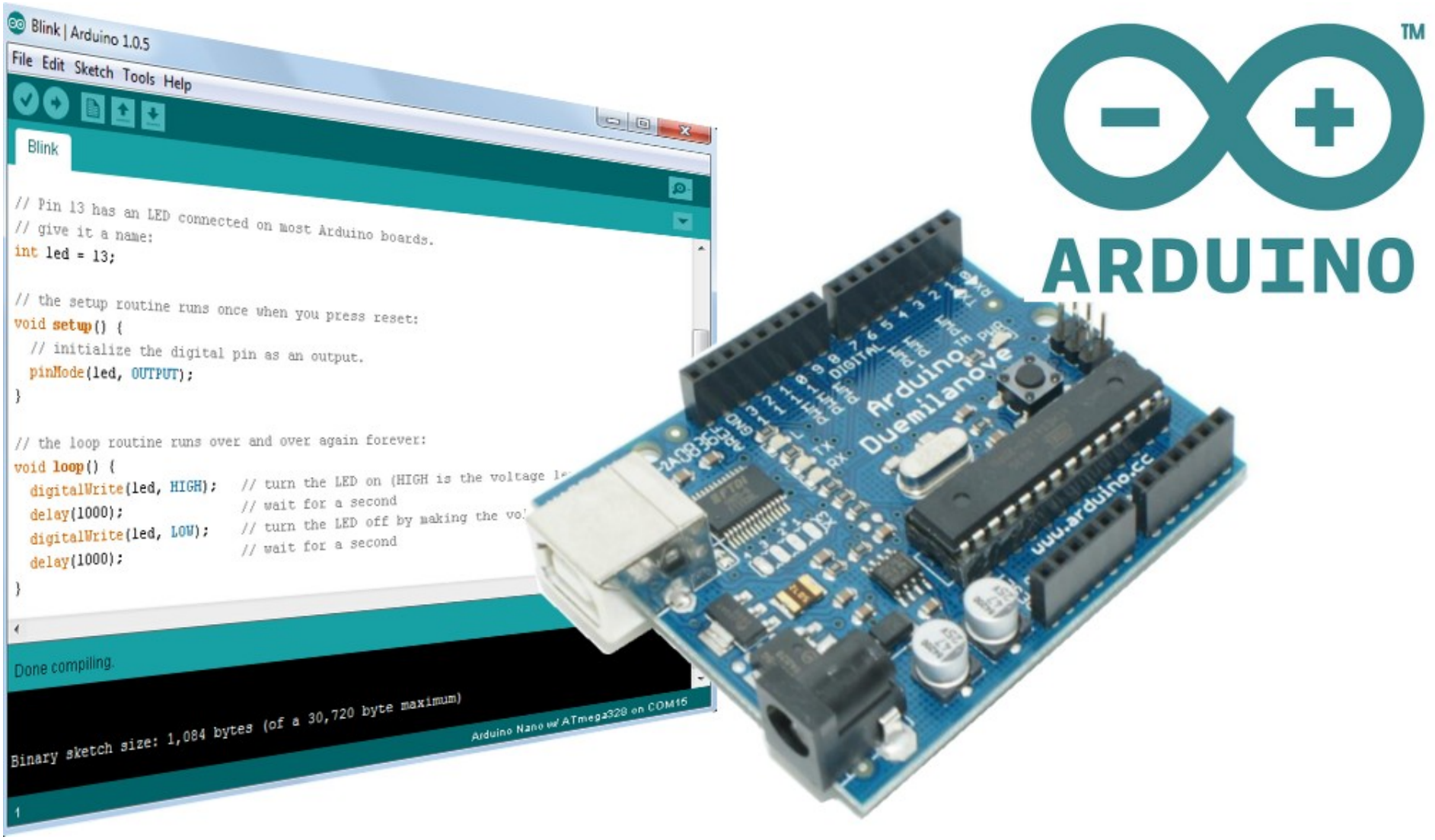
Arduino tanfolyam kezdőknek és haladóknak



17. Véges állapotgépek, ébresztőóra projekt

Ajánlott olvasmányok

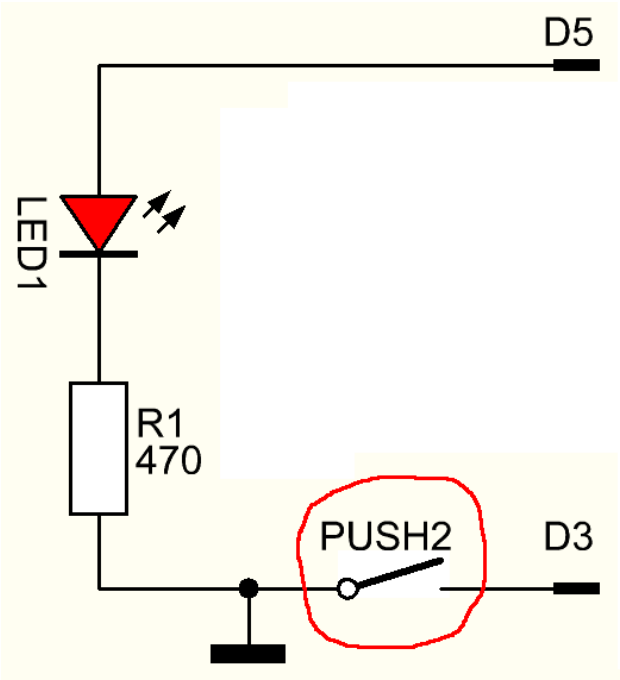
- **TM1637 4-digites kijelző:**
  - Titan Microelectronics: TM1637 adatlap
  - Hétszegmenses LED kijelzők 2. rész (2019. május 2.)
    előadásvázlat, mintaprogramok
  - Hétszegmenses kijelző alkalmazások (2019. május 16.)
    előadásvázlat mintaprogramok
- **DS3231 RTC modul:**
  - Maxim Integrated: DS3231 adatlap
  - A DS1307 és DS3231 valósidejű órák használata (2016. március 10.)
    előadásvázlat, mintaprogramok
- **ST7735 kijelző modul:**
  - Sitronix **ST7735 adatlap**
  - TFT library reference: arduino.cc/en/Reference/TFTLibrary

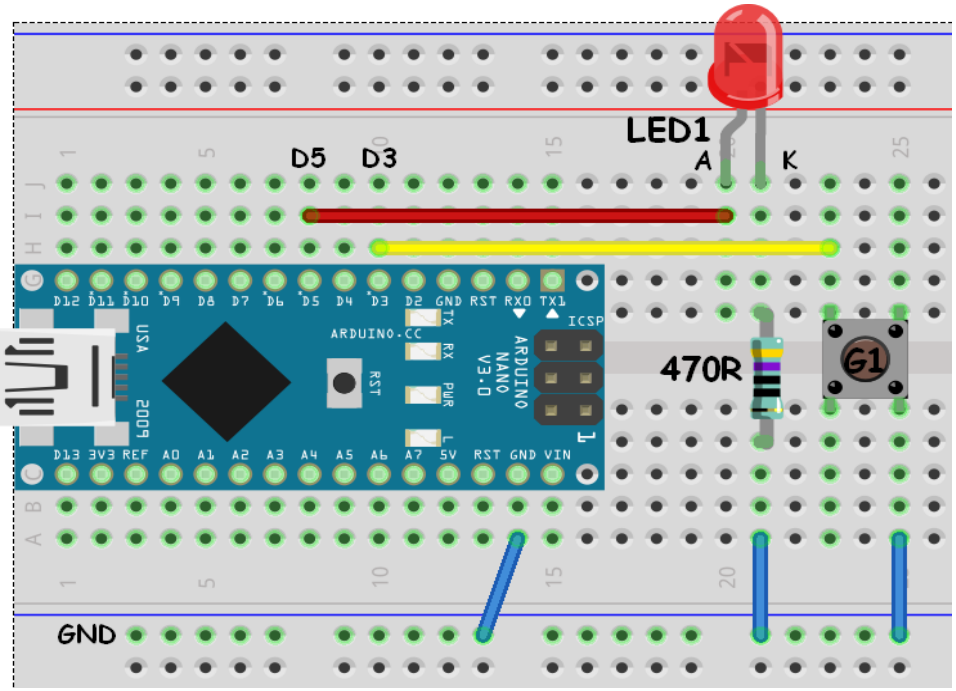
LED ki/bekapcsolása nyomógombbal

- Ennél az egyszerű kapcsolásnál oldjuk meg az alábbi feladatot:
  - Első (és minden páratlan) lenyomásra a LED világítson!
  - Második (és minden páros) lenyomásra a LED ne világítson!
- A legrosszabb, amit megtehetünk, ha így fogunk hozzá:

```
if (digitalRead(PUSH2) == LOW) {
    digitalWrite(RED_LED, !digitalRead(RED_LED));
}
```

Miért rossz ez a megközelítés?





