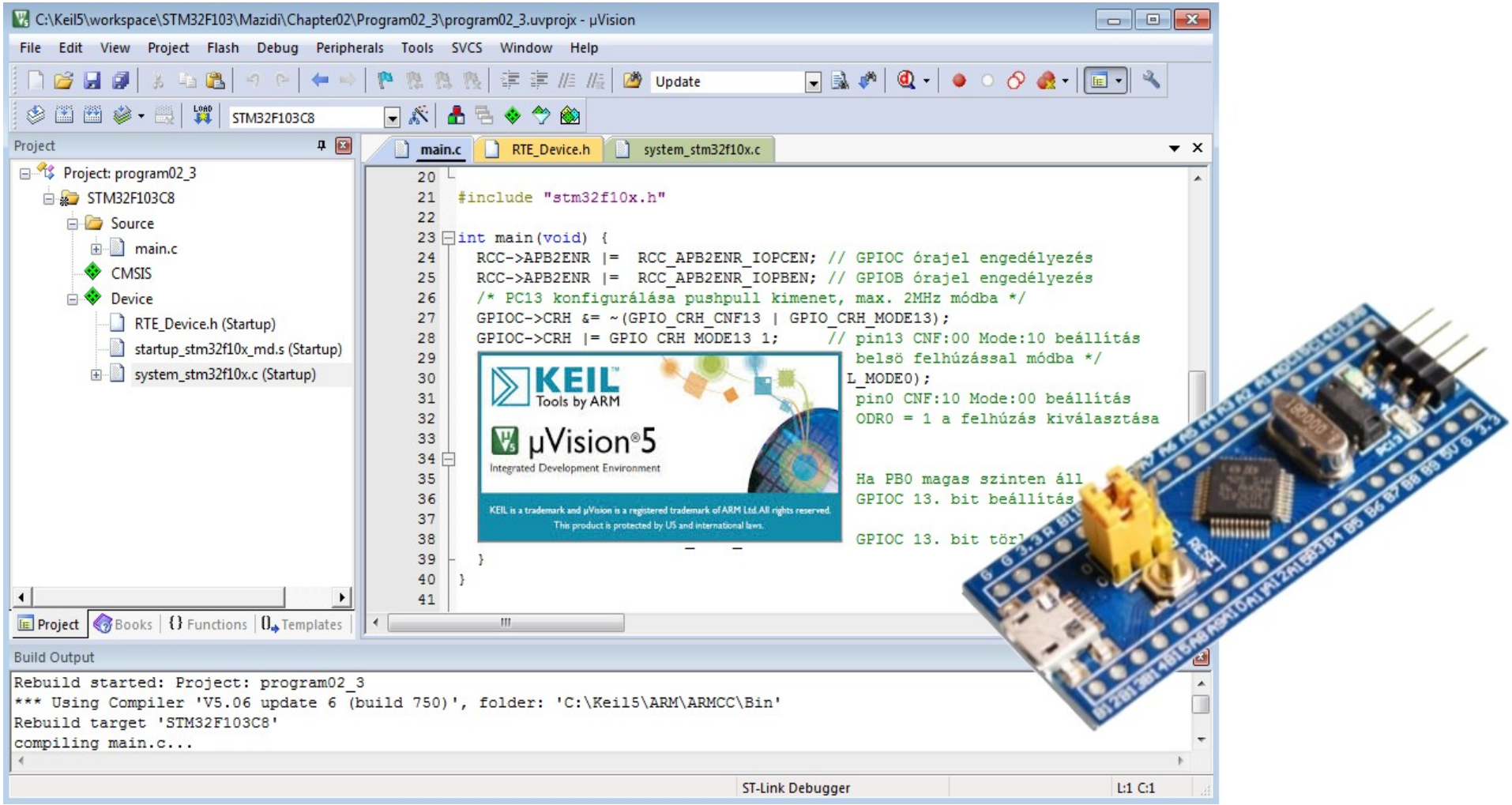
STM32 mikrovezérlők programozása ARM Keil környezetben



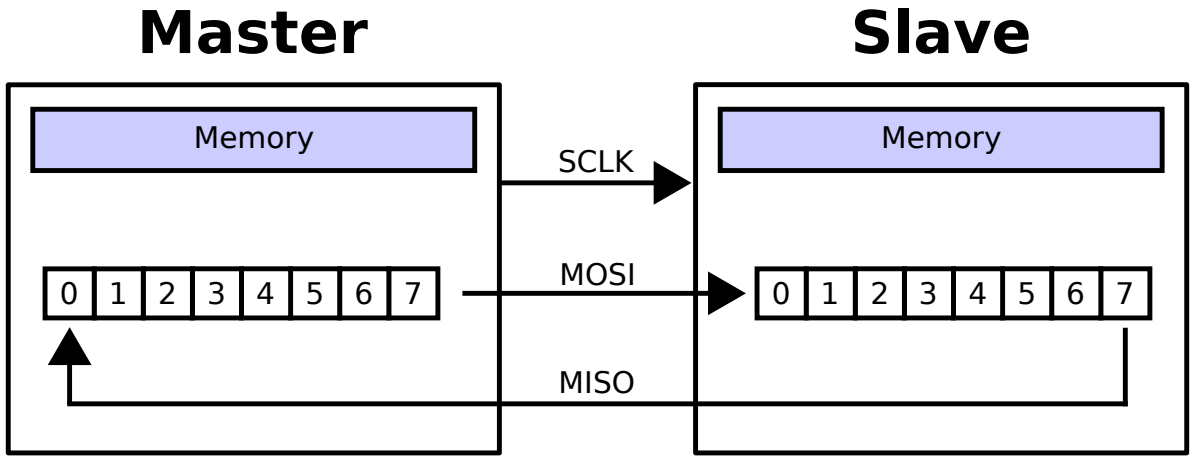
12. Soros Periféria Illesztő (SPI)

Felhasznált és ajánlott irodalom

- Joseph Yiu: Cortex-M for Beginners
- Joseph Yiu: The Definitive Guide To The ARM CORTEX-M3
- Muhammad Ali Mazidi, Shujen Chen, Eshragh Ghaemi: STM32 Arm Programming for Embedded Systems
- Alexander Tarasov: **Курс «Штурмуем STM32»**
- Warren Gay: Beginning STM32 - Developing with FreeRTOS, libopencm3 and GCC
- ARM Keil MDK Getting started
- **STM32F103C8** adatlap és termékinfo
- **STM32F103** Family Reference Manual

Az SPI kommunikációs csatorna

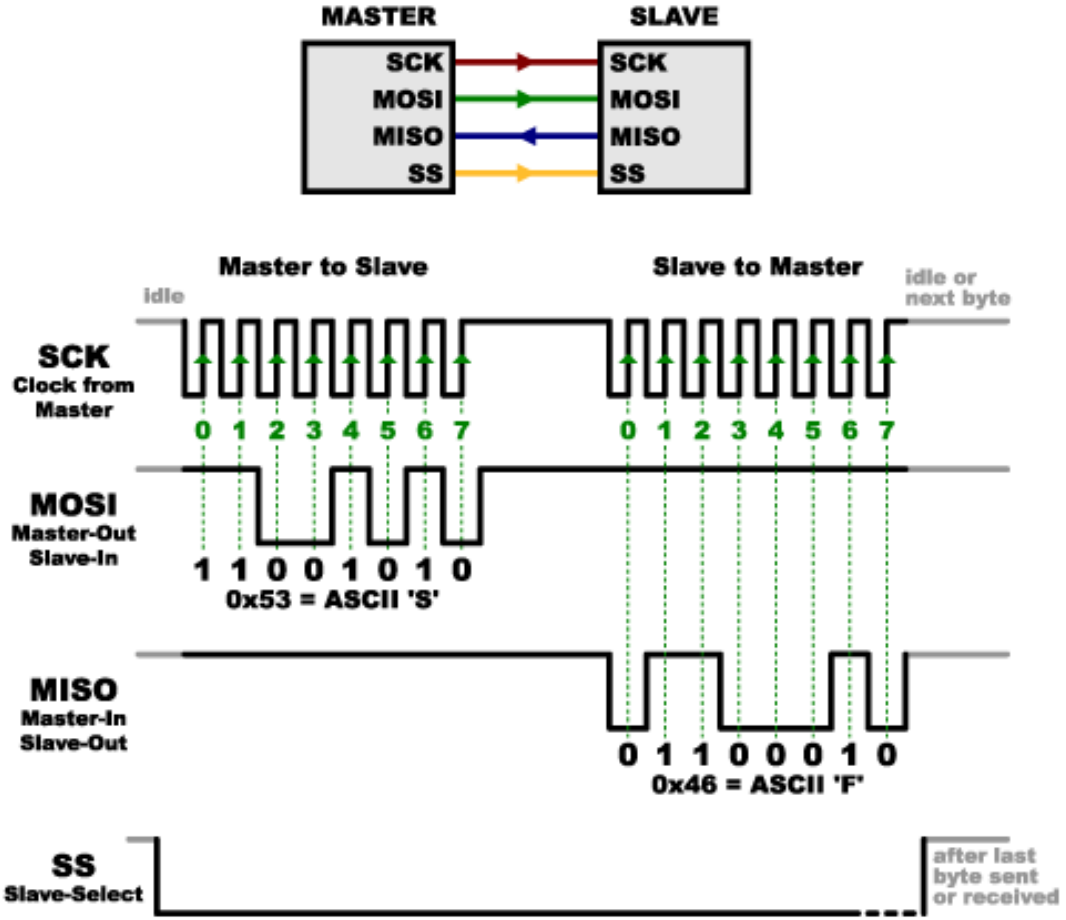
- A soros periféria illesztő (Serial Peripheral Interface, SPI) kétirányú szinkron soros kommunikációt valósít meg két eszköz között
- A kommunikációban résztvevő eszközök között master/slave (mester/szolga) viszony áll fenn, a tranzakciót a master kezdeményezi és az órajelet is a master állítja elő
- Az SPI kommunikáció során lényegében két léptetőregiszter a vezérlőjelek hatására adatot cserél (**MOSI:** master kimenet, slave bemenet; **MISO:** master bemenet, slave kimenet, **SCK:** soros órajel)



Forrás: en.wikipedia.org/wiki/Serial_Peripheral_Interface

Jelalak az SPI buszon

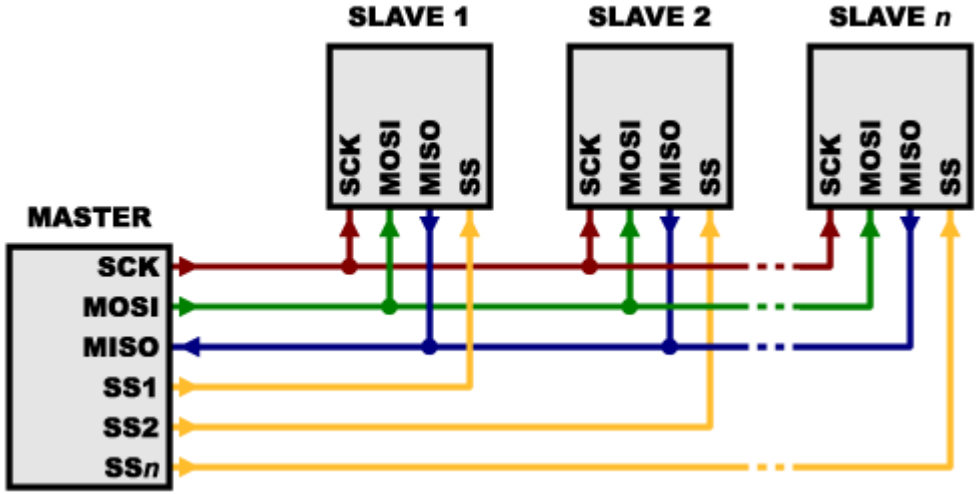
- Szokás egy negyedik jelet is használni: ***slave select*** (SS), amellyel a master megszólíthatja vagy kiválaszthatja a slave eszközt
- Az $\overline{\text{SS}}$ jel a tranzakció folyamán végig aktív állapotban legyen!



