Számítógép Architektúrák

Az ajtónyitó megvalósítása Arduino-val

Dr. Horváth Gábor
Dr. Koller István
BME Hálózati Rendszerek És Szolgáltatások Tanszék

2014. 10. 08.2012.
szeptember 12.
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

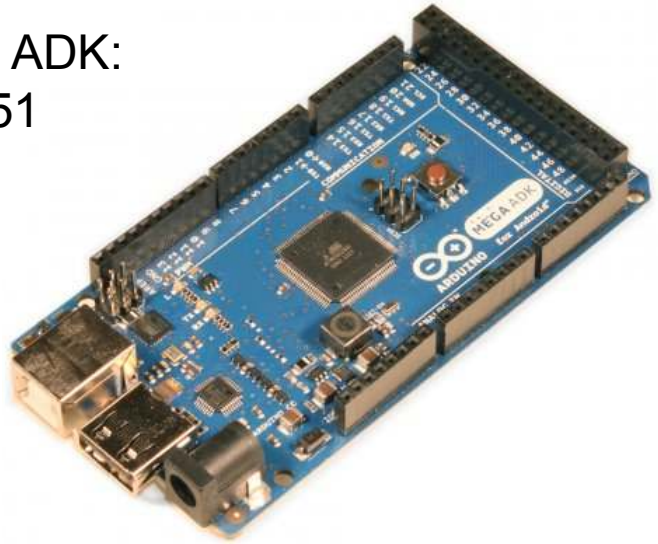
Arduino

- 2005, Ivrea, Olaszország
- Célja: egyszerű prototípusfejlesztés
- Valójában egy teljes család (http://arduino.cc)
- Ami közös bennük:
  - Atmel AVR 8 bites processzor (**Harvard** architektúra!)
    - Beépített flash memória a programnak
    - Beépített EEPROM tartós adattárolásra
    - Beépített RAM
    - Ki/bemenetek:
      - Digitális
      - Analóg
  - USB port-on át programozható (nem mindegyik)
  - C++-szerű nyelven programozható

Az Arduino család

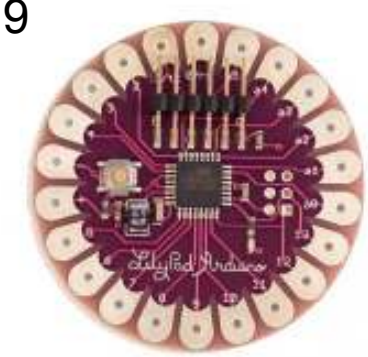
Leonardo:
€18

Mega ADK:
€51



Mini:
€15

LilyPad:
€19



Az Arduino család

- Eltérések:
  - Bemenetek/kimenetek száma
  - Memória mérete (flash/EEPROM/RAM)
  - Néhány jellegzetesség:
    - A Mega ADK-t hozzá lehet kötni az Android eszközökhöz
    - A LilyPad ruhára varrható

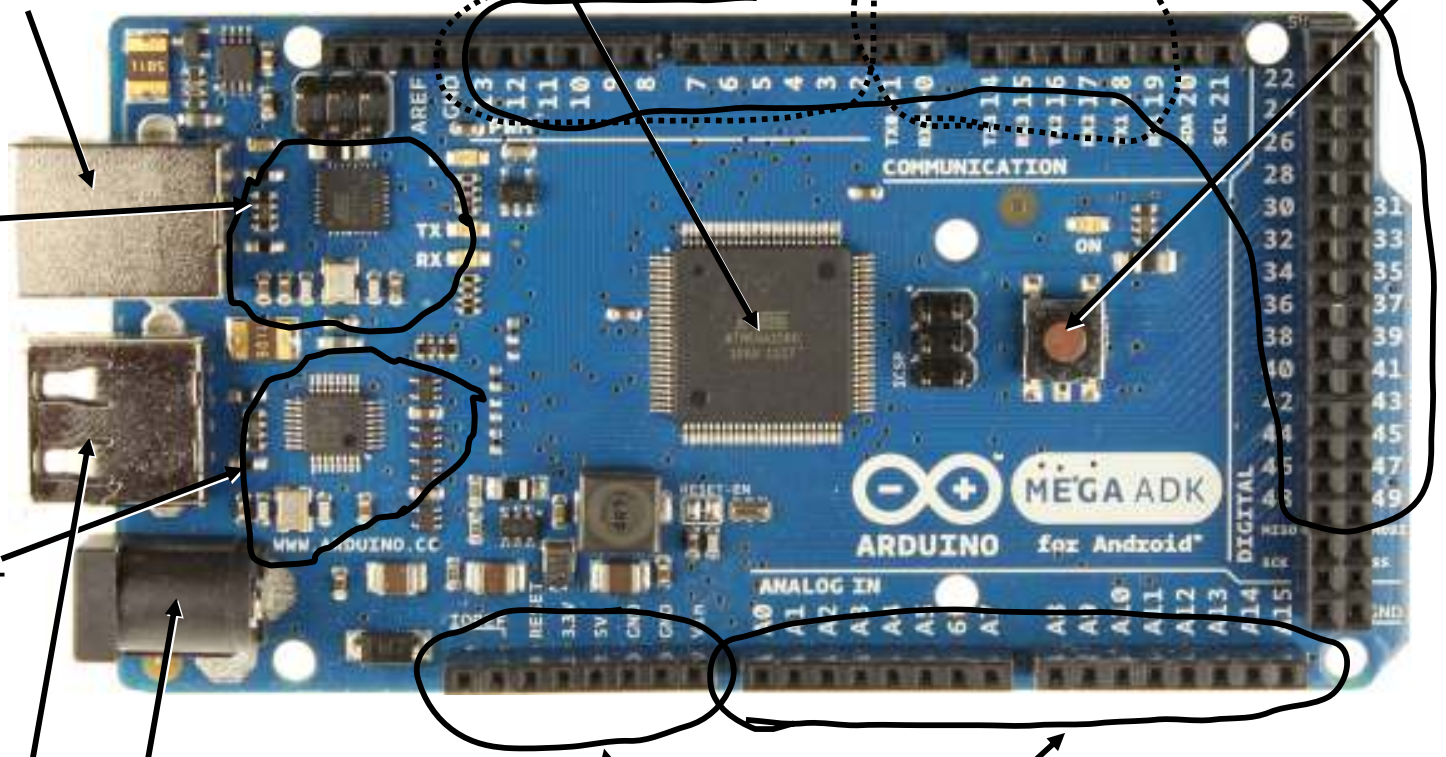
	CPU órajel	Flash	RAM	EEPROM	Digitális I/O	Analóg I/O
Leonardo	16 MHz	32 kB	2.5 kB	1 kB	20	12/7
Mega ADK	16 MHz	256 kB	8 kB	4 kB	54	16/15
Mini	16 MHz	32 kB	2 kB	1 kB	14	8/6
LilyPad	8 MHz	16 kB	1 kB	512 byte	14	6/6

Átverés!