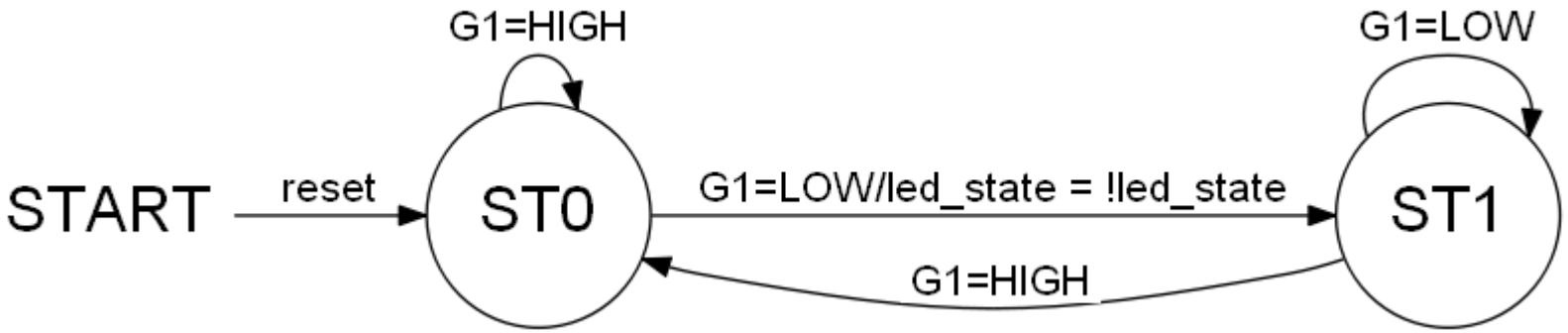
Első változat: led_switch_v1.ino

- Első változat blokkoló várakozásokkal és pergésmentesítő késleltetéssel
- A blokkoló várakozás nem mindig megengedhető, ilyenkor jön képbe például a véges állapotgép módszer, ahol állapotjelző változók tartják nyilván, hogy hol tartunk és adok feltételek esetén lépünk új állapotba

```
#define RED_LED 5                 // D5-re van kötve a LED
#define G1      3                 // D3-ra van kötve a nyomógomb
boolean led_state = false;        // LED állapot tárolója

void setup() {
  pinMode(RED_LED,OUTPUT);        // D5 legyen digitális kimenet
  pinMode(G1,INPUT_PULLUP);       // D3 bemenet, belső felhúzással
}                                 // alaphelyzetben HIGH állapotban lesz

void loop() {
  while(digitalRead(G1));         // Vár, míg D3 magas állapotban van
  led_state = !led_state;         // Lenyomáskor LED állapotváltás
  digitalWrite(RED_LED,led_state); // és kijelzés
  delay(20);                      // Pergésmentesítő várakozás
  while(!digitalRead(G1));        // Vár, amíg D3 LOW állapotba van
  delay(20);                      // Pergésmentesítő várakozás
}
```

Mi az állapotgép?

- A determinisztikus véges állapotgép a pillanatnyi állapottól és a beérkező jeltől (vagy bekövetkező eseménytől) függően lép a következő állapotba
- Ilyen állapotgépet valósít meg a következő programunknak azon része is, ami a G1 nyomógomb állapotait kezeli



- A nyomógomb két állapotából a lenyomás, illetve a felengedés eseménye viszi új állapotba az állapotgépet
- A LED állapotát átkapcsoló tevékenységet a lenyomás hatására bekövetkező állapotváltáskor kell elvégezni

Állapotjelzők és események

- **A nyomógomb állapotai:**
  - Felengedett állapot (azaz lenyomásra várunk)
  - Lenyomott állapot (azaz felengedésre várunk)
- A nyomógomb állapotát a `waitforpress` nevű logikai változóban tároljuk. Legyen `waitforpress = true` ha lenyomásra várunk, s legyen `waitforpress = false` , ha felengedésre várunk!

- **A LED állapotai:**
  `led_state = true` a bekapcsolt állapot, s `false` a kikapcsolt

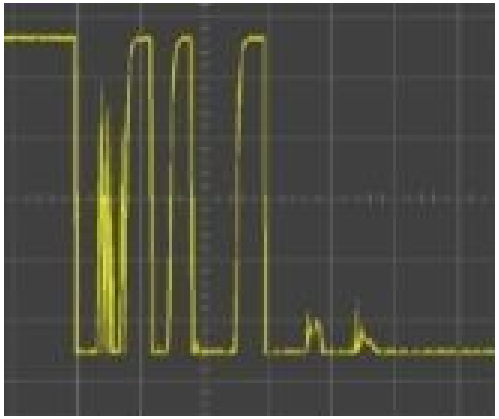
- **Eseménynek tekintjük a G1 bemenet állapotváltozásait:**
  - Ha lenyomásra vártunk és a **G1**-hez tartozó digitális bemeneten `LOW` állapotot észlelünk
  - Ha felengedésre vártunk (`waitforpress = false`) és a **G1**-hez tartozó digitális bemeneten `HIGH` állapotot észlelünk

Második változat: led_switch_v2.ino

```
#define RED_LED 5                   // D5-re van kötve a LED
#define G1      3                   // D3-ra kapcsolódik a nyomógomb
boolean led_state = false;          // Kezdetben a LED ki van kapcsolva
boolean waitforpress = true;        // Kezdetben lenyomásra várunk

void setup() {
  pinMode(RED_LED,OUTPUT);          // Legyen kimenet
  pinMode(G1,INPUT_PULLUP);         // Bemenet belső felhúzással
}

void loop() {
  if(waitforpress) {                // Ha lenyomásra várunk és
    if(!digitalRead(G1)) {          // Ha lenyomás történt...
      led_state = !led_state;       // Állapot átbillentése
      digitalWrite(RED_LED, led_state);  // LED állapot aktualizálás
      waitforpress = false;         // Következő stáció: felengedésre várunk
    }
  }
  else {                            // Ha felengedésre vártunk és
    if(digitalRead(G1)) {           // Ha felengedést észlelünk...
      waitforpress = true;          // Következő stáció: lenyomásra várunk
    }
  }
  delay(20);                        // Pergésmentesítő késleltetés
}
```

Mi az a pergésmentesítés?

- A nyomógombok, kapcsolók érintkezője „pereg”, azaz többször be- és kikapcsol, amikor átváltjuk. Ennek a bizonytalan állapotnak az ideje 10 – 15 ms.



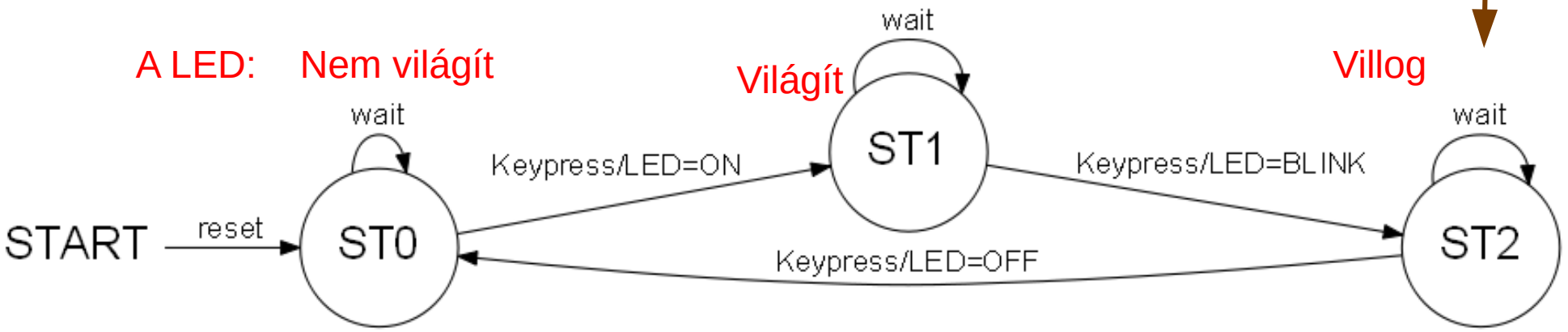
A nyomógomb lenyomásakor többször megváltozik a jelszint, mire stabilizálódik az alacsony szint.
A gomb felengedésekor is hasonló jelenség figyelhető meg

- Szoftverese pergésmentesítés lényege: a gyors változásokat figyelmen kívül hagyjuk. Többféle megoldás lehetséges:
  - Csak bizonyos időközönként (pl. 20 ms) vizsgáljuk a nyomógomb állapotát (ritka mintavételezés). Erre szolgál programunk végén a 20 ms késleltetés.
  - Csak a stabilizálódott (bizonyos ideig változatlan) állapotokat vesszük figyelembe (sűrű mintavételezés és számlálás)

led_tristate.ino

Példaprogram, amelyben nem használhatunk blokkoló várakozást a LED villogtatása miatt!

