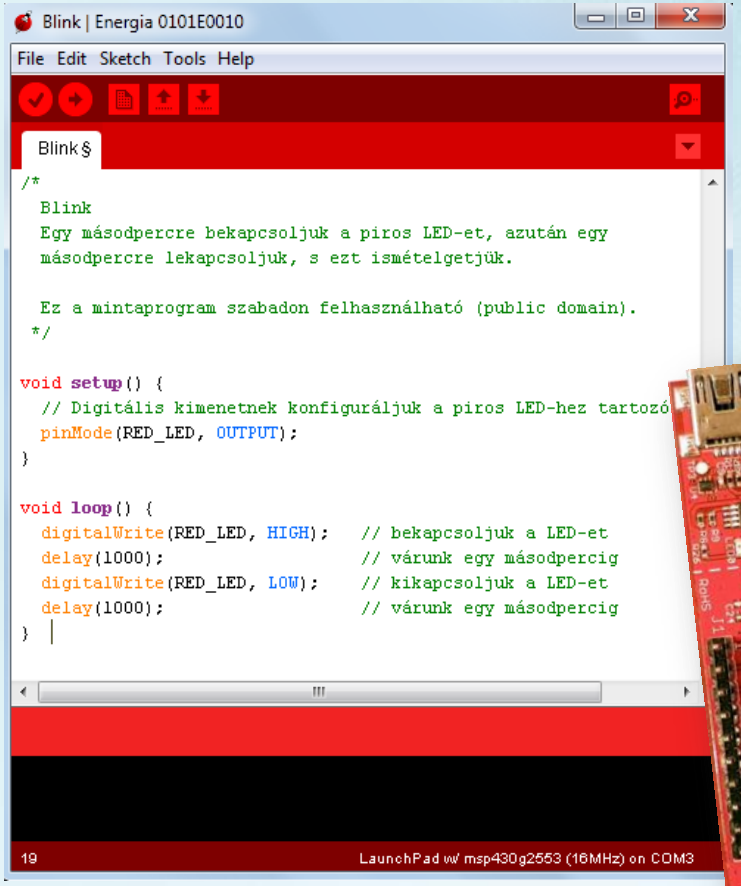
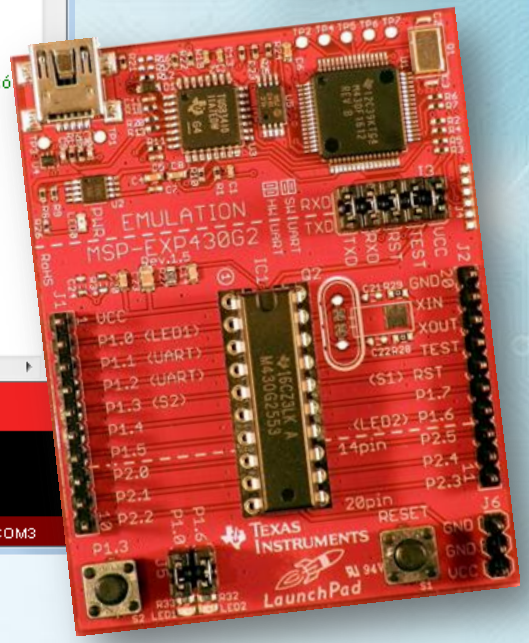




Energīa

MSP430 programozás Energia környezetben

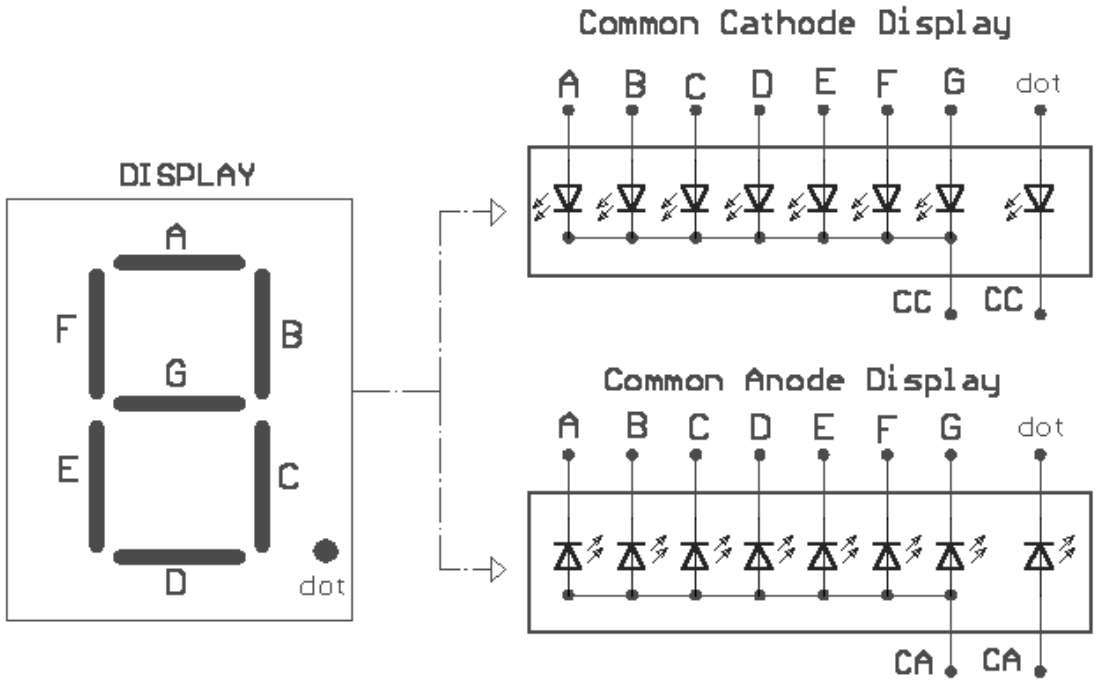
Hétszegmenses LED kijelzők

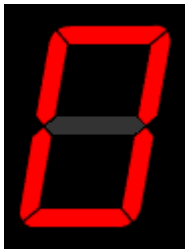
A hétszegmenses kijelző

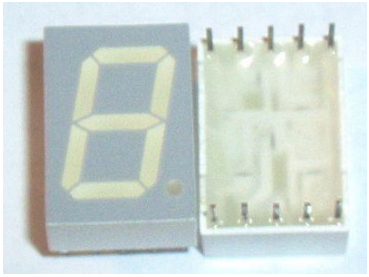
A hétszegmenses kijelzők 7 db LED-et vagy LED csoportot tartalmaznak, olyan elrendezésben, hogy a 0…9 arab számjegyeket ki lehessen jelezni.

A hét szegmenshez nyolcadikként többnyire egy tizedespont is járul.

Kivetel: Közös anódú vagy közös katódú







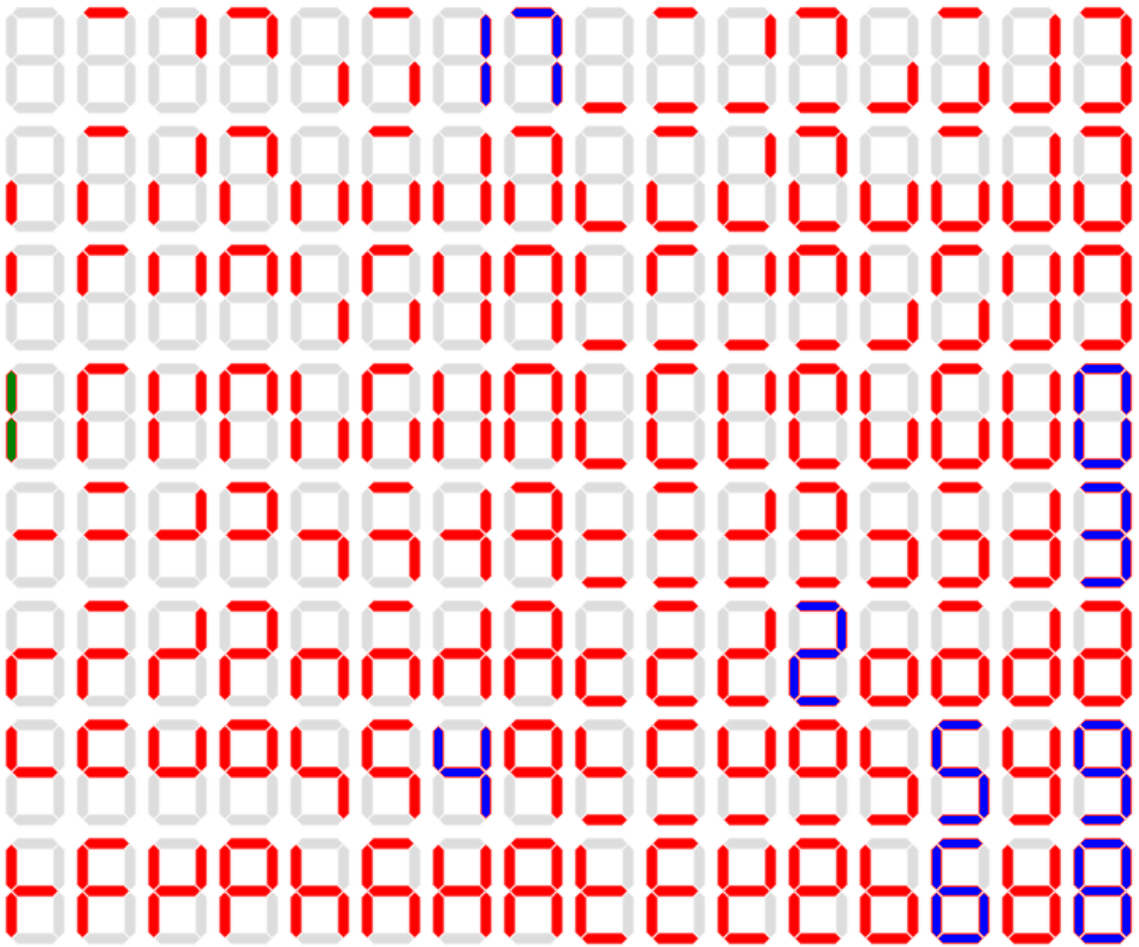
Hogyan működik a kijelzés?