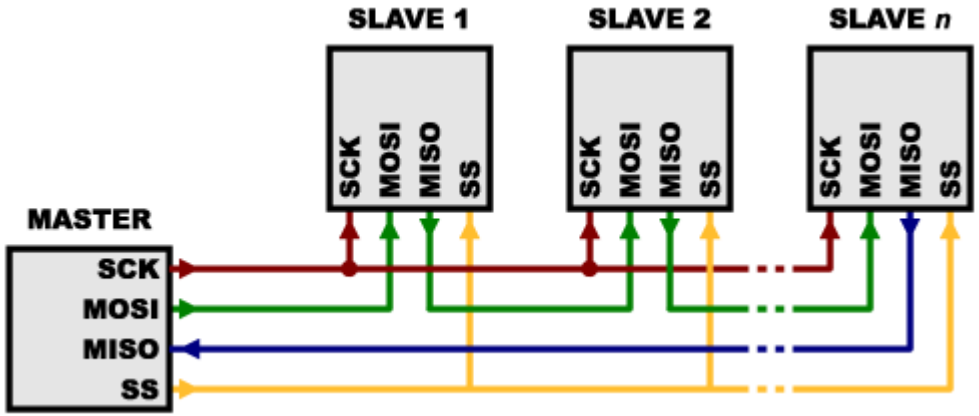
Forrás: learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

Több slave eszköz kezelése

- 1. Minden slave eszöz külön kiválasztójelet kap



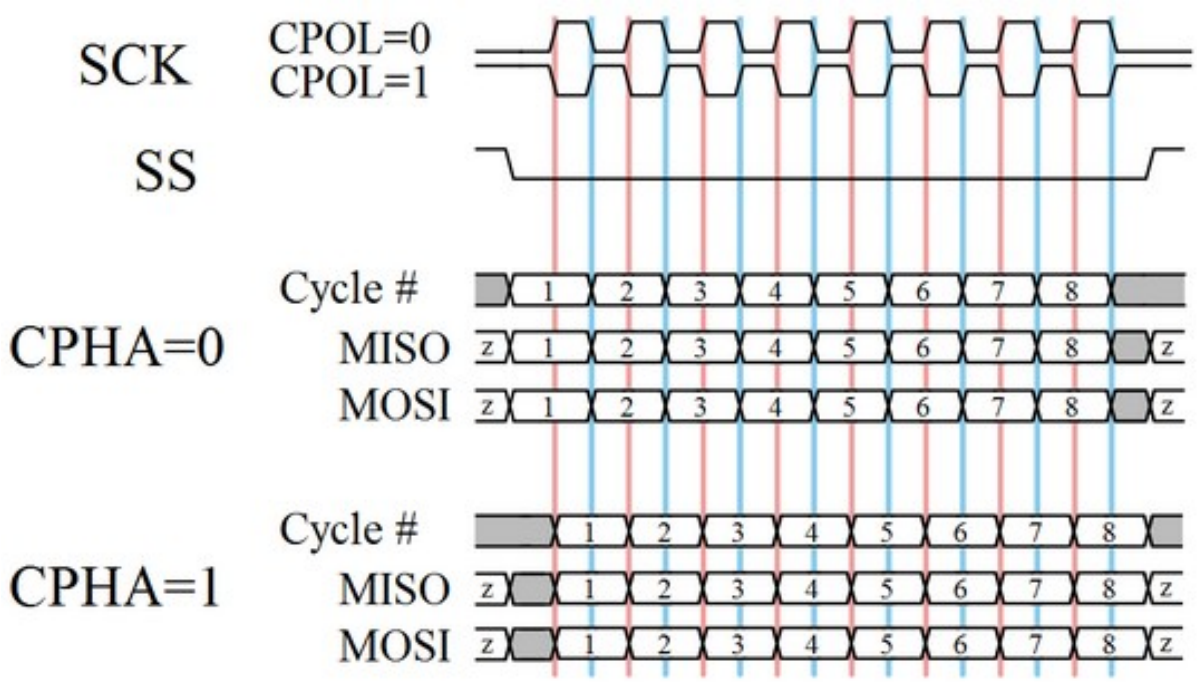
- 2. A slave eszközöket felfűzzük (daisy chain) és egyszerre írjuk



Forrás: learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

Az SPI órajel polaritása és fázisa

- Az SPI adatátvitel szinkronizálását a master által keltett órajel (SCK) biztosítja. A slave eszközhöz illeszkedően biztosítani kell az órajel megfelelő frekvenciáját, polaritását és fázisát
- Az adatlapok tartalmazzák azt az információt, hogy az adott SPI eszköz milyen üzemmódban vagy üzemmódokban képes működni



SPI mód	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Legáltalánosabban a 0. módot használják

Forrás: en.wikipedia.org/wiki/Serial_Peripheral_Interface

Az STM32F103C8 MCU SPI csatornái

- **STM32F103C8** mikrovezérlő két csattornája: **SPI1** és **SPI2**
- Master/slave mód, multimaster mód támogatása
- Szimplex (akár kétirányú is) vagy duplex mód
- 8/16 bites mód, választható bitsorrend: először LSB vagy MSB
- Nyolc fokozatban állítható bitráta (max. $f_{PCLK}/2$)
- NSS menedzselés szoftveresen vagy hardveresen
- Programozható órajel polaritás és fázis
- Hardveres CRC generálás és ellenőrzés
- I2S mód támogatása
- Egybájtos adó- és vevőoldali buffer, DMA támogatás
- Státuszbitek, hiba- és foglaltság detektálás, programmegszakítás

Az SPI*n* csatorna engedélyezése

- **RCC_APB1ENR** – 1. periféria busz engedélyező regiszter

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	BKP EN	CAN2 EN	CAN1 EN	Reserved		I2C2 EN	I2C1 EN	UART5EN	UART4EN	USART3 EN	USART2 EN	Res.
		rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWD GEN	Reserved					TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw						rw	rw	rw	rw	rw	rw

- **RCC_APB2ENR** – 1. periféria busz engedélyező regiszter

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART 1EN	Res.	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved		IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

- Az **STM32F103C8** mikrovezérlő két SPI csatornával rendelkezik, a megjelölt bitek 1-be állítása engedélyezi a hozzá rendelt modult
- SPI3 csak a nagyobb lábszámú variánsokban van implementálva

Az SPI csatornák kivezetései

- **SPI1** kivezetése alternatívan választható, **SPI2** kivezetése kötött
- **AFIO_MAPR** regiszter – alternatív funkció átirányítás (remap)
  0. bit – SPI1_REMAP 0: no remap, **1:** remap

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					SWJ_CFG[2:0]			Reserved			ADC2_ETRGREG_REMAP	ADC2_ETRGINJ_REMAP	ADC1_ETRGREG_REMAP	ADC1_ETRGINJ_REMAP	TIM5CH4_IREMAP
					w	w	w				rw	rw	rw	rw	rw
Reserved					SWJ_CFG[2:0]			Reserved							TIM5CH4_IREMAP
					w	w	w								rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PD01_REMAP	CAN_REMAP[1:0]		TIM4_REMAP	TIM3_REMAP[1:0]		TIM2_REMAP[1:0]		TIM1_REMAP[1:0]		USART3_REMAP[1:0]		USART2_REMAP	USART1_REMAP	I2C1_REMAP	SPI1_REMAP
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Kivezetés	**SPI1** (no remap)	**SPI1** (remap)	**SPI2**
NSS	PA4	PA15	PB12
SCK	PA5	PB3	PB13
MISO	PA6	PB4	PB14
MOSI	PA7	PB5	PB15

Az SPI modulok regiszterei

- **SPI*n*_CR1** – Vezérlő regiszter 1.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDI OE	CRC EN	CRC NEXT	DFF	RX ONLY	SSM	SSI	*LSB FIRST*	SPE	BR [2:0]			MSTR	CPOL	CPHA
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

DFF – adatkeret formátum (0: 8-bit, 1: 16-bit)
LSBFIRST – bit sorrend (0: MSB first, 1: LSB first)
SPE – az SPI működésének engedélyezése (0: tiltás, 1: engedélyezés)
BR – baud rate ($\mathbf{f_{PCLK}}$ / $\mathbf{2^{(BR+1)}}$ azaz f_{PCLK} /2, /4, /8, … /128, /256)
MSTR – master / slave mód választás (0: slave, 1: master mód beállítás)
CPOL, CPHA – SPI mód beállítás (órajel polaritás, órajel fázis)

- **SPI*n*_CR2** – Vezérlő regiszter 2.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXEIE	RXNEIE	ERRIE	Res.	Res.	*SSOE*	TXDMAEN	RXDMAEN
								rw	rw	rw			rw	rw	rw

Többnyire megszakítást, illetve **DMA** átvitelt engedélyező bitek,
SSOE – NSS kivezetés vezérlésének engedélyezése (csak master)

Az SPI modulok regiszterei

- **SPI*n*_SR** – Állapotjelző regiszter

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								BSY	OVR	MODF	CRC ERR	UDR	CHSIDE	TXE	RXNE
								r	r	r	rc_w0	r	r	r	r

