Universitatea Tehnică din Cluj-Napoca
Departamentul Calculatoare

Proiectarea cu Micro-Procesoare

Lector: Mihai Negru

An 3 – Calculatoare și Tehnologia Informației

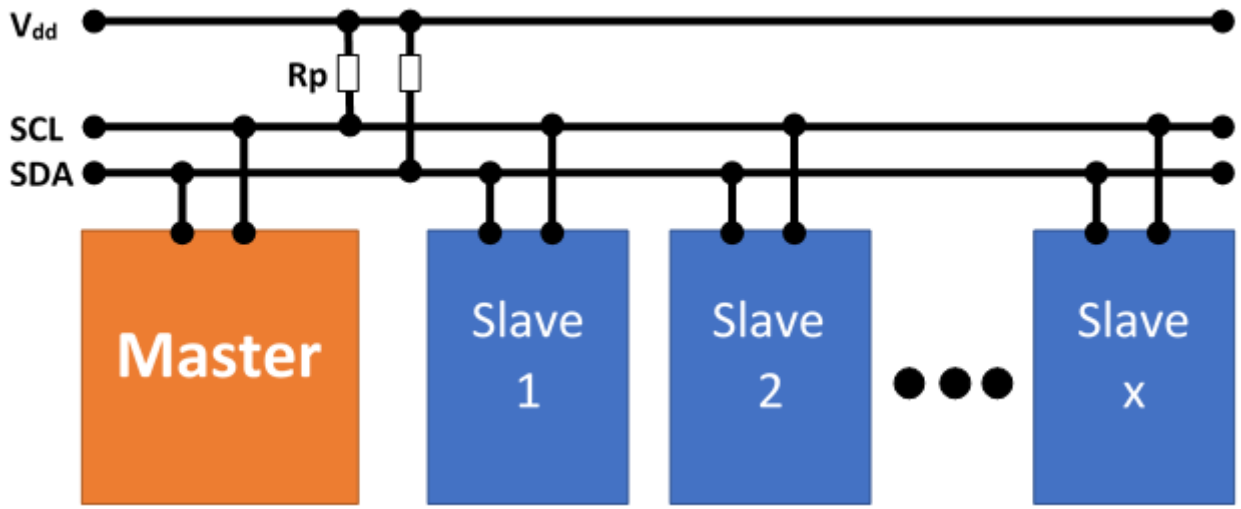
Seria B

Curs 7: Comunicare Serială 2

http://users.utcluj.ro/~negrum/

I²C (Inter-Integrated Circuit)

- **Structura**
  - *SDA, SCL* – date și clock, bidirecționale, conectate “open collector” sau “open drain”
  - Rezistente “pull up” țin linia la Vcc
  - O linie poate fi trasă la ‘0’ de orice dispozitiv – ea va deveni ‘0’ dacă cel puțin un dispozitiv scrie ‘0’, altfel e ‘1’
  - Fiecare dispozitiv are o adresă de 7 biți
  - Exista 16 adrese rezervate → 112 adrese disponibile → max 112 dispozitive



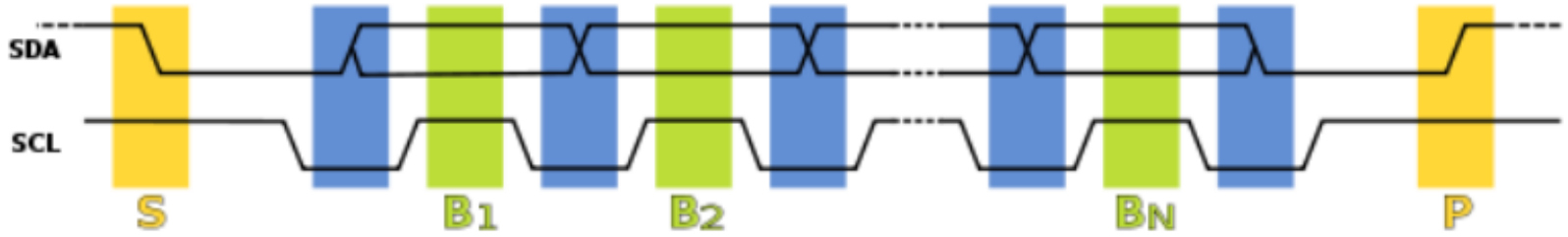
I²C (Inter-Integrated Circuit)

- **Tipuri de dispozitive (noduri)**
  - *Master – generează semnalul de ceas, și adresele*
  - *Slave – primește semnal de ceas, și adresă*
  - Rolul de master și slave se poate schimba pentru același dispozitiv

- **Moduri de operare**
  - *master transmit* — nodul master trimite date la slave
  - *master receive* — nodul master recepționează date de la slave
  - *slave transmit* — nodul slave trimite date la master
  - *slave receive* — nodul slave primește date de la master

I²C (Inter-Integrated Circuit)

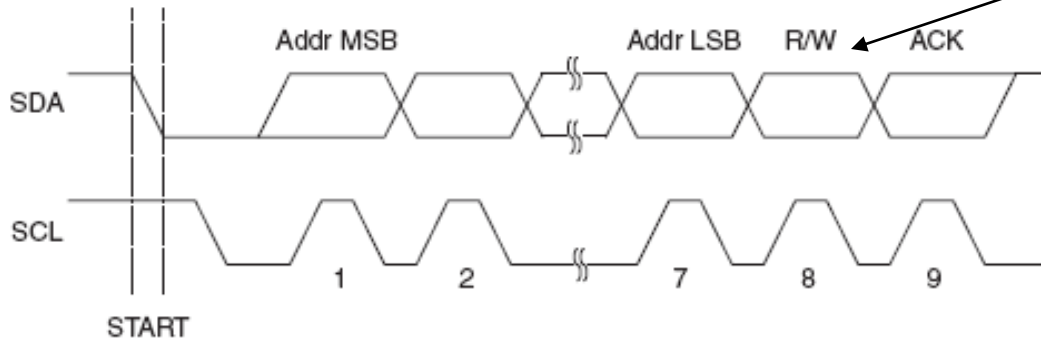
- **Diagrame de timp**



- **Moduri de operare**
  - *Start:* o tranzitie a SDA din ‘1’ in ‘0’, cu SCL menținut pe ‘1’
  - *Transfer biți:* valoarea bitului în SDA se schimba cand SCL e ‘0’, și este menținută stabilă pentru preluare când SCL e ‘1’
  - *Stop*: o tranziție a SDA din ‘0’ in ‘1’ când SCL e ‘1’

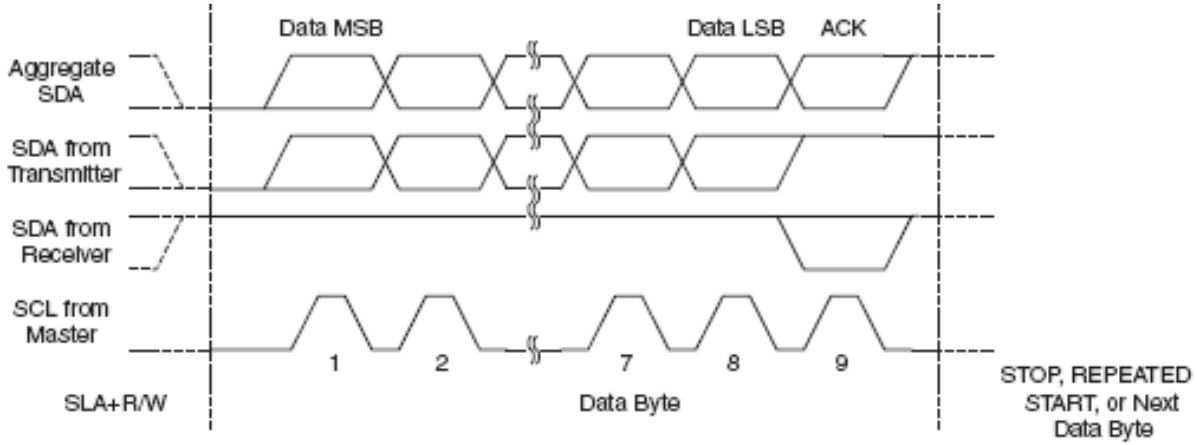
I²C (Inter-Integrated Circuit)

