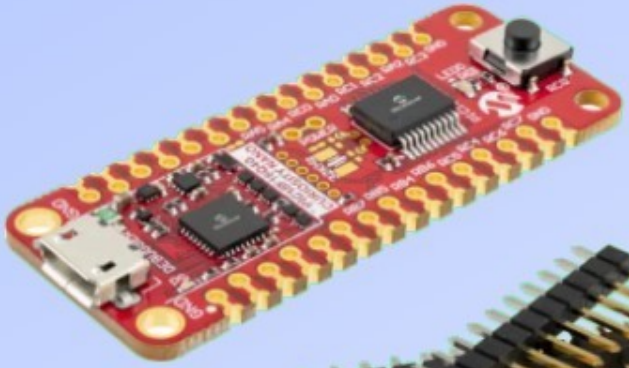
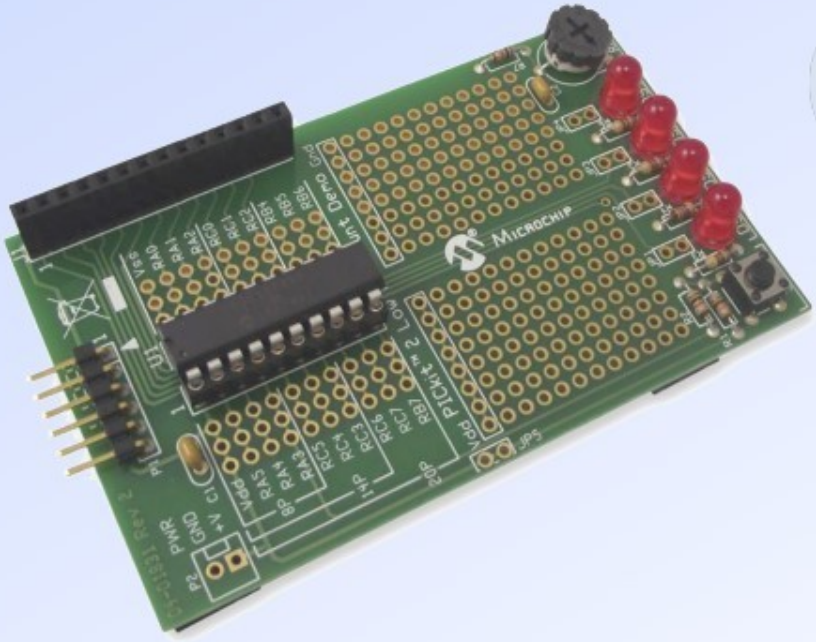


MICROCHIP

PIC mikrovezérlők

4. rész

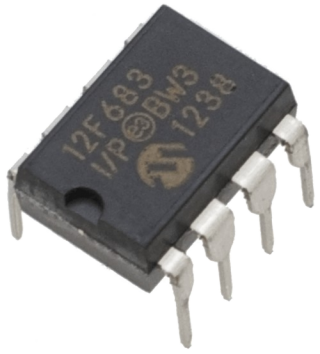


Felhasznált és ajánlott irodalom

- **Milan Verle:** PIC MicrocontrollersProgramming in Assembly
- **Microchip:** PICmicro Mid-Range MCU Family Reference Manual
- **T&T:** Közepes teljesítményű PIC mikrovezérlők Felhasználói Kézikönyv
- **SimulIDE Community:** SimulIDE Tutorials
- **The Jallib Team:**
  - Have fun with PIC microcontrollers, Jal v2 and Jallib
  - Jal v2 Compiler Documentation

Adatlapok:

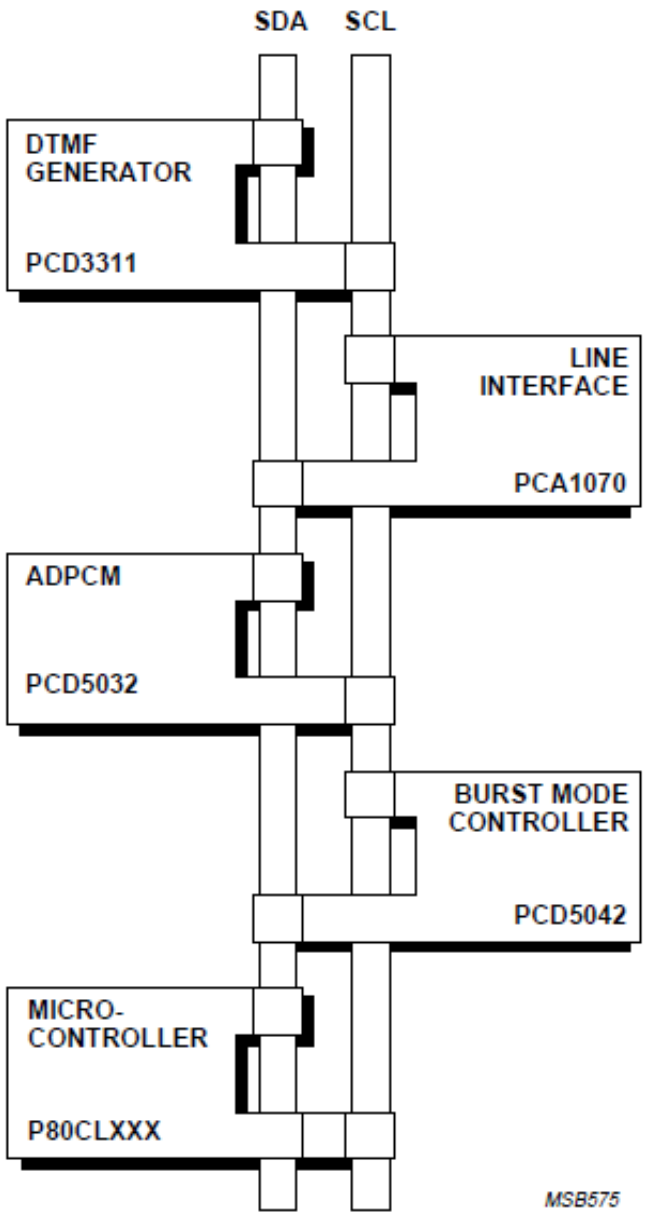
- **Microchip:** PIC12F683 adatlap és termékinfo
- **Analog Devices:** DS3231 real-time óra adatlap
- **Titan Micro Electronics:** TM1637 7-szegmenses LED meghajtó
- **Microchip:** PICkit2 programmer User's Guide
- **Icircuit Technologies:** iCP02v2 USB PIC/EEPROM programmer manual



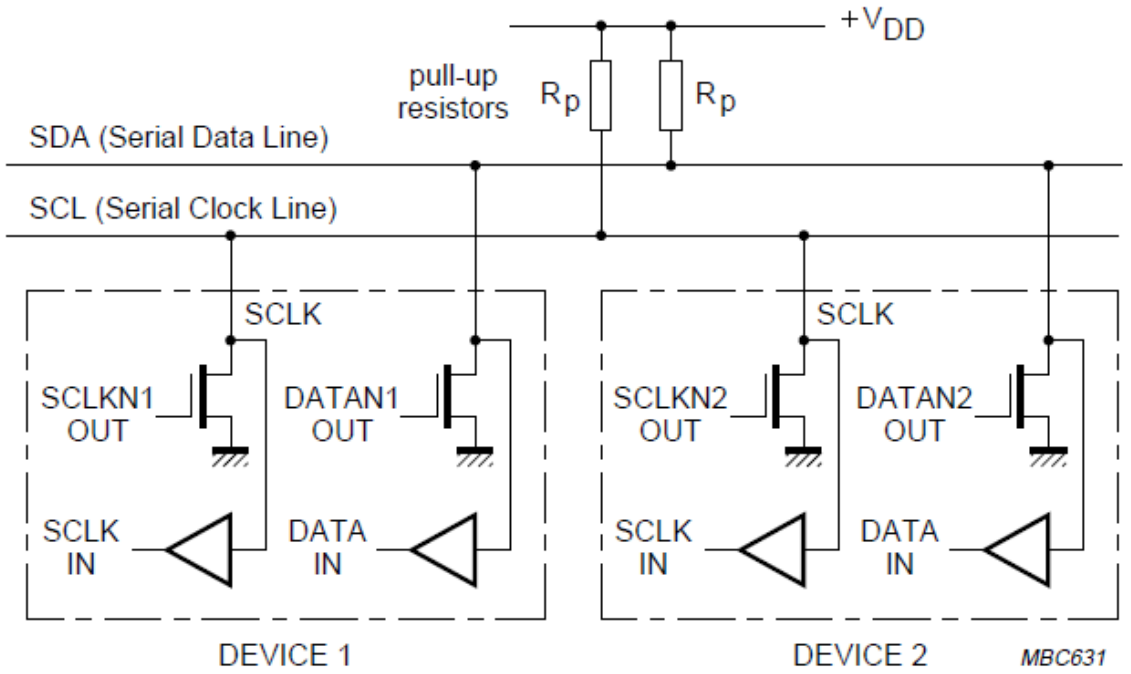
I2C kommunikáció a PIC12F683 mikrovezérlővel

