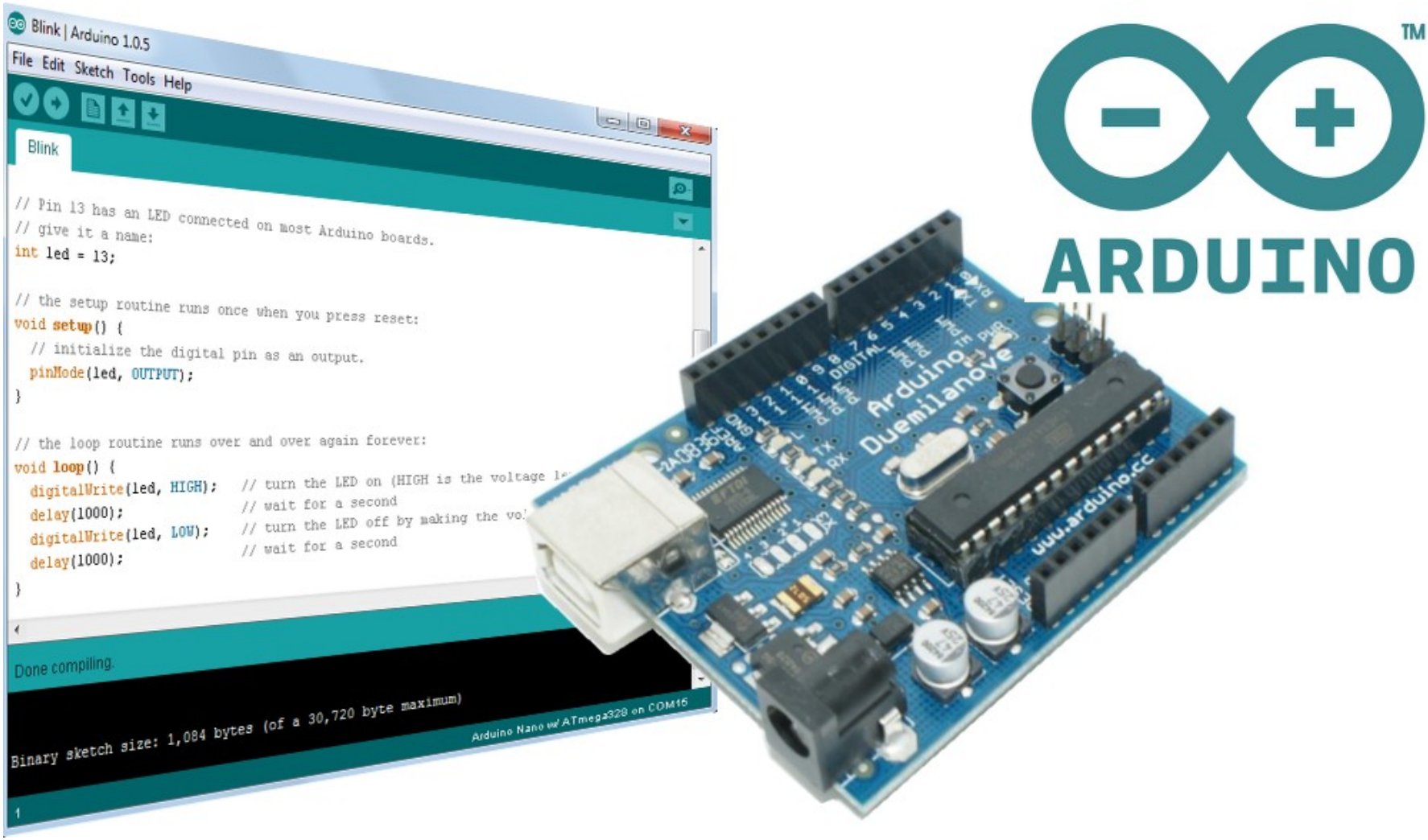
Bevezetés az elektronikába



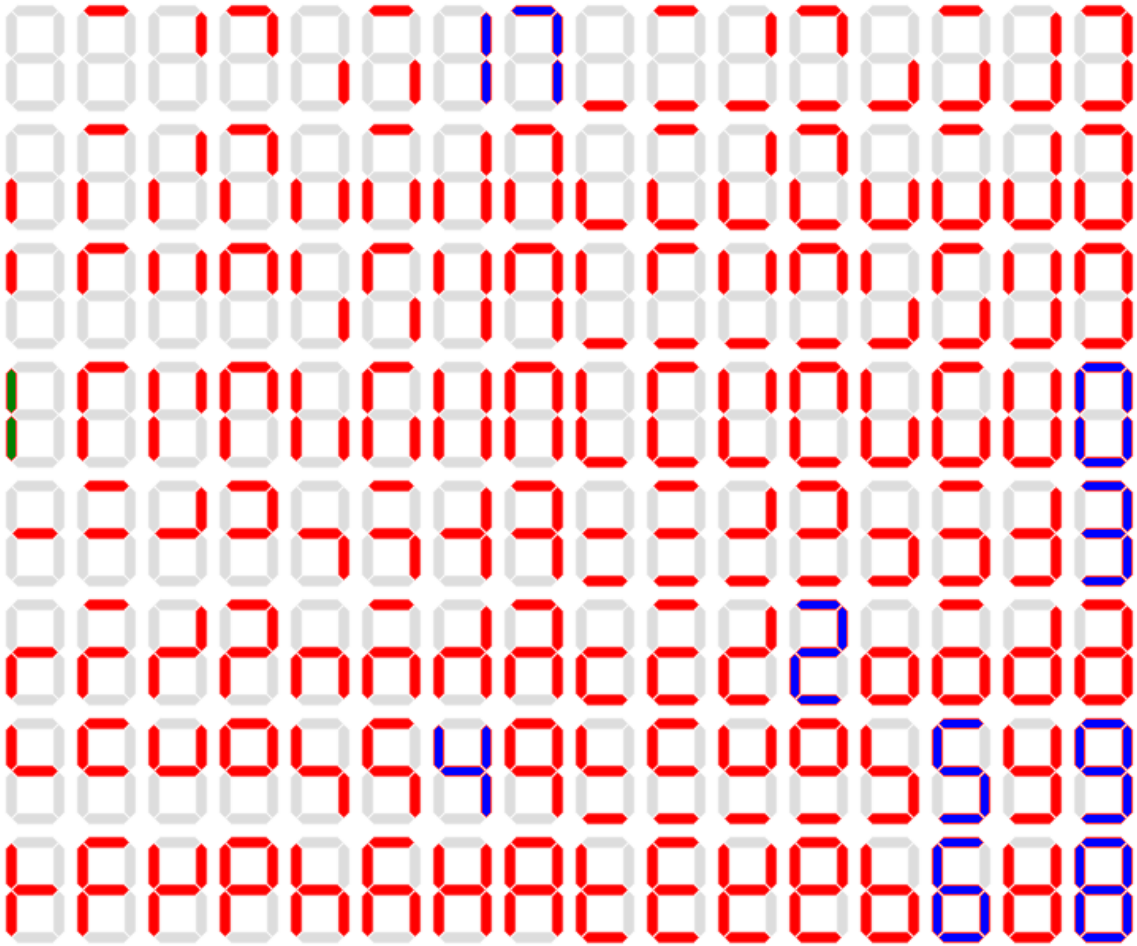
15. Arduino programozás

Hétszegmenses kijelzők – 2. rész

Betűvadászat

- A 7 db szegmens mindegyike lehet ki- vagy bekapcsolt állapotban.