- **Formatul pachetelor – adresa**



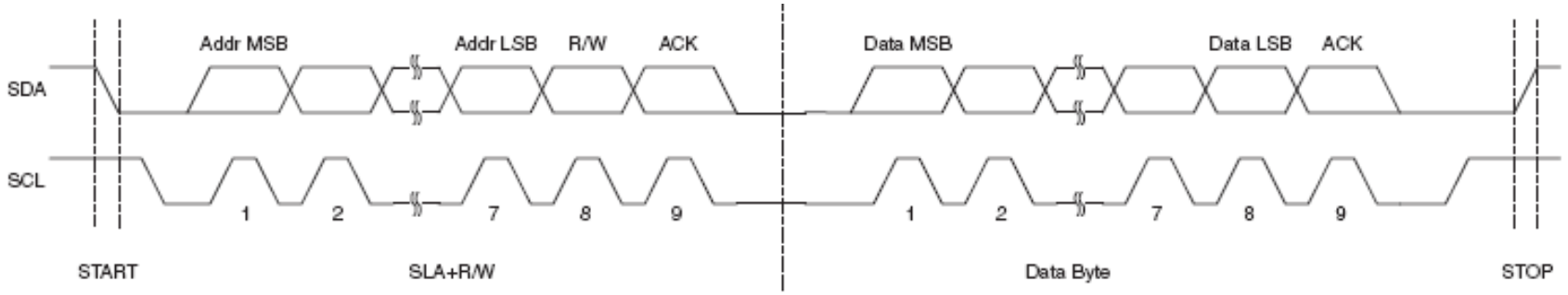
Specifică dacă se dorește citire sau scriere

- **Formatul pachetelor – date**



I²C (Inter-Integrated Circuit)

- **Transmisie tipică**



- **Arbitrare**
  - Fiecare master monitorizează semnalele de START și STOP, și nu inițiază un mesaj cât timp alt master ține bus-ul ocupat
  - Dacă doi master inițiază transmisie în același timp, se produce arbitrarea, pe baza liniei SDA
    - Fiecare master verifică linia SDA, și dacă nivelul acesteia nu este cel așteptat (cel scris), acel master pierde arbitrarea
    - Primul master care pune pe linie ‘1’ când altul pune ‘0’ pierde
    - Master-ul care pierde asteaptă un semnal de stop, apoi încearcă din nou

I²C (Inter-Integrated Circuit)

- **Reglarea vitezei de transmisie**
  - "Clock stretching"
  - Un dispozitiv slave poate ține linia de ceas (SCL) la '0' mai mult timp, indicând necesitatea unui timp mai lung pentru procesarea datelor
  - Dispozitivul master va incerca să pună linia SCL pe '1', dar va eșua din cauza configurației 'open collector'
  - Dispozitivul master va verifica dacă linia SCL a fost pusă pe '1' și va continua transmisia cand acest lucru se intamplă

  - Nu toate dispozitivele suportă "clock stretching"
  - Există limite pentru intervalul de stretching

I²C (Inter-Integrated Circuit)

- **Aplicații**
  - Citirea datelor de configurare din SPD EEPROM-ul din SDRAM, DDR SDRAM, DDR2 SDRAM
  - Interfațare senzori digitali
  - Accesarea convertoarelor DAC și ADC.
  - Schimbarea setărilor în monitoare video (Display Data Channel).
  - Schimbarea volumului în boxele inteligente.
  - Citirea stării senzorilor de diagnostic hardware, precum termostatul CPU și viteza ventilatorului.

  - Posibilitatea ca un microcontroller să controleze multiple dispozitive atașate prin doar două fire.
  - Posibilitatea ca dispozitive să fie atașate sau eliminate de pe bus în timpul funcționarii

Comunicare I2C la Arduino

- Se utilizează biblioteca **Wire**, din pachetul software Arduino
- O placa Arduino poate fi I2C master sau I2C slave
- Metodele obiectului Wire:
  - **Wire.begin(address)** – activează interfața I2C. Parametrul address indică adresa de 7 biți cu care Arduino se atasează la bus-ul I2C ca slave. Daca funcția este apelată fară parametri, dispozitivul este master.
  - **Wire.beginTransmission(address)** – începe procesul de transmisie dinspre master către un slave specificat de **address**. După apelul acestei funcții, datele pot fi scrise cu write().
  - **Wire.endTransmission()** – apelat de master pentru a realiza efectiv transmisia începută cu beginTransmission, cu datele pregătite prin apelul **Wire.write()**.

Comunicare I2C la Arduino

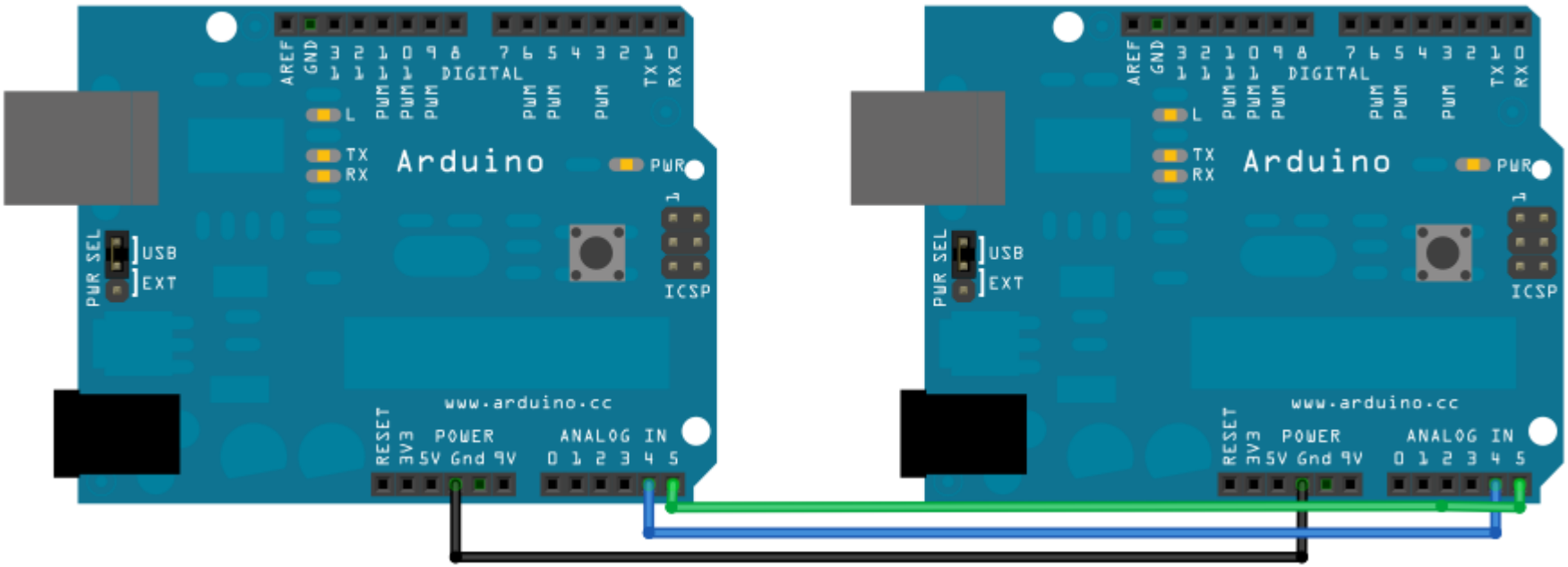
- **Wire.write(value)** – scrie un byte. Funcția poate fi apelată de slave, dacă a fost solicitat de către master, sau de master după ce a apelat beginTransmission. Alternative: Wire.write(string), sau Wire.write(data, length).
- **Wire.requestFrom(address, quantity)** – master cere o cantitate (**quantity)** de date de la un slave identificat prin **address**. Funcția poate fi apelată și ca **Wire.requestFrom(address, quantity, stop)**, stop fiind o valoare booleană specificând dacă masterul va elibera bus-ul, sau va menține conexiunea activă.

- **Wire.available()** – returnează numărul de bytes disponibili pentru a fi citiți.
- **Wire.read()** – citește un byte, dacă available() > 0. Apelabilă și de master, și de slave.

