Gingl Zoltán, 2017, Szeged

Mikrovezérlők Alkalmazástechnikája

Kommunikációs áramkörök

Kommunikációs áramkörök

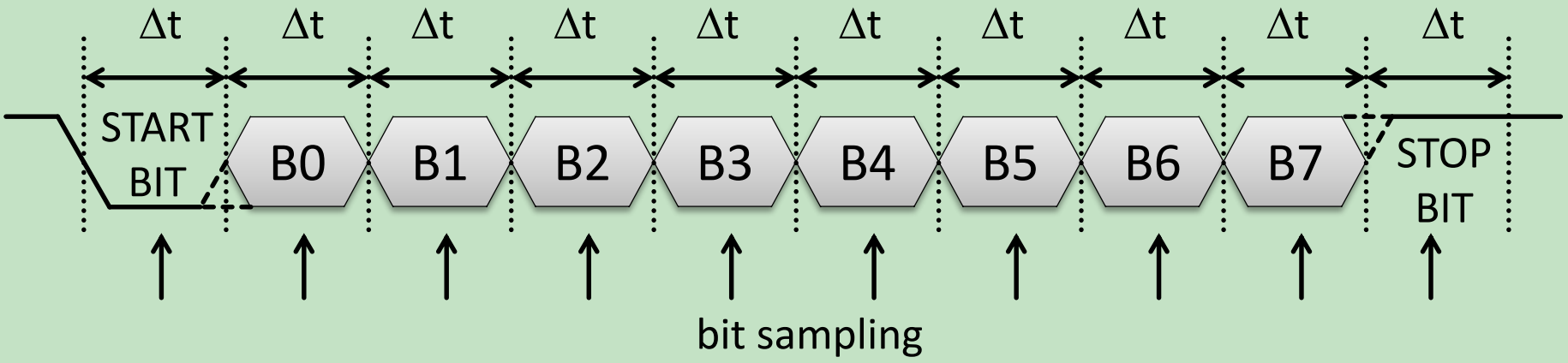
- Processzoroknál tipikusan párhuzamos átvitel
  - adatbusz
  - címbusz
  - vezérlőjelek, szinkronizálás
- Mikrovezérlőknél soros átvitel
  - Kevés vezeték illetve jel
  - Modulációt igényel
    - az információt alakítsuk akár egyetlen skalár jellé, bitfolyammá
  - Demoduláció szükséges
  - Információhordozó
    - feszültség, áram, fény, elekromágneses hullám, hang
  - Forma: amplitúdó, frekvencia, fáziskülönbség

UART áramkörök

UART: Universal Asynchronous Receive/Transmit

- **Processzorok és eszközök** közti kommunikációhoz
- Egy byte átvitele bitenként
- Egyetlen kétállapotú jel
- Nincs szinkronizáló jel
- Az átvitel bármikor történhet
- Az átvitel kezdetét detektálni kell a vevőnek
- A kommunikáló eszközök azonos időalappal küldik és fogadják a biteket (baud rate=bit/s)
- Gyakran használt
- Sokféle hardveres megoldás (RS232, RS485, stb.)

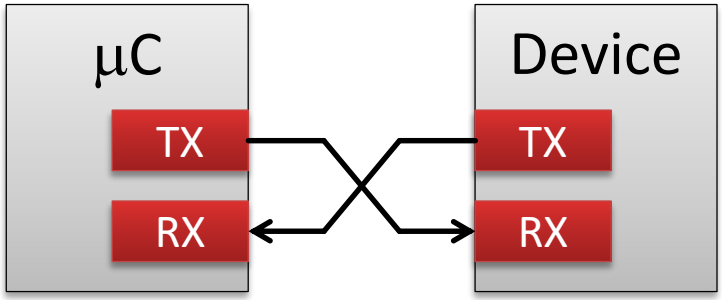
UART ▸ Idődiagram



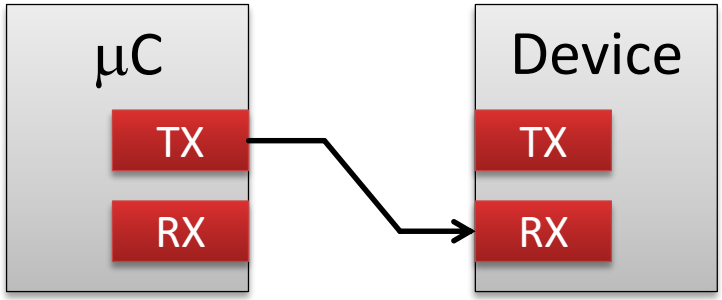
- Start bit (logikai 0), átvitel detektálására
- Stop bit (logikai 1), minimum szünet küldések között
- Byte átviteli idő: $10 \cdot \Delta t$
- 9 bites átvitel is választható:
  - az utolsó bit külön programozható (TB8, olvasáskor RB8)
  - paritásbitként, multiprocesszoros adatátvitelhez
  - $11 \cdot \Delta t$ byte átviteli idő

UART ▸ Kötési módok

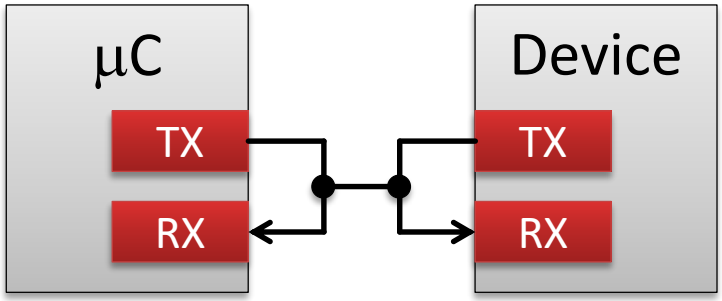
FULL DUPLEX



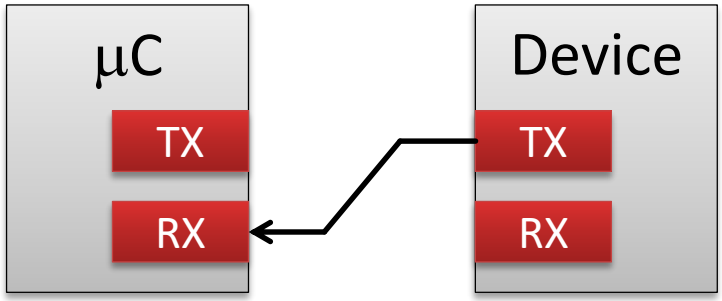
SIMPLEX



HALF DUPLEX



SIMPLEX



TX KIKAPCSOLHATÓ

UART ▸ Időalap előállítása

- Timer túlcsordulások vezérlik, kétféle megoldás:
  - Baud rate = Timer overflow rate / 16 (pl. F120)
  - Baud rate = Timer overflow rate / 2 (pl. F410)
- Egy (Timer1) vagy több (Timer1-4) timer
- Néhány processzoron akár más TX és RX ráta
- Maximális ráta
  - SYSCLK/16
  - vagy SYSCLK/2

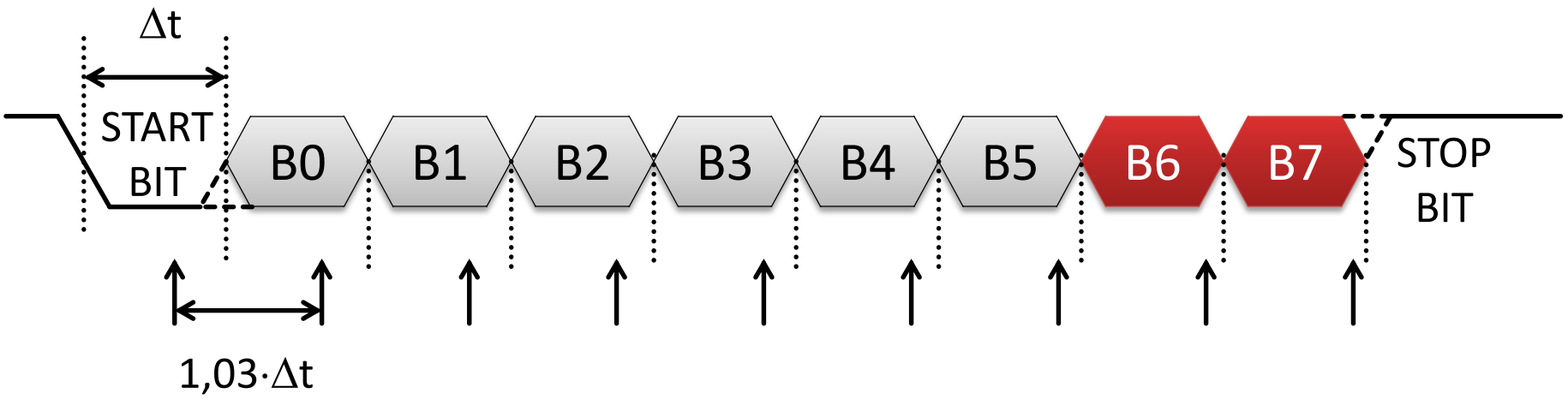
UART ▸ Baud rate beállítás

- Baud rate = Timer overflow rate / 16 (pl. F120)
  - Timer 1 : TH1=256-SYSCLK/(16*Baud rate)
  - Timer 2-4 : TMRRL=65536-SYSCLK/(16*Baud rate)
- Baud rate = Timer overflow rate / 2 (pl. F410)
  - Timer 1 : TH1=256-SYSCLK/(2*Baud rate)
- Érdemes figyelni a kerekítési hibára!
  - A C-ben az egész osztás nem a legközelebbire kerekít!
  - Mindig ellenőrizzük a beállított érték okozta hibát!