- Micsoda??
  - €51, 8 bites CPU-ért, 16 MHz-ért, pár kB memóriáért?
  - Amikor ott a Raspberry Pi 35$-ért?

    (700 MHz 32 bites ARM, 256 kB RAM, erős GPU, HD filmlejátszás, HDMI kimenet, Ethernet port, stb.)
- Más a célja:
  - Arduino:
    - Hangsúly: I/O, minden mennyiségben, pofonegyszerűen
    - Bekapcsolom, és 1 másodperc múlva már megy is
    - Az €51 a legnagyobb modell, az olcsó Leonardo a legtöbb dologra elég
  - Raspberry Pi:
    - Hangsúly: általános célú (programozás oktatás)
    - Op. rendszer (linux) kell rá! Sokáig tart, mire elindul.
    - Van programozható periféria ki/bemenete, de nehéz használni (érteni kell a linux kernelhez)

Arduino Mega ADK



Mikrokontroller ATMEGA2560

PWM kimenetek

Soros portok

Reset kapcsoló

USB programozáshoz és tápellátáshoz - device

AVR mikro-kontrol-ler USB inter-fész

USB host kontrol-ler csip

Digitális be- és kimenetek

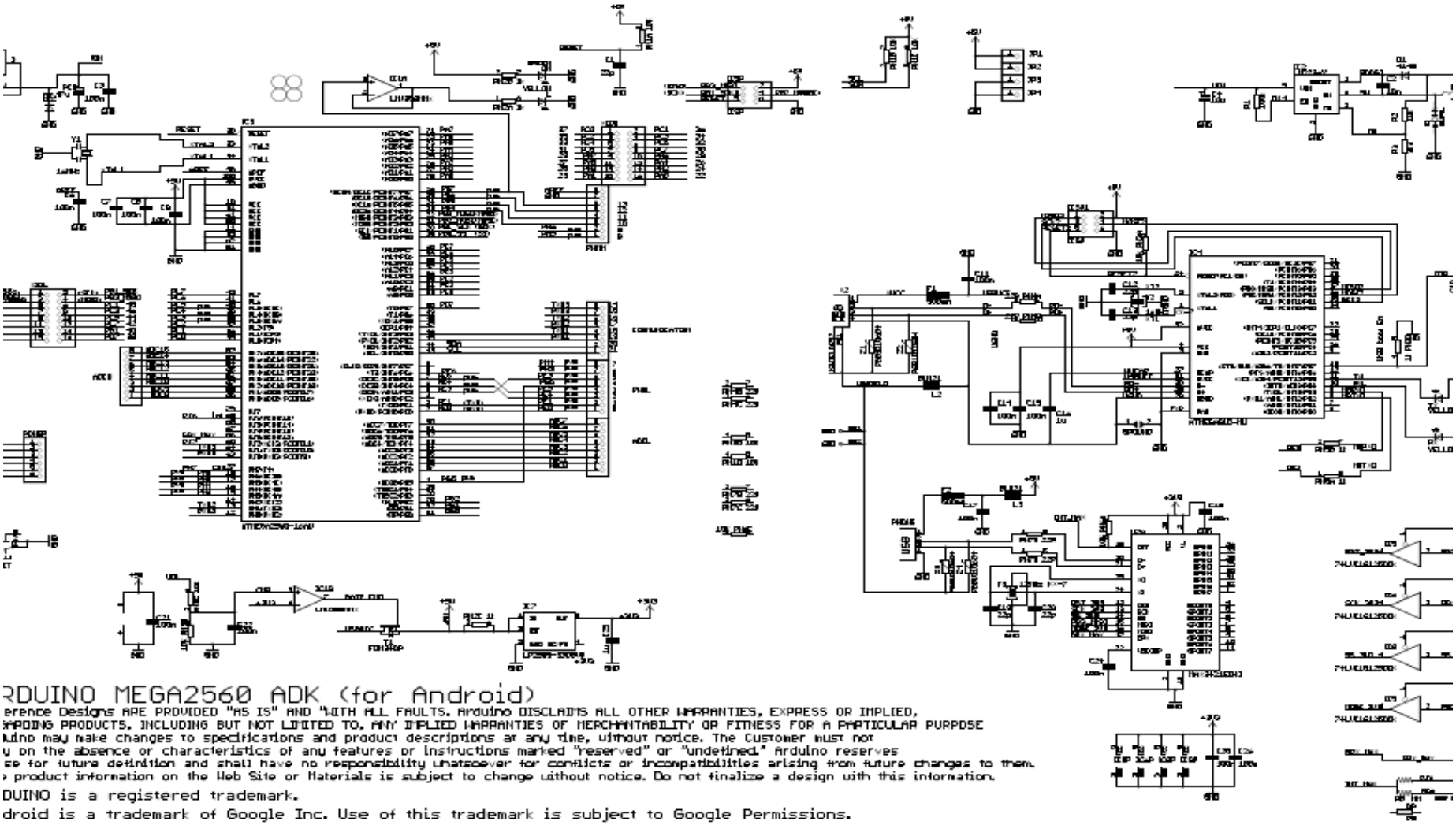