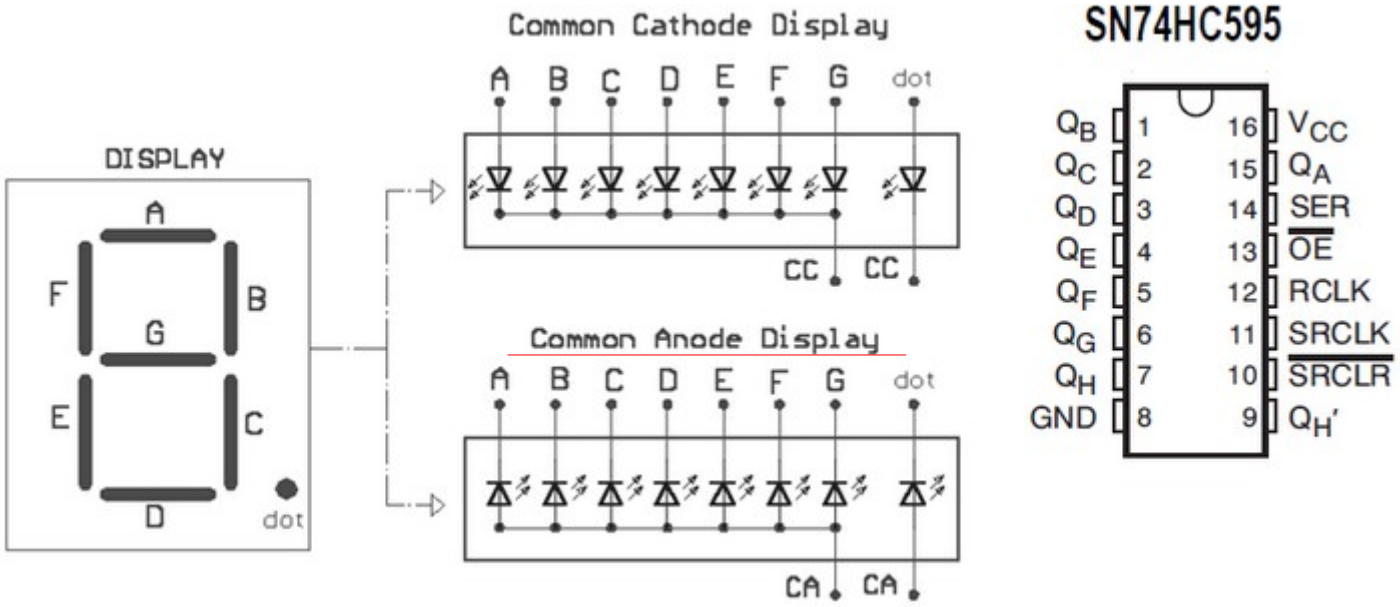
BSY – foglaltság jelzése
TXE – a küldő oldali adatregiszter szabad
RXNE – a vevőoldali adatregiszter nem üres

- **SPI*n*_DR** – Adat regiszter
Az **SPI*n*_CR1** regiszter **DFF** bitjének beállításától függően az adatküldés 8- vagy 16-bites. Nyolcbites módban csak az adat-regiszter alsó 8 bitje számít
SPI*n*_DR valójában két regiszter: íráskor a küldő buffert írjuk, olvasáskor a vevő buffert olvassuk

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

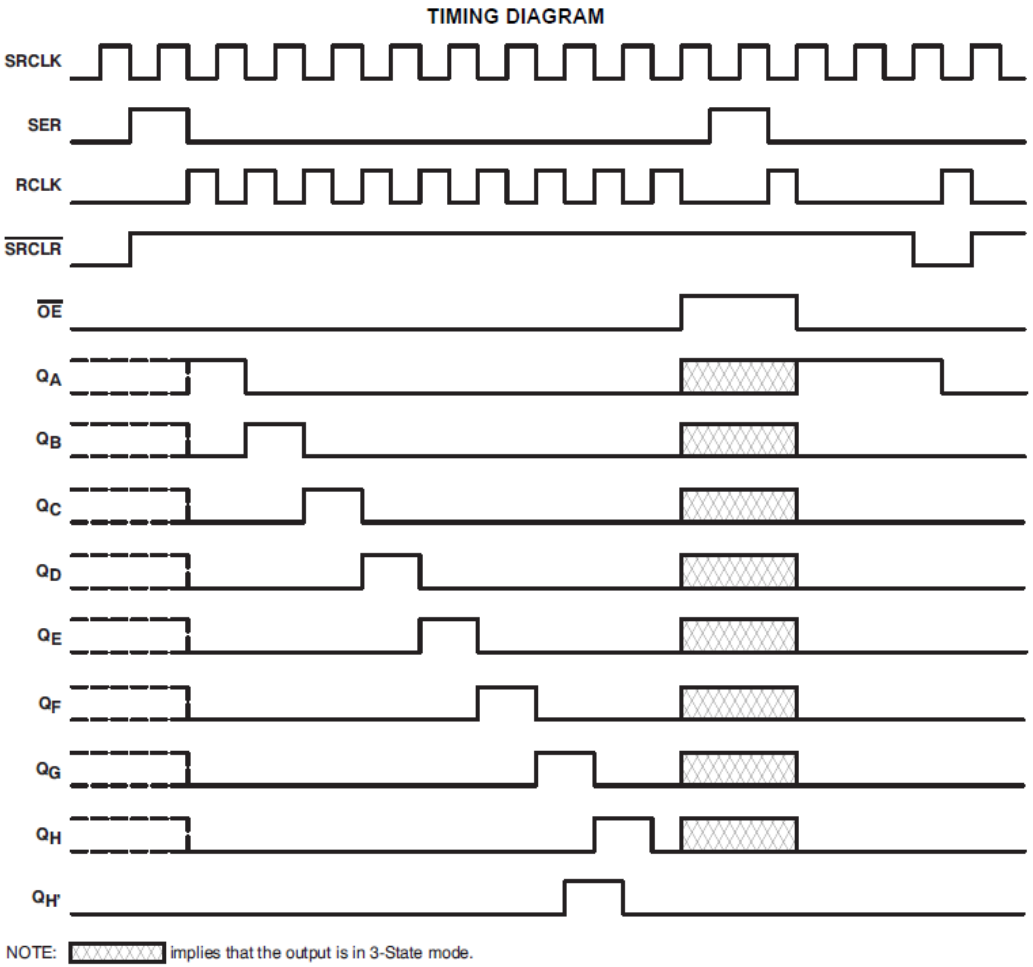
Adatküldés az SPI1 csatornán

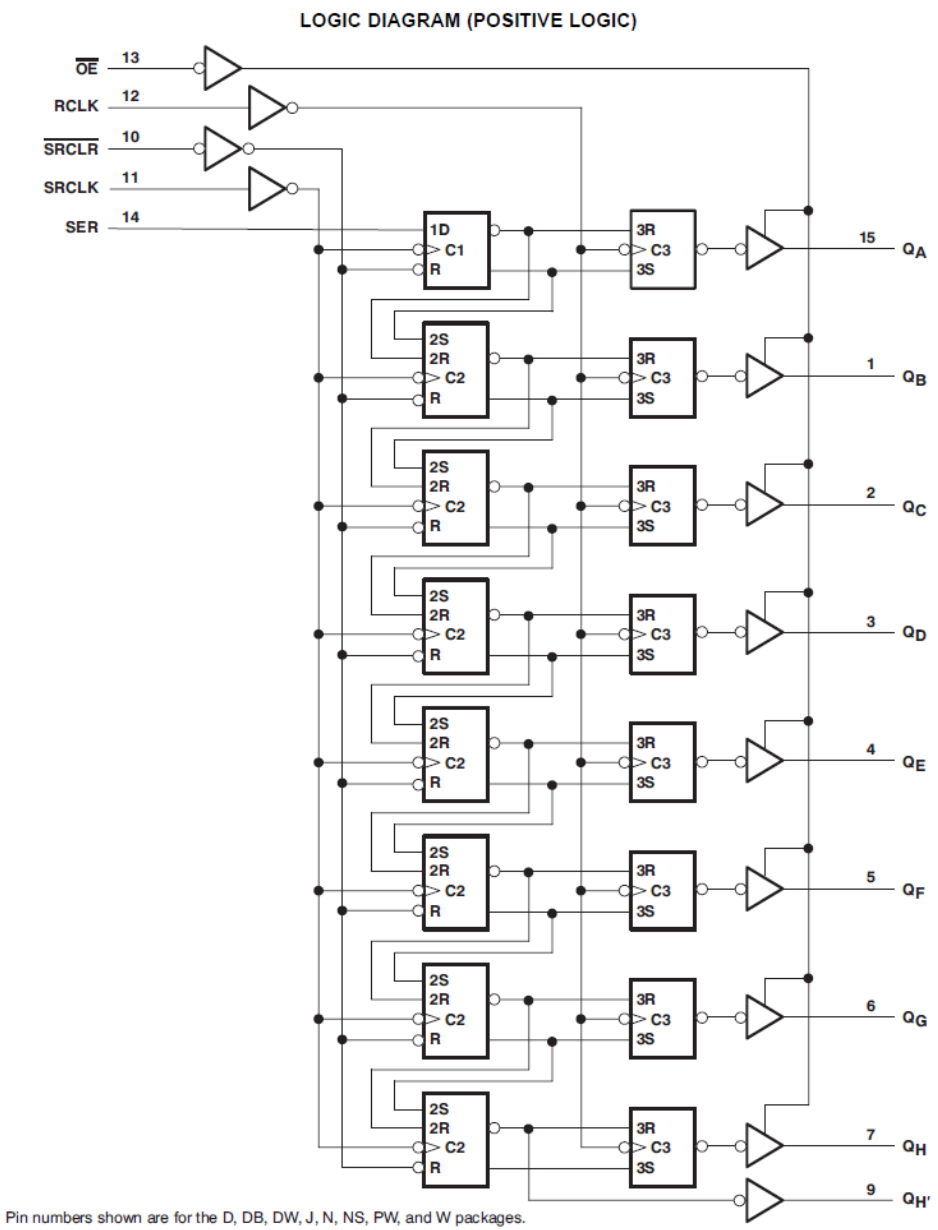
- Az első mintaprogram bemutatja az **SPI1** csatorna konfigurálását és az adatküldést
- Hogy láthassuk is az eredményt, két **74HC595** léptetőregiszterből építettünk egy SPI perifériát, amely két **HD1131R** típusú (közös anódú) 7 szegmenses LED számkijelzőt vezérel