UART ▸ Időalapok különbsége

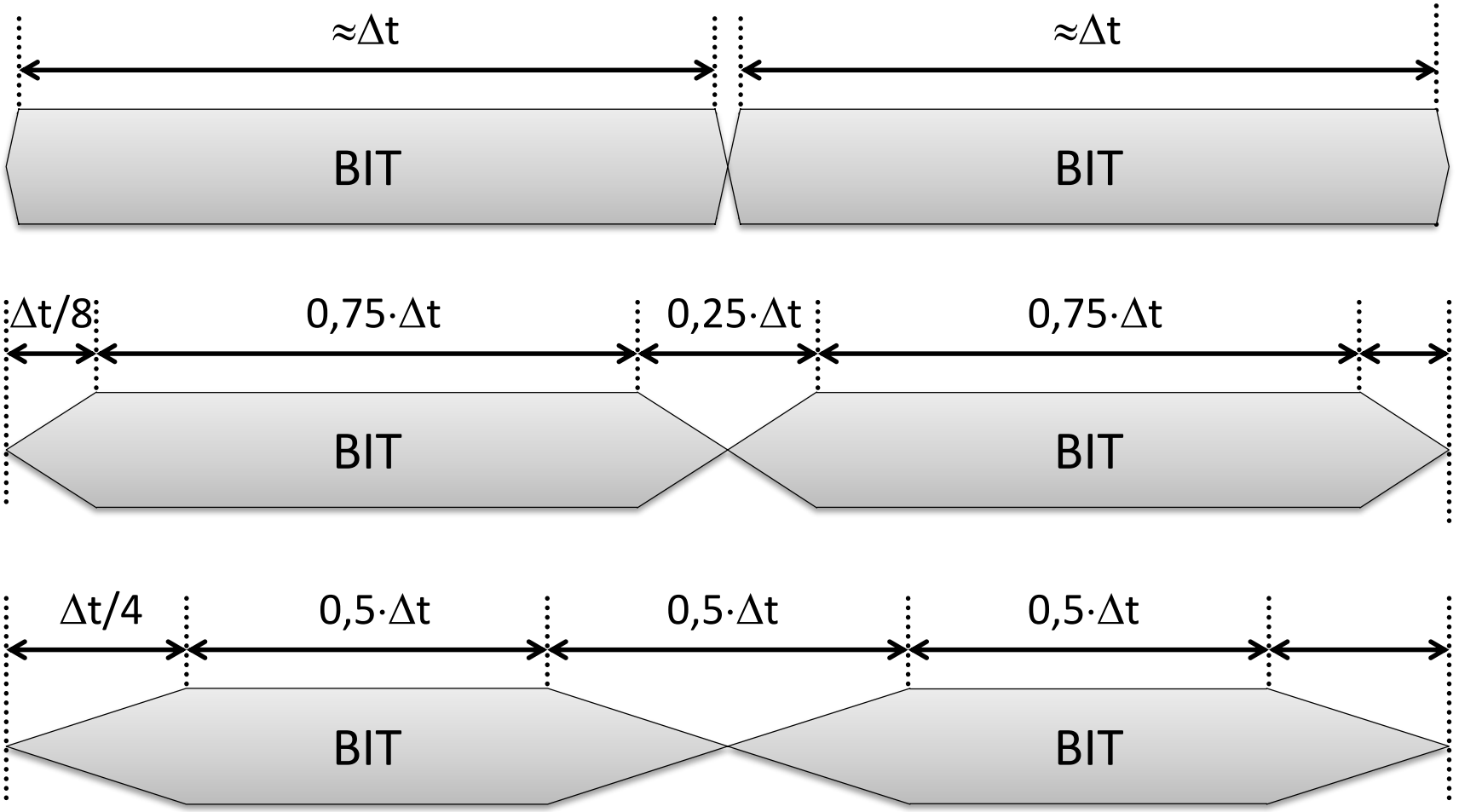
- Az időalapok különbözők lehetnek
- Az egységek órajele nem egzakt
- A beállítás frekvencia egésszel osztásával
  - nem pontosan a kívánt érték
- A kettő eltérése mennyi lehet?
  - maximum 3%, szigorúbban 2%
  - teljesül?
  - http://pdfserv.maximintegrated.com/en/an/AN2141.pdf

UART ▸ Az időalapok eltérése ▸ 3% példa



- Az adó és vevő hibája együttesen hat – hogyan?
- A detektálás ideálisan a bitek közepén lenne
- A start bit detektálása sem pontos
- A megbízhatóság függ a jelváltások időtartamától
- Függ a 2-szeres vagy 16-szoros leosztási értéktől

UART ▸ Az időalapok eltérése ▸ jelalakok



UART ▸ Az időalapok eltérése ▸ példa

- Tegyük fel
  - elég rövid jelváltási idők, 3% tolerancia megengedett
  - 9600 bit/s kívánt érték
- F410 beállítás
  - SYSCLK = 191406Hz±2%
  - Timer1 8-bit auto reload mode: TH1 = 246
  - Baud rate = 191406Hz ±2% /10/2=9570Hz ±2%

```
void Timer1_Init()
{
  TMOD  = 0x20;  // 8-bit auto reload
  CKCON = 0x08;  // timer clock = system clock
  TH1   = 0xF6;  // reload value = 246
  TL1   = 0xFF;  // one step from overflow
  TCON  = 0x40;  // run timer
}
```

UART ▸ Az időalapok eltérése ▸ példa

- Baud rate = 191406Hz/10/2 = 9570Hz
- 9570Hz $\pm$2%
- Megfelel?
  - 9600$\pm$3% $\rightarrow$ 9312-9888
  - 9570$\pm$2% $\rightarrow$ 9378-9761
- A másik eszköz hibáját nem vettük figyelembe!

UART ▸ Működési módok, beállítások

- A szinkron módokkal nem tárgyaljuk
- 8 vagy 9 bites aszinkron mód
- Az adat fogadása engedélyezhető, tiltható
- A stop bit detektálása engedélyezhető
  - csak akkor van vett adat, ha a stop bit értéke 1
  - 9-bites módban a 9. bitet vizsgálja! Ez alkalmas címzésre, multiprocesszoros kommunikációra
  - Ha a 9. bit 1, akkor cím érdekezik a vevőkhöz
  - A címzett vevő kikapcsolja a bit monitorozását a vétel idejére

UART ▸ Küldés és fogadás ▸ Polling mód

```
void UART_Init()
{
  SCON0 = 0x10; // 8-bit, variable baud mode
  TI=1;         // assume empty output buffer
}

unsigned char UARTIn(void)
{
  while (!RI); // wait for a byte
  RI=0;        // clear flag (prepare for next reception)
  return SBUF; // return the byte
}

void UARTOut(char a)
{
  while (!TI); // wait for end of previous transmission
  TI=0;        // clear flag
  SBUF=a;      // transmit a byte, don't wait for end
}
```

UART ▸ Küldés és fogadás ▸ Interrupt mód

```
#define BUFFERSIZE 8

// ring buffers
volatile unsigned char TxBuffer[BUFFERSIZE];
volatile unsigned char RxBuffer[BUFFERSIZE];

// TX buffer read and RX  buffer write pointers
// used in the interrupt routine
volatile unsigned char TxReadPtr=0, RxWritePtr=0;

// TX buffer write and RX buffer read pointers
unsigned char TxWritePtr=0, RxReadPtr=0;

// Number of data in the buffers
volatile unsigned char TxNumberOfData=0;
volatile unsigned char RxNumberOfData=0;
```

UART ▸ Küldés és fogadás ▸ Interrupt mód

```
void UARTInterrupt(void) __interrupt UART_VECTOR
{
  if (RI)
  {
    RI=0;
    if (RxNumberOfData < BUFFERSIZE)
    {
      RxBuffer[RxWritePtr]=SBUF;
      RxWritePtr = (RxWritePtr+1) % BUFFERSIZE;
      RxNumberOfData++;
    }
  }
```

UART ▸ Küldés és fogadás ▸ Interrupt mód

```
  if (TI)
  {
    TI=0;
    if (TxNumberOfData)
    {
      SBUF=TxBuffer[TxReadPtr];
      TxReadPtr = (TxReadPtr+1) % BUFFERSIZE;
      TxNumberOfData--;
    }
  }
}
```

UART ▸ Küldés és fogadás ▸ Interrupt mód

```
unsigned char UARTIn(unsigned char *c)
{
  if (RxNumberOfData)
  {
    RxNumberOfData--;
    *c=RxBuffer[RxReadPtr];
    RxReadPtr = (RxReadPtr+1) % BUFFERSIZE;
    return 0;
  }
  return 1;
}
```

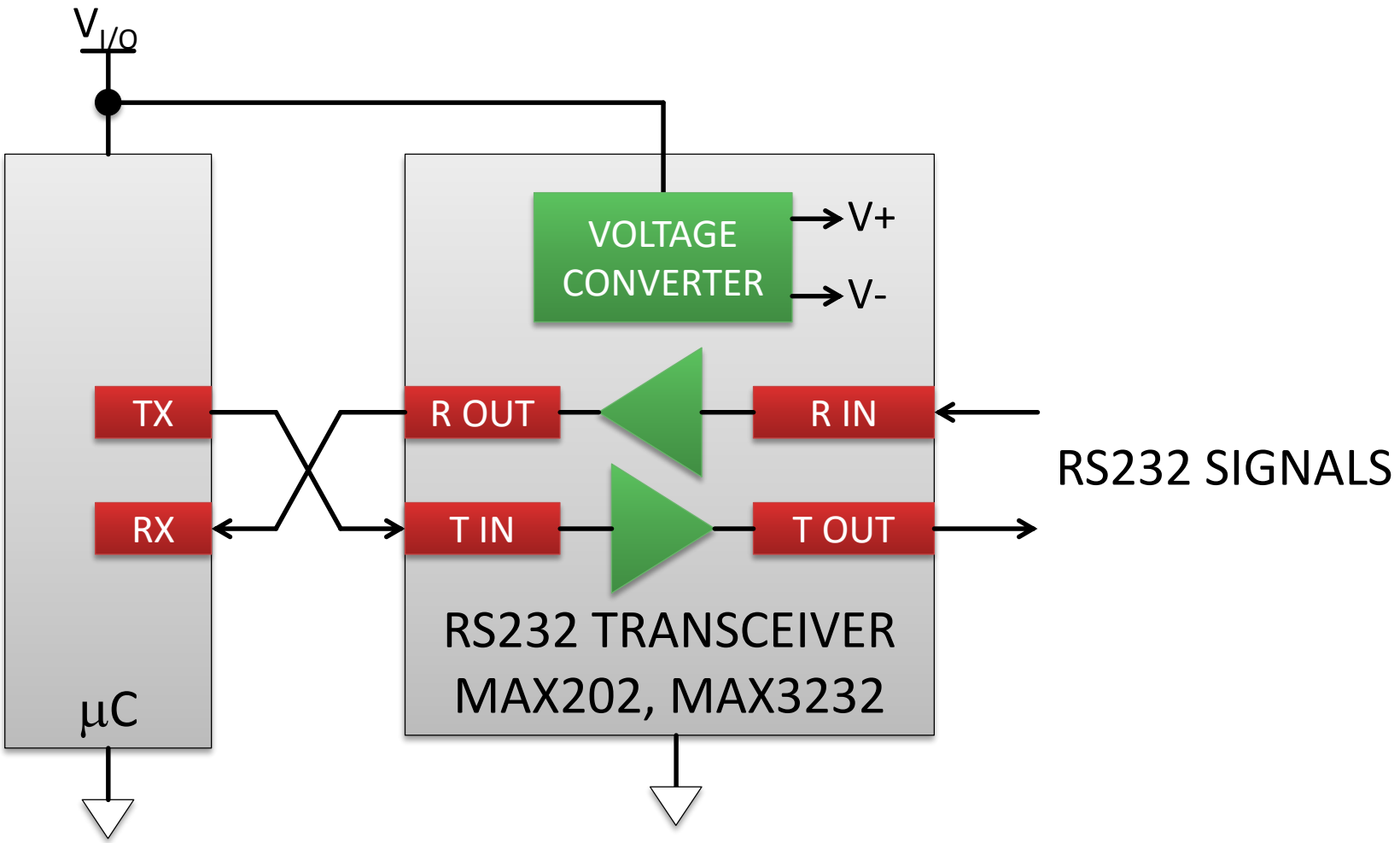
UART ▸ Küldés és fogadás ▸ Interrupt mód

```
unsigned char UARTOut(unsigned char c)
{
  if (TxNumberOfData < BUFFERSIZE)
  {
    TxNumberOfData++;
    TxBuffer[TxWritePtr]=c;
    TxWritePtr = (TxWritePtr+1) % BUFFERSIZE;
    return 0;
  }
  return 1;
}
```

