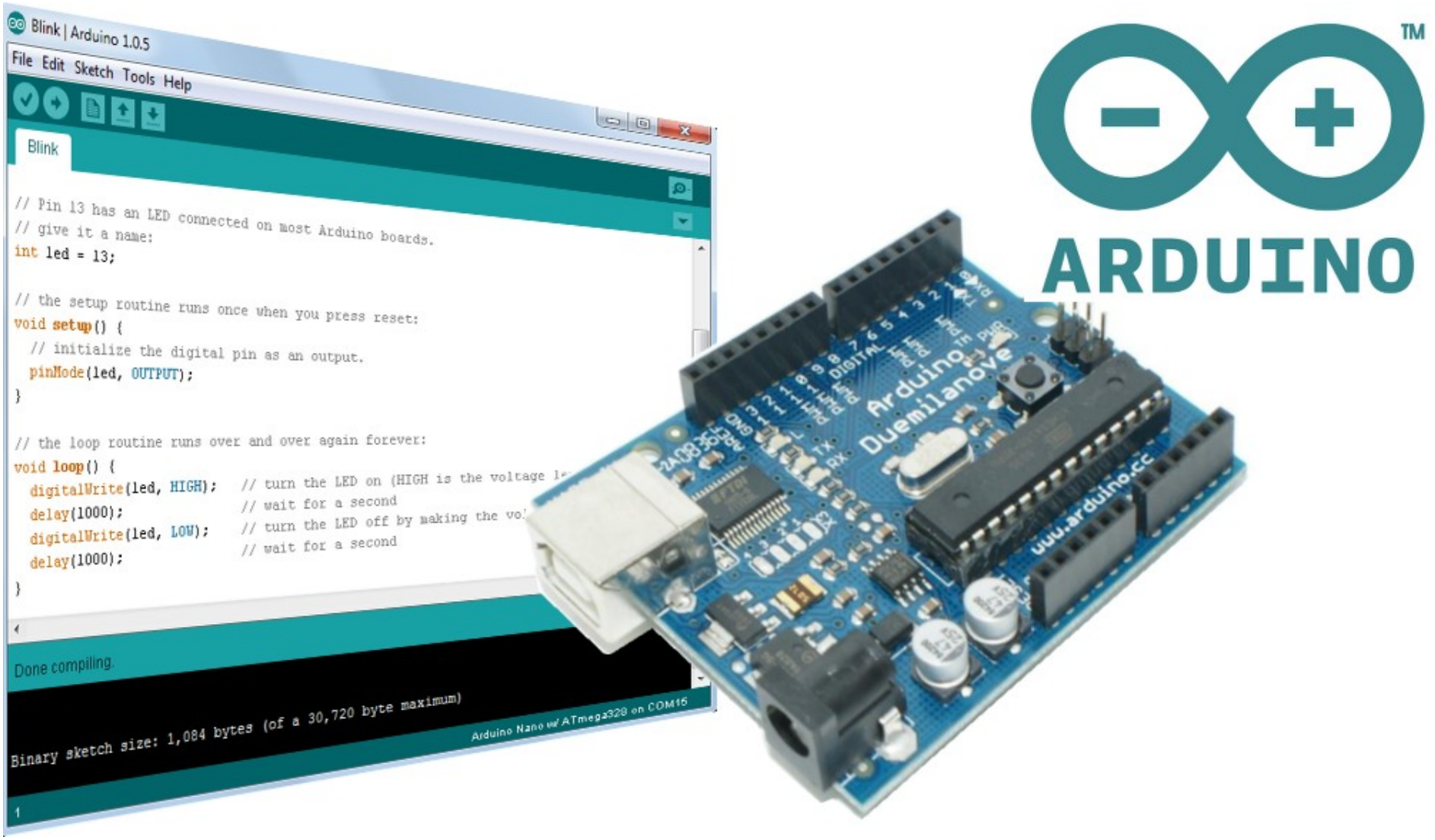
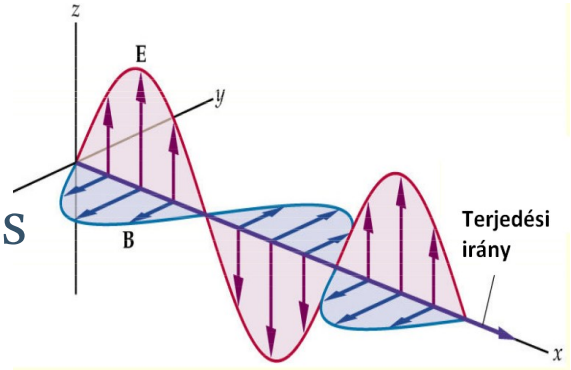
Arduino tanfolyam kezdőknek és haladóknak

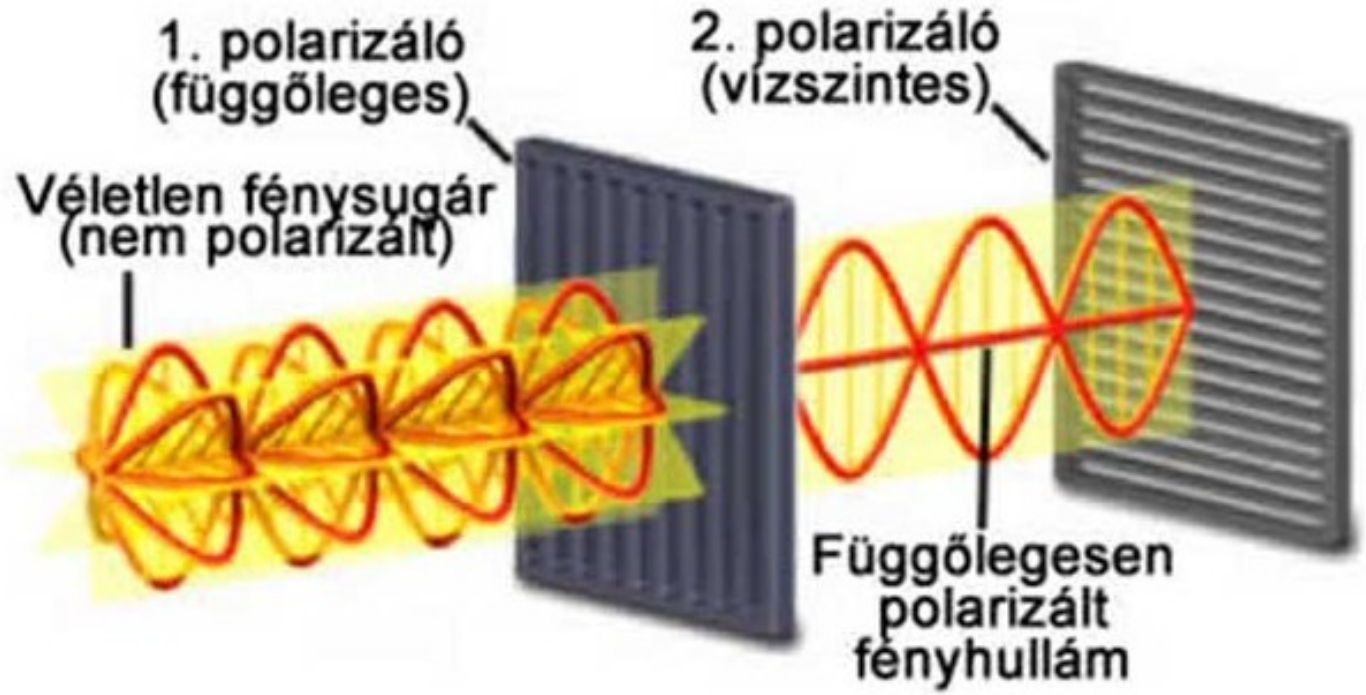


10. Az I2C kommunikációs csatorna

A fény polarizációja

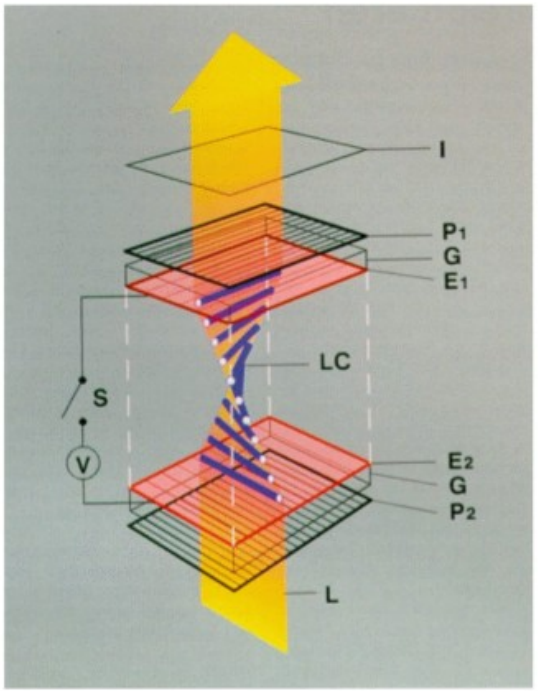
- A fény elektromágneses sugárzás, benne az elektromos és a mágneses tér folytonosan egymásba alakul
- A fény transzverzális hullám, az elektromos és mágneses térerősség a haladási irányra merőleges
- A polarizált fény csak „egy síkban rezeg”

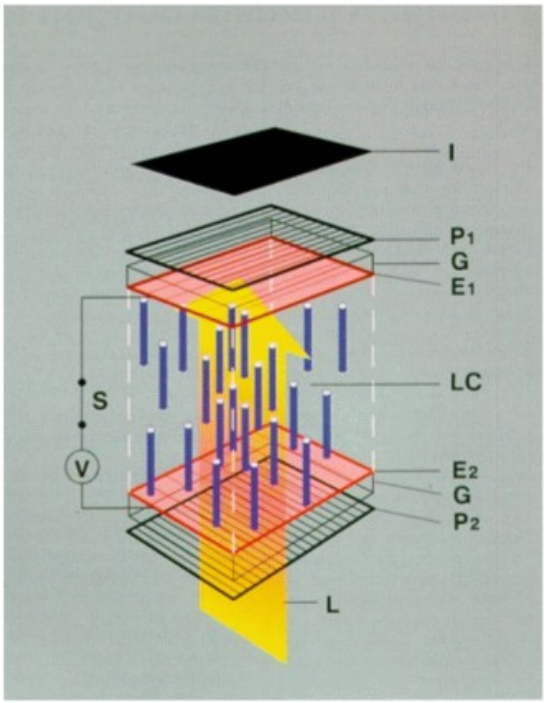




LCD = Liquid Crystal Display

- **Folyadékkristály (Liquid Crystal):** olyan (szerves) anyag, mely sűrű folyadéknak tekinthető, ugyanakkor molekulái – a kristályokhoz hasonlóan – képesek struktúrákba rendeződni.



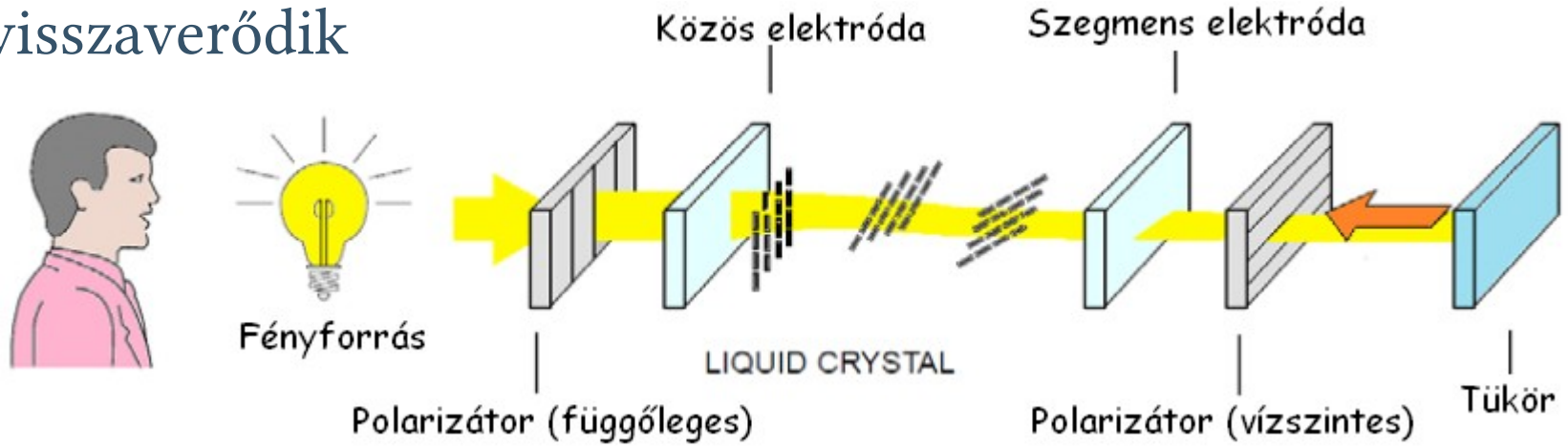


- Az alapesetben csavart struktúra segít átjuttatni a polarizált fényt az elforgatott polárszűrőkön. Az elektromos tér átrendezi a molekulákat, már nem segítik a fény átjutását.

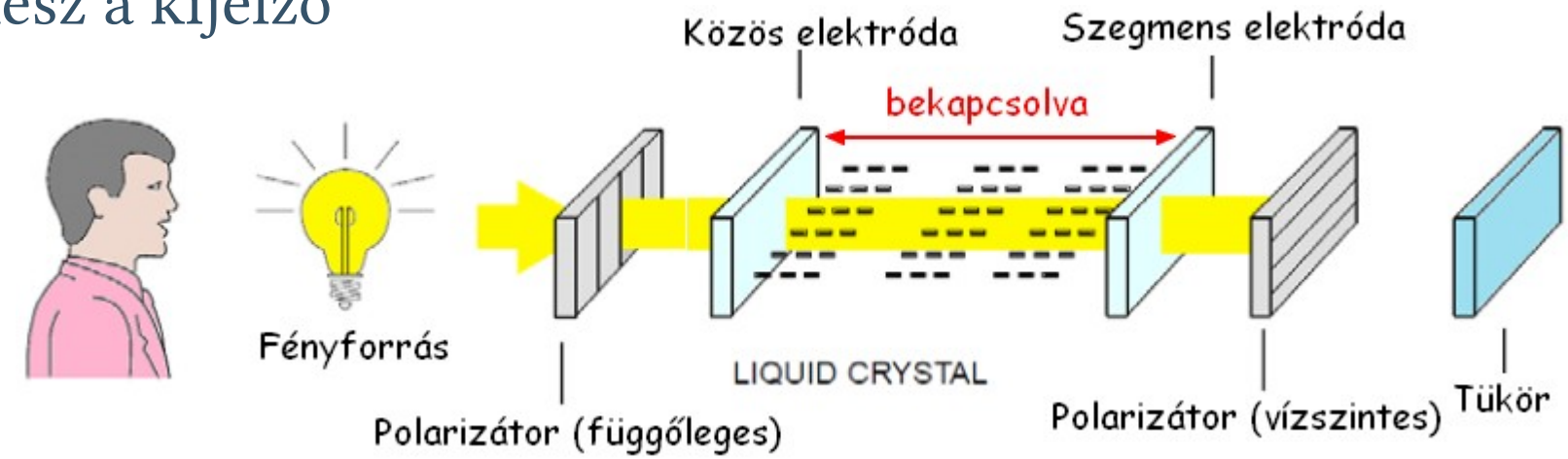
Forrás: https://en.wikipedia.org/wiki/Twisted_nematic_field_effect

Reflektív (fényvisszaverő) kijelző

- A belépő polarizált fénynyaláb a molekulák fokozatos elfordulását követve elfordul, majd a hátul elhelyezett tükörről visszaverődik