USB Android kapcsolathoz -host

Analóg bemenetek

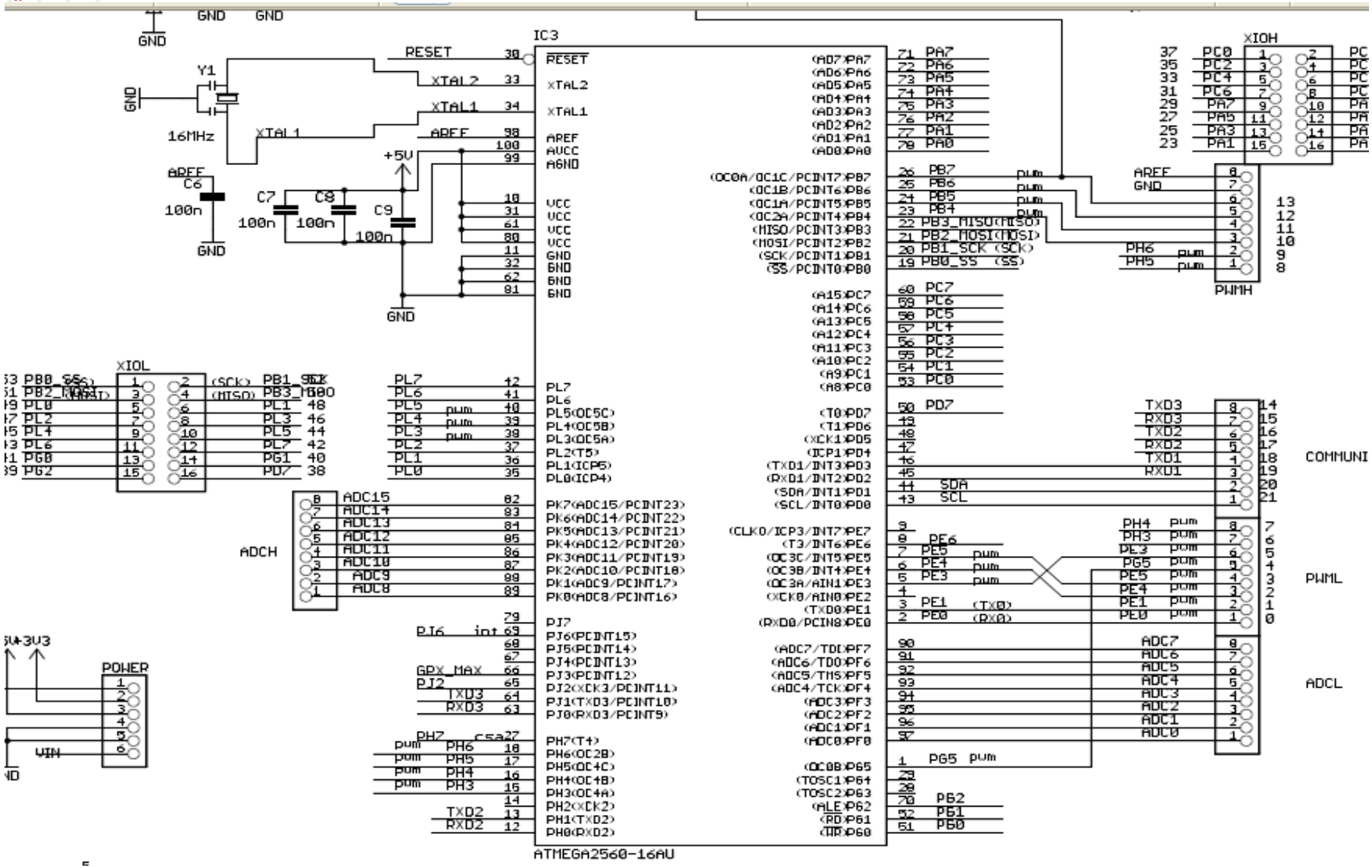
Tápellátás a perifériáknak

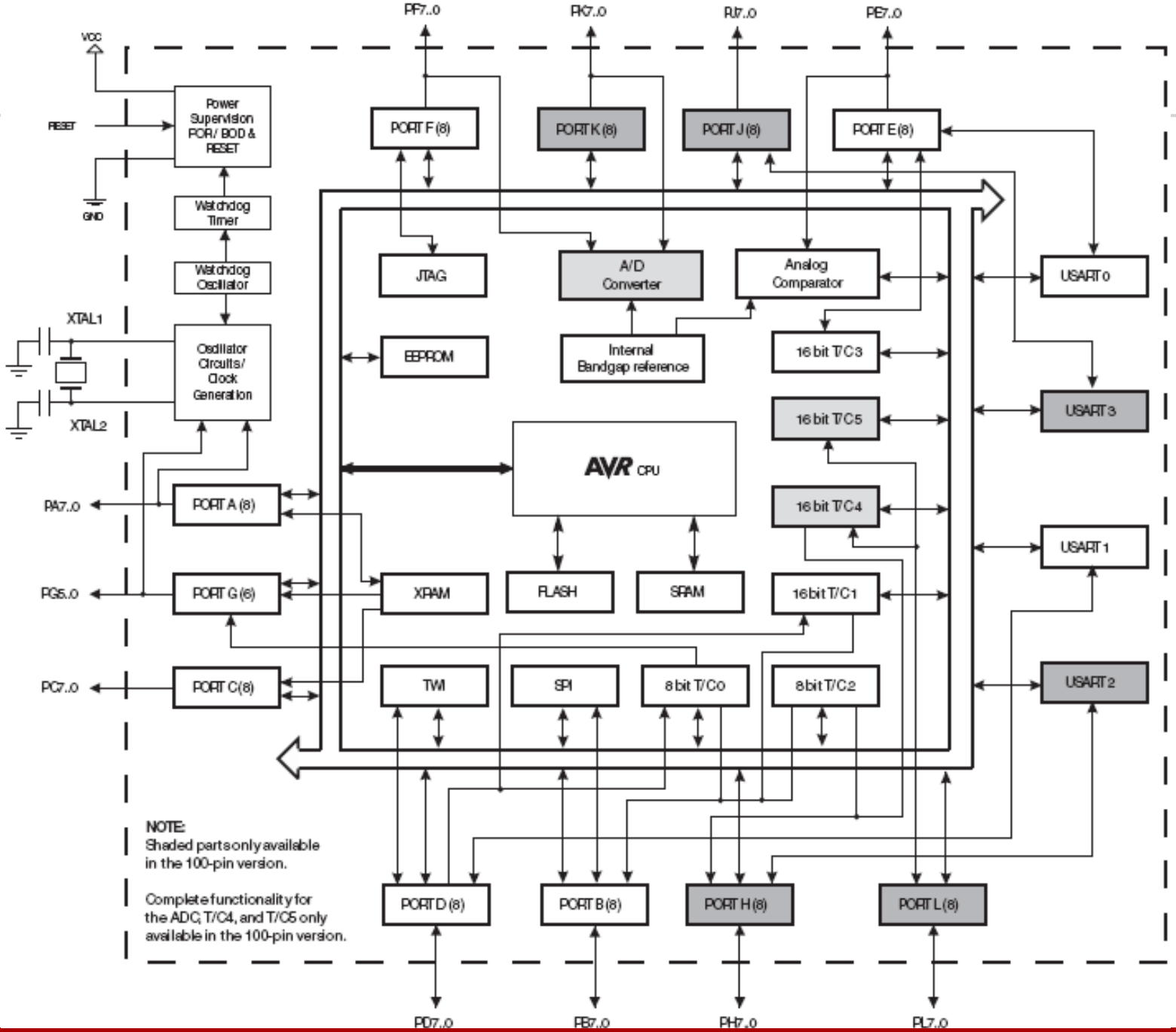
Alternatív tápellátás

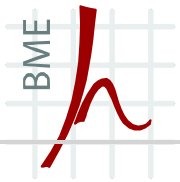
Kapcsolási rajz



Kapcsolási rajz




VCC
RESET
GND
Power Supervision POR/ BOD & RESET
Watchdog Timer
Watchdog Oscillator
XTAL1
XTAL2
Oscillator Circuits/ Clock Generation
PA7..0
PG5..0
PC7..0
PORT A (8)
PORT G (6)
PORT C (8)
PF7..0
PK7..0
PJ7..0
PE7..0
PORT F (8)
PORT K (8)
PORT J (8)
PORT E (8)
JTAG
A/D Converter
Analog Comparator
EEPROM
Internal Bandgap reference
AVR CPU
XRAM
FLASH
SRAM
TWI
SPI
8 bit T/C0
16 bit T/C3
16 bit T/C5
16 bit T/C4
16 bit T/C1
8 bit T/C2
USART 0
USART 3
USART 1
USART 2
PORT D (8)
PORT B (8)
PORT H (8)
PORT L (8)
PD7..0
PB7..0
PH7..0
PL7..0
NOTE:
Shaded parts only available in the 100-pin version.
Complete functionality for the ADC, T/C4, and T/C5 only available in the 100-pin version.