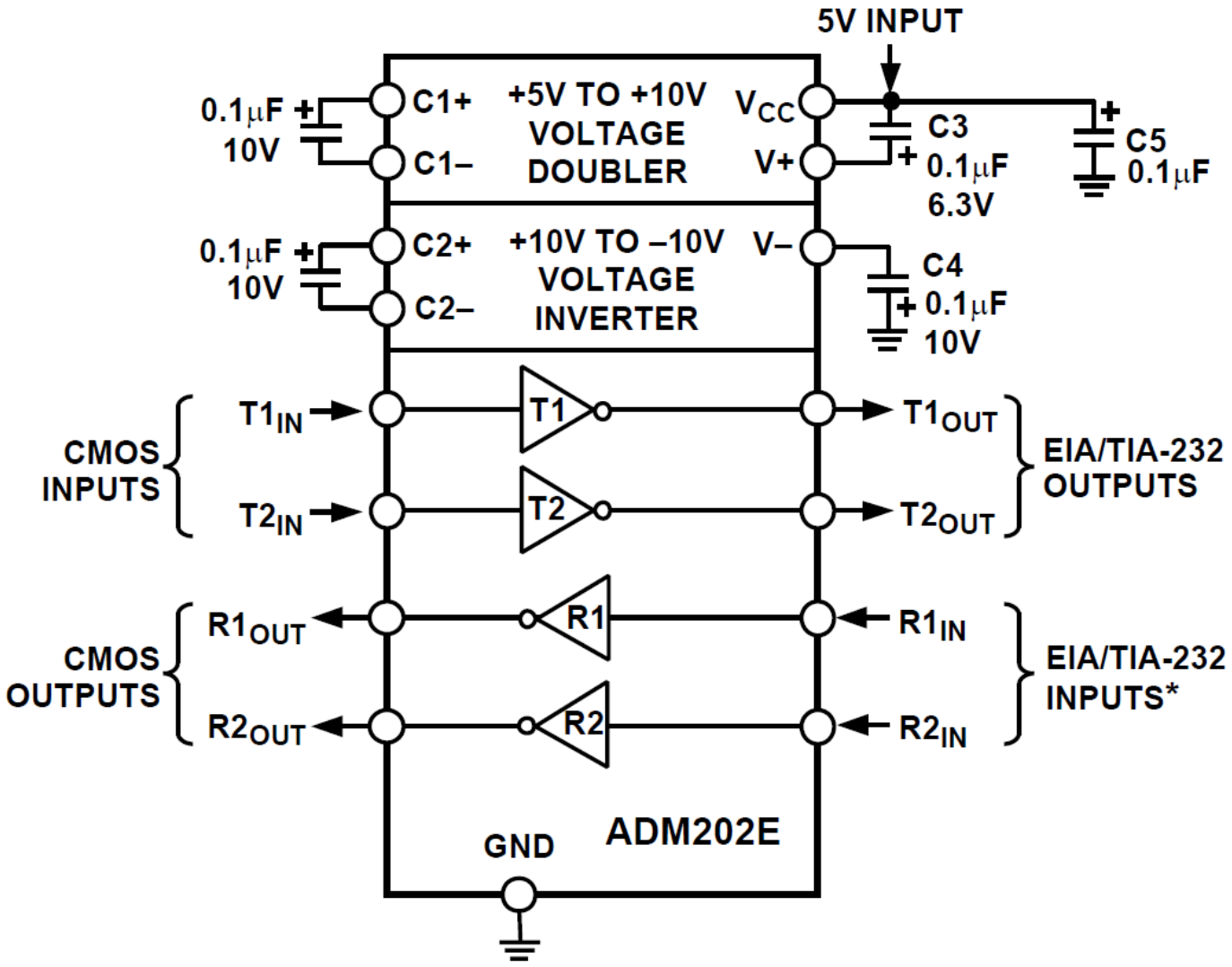
UART ▸ Fizikai rétegek

- Meghajtó áramkörök nélkül
- Meghajtóáramkörökkel
  - nagyobb távolságok
  - PLC-k
  - Műszerek
    - oszcilloszkóp
    - DVM
    - …
  - Speciális eszközök

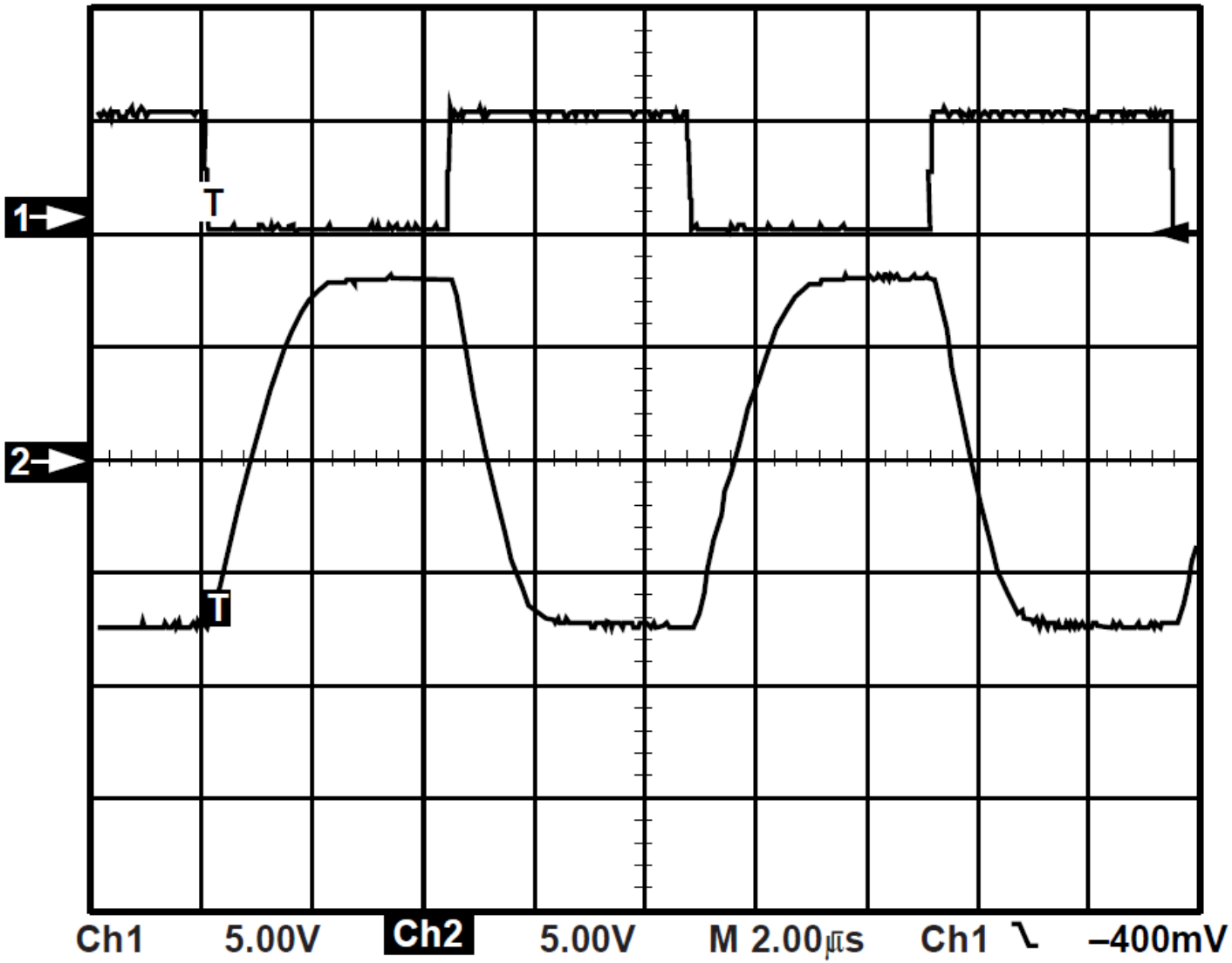
UART ▸ Fizikai rétegek ▸ RS232, full duplex