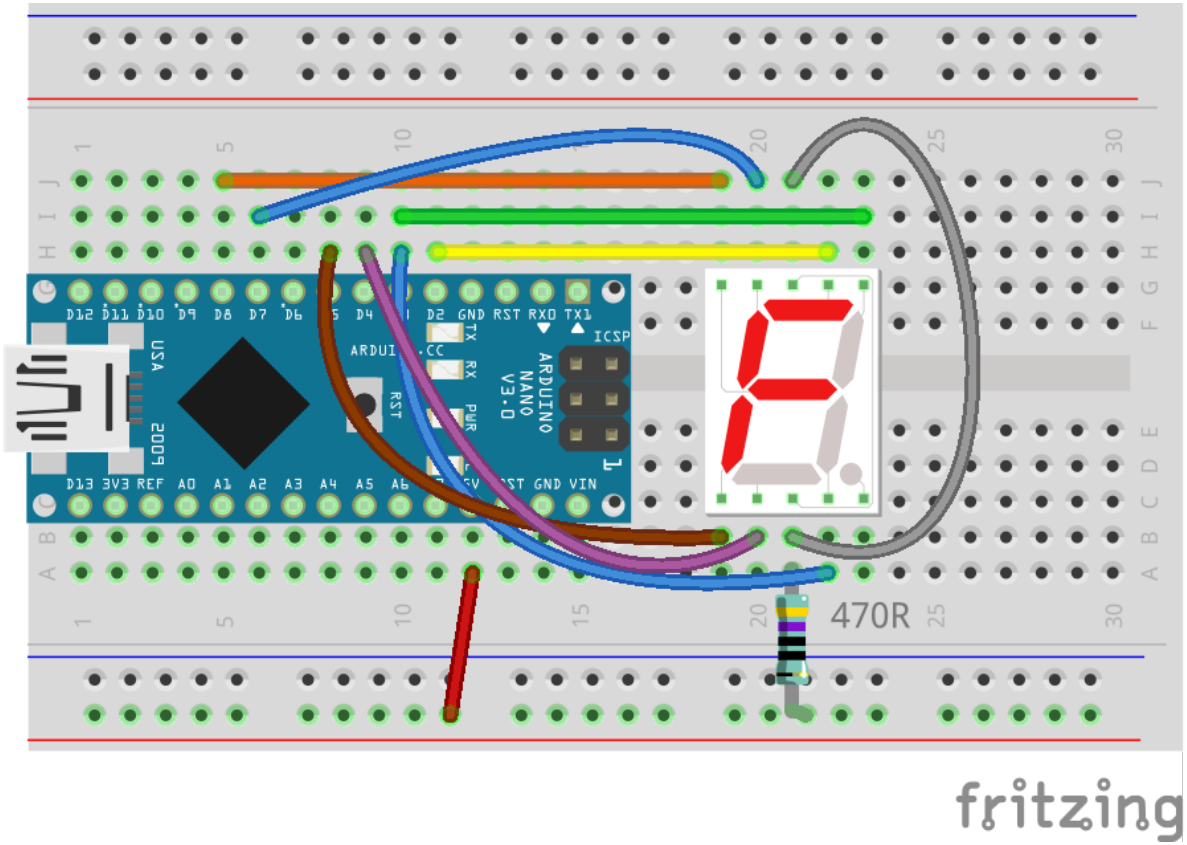
- A lehetséges állapotok száma: $2^7 = 128$
- Ezek közül nem mindegyik értelmes kombináció.



- **Vadásszunk betűkre!**
  A mellékelt ábrából keressünk olyan kombinációkat, amelyek kis- vagy nagybetűt mutatnak!

Időosztásos kijelzés „lusta kapcsolásal”

- A „lusta kapcsolás” itt abban áll, hogy csak a közös ágban, a közös anódnál használunk áramkorlátozó ellenállást
- A szegmenseket egymás után villantjuk fel egy-egy rövid időre
- **Az időosztásos kijelzés hátrányai:**
  - A fényerő kisebb lesz
  - A kijelzés folyamatosan lefoglalja a mikrovezérlőt
  - Bonyolultabb a program



LED_7seg_multiplex.ino

```
//számjegyek rajzolata, sorrend: a b c d e f g DP
const byte digit[10] = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE0, 0xFE, 0xF6};
#define commonAnode true
byte myPins[7] = {2, 3, 4, 5, 6, 7, 8};  // Bekötések definiálása

void displayNumber(byte n, int cycles) { // Számjegy kiírása (n = 0..9)
  byte data;
  for (int cy = 0; cy < cycles; cy++) {
    data = digit[n];
    if ( commonAnode ) data = ~data;
    for (int i = 0; i < 7; i++) {
      digitalWrite(myPins[i], bitRead(data, 7 - i));
      delay(1);
      if ( commonAnode ) digitalWrite(myPins[i], HIGH);
      else digitalWrite(myPins[i], LOW);
    }
  }
}
void setup() {                              // Kimenetek beállítása
  for (int i = 0; i < 7; i++) pinMode(myPins[i], OUTPUT);
}

void loop() {                               // Ciklikusan ismétlődő rész
  for (int i = 0; i < 10; i++) {            // Számlálás 0-tól 9-ig
    displayNumber(i,140);                   // Kiíratjuk az aktuális számot kb. 1 mp-ig
  }
}
```

Felvillantjuk a szegmenst – ha aktív, majd egy idő múlva lekapcsoljuk

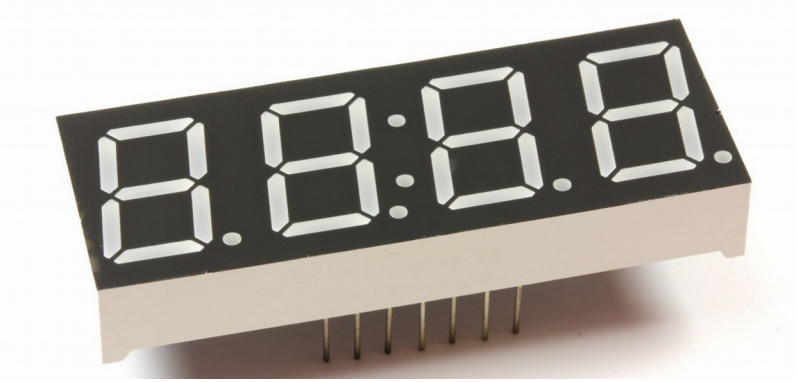
LED_7seg_effects.ino

- Egyszerű fényeffektus: a fény 8-as alakban szaladgál. Az effect tömb szabja meg, hogy mikor melyik szegmens gyulladjon ki

