetlacza LCD

RS Enable D4 D5 D6 D7

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

Ilość kolumn Ilość wierszy

```
void setup() {
  lcd.begin(16, 2);
  lcd.print("hello, world!");
  lcd.setCursor(0, 1);
  lcd.print("Dolna linia");
}
```

Kolumna wiersz

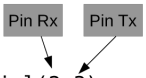
LCD - drukowanie

```
int i;
  ...
i=2;
  ...
lcd.print("LICZBA TO ");
lcd.print(i);
  …
```


```
float k=32.23;
lcd.print(k,1);
lcd.setCursor(0,1);
lcd.print(k,2);
```

Arduino i drugi RS232

- Software'owy RS232: Biblioteka SoftwareSerial.
  - Brak buforowania.
- Użycie:




```
SoftwareSerial mySerial(2,3);

//w Setup:

mySerial.begin(4800);
```

A dalej jak z typowym portem szeregowym.

Przykładowa implementacja protokołu

```
SoftwareSerial mySerial(10, 11);

void setup()
{
  Serial.begin(9600);   //inicjalizacja sprzętowego
  mySerial.begin(4480); //inicjalizacja programowego
}
```

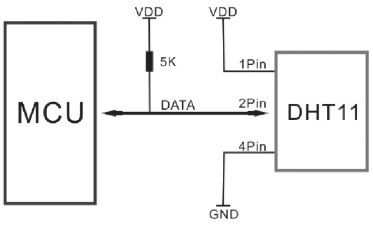
```
void loop()
{
  if (mySerial.available())  //Parsowanie polecen z zewnątrz
  {
   byte k=mySerial.read();
   if (k=='I')               //IDENTIFY
   {
    mySerial.print('2');
   }
   if (k=='R')               //READ
   {
     mySerial.print(temp);
   }
  }
//-----------------------------------------------------------------------------
 if (Serial.available())    //Parsowanie polecenia od użytkownika
 {
  if (Serial.read()=='G')   //Polecenie: Odczytaj wszystkie wyniki
  {
   Serial.print(temp);              //Pokaz temperature
   Serial.print(" but on device "); //ale na urządzeniu numer...

    mySerial.print('I');             //na szybko odczytaj numer podłączonej płytki
    int q=0;
    while ((!mySerial.available())&&(q<4000)) {q++;}; //dajemy czas na budowanie odpowiedzi
    Serial.print((char)mySerial.read());     //...Drukujemy pozyskany numer

    Serial.print(" it is ");

    mySerial.print('R');          //odczytaj temperaturę z zewnętrznego urządzenia
    q=0;
    while ((!mySerial.available())&&(q<4000)) {q++;}; //dajemy czas na budowanie odpowiedzi
    while (mySerial.available())
    {
     Serial.print((char)mySerial.read()); //Drukujemy kolejne znaki
     delay(3);                            //odstęp czasowy pomiędzy znakami
    }
    Serial.println();                     //koniec odpowiedzi dla użytkownika
  }
 }
}
```

Software Serial – częste pytania

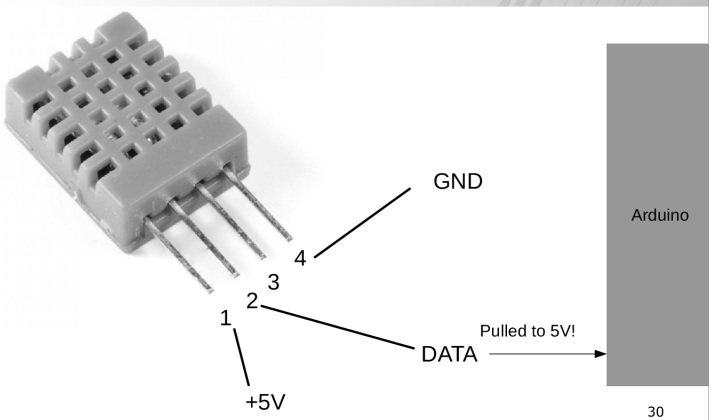
- Użytkowanie do celów debuggowania?
  - Nie bez modyfikacji – W Arduino Uno połączenie USB jest podłączone na stałe do pinów 0 i 1.
- Łączenie dwóch płytek: Jak?
  - Połączyć masy
  - Rx → Tx
  - Tx → Rx
- Maksymalna prędkość?
  - Zależna od „obciążenia" procesora, bezpiecznie jest do ok. 19200bps.

DHT-11

- Sensor temperatury i wilgotności działający na jednoprzewodowej magistrali.
- Zasilanie: 5V
- Podłączenie: --->
- Komunikacja dwustronna
- 0..50 *C, max 80RH
- Uwaga na pin 3/4!