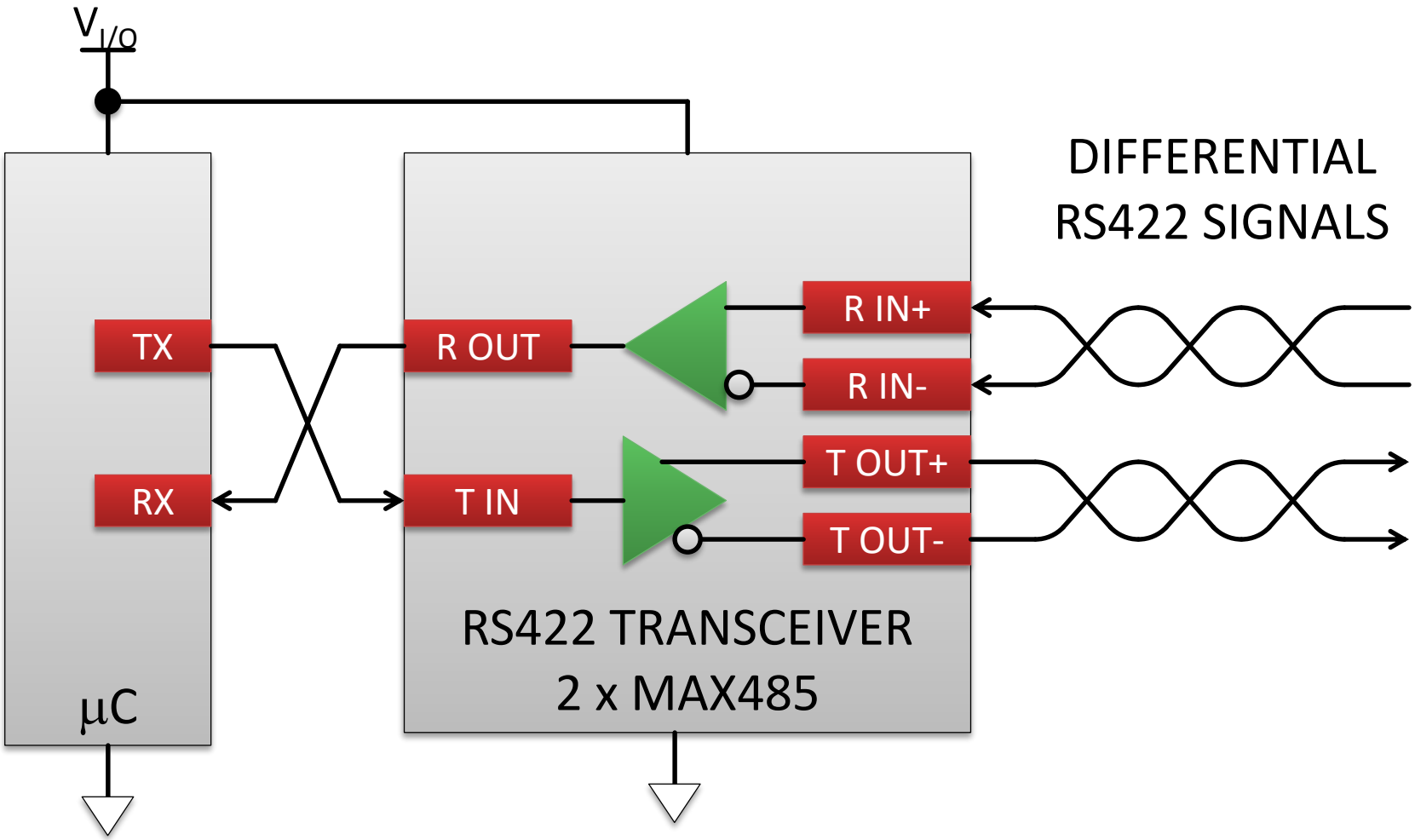
UART ▸ Fizikai rétegek ▸ RS232



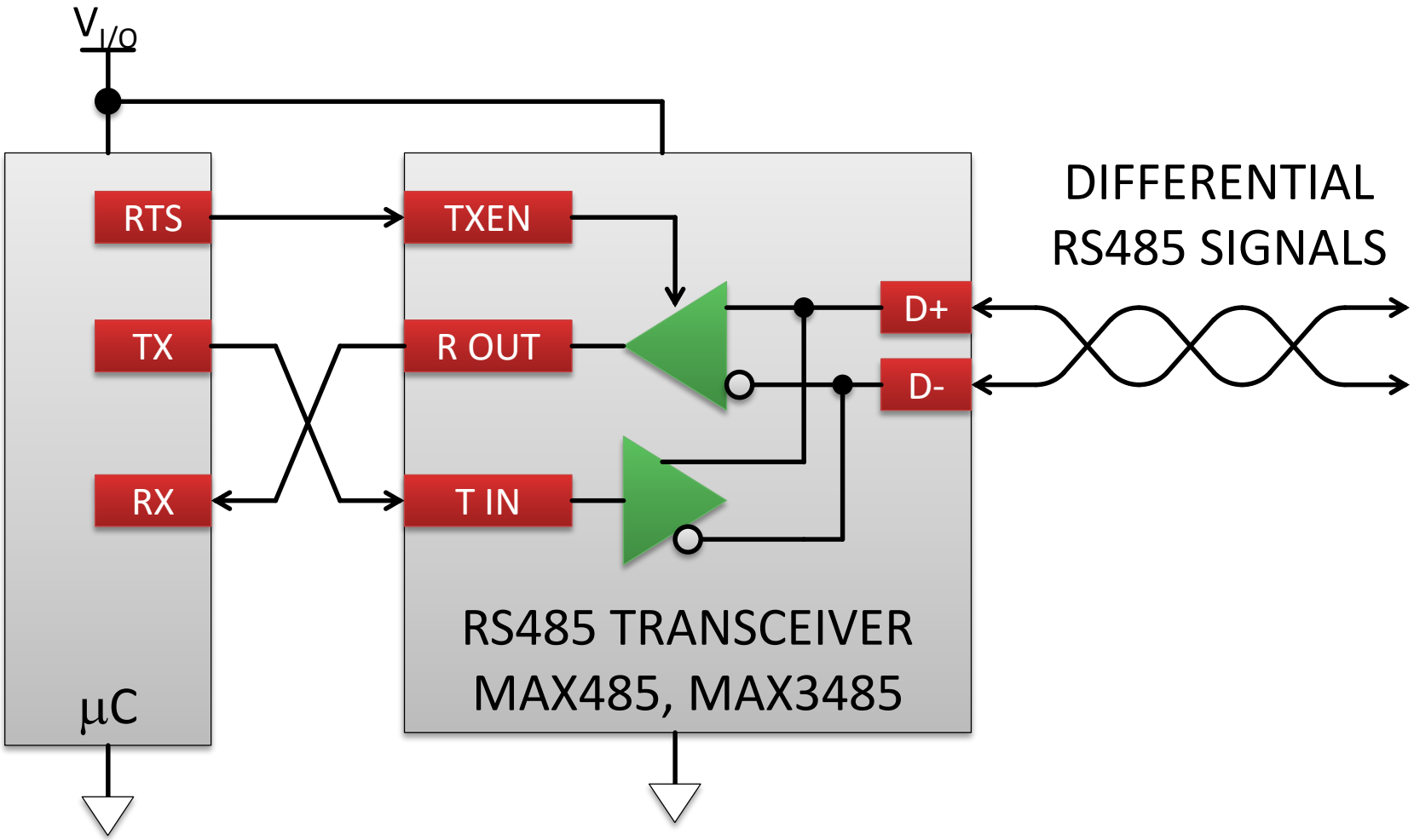
UART ▸ Fizikai rétegek ▸ RS232



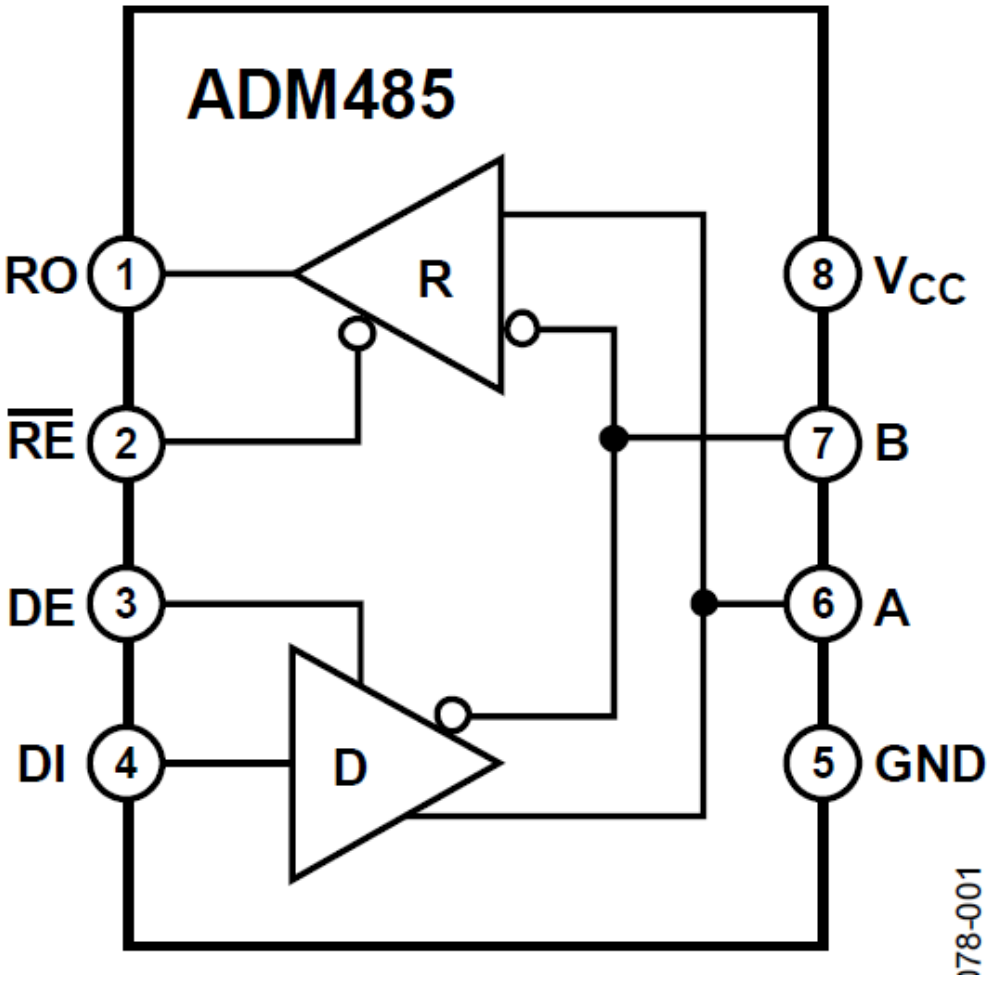
UART ▸ Fizikai rétegek ▸ RS422, full duplex



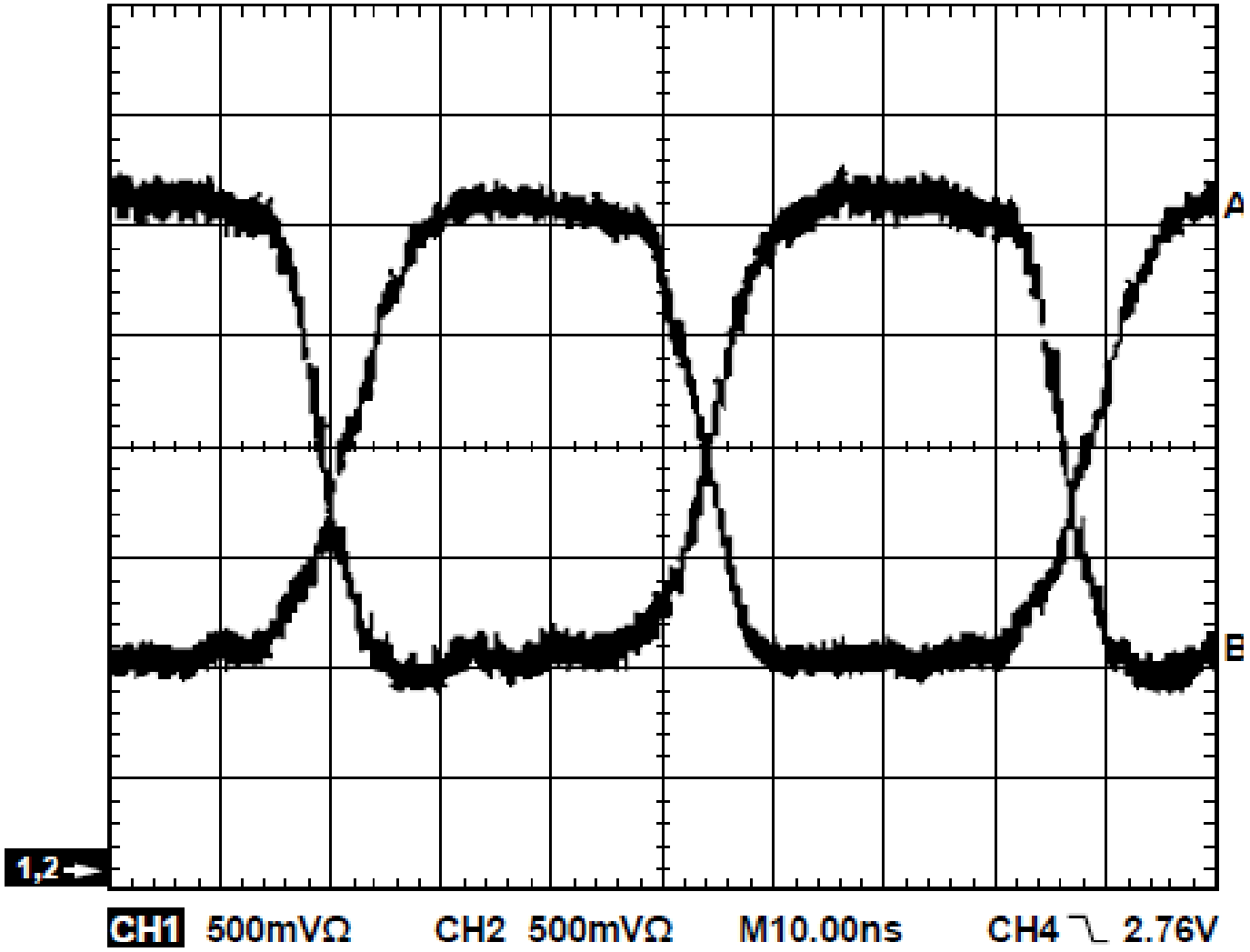
UART ▸ Fizikai rétegek ▸ RS485, half duplex