Programozás

Programozás

- Open-source cross-platform fejlesztői környezet
- Programnyelv: C++-hoz hasonló (kiterjesztés: .ino)
- Fordítás:
  - A fejlesztőkörnyezet AVR kódot fordít (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

A fejlesztőkörnyezet

Szintaktikai ellenőrzés

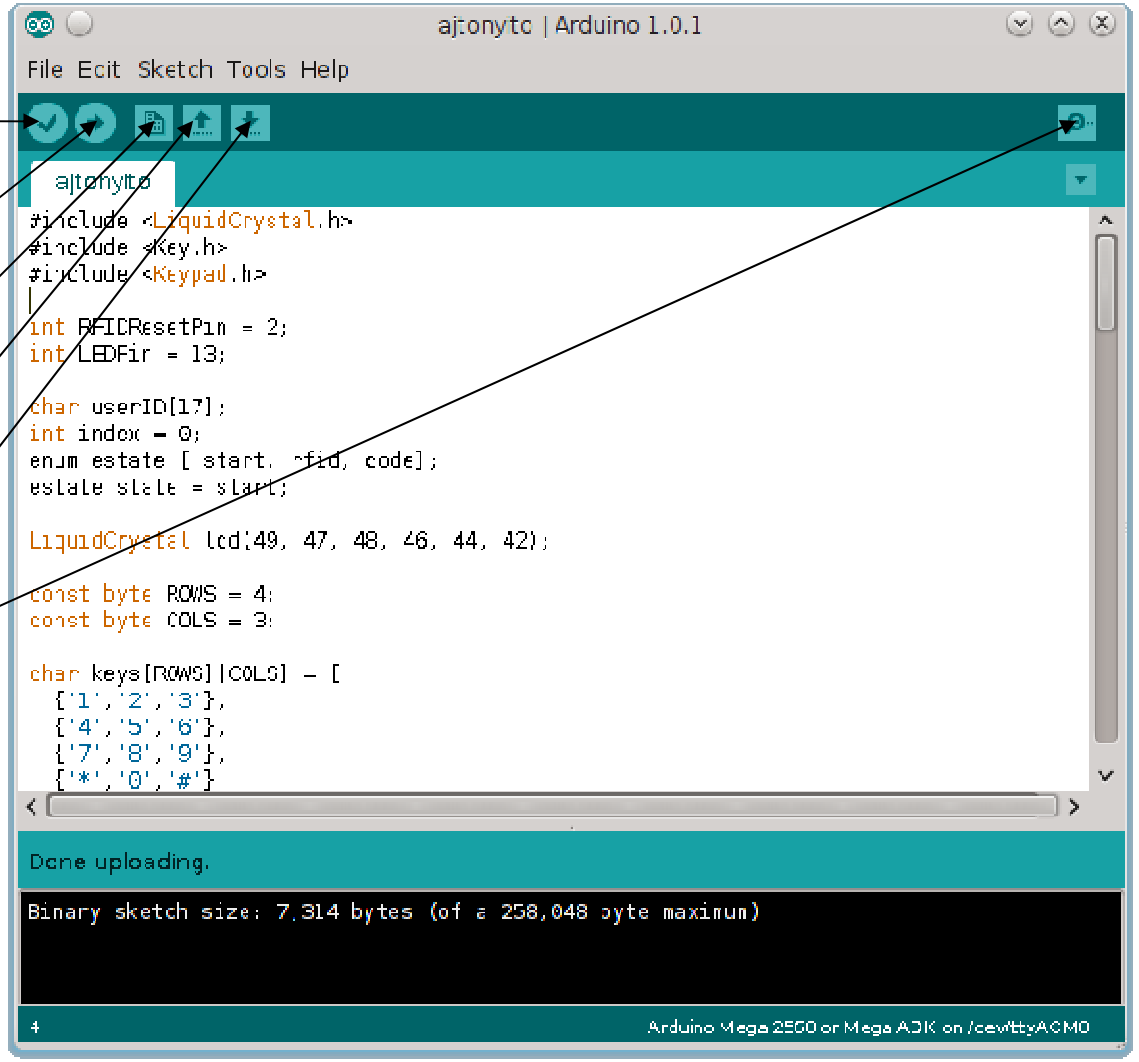
Feltöltés az Arduino-ra

Új fájl

Fájl megnyitás

Fájl mentés

Soros port monitor



A programozási nyelv

- Az Arduino a programot „sketch”-nek hívja
- Adattípusok:
  - int: 16 bites egész
  - long: 32 bites egész
  - boolean: logikai típus (true/false), 1 bitet foglal (ilyen C++-ban nincs)
  - float: 32 bites lebegőpontos
  - char: ASCII karakter tárolására szolgál (1 byte)
  - stb.
- Operátorok, ciklusok, elágazások: mint a C++
- class-ok használata megengedett, preprocessor direktívákat elfogad
- Két függvény van, amit kötelező megírni:
  - **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)

Ki/Bemenetek

Digitális ki- és bemenetek

- A digitális lábak ki és bemenetként is szolgálhatnak egyaránt
  - De egyszerre csak az egyik szerepet tölthetik be
  - Beállítás:
    - **pinMode (4, INPUT);** – a 4-es lábat bemenetre állítja
    - **pinMode (5, OUTPUT);** – az 5-ös lábat kimenetre állítja
- 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) …**