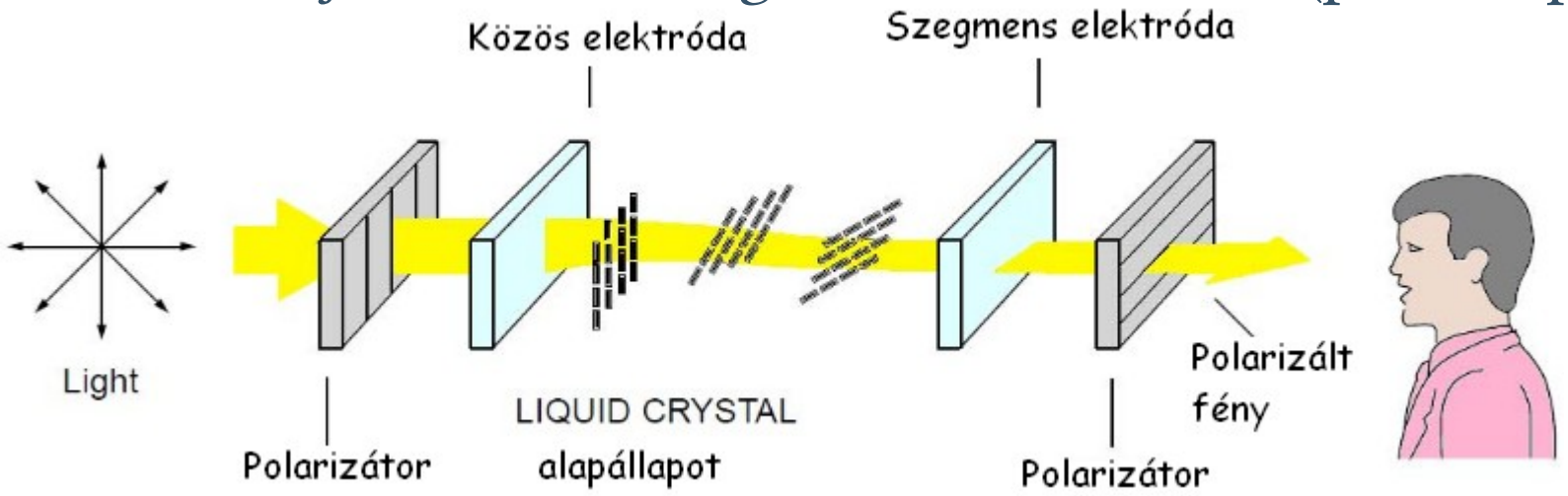


- A „bekapcsolt” helyeken a visszavert fény hiánya miatt sötétebb lesz a kijelző

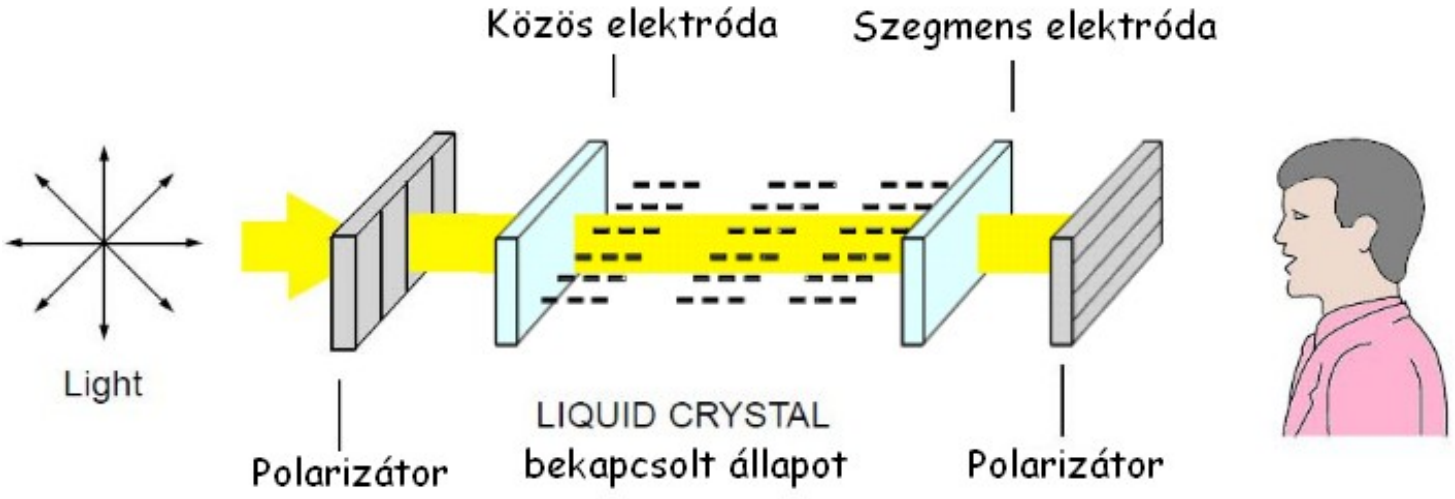


Transmissive (áteresztő) kijelző

- Az áteresztő kijelző háttérvilágítással rendelkezik (pl. LED panel)

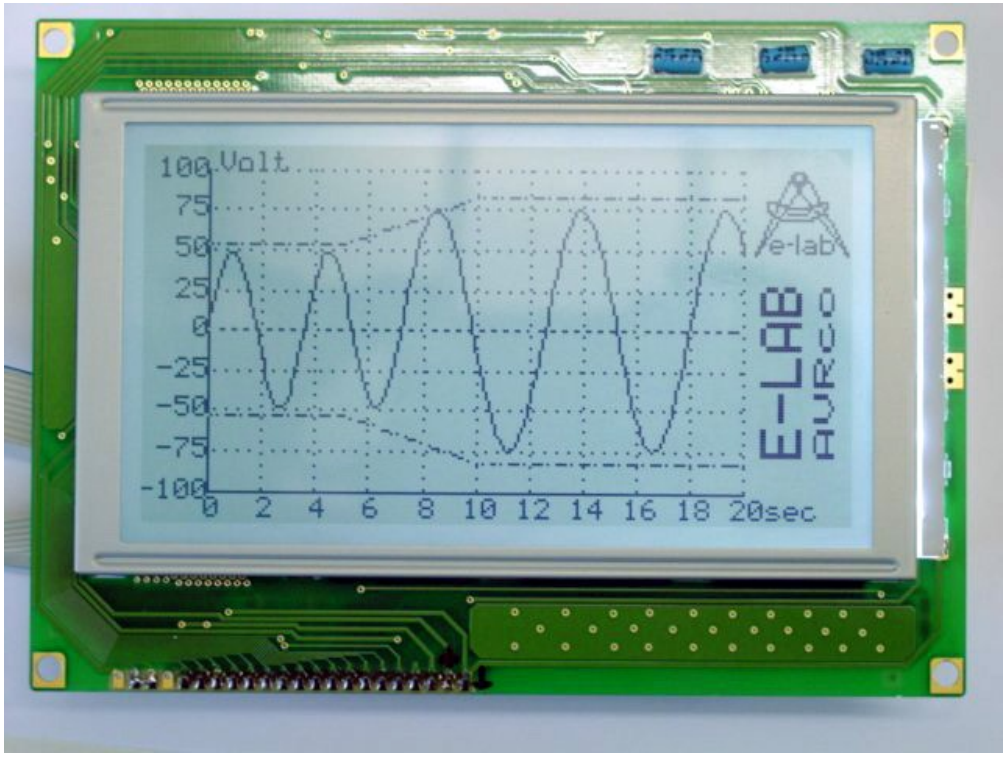


- Az elektromos tér hatására a folyadékkristály molekulái átrendeződnek, a fényáteresztés lecsökken.

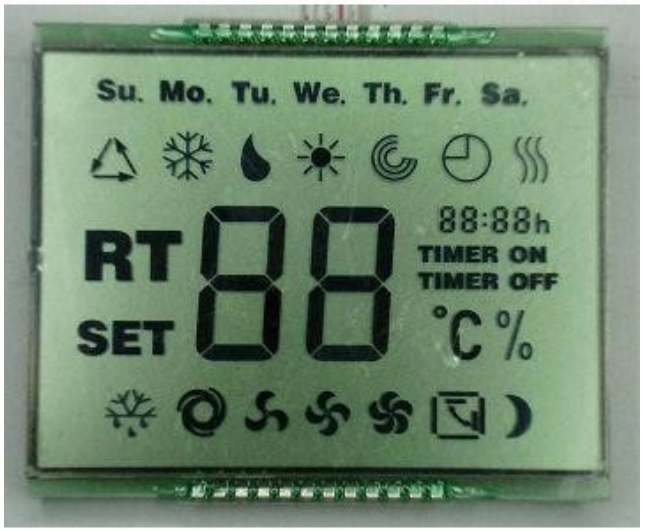


LCD kijelző típusok

- Grafikus pontmátrix kijelző



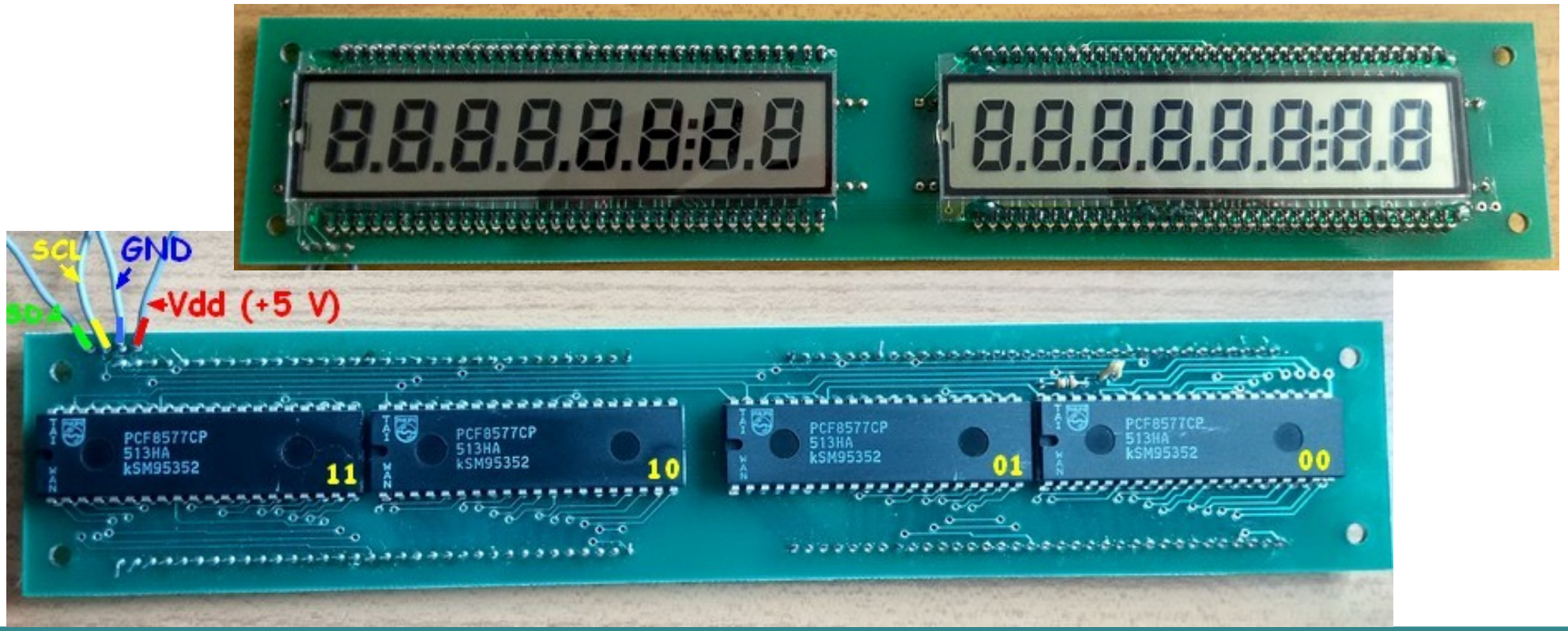
Szegmens kijelző



Alfanumerikus pontmátrix kijelző (4x20karakter)

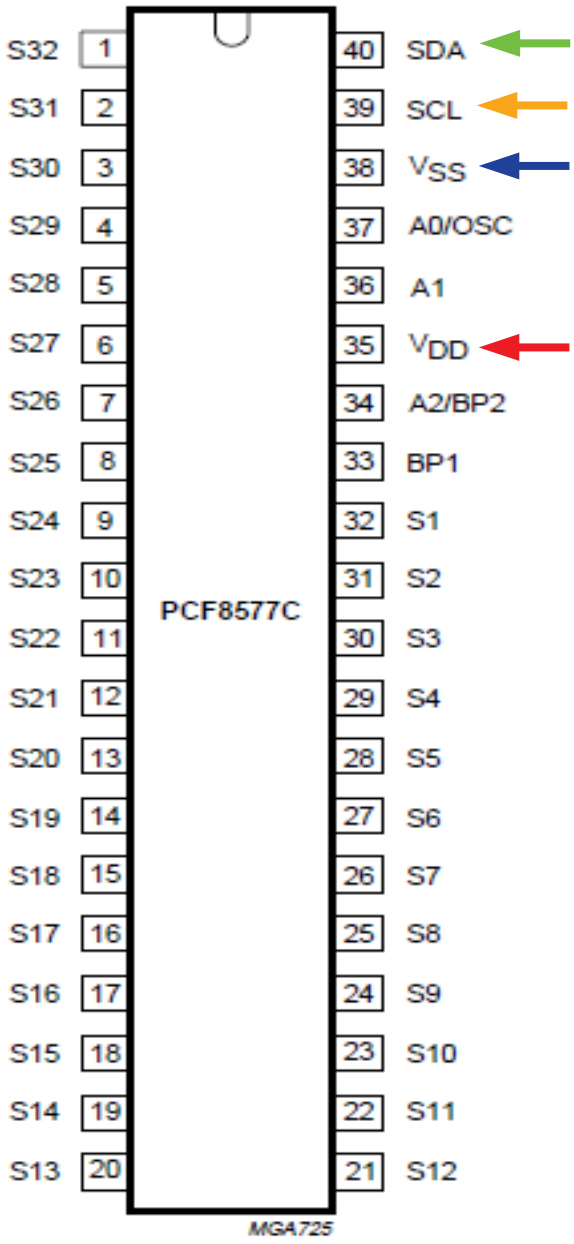
Az ismeretlen kijező panel jellemzői

- 2 db 7 szegmenses LCD számkijelző (kijelzőnként 64 db szegmens: 8 db számjegy, 7 db tizedespont, 1 db kettőspont)
- 4 db Philips PCF8577CP vezérlő (I2C vezérlés, 4 x 32 szegmens meghajtása)
- 4 vezetékes meghajtás (**SDA, SCL, GND, +5V**)