- LED vezérlés lesz ez is, nyomógomb segítségével, de egy kicsit bonyolítjuk a feladatot:
  - első lenyomásra világítson a LED
  - második lenyomásra villogjon a LED (kb. 1 Hz)
  - harmadik lenyomásra kialszik a LED (alaphelyzetbe tér)
- Minden gomblenyomás egy újabb állapotba viszi a LED-et:
  - ST0: A LED nem világít (a kimenet LOW)
  - ST1: A LED folyamatosan világít (a kimenet HIGH)
  - ST2: A LED villog (a kimenet állapotát 500 ms-onként átbillentjük)
- A LED állapotát most is a **led_state** változóban tároljuk



led_tristate.ino

```
#define RED_LED 5                   // D5-re van kötve a LED
#define G1      3                   // D3-ra van kötve a nyomógomb

boolean waitforpress = true;        // Kezdetben lenyomásra várunk
int led_state = 0;                  // Kezdetben a LED ki van kapcsolva
int timecount = 25;                 // Villogás félperiódusának ideje
                                    // 20 ms egységekben (500 ms = 25*20 ms)
void setup() {
  pinMode(RED_LED,OUTPUT);          // Legyen kimenet
  pinMode(PUSH2,INPUT_PULLUP);      // Bemenet belső felhúzással
}
```

Folytatás a következő oldalon ...

A változók szerepe:

waitforpress: a nyomógomb *lenyomás – felengedés* ciklusának nyilvántartására használjuk (true = lenyomásra várunk, false = felengedésre várunk)

led_state: ebben tartjuk nyilván a LED állapotát (0 = nem világít, 1 = világít, 2 = villog)

timecount: a 20 ms-os időszeletek (vissza)számlálója villogtatáskor

led_tristate.ino

```
void loop() {
  if(waitforpress) {                  //Ha lenyomásra várunk és
    if(!digitalRead(G1)) {            //Ha lenyomás történt...
      if(++led_state > 2) led_state = 0;     //Állapot léptetés
      digitalWrite(RED_LED, led_state > 0); //LED állapot beállítása
      waitforpress = false;           //Következő stáció: felengedésre várunk
    }
  }
  else {                              //Ha felengedésre vártunk és
    if(digitalRead(G1)) {             //Ha felengedést észlelünk...
      waitforpress = true;            //Következő stáció: lenyomásra várunk
    }
  }
  delay(20);                          //pergésmentesítő késleltetés
  if(led_state == 2)                  //Ha villogtatás állapotban vagyunk
    if(--timecount == 0) {            //Ha az eltelt idő 25*20 ms, akkor
      digitalWrite(RED_LED,!digitalRead(RED_LED)); //LED átbillentése
      timecount = 25;                 //Következő billentés 25*20 ms múlva...
    }
}
```

KEYPRESS

A nyomógombot kezelő állapotgép megegyezik az előző programban használttal

BLINK

Villogtatás esetén 25 x 20 ms-onként átbillentjük a LED állapotát

led_tristate2.ino

- Írjuk át az előző programot úgy, hogy válasszuk szét benne a nyomógomb és a LED kezelését!
- A program így nyilván terjedelmesebb lesz és kevésbé hatékony, de talán egy picit áttekinthetőbb – vagy legalább közelít ahhoz, amire az óraprojektben szükségünk lesz
- A program eleje:

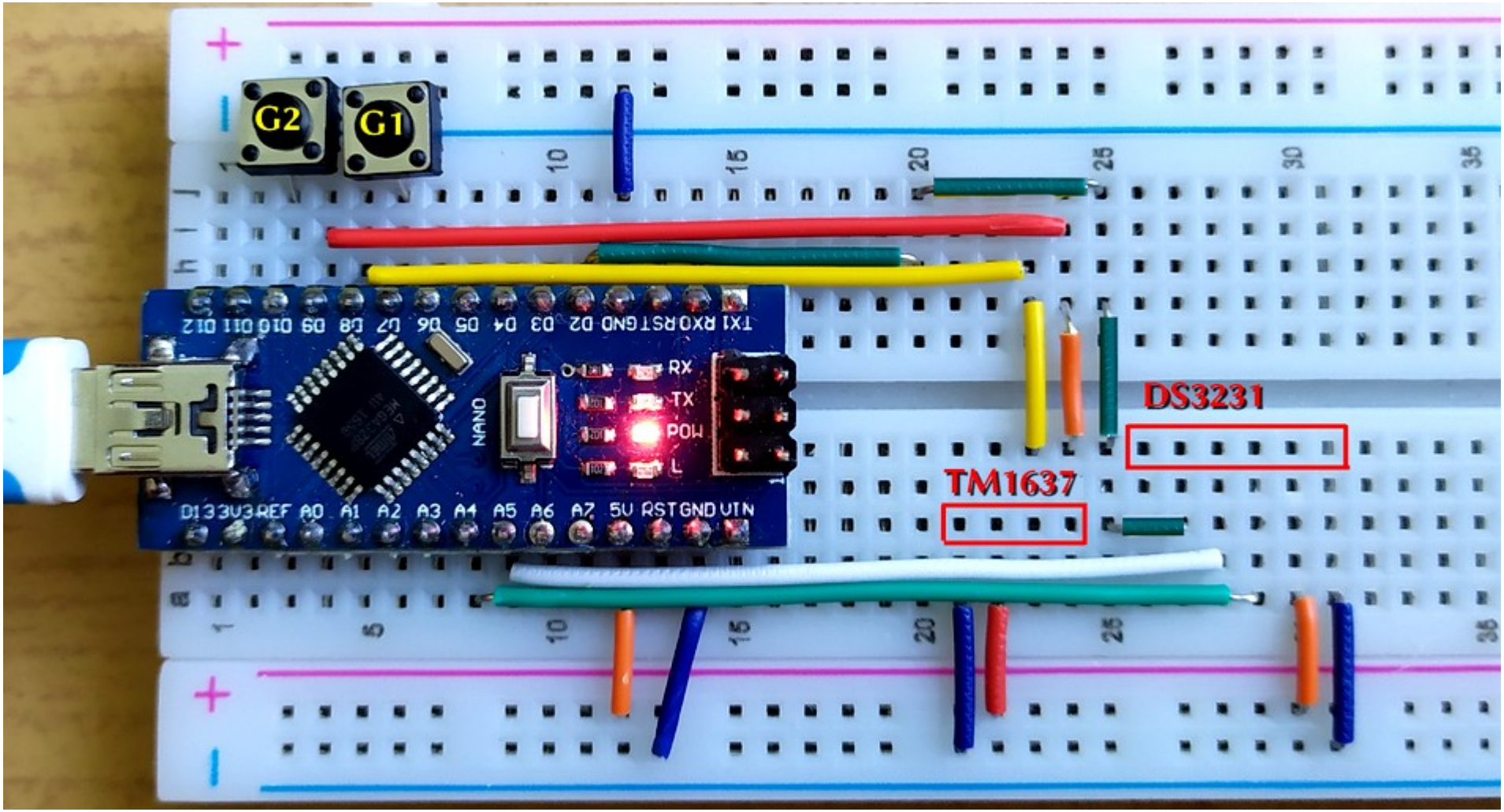
```
#define RED_LED 5                   // D5-re van kötve a LED
#define G1      3                   // D3-ra van kötve a nyomógomb
boolean waitforpress = true;        // Kezdetben lenyomásra várunk
int led_state = LOW;                // Kezdetben a LED ki van kapcsolva
int timecount = 25;                 // Villogás félperiódusának ideje

void setup() {
  pinMode(RED_LED, OUTPUT);         // legyen kimenet
  pinMode(G1, INPUT_PULLUP);        // Bemenet belső felhúzással
}
```

led_tristate2.ino

```
void loop() {
  if (waitforpress) {              //Ha lenyomásra várunk és
    if (!digitalRead(G1)) {        //Ha lenyomás történt...
      led_state = (led_state + 1) % 3;  //Állapot léptetése (2 után körülfordulás...)
      waitforpress = false;        //Következő stáció: felengedésre várunk
    }
  }
  else {                           //Ha felengedésre vártunk és
    if (digitalRead(G1)) {         //Ha felengedést észlelünk...
      waitforpress = true;         //Következő stáció: lenyomásra várunk
    }
  }
  switch (led_state) {
    case 0:
      digitalWrite(RED_LED, LOW); //Lekapcsoljuk a LED-et
      break;
    case 1:
      digitalWrite(RED_LED, HIGH); //Felkapcsoljuk a LED-et
      break;
    default:                       //Villogtatjuk a LED-et
      if (--timecount == 0) {      //Ha az eltelt idő 25*20 ms, akkor
        digitalWrite(RED_LED, !digitalRead(RED_LED)); //LED állapot átbillentése
        timecount = 25;            //Következő billentés újabb 25*20 ms múlva...
      }
  }
  delay(20);                       //Időalap és pergésmentesítő késleltetés
}
```