Az I2C busz

- „Inter-Integrated Circuit” busz – vagy „kétvezetékes busz”, amelyet eredetileg a Philips cég dolgozott ki 1982-ben.
- **Több eszköz** (master és slave) **fűzhető fel a buszra**
- A buszt a **mester (master) eszközök** vezérlik és kezdeményezik az adatforgalmat. **A szolga (slave) eszköz** akkor válaszol, ha címzéssel megszólítják
- **Az I²C busz két jelvezetéket használ**
  - **SCL:** szinkronizáló órajel
  - **SDA:** soros adat
- **A részletes leírás** az ”Az I²C-busz specifikációja és használata” című dokumentumban olvasható

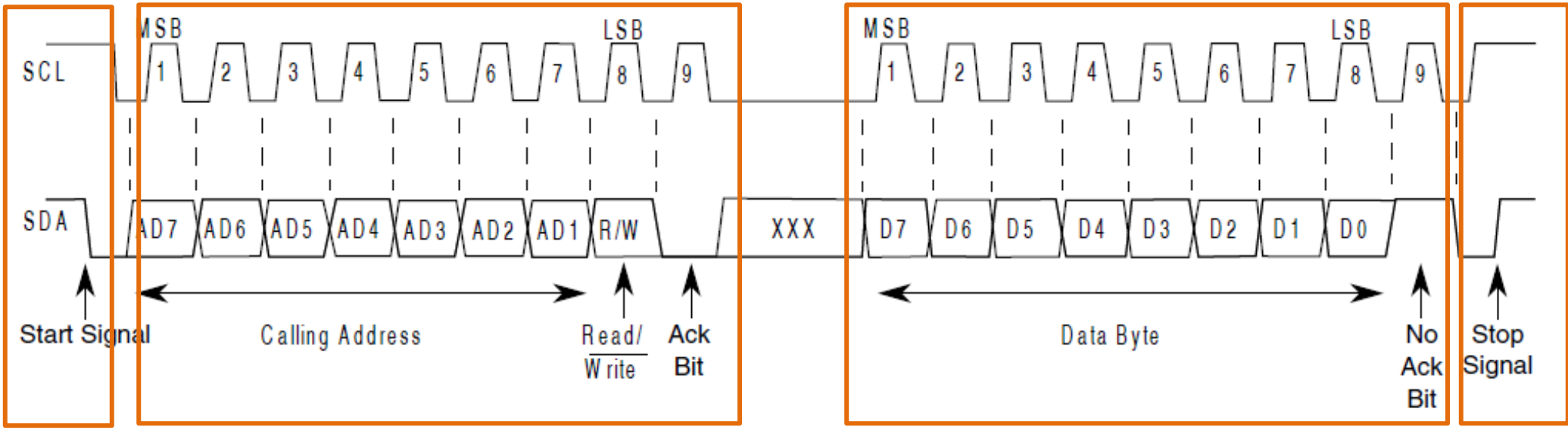


Az I2C busz vezérlése



- A jelvezetékeket ellenállások húzzák fel tápfeszültségre V_{DD}
- Nyitott nyelőelektródás FET-ek húzzák le a vonalakat alacsony szintre
- A buszt vezérlő mester eszköz állítja elő az SCL órajelet
  - normál mód: 100 kHz
  - gyors mód: 400 kHz
  - nagysebességű mód: 1 MHz, vagy több, ez esetben már aktív felhúzással.

I2C üzenetformátum



Üzenet-orientált adatátvitel négy felvonásban:

1. **Start** feltétel
2. **A szolga eszköz megcímzése**
   - 7-bites cím
   - Parancsbit (1: olvasás, 0: írás)
   - Nyugtázás (a vevő visszajelzése)
3. **Adatmező**
   - Adatbájt
   - Nyugtázás (a vevő visszajelzése)
4. **Stop** feltétel

Nyugtázás (ACK): a 9. órajel impulzus tartamára a vevő alacsony szinten tartja az **SDA** jelvezetéket.

Negatív nyugtázás (NAK): a 9. órajel impulzus idején senki sem húzza le az **SDA** jelvezetéket – az magas szinten marad.

i2c_software.jal

- A **PIC12F683** mikrovezérlőnek nincs soros perifériája, ezért csak szoftveres emulációval képes **I2C** kommunikációra (**i2c_sorftware.jal**)
- Az **I2C** busz vonalait ellenállások húzzák fel, az MCU kimenete csak lefele húzhat, vagy inputra állítva „felengedi” a vonalat
- **Inicializálásnál** alacsonyra állítjuk a kimeneti adatregiszter megfelelő bitjeit (SCL és SDA) és bemenetre állítjuk ezen kivezetéseket
- **START feltétel** generálás: magas órajelszint mellett magasról alacsonyra vált az adatkimenet, majd alacsonyra állítjuk az órajel kimenetet is
- **STOP feltétel:** magas órajelszint mellett alacsonyról magasra vált az adatvonal

```
procedure i2c_start() is
   _i2c_wait()
   i2c_sda_direction = high  -- SDA high
   _i2c_wait()
   i2c_scl_direction = high  -- CLK high
   _i2c_wait()
   i2c_sda_direction = low   -- SDA low
   _i2c_wait()
   i2c_scl_direction = low   -- CLK low
   _i2c_wait()
end procedure
```

```
procedure i2c_stop() is
   _i2c_wait()
   i2c_sda_direction = low  -- SDA low
   _i2c_wait()
   i2c_scl_direction = low  -- CLK low
   _i2c_wait()
   i2c_scl_direction = high -- CLK high
   _i2c_wait()
   i2c_scl_direction = high - SDA high
   _i2c_wait()
end procedure
```

i2c_software.jal

Eljárások:

- **i2c_initialize()** – Az I2C busz inicializálása
- **i2c_stop()** – STOP feltétel generálás (tranzakció vége)
- **i2c_start()** – START feltétel generálás (tranzakció kezdődik)
- **i2c_restart()** – RESTART feltétel generálás (tranzakció folytatás)

Függvények:

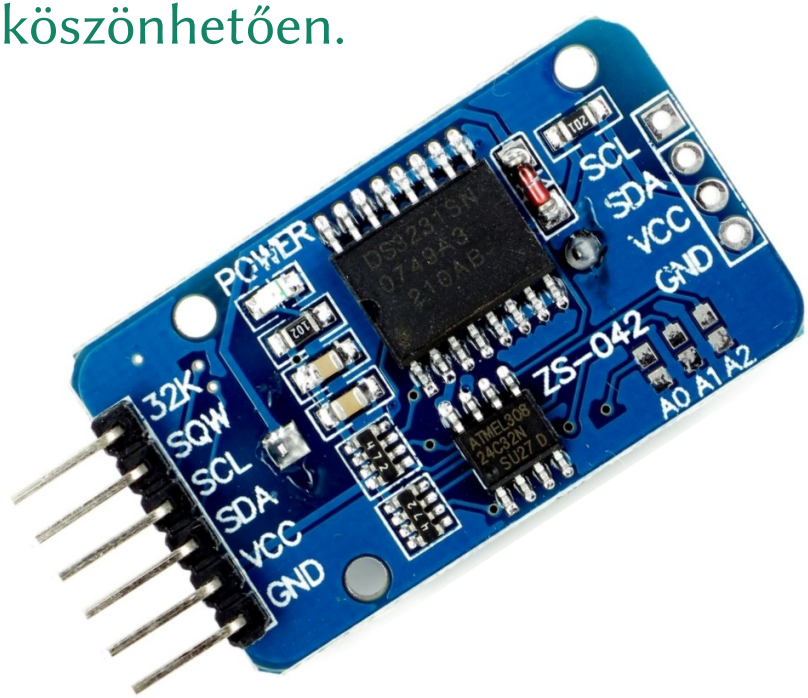
- **i2c_transmit_byte(** *byte in* ***x*** **)** return bit – bájt kiküldése, *ACK* jelzése
- **i2c_receive_byte(***bit in* ***ack*****)** return byte – bájt olvasása, *ACK* küldése