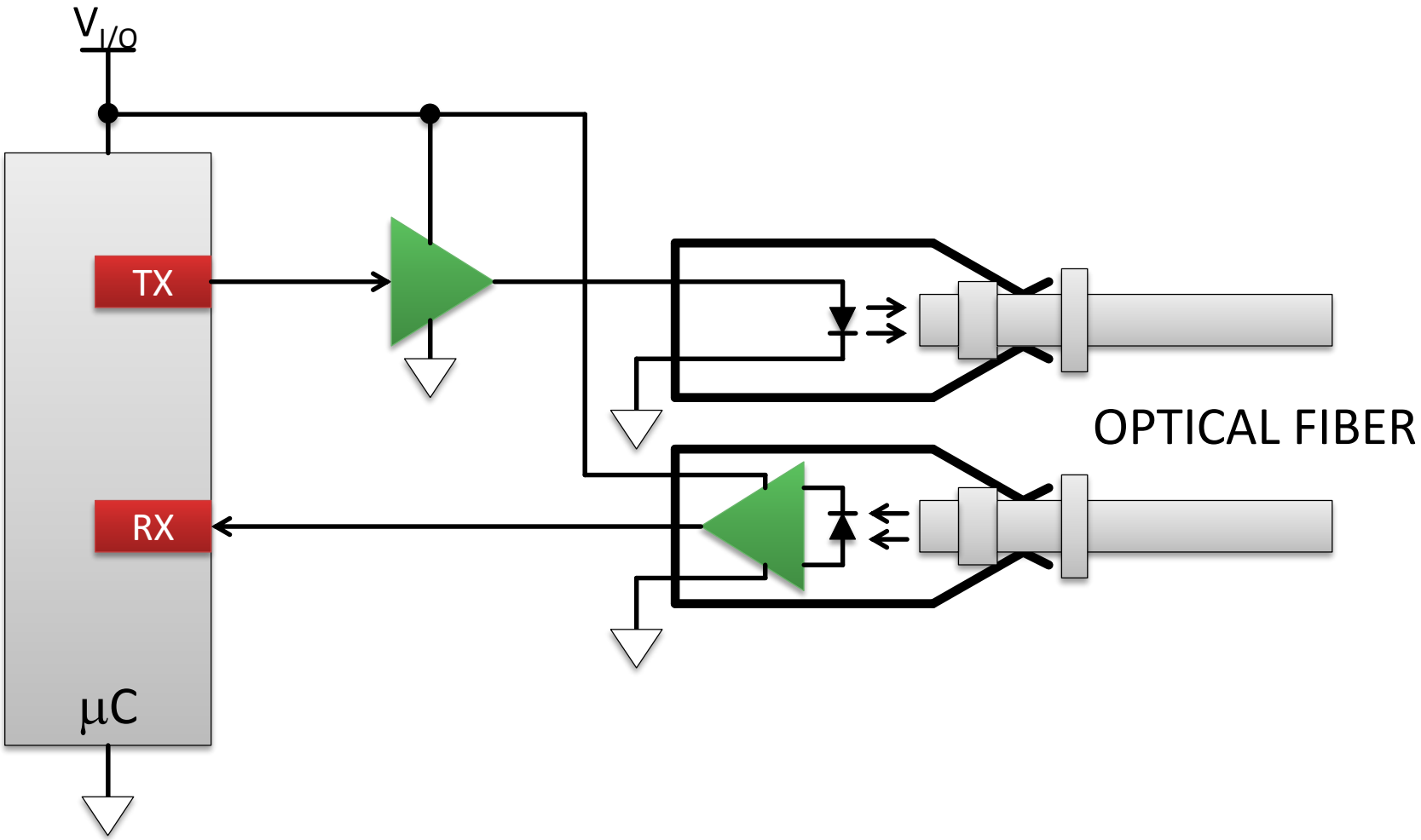
UART ▸ Fizikai rétegek ▸ RS485



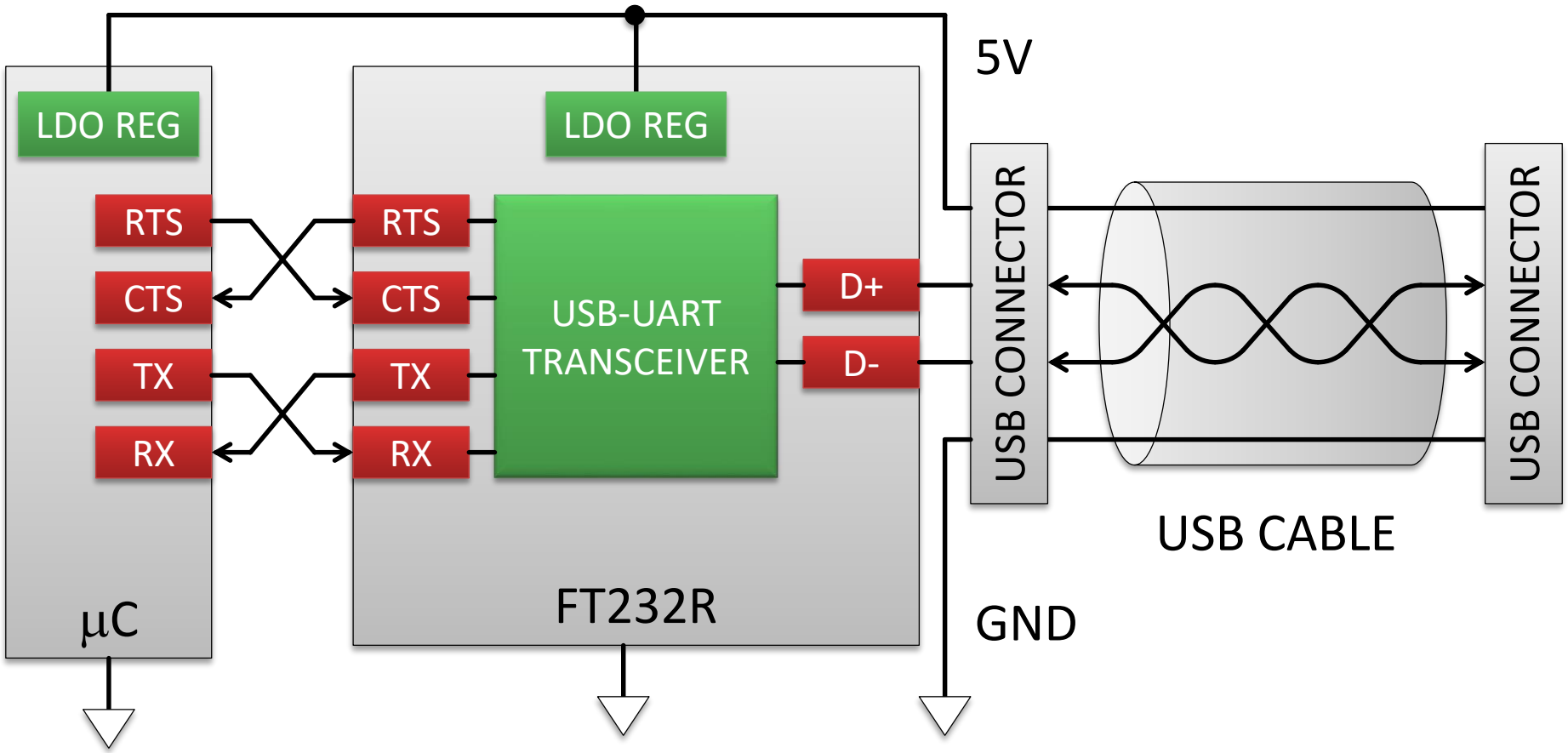
UART ▸ Fizikai rétegek ▸ RS485, half duplex



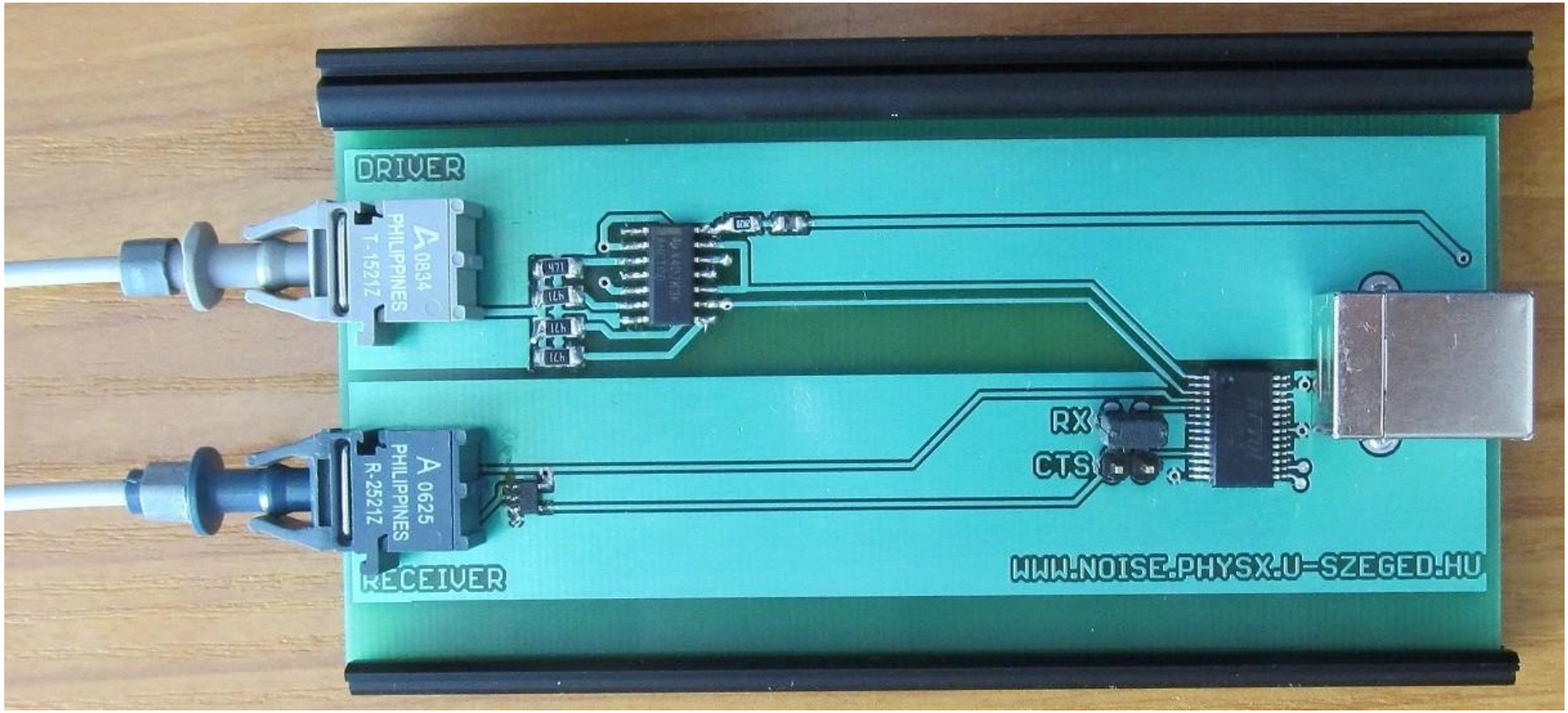
UART ▸ Fizikai rétegek ▸ Optikai, full duplex



UART ▸ Fizikai rétegek ▸ USB-UART



UART ▸ Fizikai rétegek ▸ USB-Optikai UART



UART ▸ UART-Bluetooth

- Bluetooth modulok UART interfésszel
- SPP (Serial Port Profile) standard protokoll
- pl. BTM112

UART ▸ Mire figyeljünk

- Timer beállítás, engedélyezés
- A timer másra nem használható
- Ne keverjük: polling és interrupt mód
- TX legyen push-pull → rövid jelváltási idők
- Baud rate hibájának figyelembe vétele

Port I/O ▸ C stdio

- stdio átirányítás? printf?