A 74HC595 működése

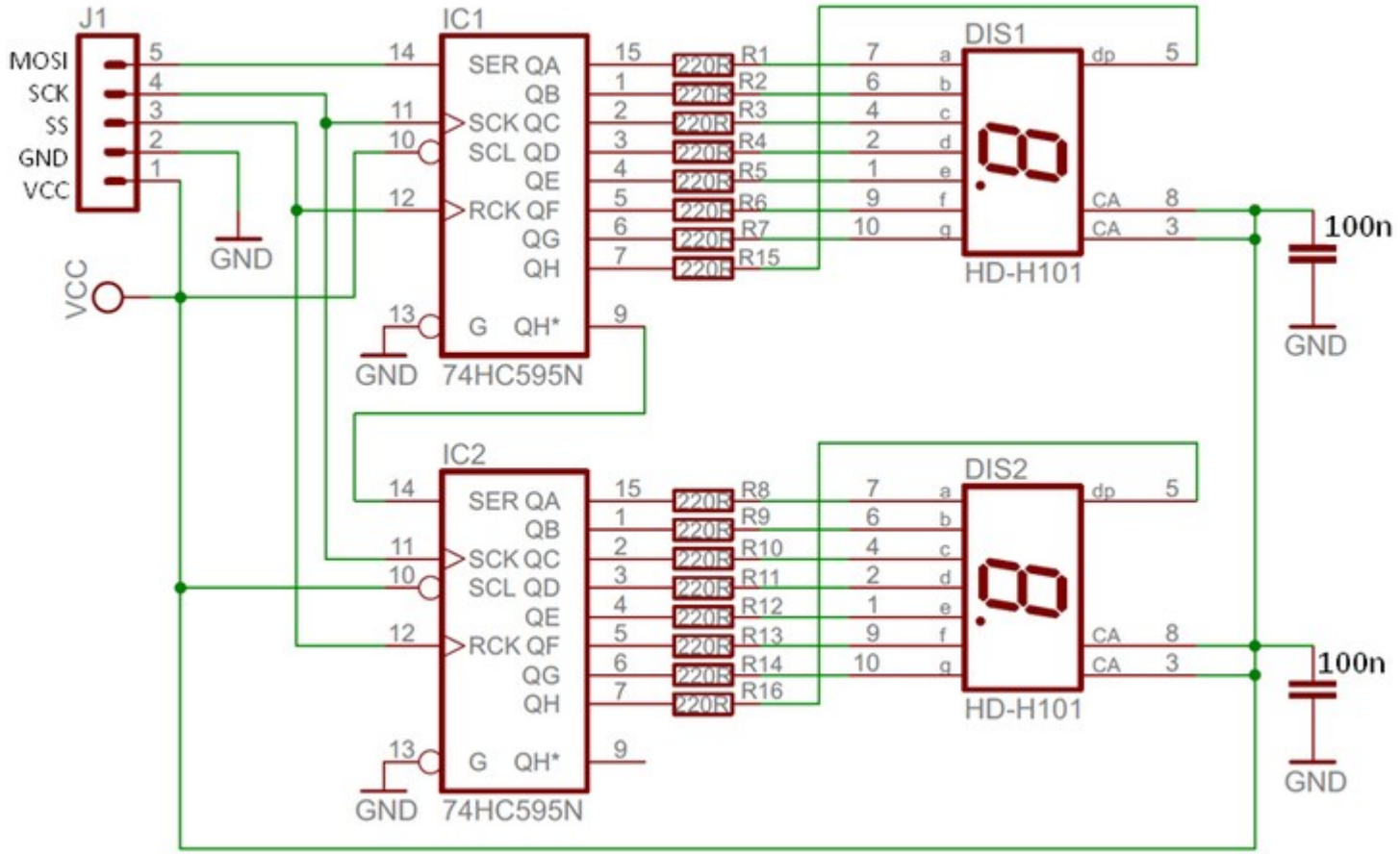
- A 74HC959 shift regiszter felfűzhet
  a kimeneti regiszter csak külön
  parancsra írja át az adatot





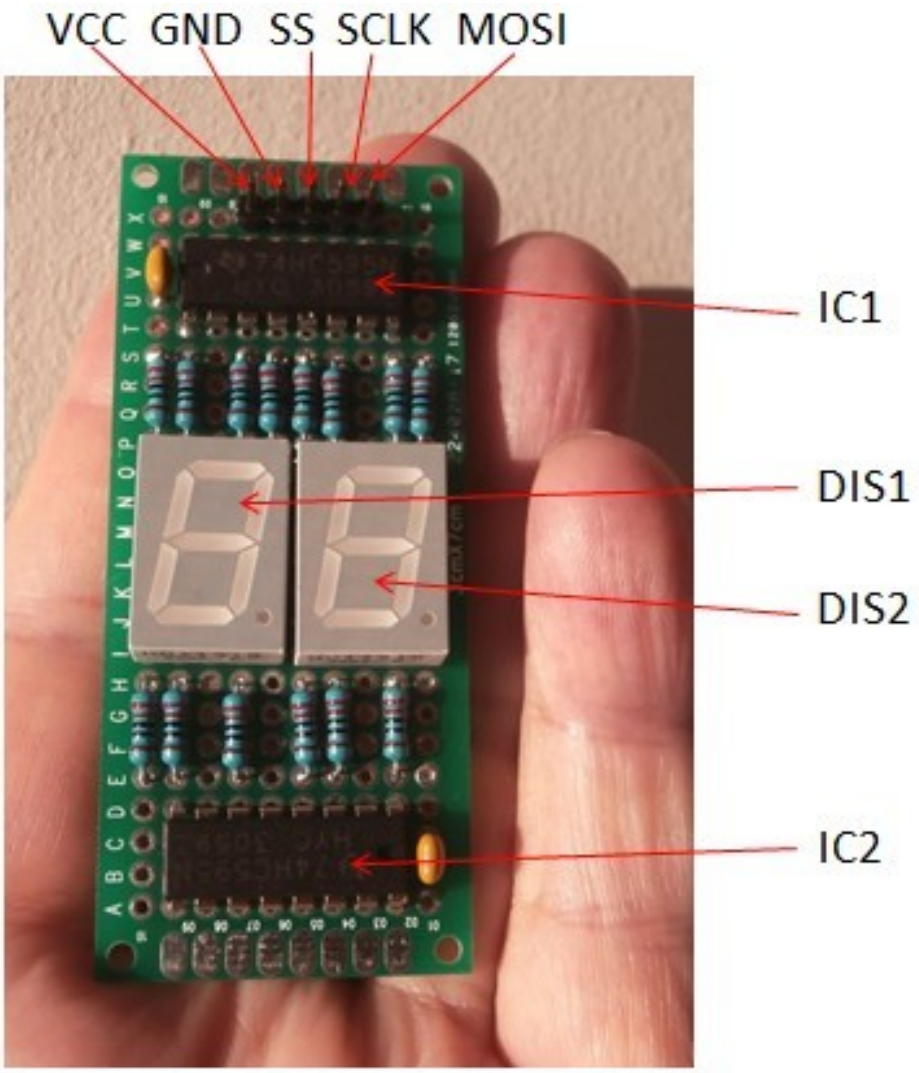
Adatküldés az SPI1 csatornán

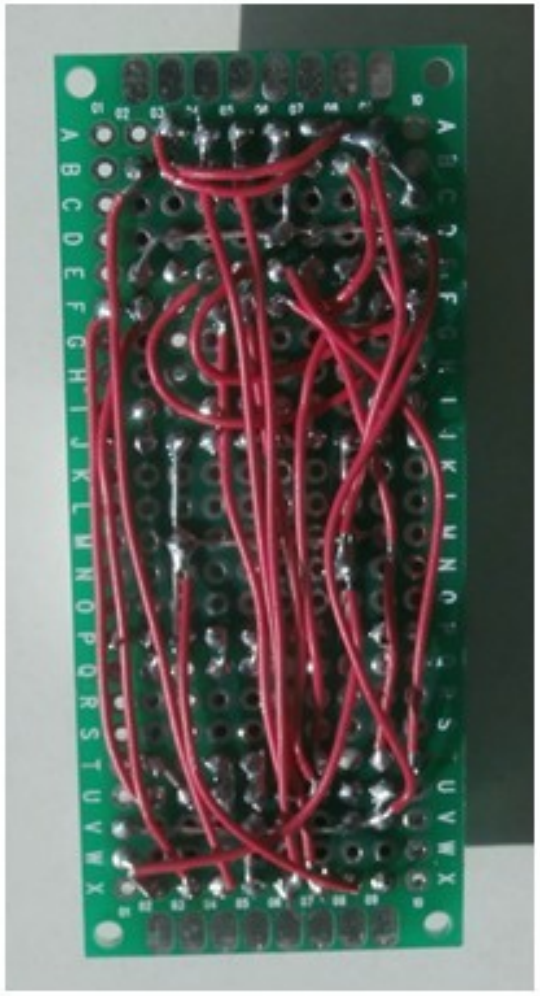
- Az ábrán látható kapcsolás közös anódú kijelzőkhöz készült, de könnyen átalakítható közös katódú kijelzőkhöz is
- Az SS jel szerepe itt az, hogy felfutó élre a shift regiszter tartalmát átírja a kimeneti regiszterbe



Adatküldés az SPI1 csatornán

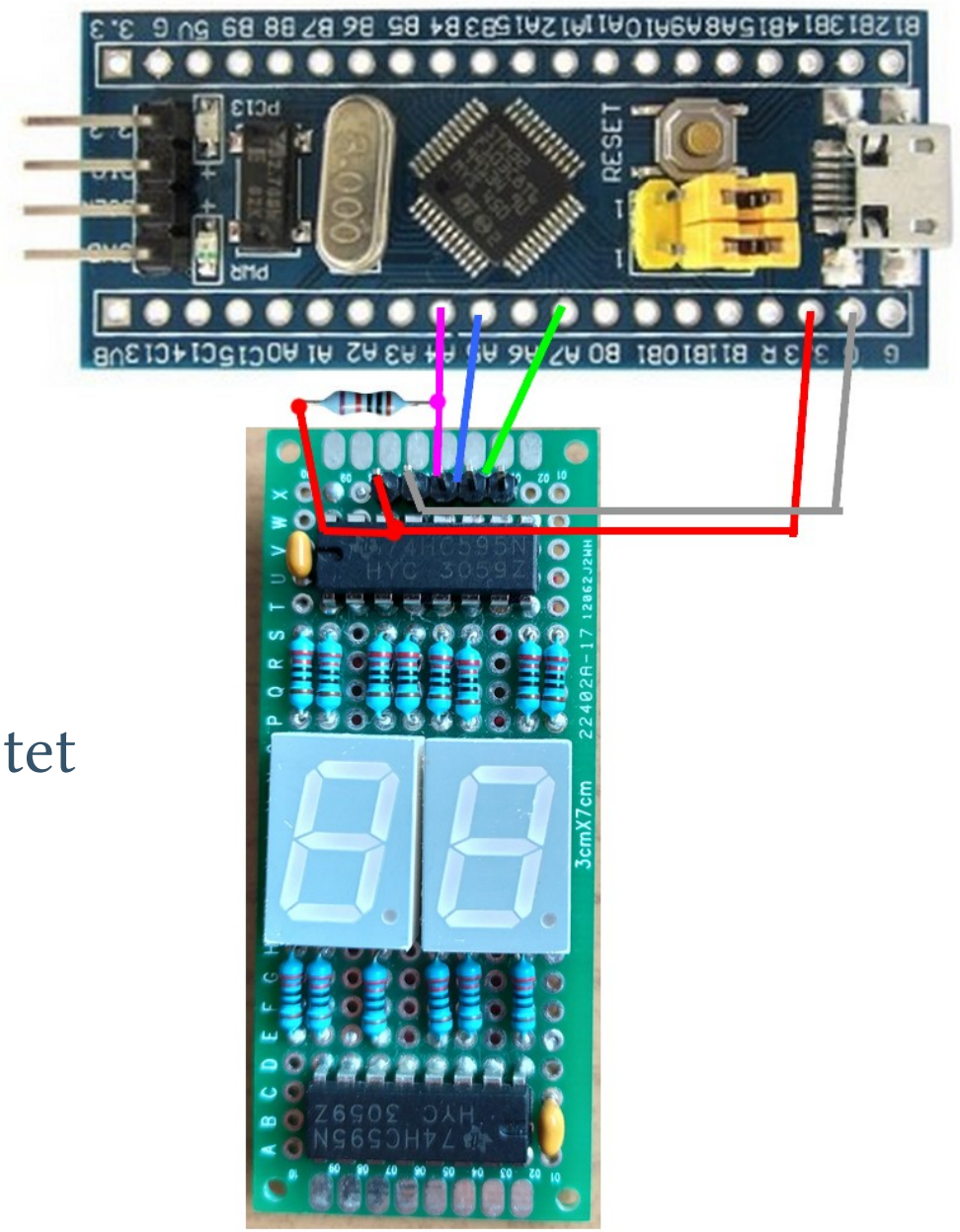
- A kapcsolást hevenyészett kivitelben egy 3x7 cm-es próbapanelon is megépíthetjük





