Systemy Wbudowane

Arduino, AVR

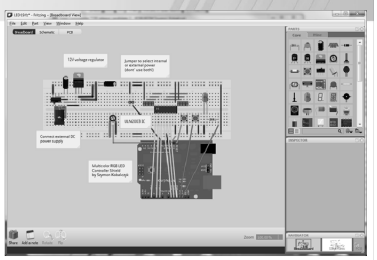
(wersja 2026)

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

Arduino

- Mikrokontroler
- Platforma Arduino
  - Mikrokontroler AVR
  - Arduino Uno
- Arduino IDE:
  - Środowisko
  - Preprocesor kodu
  - Terminal
  - Uruchamianie

Oprogramowanie



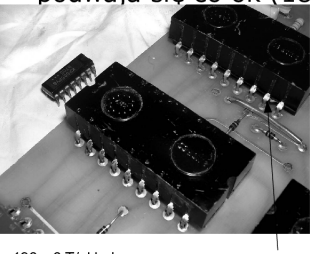
- Arduino IDE:
  - http://www.arduino.cc
- Fritzing:
  - http://fritzing.org
- KiCAD:
  - http://www.kicad-pcb.org
- WinAVR:
  - http://winavr.sourceforge.net/
- UnoArduSim:
  - https://www.sites.google.com/site/unoardusim/
- DIY Layout Creator:
  - http://diy-fever.com/software/diylc/

Mikrokontroler

- System mikroprocesorowy w postaci jednego układu scalonego.
- Zintegrowane:
  - CPU
  - RAM
  - Pamięć programu
  - Urządzenia I/O
  - Dodatkowe urządzenia
- Użycie jednego układu: oszczędność miejsca, energii, łatwa rozbudowa i programowanie systemu.

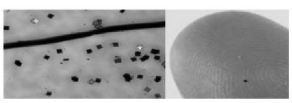
Mikroelektronika

- Prawo Moore'a: Liczba tranzystorów w układzie podwaja się co ok (18..24) miesięcy.

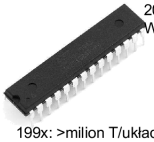


196x: 6 T/układ

198x: 10 000 T/układ
Miniaturyzacja



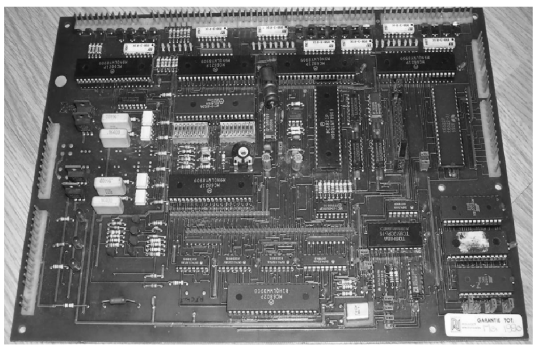
200x: >100 milionów T/układ
Większa miniaturyzacja



199x: >milion T/układ

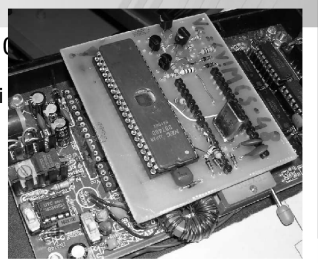
Rys historyczny

- 1970... Uniwersalne bloki sterujące. Jeden moduł, cechy późniejszych mikrokontrolerów.

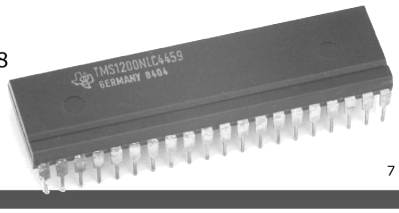


Rys historyczny

- 1971 – Texas Instruments TMS1
  - Pierwszy mikrokontroler
  - Wewnętrzne źródło częstotliwości
  - Programowanie: Mask-ROM
  - Bardzo wysokie ceny