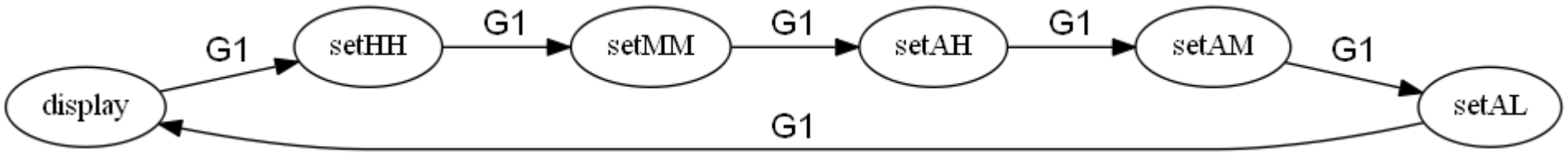
Ébresztőóra projekt

- Készítsünk ébresztőórát egy **TM1637** 4 digites kijelző és egy **DS3231** modul felhasználásával! A kapcsolás ugyanaz, mint az előző előadásban



A program megtervezése

- A **G1** gomb megnyomása a funkciót lépteti, **G2** pedig az értéket:
  - **Display** mód – kijelzés (**G2** hatástalan)
  - **setHH** mód – az órák beállítása (**G2** lépteti az értéket)
  - **setMM** mód – a percek beállítása (**G2** lépteti az értéket)
  - **setAH** mód – ébresztési óra beállítása (**G2** lépteti az értéket)
  - **setAM** mód – ébresztési perc beállítása (**G2** lépteti az értéket)
  - **setAL** mód – ébresztési funkció ki/bekapcsolása (**G2** lépteti)
- **Beállítás módban** az első két számjegyen a paraméter kódját *(HH, PP, AH, AP, AL)* az utolsó kettőn pedig az értékét írjuk ki
- Az állapotokon úgy lépkedünk végig, mint a `led_tristate` programban:



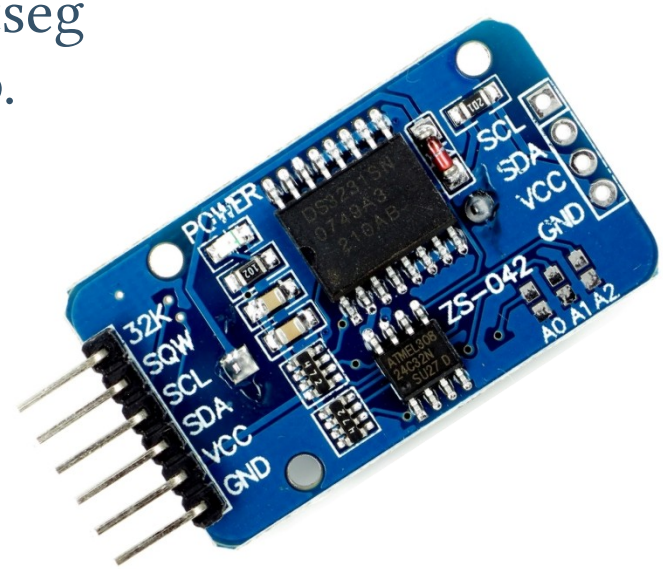
A TM1637Display programkönyvtár metódusai

- **setBrightness**(brightness) – fényerő beállítása (0 – 7) és bekapcsolás (8)
- **showNumberDec**(*num, leading_zero = false, length=4, pos=0*) – szám kiíratása decimális alakban, ahol ***num*** = 0 – 9999, negatív számok esetén pedig 0-tól –999-ig, ***length*** a számjegyek száma, ***pos*** a kezdő pozíció (0-3)
- **showNumberDecEx**(*num, dots=0, leading_zero=false, length=4, pos=0*) – szám kiíratása tizedespont vezérléssel (***dots = 64*** esetén kigyújtja a középső kettőspontot, ***dots = 0*** esetén pedig kioltja)
- **setSegments**(*segments[], length=4, pos=0*) - szegmensek beállítása a ***segments[]*** tömb szerint
- byte **encodeDigit**(digit) – visszaadja a ***digit*** számjegy szegmenseinek kódját, pl. 1 esetén `SEG_B | SEG_C` értékét
- A karakterek kiíratáshoz szükséges kódok:
  A: 0x77, **H:** 0x76, **P:** 0x73, **L:** 0x38

```
#define SEG_A   0b00000001
#define SEG_B   0b00000010
#define SEG_C   0b00000100
#define SEG_D   0b00001000
#define SEG_E   0b00010000
#define SEG_F   0b00100000
#define SEG_G   0b01000000
```

DS3231 Real-time óra modul

- Oszcillátor, óra és naptár egy tokban. A tápfeszültség megszűnésekor a hátoldali telepről üzemel tovább.
- DS3231 I2C címe: **0x68**
- Két, független ébresztési időpont állítható be, mi a másodikat (Alarm2) fogjuk használni, ennek használata valamivel egyszerűbb
- Az ébresztési mód az A2M[4:2] bitekben állítható be



- A DY/DT bit szabja meg, hogy a hónap vagy a hét napja számít

DY/DT	ALARM 2 REGISTER MASK BITS (BIT 7)			ALARM RATE
	A2M4	A2M3	A2M2	
X	1	1	1	Alarm once per minute (00 seconds of every minute)
X	1	1	0	Alarm when minutes match
X	1	0	0	Alarm when hours and minutes match
0	0	0	0	Alarm when date, hours, and minutes match
1	0	0	0	Alarm when day, hours, and minutes match

- Adatlap: datasheets.maximintegrated.com/en/ds/DS3231.pdf

DS3231 regiszterek

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	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	—

A DS3231 programkönyvtár

- Mintapéldánkban most is ***Andrew Wickert*** programkönyvtárát használjuk: https://github.com/NorthernWidget/DS3231

A legfontosabb függvények:

- **checkIfAlarm()** – ébresztési állapot lekérdezése (és törlése)
- **getHour**(), **getMinute(), getSecond()** – az idő adatok kiolvasása
- **setHour(), setMinute(), setSecond()** – az idő adatok beállítása
- **getA2Time()** – Alarm2 ébresztési időpont és mód lekérdezése
- **SetA2Time()** – Alarm2 ébresztési időpont és mód beállítása
- **checkAlarmEnabled()** – ébresztés engedélyezésének lekérdezése
- **turnOnAlarm(), turnOffAlarm()** – ébresztés be- vagy kikapcsolása
- **enableOscillator**() – az oszcillátor engedélyezése, SQW kimenő jel konfigurálása

tm1637_clock3.ino

```
#include <TM1637Display.h>
#include <DS3231.h>
#include <Wire.h>
#define CLKPIN  9                       // TM1637 CLK láb D9-re
#define DIOPIN  8                       // TM1637 DIO láb D8-ra
#define G1      7                       // Button 1
#define G2     10                       // Button 2
#define SQW     2                       // Megszakítás
#define BUZZER 13                       // Alarm kimenet
DS3231 Clock;                           // Az RTC példányosítása
TM1637Display display(CLKPIN, DIOPIN); // 4-számjegyű kijelző példányosítása
bool h12, PM;                           // 12/24 mód és AM/PM jelzőbitek
int hh = 0;                             // órák száma
int mm = 0;                             // percek száma
int dp = 0;                             // Kettőspont kijelzés (0 vagy 64)
uint32_t g1low = 0;                     // G1 alacsony állapot számlálója
uint32_t g1high = 0;                    // G1 magas állapot számlálója
byte state = 0;                         // Az óra melyik állapotban van
byte A2Day;                             // Változók Alarm2 beállításának lekérdezéséhez
byte A2Hour;
byte A2Minute;
byte AlarmBits;
bool A2Dy;
bool A2h12;
bool A2PM;
bool alarm;                             // Ébresztés engedélyezés lekérdezéséhez
```

tm1637_clock3.ino

- **A hardver beállítások** hasonlóan történnek, mint a múlt órai mintapéldában, csak az ébresztésjelző kimenettel bővült a felhasznált kivezetések száma
- A programban a **D13**-as lábat választottuk ébresztésjelző kimenetnek, így a beépített LED közvetlenül jelzi az ébresztést. Természetesen köthetünk a **D13**-as kivezetésre egy saját oszcillátorral rendelkező piezó csipogót is (ezeknek általában fekete műgyantával kiöntött a hátulja). Ügyeljünk a polaritásra!


```
void setup() {
  pinMode(G1, INPUT_PULLUP);          // G1 állapot léptető gomb bemenet
  pinMode(G2, INPUT_PULLUP);          // G2 paraméter állító gomb bemenet
  pinMode(SQW, INPUT_PULLUP);         // 1 Hz négyszögjel bemenet
  pinMode(BUZZER, OUTPUT);            // Ébresztő kimenet konfigurálása
  digitalWrite(BUZZER, LOW);
  Wire.begin();                       // Az I2C illesztő inicializálása
  display.setBrightness(0x0F);        // a kijelző bekapcsolása max. fényerőn
  Clock.enableOscillator(true, 0, 0);
}
```