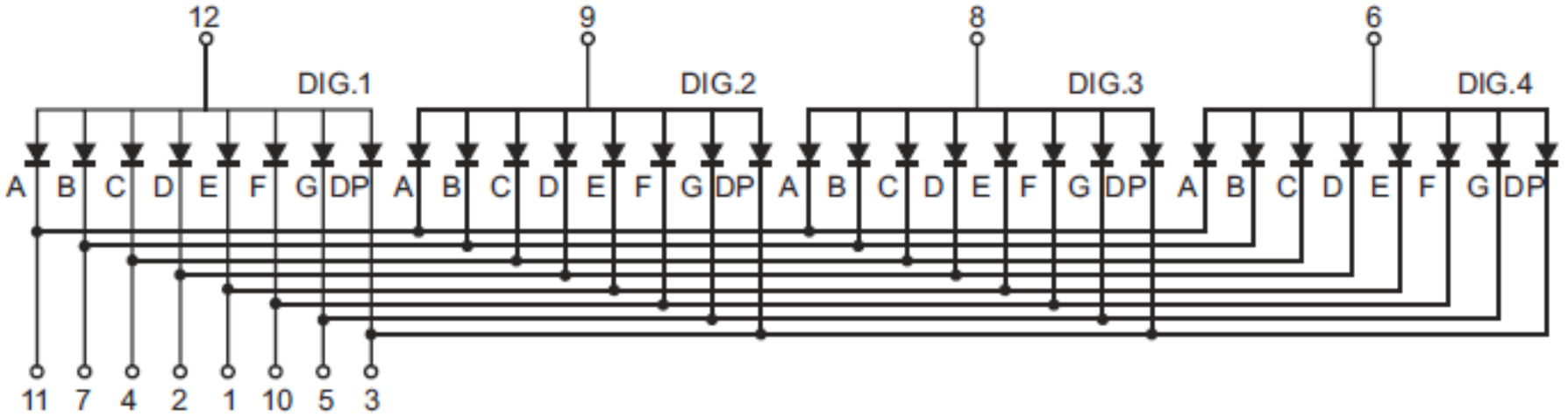
```
const byte effect[] = {0,1,6,4,3,2,6,5}; // Nyolcas alak
#define commonAnode true
byte myPins[7] = {2, 3, 4, 5, 6, 7, 8};  // Bekötések: a b c d e f g

void setup() {                            // Kimenetek beállítása
  for (int i = 0; i < 7; i++)
    pinMode(myPins[i], OUTPUT);
}

void loop() {                           // Ciklikusan ismétlődő rész
  for (int i = 0; i < 8; i++) {         // Végigmegy az effektusokon
    byte p = effect[i];                 // Melyik láb legyen aktív?
    digitalWrite(myPins[p],!commonAnode);  // Szegmens kigyújtása
    delay(100);                            // 100 ms várakozás
    digitalWrite(myPins[p],commonAnode);   // Szegmens leoltása
  }
}
```

Többszámjegyű kijelők

- A többszámjegyű kijelzőknél a szegmenskivezetések össze vannak kötve (egyszerre csak egy számjegy jeleníthető meg)

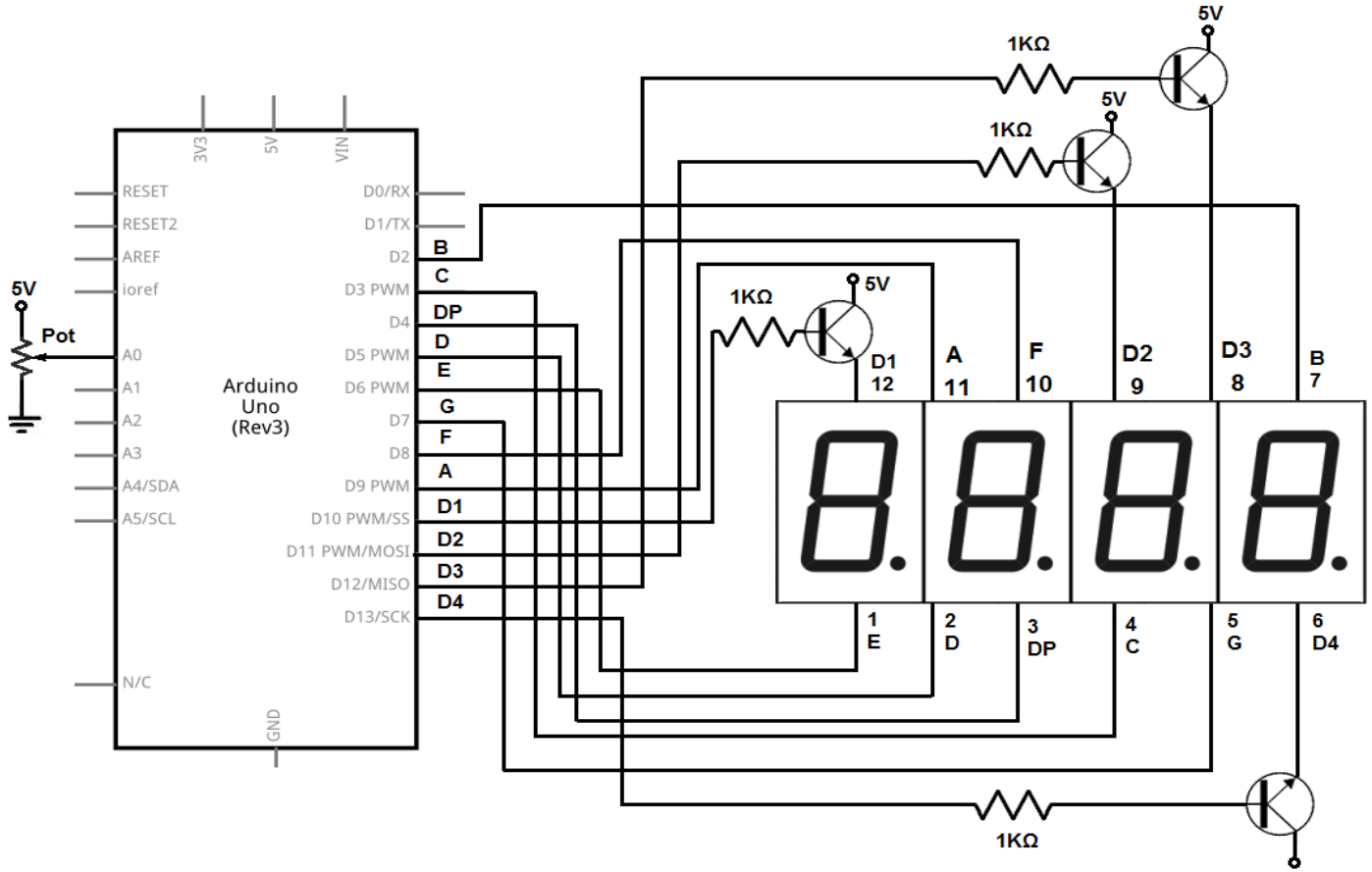


- A közösített elektróda számjegyenként van kivezetve, amelyeket időosztásos rendszerben egyenként aktiválunk, de ezek árama 100 mA fölötti is lehet (célszerű teljesítményillesztést használni, pl. tranzisztort)