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

A lehetséges állapotok száma:

$2^7 = 128$

Ezek közül nem mindegyik értelmes kombináció.

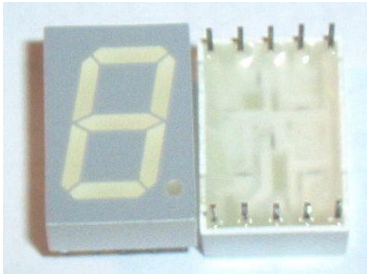


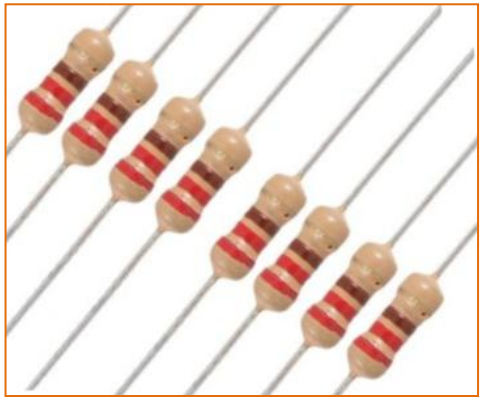
Lab12

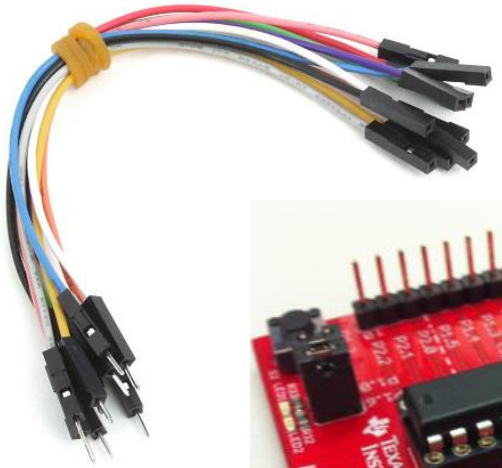
- **LED_7segment** – Egyszerű mintaprogram LED kijelzővel
- **LED_7segment_Multiplexer** – LED kijelző háromvezetékes soros meghajtással és multiplex megjelenítéssel
- **LED_7segment_2digit** – kétszámjegyű LED kijelző, soros meghajtással és multiplex megjelenítéssel
- **LED_7segment_thermometer** – hőmérő alkalmazás a kétszámjegyű LED kijelző felhasználásával

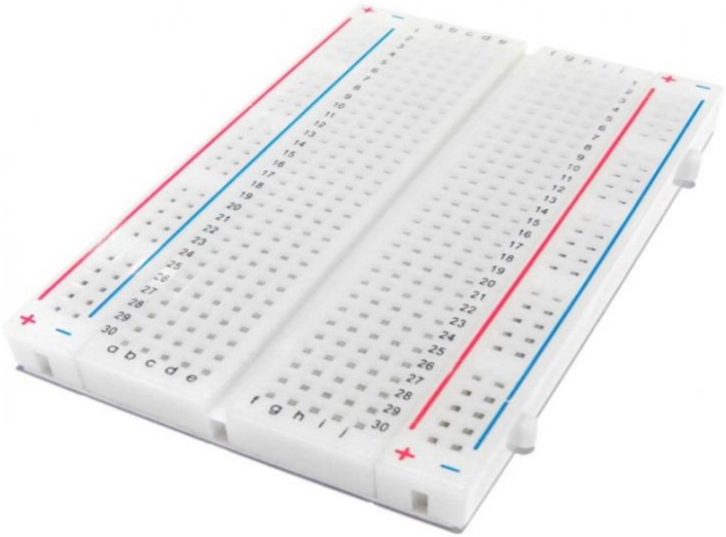
Hozzávalók

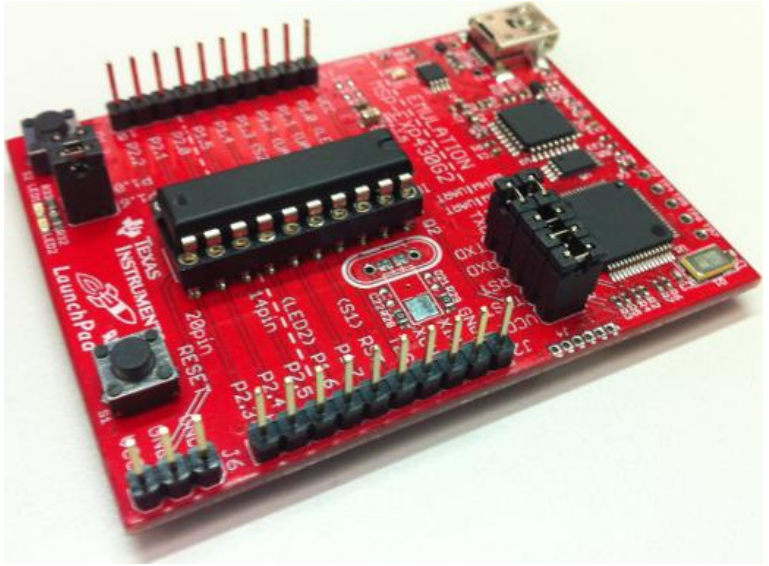
1 db hétszegmenses LED kijelző
8 db átkötő vezeték (F + M)
7 db 220R ellenállás
1 db dugaszolós próbapanel
1 db Launchpad kártya