- **Wire.onReceive(handler)** – configurează o **funcție handler**, la dispozitivul slave, care va fi apelată automat la primirea datelor de la master.
- **Wire.onRequest(handler)** – configurează o funcție handler, la dispozitivul slave, care va fi apelată automat atunci când masterul cere date.

Comunicare I2C la Arduino

- **Exemplu:** conectarea a două plăci Arduino prin I2C, una având rol de master, care va transmite datele, și una de slave, care le va recepționa.
- Sursa: http://arduino.cc/en/Tutorial/MasterWriter



Comunicare I2C la Arduino

- **Exemplu:** conectarea a două plăci Arduino prin I2C, una având rol de master, care va transmite datele, și una de slave, care le va recepționa.

Cod master:

```
#include <Wire.h>
void setup() {
  Wire.begin(); // activeaza interfata I2C ca master
}

byte x = 0; // valoarea de transmis, se va incrementa

void loop() {
  Wire.beginTransmission(4);  // incepe un proces de transmisie, catre adresa slave 4
  Wire.write("x is ");        // scriere string
  Wire.write(x);              // scriere un byte
  Wire.endTransmission();    // finalizeaza tranzactia de scriere
  x++; // incrementare valoare
  delay(500);
}
```

Comunicare I2C la Arduino

- **Exemplu:** conectarea a două plăci Arduino prin I2C, una având rol de master, care va transmite datele, și una de slave, care le va recepționa.

Cod slave:

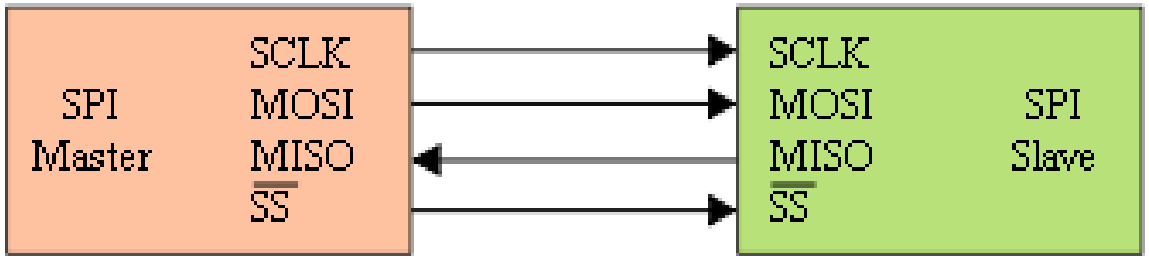
```
#include <Wire.h>
void setup() {
  Wire.begin(4);                 // activeaza interfata i2c ca slave, cu adresa 4
  Wire.onReceive(receiveEvent);  // inregistreaza functia receiveEvent pentru a fi apelata la venirea datelor
  Serial.begin(9600);            // activeaza interfata seriala, pentru a afisa pe PC datele primite
}

void loop()  { // functia loop nu face nimic
  delay(100);
}

void receiveEvent(int howMany) // functie apelata cand slave primeste date
{
  while(1 < Wire.available()) // parcurge cantitatea de date, lasand ultimul octet separat
  {
    char c = Wire.read(); // citeste caracter cu caracter
    Serial.print(c);         // trimite caracterul la PC
  }
  int x = Wire.read();    // ultimul caracter este tratat ca o valoare numerica
  Serial.println(x);         // si va fi tiparit ca atare
}
```

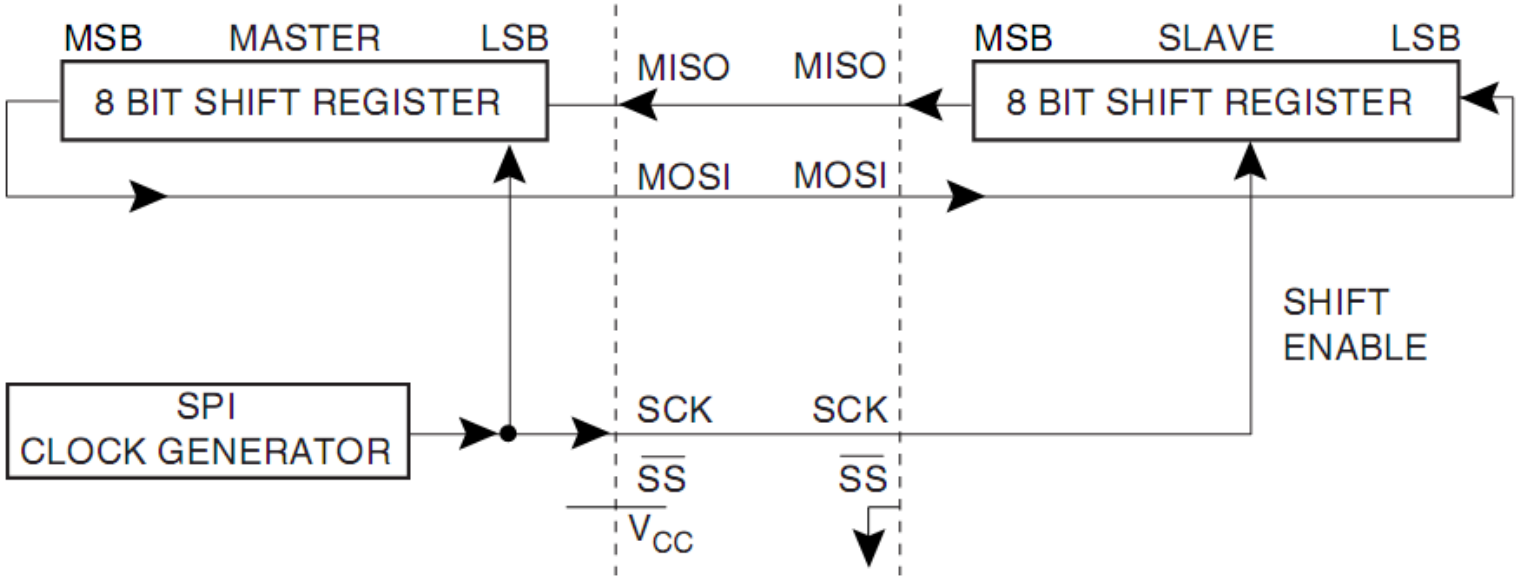
Serial Peripheral Interface (SPI)

- **Semnale**
  - SCLK – Serial clock, generat de Master
  - MOSI – Master Output Slave Input, date transmise de Master
  - MISO – Master Input Slave Output, date receptionate de Master
  - SS – Slave Select, activarea dispozitivului Slave de catre Master, activ pe zero



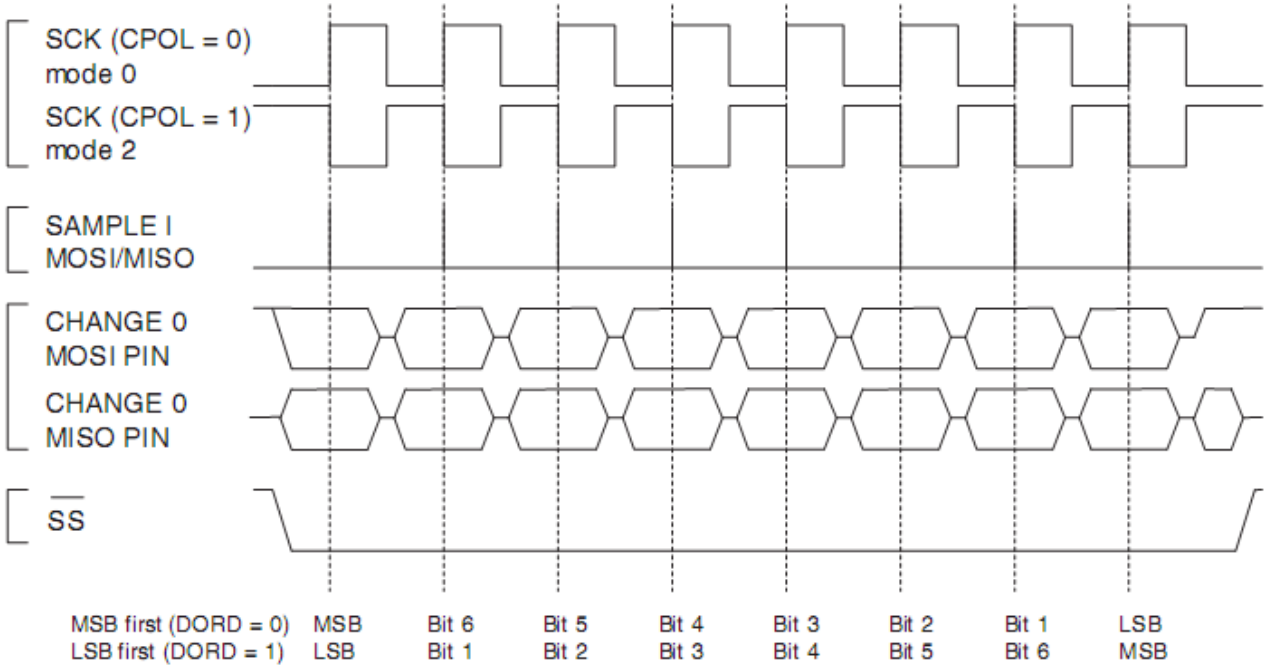
- **Funcționare**
  - Master-ul inițiază comunicația prin activarea SS (SS → 0)
  - Master generează semnalul de ceas SCLK
  - La fiecare perioadă de ceas un bit se transmite de la master la slave, și un bit de la slave la master – sincron
  - După fiecare pachet de date (8, 16 biți,…) SS este dezactivat (SS → 1), pentru sincronizarea transmisiei