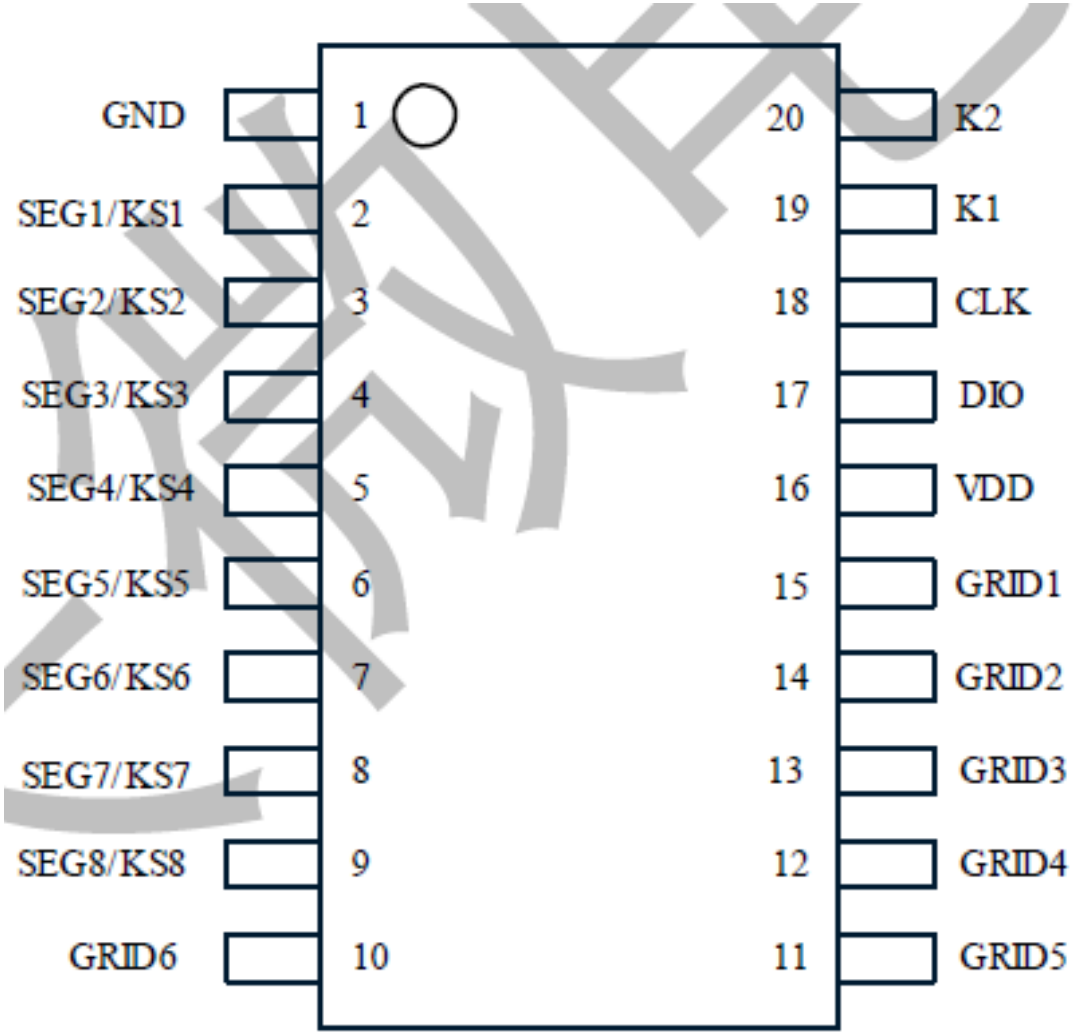
4-jegyű 7-szegmens LED kijelző

- Egy tipikus kapcsolás, közös anódú, 4 számjegyű kijelző vezérlése Arduinoval – láthatóan bonyolult megoldás.
- Egyszerűsítési lehetőség: speciális kijelzővezérlő IC használata



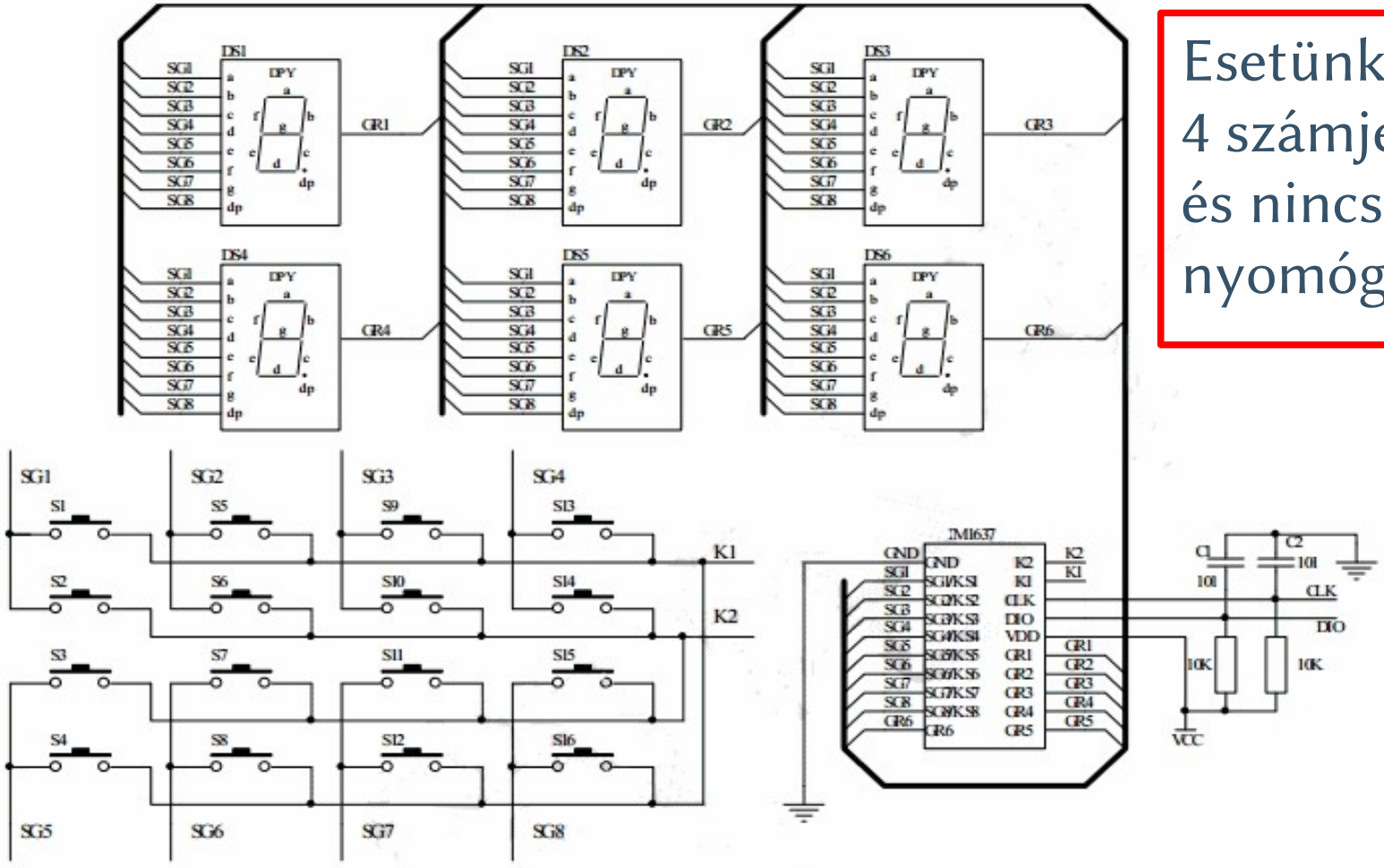
Négyszámjegyű kijelző TM1637 vezérlővel

- Négyszámjegyű kijelző (közös anódú, 3642BH típusú LED modul)
- Óra típusú kijelző (két pont középen)
- 5 V tápfeszültség
- TM1637 vezérlő kétvezetékes meghajtó
  (Shenzen Titan Microelectronics)



Mit tud a TM1637 vezérlő?