Események kezelése

- A **G1** gomb kezelését a **led_tristate2** programhoz hasonlóan itt is különválasztjuk a többi résztől, itt csak az állapotot léptetjük:

```
void loop() {
  uint16_t d12, d34;
  uint8_t data[] = { 0, 0, 0, 0 };

  /*-------------------------------------------
     A G1 gomb lenyomásának vizsgálata
     Érvényes lenyomás esetén állapotot váltunk
    -------------------------------------------*/
  if ( digitalRead(G1) == LOW) {        //Ha le van nyomva...
    g1low++;
    if ((g1low > 5) && (g1high > 5)) { //és érvényes a lenyomás
      g1high = 0;
      g1low = 0;
      state = (state + 1 ) % 6;         //állapot léptetése
    }
  } else {                              //Ha felengedett állapotban van...
    g1high++;
    g1low = 0;
  }
... // Folytatás a következő oldalon
```

G1 lenyomása akkor érvényes, ha legalább 6 időszeletig fel volt engedve és legalább ugyanennyi ideje le van nyomva

Események kezelése

- Az óra állapotaihoz tartozó tevékenységek elvégzése itt kezdődik egy `switch(state) {}` utasítás formájában. Mindig csak az aktuális állapothoz tartozó ág aktivizálódik
- A `state = 0` állapot a kijelzés módot jelenti. Itt történik az ébresztés végrehajtása, illetve az 1 Hz-es négyszögjel figyelése

```
switch (state) {
  case 0:                                  // Kijelzés mód
    //-- Ébresztés, ha itt az idő -------------
    if (Clock.checkIfAlarm(2) ) {
      digitalWrite(BUZZER, HIGH);          // Alarm aktiválás
    }
    //-- Ébresztés megszüntetése 10 mp után ---
    if (Clock.getSecond() > 10) {
      digitalWrite(BUZZER, LOW);           // Alarm vége
    }
    //-- Az 1 Hz-es jel figyelése -------------
    dp = digitalRead(SQW) ? 64 : 0;
    break;                                 // Továbblépés a kijelzéshez
```

Az idő (óra, perc) beállítása

```
    case 1:                                   // setHH mód
      hh = Clock.getHour(h12, PM);
      if ( digitalRead(G2) == LOW) {          // Ha a G2 gomb le van nyomva...
        hh = (hh + 1) % 24;                   // Az órák léptetése
        Clock.setHour(hh);                    // Az új érték eltárolása
        delay(100);
      }
      d12 = 0x7676;                           // HH kijelzés
      d34 = hh;                               // óra kijelzés
      break;                                  // Továbblépés a kijelzéshez

    case 2:                                   // setMM mód
      mm = Clock.getMinute();
      if ( digitalRead(G2) == LOW) {          // Ha a G2 gomb le van nyomva...
        mm = (mm + 1) % 60;                   // A percek léptetése
        Clock.setMinute(mm);                  // Percek eltárolása
        Clock.setSecond(0);                   // Másodpercek nullázása
        delay(100);
      }
      d12 = 0x7373;                           // PP kijelzés
      d34 = mm;                               // percek kijelzése
      break;                                  // Továbblépés a kijelzéshez
```

Az ébresztési idő (óra) beállítása

- Ha az ébresztési óra kisebb, mint a mostani idő, akkor az ébresztés nem ma, hanem holnap esedékes, amit itt az **A2Day** változó értékének léptetésével oldunk meg
- Egyszerűbb lett volna az **AlarmBits = 4** beállítás használata…

```
case 3:                                 // setAH mód
  Clock.getA2Time(A2Day, A2Hour, A2Minute, AlarmBits, A2Dy, A2h12, A2PM);
  if ( digitalRead(G2) == LOW) {        // Ha a G2 gomb le van nyomva...
    A2Day = Clock.getDoW();             // ébresztés ma, vagy holnap
    A2Dy = true;                        // A hét napja egyezést figyelje
    AlarmBits = 8;                      // Alarm when DoW, hour, min match
    A2Hour = (A2Hour + 1) % 24;         // Az ébresztési óra léptetése
    if (hh > A2Hour) {
      A2Day = (A2Day % 7) + 1;          // Az ébresztés csak holnap esedékes
    }
    Clock.setA2Time(A2Day, A2Hour, A2Minute, AlarmBits, A2Dy, A2h12, A2PM);
    delay(100);
  }
  d12 = 0x7776;                         // AH kijelzése
  d34 = A2Hour;                         // Ébresztési óra kijelzése
  break;                                // Továbblépés a kijelzéshez
```

Az ébresztési idő (perc) beállítása

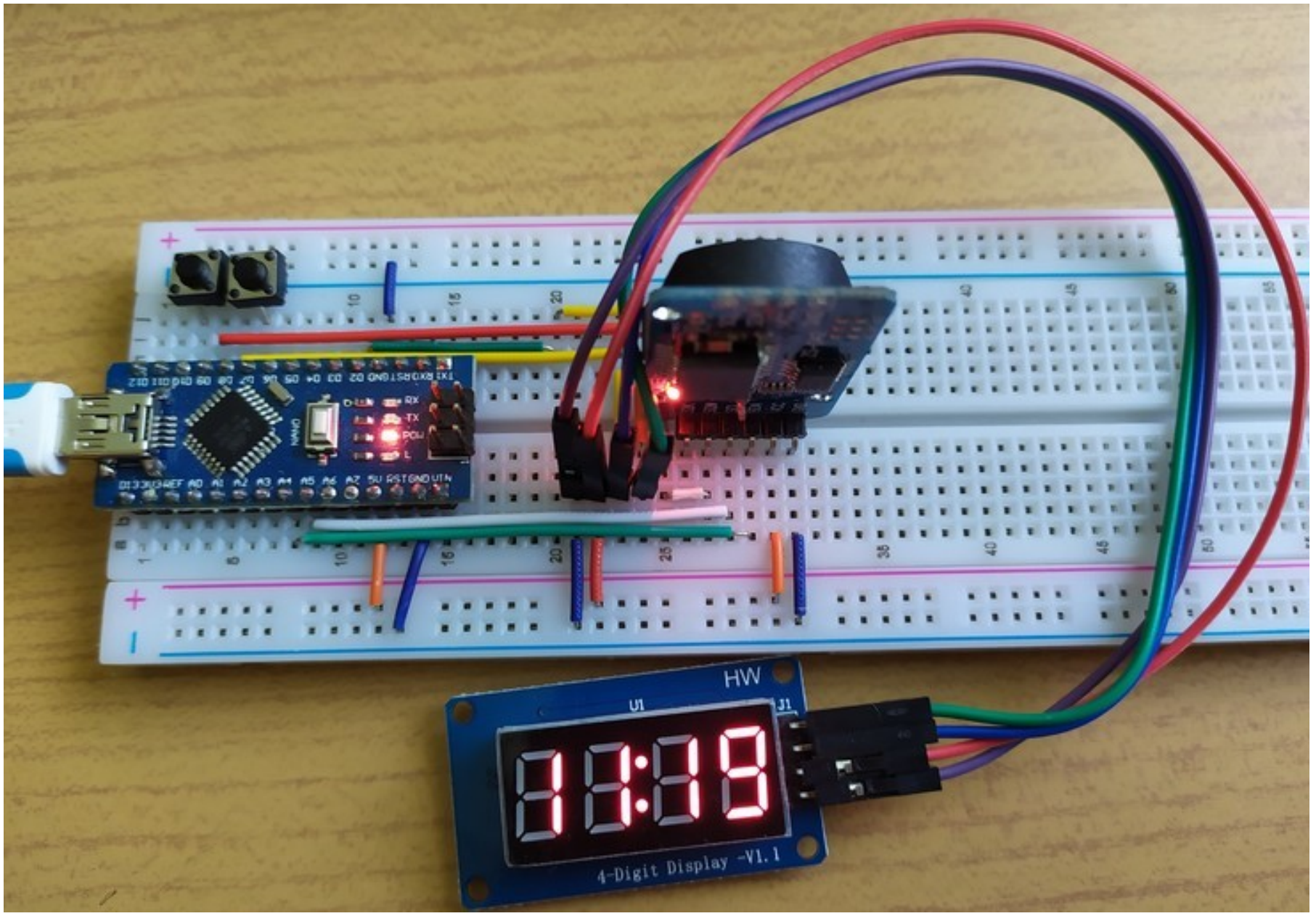
- Ha a mostani idő túlhaladta az ébresztési időt, akkor az ébresztés nem ma, hanem holnap esedékes
- Egyszerűbb lett volna az **`AlarmBits = 4`** beállítás használata...

```
case 4:                                    // setAM mód
  Clock.getA2Time(A2Day, A2Hour, A2Minute, AlarmBits, A2Dy, A2h12, A2PM);
  if ( digitalRead(G2) == LOW) {           // Ha a G2 gomb le van nyomva...
    A2Dy = true;
    AlarmBits = 8;                         // Alarm when DoW, hour, min match
    A2Minute = (A2Minute + 1) % 60;        // Az ébresztési perc léptetése
    if (hh == A2Hour) {
      A2Day = Clock.getDoW();
      if (mm >= A2Minute) {
        A2Day = (A2Day % 7) + 1;           // Az ébresztés holnap esedékes
      }
    }
    Clock.setA2Time(A2Day, A2Hour, A2Minute, AlarmBits, A2Dy, A2h12, A2PM);
    delay(100);
  }
  d12 = 0x7773;                            // AP
  d34 = A2Minute;
    break;
```