- **Principiul de functionare**
  - Ambii parteneri au câte un registru de deplasare intern, ieșirile și intrările fiind conectate prin MISO/MOSI
  - Ambii registri au același ceas, SCLK
  - Cei doi regiștri formează împreună un registru de rotație
  - După un număr de perioade de ceas egal cu dimensiunea unui registru, Master și Slave fac schimb de date

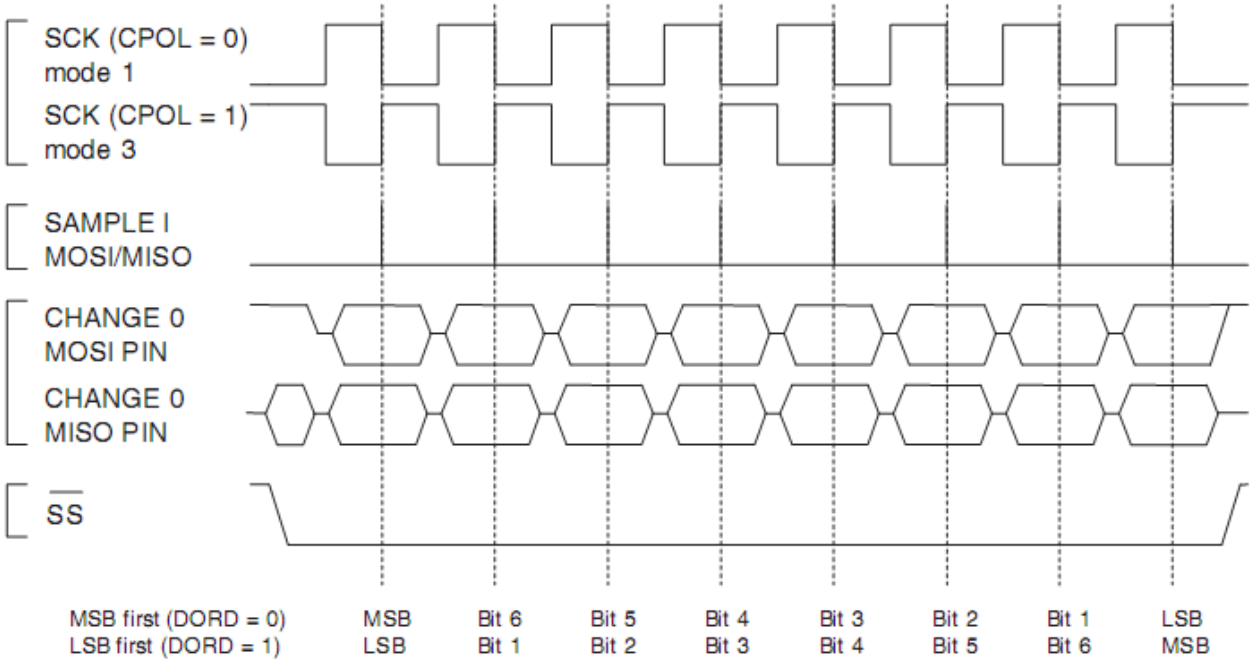


Serial Peripheral Interface (SPI)

- **Sincronizarea datelor cu semnalul de ceas**
  - Deplasarea (shiftare) datelor și preluarea lor se fac pe fronturi opuse
  - CPOL – clock polarity: primul front e crescător sau descrescător
  - CPHA – clock phase
  - Pentru CPHA = 0
    - Pe primul front se face preluarea datelor
    - Pe al doilea front se face stabilizarea (deplasarea)

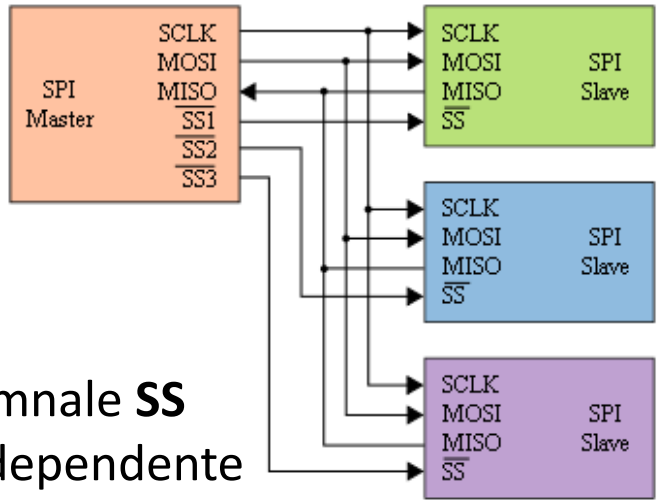


- **Sincronizarea datelor cu semnalul de ceas**
  - Deplasarea (shiftare) datelor și preluarea lor se fac pe fronturi opuse
  - CPOL – clock polarity: primul front e crescător sau descrescător
  - CPHA – clock phase
  - Pentru CPHA = 1
    - Pe primul front se face deplasarea
    - Pe al doilea front se face preluarea datelor

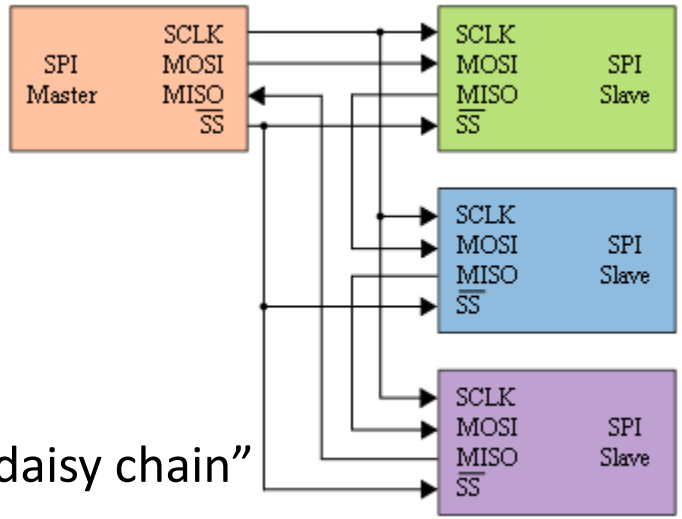


Serial Peripheral Interface (SPI)

- **Utilizarea semnalului SS**
  - Pentru un dispozitiv “slave” **SS** este semnal de intrare
    - SS = 0 → activarea dispozitivului slave. O tranziție din 0 in 1 → marchează sfârsitul unui pachet (resetarea ciclului de transfer)
    - SS = 1 → dispozitiv slave inactiv
  - Pentru un dispozivit “master” **SS** poate fi:
    - Ieșire – prin el se activează dispozitivul “slave” pentru comunicare
    - Intrare – daca se permit mai multe dispozitive master, o valoare ‘0’ la intrarea SS trece dispozitivul curent in modul “Slave”
  - Configuratii cu mai multe dispozitive