- Legfeljebb 6 jegyű 7-szegmens kijelző és 16 nyomógomb kezelése, I2C-hez hasonló, de nem szabványos soros kommunikációval
- Tipikus felhasználás: DVD lejátszó vagy ébresztőórák előlapjához

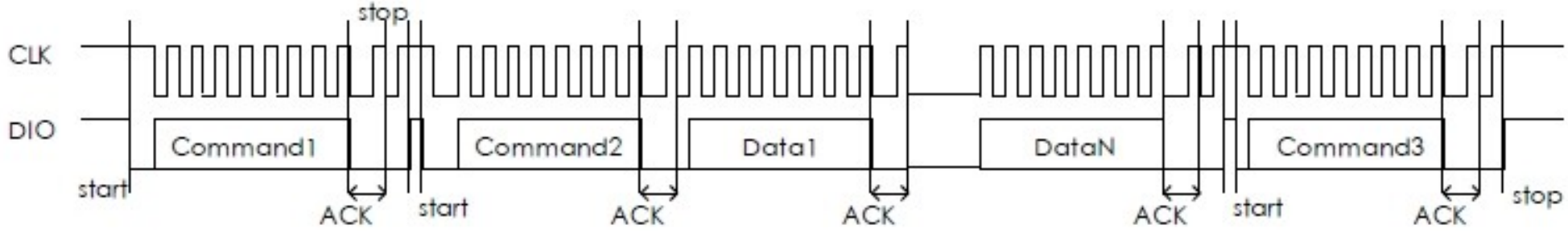


Esetünkben csak 4 számjegy van és nincs nyomógomb

TM1637 soros kommunikáció

- Két vezetékes, kétirányú, szinkron soros kommunikáció nyugtázással. Az átvitelt a mikrovezérlő irányítja (ő a master).
- **START** feltétel: magas CLK mellett H⟶L átmenet a DIO vonalon
- **STOP** feltétel: magas CLK mellett H⟶L átmenet a DIO vonalon
- **ACK:** DIO lehúzása a 9. órajelnél pozitív nyugtázást jelent
- **Adatküldés:** bájtonként, a legkisebb helyiértékű bit (LSB) van elöl
- Command1: adatküldés beállítása (**01**00 xxxx)
- Command2: regisztercím beállítása (**11**00 0xxx)
- Command3: megjelenítési mód beállítása (**10**00 xxxx)

Write SRAM data in address auto increment 1 mode.



Command1

- Írás a kijelzőre: **01**00 0**00**0_2 (automatikus címléptetéssel)
- Nyomógomb lenyomás lekérdezése: **01**00 00**1**0_2
  (válasz S0 S1 S2 K1 K2 0 0 0 alakban) – egyidejűleg több gomb lenyomását nem lehet detektálni
- Írás a kijelzőre automatikus címléptetés nélkül: **01**00 0**10**0_2
- Teszt mód: nem specifikált, a gyártó használja

b7	b6	b5	b4	b3	b2	b1	b0	Funkció	Leírás
0	**1**	0	0			0	0	Adat írás/olvasás	Write to display
0	**1**					1	0		Read keys
0	**1**				0			Regiszter címzés	Auto increment
0	**1**				1				Fixed address
0	**1**			0				Teszt mód (internal use)	Normal mode
0	**1**			1					Test mode

Command2

- A **Command2** paranccsal az adatbeírásra kijelölt regisztercímet adhatjuk meg
- Többnyire a nulla címet adjuk meg, ami a bal szélső számjegy szegmensvezérlő bitjeit tartalmazza
- A hat adatregiszter a legfeljebb hat számjegy szegmensvezérlő bitjeit tartalmazza (**1:** on, **0:** off)
- A 8 adatbit az *A – G* szegmensek és a tizedespont *(DP)* állapotát írja le

b7	b6	b5	b4	b3	b2	b1	b0	Regisztercím
1	1			0	0	0	0	C0H
1	1			0	0	0	1	C1H
1	1			0	0	1	0	C2H
1	1	0	0	0	0	1	1	C3H
1	1			0	1	0	0	C4H
1	1			0	1	0	1	C5H

b7	b6	b5	b4	b3	b2	b1	b0	
DP	G	F	E	D	C	B	A	
seg8	seg7	seg6	seg5	seg4	seg3	seg2	seg1	
								GRID1
								GRID2
								GRID2
								GRID4
								GRID5
								GRID6

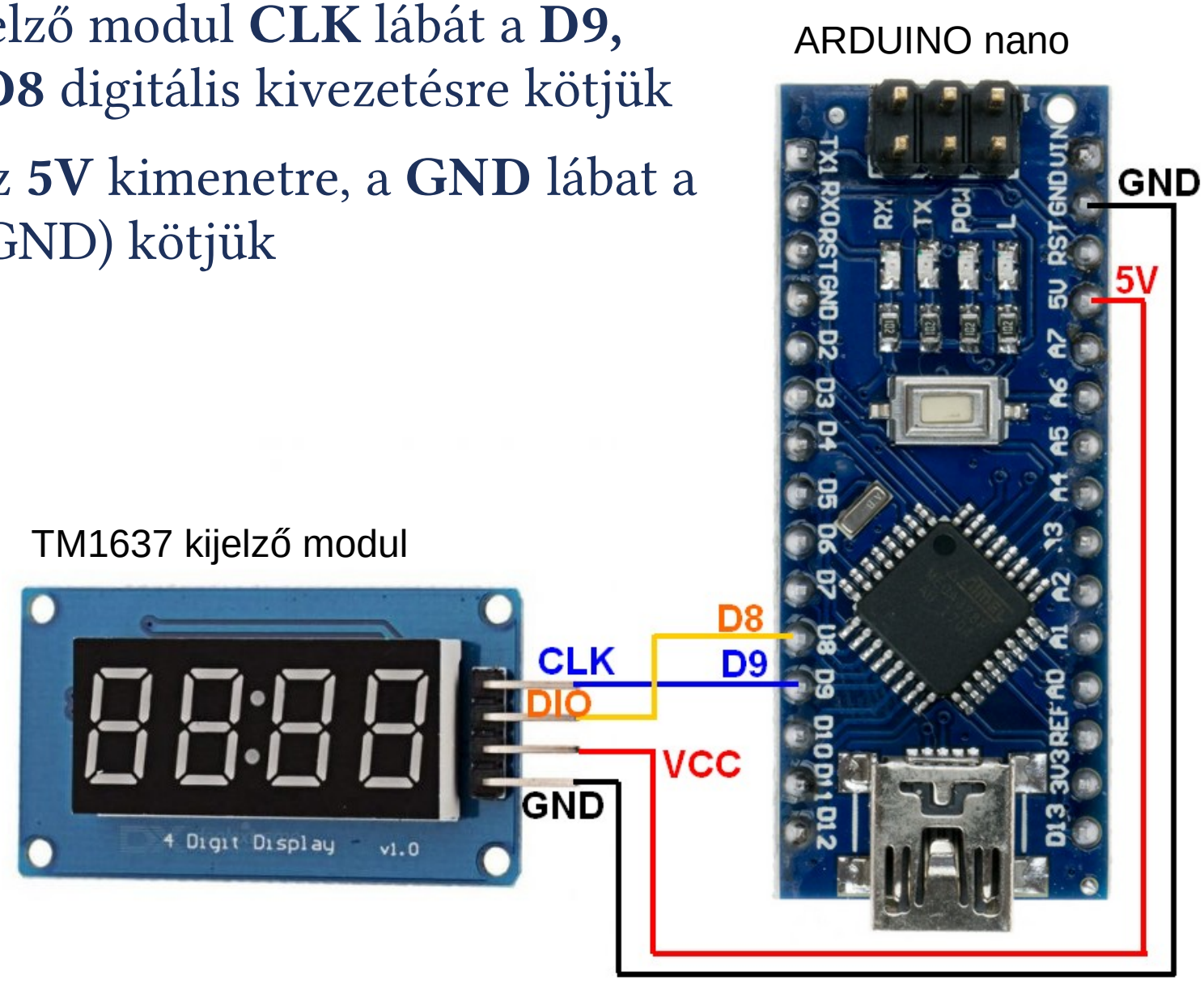
Command3

- A Command3 parancs a kijelző fényerejének beállítására (8 fokozatban), illetve ki- és bekapcsolására szolgál
- A nagyobb PWM kitöltés nagyobb fényerőt jelent
- A maximális fényerőt az **10**00 **1111**$_2$ parancs állítja be

b7	b6	b5	b4	b3	b2	b1	b0	Funkció	Leírás
1	0	0	0		0	0	0	FÉNYERŐ	PWM 1/16
1	0				0	0	1		PWM 2/16
1	0				0	1	0		PWM 4/16
1	0				0	1	1		PWM 10/16
1	0				1	0	0		PWM 11/16
1	0				1	0	1		PWM 12/16
1	0				1	1	0		PWM 13/16
1	0				1	1	1		PWM 14/16
1	0			0				Display on/off	Display OFF
1	0			1					Display ON

