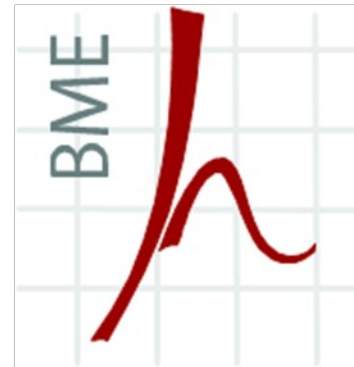




Számítógép Architektúrák

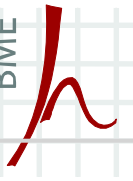
Az ajtónyitó megvalósítása Arduino-val

Horváth Gábor
egyetemi tanár

BME Hálózati Rendszerek és Szolgáltatások Tsz.

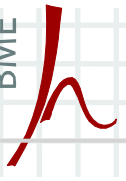
horvath.gabor@vik.bme.hu

2022. márc. 8.
Budapest



Tartalom

- A gyakorlat célja:
 - Megmutatni, hogy a manapság elterjedt és olcsó mikrokontrollerekkel milyen könnyű dolgozni
- Tartalomjegyzék:
 - Az Arduino bemutatása
 - Hardver
 - Programozás
 - Szenzorok
 - Az ajtónyitó megvalósítása
 - A kijelző
 - Az RFID olvasó
 - A billentyűzet
 - A teljes hardver
 - A teljes kód



Arduino

- 2005, Ivrea, Olaszország
- Célja: egyszerű prototípus fejlesztés
- Kezdetben egyszerű hardverrel:
 - Atmel AVR 8 bites CPU
(**Harvard** architektúra, 8 – 16MHz)
 - Beépített flash memória a programnak (32 – 256kB)
 - Beépített EEPROM tartós adattárolásra (1 – 4kB)
 - Beépített RAM (2 – 8kB)
 - Digitális és analóg ki- és bemenetek
- Óriási piaci igény van rá!
 - Olcsó
 - USB port-on át programozható (nem kell speci eszköz hozzá)
 - Magas szinten, C++ nyelven programozható



- Open source hardver → milliányi variáns milliányi hardverrel
 - **Ami közös bennük: az interfész**
 - Az eredeti Arduino-val kompatibilis tűskesor
 - Az eredeti Arduino-val kompatibilis programozói felület
 - Kvázi szabvány lett!
 - Az „Arduino” ma már ezt jelenti
 - ...és nem konkrét gyártót/típust
 - Google „Arduino” → 114 000 000 találat!
 - Hatalmas a közösség
- Az Arduino ma már nem „játékszer”, komoly szereplő
- „Konkurencia”: Raspberry Pi
 - A másik kvázi szabvány (tűskesor + API)
 - Szintén hatalmas közösség: 200 000 000 találat

Arduino vs Raspberry Pi

Arduino Mega

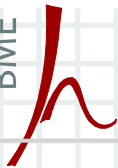
CPU: 16 MHz (8 bit)
RAM: 8 kB
Flash: 256 kB
I/O: 54 digitális, 16 analóg
Ár: 35€

Raspberry Pi 4

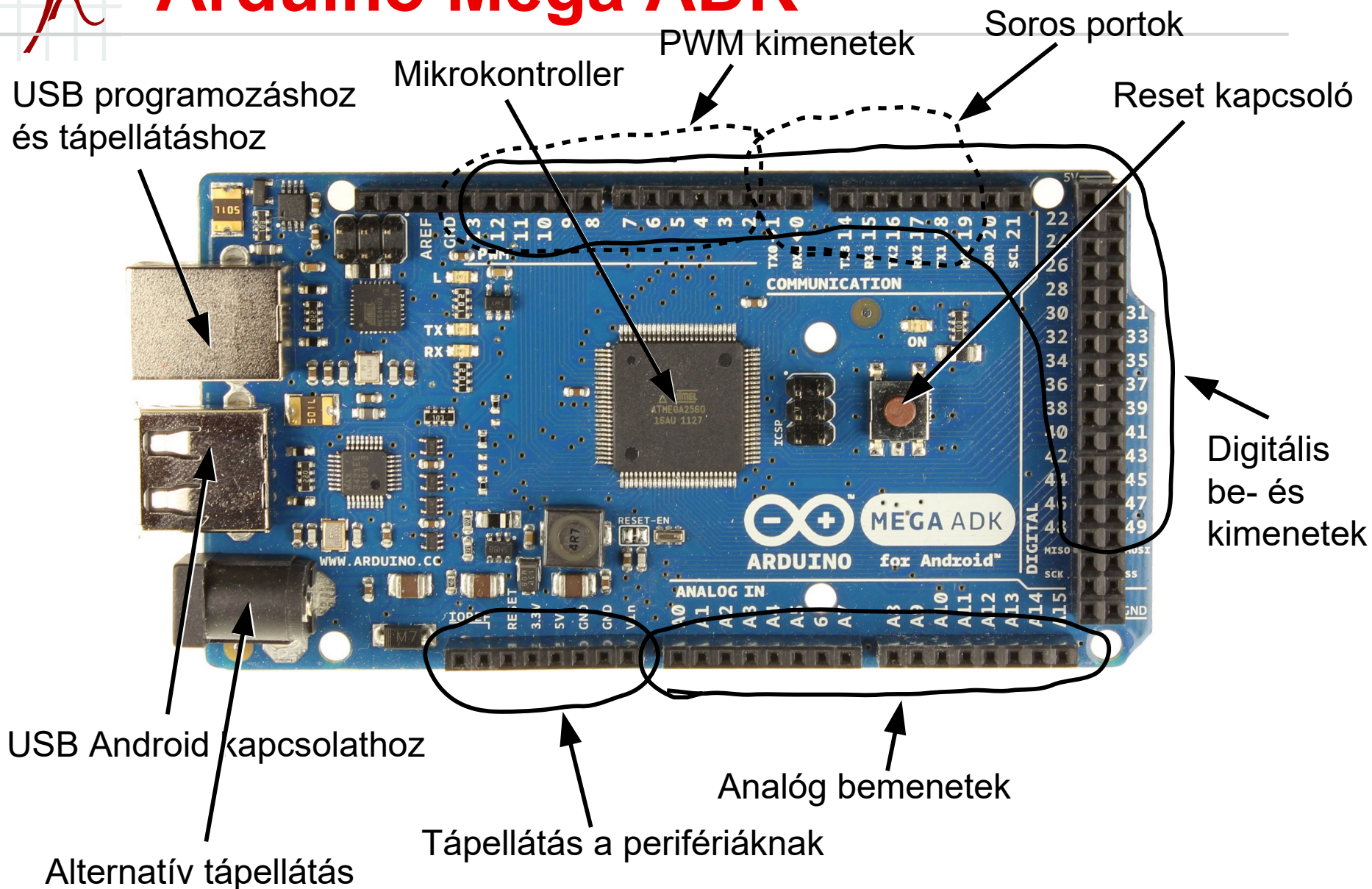
CPU: 1.5 GHz, 4 mag (64 bit)
RAM: 1 GB LPDDR4
Flash: MicroSD (64 GB)
I/O: 40 GPIO
+ Gb Ethernet, 2xHDMI, USB3
Ár: 35\$

