Inteligentne Systemy Pomiarowe

Wykład 1

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

Tematyka zajęć

- Pomiary,
- Przetworniki analogowo-cyfrowe i ich użycie,
- Czujniki i przetworniki,
- Komunikacja między urządzeniami,
- Komunikacja między urządzeniem a serwerem.

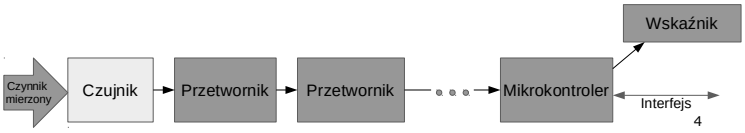
Pomiary

Do czego wykorzystać pomiary?

- Pobieranie informacji z zewnątrz
  - Systemy stricte pomiarowe,
  - Pomiar jako interfejs użytkownika,
- Utrzymanie zadanego stanu (sprzężenie zwrotne),
  - Systemy sterujące i kontrolne,
  - Serwomechanizmy,
- Zapewnienie działania urządzenia
  - Systemy zapewniające niezawodność,
  - Sterowanie nadmiarowością,
  - Wykrywanie awarii.

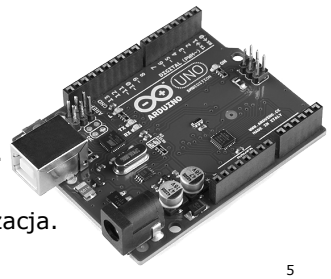
Istota pomiarów

- Wartość mierzona przetwarzana jest przez **czujnik** na wartość elektryczną. Owa wartość jest przetwarzana przez jeden lub więcej **przetworników**, których wyjściem jest wartość cyfrowa wprowadzana do jednostki centralnej (mikrokontrolera).
- Mikrokontroler może wyprowadzić wartość na **urządzenie wyjścia**, np. wyświetlacz, lub podejmować na jej podstawie decyzje, albo przesłać ją dalej.

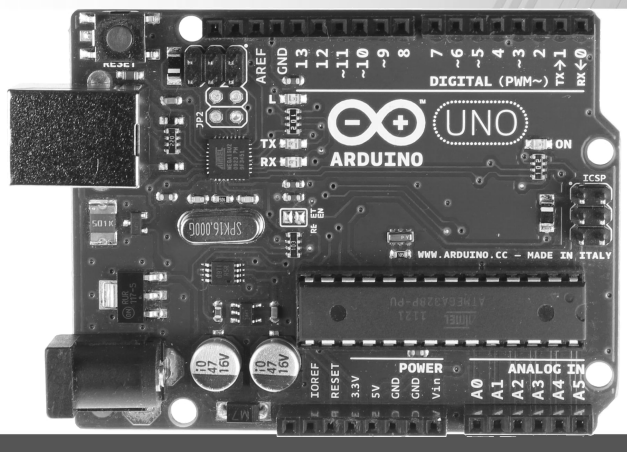


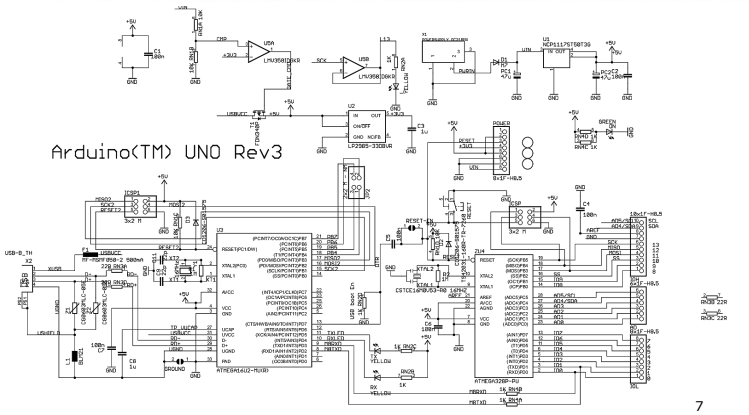
Powtórka z Arduino

- Mikrokontroler: AVR ATMega328 16MHz.
- 32kB pamięci Flash na program (31kB dostępne - bootloader)
- 2kB RAM
- 1kB EEPROM
- GPIO: 14 pinów (6 PWM)
- 6 wejść analogowych
- Interfejs z komputerem: USB-RS232.
- Programowanie przez USB.
- Zasilanie: 5V, własna stabilizacja.