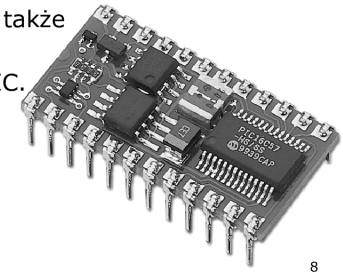


- 1976 – Intel 8048 (MCS-48):
  - Programowanie jak EPROM lub Mask-ROM.
  - 1kB ROM
  - 128B RAM
  - Początek serii MCS-48 i późniejszych 51.
  - Programowanie:
    - Assembler



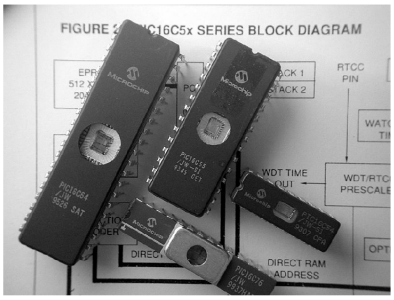
Rys historyczny

- 198x - BASIC Stamp – miniaturyzacja, niska cena, łatwe zastosowania hobbystyczne.
  - Układ hybrydowy,
  - Łatwe programowanie, także in-system,
  - Programowanie w BASIC.
  - Wciąż produkowane.



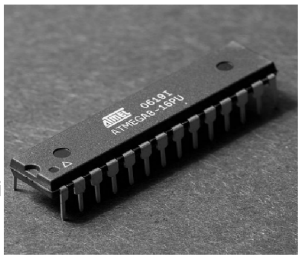
Rys historyczny

- 1975-85 – PIC:
  - Programowanie jak EPROM,
  - 1993 – Pamięć EEPROM.
  - Różnorodność modeli.
  - Programowanie:
    - Assembler
    - C
  - Komercyjne narzędzia.
  - Wciąż rozwijane.



Rys historyczny

- 1995-97 – Atmel AVR:
  - Duża pamięć programu (4-512kB)
  - Wiele urządzeń wewnętrznych
  - Różnorodność modeli: od ATTiny do AVR32
  - Pamięć Flash dla programu
  - Pamięć Flash dla użytkownika
  - Niska cena
  - Łatwe programowanie:
    - Assembler
    - Basic (BASCOM)
    - C
  - Obecność platform np. Arduino.
  - Otwarte narzędzia deweloperski
  - Wciąż rozwijane.



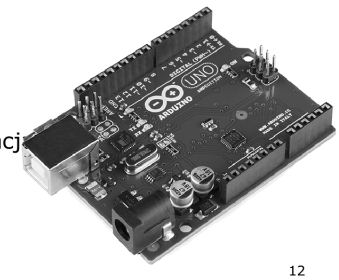
Arduino

- Platforma programistyczna dla systemów wbudowanych.
- Open Hardware (z wyjątkami).
- Pojedynczy moduł.
- Mikrokontroler AVR.
- Wbudowany interfejs mikrokontroler-komputer.
- Programowanie: Arduino C.
- Środowisko: Arduino IDE.

ARDUINO™

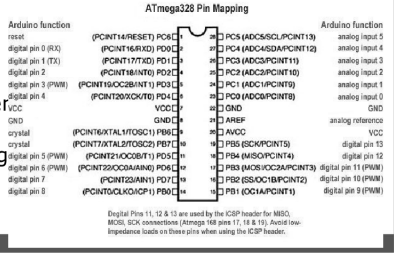
Arduino Uno

- 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



AVR

- 8-bitowy, jednoukładowy mikrokontroler RISC.
- Pamięć programu: Flash,
- Pamięć operacyjna: Statyczny RAM,
- Dodatkowa pamięć Flash dla programów użytkownika,
- Wyprowadzenia wielofunkcyjne,
- Wbudowane interfejsy i przetworniki,
- BOR, WDT,
- Możliwość pracy z wewnętrznym oscylatorem,
- Programowanie ISP (In-System Programming)



AVR: Co otrzymujemy?

- 2 timery 8-bitowe
- 1 timer 16-bitowy
- 2 liczniki
- 6 kanałów PWM
- 6 kanałów ADC, 10-bitowe
- 2 przerwania z GPIO

Arduino C

- Eliminuje konieczność ustawiania wbudowanych urządzeń przez konfigurację rejestrów sterujących.
- Ułatwia budowę programu.
- Znacznie przyspiesza testowanie i prototypowanie.
- Zabezpiecza przed problemami z konfiguracją wstępną (Fuse-bity).
- Dostarcza użytecznych bibliotek funkcji.
- Kod podobny do C++, konwertowany do C.

