Podstawy Systemów Wbudowanych

Wykład 4: Biblioteki do obsługi urządzeń peryferyjnych

Angelika Tefelska Dariusz Tefelski

Zakład Fizyki Jądrowej, Wydział Fizyki PW

22 marca 2017

O czym będzie wykład...

Biblioteki podstawowe

- **EEPROM** - zapis i odczyt z pamięci EEPROM.
- **Ethernet / Ethernet 2** - biblioteka do komunikacji z internetem wykorzystywana przy Arduino Ethernet Shield, Arduino Ethernet Shield 2 i Arduino Leonardo ETH.
- **Firmata** - biblioteka do komunikacji z aplikacjami na komputerze przy użyciu standardowego protokołu.
- **GSM** - biblioteka do komunikacji z GSM/GRPS dla GSM shield.
- **LiquidCrystal** - biblioteka do obsługi wyświetlaczy LCD.
- **SD** - biblioteka do zapisu i odczytu z karty pamięci SD.
- **Servo** - biblioteka do sterowania serwomechanizmami.
- **SPI** - biblioteka do komunikacji przez magistralę SPI.
- **SoftwareSerial** - biblioteka, która umożliwia komunikację z dowolnego pinu cyfrowego jak przez port szeregowy.
- **Stepper** - biblioteka do sterowania silnikami krokowymi.
- **TFT** - biblioteka do obsługi wyświetlaczy Arduino TFT.
- **WiFi** - biblioteka do komunikacji przez WiFi dla Arduino WiFi shield.
- **Wire** - biblioteka do komunikacji przez magistralę I2C.

Legenda: biblioteki omówione na poprzednich wykładach, biblioteki, które poznamy dzisiaj, biblioteki które będą omówione na kolejnych wykładach.

Biblioteki dodatkowe

- Oprócz bibliotek podstawowych, dostępne są biblioteki dedykowane do konkretnych modeli Arduino np. biblioteka **Keyboard czy Mouse** (umożliwiające podłączenie Arduino Leonardo jako klawiatury lub myszki do komputera).
- Dodatkowo istnieje możliwość skorzystania z wielu gotowych bibliotek opracowanych przez firmy, które produkują czujniki (np. **OneWire** czy **DallasTemperature** firmy Dallas Semiconductor) oraz osoby prywatne.
- Strony do wyszukiwania gotowych bibliotek:
  - http://playground.arduino.cc/Main/LibraryList - oficjalne biblioteki arduino zarówno podstawowe jak i dodatkowe.
  - http://www.arduinolibraries.info/ - biblioteki zarówno oficjalne, udostępnione przez firmy jak i stworzone przez prywatnych użytkowników.
- Po ściągnięciu gotowej biblioteki trzeba ją wypakować do katalogu *libraries* oraz ponownie uruchomić środowisko IDE.

Biblioteki do czujników

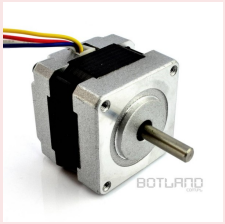
- Obecnie liczba gotowych bibliotek do obsługi czujników wynosi ok 200 i wciąż rośnie. Znaczna część to biblioteki firm: Adafruit, Blue Robotics, FaBo, Smart Everything oraz Spark Fun.
- Generalnie jeśli znajdziemy gotową bibliotekę do czujnika, który chcemy obsłużyć i wypakujemy ją do katalogu *libraries* to w środku biblioteki znajdziemy zazwyczaj katalog *example*. W nich jest przykład jak użyć daną bibliotekę.
- Jako przykład posłuży nam biblioteka **Ultrasonic**, za pomocą której możemy sterować ultradźwiękowym czujnikiem do pomiaru odległości.