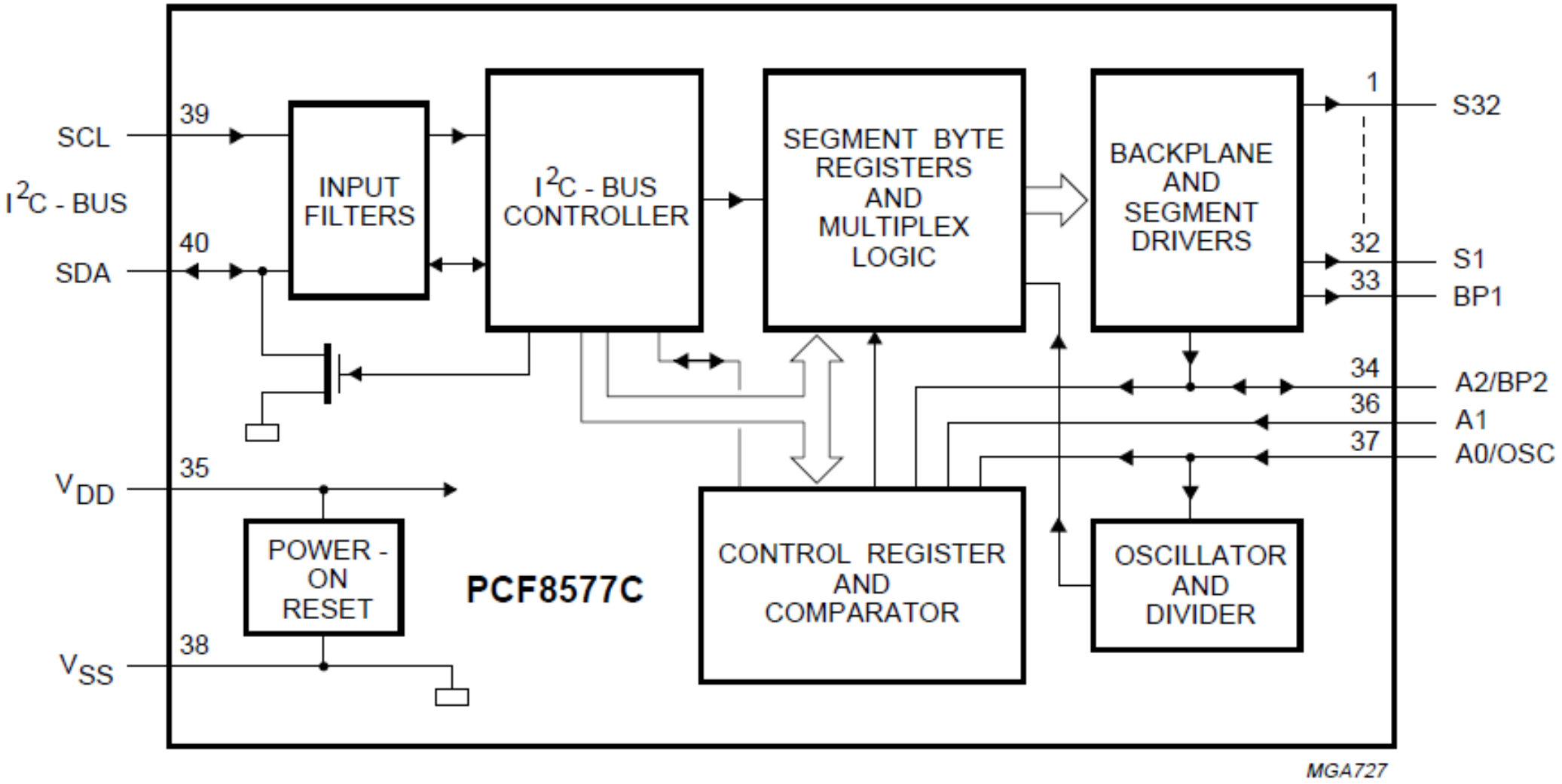
PCF8577C adatlap

- LCD direct/duplex meghajtó
- I2C-busz illesztővel
- Tápfeszültség: 2.5 – 6 V
- Auto-inkrementális betöltési címzés, eszköz alcímeken keresztül is
- I2C-busz cím: 011 1010_2 (0x3A)
- **SDA:** I2C adatvonal
- **SCL:** I2C órajel
- **VSS:** közös pont (GND)
- **VDD:** tápfeszültség (+5 V)
- A0, A1, A2: címvonalak
- BP1: hátlap (back plane)



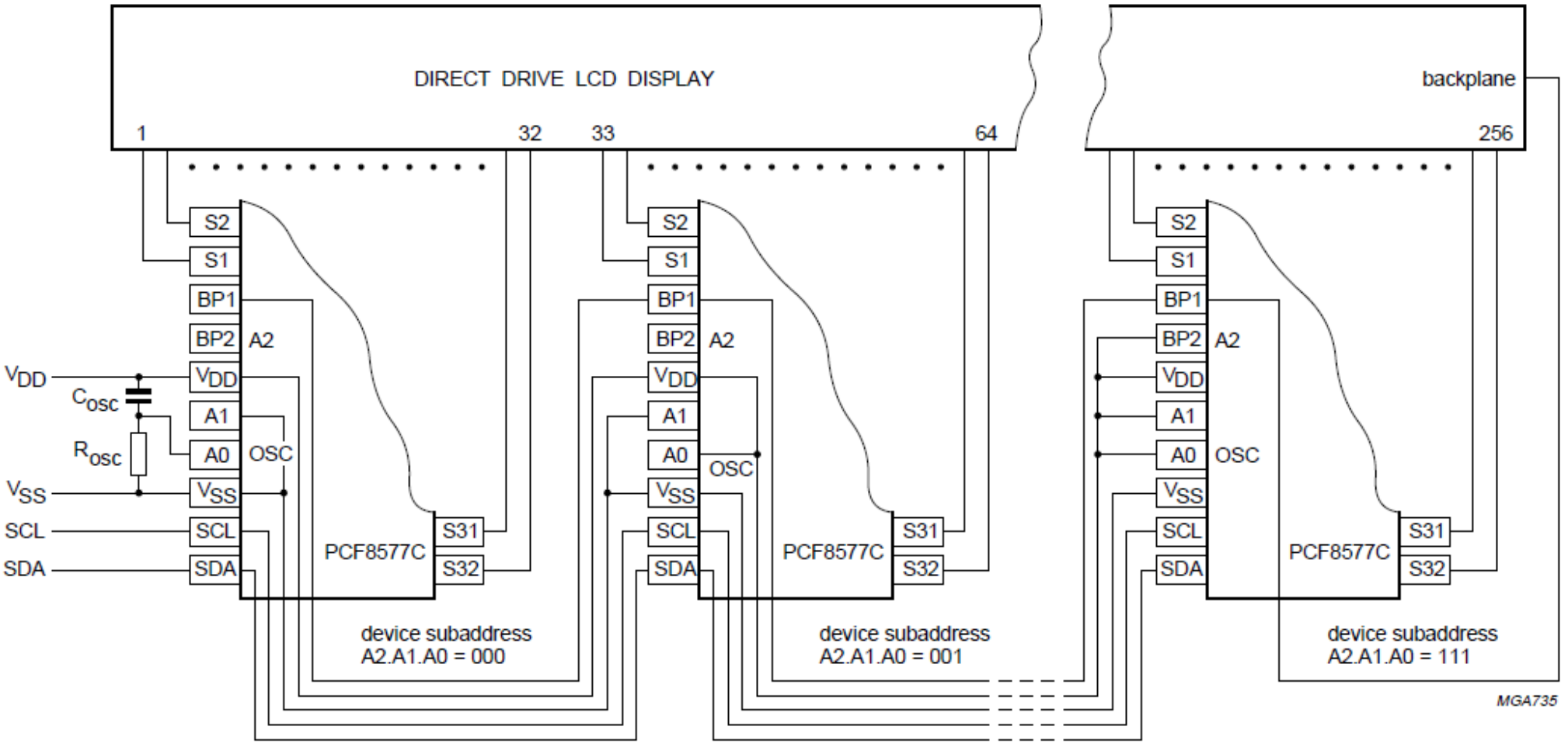
PCF8577C blokkvázlat

- Az eszköz akkor reagál, ha az A0...A2 bemenetek állapotai és a vezérlő regiszter megfelelő bitjei megegyeznek



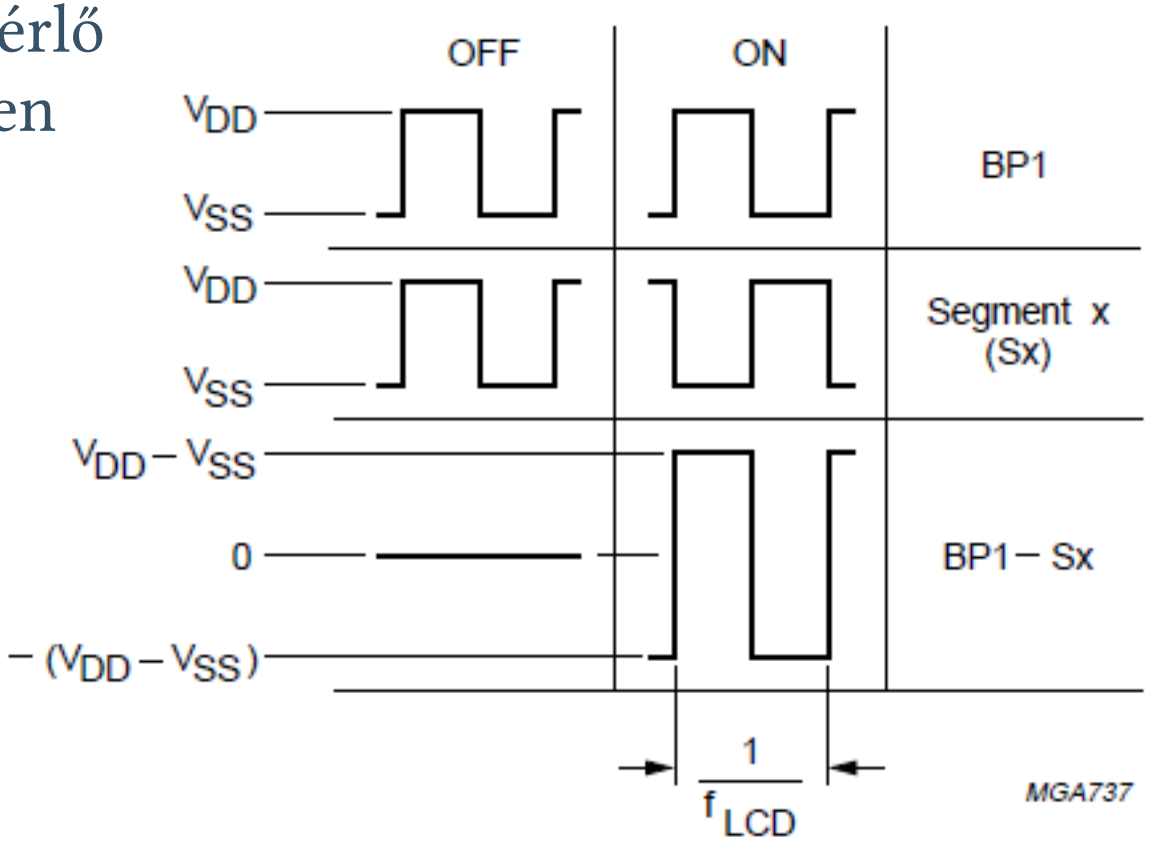
Több vezérlő felfűzése

- Az oszcillátor frekvenciáját az első (000 cím) eszközön kell beállítani a C_{osc} (680 pF) és R_{osc} (1 MΩ) időzítő taggal.
- A címeket az A0/A1/A2 bemeneteken állíthatjuk be (VDD=1, VSS=0)



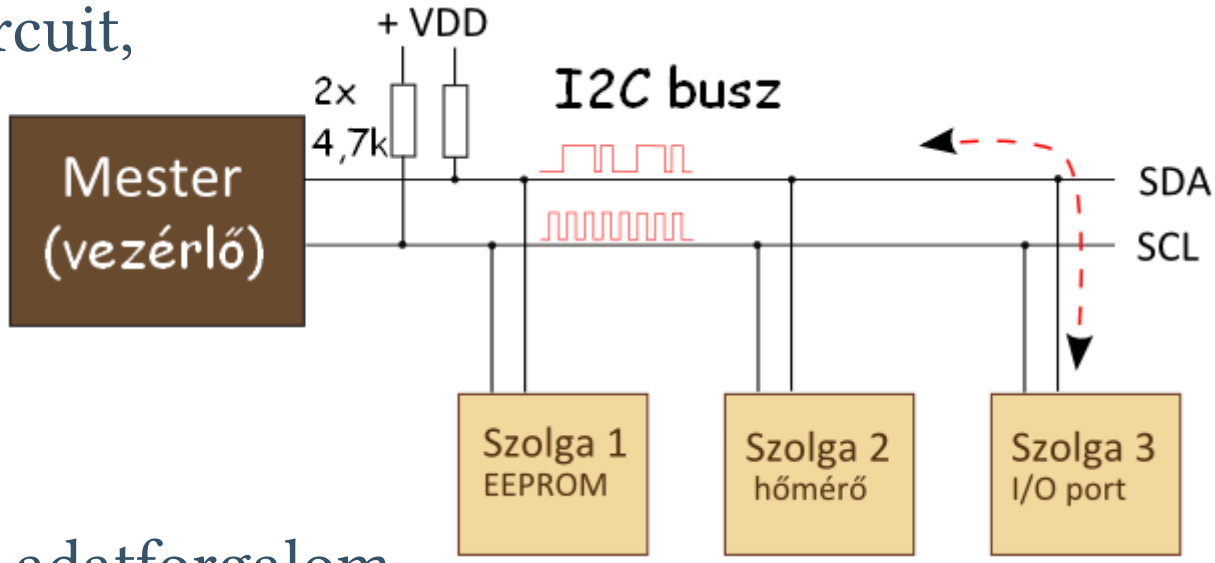
LCD szegmensek meghajtása

- A folyadékkristályt váltakozó feszültséggel kell meghajtani. Az egyenáram tönkretenné a folyadékkristályt.
- A frekvencia f_{LCD} = 30..1000 Hz, amit az oszcillátor állít elő
- Váltakozó feszültség helyett BP1-et és a szegmensvezérlő elektródákat ellenütemben kapcsolgatjuk magas, ill. alacsony szintre, így a polaritás megfordul.