Kapcsolási vázlat

- **A kijelző modul és a mikrovezérlő összekötése:**
  - VCC – 3.3 V
  - GND – G
  - SS – A4 (felhúzással!)
  - SCLK – A5
  - MOSI – A7



- **Megjegyzés:** ha az **SS** jelkimenetet (esetünkben az **A4** kivezetést) az SPI modul vezérli, akkor csak a lehúzás aktív (nyitott nyelő-elektródás mód), ezért kell egy 10 kΩ-os felhúzó ellenállás is

program08_1/main.c

```
#include "stm32f10x.h"
// Számjegy kódok (abcdefgDP)
const unsigned char digit[10] = {0xFC,0x60,0xDA,0xF2,0x66,0xB6,0xBE,0xE0,0xFE,0xF6};
void delayMs(int n);
void SPI1_init16(void);
void SPI1_write16(uint16_t data);

int main(void) {
  union {
    struct {
      uint8_t   d0;
      uint8_t   d1;
    };
    uint16_t  data;
  } d;

  SPI1_init16();                          // Az SPI1 modul konfigurálása
  while(1) {
    for(int n=0; n<100; n++) {
      d.d1 = ~digit[n/10];                // Első számjegy szegmensei
      d.d0 = ~digit[n%10];                // Második számjegy szegmensei
      SPI1_write16(d.data);               // Kiírás a kijelzőre
      delayMs(500);
    }
  }
}
```

program08_1/main.c (folytatás)

```
/*-------------------------------------------------------------------------
    Az SPI1 csatorna konfigurálása 16 bites módba, 2.25 MHz bitrátával
    Az alapértelmezett PA4=SS, PA5=SCK, PA7=MOSI kivezetéseket használjuk
    NSS vezérlése hardveresen történik
 *-------------------------------------------------------------------------*/
void SPI1_init16(void) {
  RCC->APB2ENR |= RCC_APB2ENR_SPI1EN;  // SPI1 órajel engedélyezés
  RCC->APB2ENR |= RCC_APB2ENR_AFIOEN;  // AFIO órajel engedélyezés
  RCC->APB2ENR |=  RCC_APB2ENR_IOPAEN; // GPIOA órajel engedélyezés
  AFIO->MAPR &= ~AFIO_MAPR_SPI1_REMAP; // Törli SPI1 REMAP bitjét

  /* PA5, PA7 konfigurálása, mint SPI1 SCK és MOSI */
  /* PA4 konfigurálás, mint GPIO kimenet SPI1 SS-nek */
  GPIOA->CRL &= ~0xF0FF0000;           // CNF és MODE bitek törlése
  GPIOA->CRL |=  0xB0BF0000;           // CNF=10 MODE=11 (PA5,PA7)
                                       // CNF=11 MODE=11 (PA4)
  SPI1->CR1 = SPI_CR1_DFF |            // 16 bit mód
              SPI_CR1_LSBFIRST|        // LSB először, 8bit mód
              SPI_CR1_BR_2|            // Baud rate 72/32 = 2.25MHz
              SPI_CR1_MSTR;            // Master mód beállítás
  SPI1->CR2 = SPI_CR2_SSOE;            // Single Master mód
  SPI1->CR1 |= SPI_CR1_SPE;            // SPI1 engedélyezés
}
```

I/O kimenetnek konfigurálás

CNF jelentése:
00: GPIO pushpull
01: GPIO open drain
10: ALTF pushpull
11: ALTF open drain

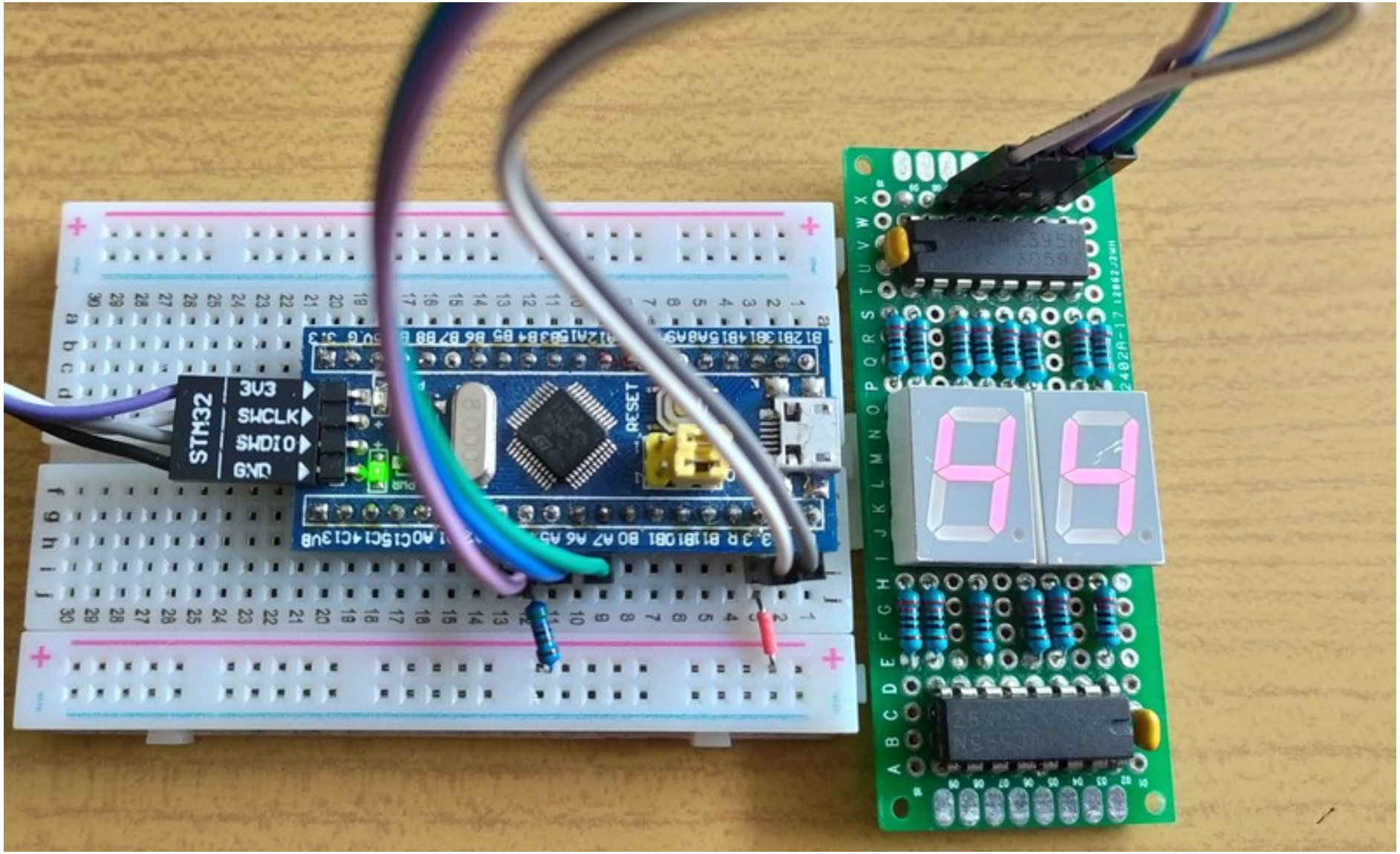
MODE jelentése:
01: 10 MHz
10: 2 MHz
11: 50 MHz

program08_1/main.c (folytatás)

```
/*-------------------------------------------------------------------
    16 bit adat kiküldése az SPI1 csatornán
 *-------------------------------------------------------------------*/
void SPI1_write16(uint16_t data) {
  SPI1->CR1 |= SPI_CR1_SPE;              // SPI1 engedélyezés
  while (!(SPI1->SR & 2)) {}             // TXE jelre várunk
  SPI1->DR = data;                       // Adat küldése (NSS itt lesz aktív)
  while (SPI1->SR & SPI_SR_BSY) {}       // Átvitel végére várunk
  SPI1->CR1 &= ~SPI_CR1_SPE;             // SPI1 letiltás (NSS itt lesz inaktív)
}

/*-------------------------------------------------------------------
    Késleltető eljárás a SysTick felhasználásával
 *-------------------------------------------------------------------*/
void delayMs(int n) {
  int i;
  SysTick->LOAD = SystemCoreClock/1000-1;  // Újratöltési érték 1 ms késleltetéshez
  SysTick->VAL  = 0;                       // Számláló törlése
  SysTick->CTRL = 0x5;                     // Enable, no interrupt, rendszer órajel
  for(i = 0; i < n; i++) {
    while((SysTick->CTRL & 0x10000) == 0); // A COUNT jelzöre várunk
  }
  SysTick->CTRL = 0;                       // SysTick leállítása
}
```

program08_1 futási eredmény



MAX7219 8x8 LED mátrix vezérlése

- A **Курс «Штурмуем STM32»** internetes tananyagban található egy MAX7219 IC-t kezelő programkönyvtár, ezt próbáljuk most ki
- A projektünkhöz hozzáadtuk a **max7219.h** és a **max7219.c** állományokat és a főprogramba becsatoltuk a **max7219.h** fejléc állományt
- **Az alábbi függvények állnak rendelkezésünkre:**

max7219_init(*br*) – inicializálás, fényerő beállítás

max7219_set_brightness(*br*) – fényerő beállítás

max7219_clear() - törlés

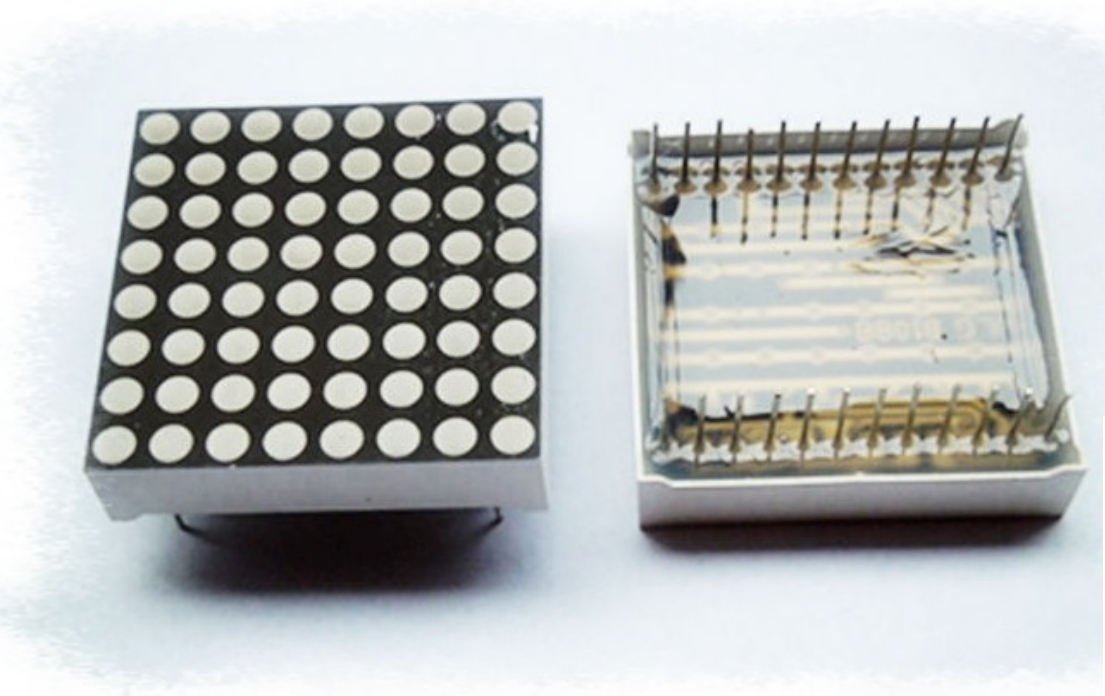
max7219_send_to_all(*reg, data*) – kiírás minden felfűzött modulra