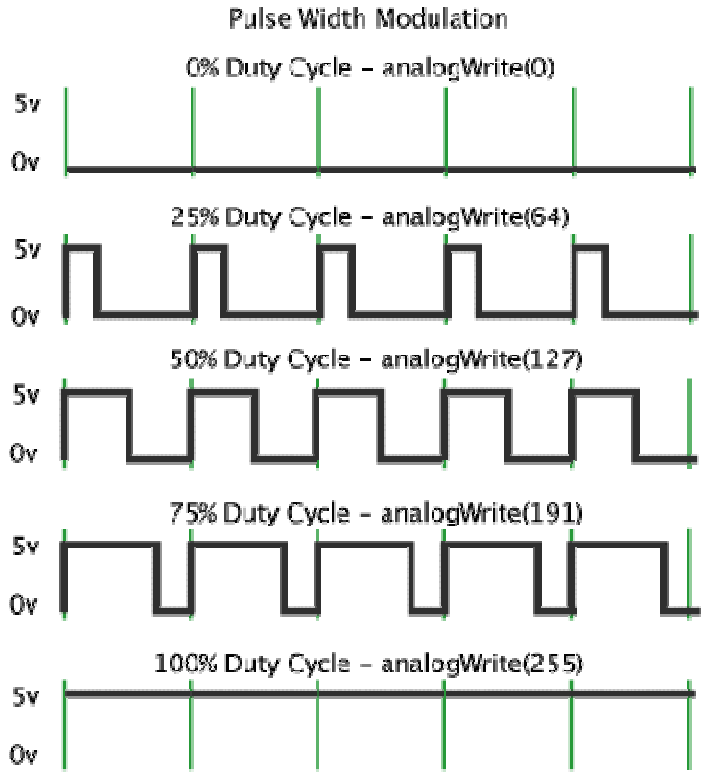
Analóg bemenetek

- Nemcsak HIGH és LOW értékeket lehet róla olvasni
- ADC (Analog-Digital Converter) 10 bites felbontással
  → 1024 különböző feszültségszintet 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()** függvénnyel, 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 kaptam=Serial1.read(); - egy megérkezett bájt kiolvasása (-1, ha nincs)
  - *Érkezett-e adat*: int mennyi = 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 Interrupt-ot lekezeli az Arduino:
    - Beleteszi az érkezett bájtot egy tárolóba (buffer)
  - A read() tulajdonképpen a tárolóból olvas
  - Az available() a tárolóban lévő bájtok számát adja vissza

Memóriák

Memóriák

- Az Arduino AVR processzora az alábbi memóriákkal rendelkezik:
  - **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 flash és a RAM

- Flash memória:
  - A fejlesztőkörnyezet tölti meg
  - A programból nem írható. De olvasható! Speciális függvényekkel.
- RAM:
  - Kevés van belőle, ezért spórolni kell
  - Nem változó tömböket inkább a flash-be írjuk:

```
int kodok[] = {21342, 2442, -34887, ...};
   char szoveg[] = „megszentségteleníthetetlenségeskedéseitekért”;
   k = kodok[4];
   c = szoveg[7];
```

  - Helyett:

```
PROGMEM prog_uint16_t kodok[] = {21342, 2442, 34887, ...};
   PROGMEM prog_uchar szoveg[]  = „megszentségteleníthetetl...”;
   k = pgm_read_word_near (kodok + 4);
   c = pgm_read_byte_near (szoveg + 7);
```

EEPROM

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

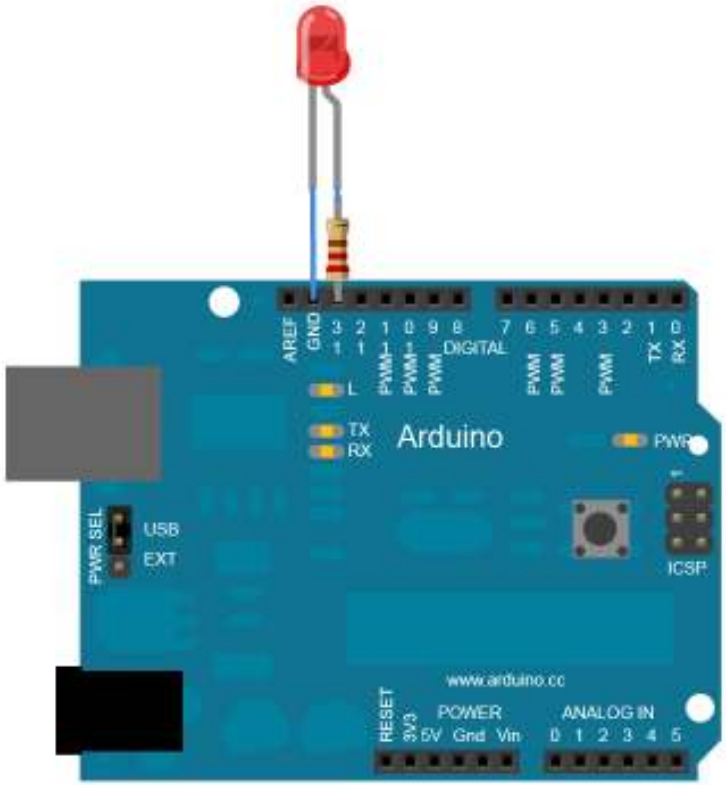
Perifériák illesztése

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.

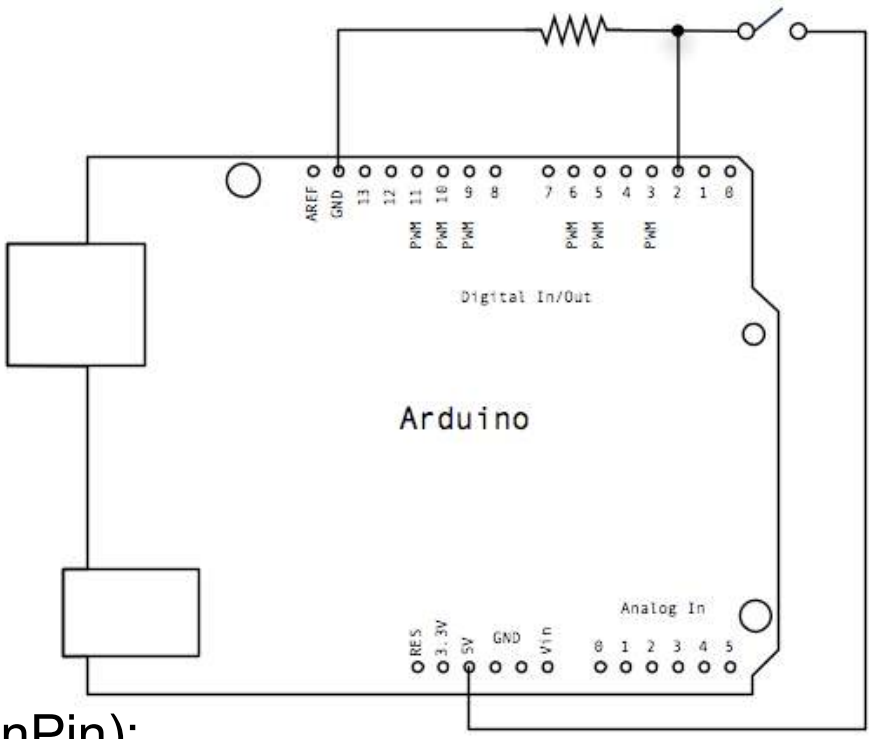


- **Kód**:

```
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

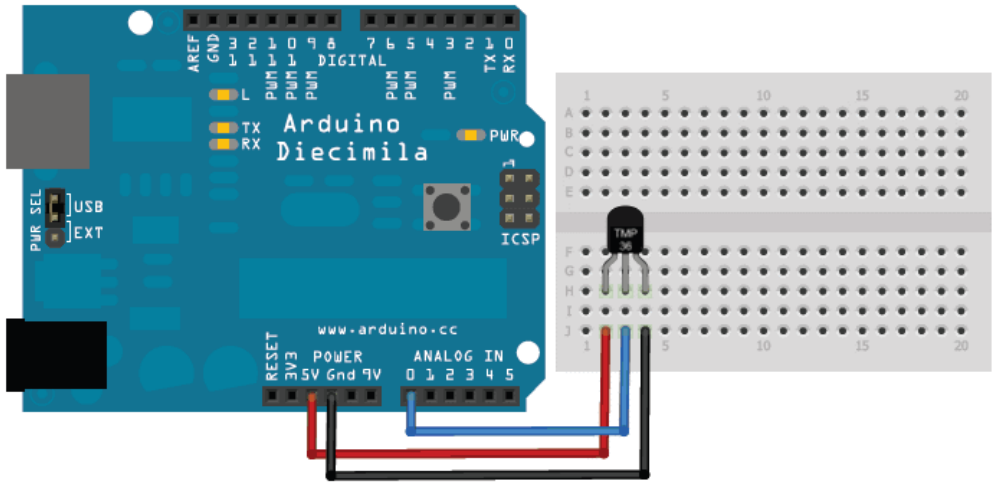


- **Kód**:

```
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**:

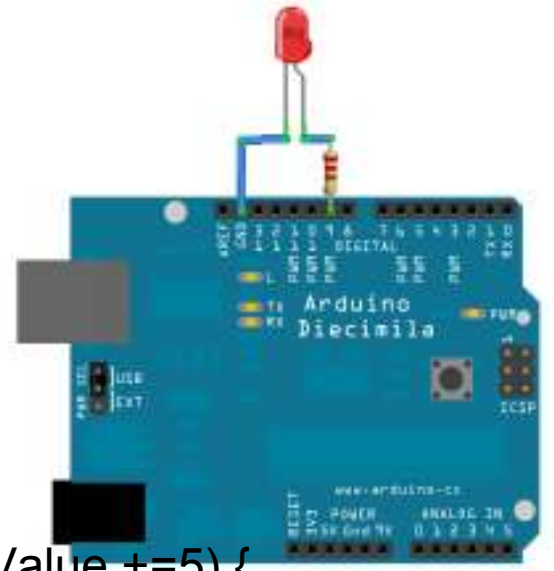
```
const int sensorPin = 0;
void setup () {
    Serial.begin (9600);
}
void loop () {
    int reading = analogRead (sensorPin);
float voltage = reading * 5.0 / 1024.0;
    float temperature = (voltage - 0.5) * 100;
    Serial.println (temperature);
    delay(1000);
}
```

Analóg bemeneti perifériák

- A legérdekesebb perifériák az analóg bemeneti 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 fénnyel vagy ultrahanggal)
    - Hőmérő
    - Páratartalommérő
    - 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 átviendő adatok is a soros porton mennek
  - Stb.

Pajzsok (Shield)

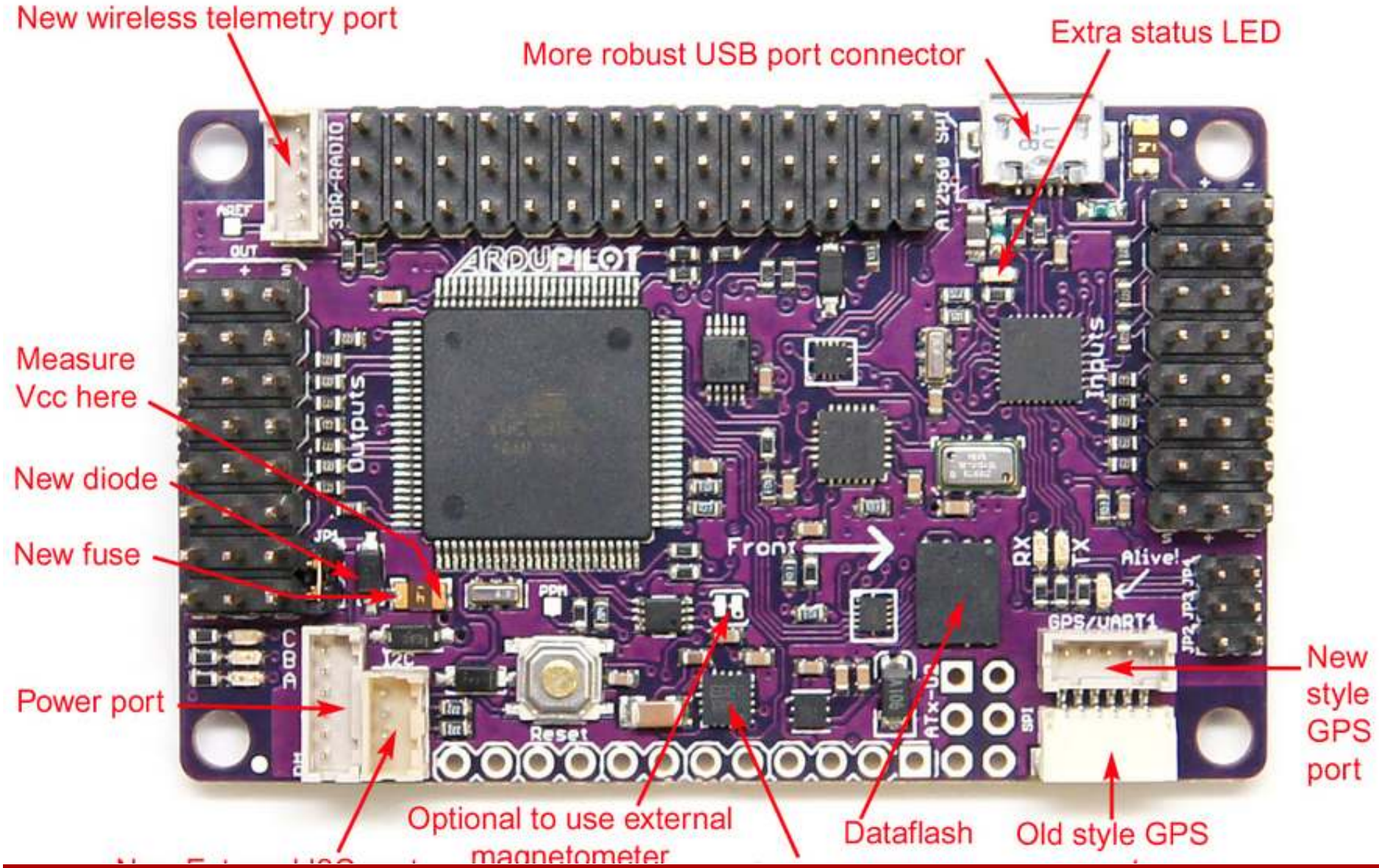
- Az Arduino tetejére illeszthető komplett perifériák
  - Nagyon sok van:
    - GPS, LCD vezérlő, SD kártya kezelő, WIFI, Bluetooth, ZigBee, GSM, ...