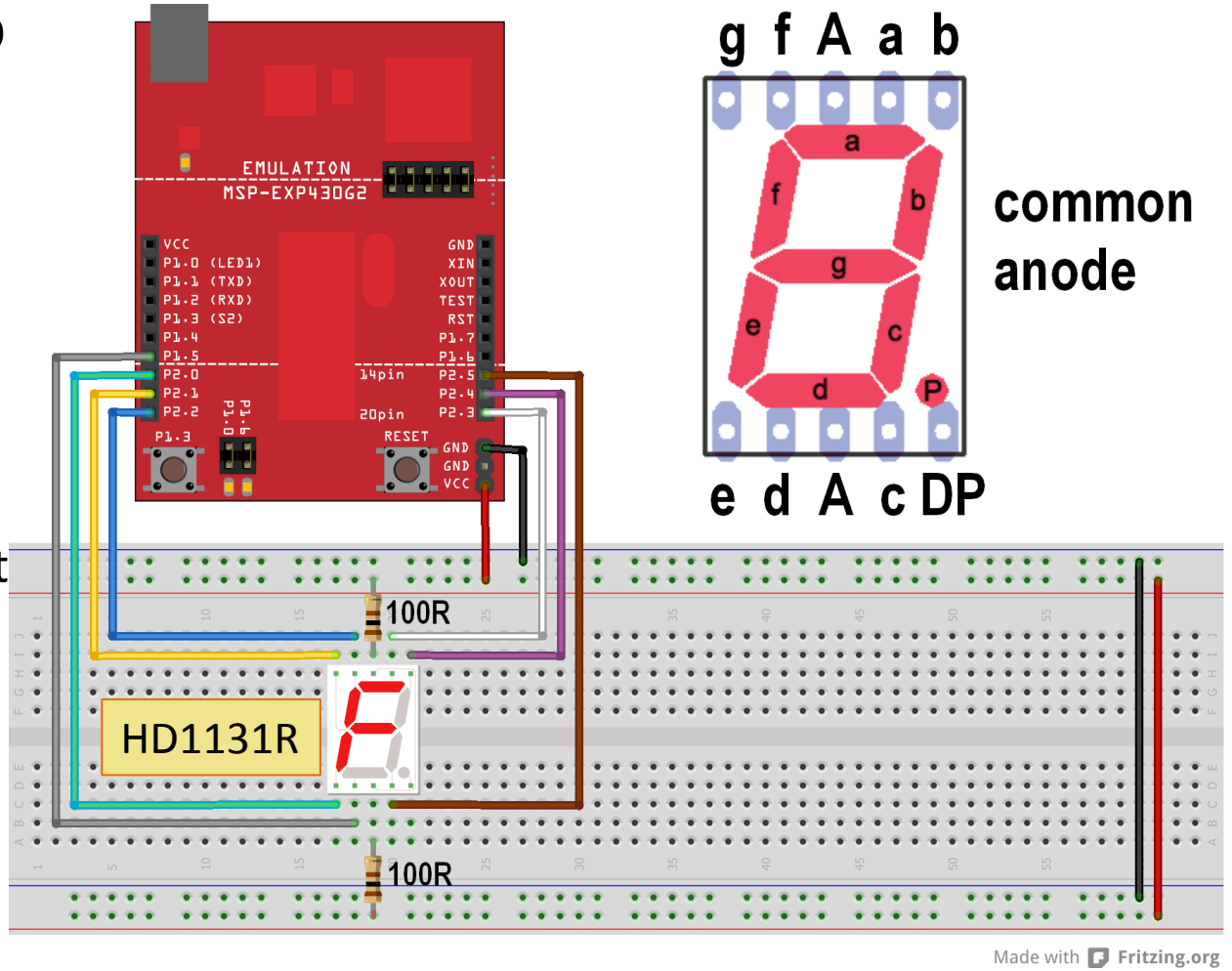


Egyszerűsített kapcsolás
(nem korrekt megoldás…)

A szegmenseket egy-egy I/O kimenet felhasználásával vezérlejük . A szegmens kivezetés a katód, tehát lehúzással (LOW állapot) lehet kigyújtani.

A tizedespont kijelzőt (DP) most nem használjuk.

A közös anódnál alkalmazott áramkorlátozás azért nem korrekt, mert a szegmensek közötti árammegosztás esetleges. Előfordulhat, hogy valamelyik LED ki sem gyullad…

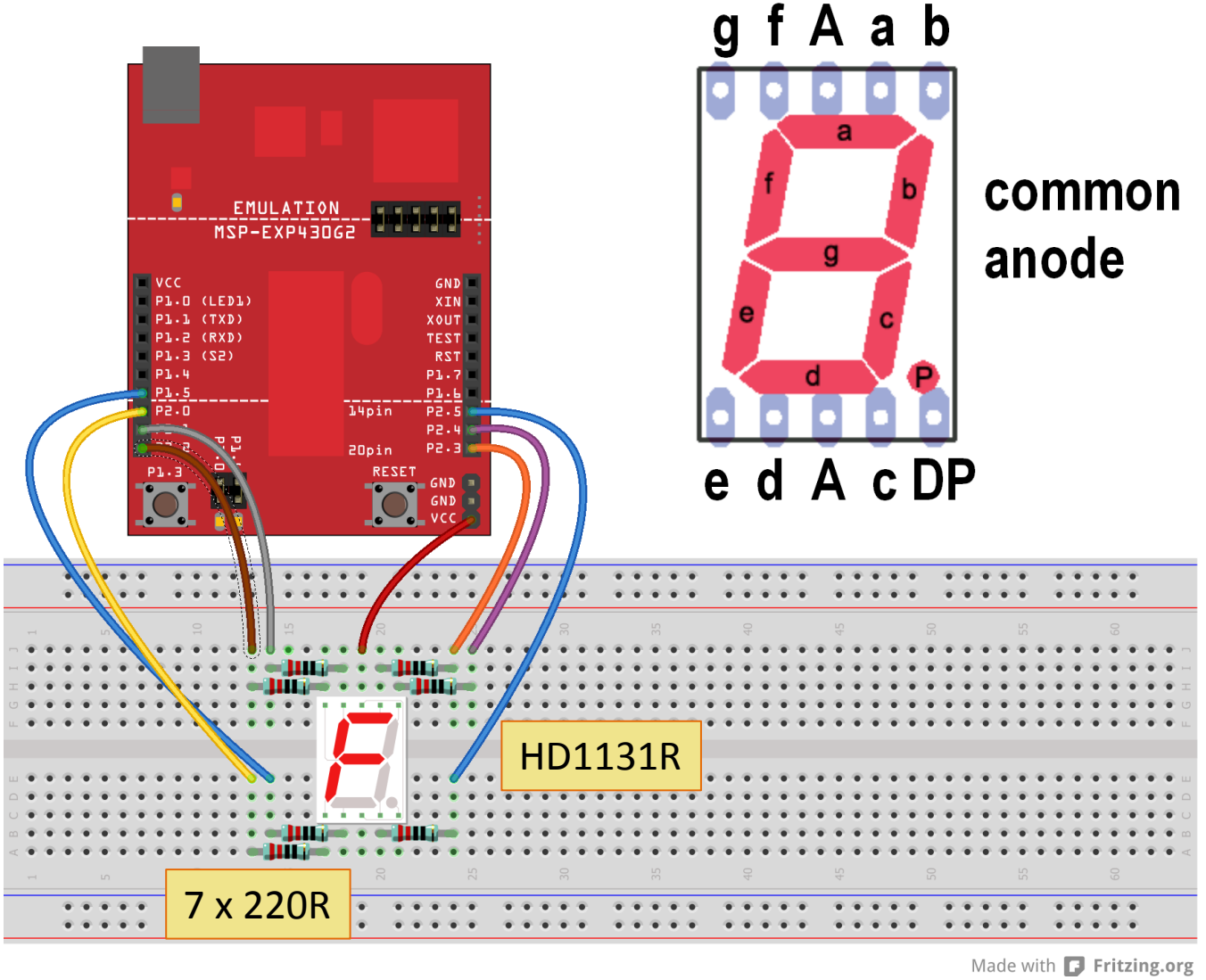


A korrekt kapcsolás

A szegmenseket egyenként egy-egy áramkorlátozó ellenálláson keresztül húzzuk földre (LOW állapot).

Az árammegosztás így megfelelő.

A megoldás hátránya, hogy így 7 db ellenállás kell.
Az ellenállások értéke 220-330 Ohm legyen.



LED_7segment.ino

```
#define segA   P2_3
#define segB   P2_4
#define segC   P2_5
#define segD   P1_5
#define segE   P2_0
#define segF   P2_1
#define segG   P2_2

void setup(){
  pinMode(segA, OUTPUT);
  pinMode(segB, OUTPUT);
  pinMode(segC, OUTPUT);
  pinMode(segD, OUTPUT);
  pinMode(segE, OUTPUT);
  pinMode(segF, OUTPUT);
  pinMode(segG, OUTPUT);
  clr();
  delay(1000);
}
```

```
void loop(){
  zero();  delay(1000);
  one();  delay(1000);
  two();  delay(1000);
  three();  delay(1000);
  four();  delay(1000);
  five();  delay(1000);
  six();  delay(1000);
  seven();  delay(1000);
  eight();  delay(1000);
  nine();  delay(1000);
}

…folyt. Köv.
```