További függvények találhatók az **i2c_level1.jal** programkönyvtárban:

- `i2c_receive_byteaddr(byte in i2c_address, byte in addr, byte in i2c_rx_count) return bit`
- `i2c_send_receive(byte in i2c_address, byte in i2c_tx_count, byte in i2c_rx_count) return bit`
- `i2c_receive_wordaddr(byte in i2c_address, word in addr, byte in i2c_rx_count) return bit`

DS3231 Real-time óra modul

- Oszcillátor, óra és naptár egy tokban. A tápfeszültség megszűnésekor a hátoldalán elhelyezett telepről üzemel tovább.
- A modul többé-kevésbé cserekompatibilis a DS1307-tel, egy 4 kB EEPROM is tartalmaz, de van néhány eltérés:
  - pontosabb 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 4.2 V-os Li akkuval is táplálható!
- EEPROM I2C címe: **0x57** (állítható)
- DS3231 I2C címe: **0x68**



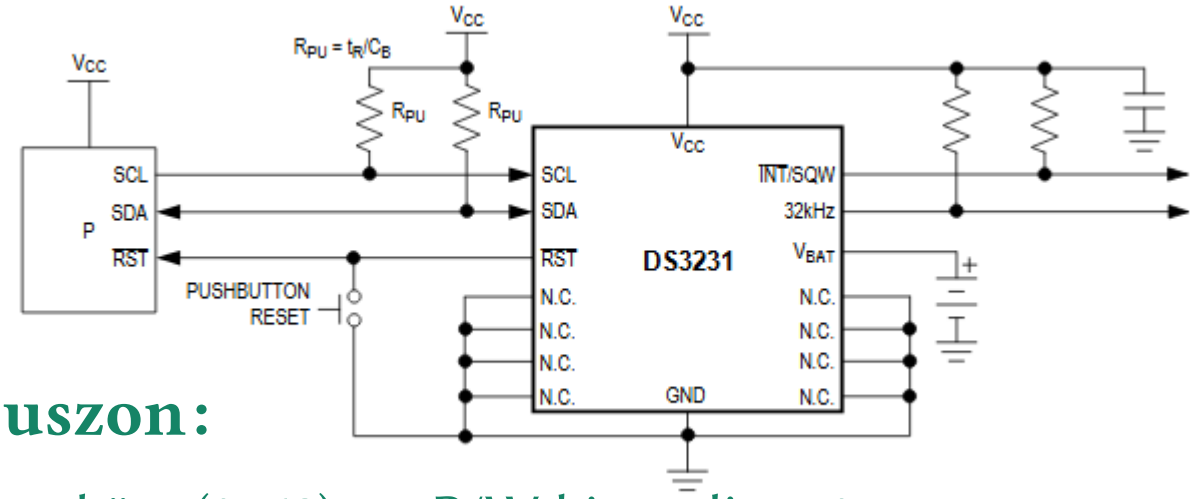
- Adatlap: datasheets.maximintegrated.com/en/ds/DS3231.pdf

DS3231 regiszterek

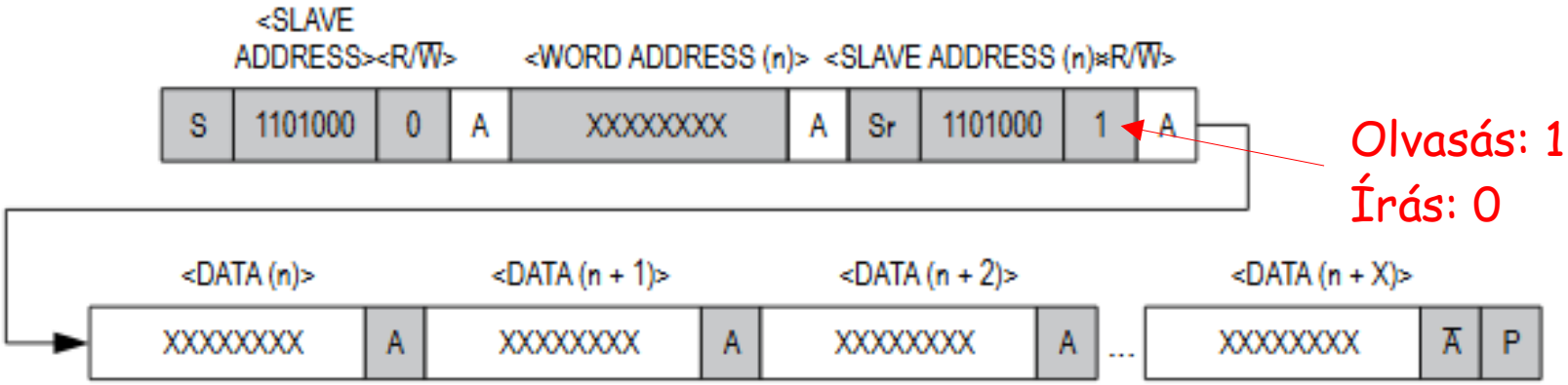
ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{AM}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
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	$\overline{EOSC}$	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	—

A DS3231 bekötése, használata

- Egy tipikus áramköri elrendezés:

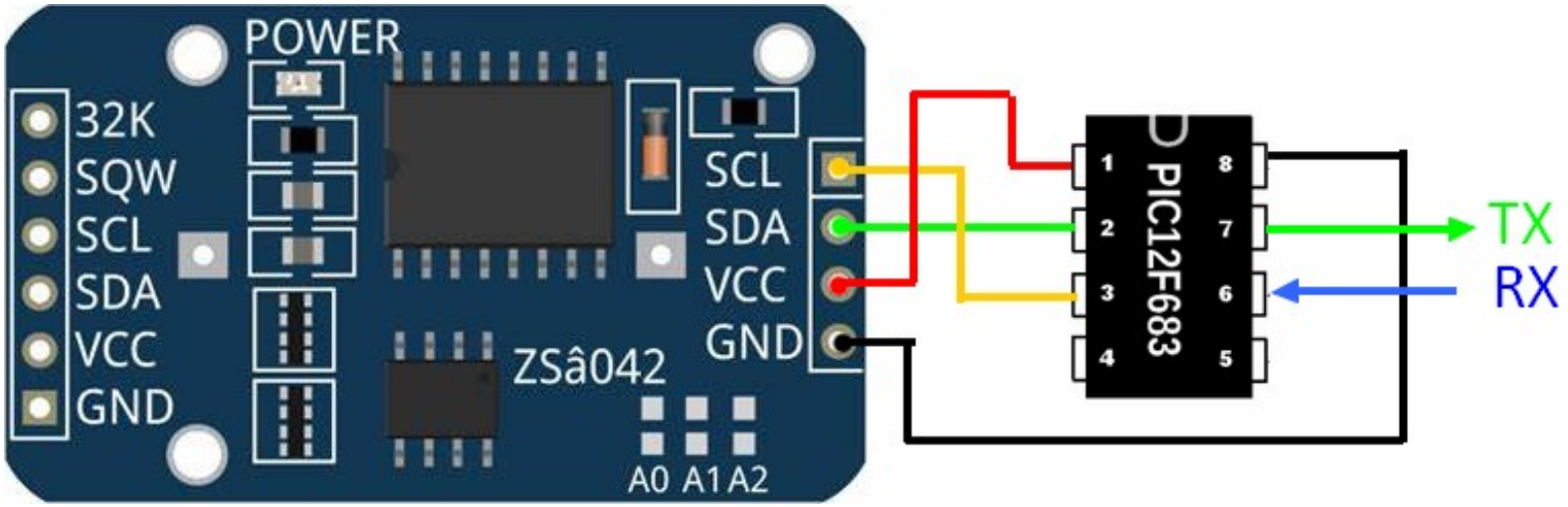


- **Adatforgalom az I2C buszon:**
  - **Íráskor** megcímezzük az eszközt (0x68), az R/W bit pedig = 0.
    A második bájt a regiszter (vagy memória) cím. A többi bájt a kiküldendő adat (a cím automatikusan inkrementálódik)
  - **Olvasáskor** megcímezzük az eszközt (0x68), az R/W bit pedig = 1.
    A többi bájt a beolvasott adat (a cím automatikusan inkrementálódik).