DHT-11



Biblioteka DHT

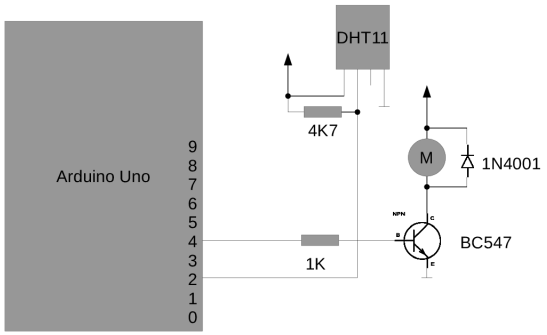
```
#include <dht11.h>

dht11 DHT;

#define DHT11_PIN 2
#define F_PIN 4
#define TEMP_MAX 29

void setup()
{
  Serial.begin(9600);
  Serial.println("Termostat ");
  Serial.println();
  pinMode(F_PIN, OUTPUT);
  digitalWrite(F_PIN,LOW);
}
```

Odczytaj wskazania do DHT.humidity i .temperature

Pin DHT11

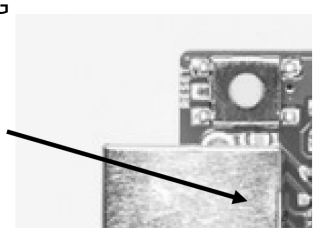
```
void loop()
{
  Serial.print("DHT11, \t");
  int chk = DHT.read(DHT11_PIN);
  Serial.print(chk);
  Serial.print(",\t");

  Serial.print(DHT.humidity, 1);
  Serial.print(",\t");
  double temp = DHT.temperature;
  Serial.println(temp, 1);
  if (temp>= TEMP_MAX)
    {
      digitalWrite(F_PIN,HIGH);
    }
  else
    {
      digitalWrite(F_PIN,LOW);
    }
  delay(2000);
}
```


Przykład zastosowania



ADC

- Za pomocą digitalRead możemy odczytać stan wysoki lub niski.
- Czasami pojawia się potrzeba zmierzenia napięcia – konkretnej wartości napięcia.
- Służy do teg oprzetwornik analogowo-cyfrowy podłączony do pinów ANALOG (A0..A5)
- Napięcie 0..5V jest dostępne jako wartość całkowita z przedziału 0...1023.



ADC - zastosowanie

- Przełączamy pin jako wejście:

```
pinMode(A0,INPUT);
```

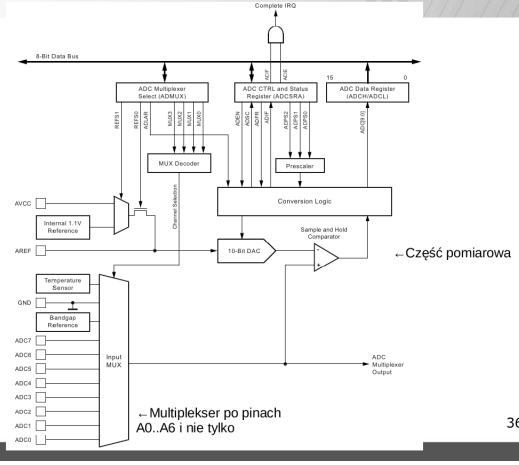
- Odczytujemy wartość i przeliczamy na napięcie (jeżeli tego potrzeba):

```
unsigned int value=analogRead(A0);
float volt = (value * 5.0) / 1024.0;
```

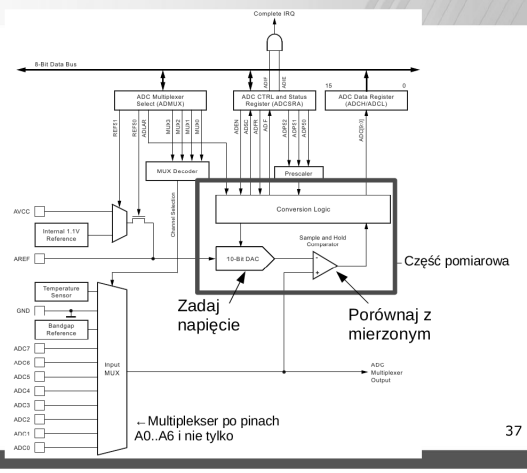

ADC i LM35 – temperatura analogowo

- LM35 zawiera w sobie gotowy dzielnik napięcia z termistorem. Charakterystyką układu jest (zgrubnie) 10mV/1°C. Stąd napięcie należy przemnożyć *100.
- Wcześniej zapewnić zasilanie układu.
- Nie podłączać zasilania odwrotnie! Materiał termistora spowoduje przepływ dużego prądu i możliwe uszkodzenie układu.

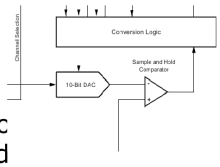
ADC w Arduino: Jak to działa?