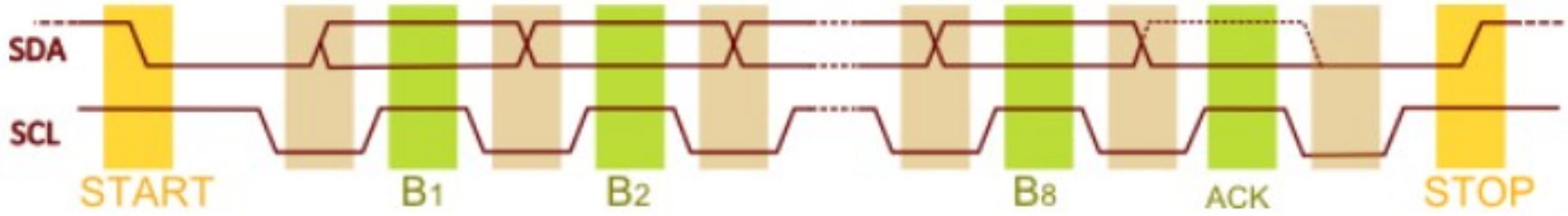


Az I2C busz és használata

- I2C = Inter Integrated Circuit, azaz ingerált áramkörök közötti kommunikáció
- A kétvezetékes buszt a PHILIPS fejlesztette ki 1995-ben



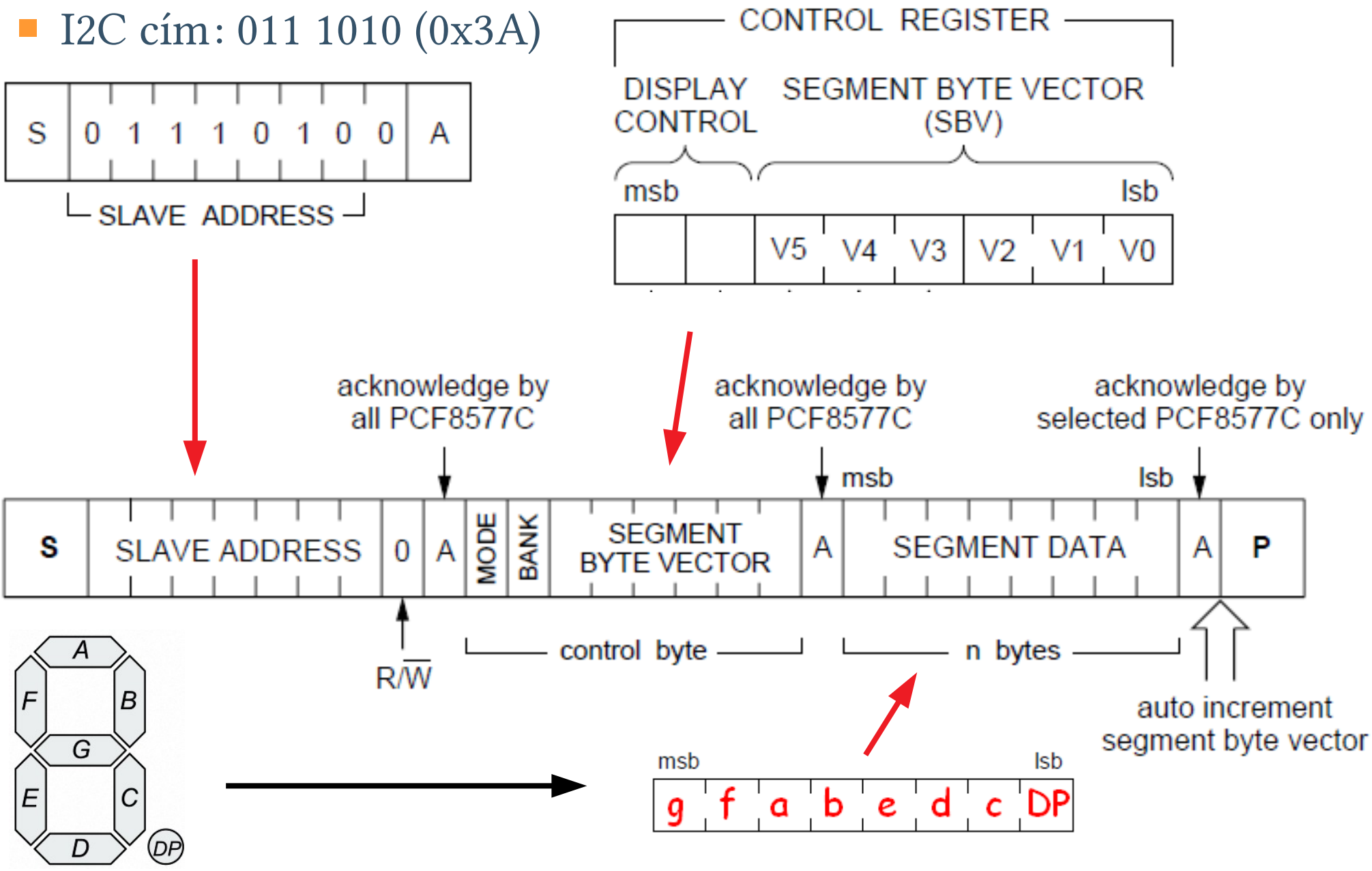
- Soros, 8-bit-es, kétirányú adatforgalom, sebessége 100 kbit/s, vagy 400 kbit/s
- Az eszközök egy 7-bites címmel címezhetők



Az I2C kommunikáció idődiagramja (egy bájt kiküldése)

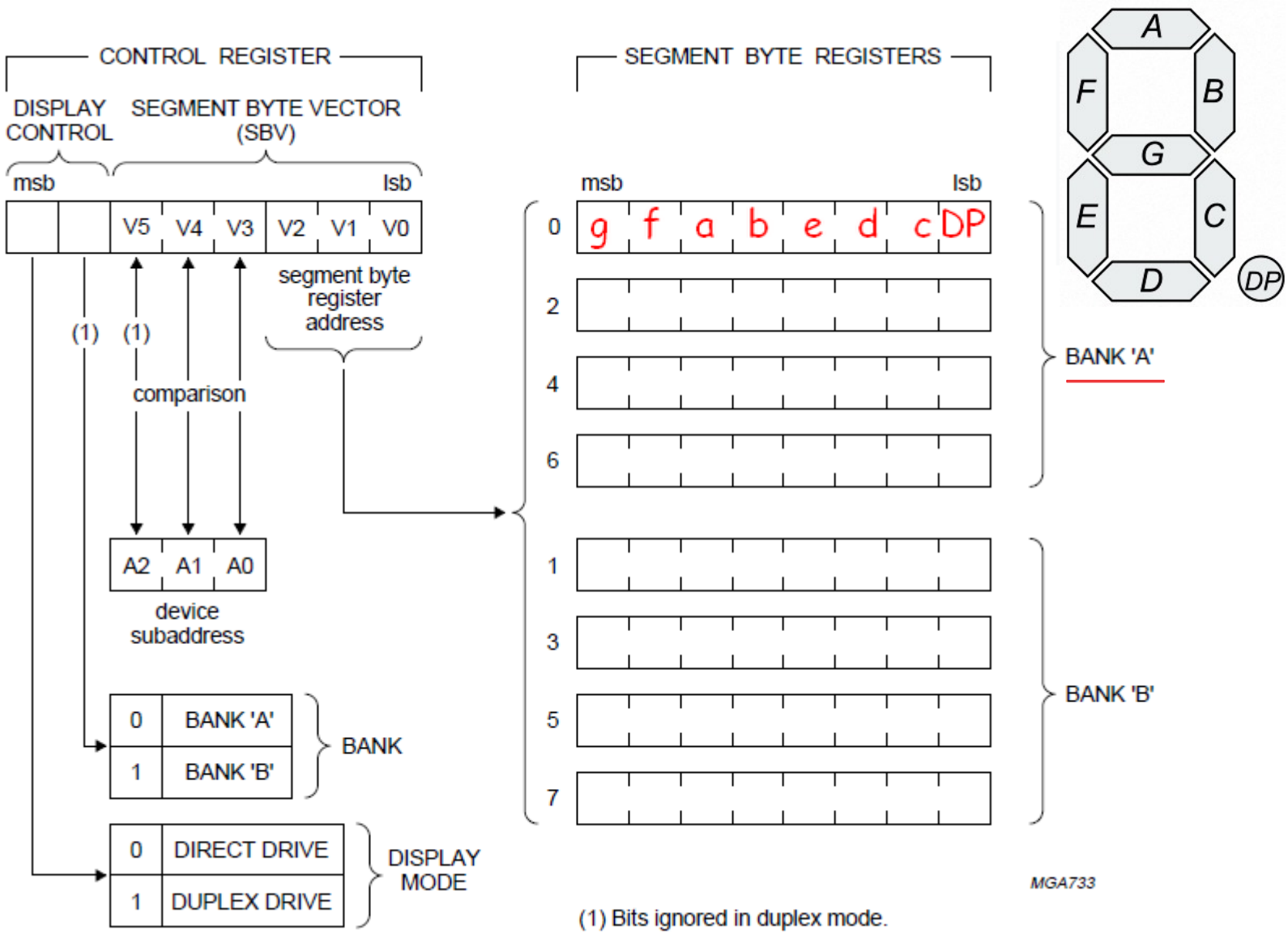
PCF8577 kommunikációs protokol

- I2C cím: 011 1010 (0x3A)



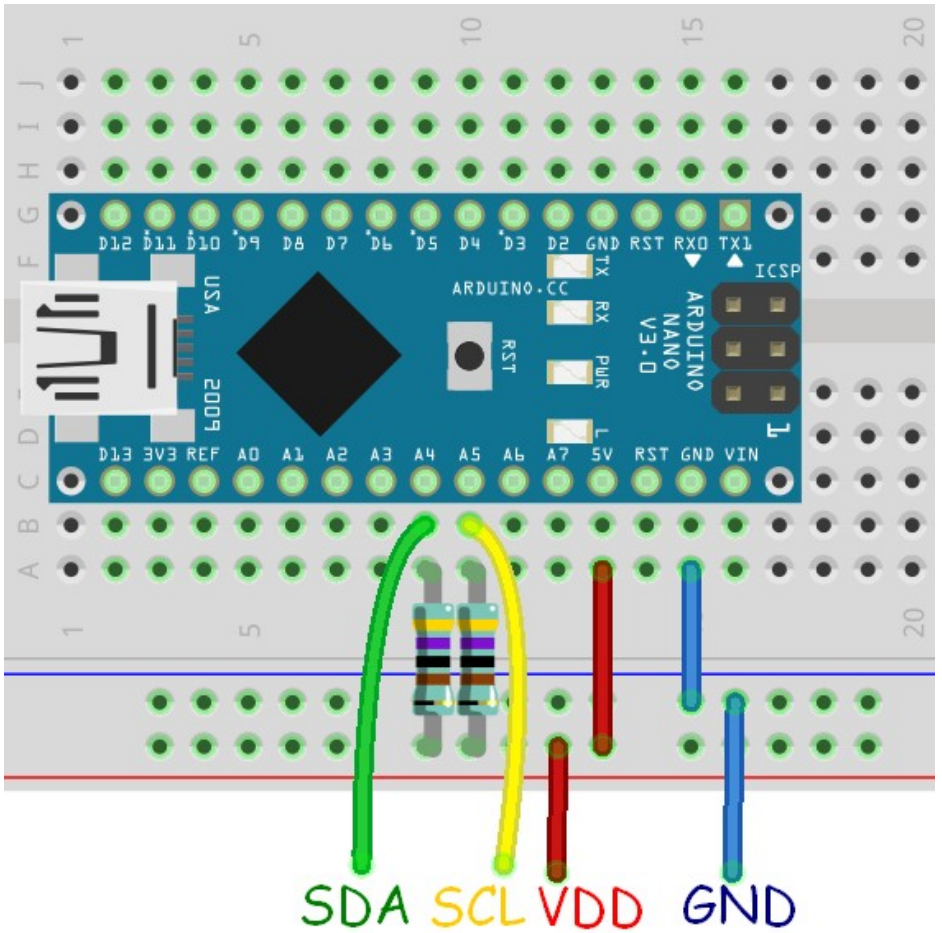
PCF8577C regiszterek szervezése

- Esetünkben MODE = 0, BANK =0, V2 = 0 (csak 4 adatregiszter kell)



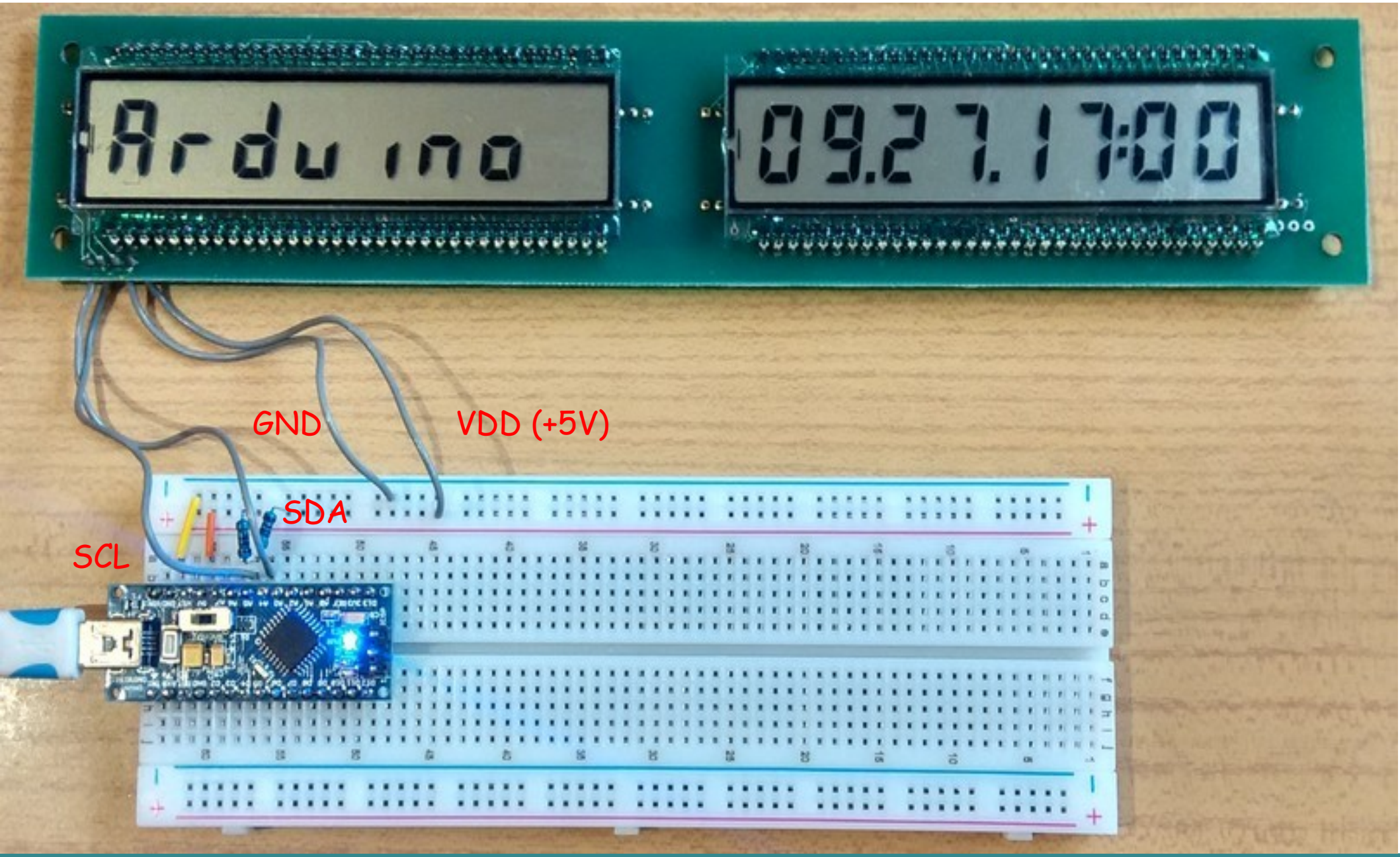
Az LCD panel vezérlése Arduinoval

- **Hardver összetevők:**
  - Az I2C vezérlésű LCD panel
  - Arduino nano v3.0 kártya
  - USB mini B kábel
  - 2 db 4,7 kΩ ellenállás
  - Számítógép USB csatlakozóval
  - Dugaszolós próbapanel