- Niższa niż w przypadku czystego C wydajność i większy rozmiar kodu.
- Toleruje techniki marnujące pamięć operacyjną.

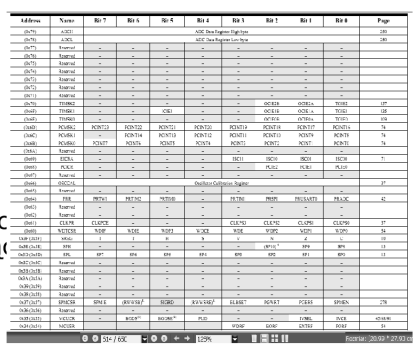
Informacje

- Noty katalogowe i instrukcje:
  - http://www.atmel.com
  - http://www.atmel.com/devices/atmega328.aspx
- Arduino:
  - http://www.arduino.cc/
  - **http://arduino.cc/en/Reference/**
  - http://ep.com.pl - „Kurs Arduino” (PL)

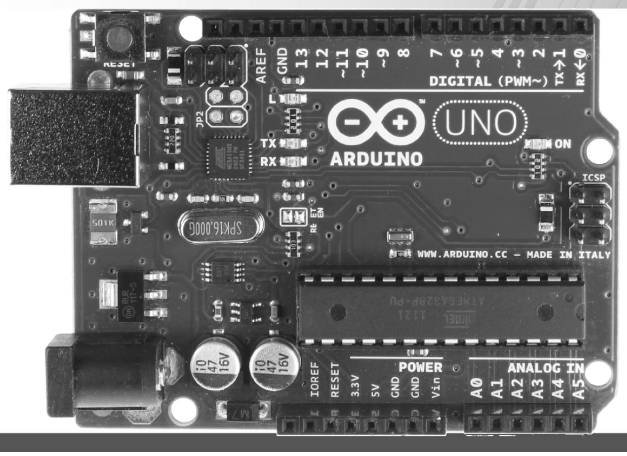
- Podręczniki:
  - Monk S. - „Arduino dla początkujących – Podstawy i szkice” - Helion 2014.
  - Evans B. - „Beginning Arduino programming” - Apress 2011.

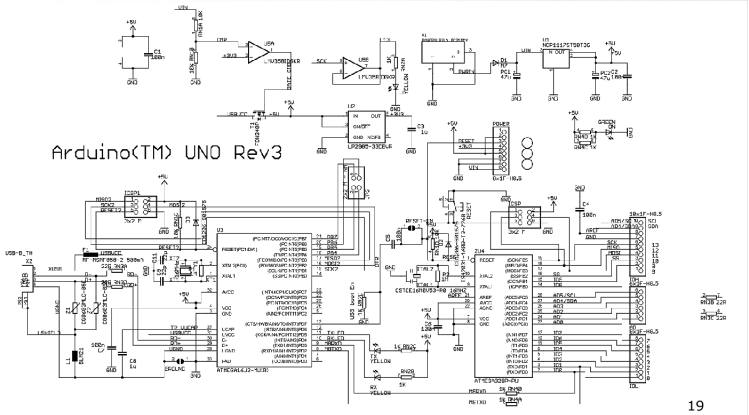
Informacje – „Manual” Co jest niezbędne?

- Parametry układu: Czy układ będzie spełniał zadanie w projekcie?
- Warunki pracy układu:
  - Zasilanie.
  - Moc wyjściowa.
  - Warunki środowiskowe.
  - Wydajność.
- Czy parametry używanyc urządzeń są wystarczają
- Programowanie: Mapa rejestrów i praca z urządzeniami.