Forrás: http://makezine.com/projects/drive-a-7-segment-led-with-an-arduino/

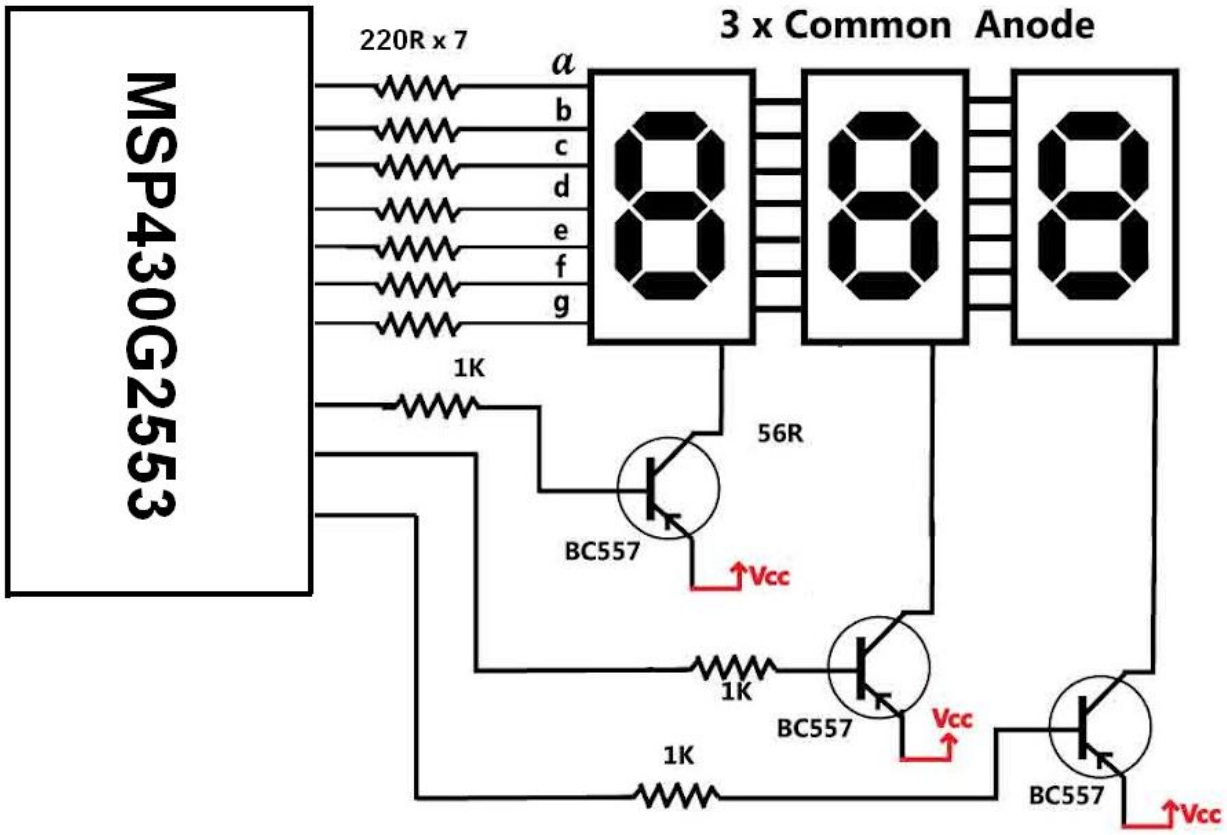
```
void one(){
  //Displays 1
  digitalWrite(segD, HIGH);
  digitalWrite(segE, LOW);
  digitalWrite(segF, LOW);
  digitalWrite(segG, HIGH);
  digitalWrite(segA, HIGH);
  digitalWrite(segB, HIGH);
  digitalWrite(segC, HIGH);
}

void two(){
  //Displays 1
  digitalWrite(segD, LOW);
  digitalWrite(segE, LOW);
  digitalWrite(segF, HIGH);
  digitalWrite(segG, LOW);
  digitalWrite(segA, LOW);
  digitalWrite(segB, LOW);
  digitalWrite(segC, HIGH);
}
```

```
void three(){
  //Displays 1
  digitalWrite(segD, LOW);
  digitalWrite(segE, HIGH);
  digitalWrite(segF, HIGH);
  digitalWrite(segG, LOW);
  digitalWrite(segA, LOW);
  digitalWrite(segB, LOW);
  digitalWrite(segC, LOW);
}

void four(){
  //Displays 1
  digitalWrite(segD, HIGH);
  digitalWrite(segE, HIGH);
  digitalWrite(segF, LOW);
  digitalWrite(segG, LOW);
  digitalWrite(segA, HIGH);
  digitalWrite(segB, LOW);
  digitalWrite(segC, LOW);
}


…és így tovább
```

Forrás: http://makezine.com/projects/drive-a-7-segment-led-with-an-arduino/

Többszámjegyű kijelzés

A felhasznált kivezetések minimalizálására a szegmenseket párhuzamosan kötjük, egyidejűleg azonban csak egy digit lehet aktív (multiplex kijelzés). A digitek vezérléshez közös anódú kijelzőknél **PNP** tranzisztor kell.



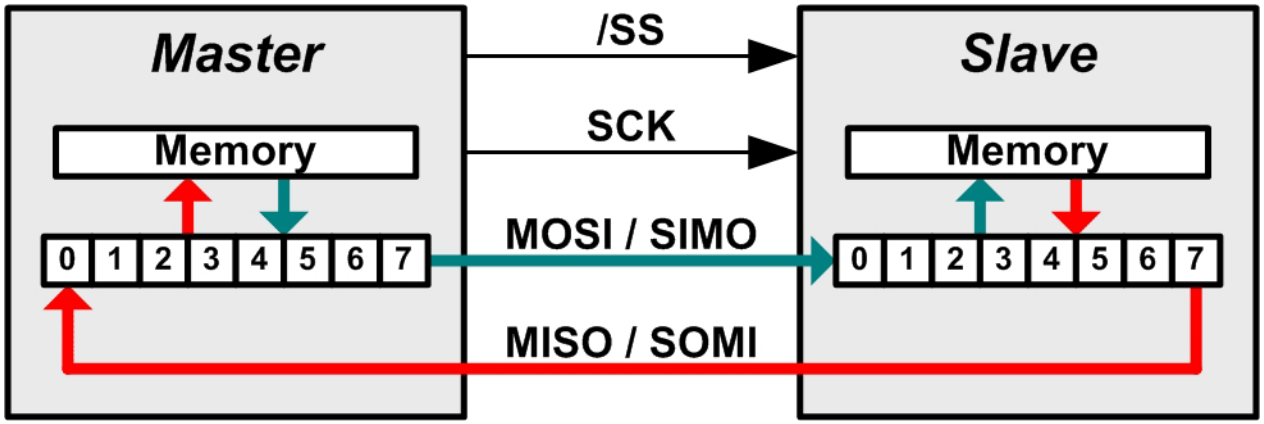
Soros meghajtás, SPI kommunikáció

A kijelző szegmenseinek vezérlése sok kivezetést lefoglal.

Megoldási lehetőségek:

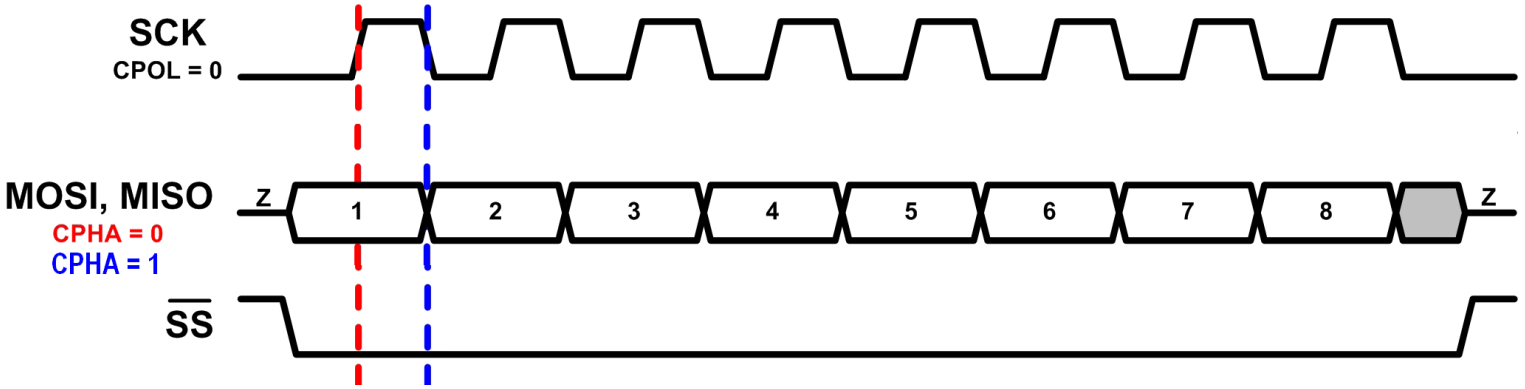
- Soros vezérlésű perifériabővítő IC-k használata
- Multiplex kijelzés
- Speciális, LED kijelző vezérlő IC-k használata (ez is soros vezérlésű)

SPI – Serial Peripheral Interface
Jellemzői: kétirányú szinkron soros kommunikáció. Az órajel és az adatjel mellett több slave esetén kiválasztójel is kell.



Ábra forrása: http://www.eetimes.com/document.asp?doc_id=1272534

SPI jelalak



Az órajel lehet pozitív (CPOL = 0), vagy negatív (CPOL = 1), az adatvonalak bekapuzása történhet az órajel homlokélénél (CPHA = 0), vagy a lefutó élénél (CPHA = 1).