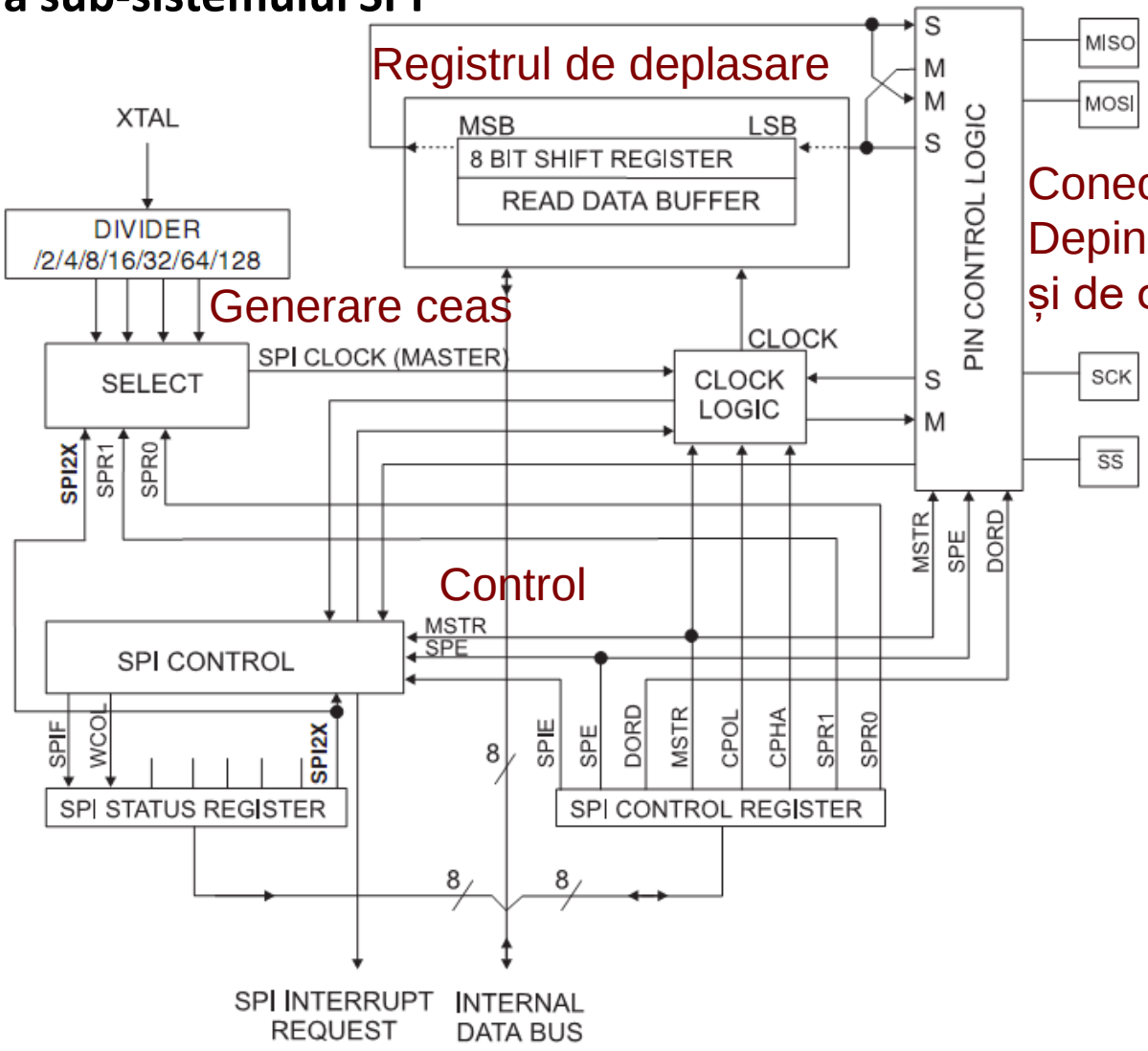
semnale **SS** independente



“daisy chain”

Serial Peripheral Interface (SPI) la AVR

- **Arhitectura sub-sistemului SPI**



Serial Peripheral Interface (SPI) la AVR

- Configurare SPI

Bit	7	6	5	4	3	2	1	0	
0x0D (0x2D)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Registrul **SPCR**:
  - SPIE – SPI Interrupt Enable, generare întrerupere la terminarea transmisiei
  - SPE – SPI Enable. Trebuie setat pe 1 pentru orice operație cu SPI
  - DORD – Data Order. 1=LSB first, 0 = MSB first
  - MSTR – 1: Master, 0: Slave
  - CPOL, CPHA – selectează polaritatea și faza semnalului SCLK

	Leading Edge	Trailing Edge
CPOL = 0, CPHA = 0	Sample (Rising)	Setup (Falling)
CPOL = 0, CPHA = 1	Setup (Rising)	Sample (Falling)
CPOL = 1, CPHA = 0	Sample (Falling)	Setup (Rising)
CPOL = 1, CPHA = 1	Setup (Falling)	Sample (Rising)

  - SPR1, SPR0 – reglează viteza SPI impreuna cu SPI2X din registrul SPSR

Serial Peripheral Interface (SPI) la AVR

- Configurare SPI

Bit	7	6	5	4	3	2	1	0	
0x0E (0x2E)	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Registrul **SPSR**:
  - SPI2X – Reglare frecvență ceas, împreună cu SPR1 și SPR0 din SPCR
  - WCOL – Write collision: setat dacă scriem in **SPDR** înainte ca SPI să transfere datele
  - SPIF – SPI Interrupt flag: setat când se termină transmisia. Daca SPIE este setat, se generează cerere de întrerupere

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

Serial Peripheral Interface (SPI) la AVR

- **Utilizare SPI (Master)**
  - Configurare directie pini I/O: Pinii SPI sunt comuni cu pinii portului B

Port Pin	Alternate Functions
PB7	OC2/OC1C(1) (Output Compare and PWM Output for Timer/Counter2 or Output Compare and PWM Output C for Timer/Counter1)
PB6	OC1B (Output Compare and PWM Output B for Timer/Counter1)
PB5	OC1A (Output Compare and PWM Output A for Timer/Counter1)
PB4	OC0 (Output Compare and PWM Output for Timer/Counter0)
PB3	MISO (SPI Bus Master Input/Slave Output)
PB2	MOSI (SPI Bus Master Output/Slave Input)
PB1	SCK (SPI Bus Serial Clock)
PB0	$\overline{SS}$ (SPI Slave Select input)

  - Configurare: scriere SPCR si SPSR cu valorile corespunzatoare pentru modul de lucru
  - Activare SS (PB(0) ← ‘0’, explicit!)
  - Scriere date in SPDR – declanșează transmisia
  - Așteptare până SPIF din SPSR este setat – transmisie completa
  - Citire date din SPDR – datele trimise de ‘slave’
  - Dezactivare SS (PB(0) ← ‘1’)