▪ Más a cél:

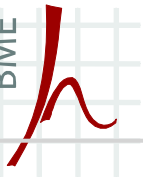
- Arduino:
 - Hangsúly: I/O, minden mennyiségben, pofonegyszerűen
 - Bekapcsolom, és 1 másodperc múlva már megy is
 - A fenténél olcsóbbak is vannak (pl. NodeMCU: 4\$, WiFi-vel)
 - **Nagyon sok az Arduino kompatibilis eszköz**
- Raspberry Pi:
 - Hangsúly: általános célú (programozás oktatás)
 - Op. rendszer (linux) kell rá! Sokáig tart, mire elindul.
 - I/O ki/bemeneteket nehezebb használni
 - Analóg ki/bemenet nincs, soros perifériák illesztése problémás



Arduino Mega ADK



- Open-source cross-platform fejlesztői környezet
 - Arduino IDE
 - Visual Studio Code + PlatformIO extension
- Programnyelv:
 - Arduino program: „sketch” $\approx$ C++ (.ino)
 - Elemi adattípusok: int, long, bool, **float**, char
- Két kötelező függvény:
 - **void setup () { ... }** - az ide írt kód induláskor fut le, egyetlenegyszer
 - **void loop () { ... }** - inicializálás után ez a függvény fut le újra és újra (ha befejeződik, újraindul a függvény elejétől)
- Fordítás:
 - A PC-s fejlesztőkörnyezet idegen kódot fordít $\rightarrow$ cross compiler
 - USB-n keresztül feltölti a mikrokontroller flash memóriájába
- Hibakeresés:
 - Amit az Arduino a default soros portjára ír, azt USB-n keresztül megmutatja a fejlesztőkörnyezet



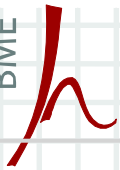
Ki/Bemenetek

Digitális ki- és bemenetek

- A digitális lábak ki és bemenetként is szolgálhatnak
 - De egyszerre csak az egyik szerepben!
 - Beállítás:
 - **pinMode (4, INPUT);** – 4-es láb → bemenet üzemmód
 - **pinMode (5, OUTPUT);** – 5-ös láb → kimenet üzemmód
- A digitális lábak magasba, ill. alacsonyba húzása:
 - **digitalWrite (5, HIGH);** – az 5-ös lábra logikai 1-et tesz (5V)
 - **digitalWrite (5, LOW);** – az 5-ös lábra logikai 0-át tesz (0V)
- A digitális lábak olvasása:
 - **int val;**
 - **val=digitalRead (4);** – a 4-es láb logikai értékének olvasása
 - **if (val==HIGH) ...**, vagy **if (val==LOW) ...**

Analóg bemenetek

- Nemcsak HIGH és LOW értékeket lehet róla olvasni
- ADC (Analog-Digital Converter) 10 bites felbontással
 - 1024 különböző feszültség szintet tud megkülönböztetni
- 0V és 5V között 1024 szint → 4.88 mV különbséget tud tenni
- Használata:
 - **int val;**
 - **val = analogRead (3);** - a 3-mas analóg bemenetről mintát vesz
 - **val** értéke: 0-tól (0V esetén) 1023-ig (5V)
- Az 5V maximum referencia változtatható az **analogReference()** függvényvel, de 5V a maximum

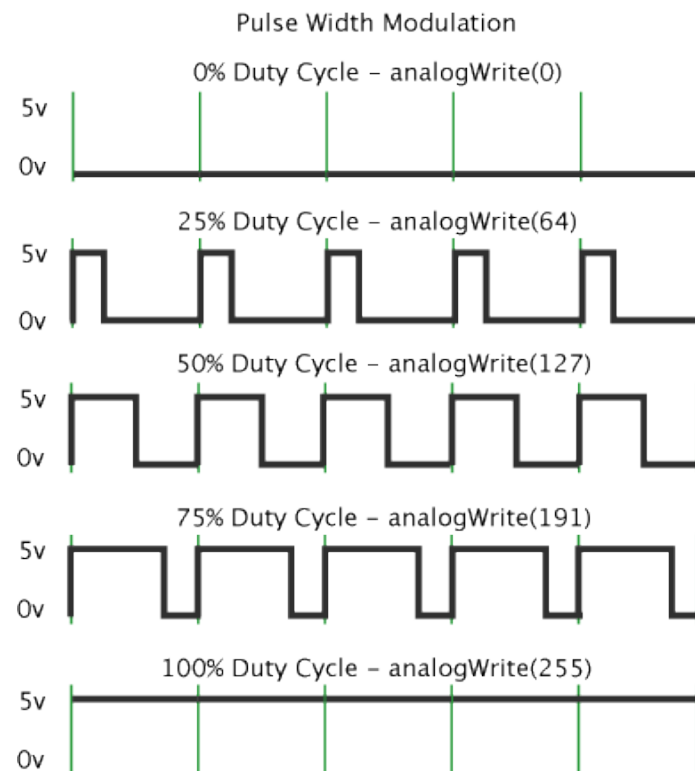


PWM kimenetek

- Folyamatos analóg kimenet (pl. 4.2V) nem tud előállítani
- Viszont a kimenet gyors ki-be kapcsolgatásával „átlagos értelemben” tetszőleges feszültséget elő tud állítani