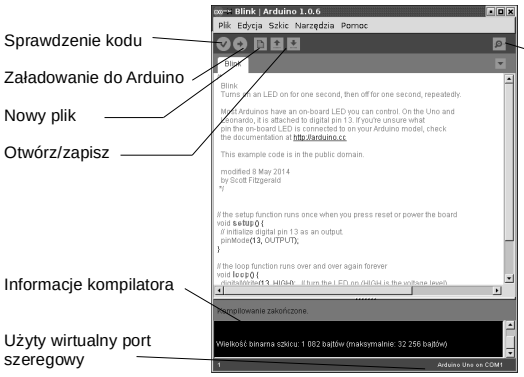


Arduino Uno



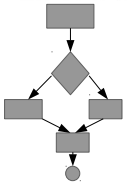


Arduino IDE – programowanie.

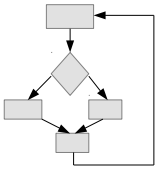


Programowanie w pętli

- Każdy program działający w systemie operacyjnym zaczyna się i kończy. Po zakończeniu programu następuje powrót do systemu operacyjnego.

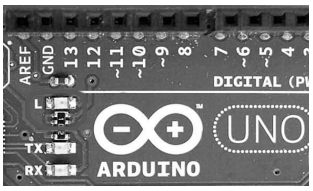


- Mikrokontroler systemu operacyjnego **nie posiada.** Program wykonuje się w nieskończonej pętli.
- Można tworzyć „pod-pętle” wprowadzając różne tryby pracy.



Budowa programu

```
#define ledPin  13


void setup()
{
  pinMode(ledPin, OUTPUT);
}


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



Budowa programu

```
#define ledPin  13


void setup()
{
  pinMode(ledPin, OUTPUT);
}


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

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

}
```

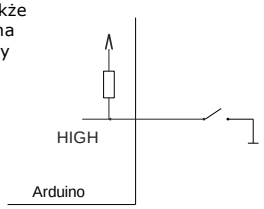

Zmienne i funkcje

```
#define ledPin 13

void setup() {
  pinMode(ledPin, OUTPUT);
}

void displayInt(int number)
{
  for (int i=0;i<number;i++)
  {
      digitalWrite(ledPin, HIGH);
      delay(250);
      digitalWrite(ledPin, LOW);
      delay(250);
  }
  delay(500);
}
```

```
unsigned int k=0;
void loop()
{
    k++;
    displayInt(k);
}
```

Wejścia – idea rezystora podciągającego

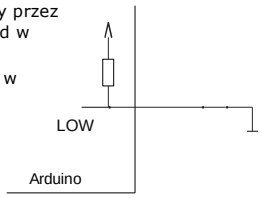
- Stan logiczny „HIGH” to ok. 2..5V. Stan „LOW” to 0..0.8V.
- Prąd pobierany przez wejście przy sprawdzaniu stanu (pomiarze) jest minimalny,
- Wejście niepodłączone - „wiszące w powietrzu” (także podłączone do otwartego łącznika) - jest podatne na zakłócenia. Stany zmieniają się w nieprzewidywalny sposób. Są to tzw. stany nieustalone,
- Niezbędne jest użycie niewielkiego prądu, który zapewniłby wysoki stan logiczny gdy wejście jest niepodłączone, Prąd ten zapewniany jest przez rezystor podciągający (pull-up resistor) o wartości kilku kΩ, podłączony do zasilania.



Arduino ma wbudowane rezystory podciągające i nie ma potrzeby używania zewnętrznych!

Wejścia – idea rezystora podciągającego

- Gdy dołączymy do wejścia stan niski (np. masę), prąd popłynie przez rezystor, a spadek napięcia na nim będzie wystarczający by na wejściu mikrokontrolera pojawił się stan niski.
- Ponieważ rezystor ma sporą oporność, prąd płynący przez niego będzie zaś niski na tyle, by nie poczynić szkód w układzie.
- Rezystory podciągające są powszechnie stosowane w układach logicznych.



Arduino ma wbudowane rezystory podciągające i nie ma potrzeby używania zewnętrznych!

Zmienne i funkcje, wejścia

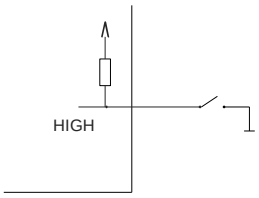
```
#define ledPin 13
#define resetPin 11