Biblioteka Ultrasonic - przykład z katalogu example

```
#include "Ultrasonic.h"
#include <LiquidCrystal.h>
LiquidCrystal lcd(11, 10, 9, 4, 5, 6, 7);
Ultrasonic ultrasonic(12,13); //trigger, echo

void setup()
{
    lcd.begin(16, 2);
    lcd.print("testing...");
}

void loop()
{
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(ultrasonic.Ranging(CM));
    lcd.print("cm");

    delay(100);
}
```

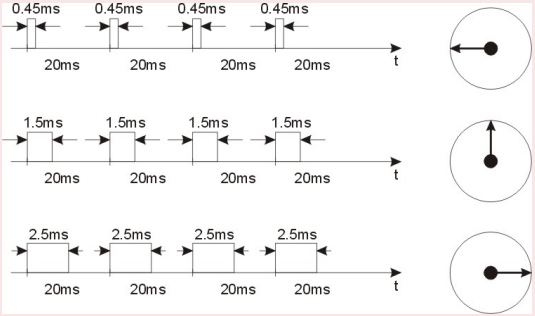
Aktuatory

- Aktuator = urządzenie wykonawcze
- Przykłady aktuatorów: serwomechanizmy, silniki krokowe, silniki elektryczne.



Zasada działania serwomechanizmu

- Serwomechanizmy są powszechnie stosowane w modelarstwie np. zdalnie sterowane samoloty, w których serwomechanizm jest odpowiedzialny za ruch klap.
- Serwomechanizm to silnik z przekładnią obracającą się o zadany kąt, zazwyczaj z przedziału $(0, 180)^o$. Z serwomechanizmu wyprowadzone są 3 przewody. Środkowy (zazwyczaj czerwony) powinien być podłączony do zasilania (5V). Skrajny o kolorze czarnym to GND a ostatni przewód (zazwyczaj w jasnym kolorze: biały/żółty) to sterowanie, które powinno być podłączone do pinu PWM.
- Sterownik umieszczony w serwomechanizmie odczytuje sygnał PWM i na jego podstawie określa kąt, o który ma się obrócić przekładnia:



Rysunek : Zasada działania silnika krokowego. Źródło: [a]

[a] http://www.henryk.mbapp.com

Obsługa serwomechanizmu

- Do obsługi serwomechanizmu służy biblioteka Servo.h, która jest domyślnie instalowana w środowisku IDE.
- Na początku należy stworzyć obiekt klasy Servo:

Servo nazwa_obiektu;

- W celu powiązania serwomechanizmu z wybranym złączem należy skorzystać z funkcji attach():

nazwa_obiektu.attach(numer pinu PWM);

- Funkcja umożliwiająca sprawdzanie czy serwomechanizm został powiązany z danym pinem to **attached()**.
- Usunięcie powiązania serwomechanizmu z danych pinem:

nazwa_obiektu.detach();

- Odczyt kąta wychylenia serwomechanizmu:

nazwa_obiektu.read();

- Ustawienie kąta wychylenia:

nazwa_obiektu.write(kąt);

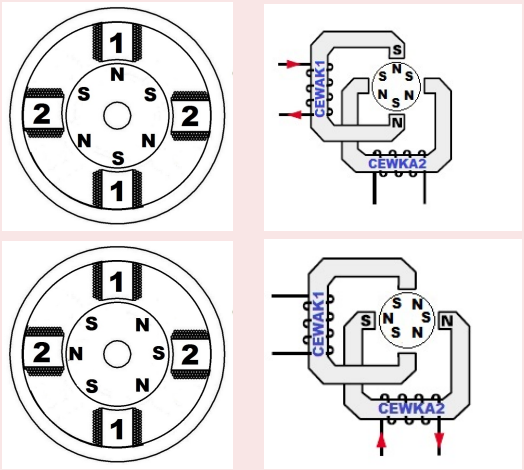
Serwomechanizm - przykład

Przykład: Obracanie serwomechanizmu od minimalnego, przez środkowe do maksymalnego położenia.

```
#include <Servo.h>
Servo serwomechanizm;

void setup()
{
    serwomechanizm.attach(7); //Podlaczenie serwomechanizmu do
        ↪ pinu PWM nr 7
}

void loop()
{
    serwomechanizm.write(0);
    delay(500);
    serwomechanizm.write(90);
    delay(500);
    serwomechanizm.write(180);
    delay(500);
    serwomechanizm.write(90);
    delay(500);
}
```

Silnik krokowy

- Silniki krokowe to typ silnika wykonujący serię pojedynczych kroków. Ze względu na wysoką precyzję i możliwość ścisłej kontroli znalazły zastosowania w drukarkach 3D, maszynach CNC czy w modułach służących do pozycjonowania teleskopów.