ADC w Arduino: Jak to działa?



Jak ADC mierzy (i dlaczego tak długo?)



- Mierzona próbka przechowywana jest w kondensatorze sample-and-hold.
- DAC generuje kolejne napięcia.
- Komparator wychwytuje czy wygenerc napięcie jest większe, czy mniejsze od mierzonego.
- Końcowy wynik transmitowany jest do pamięci.
- Generowanie kolejnych napięć zajmuje (w zależności od procesora) **12-18 cykli**.

Korzystanie z ADC

- Co mierzyć? Względem czego? - ADMUX

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Shift results to the left.

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, internal V_{REF} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V voltage reference with external capacitor at AREF pin

MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8[1]
1001	(reserved)
1010	(reserved)
1011	(reserved)
1100	(reserved)
1101	(reserved)
1110	1.1V (V_{BG})
1111	0V (GND)

Korzystanie z ADC

- Czy już mierzyć? - ADCSRA

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Włączyć ADC

Start pomiaru

Pomiar na wyzwalacz

Koniec pomiaru wyzwala przerwanie

Preskaler zegara systemowego dla ADC

PWM – modulacja szerokości impulsu

- Często zachodzi potrzeba stopniowej regulacji jakiegoś urządzenia:
  - Jasność LED,
  - Obroty silnika,
  - Dostarczenie konkretnej wartości napięcia
- W tym celu stosuje się piny obsługujące **PWM** - Pulse Width Modulation.
- W Arduino Uno piny obsługujące PWM zaznaczone są na płytce symbolem tyldy (~).

PWM

- PWM wykorzystujemy ustawiając pin jako wyjście, a następnie określając współczynnik wypełnienia (0..255) funkcją **analogWrite**:

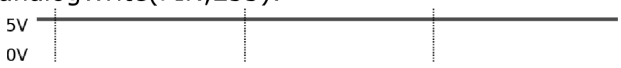
```
analogWrite(pin, wspolczynnik);
```

PWM

- analogWrite(PIN,20):

- analogWrite(PIN,127):

- analogWrite(PIN,255):



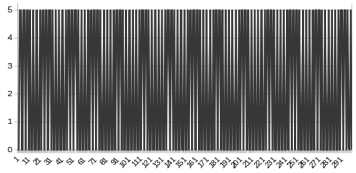
PWM

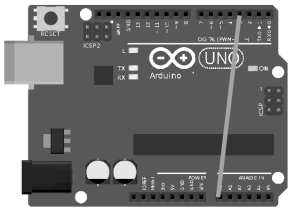
- PWM wykorzystujemy tam, gdzie możemy zapewnić **bezwładność** sterowanego układu:
  - Bezwładność wzroku przy szybko migającej LED,
  - Bezwładność wału silnika,
  - Bezwładność wskazówki mechanizmu wychyłowego,
- Tam, gdzie nie możemy tego zapewnić, stosujemy odpowiednio dobrany **układ całkujący**.
  Dokładne parametry dobieramy w zależności od planowanego poboru prądu, czasu reakcji oraz akceptowalnych szumów.

Pomiar bezpośredni

- Połączenie A0 z pinem 3 i próba z PWM:

```
void setup() {
  pinMode (A0,INPUT);
  pinMode(3,OUTPUT);
  Serial.begin(9600);
}

void loop() {
  analogWrite(3,127);
  Serial.println(analogRead(A0)*5.0/1023.0);
  delay(500);
}
```

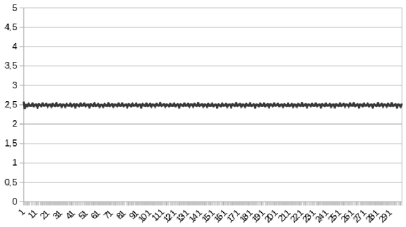


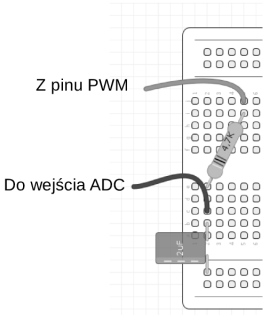


To bardzo nie wygląda jak 2.5V

PWM + RC

- Znacznie lepszy efekt:





Nieco zaszumione, ale znacznie lepiej

Przełączanie tranzystorem

Jak dobrać tranzystor?

- PNP/NPN,
- Si,
- Dopuszczalny prąd I_C (P_{tot}),
- Dopuszczalne napięcie V_{CEO}
- Parametry termiczne.

Większa moc/napięcie?

- ULN2803
  - 8x sterownik na tranzystorach w układzie Darlingtona.
  - Po podaniu 1 na wejście – na wyjściu 0.
  - Max 500mA/wyjście (moc !)
  - Max 50V
  - Pin 10 – do przełączanego napięcia.