→ **PWM: Impulzusszélesség-moduláció**

- Erre szolgál az **analogWrite ()** függvény
- Paraméterek: láb száma, kitöltöttség (Duty Cycle, 0...255 között)

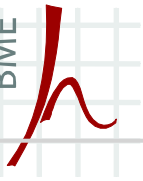


Soros ki- és bemenetek

- A kommunikációhoz max. 2 ér kell:
 - RX: a vételhez (ha szükség van vételre)
 - TX: az adáshoz (ha szükség van adásra)
- Az Arduino ADK 4 soros portot tartalmaz
 - Az elsőt használja Debug célra a fejlesztőkörnyezet USB-n keresztül
- Használata: **class Serial**-on keresztül
 - Előregyártott példányok: **Serial**, **Serial1**, **Serial2**, **Serial3**
 - *Inicializálás*: **Serial1.begin (9600);** - port megnyitása 9600 bps sebességre
 - *Írás*: **Serial1.write (...);** - elküld 1 bájtot, egy NULL terminált string-et, vagy egy tömböt
 - *Írás*: **Serial1.print (...);** - a paraméterét string-é kovertálja (ha nem az), és elküldi. A **Serial1.println (...);** még egy soremelést is utánatesz.
 - *Olvasás*: **int data = Serial1.read();** - egy megérkezett bájt kiolvasása (-1, ha nincs)
 - *Érkezett-e adat*: **int count = Serial1.available();** - megadja, hány bájt érkezett
 - *Lezárás*: **Serial1.end ();** - a port lezárása után a megfelelő láb digitális ki- és bemenetként használható tovább

Soros ki- és bemenetek

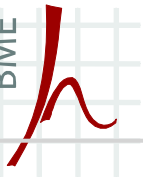
- Működése:
 - Vételkor Interrupt keletkezik
 - Az Interruptot lekezeli az Arduino:
 - Beleteszi az érkezett bájtot egy tárolóba (buffer)
 - A **read()** a tárolóból olvas
 - Az **available()** a tárolóban lévő bájtok számát adja vissza



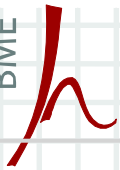
Memóriák

- Az Arduino kártyák az alábbi memóriákkal rendelkezhetnek:
 - **Flash memória**: ez tárolja a programot
 - Tápfeszültség nélkül is megmarad a tartalma
 - **RAM**: ebben vannak a változók és a stack is
 - Tápfeszültség kell a tartalom megőrzéséhez
 - **EEPROM**: tartósnak szánt adatokhoz
 - Tápfeszültség nélkül is megmarad a tartalma
- Az AVR processzorok Harvard architektúrát követnek
 - 2 címtér:
 - Az utasításokat és konstansokat a flash memóriából veszi
 - A változókhoz/stack-hez a RAM-ot használja
 - És az EEPROM?
 - Ahhoz külön függvénykönyvtár van

- A rendszer része az EEPROM könyvtár
- `#include <EEPROM.h>`
- Írás:
 - **`EEPROM.write (cím, adat);`**
 - A cím int típusú
 - Az adat byte típusú
- Olvasás:
 - **`byte a;`**
`a = EEPROM.read (42);`
 - Kiolvassa a 42-edik byte-ot az EEPROM-ból



Perifériák illesztése



Digitális kimeneti perifériák

- **Példa:**

LED villogtatás (Nem Arduino ADK, de ugyanaz az elv)

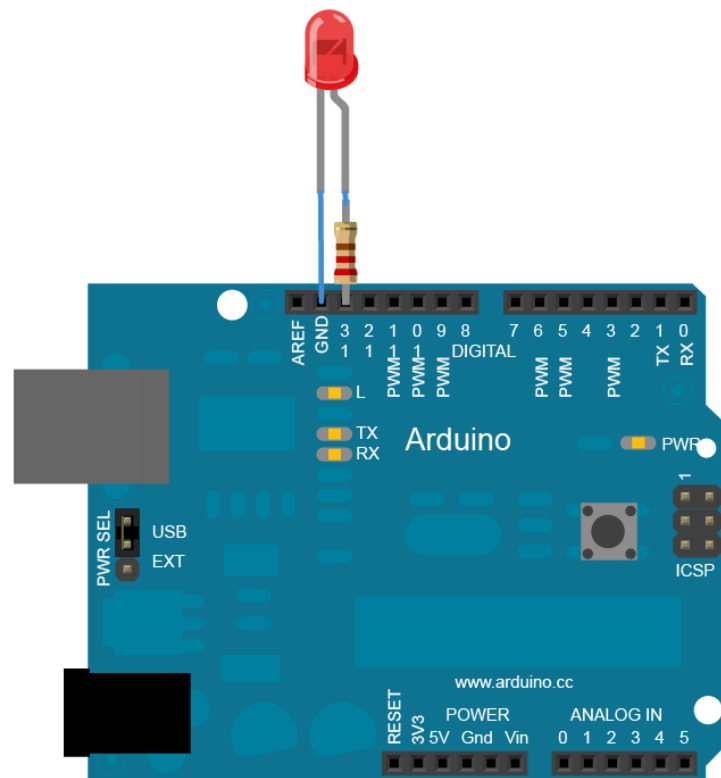
Alkatrész: LED, 220 Ohmos ellenállás

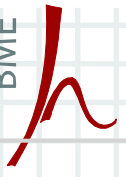
LED nélkül is megy!