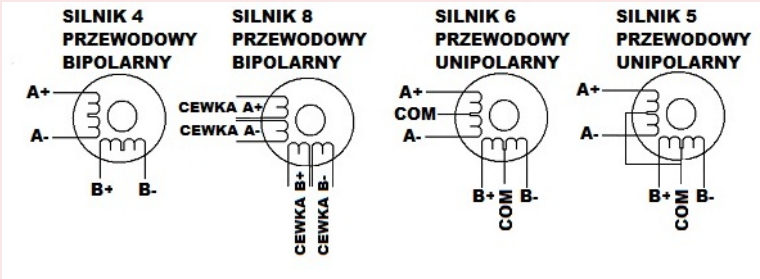


Zasada działania silnika krokowego. Źródło: [a]

[a] http://silniki-krokowe.com.pl/

Silnik krokowy

- Ze względu na budowę wyróżniamy dwa rodzaje sinków:
  - unipolarne
  - bipolarne



- Silniki unipolarne mają tą zaletę, że są łatwe w sterowaniu.
- Silniki bipolarne są bardziej wydajne i posiadają większy moment na jednostkę mocy. Natomiast wymagają zastosowania sterownika, który będzie zmieniał polaryzację zasilania cewek.

Obsługa silnika krokowego

- Do obsługi silnika krokowego zarówno unipolarnego jak i bipolarnego służy biblioteka Stepper, która domyślnie jest instalowana w środowisku IDE.
- Inicjalizacja silnika krokowego:

  Stepper stepper(liczba potrzebnych kroków aby wykonać pełny obrót, pin1, pin2);

  lub:

  Stepper stepper(liczba potrzebnych kroków aby wykonać pełny obrót, pin1, pin2, pin3, pin4);

- Ustawienie prędkości silnika krokowego:

  stepper.setSpeed(liczba obrotów na minutę);

- Obrót silnika o zadaną liczbę kroków. Ujemna liczba kroków oznacza, że silnik obróci się wprzód a dodatnia - wstecz:

  stepper.step(liczba kroków);

Obsługa silnika krokowego - przykład

Przykład: Obracanie silniczkiem w przód aż wykona pełny obrót a następnie wstecz aż wykona pełny obrót.

```
#include <Stepper.h>

Stepper stepper(200,2,3);
int steps=-1;

void setup()
{
    stepper.setSpeed(2); //Ustawienie predkosci = 2 obroty/min.
}

void loop()
{
    for(unsigned int i=1; i<201;i++) stepper.step(steps);
    steps*=(-1);
}
```

Tworzenie bibliotek

- W przypadku gdy jakiś czujnik nie posiada gotowej biblioteki i napiszemy sami kod, który obsłuży danych czujnik korzystając z dokumentacji to możemy taki kod zamienić na bibliotekę. Dzięki takiej postawie liczba bibliotek pod arduino rośnie, a obsługa różnych czujników staje się jeszcze łatwiejsza.
- W celu stworzenia biblioteki należy:
  1. Napisać plik nagłówkowy
  2. Napisać plik implementacji
  3. Napisać plik ze słowami kluczowymi
  4. Stworzyć kilka przykładów
- Biblioteka ma postać folderu, którego nazwa powinna być taka sama jak nazwa stworzonej klasy.
- Korzystając z czujnika temperatury LM75 zademonstrujemy sposób tworzenia biblioteki.