Serial Peripheral Interface (SPI) la AVR

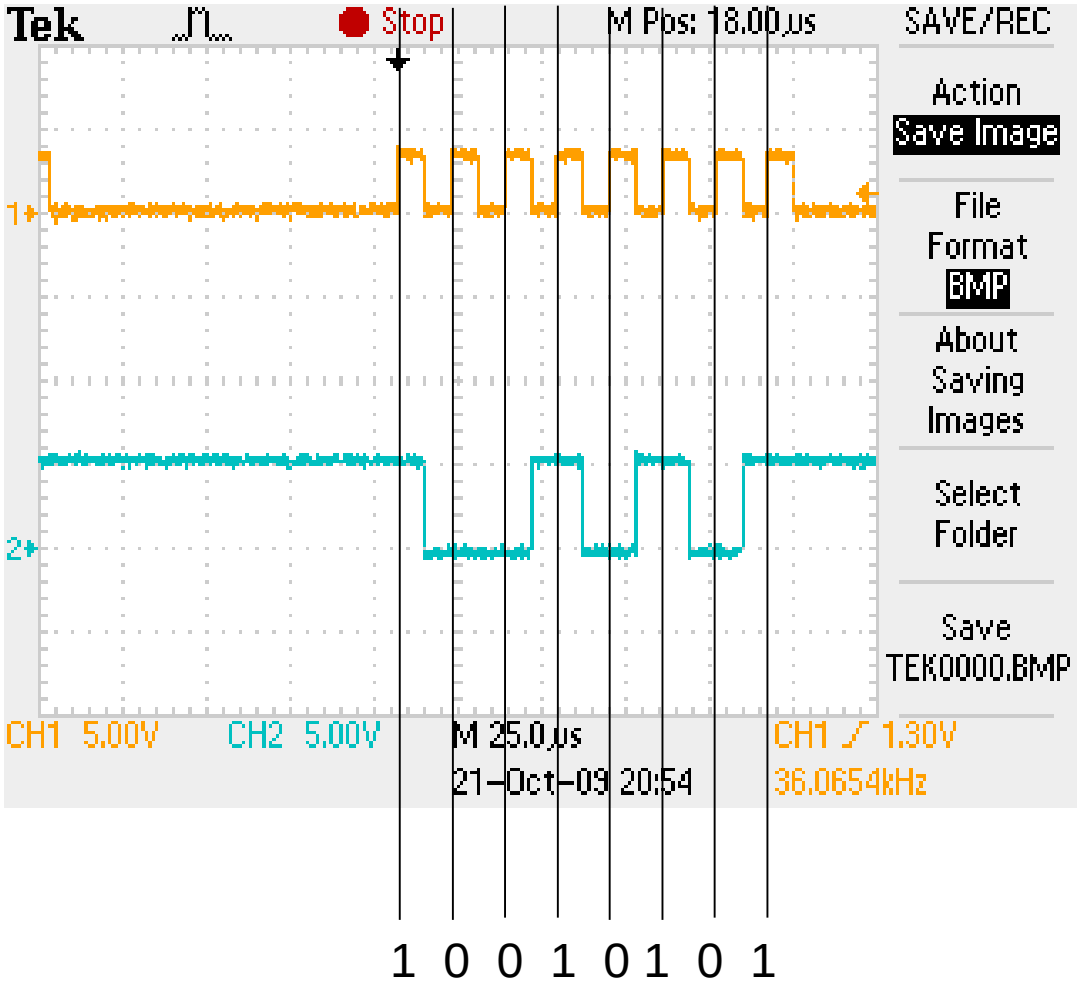
- **Utilizare SPI (Master) – Cod sursă**

```
.org 0x0000
      jmp reset
reset:
      ldi r16,0b00000111          ; MISO intrare, MOSI, SCK si SS iesire
      out DDRB,r16
      ldi r16, 0b00000001         ; Initial, SS<--1, SPI Slave inactiv
      out PORTB, r16
      cbi SPSR, 0                 ; pune bitul zero din SPSR pe zero - pentru frecventa
      ldi r16,0b01010011          ; Intreruperi dezactivate, SPI Enabled, MSB first, Master,
                                  ; CPOL=0, CPHA = 0, Frecventa cea mai lenta
      out SPCR,r16
loop:
      cbi PORTB, 0                ; SS ← 0
      ldi r16, 0b10010101         ; datele de transmis
      out SPDR, r16
wait:
      sbis SPSR, 7                ; bitul 7 din SPSR - transmisie completa
      rjmp wait
      in r16,SPDR
      sbi PORTB, 0                ; SS ←1
      ldi r18, 0
wait2:                            ; pauza intre transmisii
   dec r18
   brne wait2
rjmp  loop
```

- **Utilizare SPI (Master) – Rezultat**

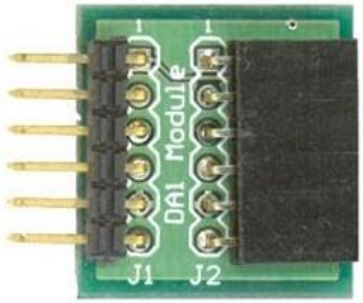
SCK

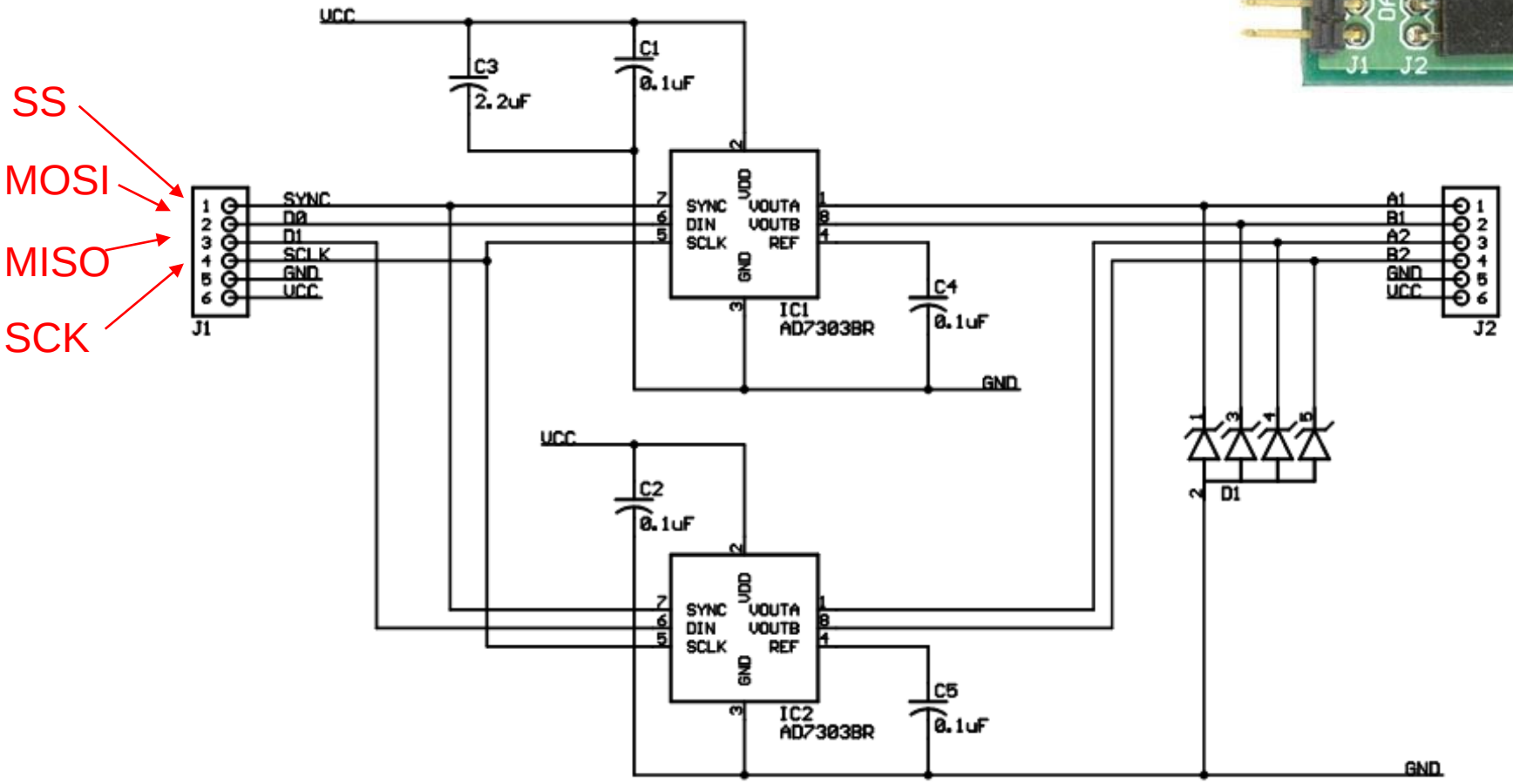
MOSI



Serial Peripheral Interface (SPI) la AVR

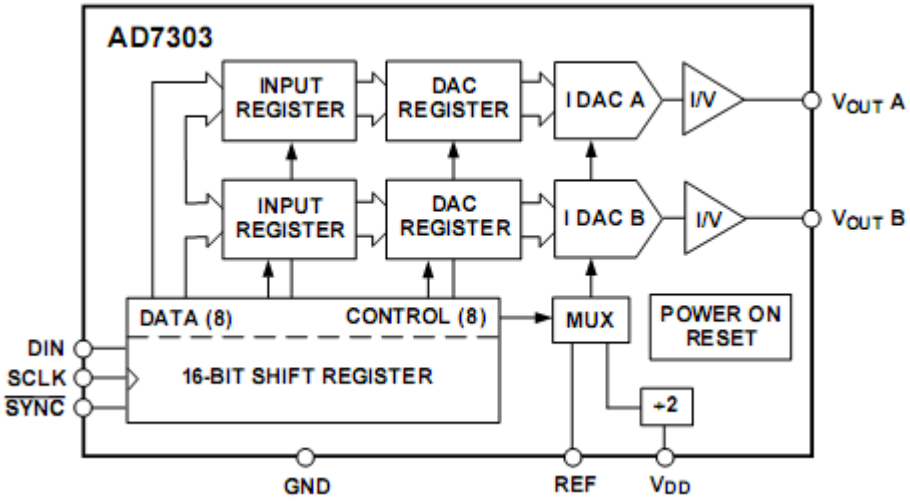
- **Conectare module prin SPI**
- Digilent **PMOD DA1** – Digital to Analog Converter