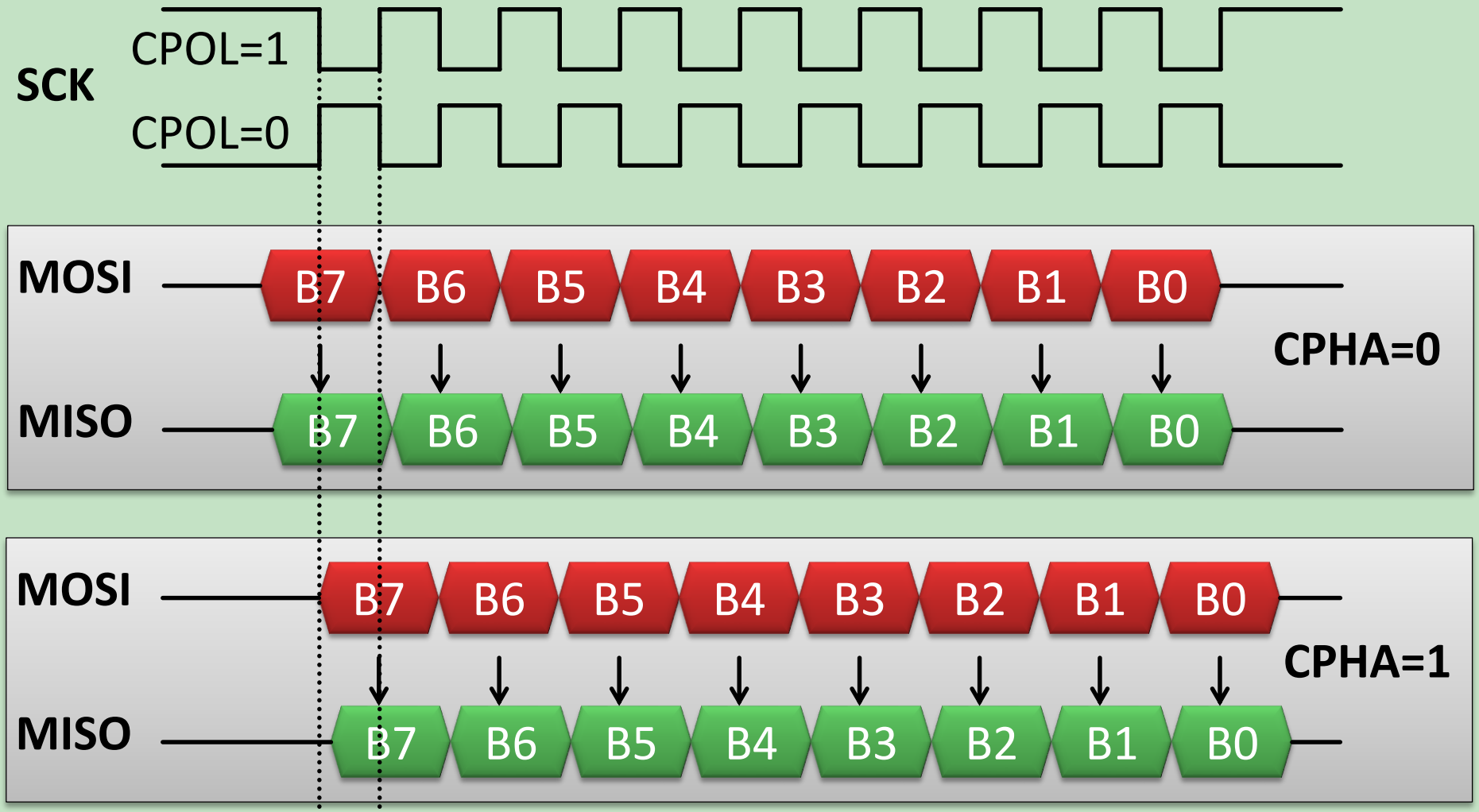
```
void putchar(char c)
{
  UARTOut(c);          // UART, SPI, SMBus, stb.
}

char getchar(void)
{
  return UARTIn(c); // UART, SPI, SMBus, stb.
}

…
printf("x=%d",x);    // write
…
```

SPI áramkörök

SPI: Serial Peripheral Interface

- Processzorok, integrált áramkörök között
- Szinkron soros adatátvitel
- Órajel váltásaikor érvényes a bit
- Master és slave eszközök
  - Az órajelet adó a master
  - Az órajel garantálja az azonos időzítést a két oldalon
- Full duplex
  - kimenet: MOSI (master out/slave in)
  - bemenet: MISO (master in/slave out)
  - órajel: SCK (serial clock)
  - Engedélyező vonal (NSS, negated slave select)

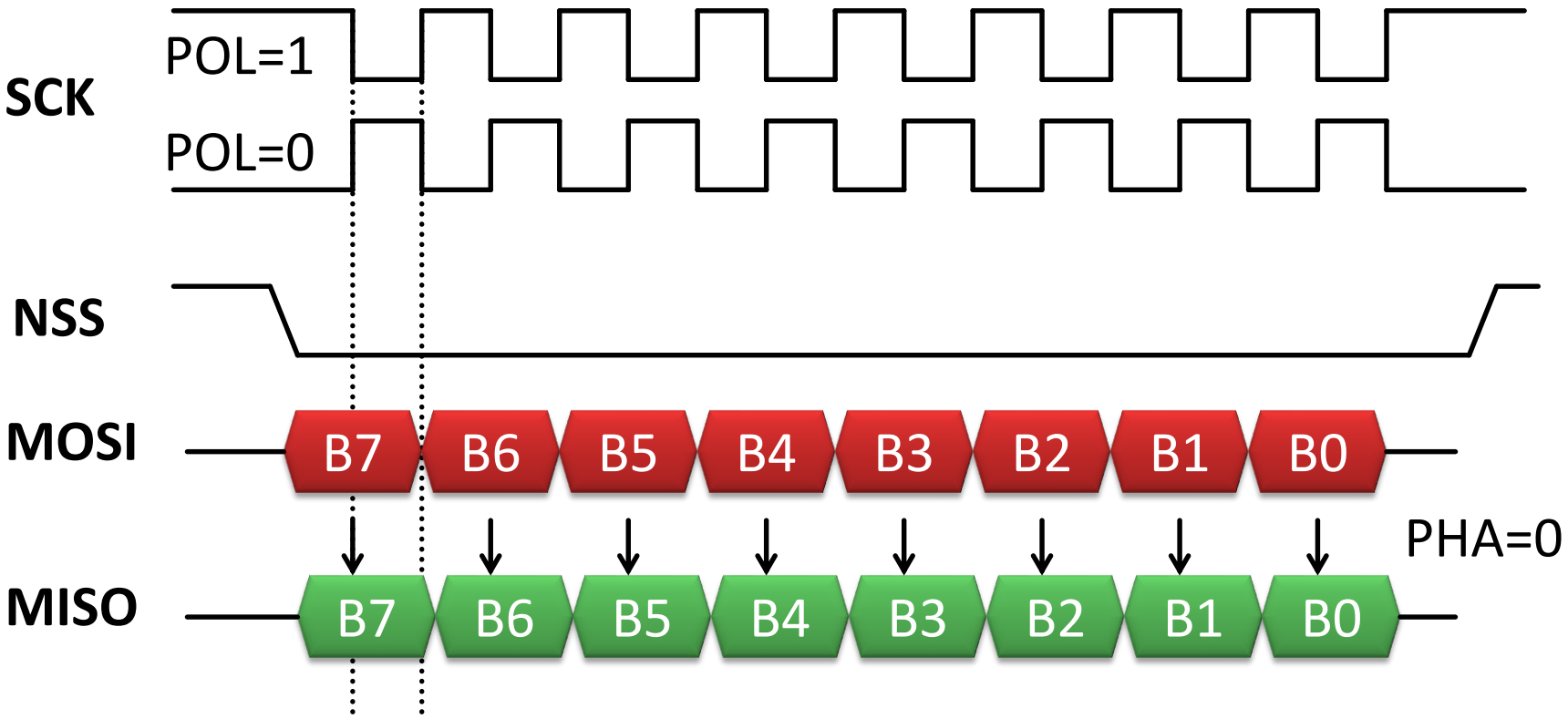
SPI ▸ Idődiagram, írás-olvasás ▸ master



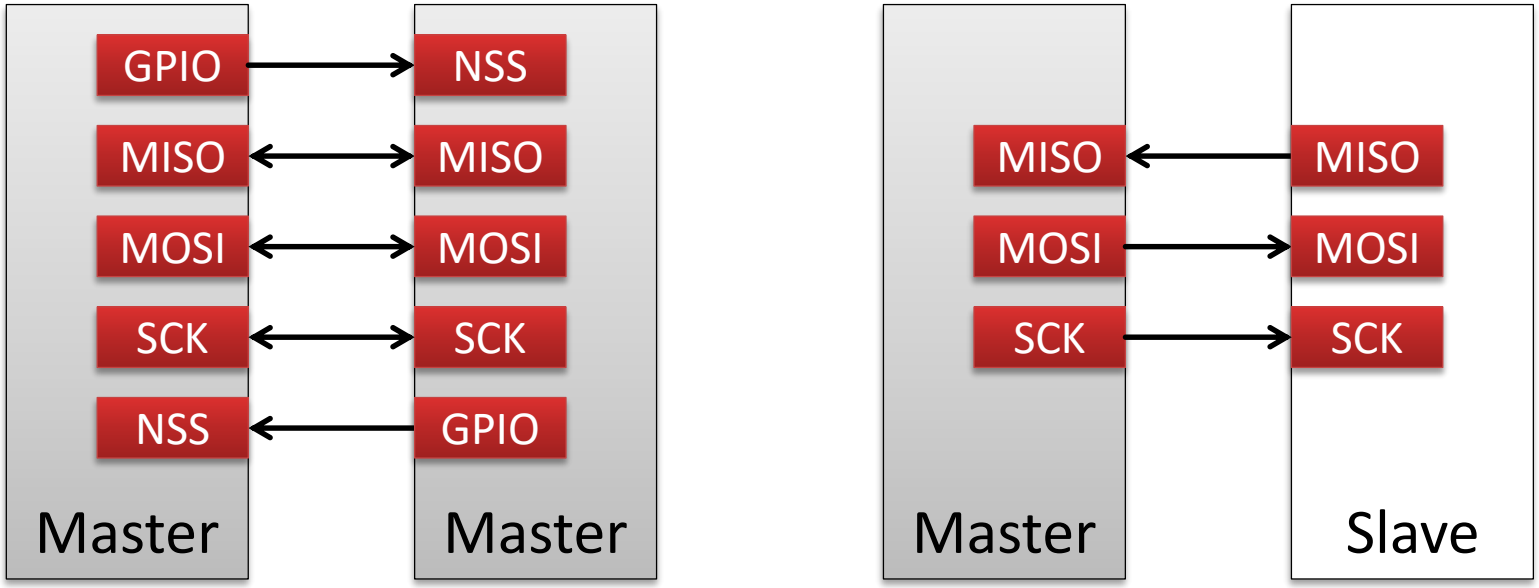
SPI ▸ Írás, olvasás

- Az adó és vevő oldal azonos módban legyen (POL, PHA)
- Az egyik SCK él a kimenő bitet beállítja, a másik a bejövő bitet olvassa
- Slave:
  - Mikor kezdődik az adat átvitele?
  - Csak az órajel indulása nem feltétlen elég
  - Kieshet a szinkronból az átvitel
  - Megoldás: Slave select (NSS) jel