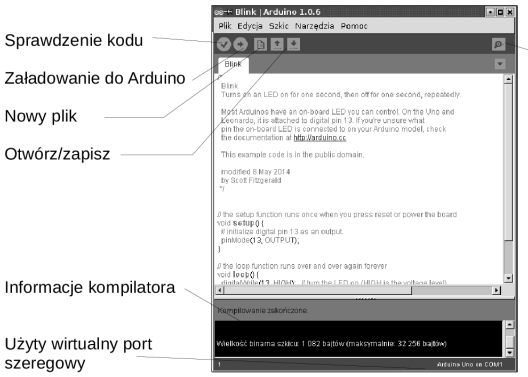


Arduino Uno





Arduino IDE – programowanie.



Sprawdzenie kodu

Załadowanie do Arduino

Nowy plik

Otwórz/zapisz

Informacje kompilatora

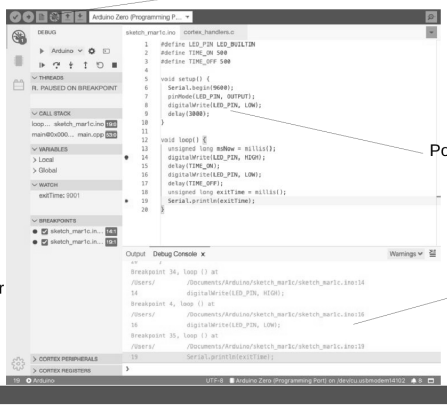
Użyty wirtualny port szeregowy

Monitor portu „terminal”

Nie stosować danych binarnych!!

Arduino IDE v. 2

Debugger na moduły typu „IoT”



Podpowiadanie składni

Panel zależny od aktualnego stanu środowiska (Programowanie, debuggowanie, projekt, menedżer sprzętu itp.)

Konsola lub plotter

IDE v. 2 - uwagi

- Debuggowanie działa wyłącznie na płytkach typu „IoT”.
- Środowisko wymaga dostępu do Internetu - nie nadaje się do zastosowań, których nie zamierzamy opublikować niekoniecznie za nasza wiedzą.
- Podczas uruchamiania nawiązywane są bez zgody użytkownika połączenia do zagranicznych IP.
- ...podczas których pobierane są dodatkowe komponenty programu (robak? wirus? złamanie GPL?).
- Podsumowując: Środowisko nie nadaje się jeszcze do zastosowań produkcyjnych, a zupełnie nie nadaje się do działań np. w firmie do komercyjnych projektów.

PlatformIO

- Alternatywne środowisko dla Arduino.
- Filozofia Uniksa - maksymalna modularność.
- Wymagane IDE: Niemal dowolne, w tym Eclipse, NetBeans, CodeBlocks, Qt Creator, VSCode, vim/EMACS.
- Kompilator: Pobiera odpowiednie paczki gcc.
- Biblioteki: dodawane przez użytkownika.
- Terminal: Używamy wbudowanego programu **pio** lub zewnętrznego np. **minicom**.
  - Ważne, by terminal nie działał jednocześnie z kompilacją/budowaniem/wgrywaniem.
  - Krótka podpiwiedź: **`pio device monitor`**

PlatformIO: Początek projektu (Wybrane IDE: CodeBlocks)

- Po zainstlaowaniu PlatformIO wchodzimy w terminalu do katalogu, w którym chcemy mieć projekt, po czym tworzymy go:

```
pio project init --ide codeblocks --board uno
```

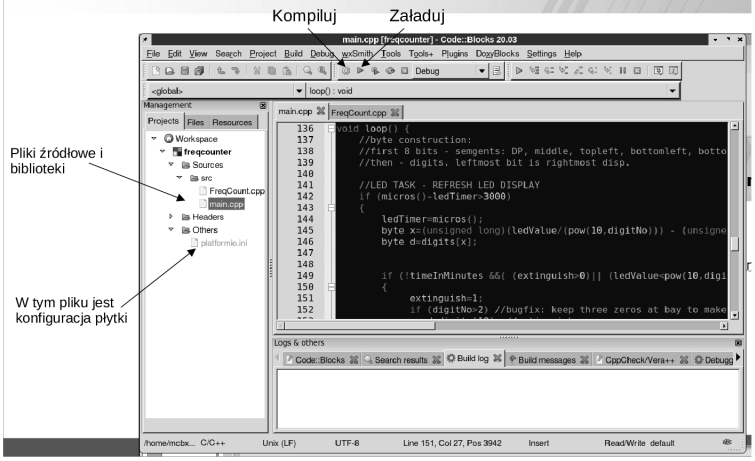
- UWAGA: Jakiekolwiek brakujące paczki zostaną pobrane z sieci! Należy mieć stabilne połączenie.
- Otwieramy CodeBlocks, otwieramy projekt, jaki wytworzył się w katalogu, dodajemy plik np. main.cpp, w którym piszemy już typowy kod Arduino.