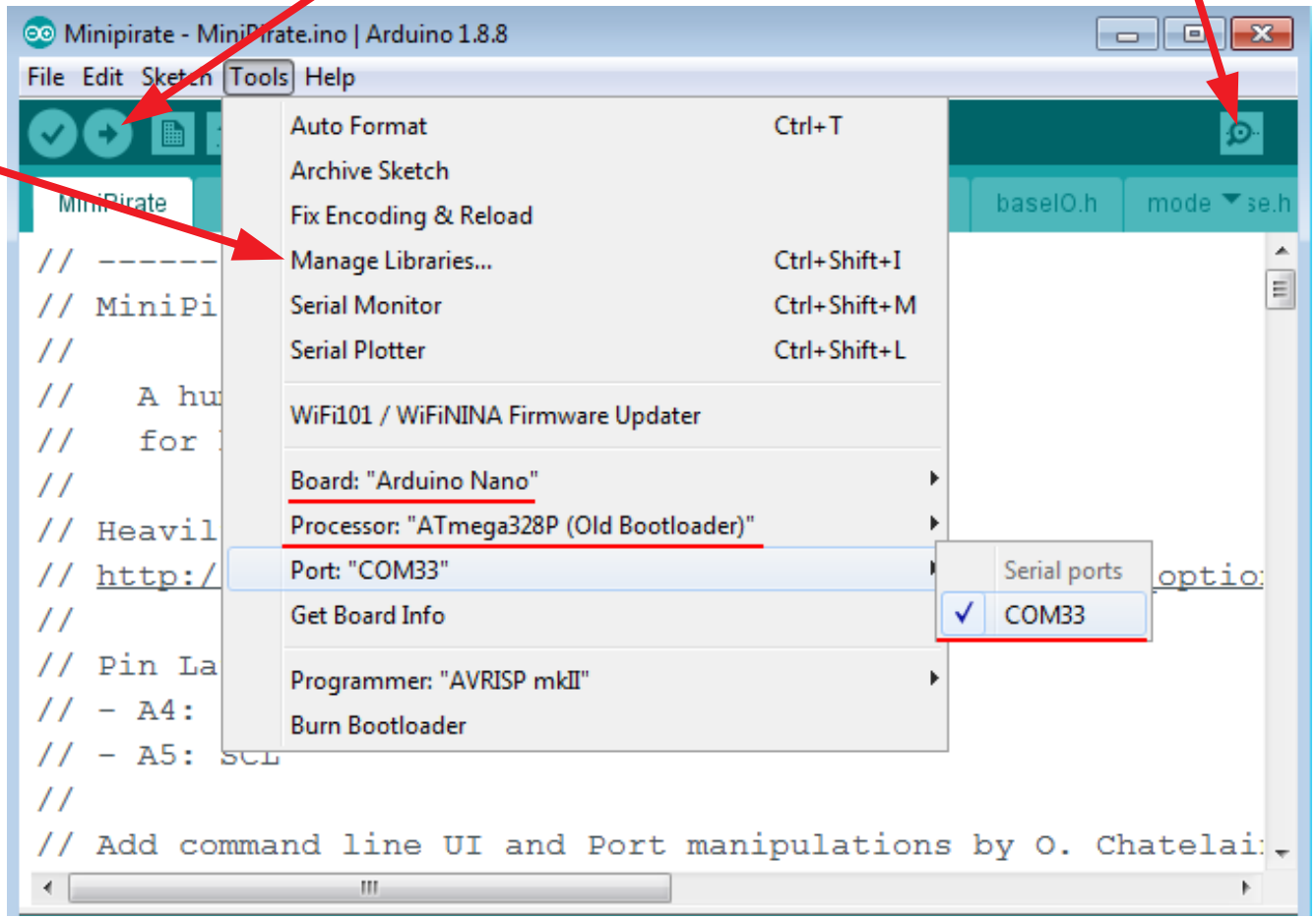
- **Működtető szoftver:**
  - Arduino IDE v1.8.8 (www.arduino.cc programletöltések)
  - Swisshandmade Mini Pirate v1.1.0 (github.com/chatelao/MiniPirate/releases)
  - Terminal emulator program (pl. az Arduino IDE beépített terminálja)

Az LCD panel vezérlése Arduinoval



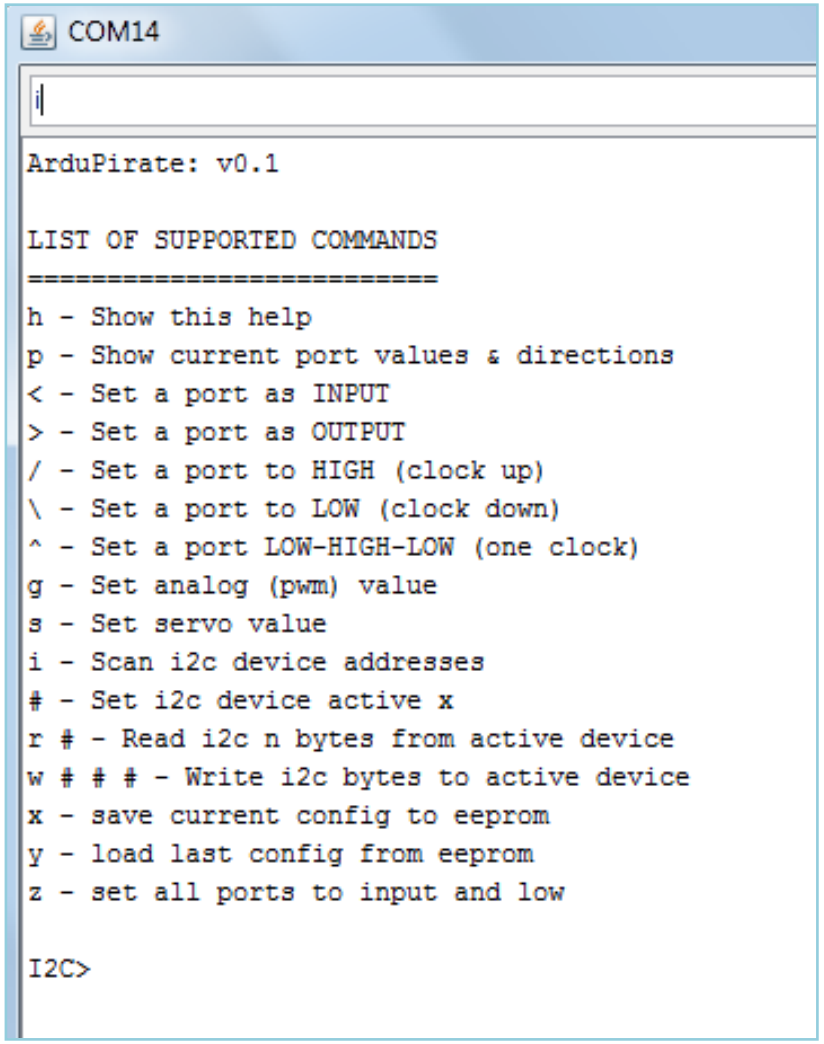
Arduino Mini Pirate előkészítése

- Beállítások:
  - Alappanel: Arduino nano/Atmega328
  - Soros port: ahová települ
- MiniPirate library telepítése itt
- Megnyitása a **File/ Examples** menüben
- Módosítás: `#include <Wire.h>` a helyes becsatolás
- Fordítás és letöltés
- Terminal ablak megnyitása



A Mini Pirate parancskészlete

- A terminal ablak beállításai: 9600 bit/s sebesség, „Soremelés” végjel



Help: A parancsok listája a **h** paranccsal íratható ki

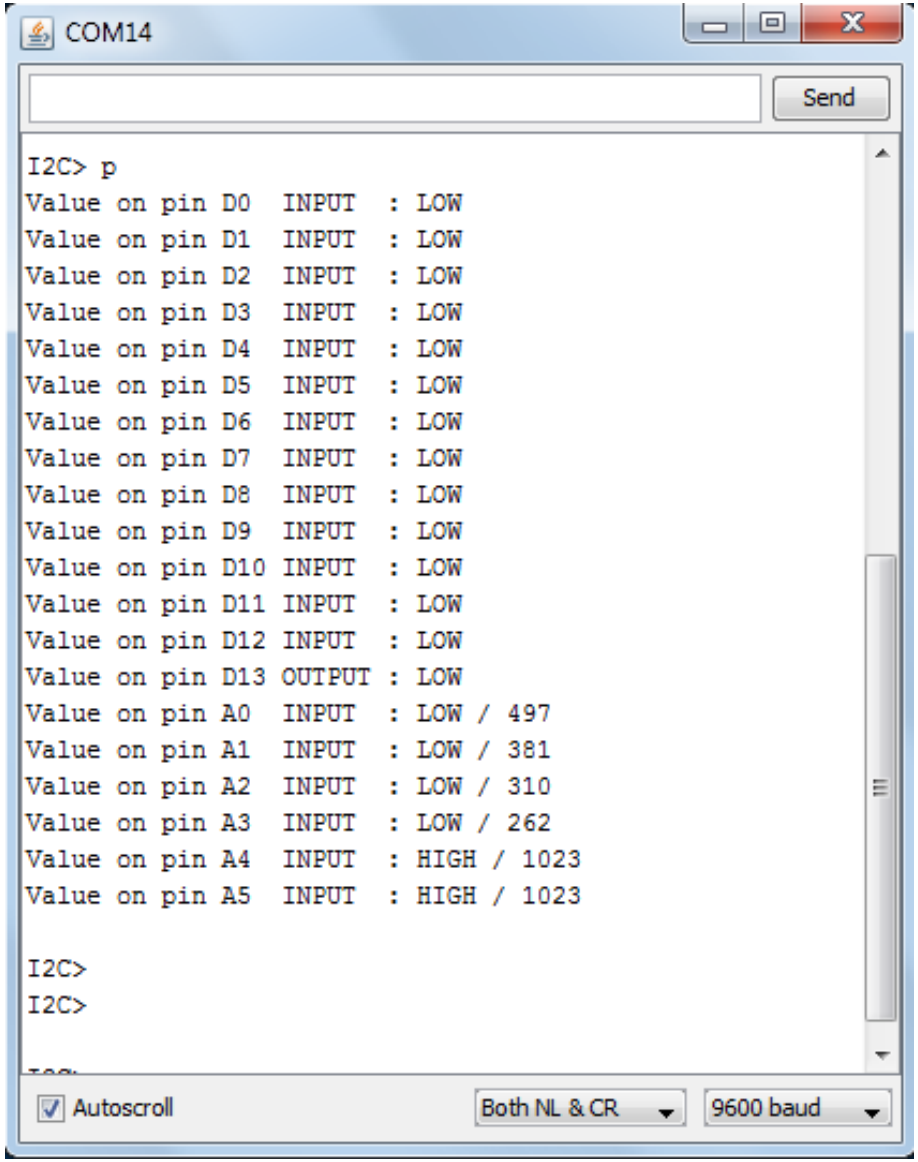
Megjegyzés: A **< > / \ ^** parancsok után egy sorszámot kell megadni (az Arduino kivezetések nevéből a **D** betűt ki kell hagyni).

Példa: A D13 lábra kötött LED bekapcsolása
>13/13 (D13 kimenet, D13 magas szintre)

Példa: A D13 lábra kötött LED kikapcsolása
\13 (D13 alacsony szintre)

Példa: A D11 kivezetésre kötött LED fényerejének változtatása PWM-mel.
>11g11 50 (D11 legyen kimenet, D11 kitöltés legyen 50%

A kivezetések állapotának lekérdezése



- A **p** paranccsal kiírathatjuk az összes kivezetés állapotát.
- Az **A0..A5** kivezetéseket digitális és analóg bemenetként is lekérdezi és kiírja.
- Az analóg értékek értelmezése:

  Feszültség = ADC * Vref/1023

Az I2C busz felderítése

- Az **i** parancs felderíti az I2C buszon található eszközöket és kilistázza azok címét.
- A kilistázott eszközök közül a sorszámmal választhatunk ki egyet
- Írás műveletnél (**w** parancs) csak a kiküldendő adatokat kell felsorolni. Például: **`> w0 127 126 50 18`**

```
I2C> i
SEARCHING I2C DEVICES...
========================
I2C devices found:
0: 0x3A - 0b00111010

I2C[0 - 0x3A] >
```

- Olvasásnál (**r** parancs) csak a beolvasandó adatok számát kell megadni. Például: **`> r5`**
- **ESETÜNKBEN AZ OLVASÁS NEM MŰKÖDIK! (A PC8577C ESZKÖZ CSAK ÍRHATÓ)**

Az LCD panel vezérlése

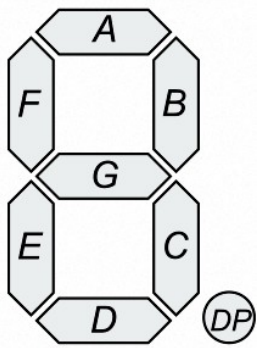
- A fényképen látható `Arduino  09.27.17:00` feliratot így írathatjuk ki:
- Számjegyek képe:

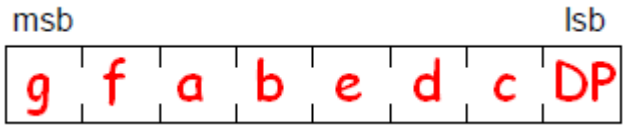
0 126
1 18
2 188
3 182
4 210
5 230
6 238
7 50
8 254
9 246

```
I2C[0 - 0x3A] > w16 0 142 138 2 14 158 136 250
Wrote 9 bytes to 0x3A

I2C[0 - 0x3A] >

I2C[0 - 0x3A] > w0 127 126 50 18 51 188 247 126
Wrote 9 bytes to 0x3A

I2C[0 - 0x3A] >
```

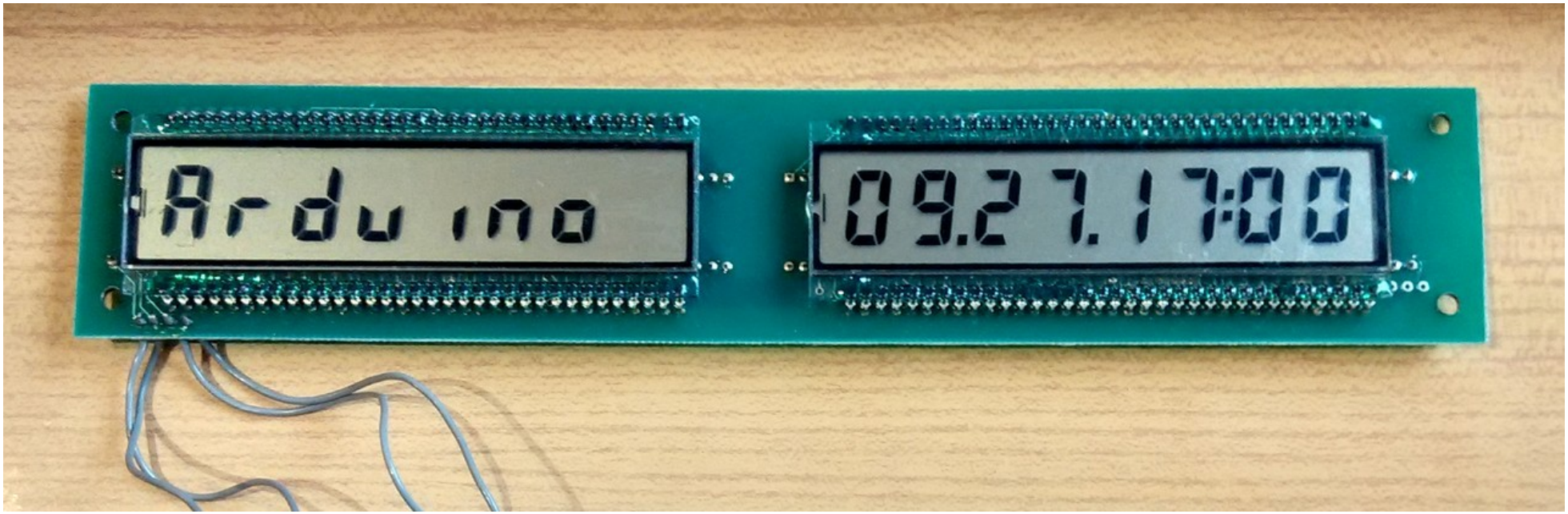
- Betűképek

A 250
r 136
d 158
u 14
i 2
n 138
o 142





A kijelző vezérlése saját programmal



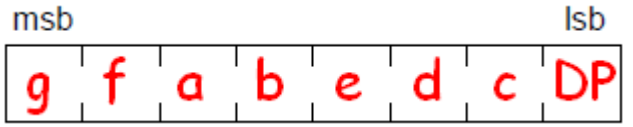
- A kijelző vezérlése az előző oldalon mutatott módon látványos, de kissé lassú és fáradságos…
- Azonban a kijezőt az **I2C** buszon keresztül saját programokkal is vezérelhetjük. A továbbiakban három példaprogramot mutatunk be:
  - **text_scroll** – szöveg görgetés a kijelzőn
  - **analog_display** – mérési eredmények kijelzése
  - **date_time** – egy **DS3231** RTC-ből kiolvasott dátum és idő kijelzése

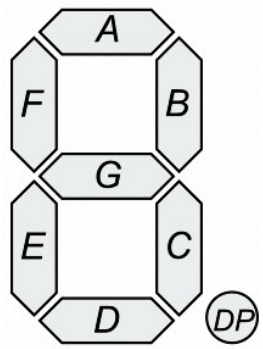
A Wire programkönyvtár

- Wire: előre telepített programkönyvtár az **I2C** busz kezeléséhez
- Az **I2C** kivezetések: **A4** (SDA) és **A5** (SCL)
- **Wire** függvények (a *Wire.h* állományt be kell csatolni!)
  Megjegyzés: csak a master mód használatát mutatjuk be
  - **Wire.begin()** - az I2C csatorna inicializálása
  - **Wire.SetClock(***frequency***)** – adatsebesség beállítása (100 000/400 000)
  - **Wire.beginTransmission(***address***)** - tranzakció indítása (START és címzés)
  - **Wire.write(***data,num***)** - adatbájt, string vagy adattönb kiküldése. Utóbbi esetben az adatbájtok számát is meg kell adni második paraméterként
  - **Wire.endTransmission()** - tranzakció vége (STOP)
  - **Wire.requestFrom(***address,num***)** - adatok lekérése (*num* a bájtok száma)
  - **Wire.available()** - a *read()*-del elővehető adatok számát adja meg
  - **Wire.read()** - a soron következő, már beolvasott bájt elővétele

text_scroll.ino

- Írjuk ki és görgessük jobbról balra az Arduino feliratot!