Ébresztés engedélyezés és a kijelzés

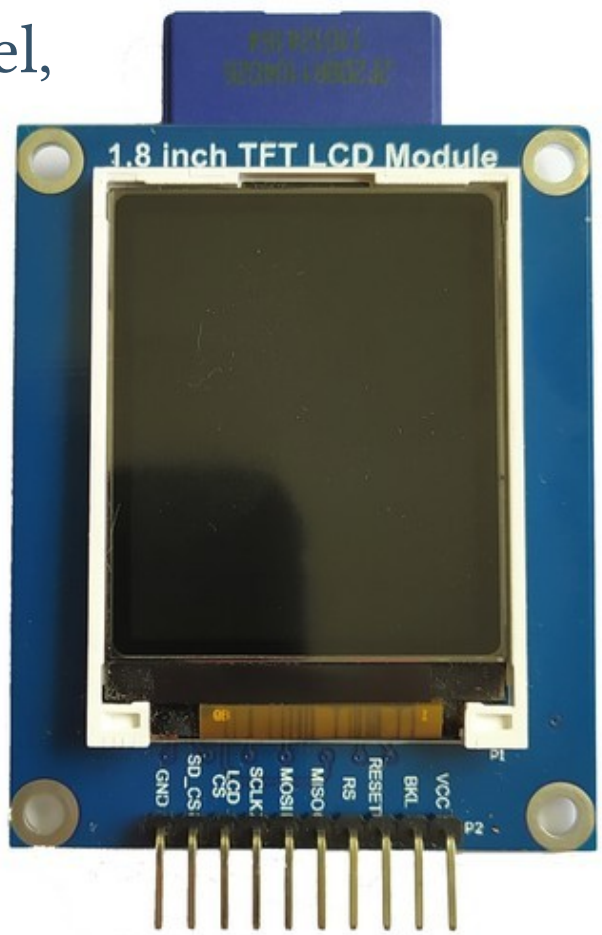
```
    case 5:                                  // setAL mód
      alarm = Clock.checkAlarmEnabled(2); // 1: ha A2 alarm engedélyezve van
      if ( digitalRead(G2) == LOW) {       // Ha a G2 gomb le van nyomva...
        alarm = !alarm;                    // A2 alarm status átbillentése
        if (alarm) Clock.turnOnAlarm(2);   // Ébresztés bekapcsolása
        else Clock.turnOffAlarm(2);        // Ébresztés kikapcsolása
        Delay(200);                        Clock.enableOscillator(true, 0, 0);
      }                                    „Lapzárta után érkezett” hír, hogy itt újra
      d12 = 0x7738; // AL                  konfigurálni kell az SQW 1 Hz-es jelkimenetet
      d34 = alarm;
  } // switch utasítás vége

  //-- A kijelzés frissítése --------------------
  if (state == 0) {                                  // Ha Kijelzés módban vagyunk
    hh = Clock.getHour(h12, PM);
    mm = Clock.getMinute();                          // Percek kiolvasása
    display.showNumberDecEx(hh, dp, false, 2, 0); // Óra kijelzés,kettőspont villogtatás
    display.showNumberDec(mm, true, 2, 2);           // percek kijelzése
  } else {
    data[0] = d12 >> 8;                              // Első karakter
    data[1] = d12 & 0xFF;                            // Második karakter
    data[2] = display.encodeDigit(d34 / 10);         // Első számjegy
    data[3] = display.encodeDigit(d34 % 10);         // Második számjegy
    display.setSegments(data);                       // Megjelenítés
  }
  delay(20);                                         // pergésmentesítés, rövid időszelet
}
```

A megépített kapcsolás



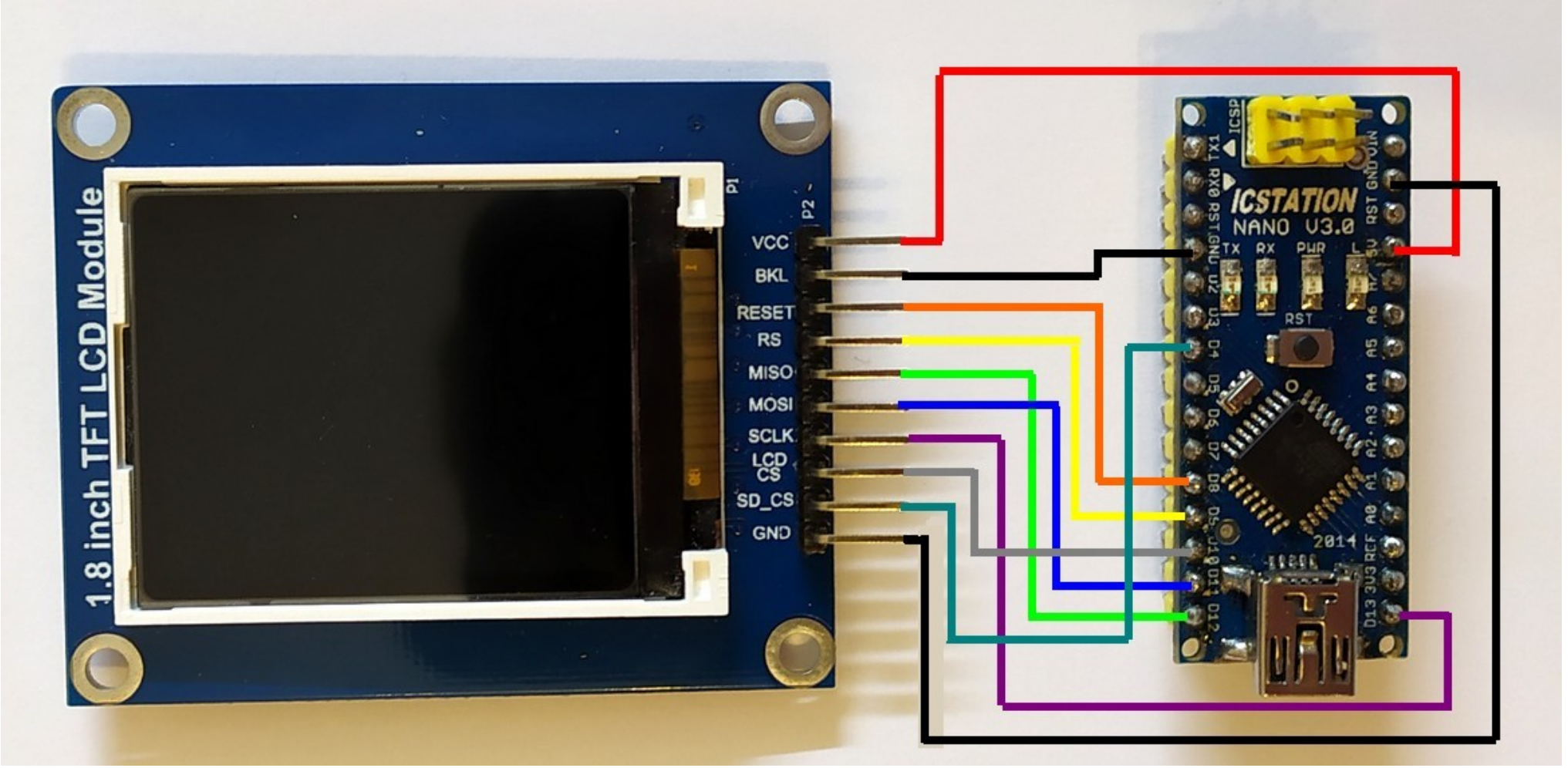
1.8” színes LCD kijelző

- Vezérlő: Sitronix **ST7735**
- Interfész: SPI, csak írás (max. 10–15 MHz)
- Data/Command választás külön vonalon
- Felbontás: 128 x 160 pixel, 65 536 szín (16 bit RGB)
- SD kártya foglalat
- Tápfeszültség: 5V
- Jelszint: a képen látható típusnál 5, illetve 3.3V is lehet (beépített fetes jelszintillesztők)
  Más típusoknál gondot okozhat az 5V jelszint!