A 13-mas lábon van beépített LED.

- **Kód:**

```
const int ledPin = 13;
void setup () {
  pinMode (ledPin, OUTPUT);
}
void loop () {
  digitalWrite (ledPin, HIGH);
  delay (1000);
  digitalWrite (ledPin, LOW);
  delay (1000);
}
```





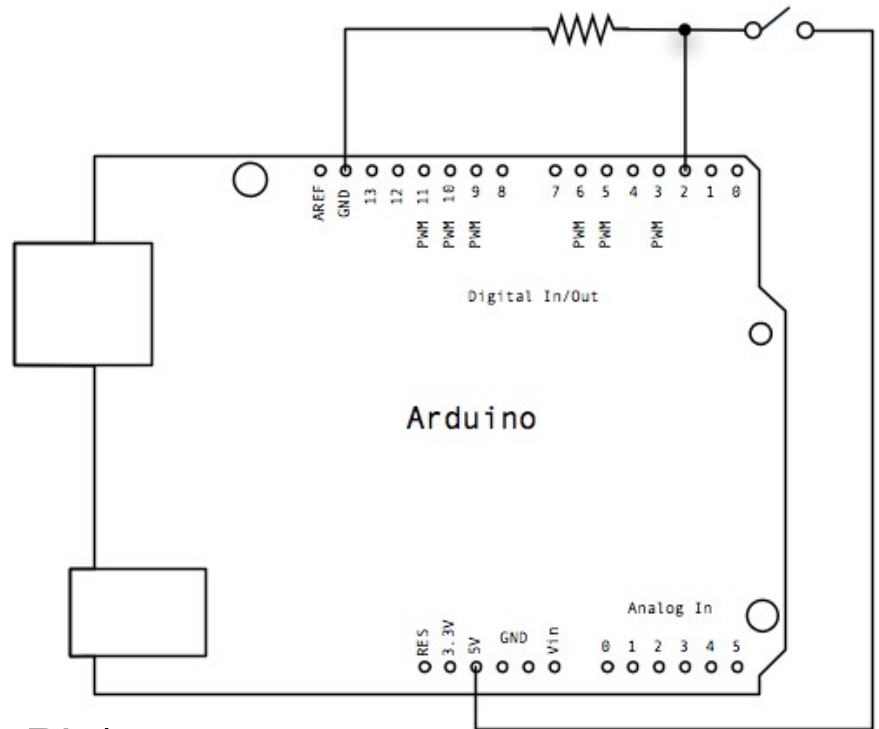
Digitális bemeneti perifériák

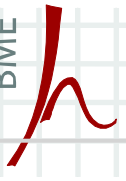
▪ Példa:

Nyomógomb állapotának lekérdezése, a beépített LED kigyújtása

▪ Kód:

```
const int buttonPin = 2;  
const int ledPin = 13;  
int buttonState = 0;  
void setup () {  
    pinMode (ledPin, OUTPUT);  
    pinMode (buttonPin, INPUT);  
}  
void loop () {  
    buttonState = digitalRead (buttonPin);  
    digitalWrite (ledPin, buttonState);  
}
```



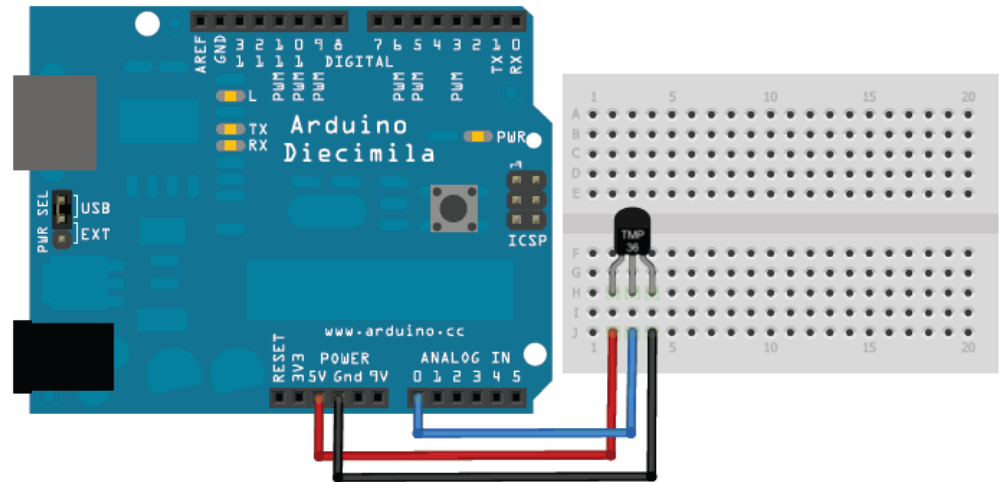


Analóg bemeneti perifériák

- **Példa:**
TMP36 Hőmérséklet szenzor illesztése

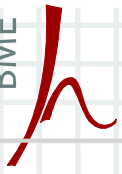
- **Kód:**

```
const int sensorPin = 0;  
void setup () {  
    Serial.begin (9600);  
}  
void loop () {  
    int reading = analogRead (sensorPin);  
    float voltage = reading * 5.0 / 1024.0;  
    float temperature = (voltage - 0.5) * 100;  
    Serial.println (temperature);  
    delay(1000);  
}
```