I2C próba

- Az **I2C** kommunikáció kipróbálására most egy **DS3231** RTC modul néhány regiszterét olvassuk ki és íratjuk ki hexadecimálisan a korábbi előadásokban bemutatott szoftveres **UART** emuláció segítségével
- Az alapértelmezett címzés szerint az eszköz 7-bites címe **0x68**
- A bekötési vázlat az alábbi rajzon látható



i2c_test_l0.jal – 3/1.

- A **DS3231** RTC regisztereit olvassuk ki és íratjuk ki soros porton

```
1   include 12f683                                  -- PIC céláramkör
2   pragma target CLOCK     8_000_000               -- oszcillátor frekvencia
3   pragma target OSC       INTOSC_NOCLKOUT         -- belső oszcillátor 8MHz-en
4   pragma target WDT       disabled
5   OSCCON_IRCF   = 0b_111                          -- Fosc = 8 MHz beállítása
6   include print                                   -- Kiírató eljárások
7   enable_digital_io()
8
9   -- Soros port konfigurálása ----------
10  alias serial_sw_tx_pin           is pin_A0
11  alias serial_sw_tx_pin_direction is pin_A0_direction
12  serial_sw_tx_pin_direction = output
13  alias serial_sw_rx_pin           is pin_A1
14  alias serial_sw_rx_pin_direction is pin_A1_direction
15  serial_sw_rx_pin_direction = input
16  const serial_sw_baudrate = 9600
17  const serial_sw_invert = true
18  include serial_software
19  serial_sw_init()
20
```

i2c_test_l0.jal – 3/2.

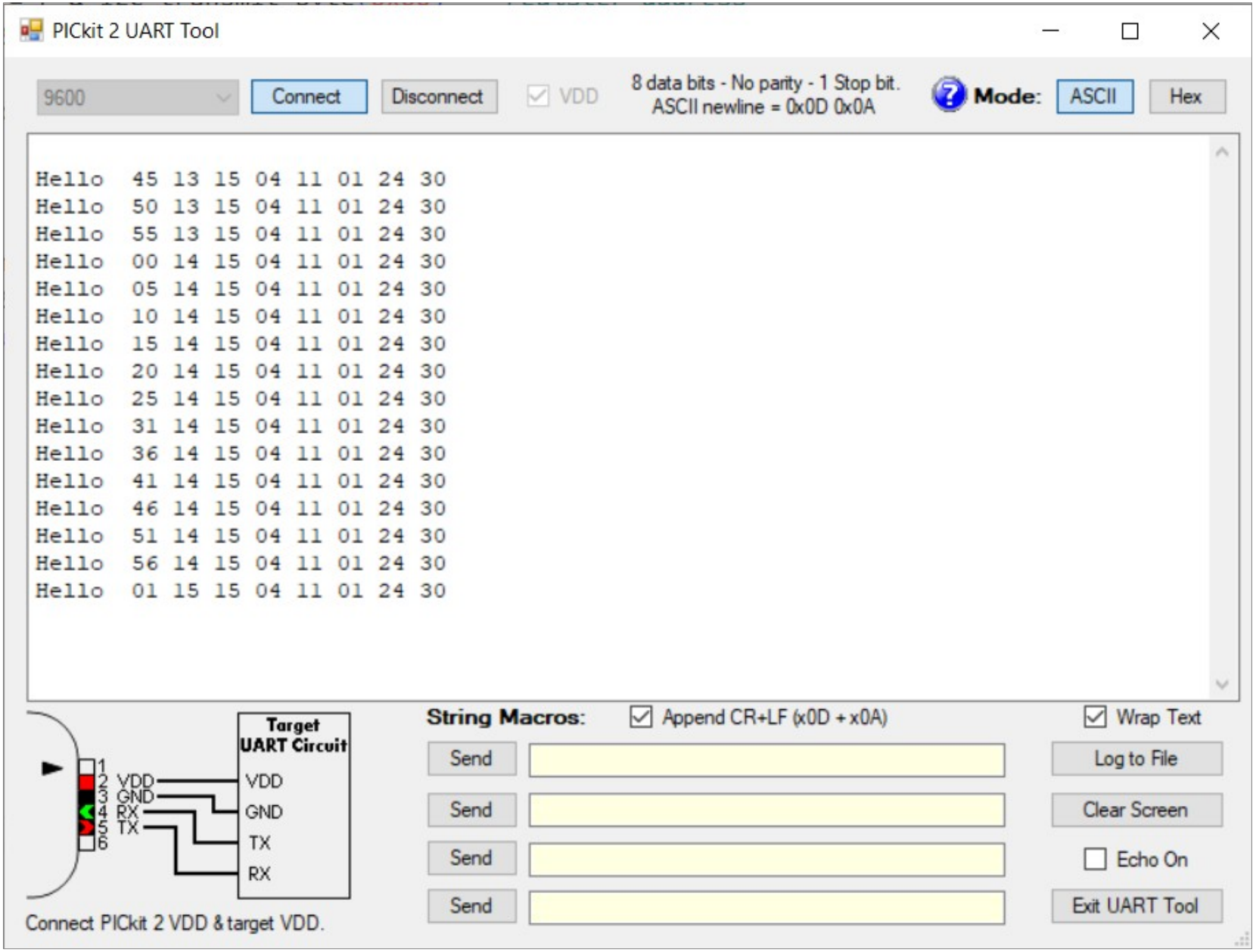
```
21  -- I2C emuláció konfigurálása ---------
22  alias i2c_scl             is pin_A4
23  alias i2c_scl_direction   is pin_A4_direction
24  alias i2c_sda             is pin_A5
25  alias i2c_sda_direction   is pin_A5_direction
26
27  const word _i2c_bus_speed = 1 ; 100kHz
28  const bit _i2c_level = true   ; i2c levels (not SMB)
29  include i2c_software
30  i2c_initialize()
31  _usec_delay(5_000_000)          -- 5 s várakozás
32
33  -- IIC address, read and write
34  const byte RTC_IIC_WR_ADDRESS = 0b1101_0000      ; 0x68«1
35  const byte RTC_IIC_RD_ADDRESS = 0b1101_0001      ; 0x68«1 + 1
36  var BYTE i
37  var BIT r
38
```

i2c_test_l0.jal – 3/3.