max7219_send_to_chip(*reg, data, chip*) – kiírás adott modulra

max7219_send(*reg, data0, data1*) – írás két felfűzött modulra

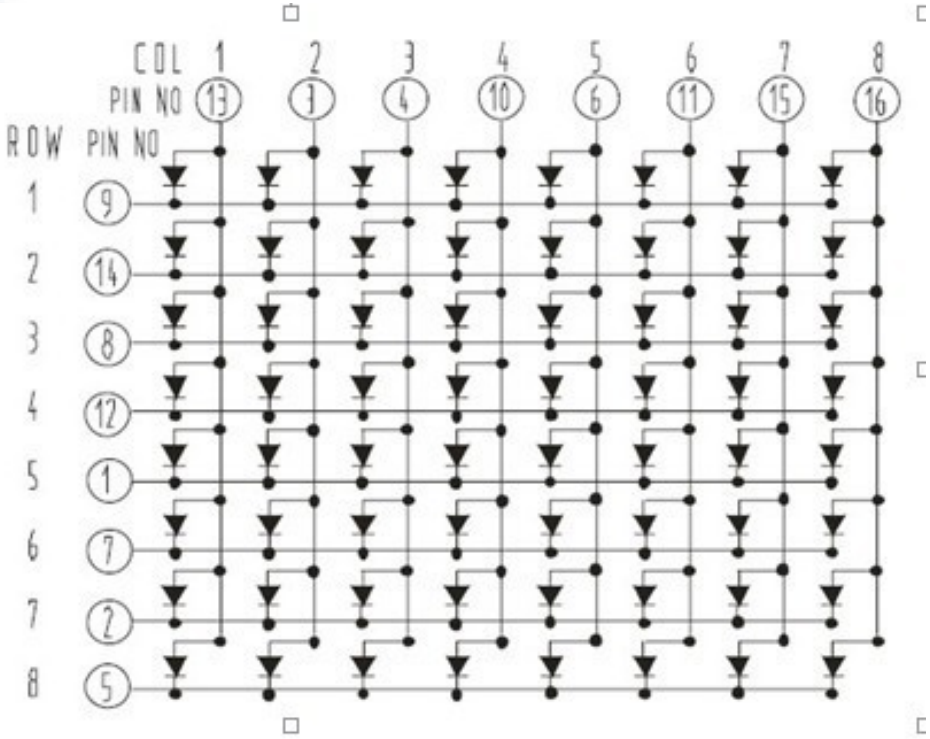
max7219_test() - teszt ábra megjelenítése

8x8 LED mátrix



3 mm-es piros LED-ek 8x8-as mátrixba szervezve

A **1088AS** típusnál a sorkiválasztó vonalak a katódokat közösítik



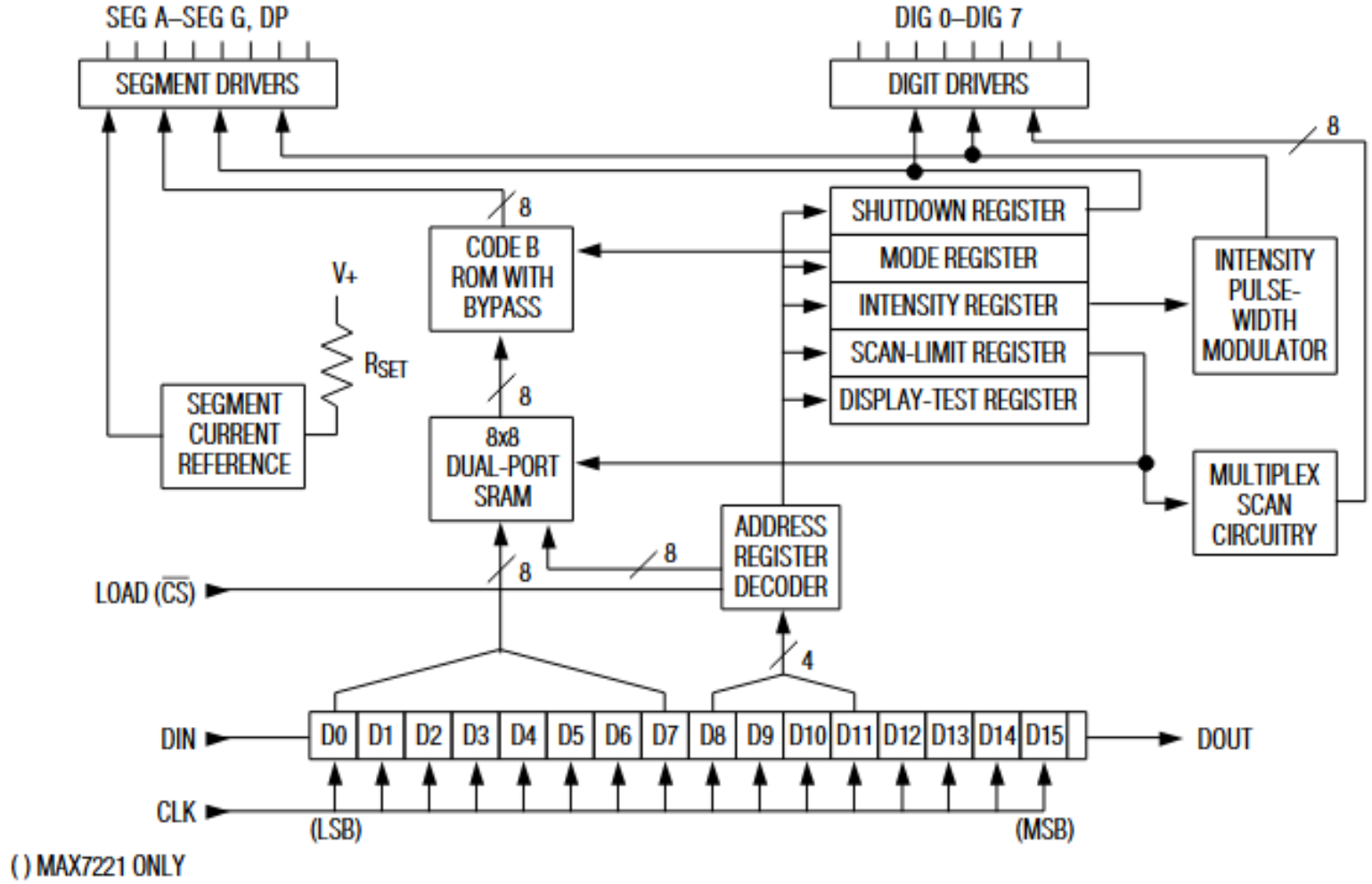
Multiplex kijelzés, egyidejűleg legfeljebb egy sor, vagy egy oszlop lehet aktív.

Kényelmes meghajtás:

- 1 db **MAX7219**, vagy
- 2 db **74HC595** (+ meghajtó +áramkorlátozás)
- 1 db **MCP23017** (+ meghajtó +áramkorlátozás)

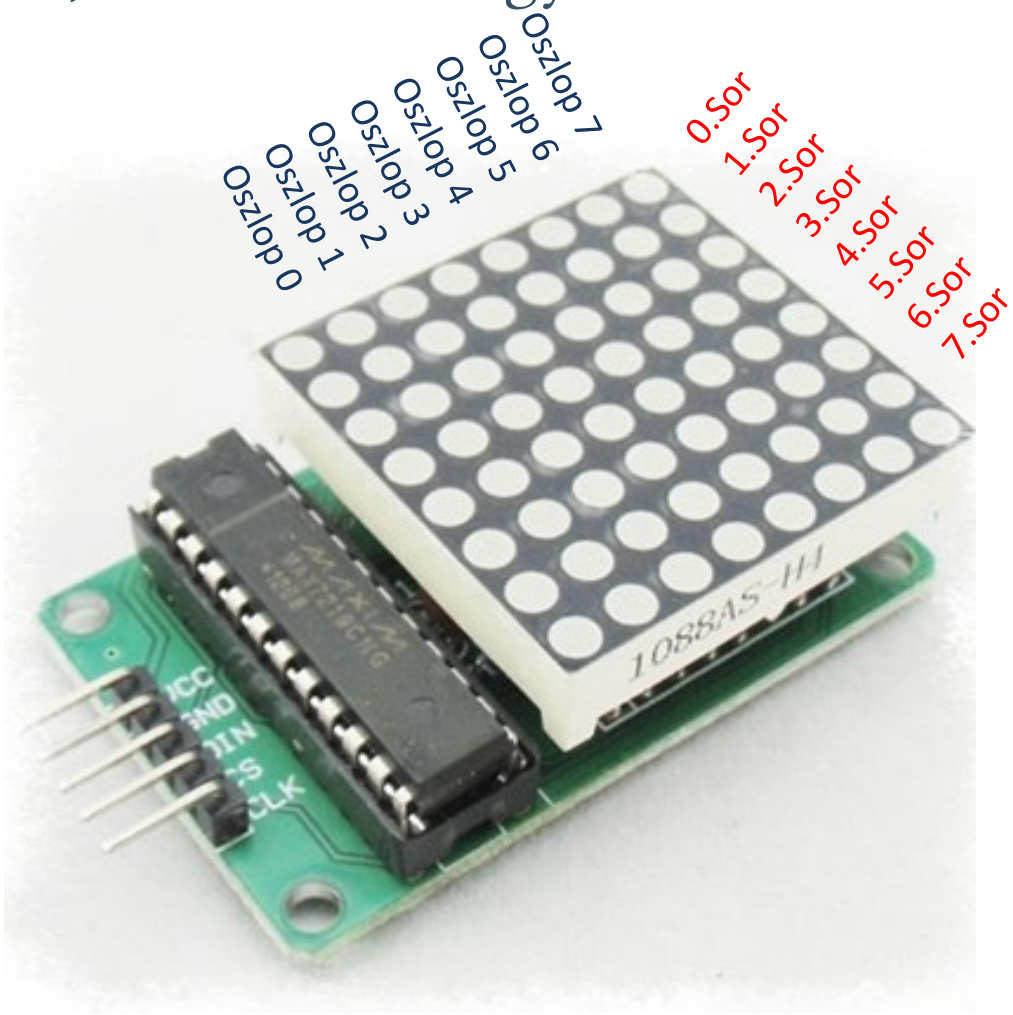
MAX7219

LED meghajtó IC, beépített áramkorlátozással. 7 szegmens kijelző esetén 1-8 db számjegy meghajtása (opcionálisan beépített dekódolással), vagy 8x8 LED mártix meghajtása. Az IC vezérlése SPI buszon történhet.