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(resetPin, INPUT);
  digitalWrite(resetPin,HIGH);
}

void displayInt(int number)
{
  for (int i=0;i<number;i++)
  {
      digitalWrite(ledPin, HIGH);
      delay(250);
      digitalWrite(ledPin, LOW);
      delay(250);
  }
  delay(500);
}
```

```
unsigned int k = 0;

void loop()
{
  k++;
  displayInt(k);

  if (!digitalRead(resetPin))
  {
    delay(100);
    k=0;
  }
}
```


Zmienne i funkcje, wejścia

```
#define ledPin 13
#define resetPin 11

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(resetPin, INPUT);
  digitalWrite(resetPin,HIGH);
}

void displayInt(int number)
{
  for (int i=0;i<number;i++)
  {
      digitalWrite(ledPin, HIGH);
      delay(250);
      digitalWrite(ledPin, LOW);
      delay(250);
  }
  delay(500);
}
```

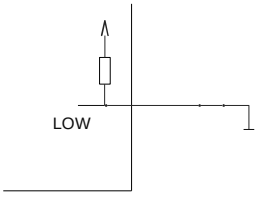
```
unsigned int k = 0;

void loop()
{
  k++;
  displayInt(k);

  if (!digitalRead(resetPin))
  {
    delay(100);
    k=0;
  }
}
```



Zmienne i funkcje, wejścia

```
#define ledPin 13
#define resetPin 11

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(resetPin, INPUT);
  digitalWrite(resetPin,HIGH);
}

void displayInt(int number)
{
  for (int i=0;i<number;i++)
  {
      digitalWrite(ledPin, HIGH);
      delay(250);
      digitalWrite(ledPin, LOW);
      delay(250);
  }
  delay(500);
}
```

```
unsigned int k = 0;

void loop()
{
  k++;
  displayInt(k);

  if (!digitalRead(resetPin))
  {
    delay(100);
    k=0;
  }
}
```



INPUT + digitalWrite(HIGH) = **INPUT_PULLUP**

Przerwania

```
#define ledPin 13
#define resetPin 2
volatile unsigned int k=0;

void reset()
{
  k=0;
}

void setup()
{
  pinMode(ledPin, OUTPUT);
  attachInterrupt(0,reset,LOW);
  pinMode(resetPin, INPUT);
  digitalWrite(resetPin,HIGH);
}
```

```
void displayInt(int number)
{
  for (int i=0;i<number;i++)
  {
      digitalWrite(ledPin, HIGH);
      delay(250);
      digitalWrite(ledPin, LOW);
      delay(250);
  }
  delay(500);
}


void loop()
{
    k++;
    displayInt(k);
}
```

Przerwania

```
#define ledPin 13
#define resetPin 2
volatile unsigned int k=0;

void reset()
{
  k=0;
}

void setup()
{
  pinMode(ledPin, OUTPUT);
  attachInterrupt(0,reset,LOW);
  pinMode(resetPin, INPUT);
  digitalWrite(resetPin,HIGH);
}
```

```
void displayInt(int number)
{
  for (int i=0;i<number;i++)
  {
      digitalWrite(ledPin, HIGH);
      delay(250);
      digitalWrite(ledPin, LOW);
      delay(250);
  }
  delay(500);
}


void loop()
{
    k++;
    displayInt(k);
}
```

Przerwanie	Pin
0	2
1	3

~~*delay();*~~
~~*millis(); = const*~~

Serial.Read()...!

Port szeregowy – komunikacja z komputerem

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


void loop()
{
    Serial.print("TO JEST TEST\n");
    unsigned int k = 111;
    Serial.println(k);
    Serial.println(k,DEC);
    Serial.println(k,HEX);
    Serial.println(k,BIN);
    Serial.Write(k);
    Serial.println(”/nKoniec.”);
}
```


Port szeregowy – komunikacja z komputerem

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


void loop()
{
    Serial.print("TO JEST TEST\n");
    unsigned int k = 111;
    Serial.println(k);
    Serial.println(k,DEC);
    Serial.println(k,HEX);
    Serial.println(k,BIN);
    Serial.Write(k);
    Serial.println(”/nKoniec.”);
}
```

```
TO JEST TEST
111
111
6F
11001111
0
Koniec.
```


Port szeregowy – komunikacja z komputerem

Odbieranie danych:

```
void loop()
{
    char bajt=0;
    if (Serial.available() > 0)
    {
        bajt = Serial.read();
        Serial.print("Bajt: ");
        Serial.println(bajt, DEC);
    }
}
```

Tryb automatyczny:

```
void serialEvent()
{
    ...
}
```


Port szeregowy – komunikacja z komputerem

Odbieranie danych:

```
void loop()
{
    char bajt=0;
    if (Serial.available() > 0)
    {
        bajt = Serial.read();
        Serial.print("Bajt: ");
        Serial.println(bajt, DEC);
    }
}
```

Tryb automatyczny:

```
void serialEvent()
{
    ...
}
```

To NIE jest przerwanie!!!

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

}
```


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));
  }

//tutaj możemy rozbudowywać!

}
```

Wersja z millis – kod otwarty na rozbudowę

EEPROM

- Pamięć nieulotna, ale jedną komórkę można zapisać tylko ok. 100 000 razy!
- Mamy 1024 bajty (0..1023).
- Polecenia biblioteki EEPROM:
  - `EEPROM.write(adres,wartosc);`
  - `EEPROM.read(adres);` - zwraca wartość byte
  - `EEPROM.put(adres,wartosc);` - zapis struktury, używa `sizeof(wartosc)` bajtów,
  - `EEPROM.get(adres,dane)` – odczytuje `sizeof(dane)` bajtów do zmiennej dane.

ADC

- Przetwornik analogowo-cyfrowy
- Dostępny jest w mikrokontrolerze (AVR, PIC) lub w postaci oddzielnego modułu.
- Umożliwia pomiar napięcia z reguły **względem masy**.
- Pomiary dokonywane są za pomocą porównania do **napięcia odniesienia**.

Wykorzystanie ADC

1. Ustawić pin jako wejście:

- `pinMode(A0,INPUT);`

2. W programie pobrać wartość:

- `int ADCValue = analogRead(A0);`
  (wartość 0..1023)

3. W miarę potrzeby przeliczyć na V:

- `float w = ADCValue*5.0 / 1023.0;`

Wykorzystanie ADC

- analogReference() - Ustawienie napięcia odniesienia (i maksimum pomiaru).
  - DEFAULT – Domyślne 5V zasilania
  - INTERNAL – Wewnętrzne źródło 1.1V (2.56V w ATMega8)
  - EXTERNAL – napięcieprzyłożone do pinu AREF (max. Vcc!)

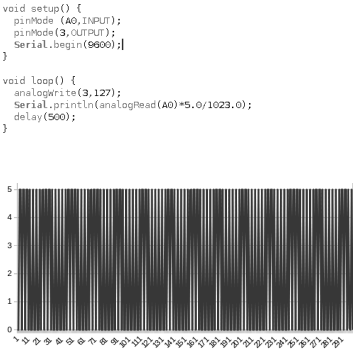
Uzyskiwanie napięcia

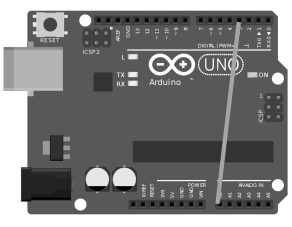
- Po co?
  - Uzyskanie napięcia wzorcowego dla pomiarów,
  - Zasilanie urządzeń różnym napięciem,
  - Spełnianie wymagań specyfikacji,
  - Konieczność sterowania np. silników – nie tylko szybkość, ale i moment.
  - Urządzenia bez bezwładności spowodowanej nieciągłymi oscylacjami PWM.

Użycie PWM?

- 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);
}
```

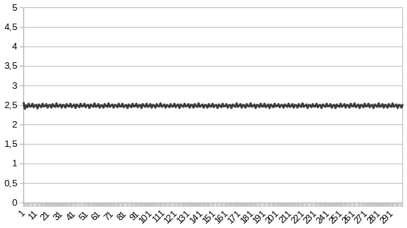


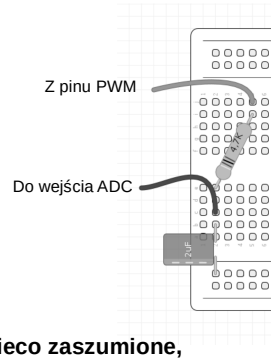


To bardzo nie wygląda jak 2.5V

PWM + RC

- Znacznie lepszy efekt:





Nieco zaszumione, ale znacznie lepiej