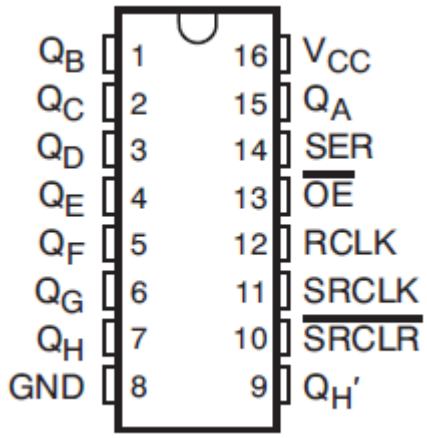
MOSI – Master out/Slave in, vagyis a master → slave irányú kommunikáció adatvonala

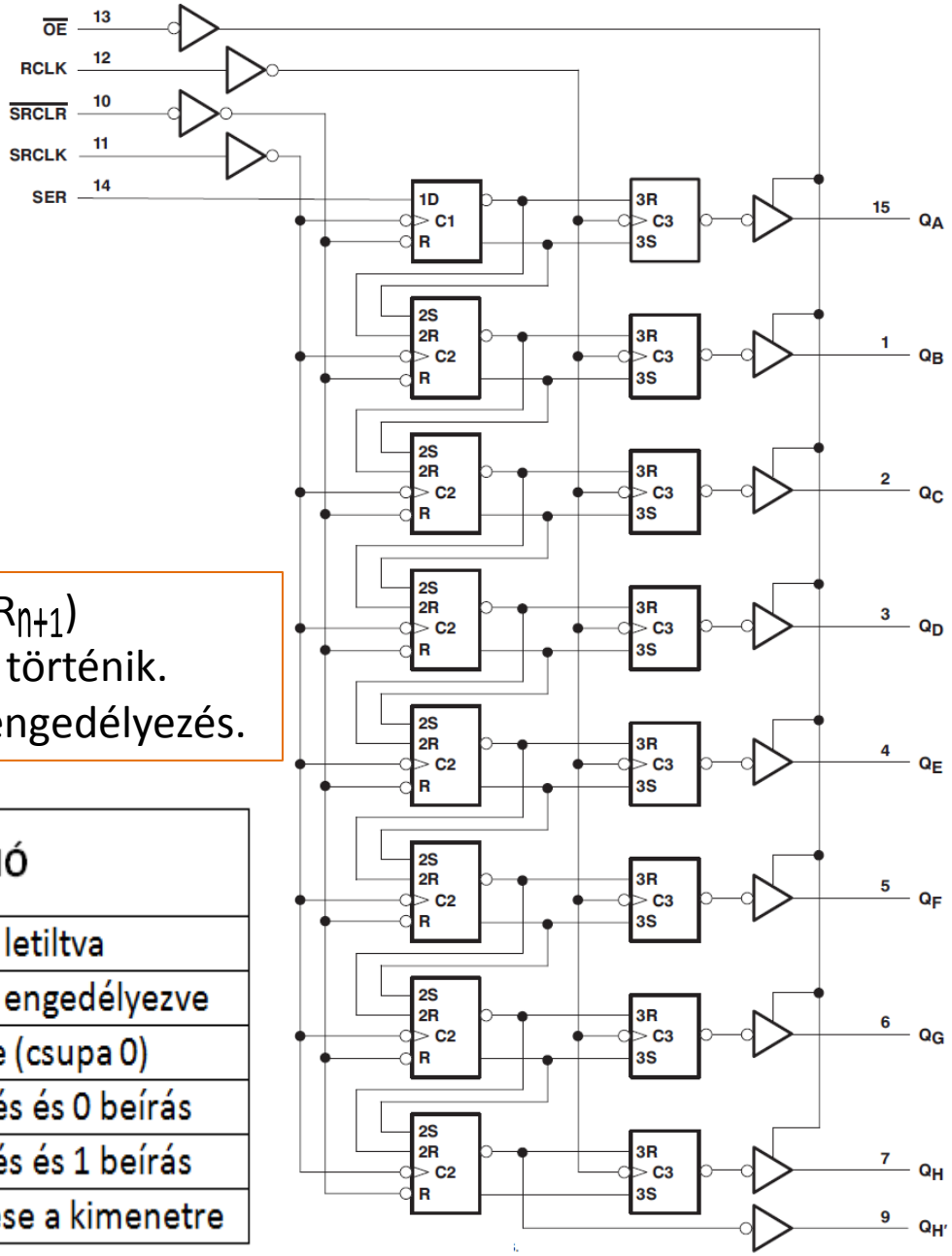
MISO – Master in/Slave out, vagyis a slave → masetr irányú adatáramlás vezetéke

Eszközök:

- Shift regiszterek (pl. 75HC165, 74HC164, 74HC595)
- Periféria bővítők (pl. MCP23S17, MCP23S08)
- LED vezérlők (pl. TLC5940, MAX7219)

SN74HC595





A **74HC595** IC-k sorbaköthetőek ($QH'_{n} \rightarrow SER_{n+1}$)
Az adatok áttöltése az **RCLK** jel felfutó élénél történik.
SRCLR = shift regiszter törlés, **OE** = kimenet engedélyezés.

BEMENETEK					FUNKCIÓ
SER	SCLK	SCLR	RCLK	OE	
X	X	X	X	H	A Q_A – Q_H kimenetek letiltva
X	X	X	X	L	A Q_A – Q_H kimenetek engedélyezve
X	X	L	X	X	Shift regiszter törlése (csupa 0)
L	↑	H	X	X	Shift regiszter léptetés és 0 beírás
H	↑	H	X	X	Shift regiszter léptetés és 1 beírás
X	X	X	↑	X	Shift regiszter áttöltése a kimenetre

Hétszegmenses LED kijelző vezérlése 74HC595 shift regiszterrel

Hozzávalók:

1-2 db hétszegmenses LED kijelző

1-2 db 74HC595 IC

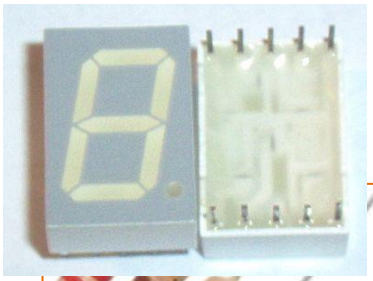
5 db átkötő vezeték (F + M)

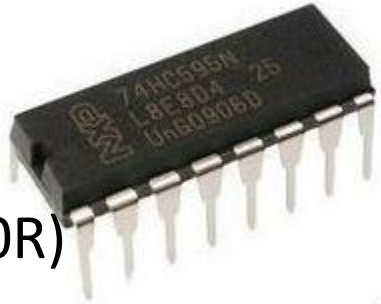
12 db átkötő vezeték (M+M)

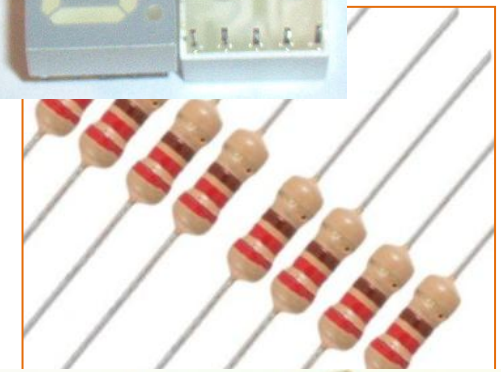
7 db 220R ellenállás (vagy 1db 100R)