Kijelző modul MAX7219 IC-vel

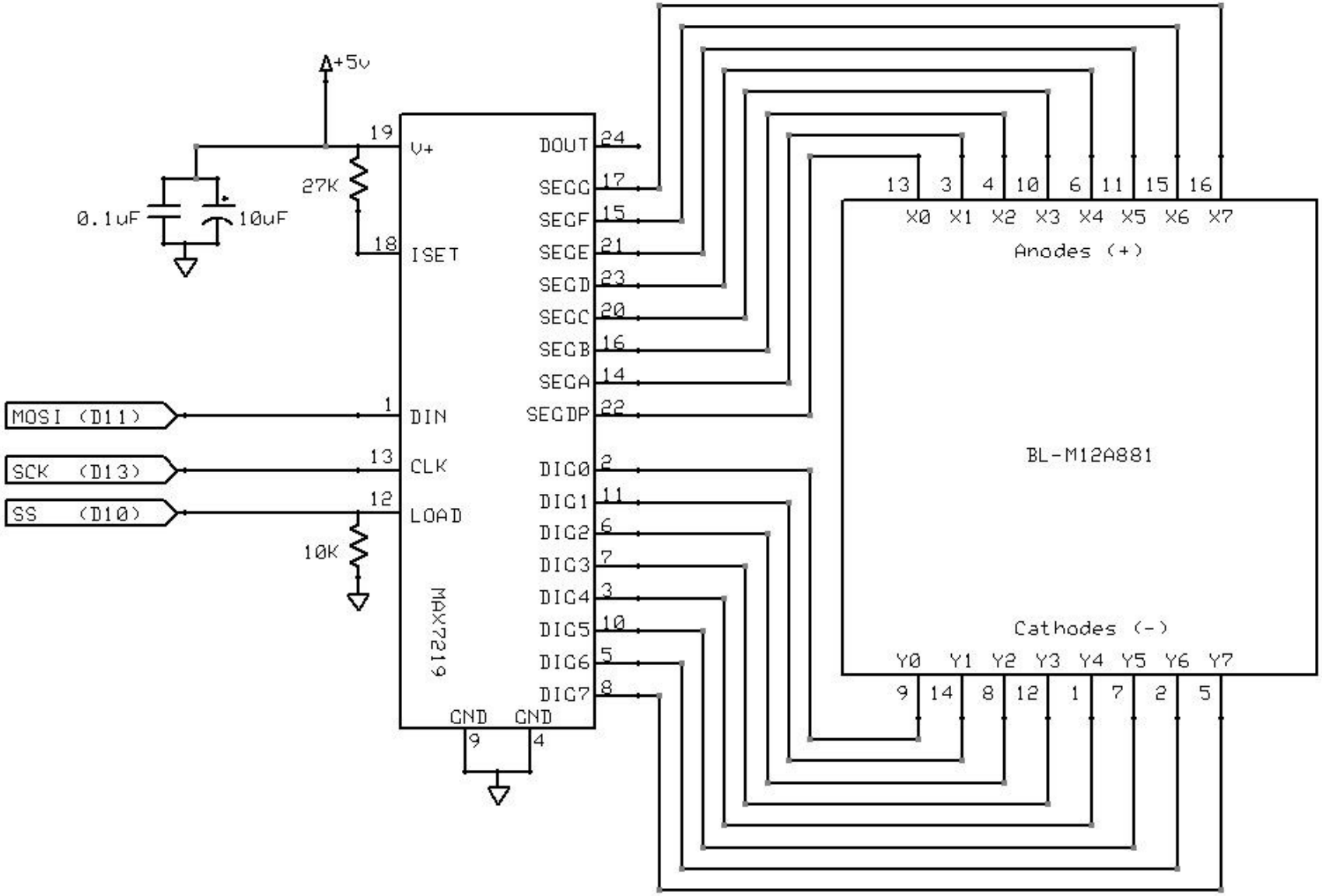
- Az E-bay kínálatában kapható, furatszerelt vagy felületszerelt kivitelben
  - 8x8 LED mátrix
  - MAX7219 vezérlő
  - Felfűzhető kivitel
  - Tápellátás: 3,5 – 5 V



Bemenetek	**Kimenetek**
1 VCC	1 VCC
2 GND	2 GND
3 DIN	3 DOUT
4 CS	4 CS
5 CLK	5 CLK

- **DIN/DOUT** – soros adat, **CLK** – szinkron órajel, **CS** – eszközkiválasztó jel, **VCC** – tápfeszültség, **GND** – a tápegység közös pontja („föld”)

Kapcsolási rajz



Technikai részletek az inicializáláshoz

Table 2. Register Address Map

REGISTER	ADDRESS					HEX CODE
	D15–D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	0xX0
Digit 0	X	0	0	0	1	0xX1
Digit 1	X	0	0	1	0	0xX2
Digit 2	X	0	0	1	1	0xX3
Digit 3	X	0	1	0	0	0xX4
Digit 4	X	0	1	0	1	0xX5
Digit 5	X	0	1	1	0	0xX6
Digit 6	X	0	1	1	1	0xX7
Digit 7	X	1	0	0	0	0xX8
Decode Mode	X	1	0	0	1	0xX9
Intensity	X	1	0	1	0	0xXA
Scan Limit	X	1	0	1	1	0xXB
Shutdown	X	1	1	0	0	0xXC
Display Test	X	1	1	1	1	0xXF

Adat

Nem használ adatot
DP a b c d e f g

0: no decode 1: decode

0 – 0xF
0 – 7
0: shutdown 1: normal mode
1: test mode 0: normal mode

program08_2/main.c

```
#include "stm32f10x.h"
#include "max7219.h"
void delayMs(int n);
const unsigned char led1[]= {0xFF,0x00,0x30,0x48,0x90,0x90,0x48,0x30};
const unsigned char led2[]= {0xFF,0x18,0x18,0xFF,0x00,0xF8,0xA8,0xA8};

int main(void) {
  max7219_init(8);          // Inicializálás, közepes fényerő
  while(1) {
    for(int i=0; i<8; i++) {
      max7219_send(i+1,led1[i],led2[i]);
      delayMs(200);
       }
    delayMs(2000);
    max7219_clear();
    delayMs(1000);
  }
}
```

Két, sorbakötött MAX7219 vezérlővel ellátott 8x8 LED mátrixon megjelenítünk egy-egy képet, majd 2 másodperc múlva töröljük a képet és 1 másodperc múlva újra kezdjük...

- A **max7219** programkönyvtárban az **SPI2** csatornát használjuk **NSS**=PB12 , **SCK**=PB13, **MOSI**=PB15, **MISO**=PB14 (nem használjuk)

program08_2/main.c

```
#include "max7219.h"
#define CS_ACTIVATE()           (GPIOB->ODR &= ~GPIO_ODR_ODR12)
#define CS_DEACTIVATE()         (GPIOB->ODR |= GPIO_ODR_ODR12)

void max7219_init(const uint8_t br) {
  RCC->APB1ENR |= RCC_APB1ENR_SPI2EN;
  RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;
  //--- SCK, MOSI pin configuration ----------------------
  GPIOB->CRH |= GPIO_CRH_MODE13;   // 11: 50M Hz
  GPIOB->CRH &= ~GPIO_CRH_CNF13_0;
  GPIOB->CRH |=  GPIO_CRH_CNF13_1; // 10: Alt. Func, pushpull
  GPIOB->CRH |= GPIO_CRH_MODE15;   // 11: 50M Hz
  GPIOB->CRH &= ~GPIO_CRH_CNF15_0;
  GPIOB->CRH |=  GPIO_CRH_CNF15_1; // 10: Alt. Func, pushpull
  //--- CS pin configuration for software mode handling ---
  GPIOB->CRH |= GPIO_CRH_MODE12;   // 11: 50 MHz
  GPIOB->CRH &= ~GPIO_CRH_CNF12;   // 00: GPIO PushPull OUTput
  CS_DEACTIVATE();
  SPI2->CR1 &= SPI_CR1_SPE;        // disable module
  SPI2->CR1 |= SPI_CR1_MSTR;       // master mode
  SPI2->CR1 |= SPI_CR1_BIDIMODE;   // 1 line
  SPI2->CR1 |= SPI_CR1_BIDIOE;     // MOSI
  SPI2->CR1 &= ~SPI_CR1_BR;        // spi_sck = fPCLK/2/16 = 2.25 MHz
  SPI2->CR1 |= SPI_CR1_BR_1;
  SPI2->CR1 &= ~SPI_CR1_LSBFIRST;  // MSB first
  SPI2->CR1 |= SPI_CR1_DFF;        // 16 bit format
```

A MAX7219 programkönyvtár a Курс «Штурмуем STM32»tananyagban található

I/O kimenetnek konfigurálás

CNF jelentése:
00: GPIO pushpull
01: GPIO open drain
10: ALTF pushpull
11: ALTF open drain

MODE jelentése:
01: 10 MHz
10: 2 MHz
11: 50 MHz

program08_2/main.c

```
  SPI2->CR1 |= SPI_CR1_SSM;         // software CS
  SPI2->CR1 &= ~SPI_CR1_CPHA;       // First edge
  SPI2->CR1 &= ~SPI_CR1_CPOL;       // SCK = 0 at rest
  SPI2->CR2 = SPI_CR2_SSOE;         // Enable NSS output
  SPI1->I2SCFGR &= ~SPI_I2SCFGR_I2SMOD; //SPI mode
  SPI2->CR1 |= SPI_CR1_SPE;         // Enable SPI2 module
  //--- set MAX7219 mode
  max7219_send_to_all(REG_SHUTDOWN,    0x01); // restart
  max7219_send_to_all(REG_DECODE_MODE, 0x00); // use normal mode
  max7219_send_to_all(REG_SCAN_LIMIT,  0x07); // use all lines
  max7219_set_brightness(br);
  max7219_clear();
}

void max7219_set_brightness(const uint8_t br) {
  max7219_send_to_all(REG_INTENSITY, br > 15 ? 15 : br);
}

void max7219_clear(void) {
  for (uint32_t i = 0; i < 8; i++) max7219_send_to_all(REG_DIGIT_0 + i, 0x00);
}

static inline void send_data(uint8_t reg, uint8_t data) {
  SPI2->DR = (uint16_t)( (reg << 8) | data );
  while ( (SPI2->SR & SPI_SR_BSY) == SPI_SR_BSY);
}
```