Analóg bemeneti perifériák

- A legérdekesebb perifériák!
 - Filléres szenzorok tömkelege:
 - 3 tengelyes gyorsulásmérő (3 analóg bemenetet használ)
 - Alkohol szenzor
 - Szénmonoxid szenzor
 - Szálló por koncentráció szenzor
 - Elhajlásmérő szenzor
 - Erő (súly) mérő szenzor
 - Vibráció szenzor
 - Giroszkóp (2 tengelyes → 2 analóg bemenet kell)
 - Távolságmérő (infra fénnnyel vagy ultrahanggal)
 - Hőmérő
 - Páratartalommmérő
 - Stb.
- A mért fizikai mennyiséget analóg jellé alakítják

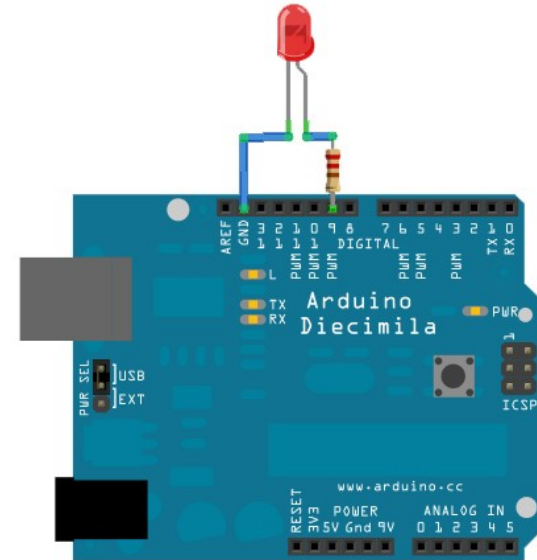


PWM kimeneti perifériák

- **Példa:**
LED halványítás/fényesítés

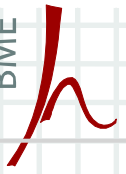
- **Kód:**

```
const int ledPin = 9;
void setup () {
}
void loop () {
  for (int fadeValue = 0 ; fadeValue <= 255; fadeValue +=5) {
    analogWrite (ledPin, fadeValue);
    delay (30);
  }
  for (int fadeValue = 255 ; fadeValue >= 0; fadeValue -=5) {
    analogWrite (ledPin, fadeValue);
    delay (30);
  }
}
```



Soros perifériák

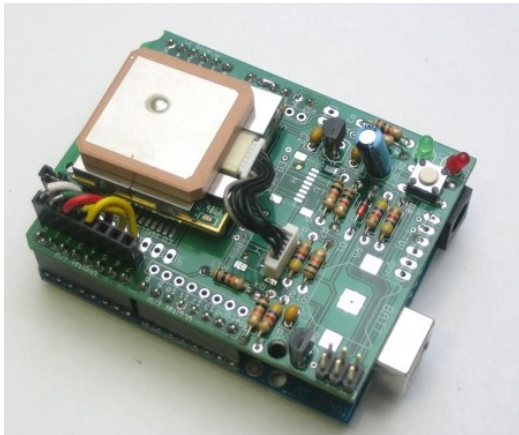
- Példák soros portra csatlakozó perifériákra:
 - RFID modul: soros porton küldi a leolvasott kártya kódját
 - GPS modul: soros porton küldi rendszeresen a koordinátákat
 - GSM/GPRS modul: soros porton kell vezérelni, illetve az átvihető adatok is a soros porton mennek
 - Stb.



Pajzsok (Shield)

- Az Arduino tetejére illeszthető komplett perifériák
 - Nagyon sok van:
 - GPS, LCD vezérlő, SD kártya kezelő, WIFI, Bluetooth, ZigBee, GSM, ...

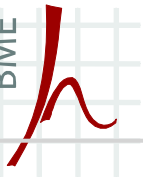
GPS Shield
€18



MP3 Shield
€25

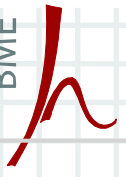
2.8" TFT és érintőképernyő
€40





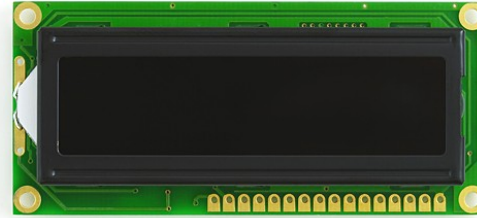
Az ajtónyitó megvalósítása

- Milyen alkatrészek is kellenek?
 - Egy kis kijelző, amire üzeneteket írhatunk a felhasználónak
 - Egy numerikus billentyűzet
 - Egy kártyaleolvasó – RFID jó lesz
 - Egy relé, ami az ajtót nyitja
 - Egy megfelelően választott Arduino a népes családból

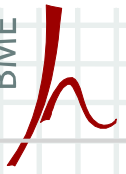


A kijelző

- Olcsó 2 soros, 16 oszlopos kijelző (2000 Ft)



- Lábak:
 - *Adatbusz*: D0...D7, de nekünk 4 bites üzemmódban is jó (D4...D7)
 - *RS*: ezzel lehet jelezni a kijelzőnek, hogy egy ASCII karaktert küldünk neki, vagy vezérlő parancsot
 - *EN*: engedélyezés, a kijelző ekkor olvassa el a neki küldött adatot
 - *RW*: ha nem adatot küldünk a kijelzőnek, hanem tőle kérdezzük le információt – ezt nem kötjük be
 - *Tápfeszültség* (5V)
 - *Kijelző kontraszt* (potméter)
 - *Háttérvilágítás tápellátása*
- 6 adatvezetékét használunk: RS, EN, D4, D5, D6, D7