GPS Shield
€18

MP3 Shield
€25

2.8” TFT és érintőképernyő
€40

Arduino -> ArduPilot APM 2.5



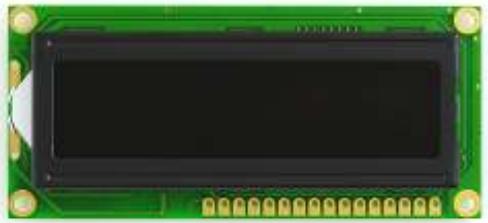
Az ajtónyitó megvalósítása

Ajtónyitó

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

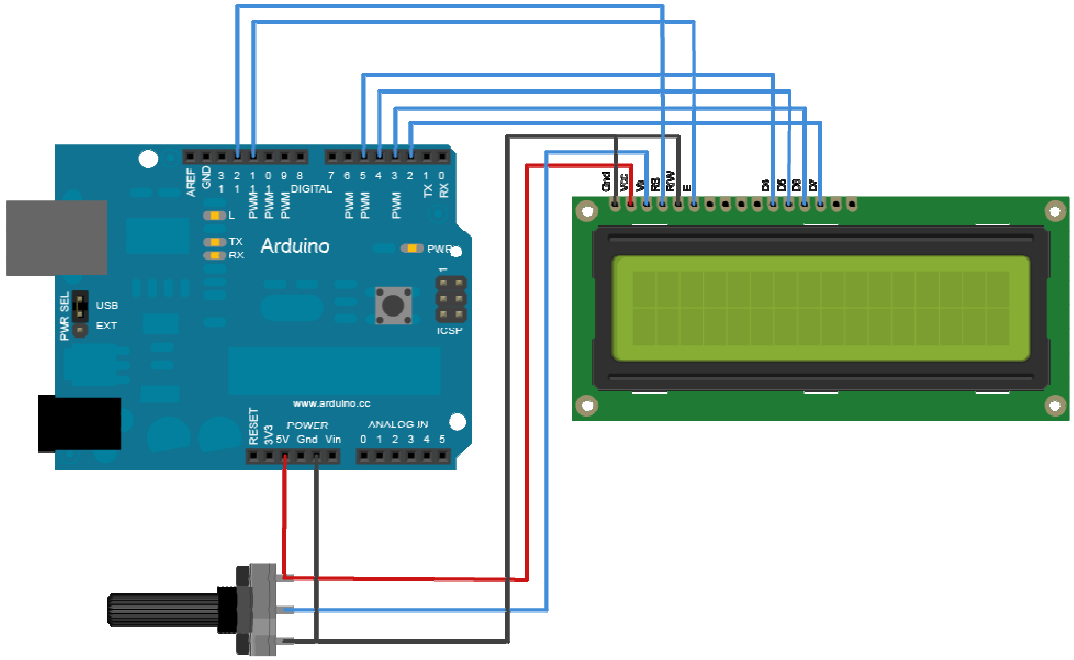
A kijelző

- Olcsón vehetünk egy 2 soros, 16 oszlopos kijelzőt (2000 Ft)



- Lábak:
  - *Adatbusz*: D0...D7, de 4 bites üzemmódban is megy, nekünk az is elég: D4...D7-et kötjük be
  - *RS*: ezzel lehet jelezni a kijelzőnek, hogy épp 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 kérdezünk le információt – ezt nem kötjük be, csak írunk
  - *Tápfeszültség* (5V)
  - *Kijelző kontraszt* (potméter)
  - *Háttérvilágítás tápellátása*
- 6 vezetéket használunk: RS, EN, D7, D6, D5, D4

A kijelző használata-1



- **Bekötés**:
  Google „arduino display”

- **Használata**:
  LiquidCrystal class példányosítása
  (write(), setCursor(), blink(), clear(), stb.)

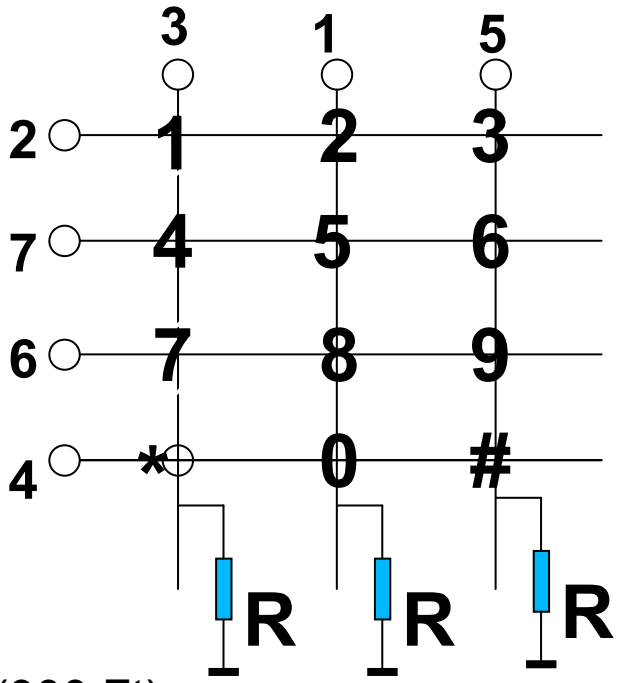
A kijelző használata-2

Példa:

```
#include <LiquidCrystal.h>
    const int numRows = 2;
    const int numCols = 16;

    LiquidCrystal lcd(49, 47, 48, 46, 44, 42); // beállítjuk, melyik lábakra
    kötöttük
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



- Könnyen 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-1

- Google „arduino keypad”
- **Használata**:
  Keypad class példányosítása
  (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 );
```

A numerikus billentyűzet használata-2

```
#define ledpin 13
    void setup () {
        digitalWrite (ledpin, HIGH);
    }
    void loop () {
        char key = keyPad.getKey();
        if(key)  {
            switch (key)  {
                case '*':
                    digitalWrite(ledpin, LOW);
                    break;
                case '#':
                    digitalWrite(ledpin, HIGH);
                    break;
            }
        }
    }
```

Az RFID olvasó

- Rádiofrekvenciás azonosítás
- Minden kártyának garantáltan egyedi, 12 hexa karakteres kódja van
- Leolvasó: drágát vettünk (7000 Ft), mert pont csak azt lehetett kapni
- 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)

125kHz RFID működése

Az RFID olvasó használata-1

- Google „arduino id-12”
- **Használata**:
  soros eszközként

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

```
const int RFIDResetPin = 2;                 // A 2-es digitális lábra kötöttük a leolvasó
                                            //reset lábát

char userID[13];                            // Ebbe a tömbbe olvassuk majd be a
                                            //kártya azonosítóját

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-t ide kötöttük)
   pinMode(RFIDResetPin, OUTPUT);           // RFID Reset láb beállítása „kimenet”
                                            //módba
   digitalWrite(RFIDResetPin, LOW);         // RFID reset, felfutó élet csinálunk.
                                            //Alacsonyba visszük...
   delay (100);                             // … kicsit várunk ...
   digitalWrite(RFIDResetPin, HIGH);        // … majd magasba emeljük.
}
```