PlatformIO: Kompilacja i wgrywanie

- Jest ważne, by plik zawierał na początku:
  `#include <Arduino.h>`
- To doda nam wszystkie funkcje biblioteki Wiring (Arduino).
- Następnie już typowo:

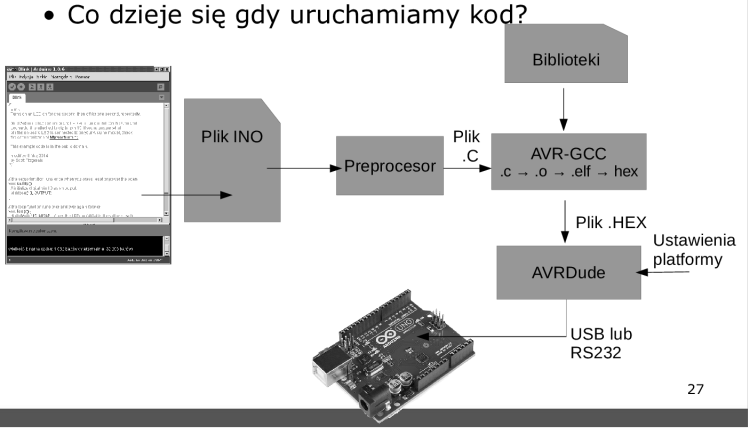
```
void setup() {
 //...
}

void loop() {
  //...
}
```


PlatformIO: Kompilacja i wgrywanie



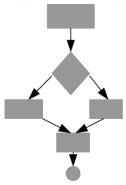
Arduino C

- Co dzieje się gdy uruchamiamy kod?

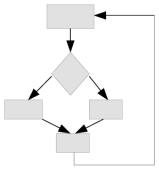


Programowanie w pętli

- 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.



Programowanie w pętli - przykład

- Zegar z budzikiem:
  - Normalnie działanie: Wyświetlanie aktualnej godziny.
  - Po naciśnięciu przycisku ustawienia czasu:
    - Przyciskami H+/M+ można ustawić bieżącą godzinę.
    - Zatwierdzamy dowolnym innym przyciskiem.
  - Po naciśnięciu przycisku ustawienia alarmu
    - Wyświetlenie godziny alarmu
    - Przyciskami H+/M+ można ustawić godzinę alarmu.
    - Zatwierdzamy dowolnym innym przyciskiem.

Programowanie w pętli - przykład

- Zegar z budzikiem:
  - Normalnie działanie: Wyświetlanie aktualnej godziny.
  - Po naciśnięciu przycisku ustawienia czasu:
    - Przyciskami H+/M+ można ustawić bieżącą godzinę.
    - Zatwierdzamy dowolnym innym przyciskiem.
  - Po naciśnięciu przycisku ustawienia alarmu
    - Wyświetlenie godziny alarmu
    - Przyciskami H+/M+ można ustawić godzinę alarmu.
    - Zatwierdzamy dowolnym innym przyciskiem.

Główna pętla:
- Odświeżanie aktualnej godziny,
- Czy wciśnięty przycisk ust. czasu?
  - Odśwież aktualną godzinę
  - Jeżeli wciśnięty H+ - zwiększ zmienną godzin,
  - Jeżeli wciśnięty M+ zwiększ zmienną minut,
  - Jeżeli wciśnięty inny przycisk – opuść pętlę.
- Czy wciśnięty przycisk ust. alarmu?
  - Odśwież godzinę alarmu,
  - Jeżeli wciśnięty H+ - zwiększ godzinę alarmu,
  - Jeżeli wciśnięty M+ zwiększ minutę alarmu,
  - Jeżeli wciśnięty inny przycisk – opuść pętlę,

Podstawowe techniki

- Dyrektywa preprocesora #define:

  #define nazwa wartość

- Podczas kompilacji wystąpienia nazwy zostaną zastąpione wartością. Np.