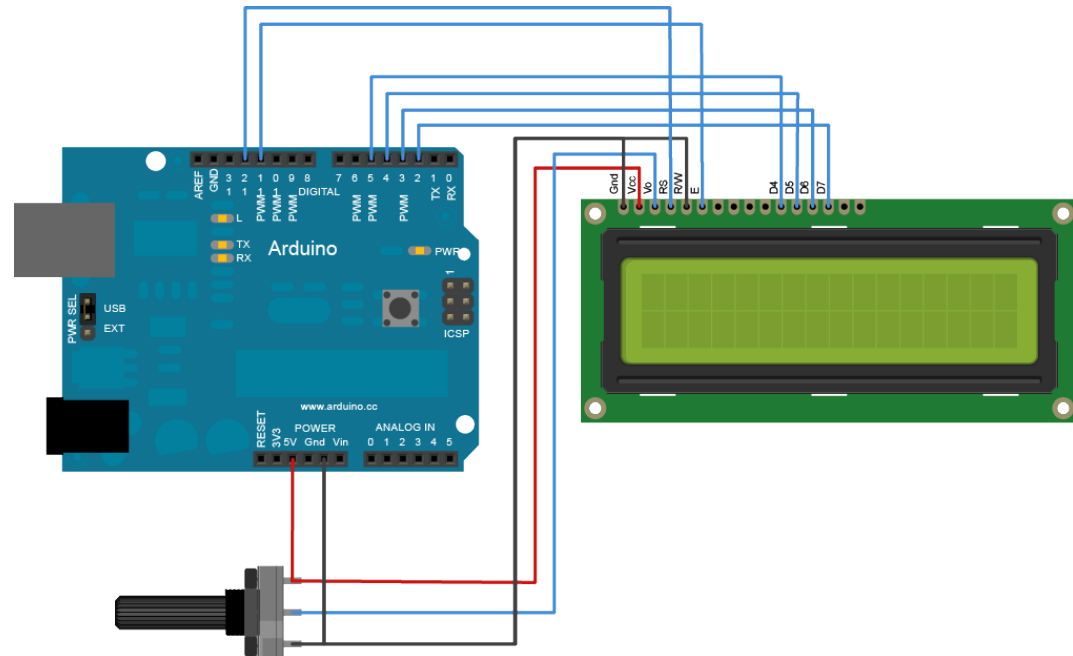


A kijelző használata

- **Bekötés:**
Google „arduino display”
- **Használata:**
LiquidCrystal class példányosítása
(write(), setCursor(), blink(), clear(), stb.)

- **Példa:**

```
#include <LiquidCrystal.h>
const int numRows = 2;
const int numCols = 16;
// beállítjuk, melyik lábakra kötöttük
LiquidCrystal lcd (12, 11, 5, 4, 3, 2);
void setup () {
  lcd.begin (numCols,numRows);
}
void loop () {
  for (int thisLetter = 'a'; thisLetter <= 'z'; thisLetter++) {
    for (int thisRow = 0; thisRow < numRows; thisRow++) {
      for (int thisCol = 0; thisCol < numCols; thisCol++) {
        lcd.setCursor (thisCol, thisRow);
        lcd.write (thisLetter);
        delay (200);
      }
    }
  }
}
```

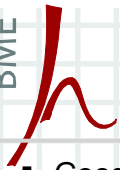


A numerikus billentyűzet

- Bárhol beszerezhető, 4x3-mas (900 Ft)



- Lábak:
 - 7 láb, a 4 sorhoz és a 3 oszlophoz
 - Ha megnyomunk egy gombot, rövidre zárja a sor és oszlopvezetékét



A numerikus billentyűzet használata

- Google „arduino keypad”

- **Használata:**

Keypad class példányosítása (utána getKey(), waitForKey(), getState(), stb.)

- **Példa:**

```
#include <Keypad.h>
```

```
const byte ROWS = 4;
```

```
const byte COLS = 3;
```

```
char keys[ROWS][COLS] = { {'1','2','3'}, {'4','5','6'}, {'7','8','9'}, {'*','0','#'} };
```

```
byte rowPins[ROWS] = {32, 22, 24, 28}; // sorok lábait hova kötöttük az Arduino-n
```

```
byte colPins[COLS] = { 30, 34, 26 }; // oszlopok lábait hova kötöttük az Arduino-n
```

```
Keypad keyPad = Keypad ( makeKeymap (keys), rowPins, colPins, ROWS, COLS );
```

```
#define ledpin 13
```

```
void setup () {
```

```
    pinMode (ledpin, OUTPUT);
```

```
    digitalWrite (ledpin, HIGH);
```

```
}
```

```
void loop () {
```

```
    char key = keyPad.getKey();
```

```
    if(key) {
```

```
        switch (key) {
```

```
            case '*':
```

```
                digitalWrite(ledpin, LOW);
```

```
                break;
```

```
            case '#':
```

```
                digitalWrite(ledpin, HIGH);
```

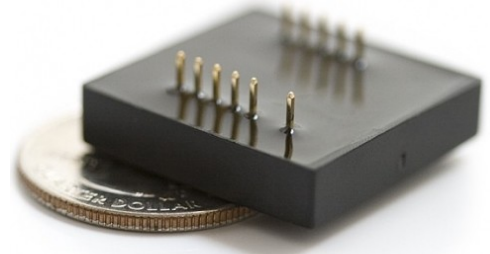
```
                break;
```

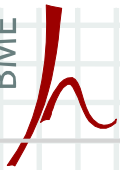
```
        }
```

```
    }
```

```
}
```

- Rádiófrekvenciás azonosítás
 - Minden kártyának garantáltan egyedi, 12 hexa karakteres kódja van
 - Leolvasó: drágán vettük (7000 Ft), már olcsóbb is van
 - Kártya: olcsó (250 Ft/db, kártya, korong, kulcstartó, stb.)
-
- Soros porton kommunikál
 - Lábak:
 - Tápfeszültség (5V)
 - Külső antenna (nekünk a belső bőven elég)
 - Formátum kiválasztó (mi az ASCII módot választjuk)
 - 2 vonal adatátvitelre (csak az egyiket használjuk, soros portként)
 - LED/berregő a kártyaleolvasáskor (nem kötünk rá ilyet)
 - Reset
 - Arduinohoz 2 vezetéket kötünk: Reset (digitális lábra), D0 (soros RX lábra)