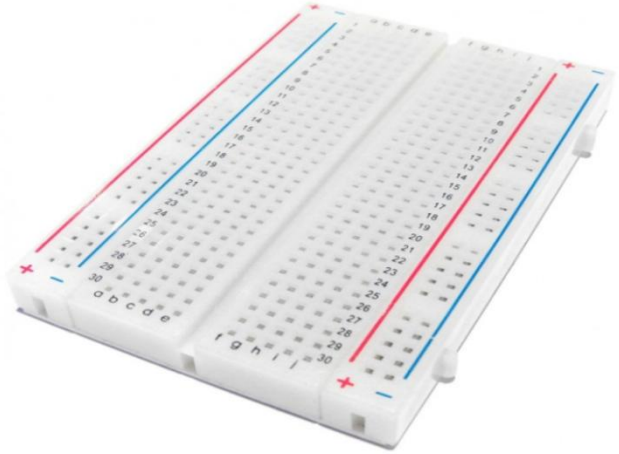
1 db dugaszolós próbapanel

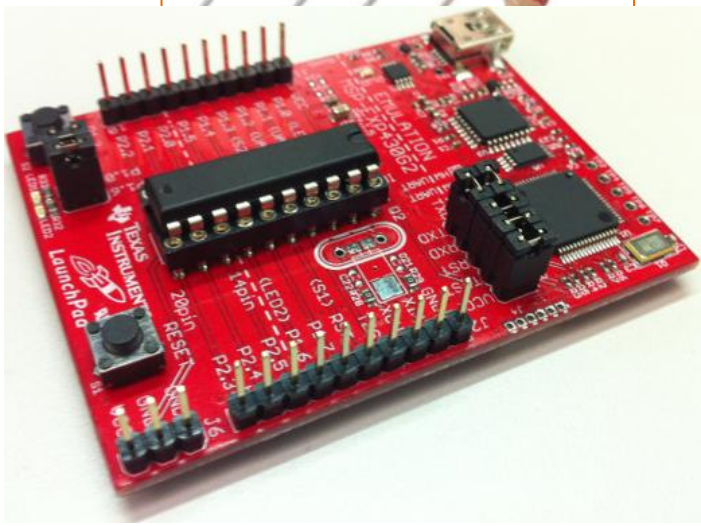
1 db Launchpad kártya



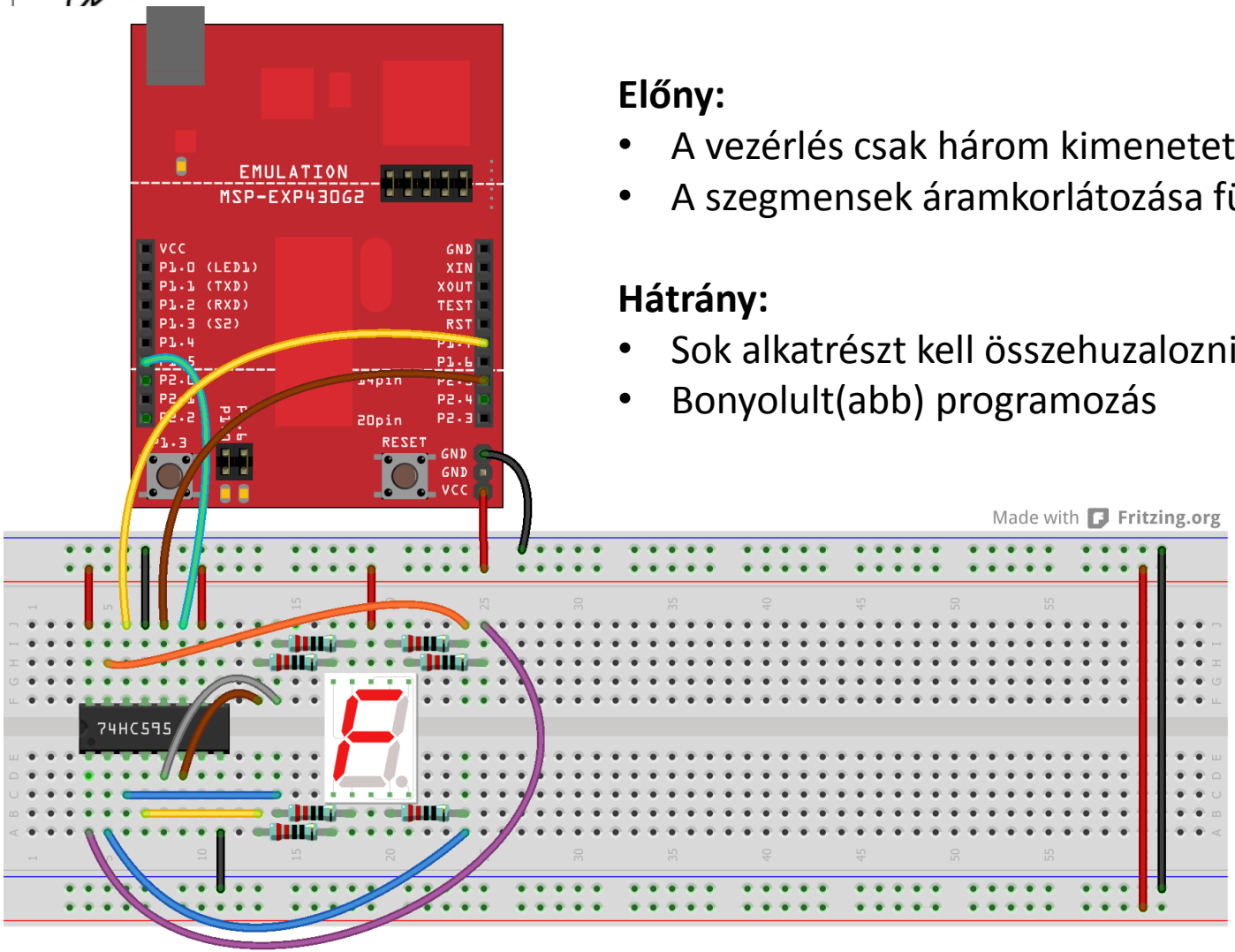








A korrekt kapcsolás



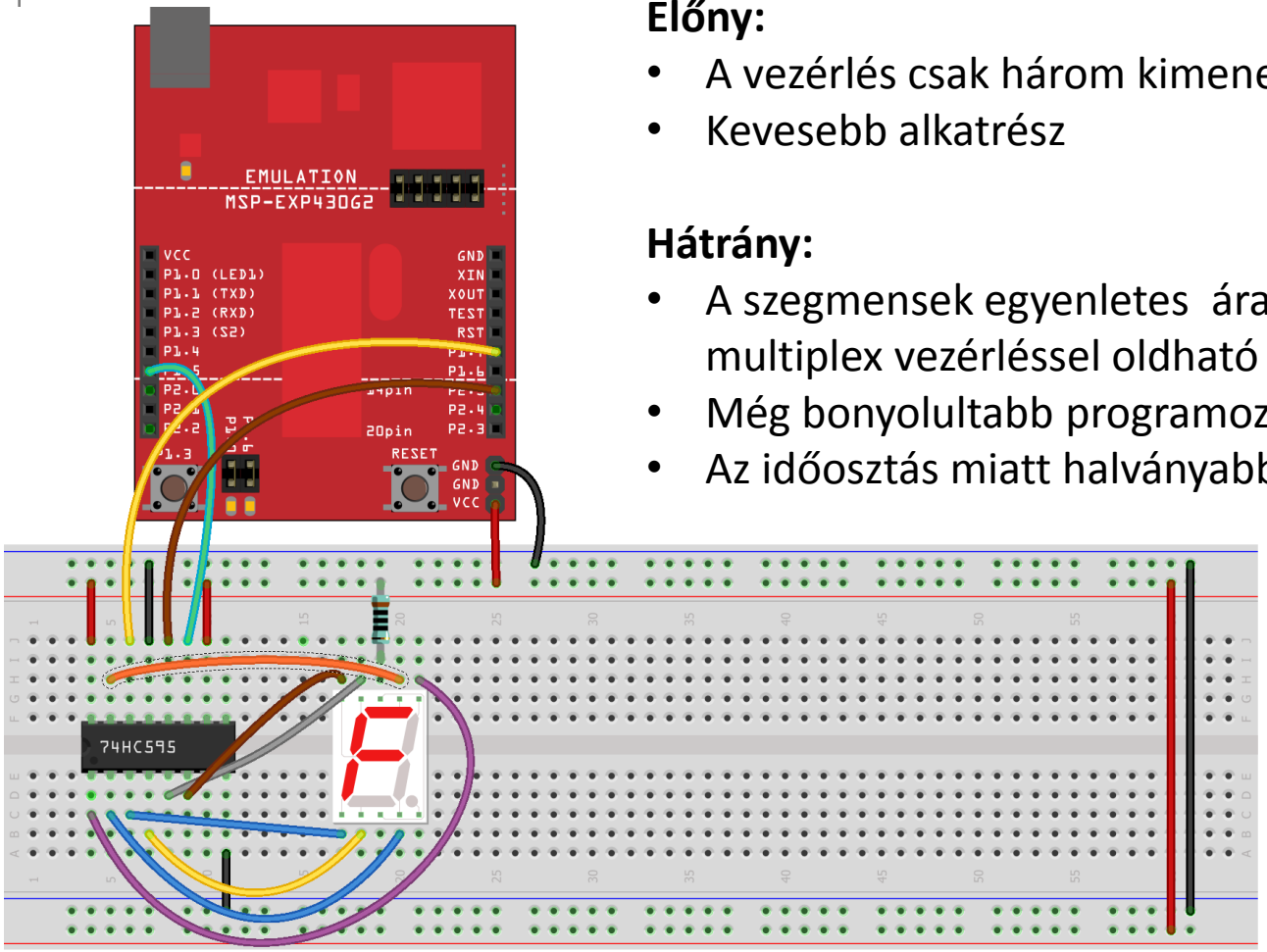
Előny:

- A vezérlés csak három kimenetet foglal
- A szegmensek áramkorlátozása független

Hátrány:

- Sok alkatrészt kell összehuzalozni (alkatrésztemető)
- Bonyolult(abb) programozás

Az egyszerűsített kapcsolás



Előny:

- A vezérlés csak három kimenetet foglal
- Kevesebb alkatrész

Hátrány:

- A szegmensek egyenletes árammegosztása csak multiplex vezérléssel oldható meg
- Még bonyolultabb programozás
- Az időosztás miatt halványabb kijelzés