  #define ledPin 13

- Zamieni występowanie ledPin na 13.
- W przypadku zmiany konstrukcji – zmiana jednej linii kodu.

Podstawowe techniki

- volatile – nie buforuje zmiennej w rejestrach:

  volatile int stan;

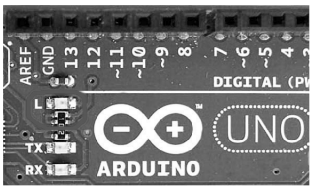
- Gdy zmienna ta zostanie zmieniona poza programem (np. w trakcje procedury przerwania) zostanie użyta jej aktualna wartość.
- W nowszych wersjach kompilator automatycznie stara się ustawić buforowanie zmiennych.

Podstawowe typy danych

- boolean – wartość logiczna (prawda/fałsz), zajmuje **1 bajt** danych.
- char, unsigned char = **byte** (0..255).
- int, unsigned int = word (0..2^16-1).
- Zmiennoprzecinkowe: float (4B), double (4B).
- string – jako tablica char'ów lub typ z biblioteki.

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();
   }

}
```


LED

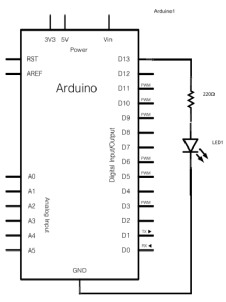
- Wymagany rezystor obniżający prąd.
- Vcc=5V
- Typowy LED 3mm:

  I_F=20mA, V_F=2V

$$R = \frac{Vcc - V_F}{I_F}$$

(5-2) / 0.02 = 150 Ω

A w praktyce – 120...470Ω



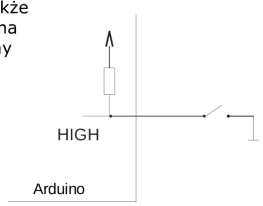
Zmienne i funkcje

```
#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

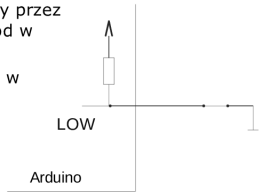
- Stan logiczny „HIGH” to ok. 2..5V. Stan „LOW” to 0..0.8V.
- Prąd pobierany przez wejście przy sprawdzaniu stanu (pomiarze) jest minimalny,
- Wejście niepodłączone - „wiszące w powietrzu” (także podłączone do otwartego łącznika) - jest podatne na zakłócenia. Stany zmieniają się w nieprzewidywalny sposób. 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

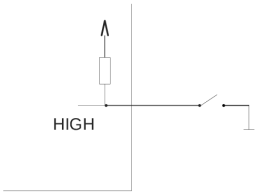
```
#define ledPin 13
#define resetPin 11

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);
}
```

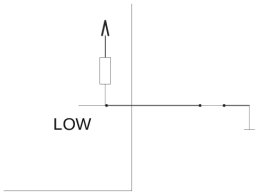
```
unsigned int k = 0;

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
o
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:

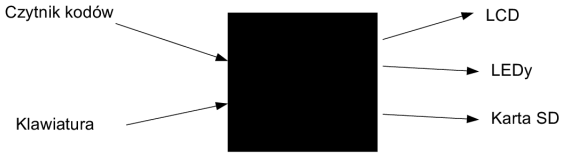
```
void serialEvent()
{
    ...
}
```

To NIE jest przerwanie!!!

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

}
```

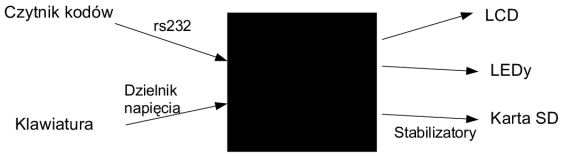
Konstrukcja urządzenia (1)