A kapcsolás

- A **TM1637** kijelző modul **CLK** lábát a **D9**, a **DIO** lábát a **D8** digitális kivezetésre kötjük
- A **VCC** lábát az **5V** kimenetre, a **GND** lábát a közös pontra (GND) kötjük



TM1637_bare_metal.ino

```
#define TM1637_I2C_COMM1    0x40
#define TM1637_I2C_COMM2    0xC0
#define TM1637_I2C_COMM3    0x80
#define BITDELAY             100
#define pinClk                 9
#define pinDIO                 8

const byte digits[] = {
 // xGFEDCBA
  0b00111111,    // 0
  0b00000110,    // 1
  0b01011011,    // 2
  0b01001111,    // 3
  0b01100110,    // 4
  0b01101101,    // 5
  0b01111101,    // 6
  0b00000111,    // 7
  0b01111111,    // 8
  0b01101111,    // 9
};

int num = 0;     // Számláló 0000 → 9999
byte data[4];    // A különválasztott számjegyek
```

- Az adatlap alapján próbáljuk meg működésbe hozni a kijelzőt!
- Definiáljuk a három parancsot, és a kivezetések sorszámát!
- Definiáljuk a számjegyek szegmensképét (hogy melyik számjegy esetén melyik szegmens világítson)!
  1: a szegmens világít
  0: a szegmens nem világít
- Deklaráljuk a globális változókat!

Folytatás a következő oldalon

TM1637_bare_metal.ino

Kiindulási állapot: CLK = DIO = HIGH legyen!

```
void TM1637_start() {
  //--- START feltétel generálása: magas CLK mellett DIO H->L átmenet ----
  pinMode(pinDIO, OUTPUT);          // DIO magasról alacsonyra vált
  delayMicroseconds(BITDELAY);      // Késleltetés
}
```

Kiindulási állapot: CLK = LOW legyen!

```
void TM1637_stop() {
  //--- STOP feltétel generálása: magas CLK mellett DIO LO->H átmenet ----
  pinMode(pinDIO, OUTPUT);          // DIO vonal alacsony legyen
  delayMicroseconds(BITDELAY);      // késleltetés
  pinMode(pinClk, INPUT);           // A CLK vonal magas legyen
  delayMicroseconds(BITDELAY);      // késleltetés
  pinMode(pinDIO, INPUT);           // DIO alacsonyról magasra vált (STOP)
  delayMicroseconds(BITDELAY);      // Késleltetés
}
```

I2C busz „nyitott nyelőelektródás” vezérlése:
INPUT mód = magas szint, OUTPUT mód = alacsony szint

```
void setup() {
  pinMode(pinClk, INPUT);           // Alapértelmezett adatáramlási irány
  pinMode(pinDIO, INPUT);           // és kimeneti szint beállítása
  digitalWrite(pinClk, LOW);        // Kimenetként csak lefelé húzzuk a vonalat
  digitalWrite(pinDIO, LOW);        // Bemenetként pedig a felhúzás érvényesül
}
```

Folytatás a következő oldalon

TM1637_bare_metal.ino

Egy adatbájt (8 bit, LSB first) kiküldése a soros buszra

```
bool writeByte(byte b) {
  byte data = b;
  for (uint8_t i = 0; i < 8; i++) {  // 8 adatbit kiküldése
    pinMode(pinClk, OUTPUT);         // CLK alacsony legyen
    delayMicroseconds(BITDELAY);     // Késleltetés
    if (data & 0x01)                 // DIO legyen magas vagy alacsony
      pinMode(pinDIO, INPUT);        // de ezt az adatirány-váltással állítjuk be
    else pinMode(pinDIO, OUTPUT);
    delayMicroseconds(BITDELAY);     // Késleltetés
    pinMode(pinClk, INPUT);          // CLK legyen újra magas
    delayMicroseconds(BITDELAY);     // Késleltetés
    data = data >> 1;                // Adatbitek léptetése
  }   //--- Nyugtázás kezelése -------
  pinMode(pinClk, OUTPUT);           // CLK legyen ismét alacsony
  pinMode(pinDIO, INPUT);            // DIO legyen bemenet (beolvasás jön)
  delayMicroseconds(BITDELAY);       // Késleltetés
  pinMode(pinClk, INPUT);            // CLK magasra vált
  delayMicroseconds(BITDELAY);       // Késleltetés
  byte ack = digitalRead(pinDIO);    // ACK esetén a külső eszköz 0-ra húz
  if (ack == 0)
    pinMode(pinDIO, OUTPUT);         // Nyugtázás esetén DIO alacsony legyen
  delayMicroseconds(BITDELAY);       // Késleltetés
  pinMode(pinClk, OUTPUT);           // CLK álljon vissza alacsonyra
  delayMicroseconds(BITDELAY);       // Késleltetés
  return ack;
}
```

Folytatás a következő oldalon

TM1637_bare_metal.ino

```
void loop() {
  int n = num;
  data[0] = n%10; n/=10;
  data[1] = n%10; n/=10;
  data[2] = n%10; n/=10;
  data[3] = n%10;
  TM1637_start();
  writeByte(TM1637_I2C_COMM1);
  TM1637_stop();
  TM1637_start();
  writeByte(TM1637_I2C_COMM2);
  writeByte(digits[data[3]]);
  writeByte(digits[data[2]]);
  writeByte(digits[data[1]]);
  writeByte(digits[data[0]]);
  TM1637_stop();
  TM1637_start();
  writeByte(TM1637_I2C_COMM3 | 0x0F);
  TM1637_stop();
  num++;
  delay(100);
}
```

A számok kiíratása 0000-tól 9999-ig (100 ms várakozással)

Egyesek száma a 10-re való osztási maradék (n%10)
n = n/10 „lecsípi” az egyeseket, most már n%10 a tízeseket adja meg, és így tovább...