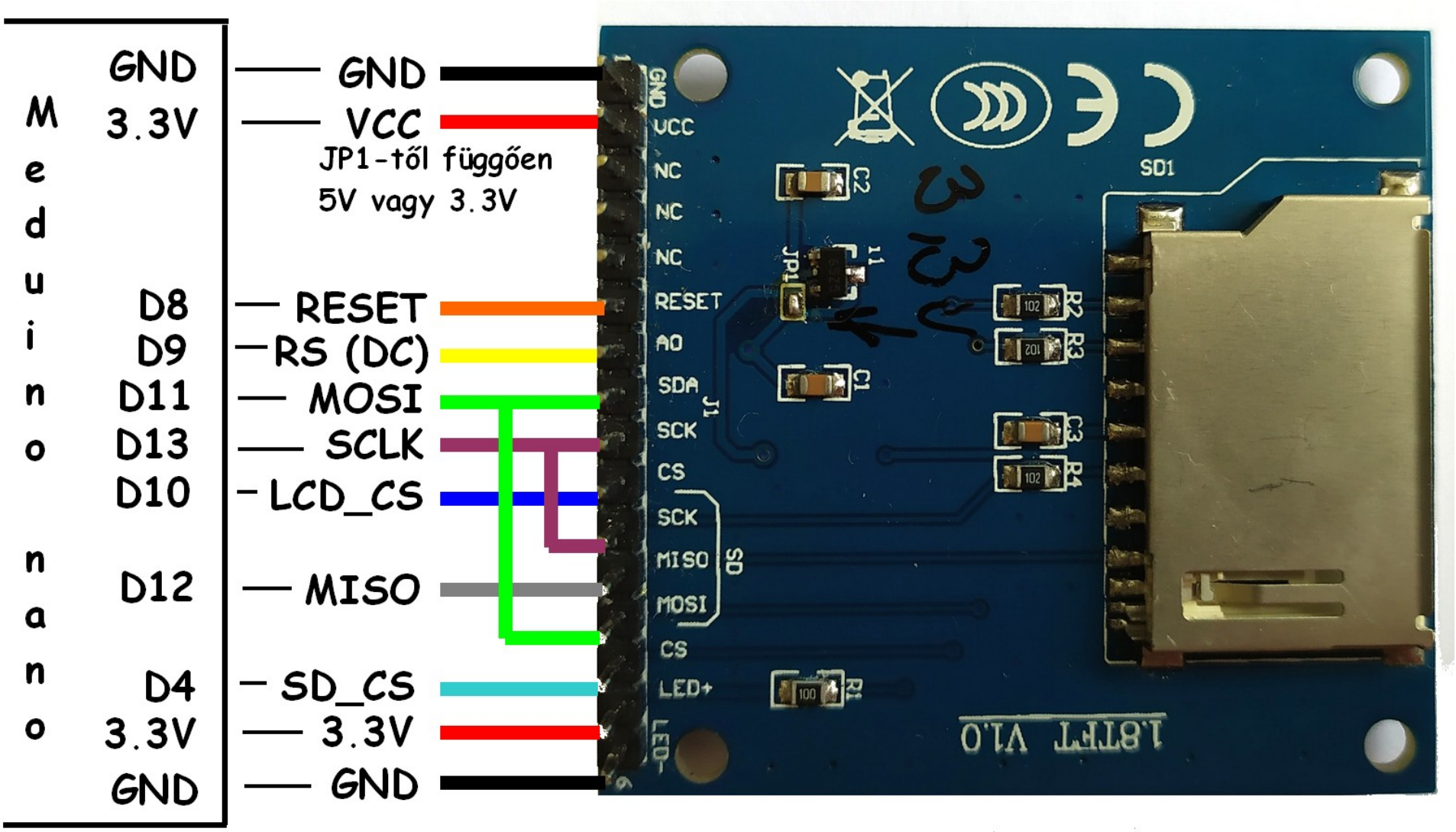
Bekötési vázlat – 5 V-os változat

- **VCC:** 5V, **BKL:** GND, **RESET:** D8, **RS(DC):** D9, **MISO:** D12, **MOSI:** D11, **SCLK:** D13, **LCD_CS:** D10, **SD_CS:** D4, **GND:** GND



Bekötési vázlat – 3.3 V-os változat

- 3.3 V-ról táplált Arduino és egy tipikus 3.3 V-os kijelző modul bekötési vázlata (ha JP1 nyitott, akkor VCC = 5V legyen)



TFT – a beépített programkönyvtár

- Az Arduino IDE egyik beépített könyvtára a TFT library amely az Adafruit_GFX és Adafruit_ST7735 könyvtárakon alapul
- Dokumentációja itt található: arduino.cc/en/Reference/TFTLibrary
- **Fontosabb metódusok:**
  - **begin()** - inicializálja a kijelzőt. `TFT.cpp` módosításra szorulhat
  - **background(***r,g,b***)** – a megadott színre törli a képernyőt
  - **stroke(***r,g,b***)** – tintaszín megadása rajzoláshoz, szövegíráshoz
  - **setTextSize(***n***)** – karakterméret megadása (*n = 1 – 5*)
  - **text(***text,xpos,ypos***)** – szöveg kiírása adott pozícióba
  - **point(***xpos,ypos***)** – pont kirajzolása adott pozícióba
  - **line(***x0,y0,x1,y1***)** – szakasz kirajzolása
  - **rect(***xpos,ypos,width,height***)** – téglalap rajzolása

TFT.cpp konfigurálása

- A TFT könyvtár egyfajta kijelzőt feltételez (fekvő állás) ezért az ArduinoIDE/libraries/tft/src mappában módosítani kell TFT.cpp-t

```
#include "TFT.h"
TFT::TFT(uint8_t CS, uint8_t RS, uint8_t RST) : Adafruit_ST7735(CS, RS, RST) {
  _width = ST7735_TFTHEIGHT;   // landscape orientation
  _height = ST7735_TFTWIDTH;
}

void TFT::begin() {
  initR(INITR_REDTAB);
  SetRotation(1);              // landscape orientation
}
```

Fekvő elrendezés

- Nálam most ez áll a TFT.cpp állományban:

```
#include "TFT.h"

TFT::TFT(uint8_t CS, uint8_t RS, uint8_t RST) : Adafruit_ST7735(CS, RS, RST) {
  _width  = ST7735_TFTWIDTH;   // portrait orientation
  _height = ST7735_TFTHEIGHT;
}

void TFT::begin() {
  initR(INITR_BLACKTAB);       // red ⇔ blue change
  SetRotation(0);              // portrait orientation
}
```

Álló elrendezés

Piros-kék színcsere

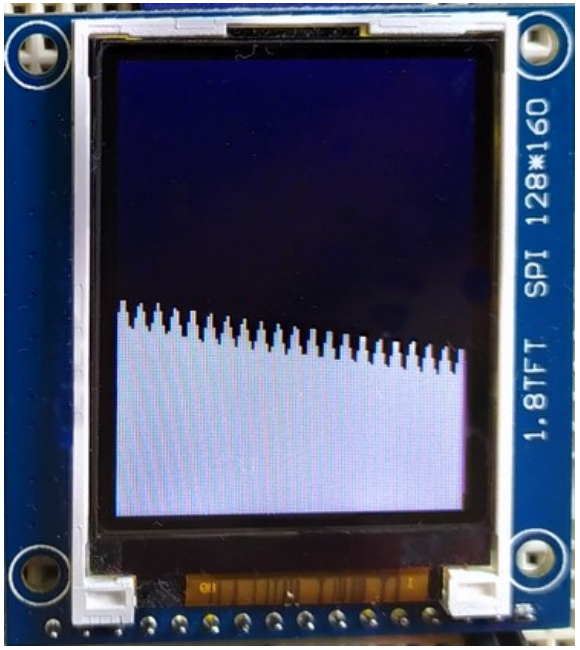
tft_proba.ino

```
#include <TFT.h>
#include <SPI.h>
#define cs   10
#define dc   9
#define rst  8

TFT TFTscreen = TFT(cs, dc, rst);

void setup() {
  TFTscreen.begin();
  TFTscreen.background(0,0,255);
  TFTscreen.setTextSize(1);
  TFTscreen.stroke(255,255,255);
  TFTscreen.text("Hobbielektronika", 6, 20);
  TFTscreen.setTextSize(2);
  TFTscreen.stroke(0, 255, 0);
  TFTscreen.text("csoport", 6, 40);
  TFTscreen.setTextSize(3);
  TFTscreen.stroke(255, 0, 0);
  TFTscreen.text("2019/20", 2, 70);
}

void loop() {}
```

- **Szövegkiíratás**
  1, 2 és 3-as méretben, más-más színnel

tft_graph.ino

A **TFTGrah.ino** „gyári" mintaprogram kissé módosított változata




```
#include <TFT.h>
#include <SPI.h>
#define cs   10
#define dc   9
#define rst  8

TFT TFTscreen = TFT(cs, dc, rst);
int xPos = 0;

void setup() {
  TFTscreen.begin();
  TFTscreen.background(0, 0, 100);
}

void loop() {
  int sensor = analogRead(A0);
  int drawHeight = map(sensor, 0, 1023, 0, TFTscreen.height());
  TFTscreen.stroke(250, 250, 250);
  TFTscreen.line(xPos, TFTscreen.height() - drawHeight, xPos, TFTscreen.height());
  if (xPos >= 128) {   // Landscape módban 160 lenne a határ...
    xPos = 0;
    TFTscreen.background(0,0,100);
  } else {
    xPos++;
  }
  delay(16);
}
```

Az **A0** analóg bemeneten mért feszültséget ábrázoljuk a grafikus kijelzőn. Kis túlzással lassú letapogatású tárolós oszcilloszkónak is nevezhetjük...