- A **Level 0** szint azt jelenti, hogy a tranzakciókat magunk építjük fel az **i2c_start()**, **i2c_restart()**, **i2c_stop()** eljárásokból és az **i2c_transmit_byte()**, valamint az **i2c_receive_byte()** függvényekből

```
38
39  forever loop
40    print_string(serial_sw_data, "\r\nHello  ")      -- üzenet kiírása
41    i2c_start()
42    r =     i2c_transmit_byte(RTC_IIC_WR_ADDRESS)
43    r = r & i2c_transmit_byte(0x00) -- register address
44    i2c_restart()
45    r = r & i2c_transmit_byte(RTC_IIC_RD_ADDRESS)
46    for 8 loop
47      i = i2c_receive_byte(true)
48      print_byte_hex(serial_sw_data, i);
49      serial_sw_data = " "
50    end loop
51    i2c_stop()
52    _usec_delay(5_000_000)          -- 5 s várakozás
53  end loop
```

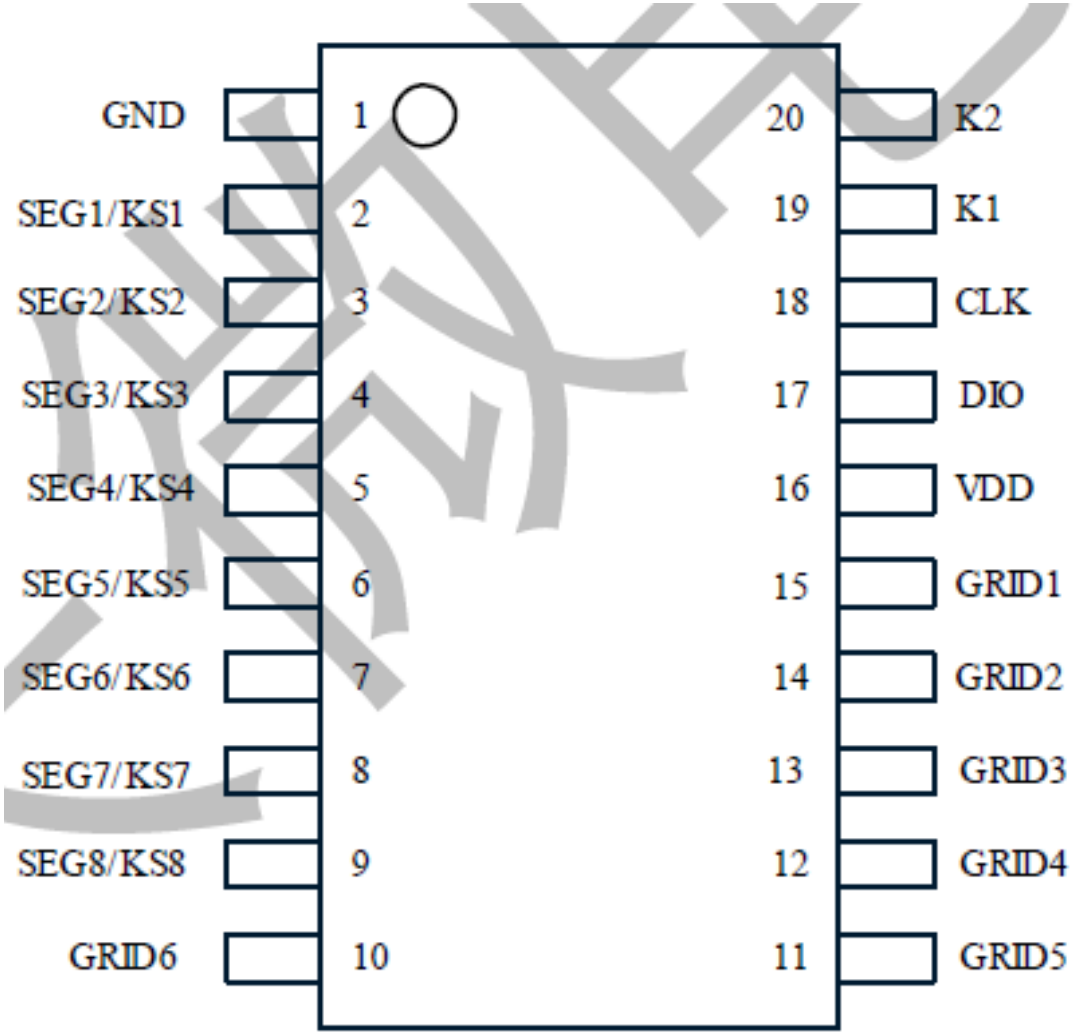
i2c_test_l0.jal futási eredmény



A TM1637 4-digites kijelző használata

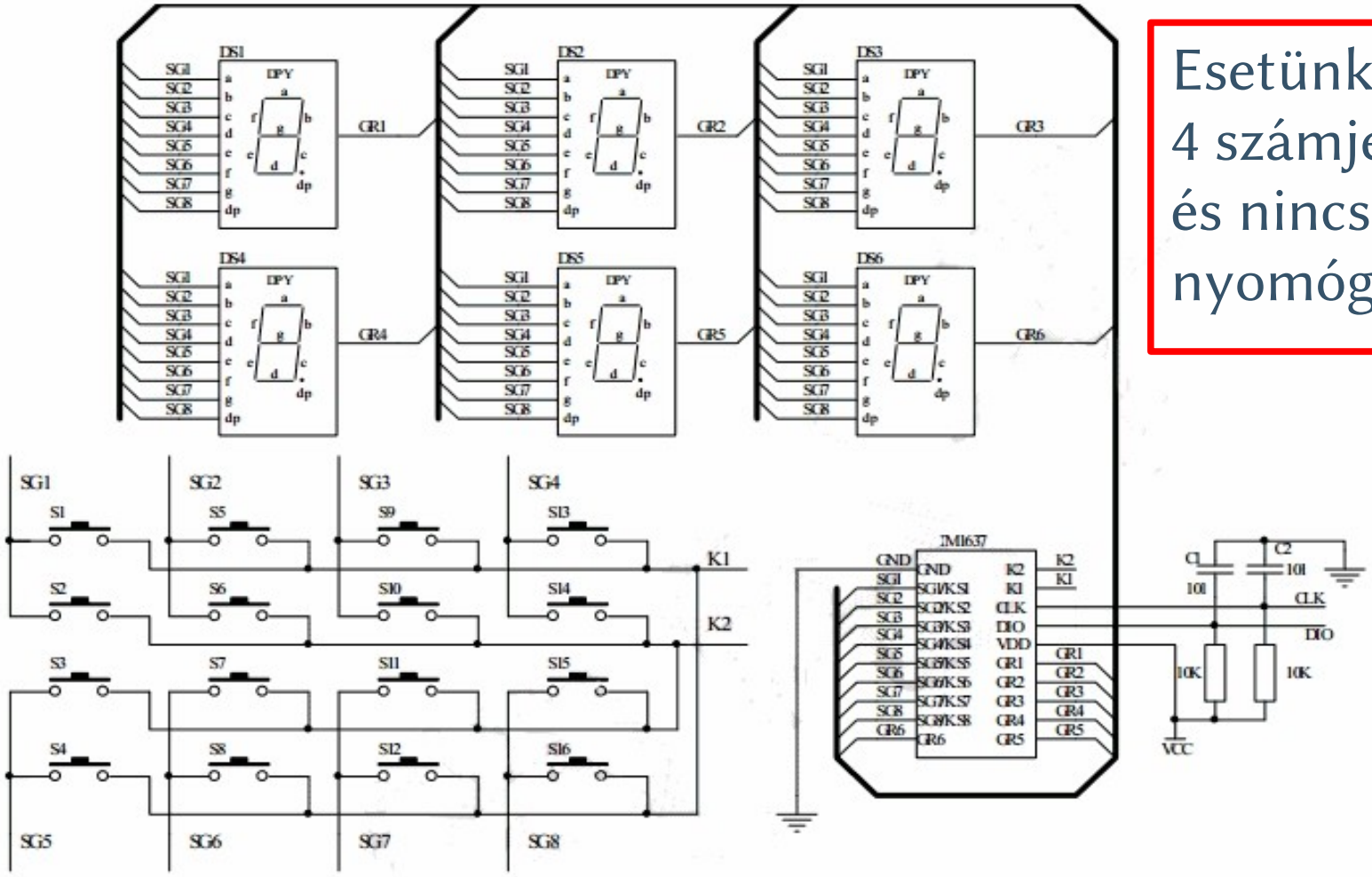
Négyszámjegyű kijelző TM1637 vezérlővel

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



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

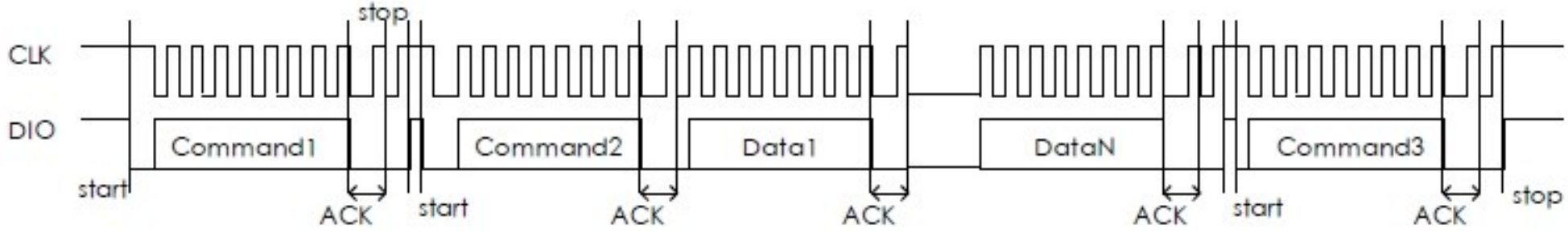


Esetünkben csak 4 számjegy van és nincs nyomógomb

TM1637 soros kommunikáció

- Két vezetékes, kétirányú, szinkron soros kommunikáció nyugtázással. Az átvitelt a mikrovezérlő irányítja (ő a „mester”).
- **START** feltétel: magas CLK mellett H⟶L átmenet a DIO vonalon
- **STOP** feltétel: magas CLK mellett L⟶H 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**1**0$0_2$
- 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