Az RFID olvasó használata

- Google „arduino id-12”

- **Használata:**
soros eszközként

- **Példa:** (reset a 2-es, D0 az RX1 lábra kötve)

```
const int RFIDResetPin = 2;
```

```
char userID[13];
```

```
void setup () {
```

```
    Serial.begin(9600);
```

```
    Serial1.begin(9600);
```

```
    pinMode(RFIDResetPin, OUTPUT);
```

```
    digitalWrite(RFIDResetPin, LOW);
```

```
    digitalWrite(RFIDResetPin, HIGH);
```

```
}
```

```
void loop () {
```

```
    if (Serial1.available()) {
```

```
        int readByte = Serial1.read();
```

```
        if (readByte == 2)
```

```
            index = 0;
```

```
        else if (readByte == 3) {
```

```
            userID[index] = 0;
```

```
            Serial.println(userID);
```

```
        }
```

```
        else
```

```
            userID[index++] = readByte;
```

```
    }
```

```
}
```

```
// A 2-es digitális lábra kötöttük a leolvasó reset lábát
```

```
// Ebbe a tömbbe olvassuk majd be a kártya azonosítóját
```

```
// 0-ás soros port beállítása 9600 bps-re (debug-hoz)
```

```
// 1-es soros port beállítása 9600 bps-re (RFID-t ide kötöttük)
```

```
// RFID Reset láb beállítása „kimenet” módba
```

```
// RFID reset, felfutó élet csinálunk. Alacsonyba visszük...
```

```
// ... majd magasba emeljük.
```

```
// Ha jött byte az RFID soros portjától, feldolgozzuk.
```

```
// Kiolvassuk a következő byte-ot
```

```
// Az ASCII 0x2 az „üzenet eleje” karakter
```

```
// Visszapörgetjük az indexet, a userID-t az elejétől töltögetjük.
```

```
// Az ASCII 0x3 az „üzenet vége” karakter
```

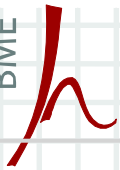
```
// Lezárjuk, NULL terminált karakterlánccá tesszük.
```

```
// Elküldjük a debug rendszernek. Ki fogja írni a PC-n.
```

```
// Ha az üzenet közepén vagyunk, eltároljuk a karaktert.
```

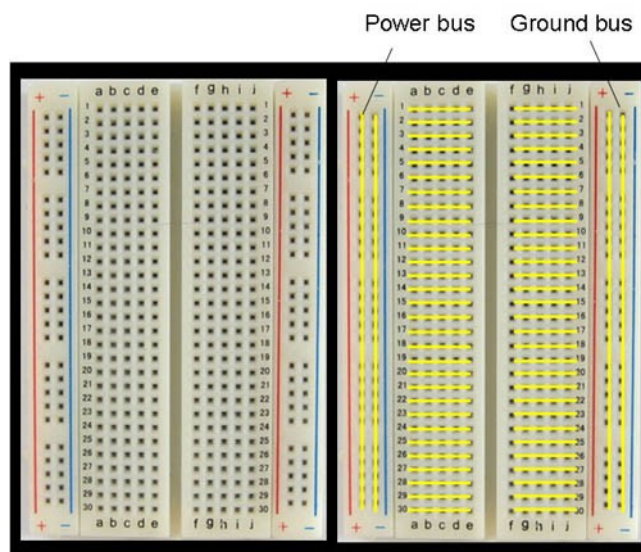
A megfelelő Arduino kiválasztása

- Szempontok:
 - Hány digitális ki- és bemenet kell:
 - A kijelzőnek: 6
 - A billentyűzetnek: 7
 - Az RFID olvasónak: 2
 - Összesen: 15
 - Az Arduino Mega ADK bőven megfelel (54-et tud)
 - További ki- és bemenetek:
 - Analóg be, PWM ki nem kell
 - Soros port kell az RFID-nek
 - Ha debug-olni is akarunk, ahhoz kell még egy sörös port (USB-n a PC serial monitor-jához)
 - Az Arduino Mega ADK 4 sörös portot tud



A kapunyitó összeállítása

- Breadboard segítségével, forrasztás nélkül
 - Breadboard: lyukrendszer szabványos lyuktávolsággal és megadott összekötöttséggel:



A kapunyitó szoftvere

- 3 állapotú állapotgépet valósítunk meg:
 - **start** állapot: kiinduló állapot, várjuk a kártyalehúzást
 - **rfid** állapot: lehúzták a kártyát, az RFID olvasó épp küldi a kártya ID karaktereit
 - **code** állapot: számjegy leütésére vár
- A 12 karakteres kártya ID-t összefűzzük a beírt 4 betűs kóddal
 - Így egy 16 karakteres string-et kapunk
- Ha ez a 16 karakter megegyezik a letároltak valamelyikével, akkor kinyitjuk az ajtót

A kapunyitó szoftvere – globális változók

```

▪ #include <LiquidCrystal.h>
  #include <Key.h>
  #include <Keypad.h>

const int RFIDResetPin = 2;
const int LEDPin = 13;
const int STX = 2;           // „Start of Text”: a soros porton érkező üzenet ezzel kezdődik
const int ETX = 3;          // „End of Text”: a soros porton érkező üzenet ezzel végződik
const int CR = 13;
const int NONE = -1;

char userID[17];             // ide kerül a kártya ID (12 karakter), utánaírva a lenyomott gombok
int index = 0;
enum estate { start, rfid, code }; // állapotgép aktuális állapota
estate state = start;

LiquidCrystal lcd(49, 47, 48, 46, 44, 42); // ide kötöttük a kijelzőt

const byte ROWS = 4;
const byte COLS = 3;

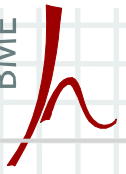
char keys[ROWS][COLS] = { {'1','2','3'}, {'4','5','6'}, {'7','8','9'}, {'*','0','#'} };

byte rowPins[ROWS] = {32, 22, 24, 28}; // ide kötöttük a billentyűzet sor-vezetékeit
byte colPins[COLS] = { 30, 34, 26 };   // ide kötöttük a billentyűzet oszlop-vezetékeit

Keypad keyPad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

// Az elfogadott kártyák és kódjaik. Első 12 karakter: kártya ID, utána írt 4 karakter: a számkód
const char* codes[] = {"010B4CF292261234", "010B4CED58F37899", "010B11C56FB19024", "010B1147E8B41290",
                       "010B11C5F12F7085", "010B112F3F0B0963", "010B11481C4F7412", "010B1148095A3254",
                       "010B1147F3AF6325", "010B114806551589", "010B1147FEA28563", "010B11C56DB33574",
                       "010B4CF0D5637412", "010B4CF26DD96521", "010B4CE9B9164589", NULL};