Az SD kártya használata

- Az **SD** programkönyvtár is az Arduino IDE-vel települ, leírása itt található: arduino.cc/en/Reference/SD
- Az **SDclass** objektumosztály metódusai:
  - **begin(***cspin***)** – az SPI busz, az SD library és a kártya inicializálása
  - **open(***filepath, mode***)** – fájl vagy mappa megnyitása
  - Továbbiak: **exists(), mkdir(), remove(), rmdir()**
- A **File** *Stream* objektumosztály, melynek példánya mappa vagy állomány, lehetővé teszi, hogy az **SD** kártyán fájlokat kezeljünk
  - **name()** - a fájl nevét adja vissza
  - **isDirectory()** - fájl vagy mappa a megnyitott File objektum?
  - **openNextFile()** - a mappában soron következő fájl megnyitása
  - **rewindDirectory()** - visszatérés a mappa első fájljához
  - **close()** - a fájl bezárása
  - Továbbiak: **read(), write(), size(), seek(), print(), println(), position(), peek(), flush(), available()**

TFTBitmap_all.ino 2/1. oldal

```
#include <SPI.h>
#include <SD.h>
#include <TFT.h>
#define sd_cs  4
#define lcd_cs 10
#define dc     9
#define rst    8

TFT TFTscreen = TFT(lcd_cs, dc, rst);
PImage logo;
File root;

void setup() {
  TFTscreen.begin();
  TFTscreen.background(0, 0, 0);
  if (!SD.begin(sd_cs)) {
    return;
  }
  root = SD.open("/");
}
```

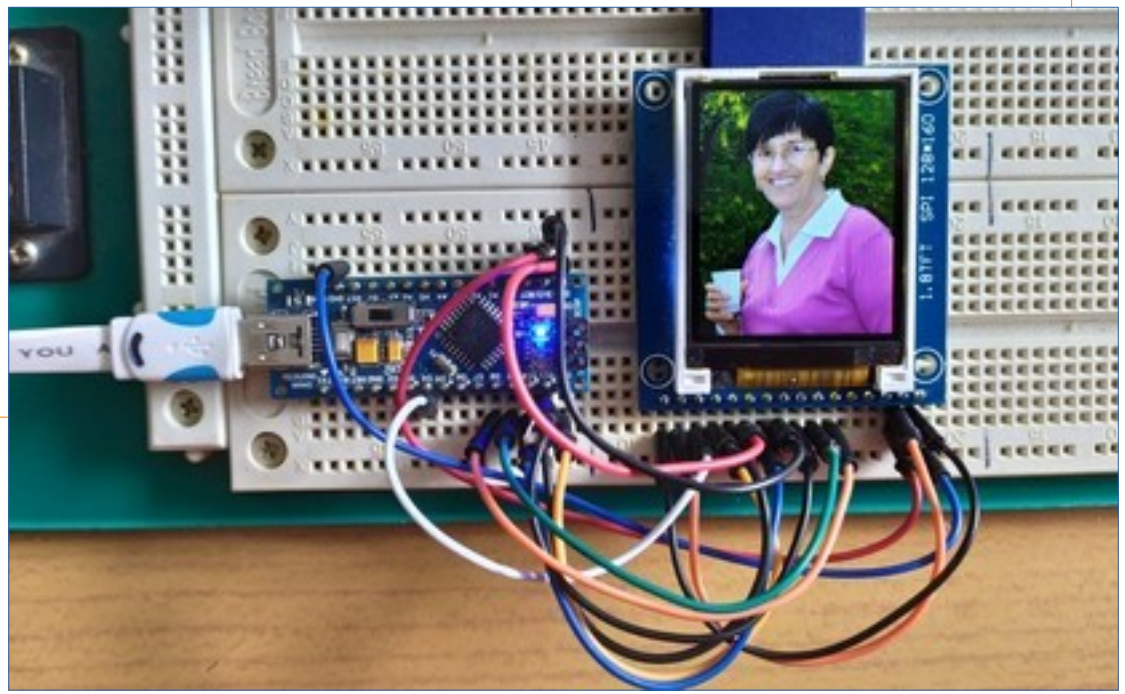
Ez a program sorra megnyitja és megjeleníti az SD kártyán található Képeket

A 128x160 pixel méretű képek BMP formátumban legyenek!

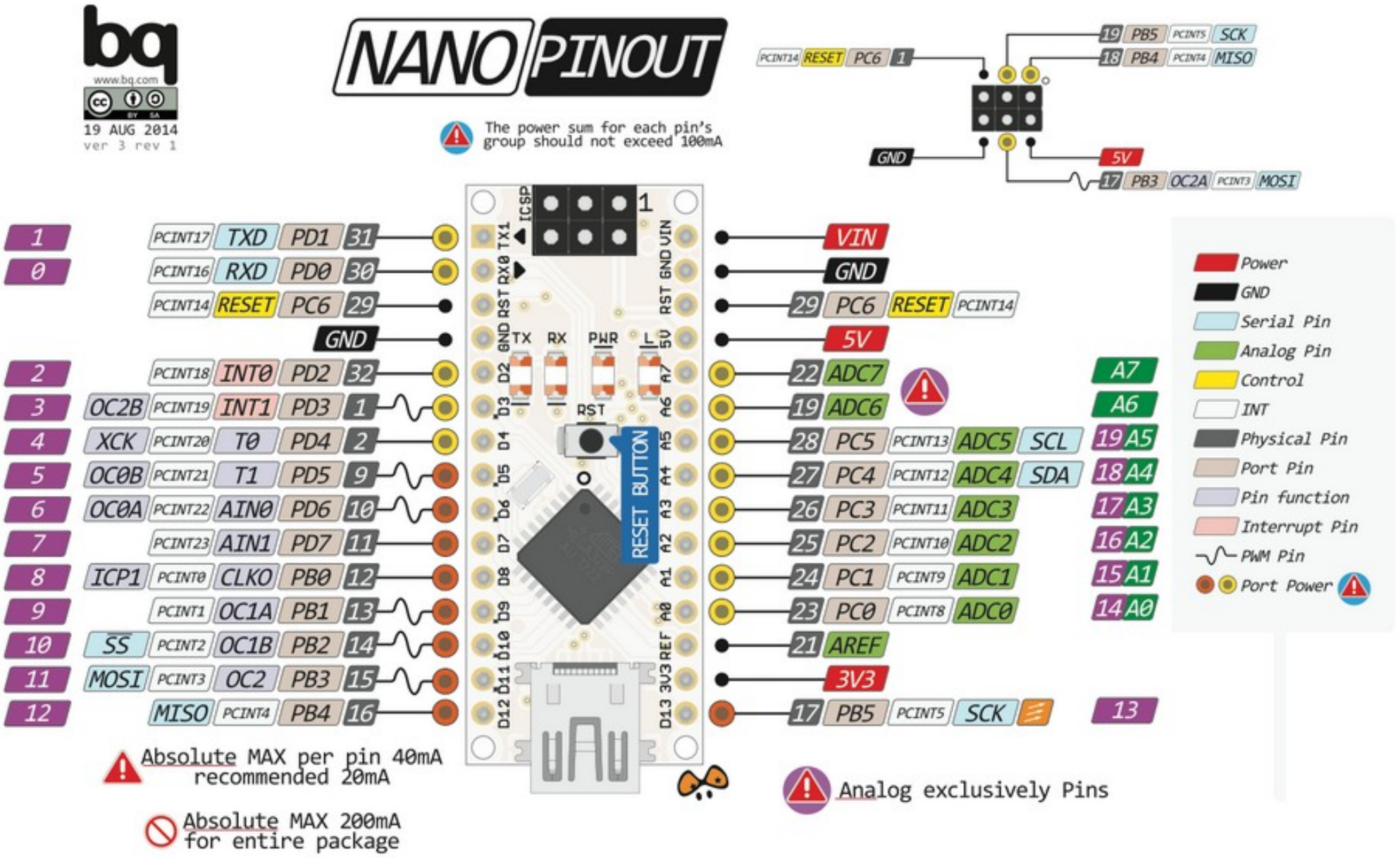
Alkönyvtárakat nem kezelünk!

TFTBitmap_all.ino 2/2. oldal

```
void loop() {
  File entry =  root.openNextFile();
  if (entry) {
    logo = TFTscreen.loadImage(entry.name());
    if (logo.isValid() == false) {
      return;
    }
    TFTscreen.image(logo, 0, 0);
    entry.close();
    delay(5000);
  }
  else {
    root.rewindDirectory();
  }
}
```



Az Arduino nano kártya kivezetései



Ellenállás színkódok