Az RFID olvasó használata-2

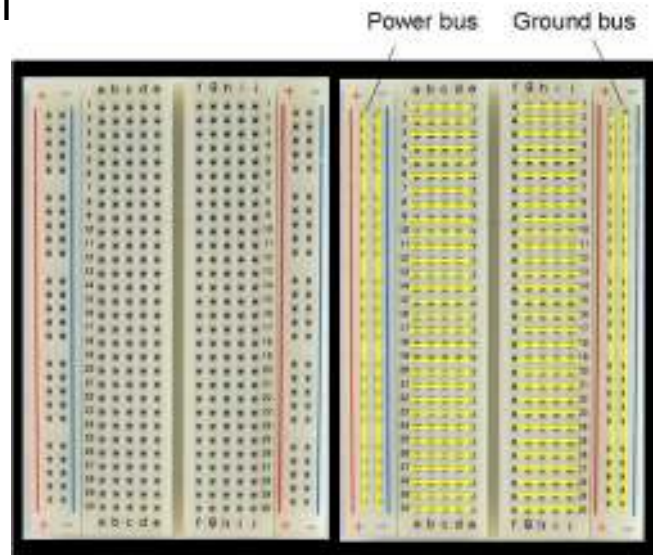
```
void loop () {
   while (Serial1.available()) {          // Ha jött byte az RFID soros
                                          // portjától, feldolgozzuk.
      int readByte = Serial1.read();      // Kiolvassuk a következő byte-ot
      if (readByte == 2)                  // Az ASCII 0x2 az „üzenet eleje”
                                          // karakter
         index = 0;                       // Visszapörgetjünk az indexet, a
                                          // userID-t az elejétől töltögetjük.
      else if (readByte == 3) {           // Az ASCII 0x3 az „üzenet vége”
                                          // karakter
         userID[index] = 0;               // Lezárjuk, NULL terminált
                                          // karakterlánccá tesszük.
         Serial.println(userID);          // Elküldjük a debug rendszernek.
                                          // Ki fogja írni a PC-n.
      }
      else
         userID[index++] = readByte;      // 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
    - A Leonardo 20-at kezel, de a beszerzéskor még nem létezett. Az elődje 14-et tudott, ezért az ADK-t választottuk. (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 soros port (USB-n a PC serial monitor-jához)
    - A Leonardo pont 2 soros portot tud (az elődje csak 1-et tudott)
- Ideális választás: **Leonardo**

A kapunyitó összeállítása

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



- De nem úsztuk meg teljesen a forrasztást:
  - A kijelzőnek és a billentyűzetnek nem volt tüskesora
    - Tüskesort vettünk és ráforrasztottuk
  - Az RFID tüskesora nem volt kompatibilis a breadboard-dal
    - Átalakítót vettünk, és ráforrasztottuk
- *Forrasztás megtanulható pár youtube tutorial-ból!*

A kapunyitó szoftvere - 1

- 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 - 2

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

    const int RFIDResetPin = 2;
    const int LEDPin = 13;

    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
```

A kapunyitó szoftvere - 3

```
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

char* codes[] = {   "010B4CF292261234", "010B4CED58F37899", "010B11C56FB19024",
                    "010B1147E8B41290", "010B11C5F12F7085",  "010B112F3F0B0963",
                    "010B11481C4F7412", "010B1148095A3254",   "010B1147F3AF6325",
                    "010B114806551589", "010B1147FEA28563", "010B11C56DB33574",
                    "010B4CF0D5637412","010B4CF26DD96521", "010B4CE9B9164589",
                    NULL};
```

A kapunyitó szoftvere - 4

- **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, LOW);// RFID reset, felfutó élet csinálunk.
                                    // Alacsonyba visszük...
   delay (100);                     // … kicsit várunk ...
   digitalWrite (RFIDResetPin, HIGH);// … majd magasba emeljük.

   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 - 5

- A kötelező **loop ()** függvény:

```
void loop () {
  while (Serial1.available()) {                  // Először feldolgozzuk az RFID olvasó által
                                                 // küldött kártya ID-t
    int readByte = Serial1.read();               // Soron következő karakter
    if (state == start && readByte == 2) {       // Ha a kiinduló állapotban megjött az „üzenet
                                                 // eleje” karakter (ASCII 0x2),
      state = rfid;                              // átmegyünk a kártya ID olvasó állapotba (rfid)
      index = 0;
     }
    else if (state == rfid && readByte == 3) {   // Ha megjött az „üzenet vége” karakter (ASCII
                                                 // 0x3),
      state = code;                              // átmegyünk a billentyűzetfigyelő állapotba (code)
      lcd.clear();                                       // Töröljük a kijelzőt,
      userID[index] = 0;                         // lezáró 0-ál teszünk a kártya ID végére,
      lcd.print(userID);                         // kiírjuk a kártya ID-t a kijelzőre
      lcd.setCursor (0,1);                       // Második sorba megyünk
      lcd.print("J\224het a k\242d!");            // Kiírjuk, hogy jöhet a kód
    }
    if (state==rfid && readByte != 2 && readByte != 10 && readByte != 13 &&
                index<12)
                // ha még nincs vége az RFID kód elküldésének
      userID[index++] = readByte;
                                                 // eltároljuk a soros porton érkezett karaktert
  }
```

A kapunyitó szoftvere - 6

```
  char key = keyPad.getKey();              // Megnézzük, volt-e gombnyomás
 if (state == code && index == 16) {       // Ha kódot olvastunk, és vége
                                           // (mert megvan a 12 betűs kártya ID és a
                                           // 4 betűs kód)
    userID[index] = 0;                     // Lezáró 0-át teszünk a végére
    Serial.println(userID);                // Kiírjuk a soros terminálra debug célból
    checkCardAndCode();                    // Ellenőrizzük a helyességét, és
                                           // végrehajtjuk, amit kell
    state = start;                         // Visszatérünk kiinduló állapotba
    lcd.clear ();                          // Kijelző törlése
    lcd.setCursor (0,0);                   // Kurzor a sarokba
    lcd.print("J\224het a k\240rtya!");    // Kiírjuk, hogy jöhet a kártya
 }
 else if (state == code && key)            // Ha jött egy számjegy, de még nem írta
                                           // be mind a 4-et
    userID[index++] = key;                 // Hozzáírjuk a többihez
}
```

A kapunyitó szoftvere - 7

- 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\242d!");        // 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...
   delay (100);                           // … kicsit várunk ...
   digitalWrite (RFIDResetPin, HIGH);     // … majd magasba emeljük.
}
```

Összegzés

- 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
  - Forrasztás csak az összekötögetések miatt kellett
  - Megírtuk a szoftverét

Kitekintés

- Ezeregy Arduino klón létezik
  - Legálisan, mert open source a hardver is
- Ezeregy Arduino-szerű mikrokontroller kártya kapható
  - OLIMEXINO: AVR mikrokontrollerrel
  - DUINOMITE: PIC mikrokontrollerrel
  - MAPLE: ARM Cortex M3 processzorral
  - NETDUINO: 32 bites ARM processzorral, .NET környezetben programozható (Visual Studio-val)
  - stb...