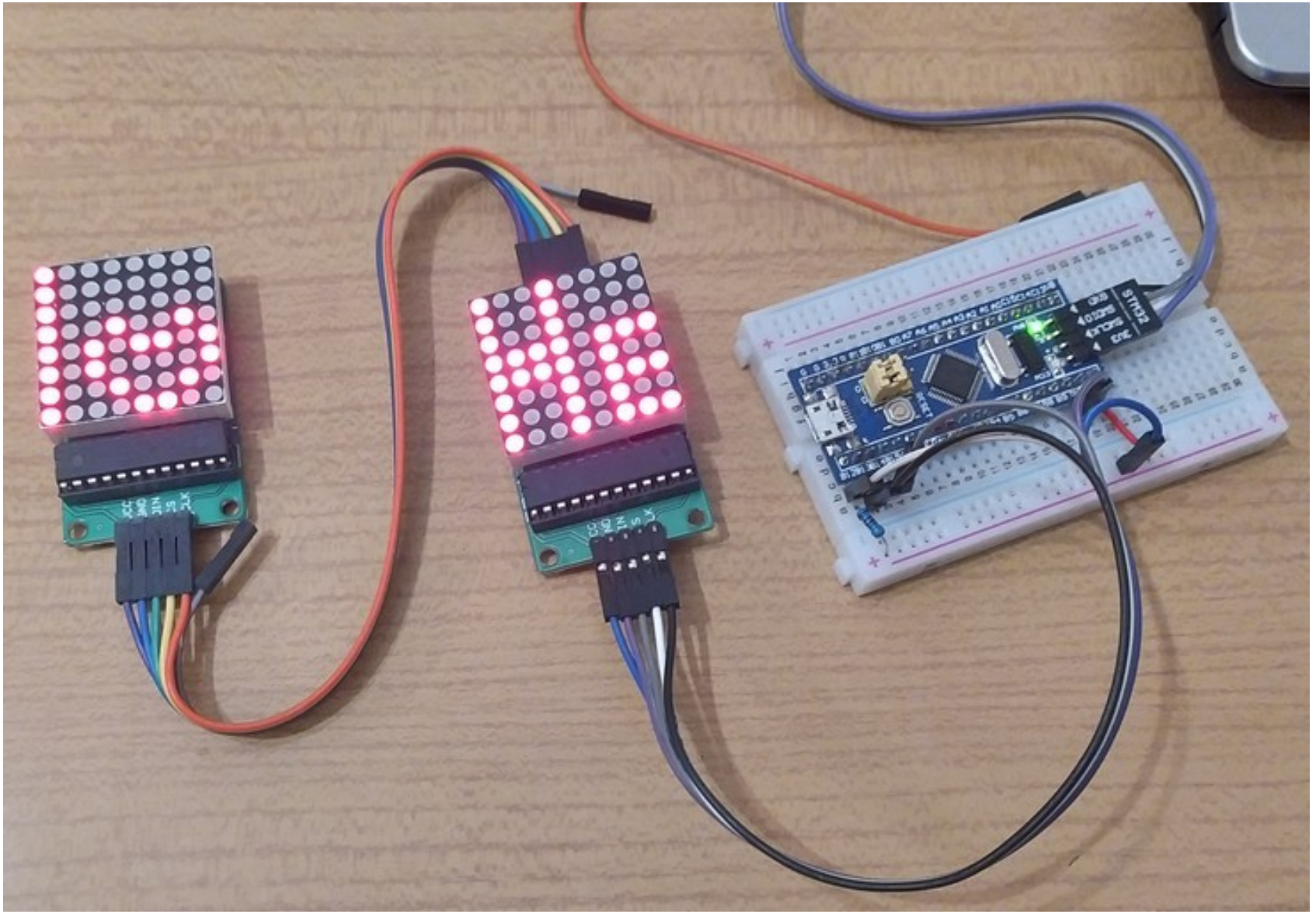
program08_2/main.c

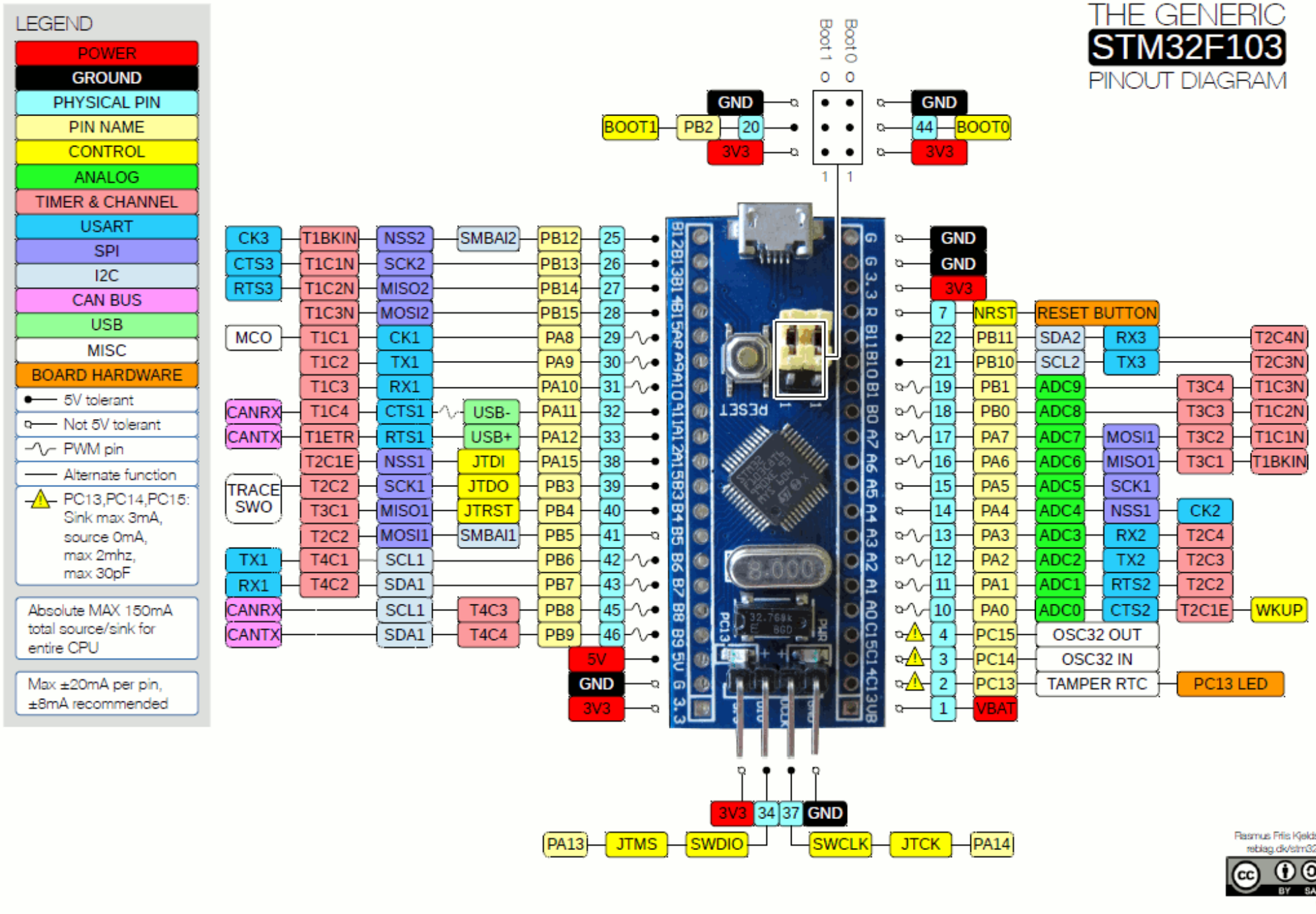
```
void max7219_send_to_all(const uint8_t reg, const uint8_t data) {
  CS_ACTIVATE();
  for (uint32_t i = 0; i < MAX7219_CHIPS; i++) send_data(reg, data);
  CS_DEACTIVATE();
}

void max7219_send_to_chip(const uint8_t reg, const uint8_t data, const uint8_t chip) {
  if (chip > MAX7219_CHIPS) {
    return;
  }
  CS_ACTIVATE();
  for (uint32_t i = 0; i < MAX7219_CHIPS; i++) {
    if ( (MAX7219_CHIPS - i - 1) == chip)
      send_data(reg, data);
    else
      send_data(REG_NO_OP, data);
  }
  CS_DEACTIVATE();
}

void max7219_send(const uint8_t reg, const uint8_t data_0, const uint8_t data_1) {
  CS_ACTIVATE();
  send_data(reg, data_0);
  send_data(reg, data_1);
  CS_DEACTIVATE();
}
```

program08_2 futási eredmény




LEGEND
POWER
GROUND
PHYSICAL PIN
PIN NAME
CONTROL
ANALOG
TIMER & CHANNEL
USART
SPI
I2C
CAN BUS
USB
MISC
BOARD HARDWARE
5V tolerant
Not 5V tolerant
PWM pin
Alternate function
PC13,PC14,PC15: Sink max 3mA, source 0mA, max 2mhz, max 30pF
Absolute MAX 150mA total source/sink for entire CPU
Max ±20mA per pin, ±8mA recommended
THE GENERIC
STM32F103
PINOUT DIAGRAM
Boot1
Boot0
GND
BOOT1
PB2
20
3V3
GND
44
BOOT0
3V3
CK3
CTS3
RTS3
MCO
CANRX
CANTX
TRACE SWO
TX1
RX1
CANRX
CANTX
T1BKIN
T1C1N
T1C2N
T1C3N
T1C1
T1C2
T1C3
T1C4
T1ETR
T2C1E
T2C2
T3C1
T2C2
T4C1
T4C2
NSS2
SCK2
MISO2
MOSI2
CK1
TX1
RX1
CTS1
RTS1
NSS1
SCK1
MISO1
MOSI1
SCL1
SDA1
SCL1
SDA1
SMBAI2
USB-
USB+
JTDI
JTDO
JTRST
SMBAI1
T4C3
T4C4
PB12
PB13
PB14
PB15
PA8
PA9
PA10
PA11
PA12
PA15
PB3
PB4
PB5
PB6
PB7
PB8
PB9
5V
GND
3V3
25
26
27
28
29
30
31
32
33
38
39
40
41
42
43
45
46
GND
GND
3V3
7
22
21
19
18
17
16
15
14
13
12
11
10
4
3
2
1
NRST
PB11
PB10
PB1
PB0
PA7
PA6
PA5
PA4
PA3
PA2
PA1
PA0
PC15
PC14
PC13
VBAT
RESET BUTTON
SDA2
SCL2
ADC9
ADC8
ADC7
ADC6
ADC5
ADC4
ADC3
ADC2
ADC1
ADC0
OSC32 OUT
OSC32 IN
TAMPER RTC
RX3
TX3
MOSI1
MISO1
SCK1
NSS1
RX2
TX2
RTS2
CTS2
T3C4
T3C3
T3C2
T3C1
CK2
T2C4
T2C3
T2C2
T2C1E
PC13 LED
T2C4N
T2C3N
T1C3N
T1C2N
T1C1N
T1BKIN
WKUP
3V3
34
37
GND
PA13
JTMS
SWDIO
SWCLK
JTCK
PA14
Rasmus Friis Kjeldsen
reblag.dk/stm32
V1.0