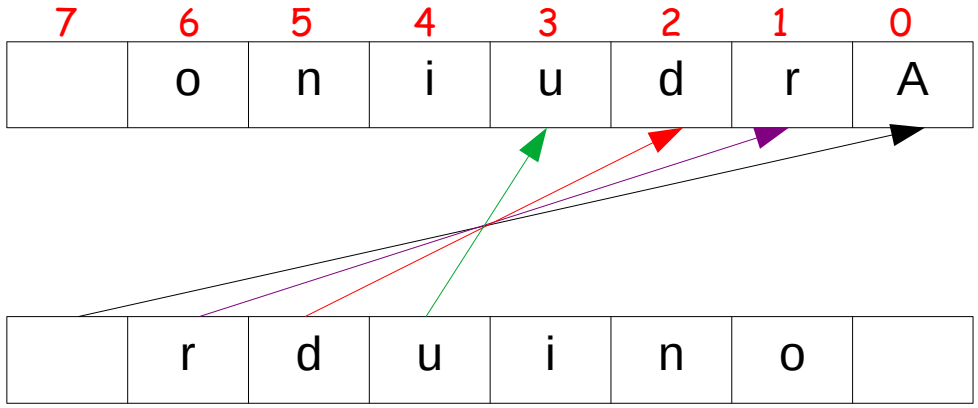


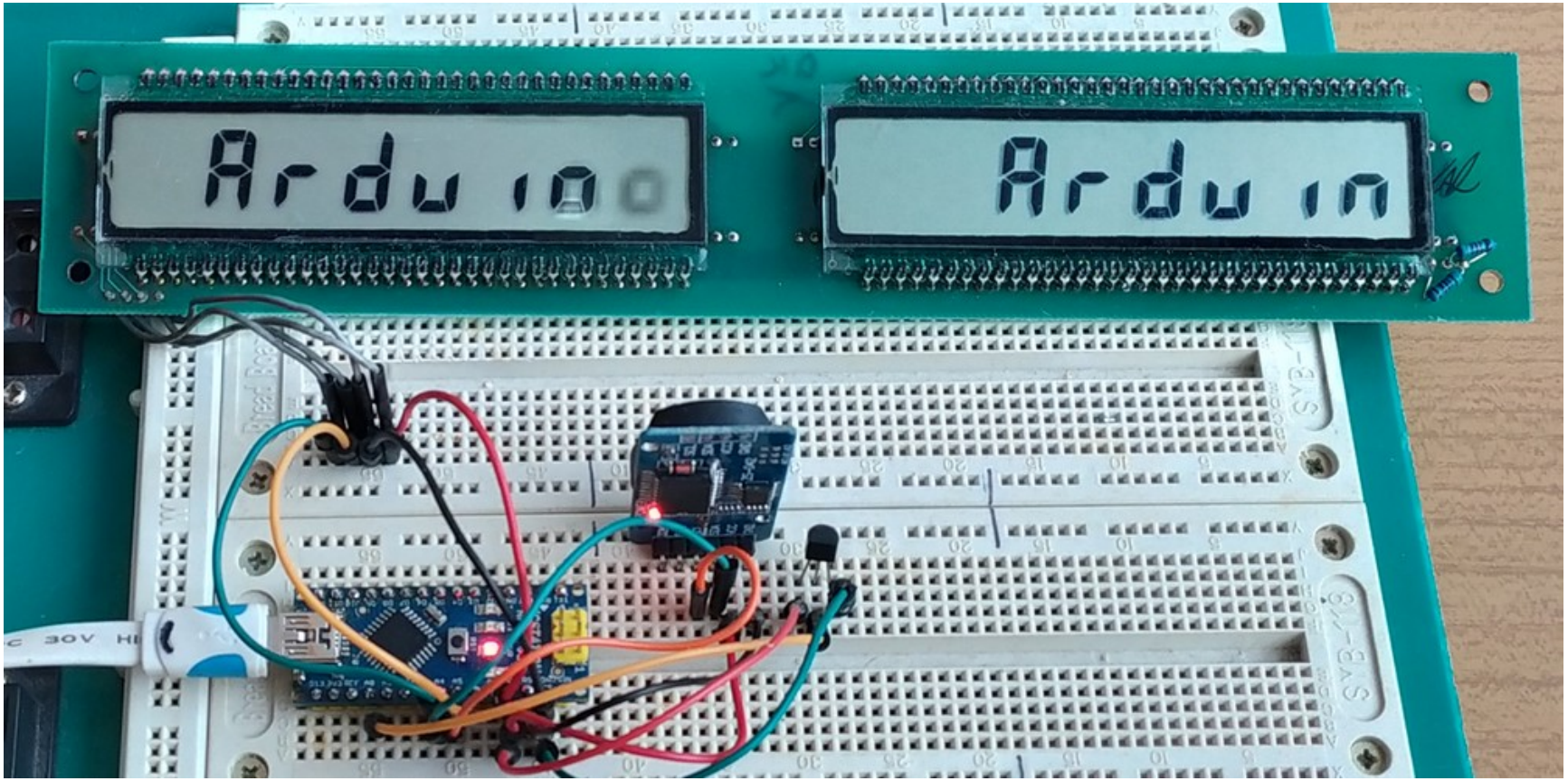



```
#include <Wire.h>

#define SLAVE_ADDRESS 0x3A
byte txt[10] = {250,136,158,14,2,138,142,0,0,0};   // Arduino felirat
byte data[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; // Display buffer

void setup() {
  Wire.begin();                 // Az I2C csatorna inicializálása
}

void loop() {
  for(int c=0; c<10; c++) {     // Ciklus fut txt[] elemein
    for(int i=15; i>0; i--) {   // Ciklus fut data[] elemein (fordított sorrend!!!)
      data[i] = data[i-1];      // Minden karakter balra lép
    }
    data[0] = txt[c];           // Következő karakter az utolsó helyre
    //--- A kijelző frissítése -----------
    Wire.beginTransmission(SLAVE_ADDRESS);
    Wire.write(0);              // regsztercím megadása (ide írunk)
    Wire.write(data, 16);       // A 16 adat kiírása egy tranzakcióban
    Wire.endTransmission();     // Kiírás vége
    delay(500);                 // Fél másodperc várakozás
  }
}
```

text_scroll.ino futási eredmény

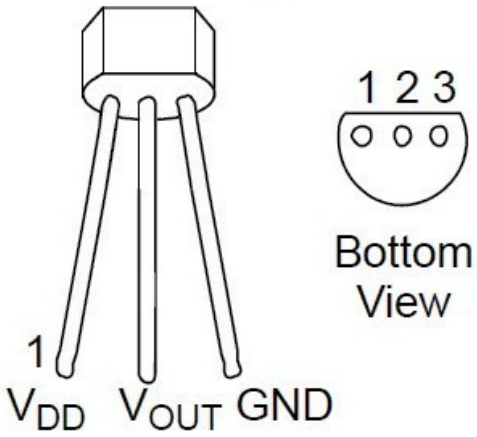
Miért kellett megfordítani az indexelést?
Azért, mert a kijelzőn fizikailag fordított a sorrend, így a **data[]** tömb indexelési sorrendjének megfordítása nélkül fordított sorrendben jelennének meg a betűk!

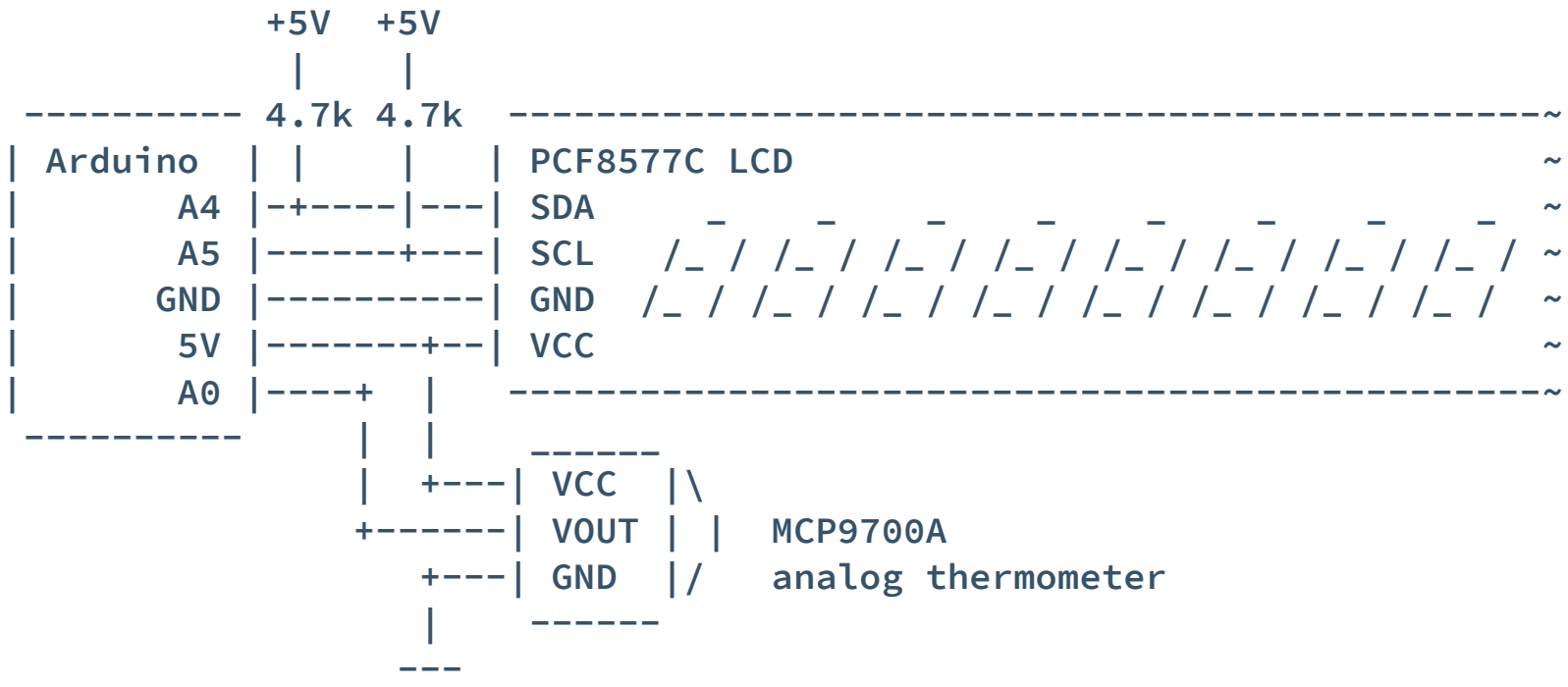




Analóg mérési eredmény megjelenítése

- Kössünk egy analóg szenzort, pl. egy **MCP9700A** hőmérőt az **A0** analóg bemenetre és jelezzük ki a mérési eredményt!
- A nagyobb felbontáshoz használjuk a belső, 1,1 V-os referenciát!
- Kapcsolási vázlat:





analog_display.ino

```
#include <Wire.h>
#define SLAVE_ADDRESS 0x3A
byte digits[10] = {126, 18, 188, 182, 210, 230, 238, 50, 254, 246}; // Számjegyek
byte data[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};   // Display buffer

void setup() {
  Wire.begin();                   // Az I2C csatorna inicializálása
  analogReference(INTERNAL);      // Belső 1,1 V-os referencia
}

void loop() {
  int a0 = analogRead(A0);        // ADC adat (0-1023)
  long mv = a0 * 1100 / 1024;     // millivolts (0-1100)
  long temp10 = mv - 500;         // tempC*10 (-500 - 600)
  clrScreen();                    // ata[] töm törlése
  outdec(a0,0,12);                // ADC adat kiírása bal szélen
  outdec(mv,0,8);                 // mért feszültség kiírása mV-ban
  outdec(temp10,1,2);             // hőmérséklet kiírása 1 tizedesre
  data[1] = 0xF0;                 // fok jel
  data[0] = 0x6C;                 // C betű
  //--- A kijelző frissítése -----------
  Wire.beginTransmission(SLAVE_ADDRESS);
  Wire.write(0);                  // regisztercím megadása (ide írunk)
  Wire.write(data, 16);           // A 16 adat kiírása egy tranzakcióban
  Wire.endTransmission();         // Kiírás vége
  delay(500);                     // Fél másodperc várakozás
}
```

analog_display.ino (folytatás)