LED_7segment_Multiplexer.ino

Forrás: http://bit.ly/sOft87

```
const int clockPin = 7;   //P1_5
const int dataPin  = 15;  //P1_7
const int latchPin = 13;  //P2_5
unsigned long t1;
unsigned long t2;
int i = 0;

void setup(){
  t1 = millis();
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  pinMode(latchPin, OUTPUT);
}
void loop() {
    t2 = millis();
    if(t2 - t1 > 1000) {
      i++;
      t1 = t2;
      if(i > 15) i = 0;
    }
    show(numbers[i]);
}
```

A numbers[16] tömb a hexadecimális karaktereket definiálja 0-tól F-ig

```
const byte numbers[16] = {
      //abcdefg
      0b11111100,
      0b01100000,
      0b11011010,
      0b11110010,
      0b01100110,
      0b10110110,
      0b10111110,
      0b11100000,
      0b11111110,
      0b11100110,
      0b11101110,
      0b00111110,
      0b10011100,
      0b01111010,
      0b10011110,
      0b10001110
};
```

LED_7segment_Multiplexer.ino

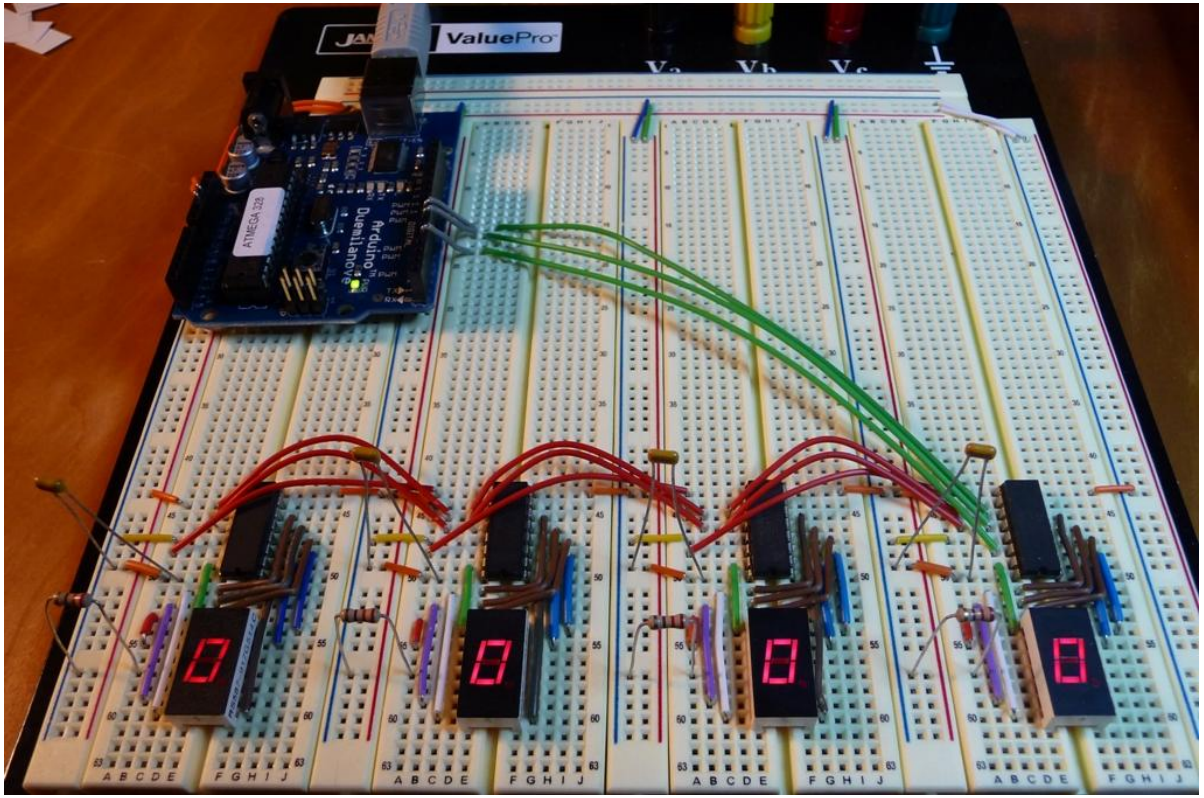
(folytatás) Forrás: http://bit.ly/sOft87

```
void show( byte number){
  for(int j=0; j<=7; j++)  {
    byte toWrite = number & (0b10000000 >> j);
    if(!toWrite) continue;
    shiftIt(toWrite);
  }
}

void shiftIt (byte data){
    digitalWrite(latchPin, LOW);
    for (int k=0; k<=7; k++) {
      digitalWrite(clockPin, LOW);
      if ( data & (1 << k) ) {
        digitalWrite(dataPin, LOW);  // turn "On"
      } else {
        digitalWrite(dataPin, HIGH); // turn "Off"
      }
      digitalWrite(clockPin, HIGH);
    }
    digitalWrite(clockPin, LOW);
    digitalWrite(latchPin, HIGH};
}
```

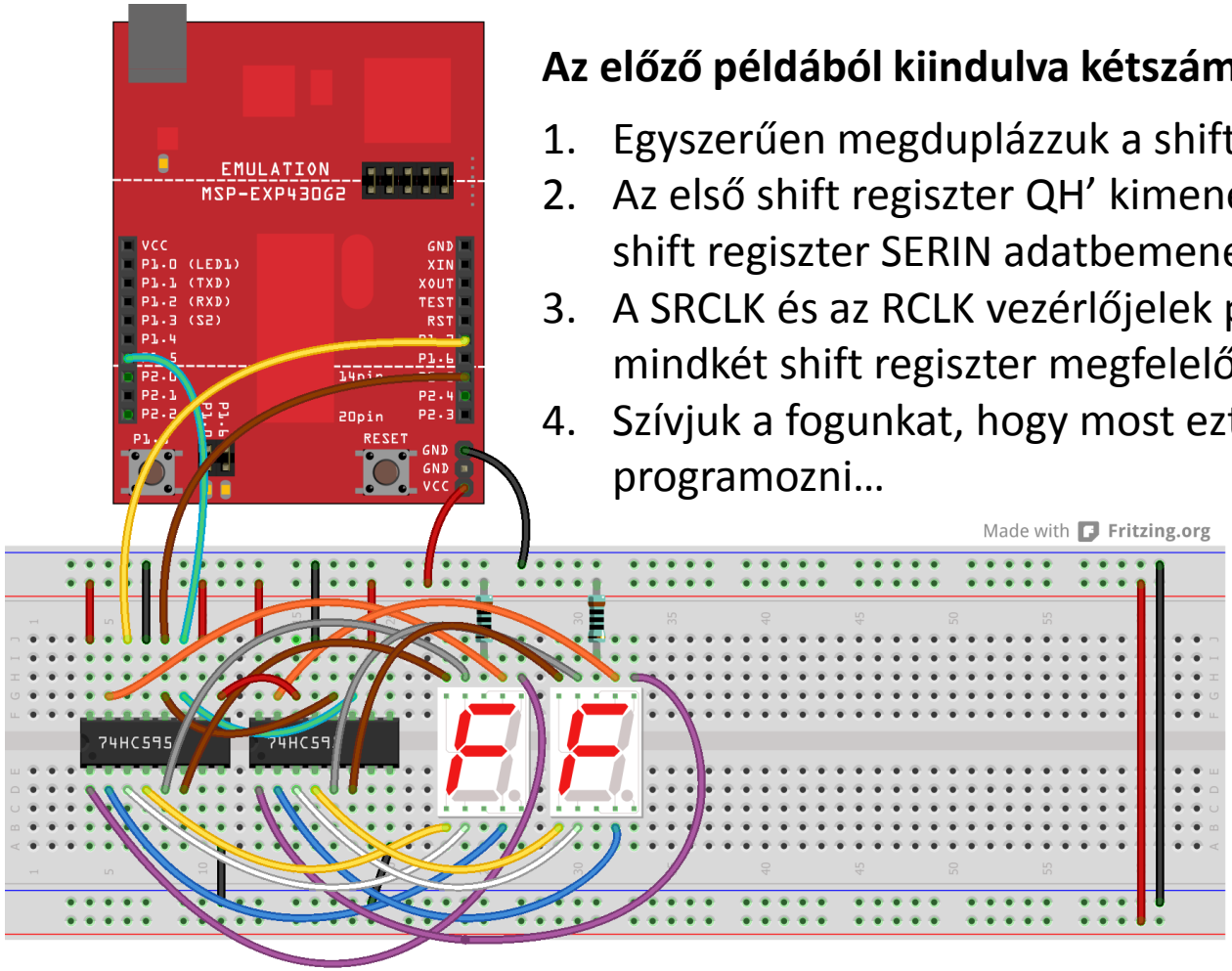
Szegmensenként történik a megjelenítés, itt ezt jelenti a multiplex kijelzés.
Korrekt kapcsolás esetén csak **`shiftIt(number)`** kellene!