Kod programu - LM75

Poniżej znajduje się kod programu obsługujący czujnik temperatury LM75, który posłuży nam do stworzenia biblioteki:

```
#include <Wire.h>
#define adressTemp (0x90>>1)
float temp=0.0;
unsigned int msb=0, lsb=0;
void setup()
{
     Wire.begin();
}
void loop()
{
      Wire.beginTransmission(adressTemp);
      Wire.write(0x00);
      Wire.endTransmission();
      Wire.requestFrom(adressTemp, 2);
      while(Wire.available()) {
        msb = Wire.read();
        lsb = Wire.read();
        lsb = (lsb & 0x80 ) >> 7;
        temp = msb + 0.5 * lsb;
      }
}
```

Utworzenie pliku nagłówkowego

- Plik nagłówkowy - zawiera definicje klas i metod znajdujących się w bibliotece. Do niego odwołujemy się wywołując: #include.

```
//Plik: LM75.h
#include <Wire.h>
#ifndef LM75_h
#define LM75_h

class LM75
{
    private:
        int _address;
        unsigned int _msb;
        unsigned int _lsb;
        float _temp;
    public:
        void LM75(int address);
        float readTemperature();
};

#endif
```

Utworzenie pliku implementacyjnego

```
//Plik LM75.cpp
#include <Arduino.h>
#include <LM75.h>
LM75::LM75(int address)
{
    _address=address;
}
float LM75::readTemperature()
{
     _temp=0.0;
     Wire.beginTransmission(_address);
     Wire.write(0x00);
     Wire.endTransmission();

     Wire.requestFrom(_address, 2);
     while(Wire.available()) {
         _msb = Wire.read();
         _lsb = Wire.read();
         _lsb = (_lsb & 0x80 ) >> 7;
         _temp = _msb + 0.5 *_lsb;
    }
    return _temp;
}
```

Utworzenie pliku ze słowami kluczowymi

- Dobrą praktyką jest umieszczanie w katalogu biblioteki pliku *keywords.txt*. Plik ten nie jest wymagany ale umieszczenie jego sprawi, że słowa kluczowe będą zaznaczone innym kolorem w szkicu.

```
###################################
# Syntax Coloring Map for LM75
###################################
###################################
# Datatypes (KEYWORD1)
###################################
LM75 KEYWORD1
###################################
# Methods and Functions (KEYWORD2)
###################################
readTemperature KEYWORD2
```

Utworzenie przykładu

- Aby stworzona biblioteka była łatwa w użyciu dla osób trzecich należy stworzyć folder *examples* w katalogu biblioteki. W środku powinien znaleźć się szkic, który będzie demonstrował użycie biblioteki.
- Szkice umieszczone w katalogu *examples* będą widoczne w *Plik* → *Przykłady*. W naszym przypadku stworzony poniżej przykład będzie można uruchomić wybierając *Plik* → *Przykłady* → *LM75* → *LM75example*.

```
//Szkic nazwany: LM75example.ino
#include <Wire.h>
#include <LM75.h>
#define adres (0x90>>1)

float temperatura=0.0;
LM75 czujnik= LM75(adres);
void setup()
{
    Wire.begin();
    Serial.begin(9600);
}
void loop()
{
    temperatura=czujnik.readTemperature();
    Serial.println(temperatura);
}
```

Publikacja biblioteki

W celu opublikowania swojej biblioteki zastosuj się do poniższych wskazówek:

1. Upewnij się, że Twoja biblioteka działa bez zarzutu. Najlepiej poproś jeszcze kogoś o niezależne przetestowanie.
2. Spakuj katalog biblioteki do archiwum o rozszerzeniu ZIP. Nazwa archiwum powinna być identyczna z nazwią biblioteki.
3. Załóż konto użytkownika na serwerze: `http://www.arduino.cc/`.
4. Umieść informacje o swojej bibliotece na stronie: `http://playground.arduino.cc/Main/LibraryList` oraz stwórz odnośnik do noty np. [[LM75]]. Tak będzie się nazywać Twoja biblioteka na liście.
5. Wypełnij utworzony odnośnik, tworząc notę do swojej biblioteki.
6. Dodaj bibliotekę do noty: Attach:LM75.zip. Po zapisaniu, kliknij na odnośnik i załaduj archiwum ze swojego komputera.