$0100\ 0000_2$ (adatküldés automatikus címléptetésel)

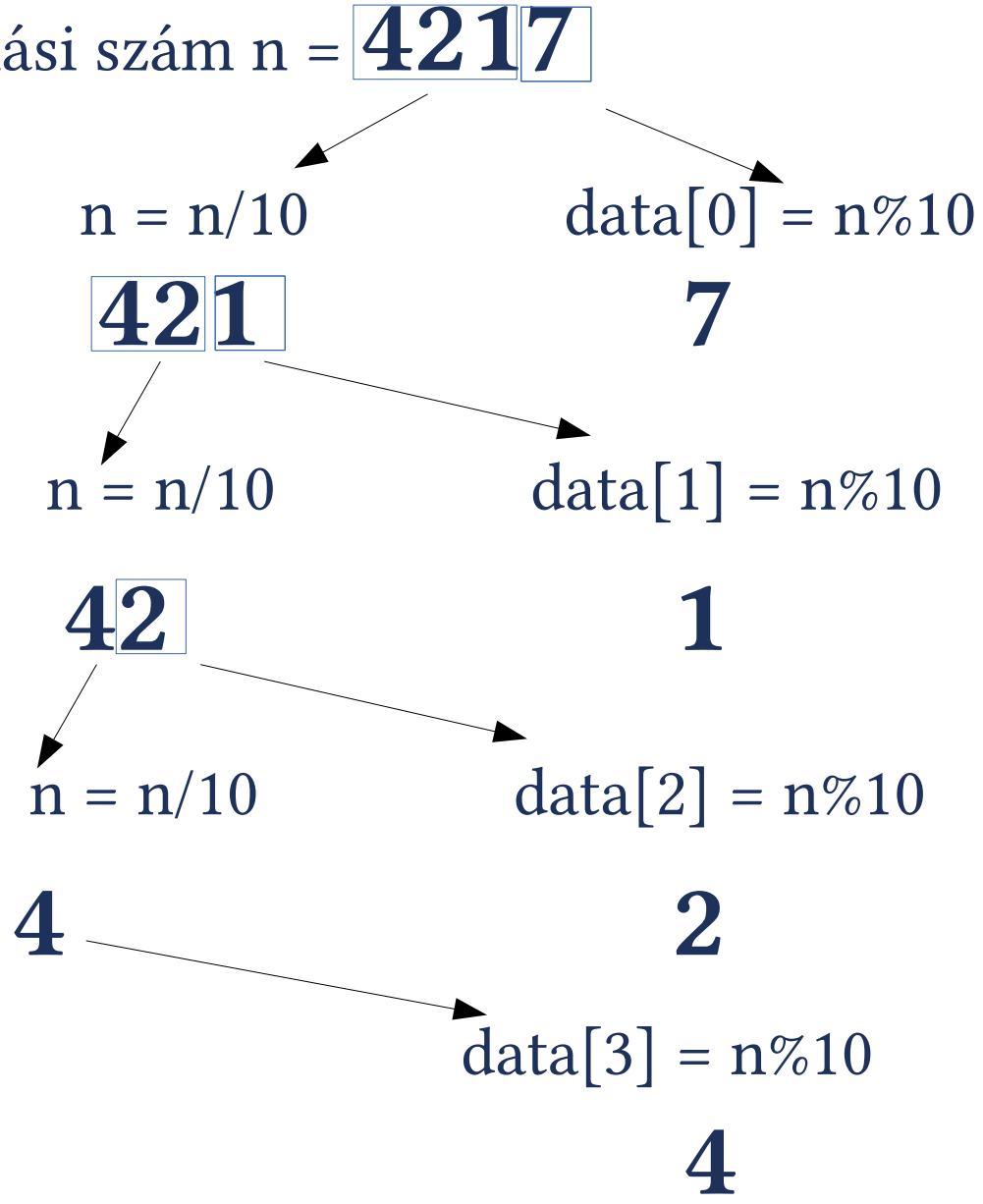
$1100\ 0000_2$ (regisztercím = 0 beállítása)
Ezresek
Százasok
Tízesek
Egyesek

Kijelző vezérlés = maximális fényerő és ON

Számláló léptetése (num = num + 1;

A számjegyekre bontás részletezve

- Legyen a kiindulási szám n = **4217**



A TM1637Display programkönyvtár

- Arduino programkönyvtár a TM1637 kényelmes kezeléséhez
- **Szerző:** Avishay Orpaz
- **Forrás:** https://github.com/avishorp/TM1637
- **Telepítés:** Arduino IDE **Tools/Manage libraries** menüben a *TM1637* keresőszóra két könyvtárat találunk, ebből az Avishay Orpaz által írtat válasszuk ki, és kattintsunk az **INSTALL** gombra!
- **Használata:** a program elejére szúrjuk be az alábbi sort:

```
#include <TM1637Display.h>
```

majd példányosítsuk a **TM1637Display** osztályt az alábbi módon:

```
TM1637Display display(CLK, DIO);
```

ahol **CLK** és **DIO** a kijelző vezérlésére használt két Arduino kivezetés sorszáma (az általunk használt kapcsolásban **D9** és **D8**)

A TM1637Display programkönyvtár metódusai

- **setBrightness**(brightness) – fényerő beállítása (0 – 7)
- **showNumberDec**(num) – szám kiíratása decimális alakban, ahol num = 0 – 9999, negatív számok esetén pedig 0-tól –999-ig
- **showNumberDecEx**(num,dots) – szám kiíratása decimális alakban, pontvezérléssel (dots = 64 kigyújtja a kettőspontot)
- **showNumberHexEx**(num,dots) – szám kiíratása hexadecimális alakban, pontvezérléssel (dots = 64 kigyújtja a kettőspontot)
- **setSegments**(data[]) - szegmensek beállítása a data[] tömb szerint
- **clear**() - a kijelző törlése
- A számkijelzésnél egy további paramétert is megadhatunk a bevezető nullák megjelenítésének vezérlésére, például:

```
display.showNumberDecEx(945,64,true);
```

ekkor a kijelzőn **09:45** jelenik meg. Ha nem adjuk meg, *false* lesz.