Többszámjegyű kijelző

A shift regiszterek sorbafűzésével könnyen bővíthetjük a kijelzőt 2-3-4 számjegyűre.
Az alábbi képen egy négyszámjegyű kijelző látható.
Forrás: Arduino: cascading shift registers to drive 7-segment displays with PC input
Mi ennél szerényebb célt tűzünk ki: 2 számjegyű kijelzőt építünk.



Kétszámjegyű kijelző



Az előző példából kiindulva kétszámjegyű kijelzőt készítünk:

1. Egyszerűen megduplázzuk a shift regiszter + kijelző együttest.
2. Az első shift regiszter QH' kimenetét összekötjük a második shift regiszter SERIN adatbemenetével.
3. A SRCLK és az RCLK vezérlőjelek párhuzamosan eljutnak mindkét shift regiszter megfelelő bemeneteire.
4. Szívjuk a fogunkat, hogy most ezt hogyan is kellene programozni…

A tisztességes megoldás természetesen az volna, ha minden szegmensre egy-egy áramkorlátozó ellenállást tennénk (összesen 14 darabot). Így nehezebb lesz…

LED_7segment_2digit.ino

Forrás: Arduino: cascading shift registers to drive 7-segment displays with PC input

```
const int  g_pinCommLatch = 13;
const int  g_pinClock     =  7;
const int  g_pinData      = 15;
const byte g_digits [10] = {
      //abcdefg
      0b11111100,  //0
      0b01100000,  //1
      0b11011010,  //2
      0b11110010,  //3
      0b01100110,  //4
      0b10110110,  //5
      0b10111110,  //6
      0b11100000,  //7
      0b11111110,  //8
      0b11100110,  //9
};
int g_numberToDisplay = 100;    // Current number being displayed
const int g_registers = 2;      // Number of shift registers in use
void setup() {
  pinMode(g_pinCommLatch, OUTPUT);
  pinMode(g_pinClock, OUTPUT);
  pinMode(g_pinData, OUTPUT);
} // setup                                          Folyt. Köv...
```

LED_7segment_2digit.ino

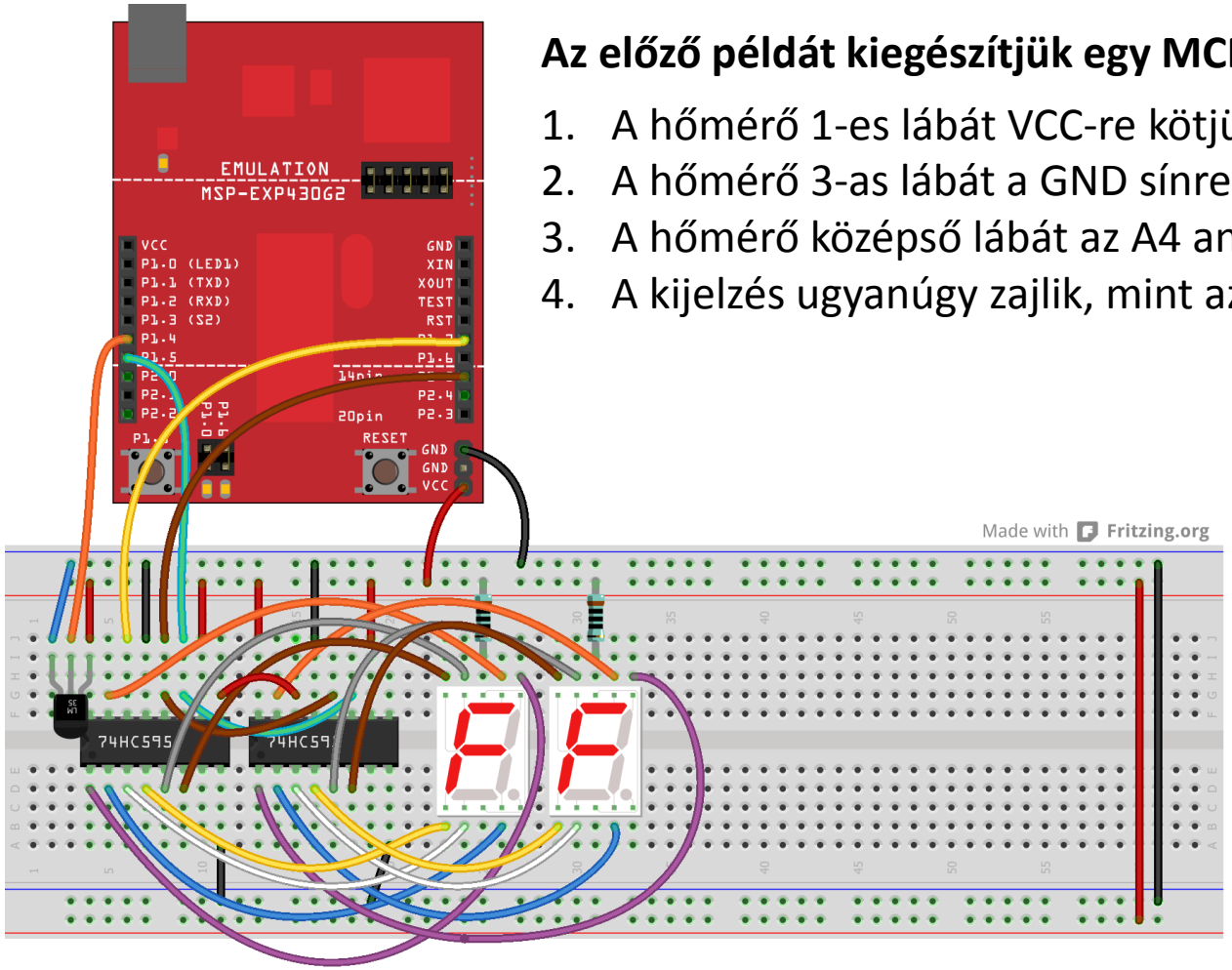
(Folytatás)

```
void setup() {
  if (--g_numberToDisplay <0 ) g_numberToDisplay= 100;
  g_registerArray[0] = g_digits[(g_numberToDisplay)/10];
  g_registerArray[1] = g_digits[g_numberToDisplay%10];
  for(int k=0; k<100; k++) sendSerialData(g_registers,g_registerArray);
} // loop

void sendSerialData(byte registerCount, byte *pValueArray) {
  for (byte bitMask = 128; bitMask > 0; bitMask >>= 1) {
    digitalWrite(g_pinCommLatch, LOW);
    for (byte reg = registerCount; reg > 0; reg--) {
      byte value = pValueArray [reg - 1] & bitMask;
      for(byte bm=0; bm<8; bm++) {
        digitalWrite(g_pinClock, LOW);
        digitalWrite(g_pinData, value & 1 ? LOW : HIGH);
        digitalWrite(g_pinClock, HIGH);
        value>>=1;
      }
    }
    digitalWrite(g_pinCommLatch, HIGH);
  }
}  // sendSerialData
```

Csak a multiplex kijelzés miatt kellenek!

Ha value 0. bitje = 1, akkor LOW, Különben HIGH értéket ad át!

Hőmérő LED kijelzővel



Az előző példát kiegészítjük egy MCP9700A analóg hőmérővel:

1. A hőmérő 1-es lábát VCC-re kötjük.
2. A hőmérő 3-as lábát a GND sínre kötjük.
3. A hőmérő középső lábát az A4 analóg bemenetre kötjük.
4. A kijelzés ugyanúgy zajlik, mint az előző példában.

A tisztességes megoldás most is az volna, ha minden szegmensre egy-egy áramkorlátozó ellenállást tennénk (összesen 14 darabot). Így multiplexelni kell…

LED_7segment_2digit.ino

Az előző programhoz képest csak az alábbi függvények módosulnak:

```
void setup() {
      analogReference(INTERNAL2V5);  // Use the 2,5 V inner reference
      pinMode(g_pinCommLatch, OUTPUT);
      pinMode(g_pinClock, OUTPUT);
      pinMode(g_pinData, OUTPUT);
      g_numberToDisplay = 0;
      g_registerArray[0] = g_digits [(g_numberToDisplay)/10];
      g_registerArray[1] = g_digits [g_numberToDisplay%10];
} // setup
void loop() {
       long  mysum = 0;
       for(int i=0; i<2500; i++) {
          mysum += analogRead (A4);
          sendSerialData(g_registers, g_registerArray);
       }
      long voltage = mysum>>10;       //divide by 1024
      long tempC = (voltage-500)/10; //convert to Celsius
      g_numberToDisplay = (int)tempC;
      g_registerArray[0] = g_digits[(g_numberToDisplay)/10];
      g_registerArray[1] = g_digits[g_numberToDisplay%10];
      for(int k=0; k<500; k++) sendSerialData(g_registers, g_registerArray);
    } // loop
```

Hőmérés átlagolással
Lásd: Talk08 és Lab08
AnalogThermometer3.ino

A hőmérő működése