```



A kapunyitó szoftvere - setup()

- A kötelező **setup ()** függvény:

```
void setup() {
```

```
    Serial.begin(9600);           // 0-ás soros port beállítása 9600 bps-re (debug-hoz)
    Serial1.begin(9600);          // 1-es soros port beállítása 9600 bps-re (RFID-hoz)

    pinMode(RFIDResetPin, OUTPUT); // RFID Reset láb beállítása „kimenet” módba
    digitalWrite (RFIDResetPin, HIGH); // ... majd a Reset jel kiadása
    pinMode(LEDPin, OUTPUT);       // Beépített LED lábának beállítása „kimenet”-re

    lcd.begin(16, 2);              // 16 oszlopos, 2 soros kijelzőnk van
    lcd.print("J\224het a k\240rtya!"); // Írjuk ki rögtön a szöveget
}
```

A kapunyitó szoftvere - loop()

```

void loop () {
  int readByte = NONE;
  char key = keyPad.getKey();
  if (Serial1.available())
    readByte = Serial1.read();
  switch (state) {
    case start:
      if (readByte == STX) {
        state = rfid;
        index = 0;
      }
      break;
    case rfid:
      if (readByte == ETX) {
        userID[index] = 0;
        state = code;
        lcd.clear();
        lcd.print(userID);
        lcd.setCursor(0,1);
        lcd.print("J224het a k242d!");
      }
      else if (readByte != STX && readByte != CR && readByte != ETX && readByte != NONE && index<12) // ha még nincs vége az RFID kód elküldésének
        userID[index++] = readByte; // eltároljuk a soros porton érkezett karaktert
      break;
    case code:
      if (index == 16) {
        userID[index] = 0;
        Serial.println(userID);
        checkCardAndCode();
        state = start;
        lcd.clear ();
        lcd.setCursor (0,0);
        lcd.print("J224het a k240rtya!");
      }
      else if (state == code && key)
        userID[index++] = key;
      break;
  }
}

```

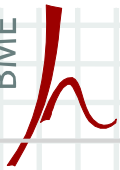
// Megnézzük, volt-e gombnyomás
 // Először feldolgozzuk az RFID olvasó által küldött kártya ID-t
 // Soron következő karakter

// Ha a kiinduló állapotban megjött az „üzenet eleje” karakter (ASCII 0x2),
 // átmegyünk a kártya ID olvasó állapotba (rfid)

// Ha megjött az „üzenet vége” karakter (ASCII 0x3),
 // lezáró 0-ál teszünk a kártya ID végére,
 // átmegyünk a billentyűzetfigyelő állapotba (code),
 // töröljük a kijelzőt,
 // kiírjuk a kártya ID-t a kijelzőre,
 // második sorba megyünk,
 // kiírjuk, hogy jöhet a kód

// Ha kódot olvastunk, és vége (mert megvan a 12 betűs kártya ID és a 4 betűs kód),
 // lezáró 0-át teszünk a végére,
 // kiírjuk a soros terminálra debug célból,
 // ellenőrizzük a helyességét, és végrehajtjuk, amit kell,
 // visszatérünk kiinduló állapotba.
 // Kijelző törlése
 // Kurzor a sarokba
 // Kiírjuk, hogy jöhet a kártya

// Ha jött egy számjegy, de még nem írta be mind a 4-et,
 // hozzáírjuk a többihez



A kapunyitó szoftvere - 5

▪ A kártya és a kód ellenőrzése:

```
void checkCardAndCode () {  
    lcd.clear ();                // kijelző törlése  
    lcd.setCursor (0, 0);        // kurzor a sarokba  
    int ix = 0;  
    while (codes[ix]) {          // végignézzük az összes tárolt kódot  
        if (!strcmp(userID, codes[ix])) { // ha egyezik, kész  
            lcd.print ("Rendben!");      // szólunk, hogy nyílik a kapu  
            digitalWrite (LEDPin, 1);    // kigyújtjuk a LED-et  
            break;                      // nem keresünk tovább  
        }  
        ix++;  
    }  
    if (!codes[ix])              // ha a lista végére értünk, és nincs meg,  
        lcd.print ("Rossz k\u00f6d!");      // közöljük a rossz hírt  
  
    delay (1000);                // várunk 1 másodpercet, hogy el lehessen olvasni  
  
    digitalWrite (LEDPin, 0);     // LED lekapcsolása  
    digitalWrite (RFIDResetPin, LOW); // RFID reset, felfutó élet csinálunk. Alacsonyba visszük...  
    digitalWrite (RFIDResetPin, HIGH); // ... majd magasba emeljük.  
}
```


- Megismertük az Arduino-t
 - Milyen kimenetei/bemenetei vannak
 - Hogyan kell ezekre perifériát illeszteni
 - Hogyan kell a hozzátartozó szoftvert megírni
- Megvalósítottuk az ajtónyitót
 - Bolti, megvásárolható eszközökből
 - Megírtuk a szoftverét