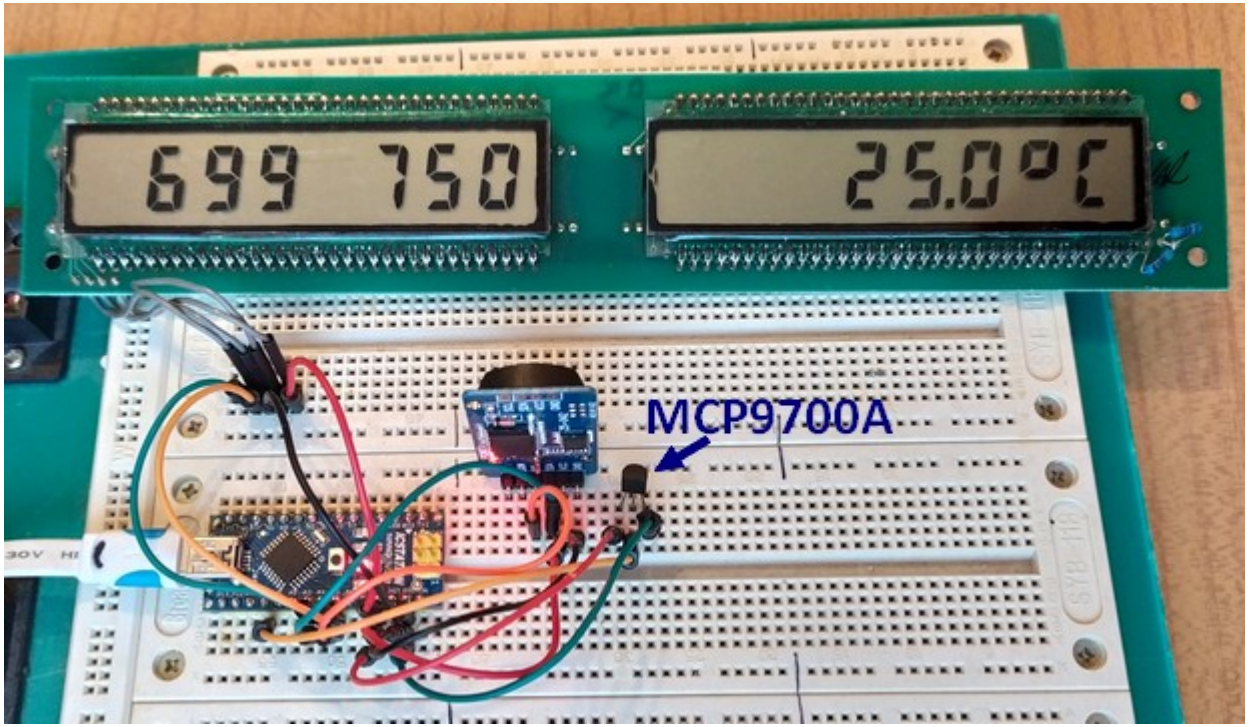
- Az előző oldalon meghívott függvények forráskódja:

```
/* A data[] tömb nullázása */
void clrScreen(void) {
  for(int i=0; i<16; data[i++]=0);
}

/** Decimális kiíratás adott számú tizedesjegyre.
   num a kiírandó szám (előjelesen)
   ndigits a kiírandó tizedesek száma
*/
void outdec(long num, byte ndigits, byte p) {
  unsigned int i, sign, dp;
  i = 0; sign = 0; dp = 0;
  if (num < 0) {
    sign = 128;
    num = -num;
  }
  do {
    data[p + i++] = digits[num % 10] | dp;
    num = num / 10;
    if (i == ndigits) dp = 1;
    else dp = 0;
  } while (num > 0);
  data[p + i++] = sign;
}
```

analog_display.ino futási eredménye

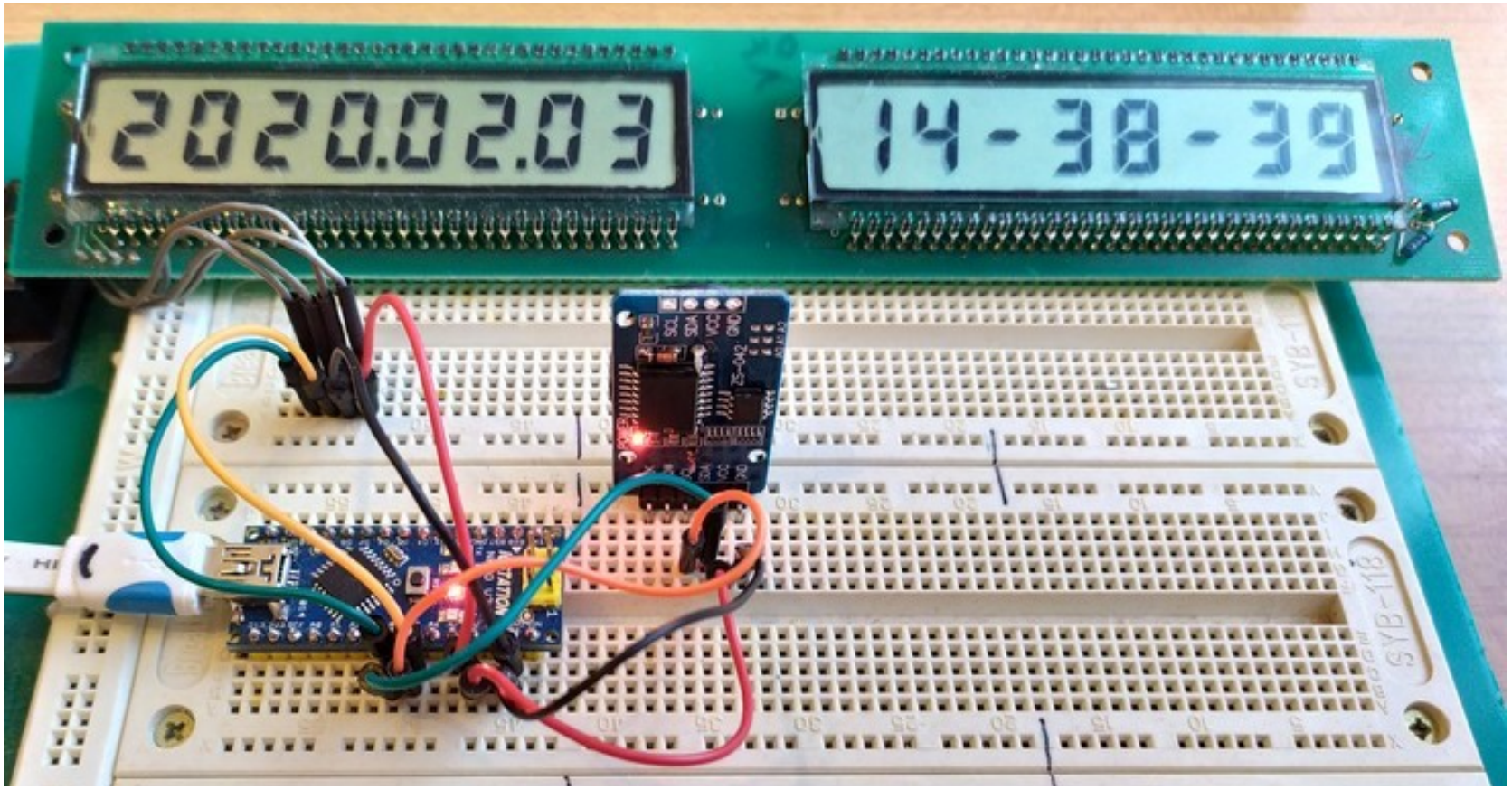
- A stabil kijelzés érdekében célszerű több mérést átlagolni.

```
long sum = 0;
for(int i=0; i<256; i++) sum += analogRead(A0);
long a0 = sum>>8;                    // ADC adat (0-1023)
long mv = a0 * 1100 / 1024;          // millivolts (0-1100)
. . .
```



Dátum és idő kijelzése

- Most két slave eszközt fűzünk fel az **I2C** buszra:
- **DS3231 RTC** – ebből kiolvassuk az aktuális időt és dátumot (feltételezzük, hogy az idő és a dátum már be van állítva!)
- **PCF8577C** vezérlőjű LCD kijelző – kijelzi a dátumot és az időt

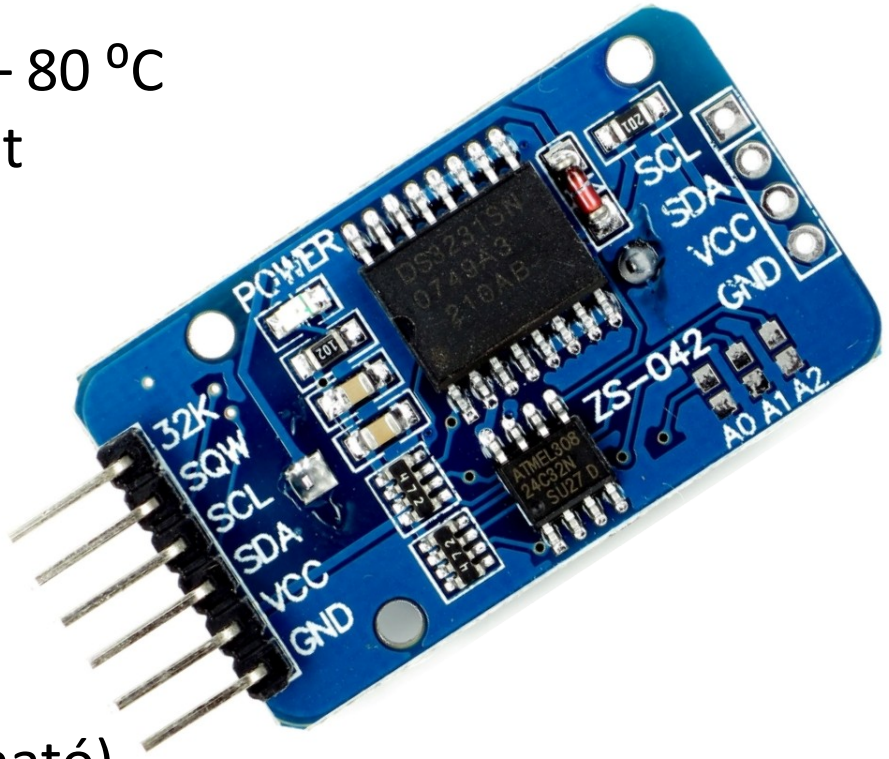


DS3231 Real-time óra modul

Oszcillátor, óra és kalendárium egy tokban. A tápfeszültség megszűnésekor az óra a modul hátoldalán elhelyezett telepről üzemel tovább.

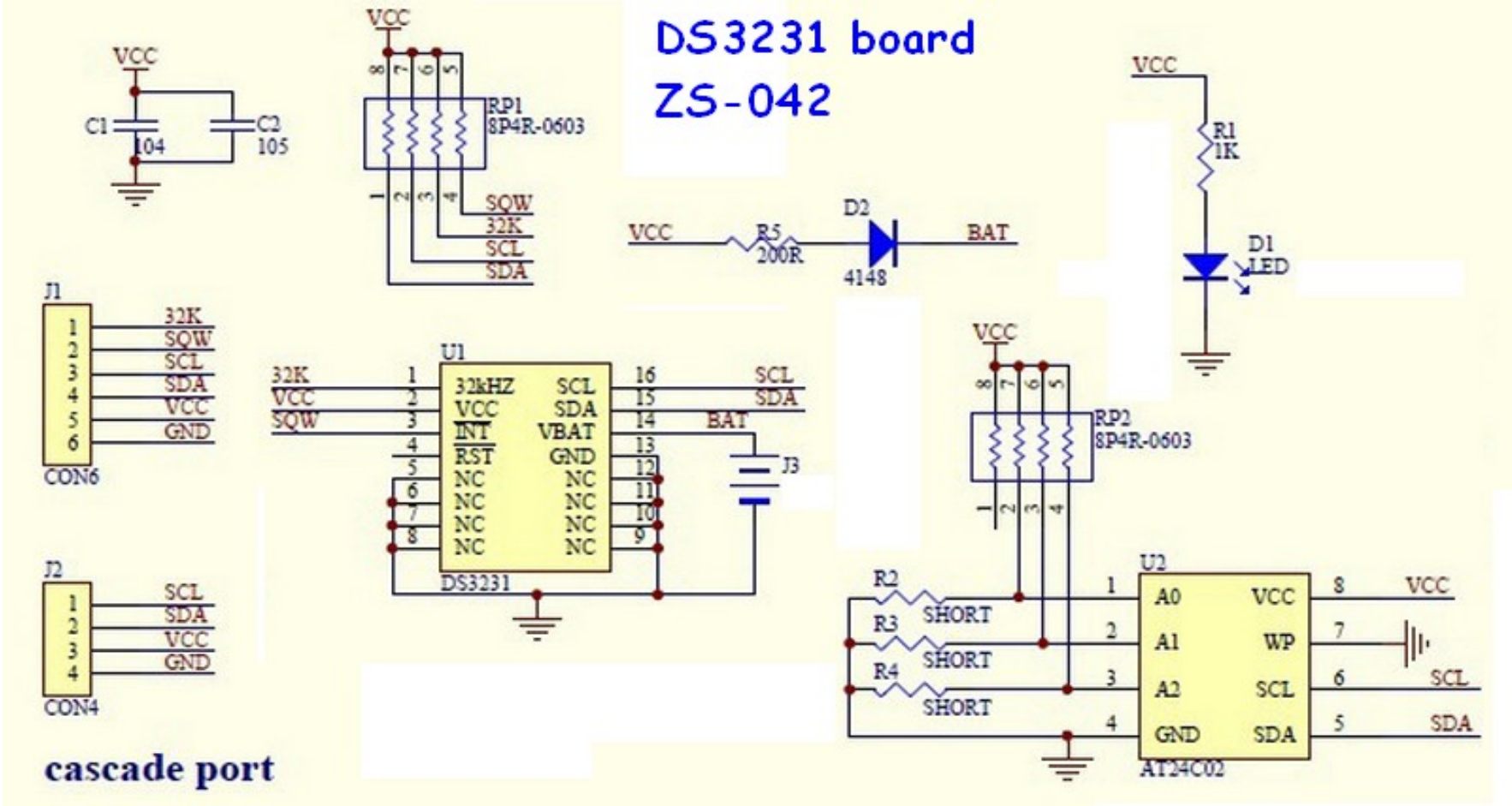
A modul többé-kevésbé cserekompatibilis a ds1307-tel, ebben is van 4 kB EEPROM, de:

- sokkal pontosabb az óra (±2 ppm a -40 – 80 °C tartományban) a beépített hőkompenzált oszcillátornak köszönhetően.
- két riasztási időpont is megadható
- van beépített hőmérője
- nincs belső RAM
- Vbat akár 5 V is lehet (4.2 V-os Li akkumulátorról is táplálható!)



EEPROM I2C címe: 0x57 (forrasztással állítható)
DS3231 I2C címe: 0x68

DS3231 kártya kapcsolási rajz



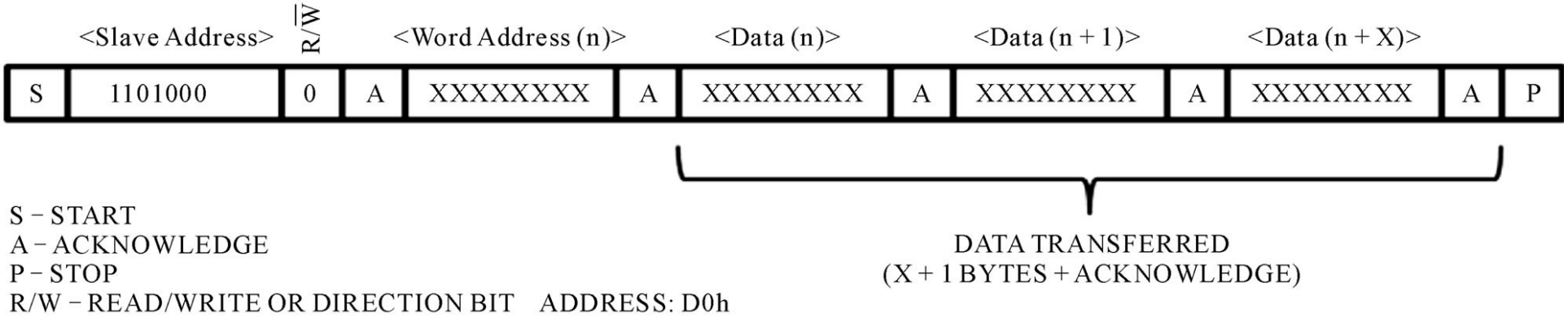
- Az **R5/D2** töltőáramkörnek csak akkor van értelme, ha akkumulátort (pl. **LiR2032**) használunk a **CR2032** elem helyett! Elemes táplálásnál célszerű megszakítani a töltőáramkört.
- A **D1** LED tulajdonképpen fölösleges.
- Az EEPROM címe (**A0, A1, A2**) forrasztással állítható.