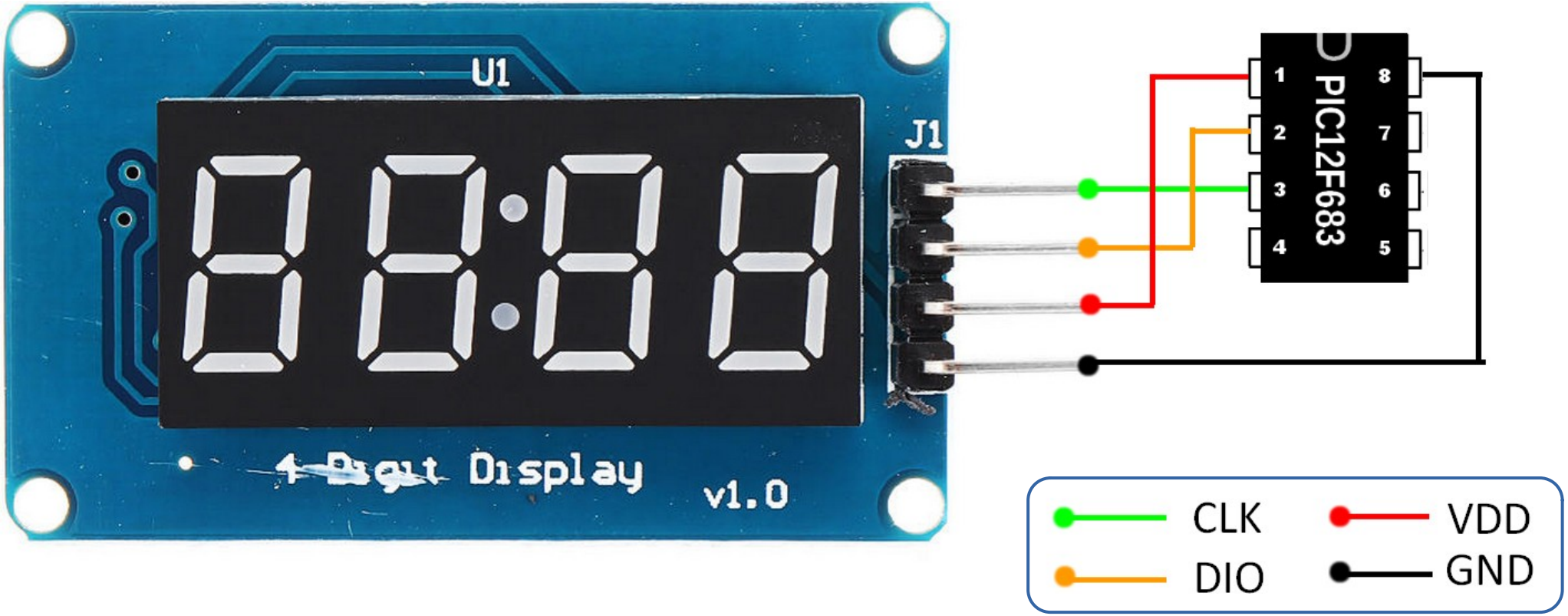
Command3

- 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 bekötés

- A **TM1637** kijelző modul **CLK** lábát a **GPIO4,** a **DIO** lábát a **GPIO5** digitális kivezetésre kötjük
- A **VCC** lábát az **5V-os** tápfeszültségre (VDD), a **GND** lábat pedig a közös pontra (GND) kötjük



A tm1637.jal programkönyvtár

- A **TM1637** kijelzővezérlőt kezelő **Jallib** programkönyvtár, melynek használatához definiálni kell a **tm1637_clk** és **tm1637_dio** kivezetés álneveket, a kijelezhető számjegyek számát és azok sorrendjét
- **Függvények:**

```
tm1637_scan_key() return byte
tm1637_digit_to_segment(byte in value) return byte
```

- **Eljárások:**

```
tm1637_display_off()
tm1637_display_nibble_hex(byte in value, byte in display_number)
tm1637_display_clear()
tm1637_init()
tm1637_display_on()
tm1637_set_brightness(byte in brightness)
tm1637_set_display(byte in segment_data, byte in display_number)
tm1637_display_byte_hex(byte in value, byte in display_number)
tm1637_display_update()
tm1637_display_word_hex(word in value, byte in display_number)
tm1637_set_dot(bit in dot_on, byte in display_number)
```

Részletek a tm1637.jal programkönyvtárból

```
-- Start a data transmission.
procedure _tm1637_start_transmission() is
   tm1637_clk_direction = input
   tm1637_dio_direction = input
   _tm1673_clock_delay()
   tm1637_dio_direction = output
end procedure
procedure _tm1637_write_byte(byte in data) is
var bit data_bit_out at data:0
   for 8 loop
      -- Clock low.
      tm1637_clk_direction = output
      if data_bit_out then
         tm1637_dio_direction = input       -- Data high
      else
         tm1637_dio_direction = output      -- Data low
      end if
      _tm1673_clock_delay()
      -- Clock high.
      tm1637_clk_direction = input
      _tm1673_clock_delay()
      data = data >> 1
   end loop
   tm1637_clk_direction = output
   _tm1673_clock_delay()
end procedure
```

```
-- Stop a data transmission.
procedure _tm1637_stop_transmission() is
   tm1637_clk_direction = input
   _tm1673_clock_delay()
   tm1637_dio_direction = input
   _tm1673_clock_delay()
end procedure

-- Acknowledge a data transmission.
procedure _tm1637_acknowlege_transmission() is
   -- Check acknowledge.
   tm1637_clk_direction = output
   _tm1673_clock_delay()
   tm1637_dio_direction = input
   _tm1673_clock_delay()
   tm1637_clk_direction = input
   _tm1673_clock_delay()
   -- Acknowledge.
   if !tm1637_dio then
      tm1637_dio_direction = output
      _tm1673_clock_delay()
   end if
   tm1637_clk_direction = output
   _tm1673_clock_delay()
end procedure
```

tm1637_test.jal – 5/1.

```
include 12f683                                  -- PIC target MCU
pragma target CLOCK     8_000_000               -- oscillator frequency
pragma target OSC       INTOSC_NOCLKOUT         -- internal oscillator at 8MHz
pragma target WDT       disabled
OSCCON_IRCF   = 0b_111                          -- Fosc = 8 MHz setting
enable_digital_io()
-- TM1637 pin definition.
alias tm1637_clk is pin_A4 -- Pin 3 for 8 pin DIP
alias tm1637_clk_direction is pin_A4_direction
alias tm1637_dio is pin_A5 -- Pin 2 for 8 pin DIP.
alias tm1637_dio_direction is pin_A5_direction
-- You can change the display width (default is 6 when not defined).
const byte TM1637_WIDTH = 4
-- Set the display order, rightmost is 0, leftmost is TM1637_WIDTH - 1.
const byte TM1637_DISPLAY_ORDER[TM1637_WIDTH] = {3,2,1,0}

include tm1637
tm1637_init()
--- Display 'FOUr' on the display.
const byte DISPLAY_HELLO[TM1637_WIDTH] =
            {
             0b0011_0011, -- R
             0b0011_1110, -- U
             0b0011_1111, -- O
             0b0111_0001  -- F
            }
```

```
-- -----------------------------------------------------------------------
-- Title: Test program for the TM1637 with four 7-segment displays.
-- Author: Rob Jansen, Copyright (c) 2020..2022 all rights reserved.
-- Adapted-by: István Cserny (PIC12F683 adaptation)
-- Compiler: 2.5r6
```

tm1637_test.jal – 5/2.

```
-- Wait for some time.
procedure wait_long() is
   _usec_delay(5_000_000)
end procedure

procedure wait_short() is
   _usec_delay(500_000)
end procedure

var sdword big_pos_counter
var sdword big_neg_counter
var word word_counter
var byte counter

forever loop
   -- Write a 'Four' message by writing the segment data directly into the
   -- global display data buffer.
   for TM1637_WIDTH using counter loop
      tm1637_display_data[counter] = DISPLAY_HELLO[counter]
   end loop
```

5 másodperc várakozás

Fél másodperc várakozás

tm1637_test.jal – 5/3.