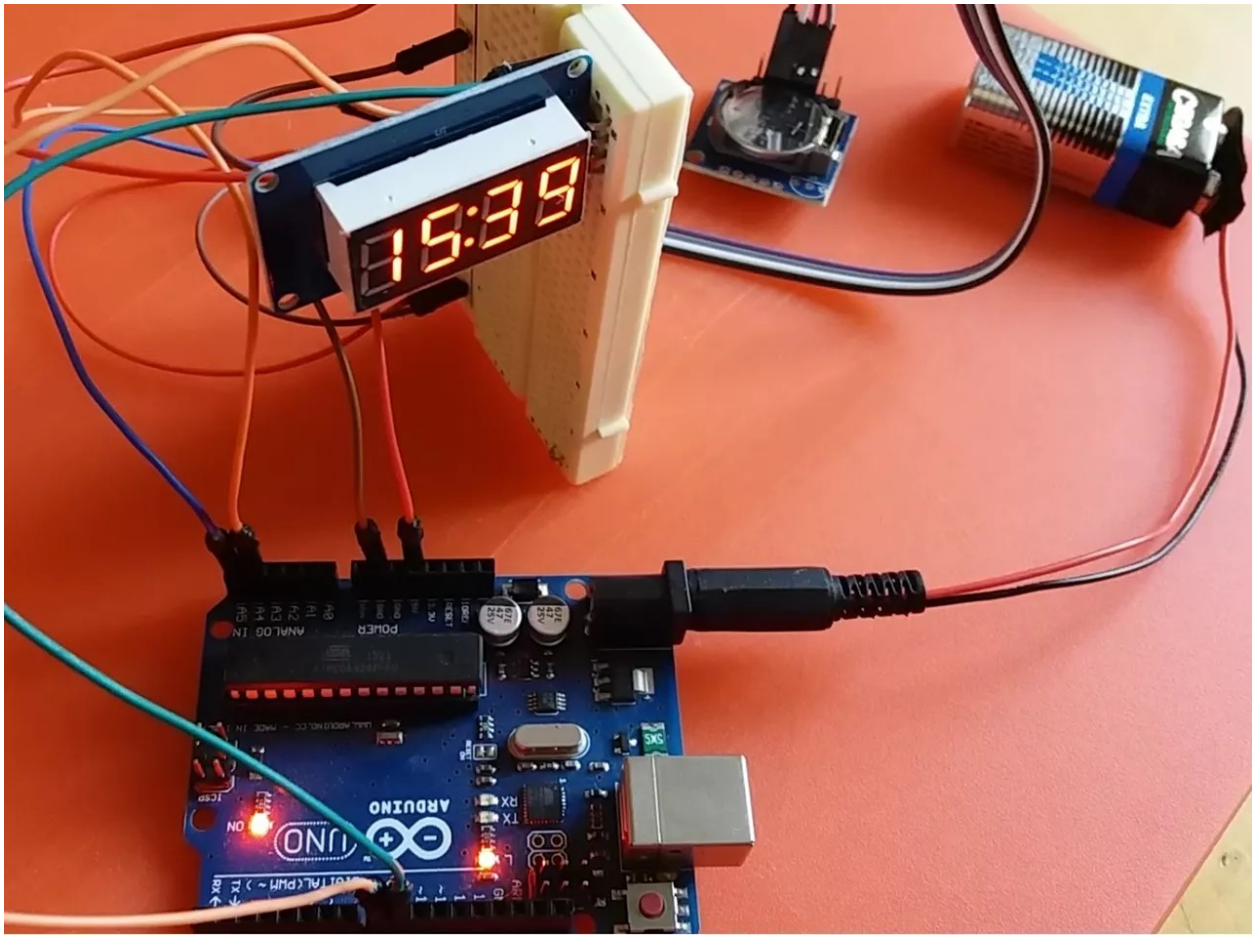
TM1637_demo.ino

- Az alábbi programban bemutatjuk az óra kijelzés és a számláló kijelzés legegyszerűbb eseteit

```
#include <TM1637Display.h>

const int CLK = 9;                      // CLK a D9 lábra kötve
const int DIO = 8;                      // DIO a D8 lábra kötve
int num = 0;                            // számláló

TM1637Display display(CLK, DIO);        // példányosítás és konfigurálás

void setup() {
  display.setBrightness(7);             // Maximális fényerő
  display.showNumberDecEx(1845,64); // 18:45 kiírása
  delay(5000);
}

void loop() {
  for (num=0; num < 9999; num++) {  // Számlálás 0-tól 9999-ig
    display.showNumberDec(num);     // num kiírása decimálisan
    delay(500);                     // 500 ms várakozás
  }
}
```

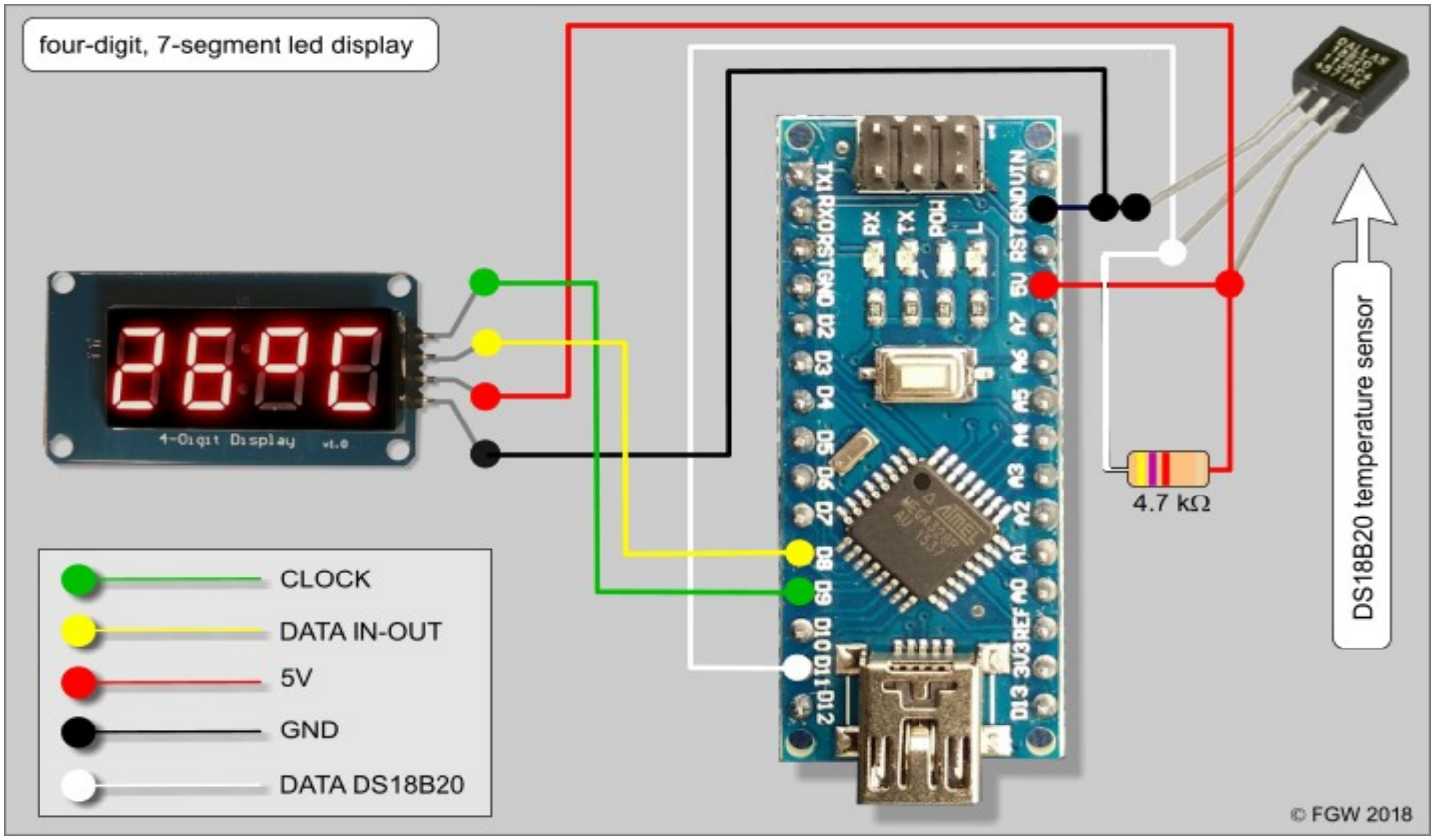
Digitális kijelzésű óra

- A kijelző modulunk egyik lehetséges felhasználása egy digitális kijelzésű óra (óra és perc kijelzéssel)
- Link: hackster.io/pentiumcadiz/4-digit-rtc-clock-85 068b
- **Hozzávalók:**
  - TM1637 kijelző
  - DS1307 vagy hasonló RTC modul
  - Arduino
  - 9V-os elem vagy hálózati adapter



Digitális hőmérő

- Ez a projekt egy **DS18B20** hőmérőt és egy négy digites **TM1637** kijelző modult használ. A kijelzőt a **TM1637Display** könyvtár segítségével kezelik. Az eredeti kapcsolás hibás, itt kijavítottuk.
- Four digit, 7-segment led display for Arduino based on the TM1637 driver
  zonnepanelen.wouterlood.com/10-four-digit-7-segment-led-display-for-arduino-based-on-the-tm1637-driver/



Az Arduino nano kártya kivezetései