A DS3231 IC regiszterei

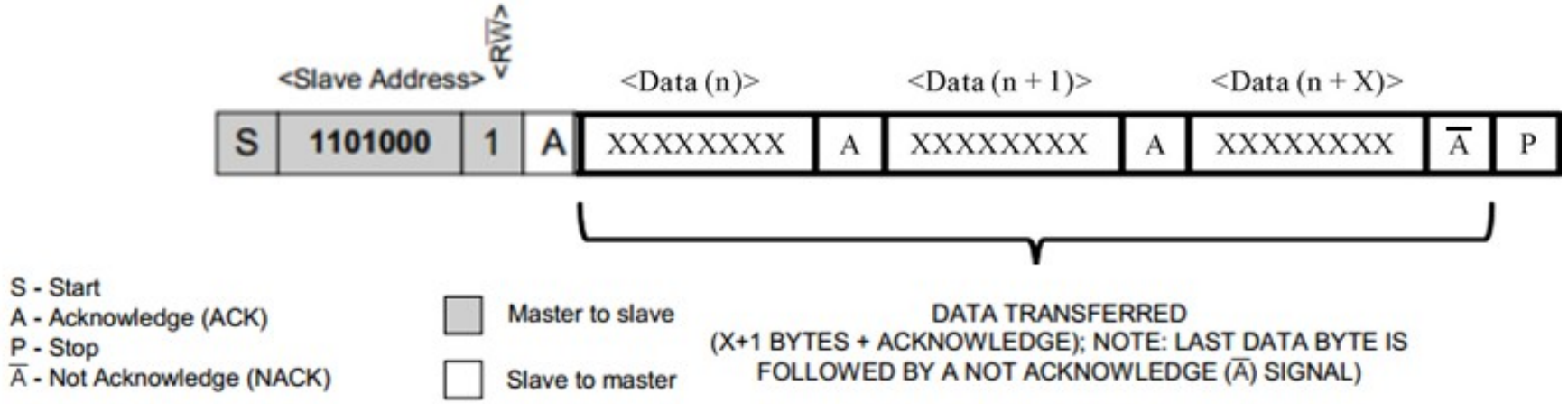
ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{AM}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59
09h	A1M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 1 Hours	1–12 + $\overline{AM}$/PM 00–23
0Ah	A1M4	DY/$\overline{DT}$	10 Date		Day				Alarm 1 Day	1–7
					Date				Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 2 Hours	1–12 + $\overline{AM}$/PM 00–23
0Dh	A2M4	DY/$\overline{DT}$	10 Date		Day				Alarm 2 Day	1–7
					Date				Alarm 2 Date	1–31
0Eh	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—
0Fh	OSF	0	0	0	EN32kHz	BSY	A2F	A1F	Control/Status	—
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	—
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	MSB of Temp	—
12h	DATA	DATA	0	0	0	0	0	0	LSB of Temp	—

DS3231 I2C kommunikáció

Íráskor megcímezzük az eszközt (0x68), az R/W bit pedig = 0.
A második bájt a regiszter (vagy memória) cím. A többi bájt a kiküldendő adat (a cím automatikusan inkrementálódik).



Olvasáskor megcímezzük az eszközt (0x68), az R/W bit pedig = 1.
A többi bájt a beolvasott adat (a cím automatikusan inkrementálódik).



date_time.ino

```
/* Kapcsolás:
     Arduino        DS3231 RTC        PCF8577C LCD 2x8 character 7-segment display
    ----------     ------------     ------------------------------------------~
   |        A4 |--| SDA     SDA |--| SDA      _    _    _    _    _    _    _   ~
   |        A5 |--| SCL     SCL |--| SCL    /_ / /_ / /_ / /_ / /_ / /_ / /_ / /_ / ~
   |       GND |--| GND     GND |--| GND   /_ /./_ /./_ /./_ /./_ /./_ /./_ /./_ /  ~
   |        5V |--| VCC     VCC |--| VCC                                            ~
    ----------     ------------     ------------------------------------------~  */
#include <Wire.h>
#define RTC_ADDRESS 0x68                    // A DS3231 RTC I2C címe
#define LCD_ADDRESS 0x3A                    // A PCF8577 vezérlő I2C címe

// A szegmensek bitsorrendje: g,f,a,b,e,d,c,DP (<-ez a legalacsonyabb helyiérték)
char digits[10] = {126, 18, 188, 182, 210, 230, 238, 50, 254, 246};
char lcd_data[16];                          // Buffer (itt állítjuk össze a képet)
char timeDate[16];                          // Ide töltjük a fogadott adatokat

void setup() {
  Wire.begin();                             // I2C csatorna inicializálása
}

void loop() {
  I2C_burstRead(RTC_ADDRESS,0,7,timeDate);  // Dátum és idő beolvasása
  displayDateTime();                        // A kiírandó kép összeállítása
  I2C_burstWrite(LCD_ADDRESS,0,16,lcd_data); // A kijelzés frissítése
  delay(500);                               // várunk fél másodpercet
}
```

date_time.ino (folytatás)

- A beolvasást írással kell kezdeni, be kell állítani azt a regisztercímet (esetünkben a nullát), ahonnan az az első adatot olvassuk
- Adatbeolvasáskor a slave eszköz automatikusan lépteti a címet
- A kijelzőre írás ugyanúgy történik, mint az előzőekben

```
//--- A dátum és az idő beolvasása az RTC-ből --------
void I2C_burstRead(char saddr, char maddr, int size, char* data) {
  Wire.beginTransmission(saddr);              // Start feltétel, slave address küldés
  Wire.write(maddr);                          // Memóriacím kiküldése
  Wire.endTransmission();                     // Írás vége
  Wire.requestFrom(saddr, size);              // Restart és adatok beolvasása
  for (int i = 0; i < size; i++) {            // Az adatok eltárolása
    data[i] = Wire.read();                    // Az beolvasott adatok eltárolása
  }
}

//--- A dátum és az idő kiírása az LCD kijelzőn ------
void I2C_burstWrite(char saddr, char maddr, int size, char* data) {
  Wire.beginTransmission(saddr);              // Start feltétel, slave address küldés
  Wire.write(maddr);                          // Memóriacím kiküldése
  Wire.write(data,16);                        // A szegmensvezérlő adatok kiírása
  Wire.endTransmission();
}
```

date_time.ino (folytatás)

- A dátum és az idő kiírásából ne feledjük, hogy az adatokat BCD kódolással kapjuk!
- Ügyeljünk a számunkra fölösleges jelzőbitek kimaszkolására!

```
//--- A dátum és az idő kiírása az lcd_data[] tömbbe ------
void displayDateTime() {
  lcd_data[0] = digits[timeDate[0] & 0x0f];         // Másodpercek
  lcd_data[1] = digits[(timeDate[0] >> 4) & 0x07]; // Tízmásodpercek
  lcd_data[2] = 128;                                // '-'
  lcd_data[3] = digits[timeDate[1] & 0x0f];         // Percek
  lcd_data[4] = digits[(timeDate[1] >> 4) & 0x07]; // Tízpercek
  lcd_data[5] = 128;                                // '-'
  lcd_data[6] = digits[timeDate[2] & 0x0f];         // Órák
  lcd_data[7] = digits[(timeDate[2] >> 4) & 0x03]; // Tízórák
  lcd_data[8] = digits[timeDate[4] & 0x0f];         // Napok
  lcd_data[9] = digits[(timeDate[4] >> 4) & 0x03]; // Tíznapok
  lcd_data[10] = digits[timeDate[5] & 0x0f] | 1;    // Hónapok + DP
  lcd_data[11] = digits[(timeDate[5] >> 4) & 0x01];// Tízhónapok
  lcd_data[12] = digits[timeDate[6] & 0x0f] | 1;    // Évek + DP
  lcd_data[13] = digits[(timeDate[6] >> 4) & 0x0f];// Évtizedek
  lcd_data[14] = digits[0];                         // Évszázad = 20xx
  lcd_data[15] = digits[2];
}
```

rtc_baremetal.ino

- Ezzel a programmal egyszerű módon beállíthatjuk az órát

```
#include "Wire.h"
#define RTC_ADDRESS 0x68
char daysOfTheWeek[7][12] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday","Sunday"};

void setup() {
  Wire.begin();
  Serial.begin(9600);
  //--- set time as: seconds, minutes, hours, day, date, month, year
  setTime(00,15,15,1,3,2,20);         // 2020-FEB-03, 15:15:00
}

void loop() {
  displayTime();                      // display time/date
  delay(1000);                        // every second
}

byte decToBcd(byte val) {             // Convert decimal numbers to BCD
  return( (val/10*16) + (val%10) );
}

byte bcdToDec(byte val) {             // Convert BCD numbers to decimal
  return( (val/16*10) + (val%16) );
}
```

rtc_baremetal.ino (folytatás)

```
void setTime(byte second, byte minute, byte hour,
     byte dayOfWeek, byte dayOfMonth, byte month, byte year) {
  Wire.beginTransmission(RTC_ADDRESS);
  Wire.write(0);                        // set register pointer to 00h
  Wire.write(decToBcd(second));         // set seconds
  Wire.write(decToBcd(minute));         // set minutes
  Wire.write(decToBcd(hour));           // set hours
  Wire.write(decToBcd(dayOfWeek));      // set day of week (1=Sunday, 7=Saturday)
  Wire.write(decToBcd(dayOfMonth));     // set date (1 to 31)
  Wire.write(decToBcd(month));          // set month
  Wire.write(decToBcd(year));           // set year (0 to 99)
  Wire.endTransmission();
}
void getTime(byte *second, byte *minute, byte *hour,
     byte *dayOfWeek, byte *dayOfMonth, byte *month, byte *year) {
  Wire.beginTransmission(RTC_ADDRESS);
  Wire.write(0);                        // set register pointer to 00h
  Wire.endTransmission();
  Wire.requestFrom(RTC_ADDRESS, 7);  // request 7 bytes starting from register 00h
  *second = bcdToDec(Wire.read() & 0x7f);
  *minute = bcdToDec(Wire.read());
  *hour = bcdToDec(Wire.read() & 0x3f);
  *dayOfWeek = bcdToDec(Wire.read());
  *dayOfMonth = bcdToDec(Wire.read());
  *month = bcdToDec(Wire.read());
  *year = bcdToDec(Wire.read());
}
```

A DS3231 RTC inicializálása

```
//--- Set RTC time/date ------------------------------------
void setTime(byte second, byte minute, byte hour,
     byte dayOfWeek, byte dayOfMonth, byte month, byte year) {
  Wire.beginTransmission(RTC_ADDRESS);
  Wire.write(0);                          // set register pointer to 00h
  Wire.write(decToBcd(second));           // set seconds
  Wire.write(decToBcd(minute));           // set minutes
  Wire.write(decToBcd(hour));             // set hours  (24 hours mode)
  Wire.write(decToBcd(dayOfWeek));        // set day of week (1=Monday, 7=Sunday)
  Wire.write(decToBcd(dayOfMonth));       // set date (1 to 31)
  Wire.write(decToBcd(month));            // set month
  Wire.write(decToBcd(year));             // set year past 2020 (0 to 99)
  Wire.endTransmission();
}
```

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{\text{AM}}$/PM / 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{\text{AM}}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99

rtc_baremetal.ino (folytatás)

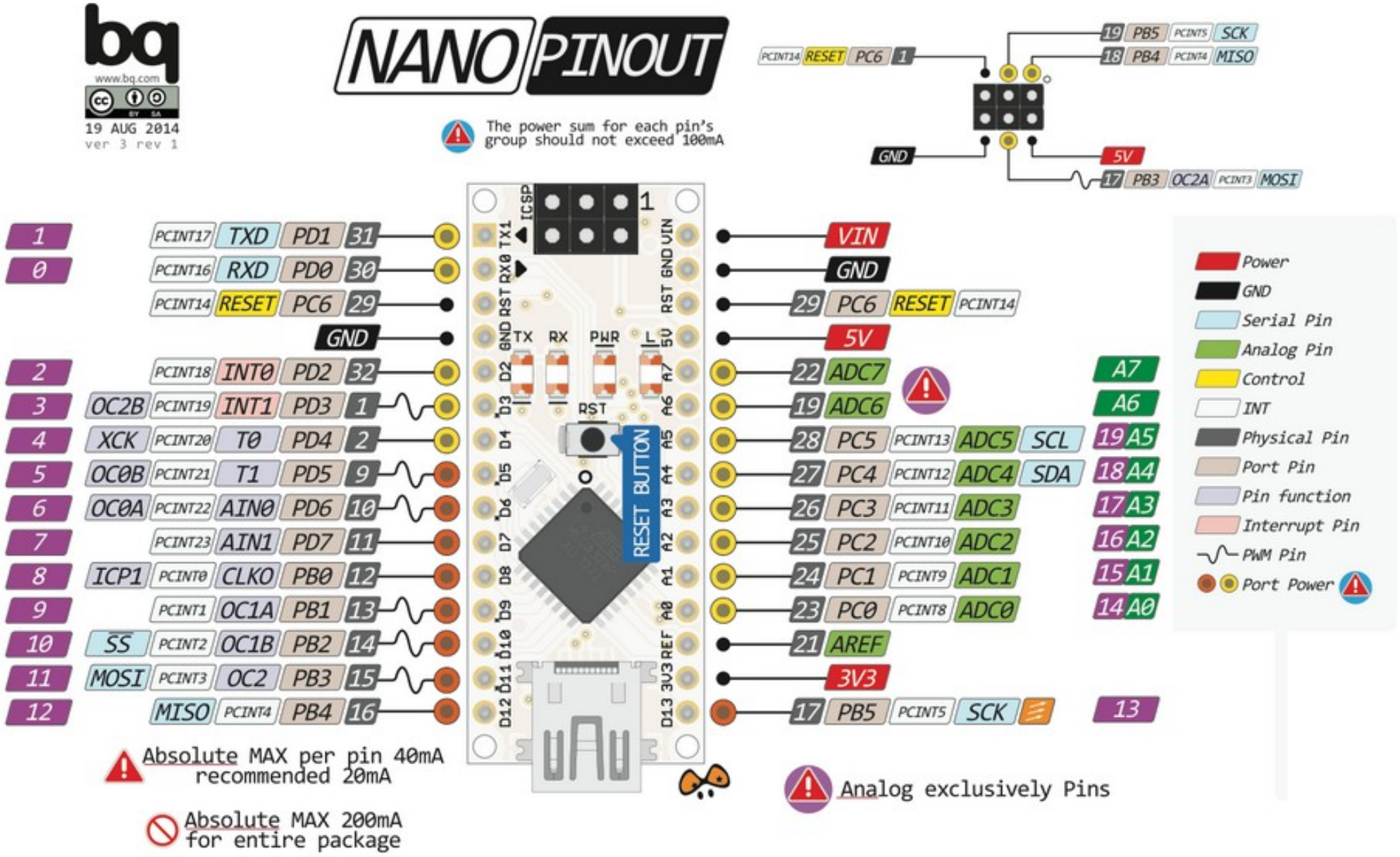
```
void displayTime() {
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
  // retrieve data from DS3231
  getTime(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);
  Serial.print(hour, DEC);
  Serial.print(":");
  if (minute < 10) Serial.print("0");
  Serial.print(minute, DEC);
  Serial.print(":");
  if (second < 10) Serial.print("0");
  Serial.print(second, DEC);
  Serial.print(" ");
  Serial.print(year+2000, DEC);
  Serial.print("/");
  Serial.print(month, DEC);
  Serial.print("/");
  Serial.print(dayOfMonth, DEC);
  Serial.print(" ");
  Serial.println(daysOfTheWeek[dayOfWeek-1]);
}
```

RTC-ben 1: Monday, de a daysOfTheWeek tömbben 0

```
17:51:33 2020/2/5 Wednesday
17:51:34 2020/2/5 Wednesday
17:51:35 2020/2/5 Wednesday
17:51:36 2020/2/5 Wednesday
17:51:37 2020/2/5 Wednesday
```

Futtatási példa

Az Arduino nano kártya kivezetései



Ellenállás színkódok