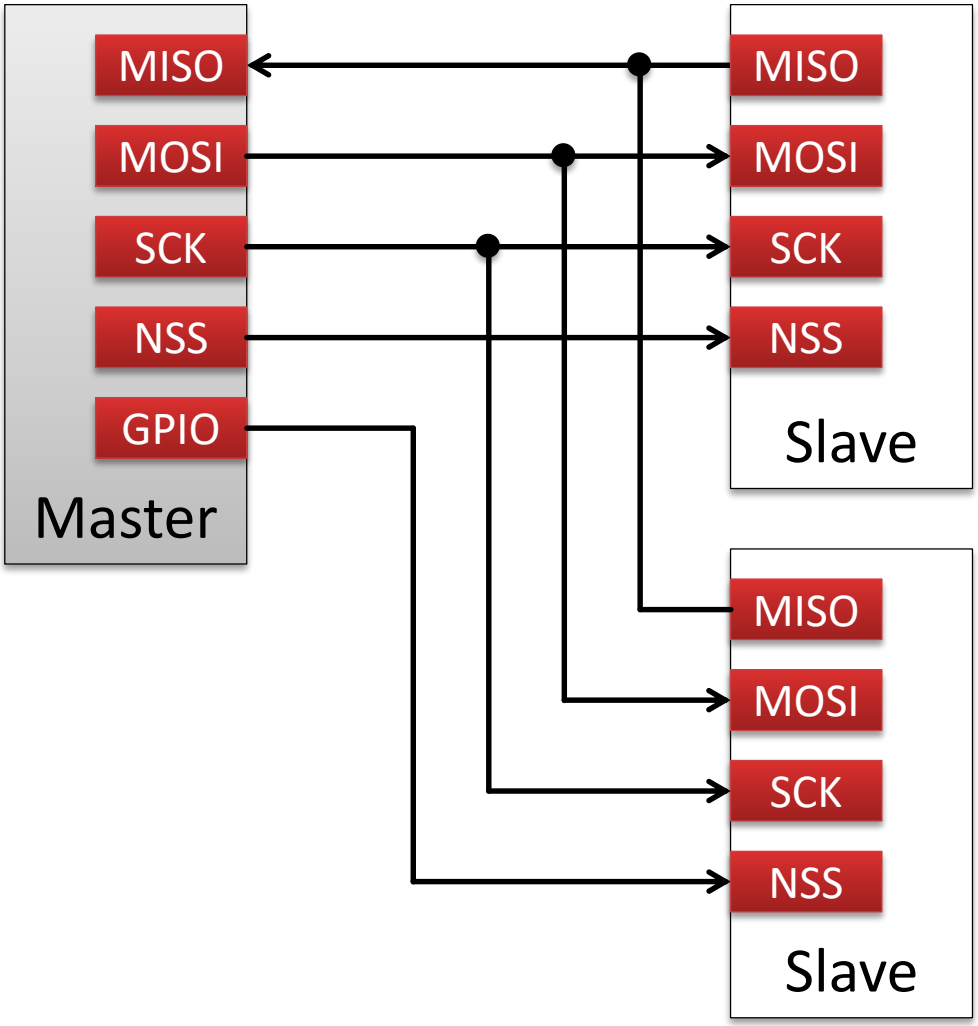
SPI ▸ Idődiagram, írás-olvasás ▸ slave



SPI ▸ Multimaster, Master-slave 3 wire



SPI ▸ Master-slave 4 wire



- Több slave
- Egyszerre egy aktív
- Slave NSS választ
- NSS adatkezdet is

SPI ▸ Példák

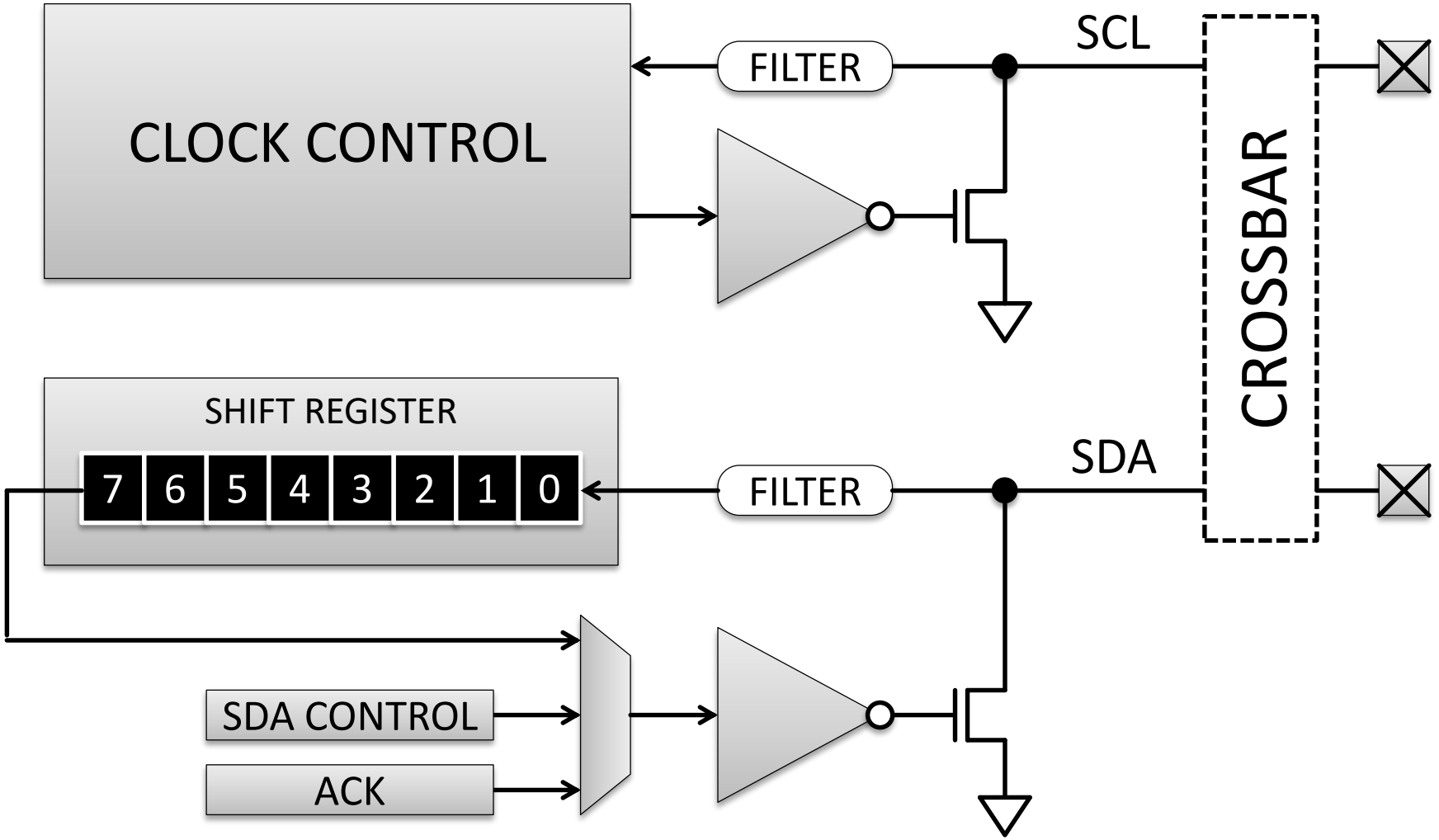
```
void SPIOut(unsigned char c)
{
  SELECT  = 0;
  SPIF    = 0;
  SPI0DAT = c;
  while (!SPIF);
  SELECT  = 1;
}

unsigned char SPIIn(void)
{
  SELECT  = 0;
  SPIF    = 0;
  SPI0DAT = 0;     // dummy write starts SPI clock
  while (!SPIF);
  SELECT  = 1;
  return SPI0DAT;
}
```

SMBus/I2C áramkörök

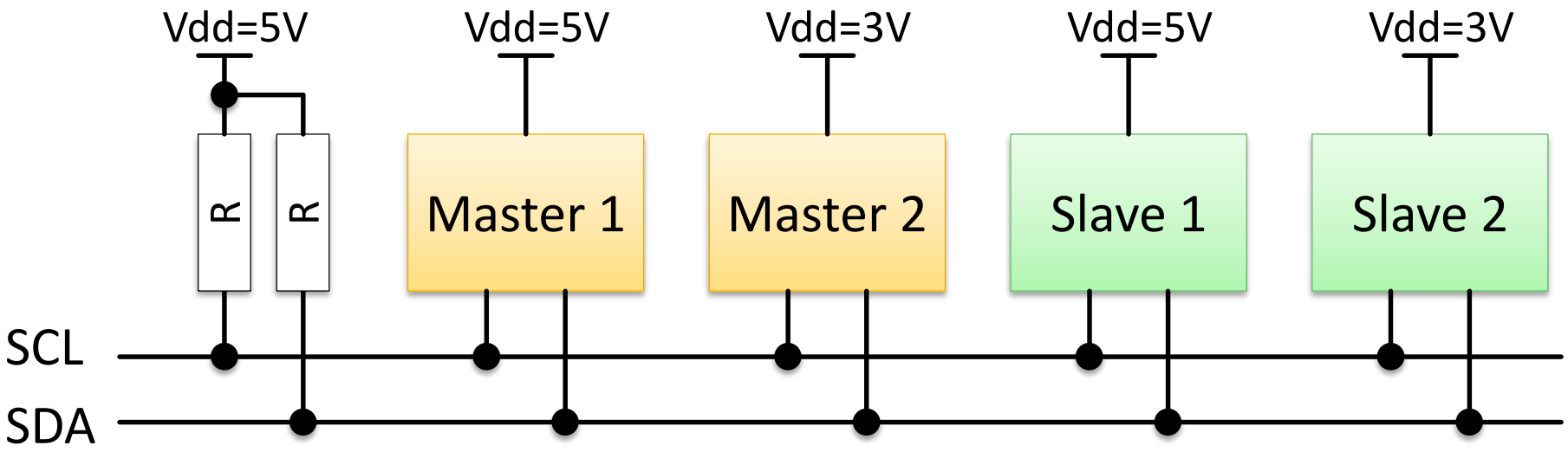
SMBus ▸ Alapok

- System Management Bus
- IIC vagy I^2C: Inter-Integrated Circuit
- Áramkörök közötti egyszerű, **kétvezetékes busz**
  - ***SCL:*** *Órajel* (100kHz vagy 400kHz)
  - ***SDA:*** *Adat*
- Két irány (half duplex)
- Master, slave, multimaster
- Párhuzamosan kapcsolhatók eszközök
- 7-bites címzés
- Tipikusan: szenzorok, A/D, RTC, EEPROM, stb.