Mivel az előző oldal alján csak a memóriát módosítottuk, frissítenünk kell a kijelző adatregisztereit (át kell másoltatni az adatokat)

```
-- Because we are using the global display data buffer we need to update
-- the display with this segment data.
tm1637_display_update()
wait_long()
-- Switch display off for some time
tm1637_display_off()
wait_short()
tm1637_display_on()

-- Change the brightness.
for (TM1637_MAX_BRIGHTNESS + 1) using counter loop
   tm1637_set_brightness(counter)
   wait_short()
end loop

-- Now clear the display, that is all segments are off.
tm1637_display_clear()
wait_short()
```

A kijelző ki- és bekapcsolása

A fényerőt 8 fokozatban szabályozhatjuk (0 - 7)

Töröljük a kijelzőt (minden szegmenst lekapcsolunk)

tm1637_test.jal – 5/4.

```
-- Set the dots on and off.
for TM1637_WIDTH using counter loop
   tm1637_set_dot(TRUE, counter)
   wait_short()
   tm1637_set_dot(FALSE, counter)
 end loop

-- Set an individual display and show its number.
tm1637_display_clear()
for TM1637_WIDTH using counter loop
   tm1637_set_display(tm1637_digit_to_segment(counter), counter)
   wait_short()
   tm1637_set_display(TM1637_SEGMENT_OFF,counter)
end loop

-- Write a hexdecimal nibble on the rightmost display.
tm1637_display_clear()
for 16 using counter loop
   tm1637_display_nibble_hex(counter,0)
   _usec_delay(200_000)
end loop
wait_short()
```

A pontok be- és kikapcsolása

A számjegyek sorszámának kiíratása (0, 1, 2, 3)

Számlálás és hexadecimális kiíratás 0-tól 15-ig

tm1637_test.jal – 5/5.

```
    -- Write a hexdecimal word on the rightmost display.
    tm1637_display_clear()
    for 512 using word_counter loop
       tm1637_display_word_hex(word_counter,0)
    end loop
    wait_short()

    -- Show a counter in another way with leading zero's.
    tm1637_display_clear()
    for 2023 using big_pos_counter loop -- Up and until the year 2022 :-)
       tm1637_display_dword_dec(dword(big_pos_counter),0,TRUE)
    end loop
    wait_short()

    -- Show a negative counter in another way without leading zero's.
    tm1637_display_clear()
    for 1000 using big_pos_counter loop
       big_neg_counter = -big_pos_counter
       tm1637_display_sdword_dec(big_neg_counter,0,FALSE)
    end loop
    wait_short()
end loop
```

Számlálás és hexadecimális kiíratás 0-tól 1FF-ig

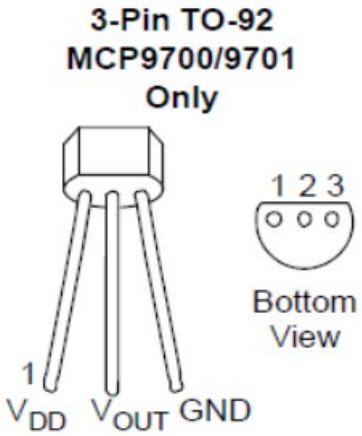
Számlálás és decimális kiíratás 0-tól 2022-ig

Számlálás és negatív szám decimális kiíratás 0-tól -999-ig

Hőmérő projekt

- A hőmérsékletet egy **MCP9700** analóg hőmérővel mérjük

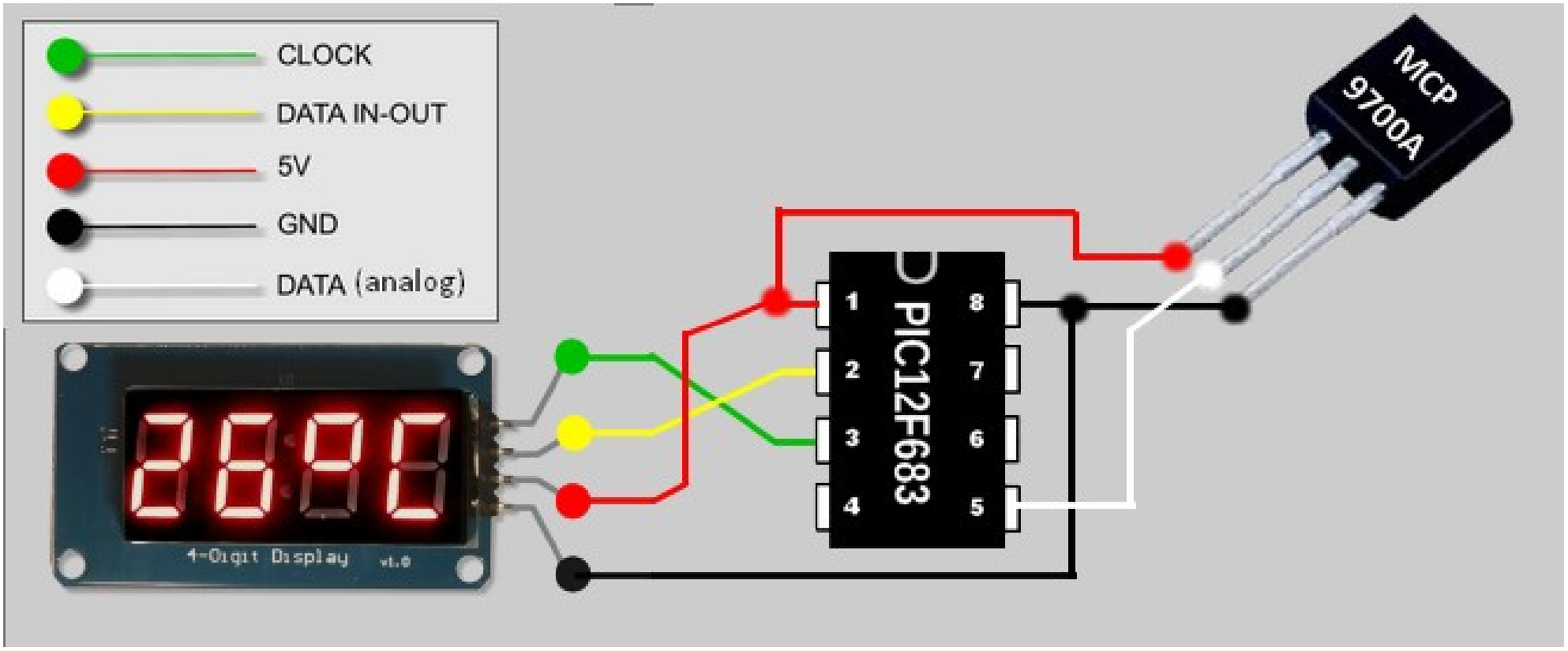
Microchip MCP9700
VDD = 2,5 – 5,5 V
Mérési tart.: -40 – 150 °C
Érzékenység: 10 mV / °C
Nullapont: 500 mV @ 0 °C