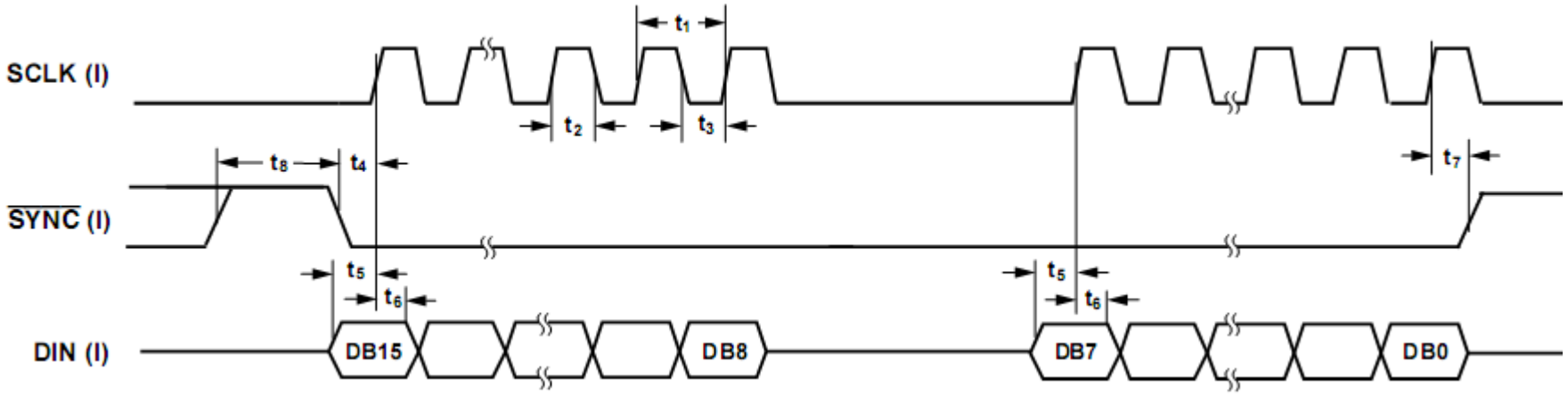




Serial Peripheral Interface (SPI) la AVR

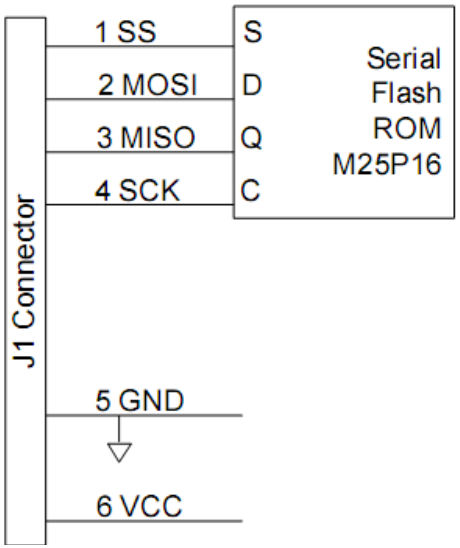
- **Conectare module prin SPI – PMOD DA1**
- Transmisie 16 biți (2x8 biți) – primii 8 date, următorii control



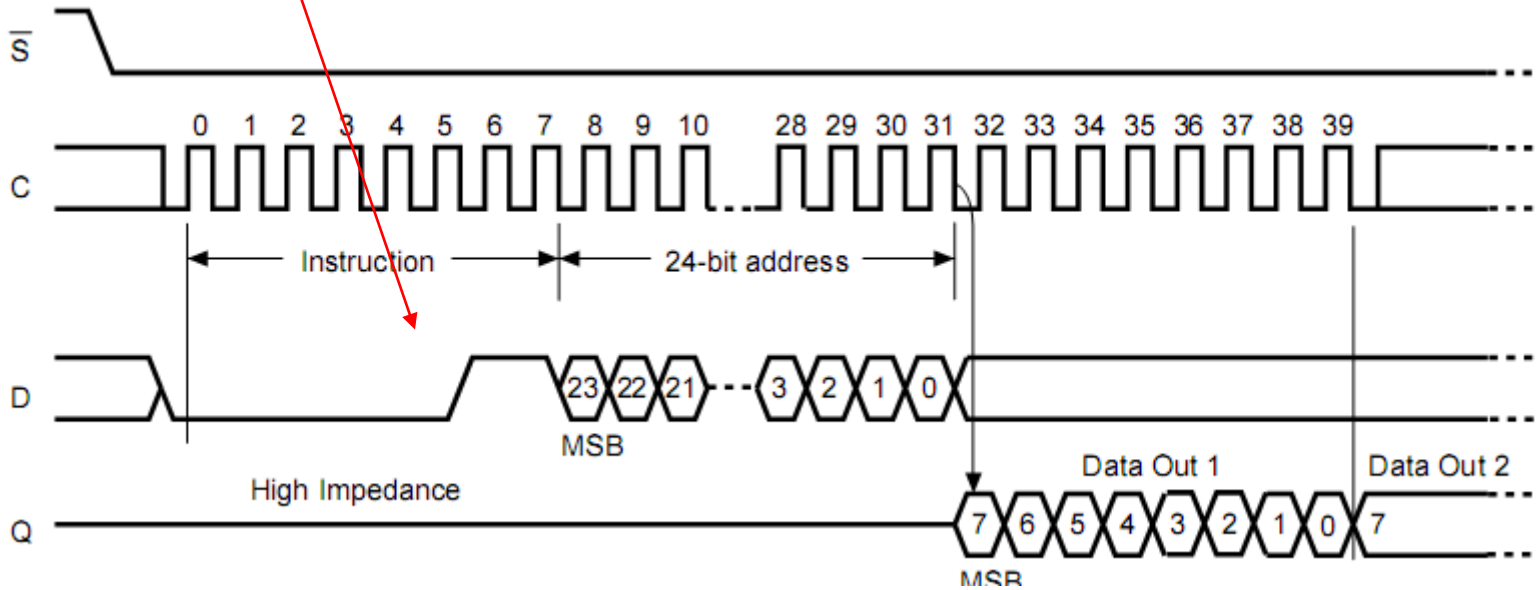


Serial Peripheral Interface (SPI) la AVR

- **Conectare module prin SPI**
- Digilent PMOD SF – Serial Flash



00000011 = 'READ'



Comunicare SPI la Arduino

- Se folosește biblioteca SPI (http://arduino.cc/en/Reference/SPI) [3]
- Funcții oferite:
  - SPI.setBitOrder(order) – ordinea biților: LSBFIRST sau MSBFIRST
  - SPI.setDataMode(mode) – faza și polaritatea transmisiei: SPI_MODE0, SPI_MODE1, SPI_MODE2 sau SPI_MODE3 (combinațiile CPHA și CPOL)
  - SPI.setClockDivider() – divizorul de frecvență: SPI_CLOCK_DIV(2 .. 128)
  - SPI.begin() – initializare interfață SPI, configurând pinii SCK, MOSI, și SS ca ieșire, punând SCK și MOSI pe LOW, și SS pe HIGH.
  - SPI.end() – dezactivează SPI, dar lasă pinii în modul în care au fost setați la inițializare
  - ReturnByte SPI.transfer(val) – transfera un byte (val) pe magistrala SPI, receptionând în același timp octetul ReturnByte.