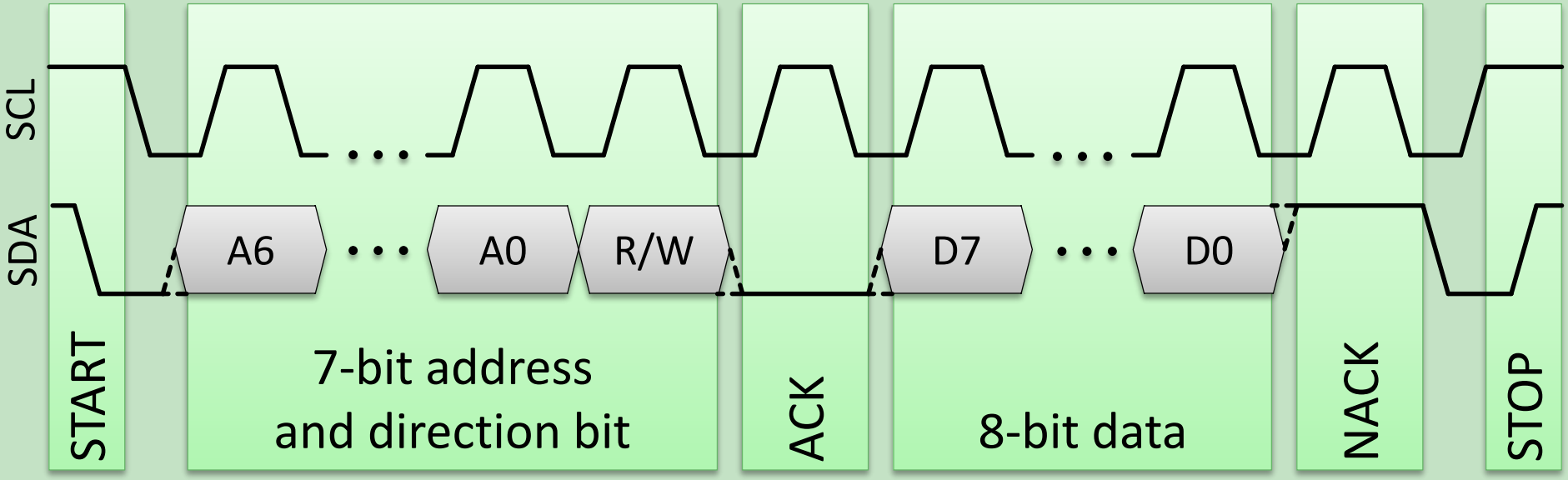
SMBus ▸ Hardverfelépítés – open-drain I/O



SMBus ▸ Buszrendszer

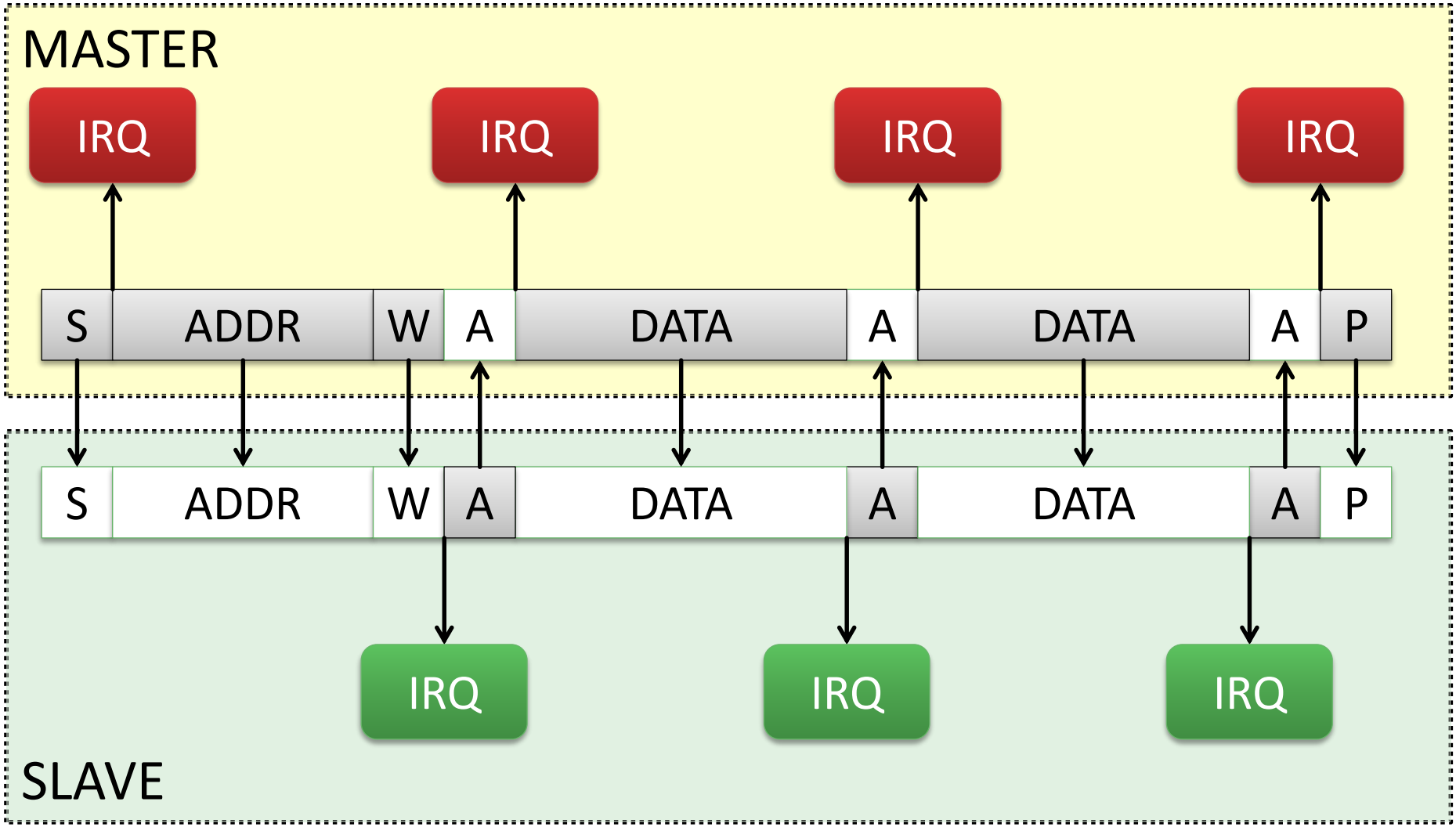


SMBus ▸ Tipikus tranzakció

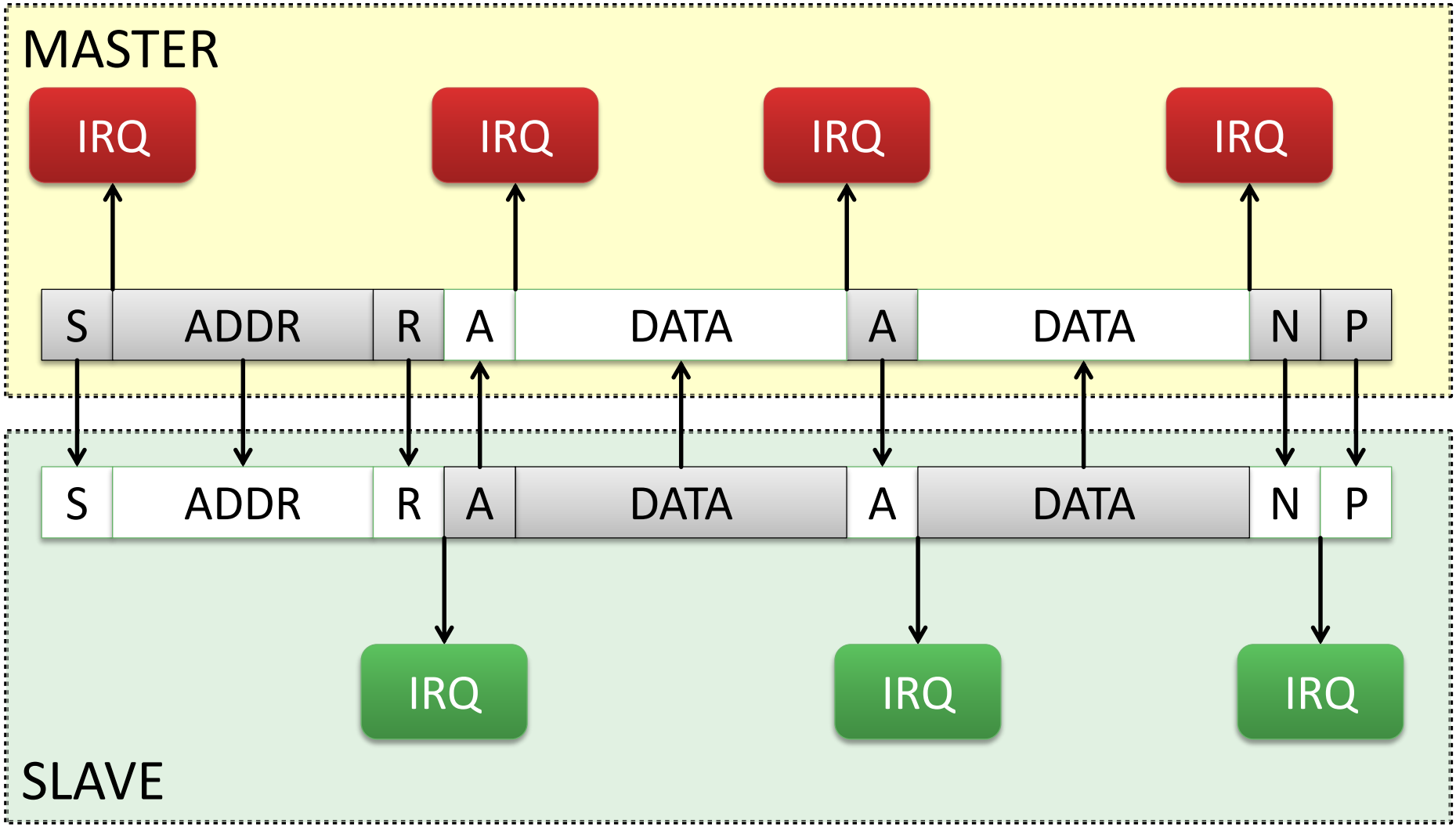


- Bármelyik vezetéket az adó és vevő is 0-ba tudja húzni
- SCK: csak a masterek húzhatják 0-ba
- Több byte is küldhető egy tranzakcióban
- ACK minden byte vételekor szükséges a vevőtől
- NACK: nem nyugtáz a vevő, vagy utolsó byte (master)
- STOP: a tranzakció befejezése

SMBus ▸ Master (küld) → Slave (fogad)



SMBus ▸ Master (fogad) ← Slave (küld)



SMBus ▸ Mechanizmus

- **Master SFR-bitek**
  - **STA**, **STO** írása a jelek generáláshoz
  - **ACK** írása és olvasása is
  - **SI** flag (megszakítás is) 1-re vált, ha egy fázis kész
  - **A busz áll, amíg SI=1, folytatódik, ha töröljük**
  - Ezért mindig **SI**=0 előtt kell írni az SFR regisztereket
- **Slave SFR-bitek**
  - **STA**, **STO** olvasása
  - **ACK** írása és olvasása is

SMBus ▸ Példák

```
void SMBusOut(unsigned char address, unsigned char c)
{
  STO = 0;
  STA = 1;                      // start transfer
  SI = 0;                       // continue
  while (!SI);                  // wait for start complete
  STA = 0;                      // manually clear STA
  SMB0DAT = address << 1;       // A6..A0 + write
  SI = 0;                       // continue
  while (!SI);                  // wait for complete
  if (!ACK)                     // not acknowledged, stop
  {
    STO = 1;                    // stop condition bit
    SI = 0;                     // generate stop condition
    return;
  }
  SMB0DAT = c;                  // put data into shift register
  SI = 0;                       // continue
  while (!SI);                  // wait for complete
  STO = 1;                      // stop condition bit
  SI = 0;                       // generate stop condition
}
```

SMBus ▸ Példák

```
unsigned char SMBusIn(unsigned char address)
{
  STO = 0;
  STA = 1;                      // start transfer
  while (!SI);                  // wait for start complete
  STA = 0;                      // manually clear STA
  SMB0DAT = (address << 1) | 1;  // A6..A0 + read
  SI = 0;                       // continue
  while (!SI);                  // wait for complete
  if (!ACK)                     // not acknowledged, stop
  {
    STO = 1;                    // stop condition bit
    SI = 0;                     // generate stop condition
    return;
  }
  ACK = 0;                      // NACK, last byte
  SI = 0;                       // continue
  while (!SI);                  // wait for complete
  STO = 1;                      // stop condition bit
  SI = 0;                       // generate stop condition
  return SMB0DAT;
}
```

További kommunikációs áramkörök

További kommunikációs áramkörök

- CAN: Controller Area Network
- LIN: Local Interconnect Network
- USB: Universal Serial Bus
- Vezeték nélküli kommunikációs áramkörök