1. Specyfikacja problemu
   - np. Zbieranie i przechowywanie informacji o dostarczonych produktach
2. Jakie urządzenia wejścia i wyjścia są potrzebne?
   - np. Wejście: Czytnik kodów kreskowych, klawiatura,
   - Wyjście: Karta SD, wyświetlacz, beeper, LEDy



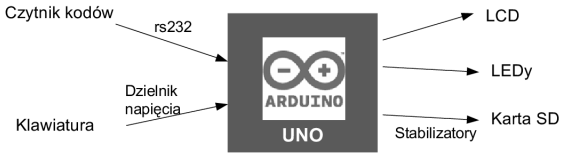
Konstrukcja urządzenia (2)

3. Czy któreś z tych urządzeń wymaga sterowników?
   - Odpowiednio dobrany sterownik oszczędza porty I/O
   - Zasilanie urządzeń – czy potrzebujemy dodatkowych źródeł zasilania?



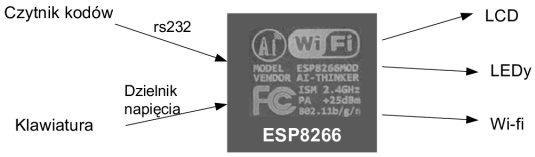
Konstrukcja urządzenia (3)

3. Wybór platformy systemu, ocena wydajności, możliwości rozbudowy i dostosowywania.



Konstrukcja urządzenia (3)

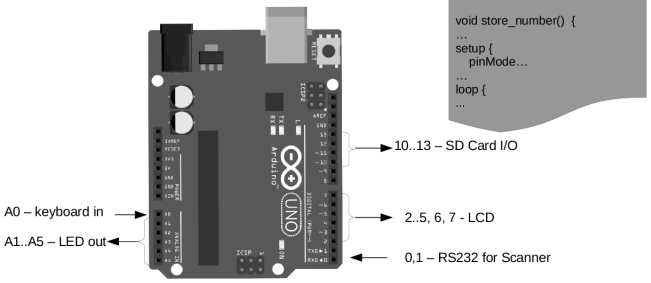
3. Wybór platformy systemu, ocena wydajności, możliwości rozbudowy i dostosowywania.



Konstrukcja urządzenia (4)

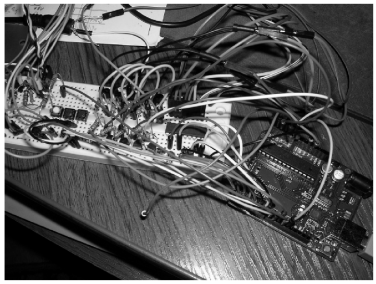
4. Szkielet programu: Definicje, Ustalenie ról wejść/wyjść, założenia programu, podstawowe procedury (+ „zaślepki" funkcji)

```
#define KEYBOARD A0
#define LED1 A1
…
void store_number() {
…
setup {
   pinMode…
…
loop {
…
```



Konstrukcja urządzenia (5)

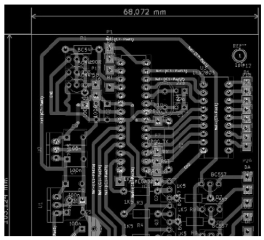
5. Przedprototyp (płytka stykowa), testowanie, dopełnianie i udoskonalanie programu korzystając z połączenia USB do Arduino. Rysowanie i poprawki schematów częściowych (sterowników poszczególnych urządzeń).

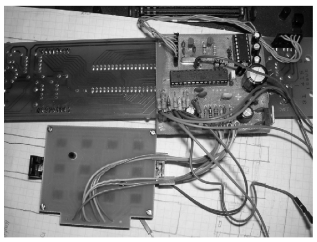


Konstrukcja urządzenia (6)

7. Końcowe rozwiązanie kwestii zasilania gotowego urządzenia

8. Projektowanie końcowego schematu. Zaprojektowanie i wykonanie płytki drukowanej łączącej mikrokontroler i niezbędne interfejsy. Końcowe testy i poprawki, umieszczenie układu w obudowie.





Dziękuję za uwagę

- Następny wykład:
  - Techniki programowania,
  - Biblioteki,
  - Układ pomiarowy DHT11 i jego podłączenie,
  - Biblioteka obsługi DHT11