- Biblioteca SPI suportă doar modul master
- Orice pin poate fi folosit ca Slave Select (SS).

Comunicare SPI la Arduino

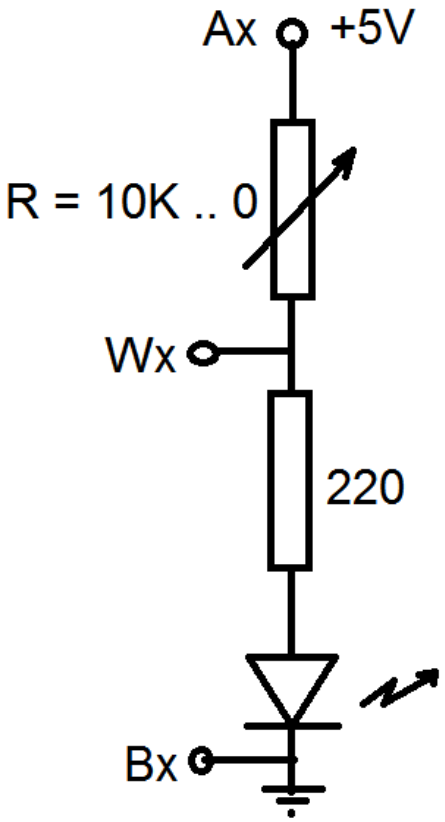
- **Exemplu (SPI)** – Controlul unui potențiometru digital folosind SPI [4]

http://www.youtube.com/watch?v=1nO2SSExEnQ

AD5206 datasheet: http://datasheet.octopart.com/AD5206BRU10-Analog-Devices-datasheet-8405.pdf

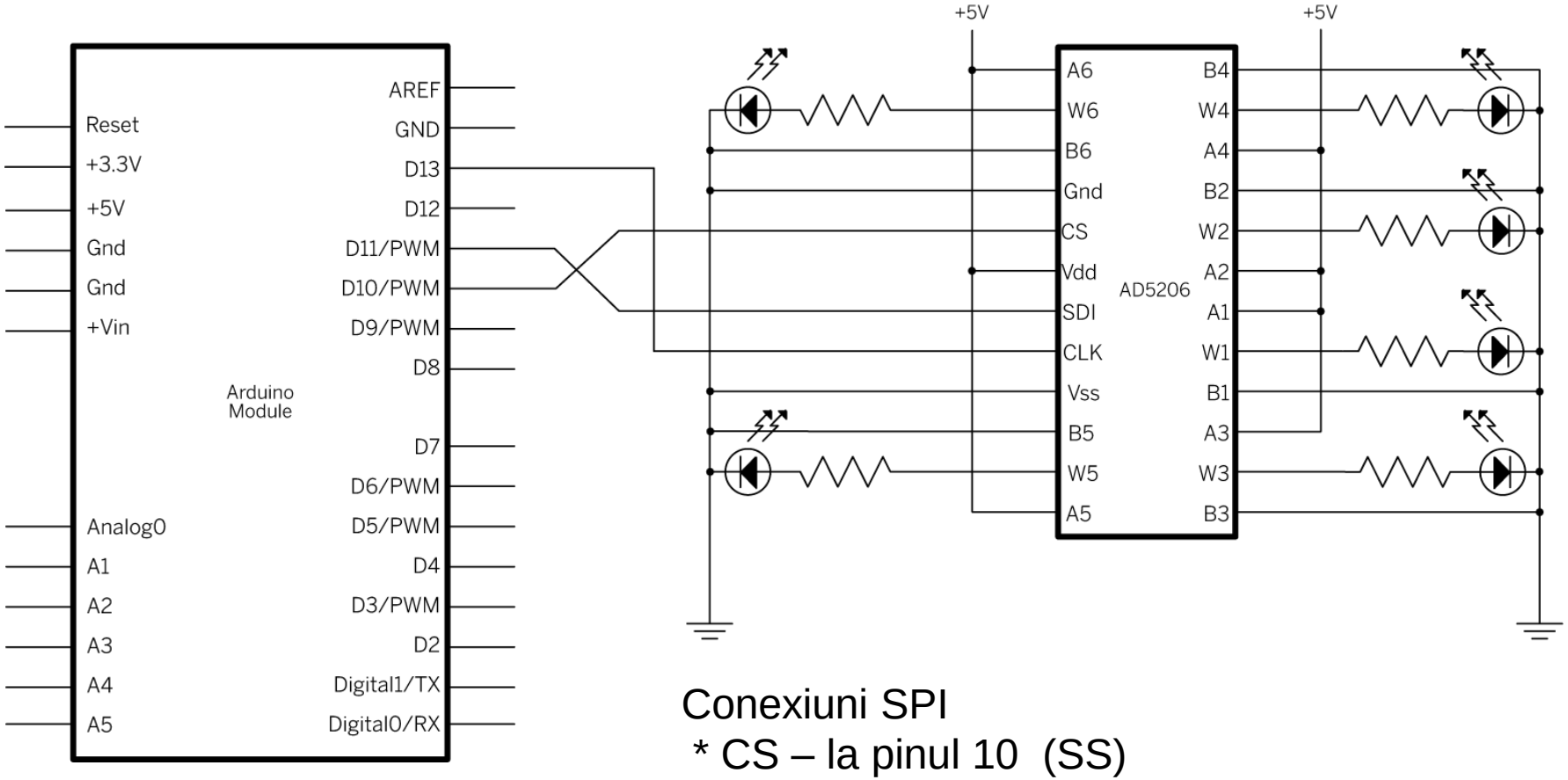
AD5206 este un potențiometru digital cu 6 canale (echivalent cu șase potențiometre individuale).

- 3 pini pe chip pentru fiecare element: Ax, Bx și Wx (Wiper – cursorul).
- pin A = HIGH, pin B = LOW și pinul W = tensiune variabilă. R are rezistența maximă de 10 Kohm, împărțită în 255 pasi.
- Pentru controlul rezistenței, se trimit pe SPI doi octeti: primul pentru selecția canalului (0 – 5) și al doilea cu valoarea rezistenței pentru fiecare canal (0 – 255) .



Comunicare SPI la Arduino

- **Exemplu (SPI)** – Controlul unui potențiometru digital folosind SPI [4]



Conexiuni SPI
* CS – la pinul 10 (SS)
* SDI – la pinul 11 (MOSI)
* CLK – la pinul 13 (SCK)

Comunicare SPI la Arduino

- **Exemplu (SPI)** – Controlul unui potențiometru digital folosind SPI [4]

```
#include <SPI.h>
const int slaveSelectPin = 10; // pin 10 ca SS

void setup() {
  // SS trebuie configurat ca iesire
pinMode (slaveSelectPin, OUTPUT);
SPI.begin();   // activare SPI
}

void loop() {
  // se iau cele 6 canale la rand
  for (int channel = 0; channel < 6; channel++) {
    // se schimba rezistenta fiecarui canal de la min la max
    for (int level = 0; level < 255; level++) {
      digitalPotWrite(channel, level);
      delay(10);
    }
     delay(100); // asteptam 100 ms cu rezistenta maxima
    // se schimba rezistenta de la max la min
    for (int level = 0; level < 255; level++) {
      digitalPotWrite(channel, 255 - level);
      delay(10);
    }
  }
}
```

```
// functia care foloseste SPI pentru actualizarea
// rezistentelor
void digitalPotWrite(int address, int value) {
  // activeaza SS prin scriere LOW
  digitalWrite(slaveSelectPin, LOW);
  // se trimit pe rand canalul si valoarea:
  SPI.transfer(address);
  SPI.transfer(value);
  // inactiveaza SS prin scriere HIGH
  digitalWrite(slaveSelectPin, HIGH);
}
```

Referințe

1. Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V datasheet
2. Atmel Atmega64 datasheet
3. Arduino SPI reference guide: http://arduino.cc/en/Reference/SPI
4. Arduino SPI Tutorials: http://arduino.cc/en/Tutorial/SPIDigitalPot
5. Arduino I2C Library: http://arduino.cc/en/reference/wire
6. Arduino I2C tutorial: http://arduino.cc/en/Tutorial/MasterWriter