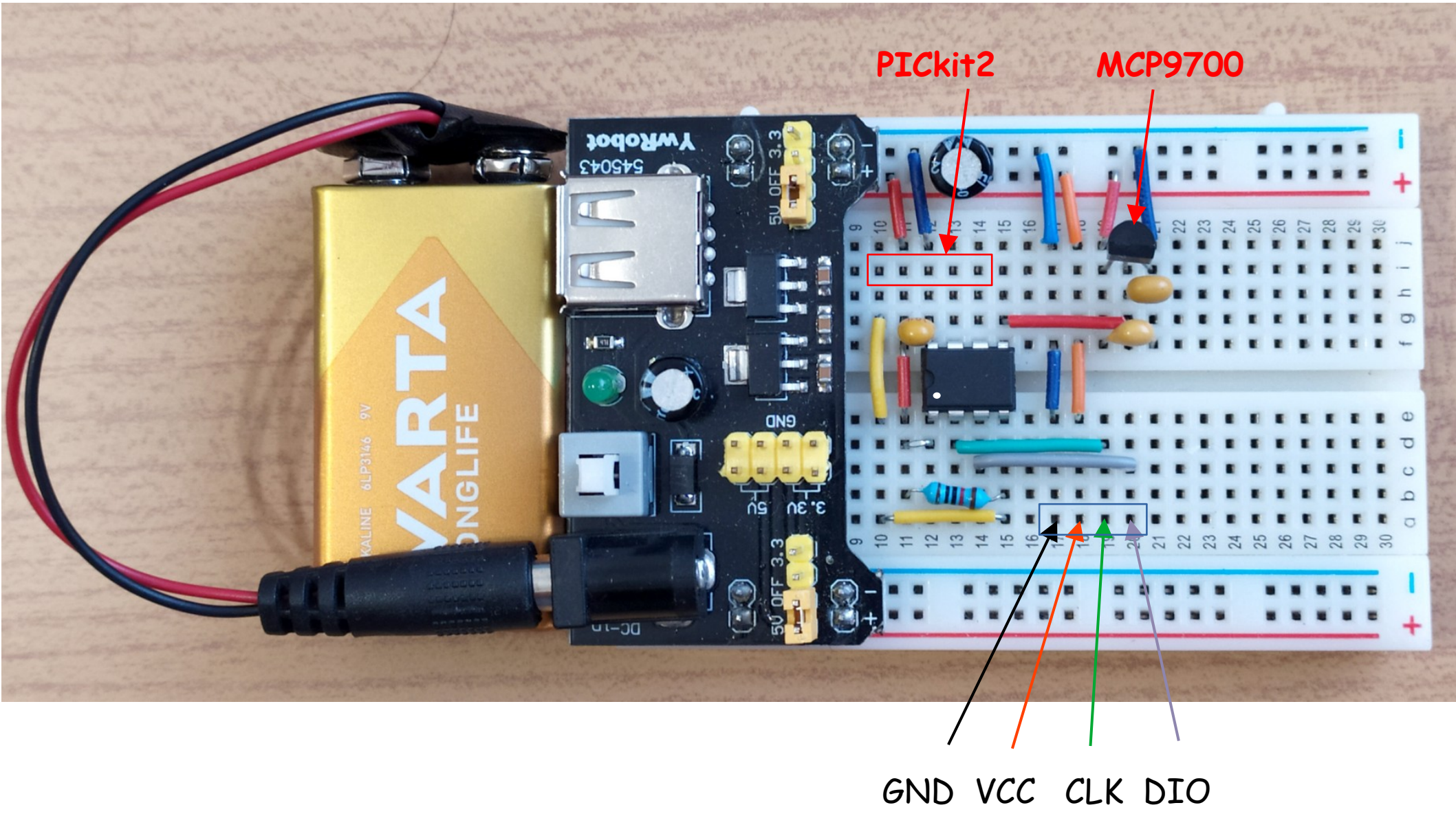
- A **TM1637** kijelzőt a **PIC12F683** két kivezetésével vezéreljük (**GPIO5** és **GPIO4**)
- Az ingadozások és zavarjelek kiszűrésére a kijelzett érték sok mérés átlaga
- A tápellátáshoz egy 9 V-os elemet és egy 5 V-os feszültségstabilizátort használunk

Kapcsolási vázlat

- Az analóg hőmérő jelét az **AN2** analóg bemenetre (**GPIO2**) kötjük
- A kijelzőt vezérlő órajelet a **GPIO4**, az adatvonalat a **GPIO5** kimenetre kötjük
- A közös tápfeszültség névleg 5 V (esetemben ténylegesen 4,9 V)



A megépített kapcsolás



averaging_thermometer.jal – 3/1.

```
1   include 12f683                                  -- PIC céláramkör
2   pragma target CLOCK     8_000_000               -- oszcillátor frekvencia
3   pragma target OSC       INTOSC_NOCLKOUT         -- belső oszcillátor 8MHz-en
4   pragma target WDT       disabled
5   OSCCON_IRCF   = 0b_111                          -- Fosc = 8 MHz beállítása
6   include delay                                   -- Késleltető eljárások
7   enable_digital_io()
8
9   -- TM1637 bekötés definiálása ---------
10  alias tm1637_clk is pin_A4                      -- GPIO4 az órajel
11  alias tm1637_clk_direction is pin_A4_direction
12  alias tm1637_dio is pin_A5                      -- GPIO5 az adatvonal
13  alias tm1637_dio_direction is pin_A5_direction
14
15  -- TM1637 konfigurálása ---------------
16  const byte TM1637_WIDTH = 4                     -- Számjegyek száma
17  const byte TM1637_DISPLAY_ORDER[4] = {3,2,1,0} -- Számjegyek sorrendje
18  include tm1637
19  tm1637_init()
20  tm1637_set_brightness(TM1637_MAX_BRIGHTNESS)
```

averaging_thermometer.jal – 3/2.

- Az ADC-t 10 bites üzemmódba programozzuk, s jobbra igazítva olvassuk ki az eredményt

```
21
22  -- ADC és AN2 konfigurálása ----------
23  const byte ADC_CHANNEL = 2                  -- A potméter pin_AN2-hoz kötve
24  ANSEL_ANS2 = TRUE                           -- AN2 analóg móba állítva
25  ANSEL_ADCS = 0b_101                         -- Fosc/16 legyen az ADC órajel   (TAD=2us)
26  pin_AN2_direction = input                   -- AN2 bemenetre állítva
27  ADCON0_VCFG = FALSE                         -- VDD és VSS a referencia
28
29  const ADC_RSOURCE = 1_000                   -- Kis kimenőellenállásra számítunk
30  const ADC_HIGH_RESOLUTION = TRUE            -- 10 bites felbontású ADC mód
31  include adc                                 -- ADC könyvtár becsatolása
32  adc_init()                                  -- ADC inicializálás
33  ADCON0_ADFM = true                          -- right justification
34  ADCON0_CHS = ADC_CHANNEL                    -- Select channel
35  ADCON0_ADON = true
36  _usec_delay(10)
37
38  var DWORD n_adc, mv
39  var SDWORD tempC
40
```

averaging_thermometer.jal – 3/3.

```
41  forever loop
42    n_adc = 0                                   -- összegzéshez nullázzuk
43    for 4900 loop                               -- ha 4900 mV a tápfesz...
44      ADCON0_GO = true                            -- start conversion
45      while ADCON0_GO == TRUE loop                -- wait until conversion completed
46        -- Empty loop.
47      end loop
48      n_adc = n_adc + adc_read_high_res(ADC_CHANNEL)
49    end loop
50    mv = n_adc/1024
51    tempC = (mv-500+5)/10                       -- MCP9700A: mv = (tempC*10 + 500)
52    if tempC<0 then
53      tm1637_display_sdword_dec(tempC,1,FALSE)
54      tm1637_set_display(0b0011_1001,0)         -- write Celsius sign
55    else
56      tm1637_display_dword_dec(tempC,2,FALSE)
57      tm1637_set_display(0b0110_0011,1)         -- write degree sign
58      tm1637_set_display(0b0011_1001,0)         -- write Celsius sign
59    end if
60    delay_1ms(5000)
61  end loop
62
```