Zwiększenie liczby portów szeregowych

- Czasami istnieje konieczność skorzystania z kilku urządzeń korzystających z portu szeregowego np. GPS, niektóre termometry itd. Arduino MEGA zawiera aż 4 porty szeregowe natomiast Arduino UNO posiada tylko jedno.
- Jeśli potrzebujesz skorzystać z mniejszej płytki niż Arduino MEGA oraz skorzystać z kilku portów szeregowych to istnieje biblioteka *SoftwareSerial*, która zasymuluje na danych dwóch pinach cyfrowych funkcjonalność portu szeregowego.
- Skorzystanie z kilku symulowanych portów szeregowych ma pewne ograniczenia. Po pierwsze takie porty mogą przesyłać dane z prędkością maksymalną 115 000 bodów. Po drugie w tym samym czasie nie można nasłuchiwać na kilku symulowanych portów szeregowych jednocześnie.

SoftwareSerial

- Utworzenie symulowanego portu szeregowego:

  SoftwareSerial nazwa(pin który ma służyć do obioru danych RX, pin który ma służyć do wysyłania danych TX);

- Uaktywnienie portu i ustawienie szybkości transmisji danych:

  nazwa.begin(szybkość transmisji);

- Funkcja sprawdzająca czy dane są gotowe do obioru:

  nazwa.available();

- Funkcja sprawdzająca czy jest aktywny dany port:

  nazwa.isListening();

- Funkcja, która nasłuchuje na porcie i czyni go aktywnym:

  nazwa.listen();

- Funkcja, która kończy pracę portu:

  nazwa.end();

- Funkcja do odczytu danych z portu:

  nazwa.read();

- Funkcja do wysyłania danych przez port:

  nazwa.write(byte lub string);

SoftwareSerial - przykład

Przykład: Odczyt danych z GPS i termometru przez port szeregowy korzystając z Arduino UNO:

```
#include <SoftwareSerial.h>

SoftwareSerial gpsPort(2,3);
SoftwareSerial tempPort(4,5);

void setup()
{
    gpsPort.begin(9600);
    tempPort.begin(9600);
}
void loop()
{
    gpsPort.listen();
    while(gpsPort.available())
    {
        char gpsByte=gpsPort.read();
    }

    tempPort.listen();
    while(tempPort.available())
    {
        char tempByte=tempPort.read();
    }
}
```

Obsługa karty SD

- Do obsługi karty SD jest gotowa biblioteka *SD*, która domyślnie instalowana jest w środowisku IDE. W środku tej biblioteki mamy do dyspozycji dwie klasy: SD i File.
- Klasa SD posiada 6 funkcji:
  - Funkcja do inicjalizacji karty SD:

    SD.begin();
  - Funkcja do sprawdzenia czy na karcie SD istnieje katalog lub plik:

    SD.exists(nazwa katalogu lub pliku);
  - Funkcja, która tworzy katalog:

    SD.mkdir(nazwa katalogu);
  - Funkcja, która usuwa katalog z karty pamięci:

    SD.rmdir(nazwa katalogu);
  - Funkcja, która otwiera plik z karty pamięci:

    SD.open(nazwa pliku, FILE_READ lub FILE_WRITE);
  - Funkcja, która usuwa plik z karty pamięci:

    SD.remove(nazwa pliku);

Obsługa karty SD

- Klasa File posiada 15 funkcji. Najczęściej stosowane to:
  - Utworzenie obiektu klasy File:

    File file=SD.open(nazwa pliku, FILE_READ lub FILE_WRITE);
  - Funkcja, która sprawdza czy z pliku można odczytać kolejne bajty:

    file.available();
  - Funkcja, która zamyka dostęp do pliku:

    file.close();
  - Funkcja, która ustawia nową pozycje w pliku:

    file.seek(pozycja zapisana w zmiennej typu unsigned long);
  - Funkcja, która odczytuje aktualną pozycje w pliku:

    file.position();
  - Funkcja, która zwraca rozmiar pliku:

    file.size();
  - Funkcje, które zapisują dane do pliku:

    file.write(dane w formacie byte, char lub string);
    file.print(dane w formacie char, byte, int, long, lub string);
    file.println(dane w formacie char, byte, int, long, lub string);
  - Funkcja która odczytuje dane z pliku:

    file.read();

Obsługa karty SD - przykład

Przykład: Zapis danych na kartę SD.

```
#include <SD.h>
#define SS 53

File file;

void setup()
{
    Serial.begin(9600);
    if(!SD.begin(SS)) Serial.println("Karta SD - blad przy
        ↪ inicjalizacji");
    file=SD.open("wyniki.txt", FILE_WRITE);
}
void loop()
{
    file=SD.open("wyniki.txt", FILE_WRITE);
    if(SD.exists("wyniki.txt"))
    {
        Serial.println("Zapis wynikow na karte SD");
        file.println(10);
    }
    file.close();
}
```

Obsługa klawiatury numerycznej

- Do obsługi klawiatury numerycznej jest gotowa biblioteka *Keypad*, którą można pobrać ze strony: http://playground.arduino.cc/uploads/Code/Keypad.zip
- **Przykład:** Odczyt danych wprowadzonych przy użyciu klawiatury numerycznej i wyświetlenie ich w monitorze portu szeregowego.

```
#include "Keypad.h"

const byte ROWS=4; //Liczba wierszy w klawiaturze
const byte COLS=3; //Liczba kolumn w klawiaturze

char keys[ROWS][COLS]={ {'1','2','3'}, {'4','5','6'},{'7','8','
    ↪ 9'},{'*','0','#'}}; //Ulozenie cyfer na klawiaturze

byte rowPins[ROWS]={5,4,3,2};
byte colPins[COLS]={8,7,6};

Keypad keypad = Keypad(makeKeymap(keys),rowPins,colPins,ROWS,
    ↪ COLS); //Inicjalizacja klawiatury - podanie may klawiszy
    ↪ , numery pinow wierszy, numery pinow kolumn, liczby
    ↪ wierszy i liczby kolumn.
```

Obsługa klawiatury numerycznej

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

void loop()
{
    char key=keypad.getKey();
    if(key!=NO_KEY) Serial.println(key);
}
```

Liczniki (Timery)

- Liczniki (Timery) są niezwykle przydatne do np: wywoływanie wewnętrznie przerwań co określony czas lub generowania sygnału PWM. Do obsługi liczników jest kilka różnych bibliotek lecz najczęściej spotykana to: *TimerOne*. Jej działanie zademonstrujemy korzystając z dwóch przykładów:
- **Przykład 1:** Generowanie sygnału prostokątnego z częstotliwością 1kHz:

```
#include <TimerOne.h>
int pin=9;
volatile int value=LOW;

void setup()
{
    pinMode(pin, OUTPUT);
    Timer1.initialize(500); //Okreslenie co ile ma byc wykonywane
        ↪ przerwanie - czas w mikrosekundach.
    Timer1.attachInterrupt(signal); //Okreslenie funkcji przerwania
}
void signal()
{
   digitalWrite(pin, value);
   value=!value;
}
void loop()
{

}
```

Liczniki (Timery)

- **Przykład 2:** Generowanie sygnału PWM z częstotliwością 1kHz:

```
#include <TimerOne.h>
int pin=9;

void setup()
{
    pinMode(pin, OUTPUT);
    Timer1.initialize(1000); // 1000us=1ms -> 1kHz
    Timer1.pwm(pin, 512); //Generowanie sygnalu PWM o
        ↪ wypelnieniu 50 % -> zakres wypelnienia (0,1023).
}

void loop()
{

}
```

THAT'S ALL FOR TODAY....
ANY QUESTION??