Serial Peripheral Interface (SPI) – Lucrarea 6

UART, I2C și SPI sunt 3 dintre cele mai comune interfețe de transmisie serială a datelor prin mai multe fire. Interfața ***Serial Peripheral Interface (SPI)*** dezvoltată de Motorola la mijlocul anilor '80 este utilizată pentru a transfera date între circuite integrate, utilizând un număr redus de linii de date; prin utilizarea a 4 linii de date, magistrala SPI oferă ***comunicare serială sincronă de tip full-duplex*** între un dispozitiv master și unul slave.
Busul de interfață SPI este simplu și versatil, permițând o comunicare simplă și rapidă cu o varietate de periferice, precum: memorii flash, senzori, ceasuri în timp real (RTC), convertoare analog-digitale, LCD-uri (afișaje cu cristale lichide) și multe altele, acesta fiind utilizat în principal în sistemele încorporate (pentru comunicarea pe distanțe scurte).

Dispozitivele SPI comunică în modul full duplex utilizând o ***arhitectură master-slave cu un singur master***. Magistrala SPI poate funcționa cu un singur dispozitiv master și un singur slave (Fig.1) sau mai multe dispozitive slave (Fig.2).
Dispozitivul master generează cadrul pentru citirea și scrierea datelor: datele transmise între master și slave sunt sincronizate cu ajutorul unui semnal de ceas generat de master. Dispozitivele SPI acceptă frecvențe de ceas mult mai mari comparativ cu interfețele I2C (se recomandă consultarea fișei tehnice a produsului pentru specificația frecvenței de ceas a interfeței SPI).

Într-un sistem, sunt acceptate multiple dispozitive slave, dar cu selectarea individuală a slave-urilor prin liniile de selectare a cipului (semnale chip select /CS). Semnalul /CS pentru selecția slave-ului este în mod normal un semnal activ LOW și poartă numele de ***slave select (SS)***. Când SS este în starea LOW, slave-ul respectiv este selectat, iar când este în starea HIGH slave-ul respectiv este deconectat de la magistrala SPI. Când se utilizează mai multe dispositive slave, este necesar un semnal individual de selectare a fiecărui slave în parte de către master.

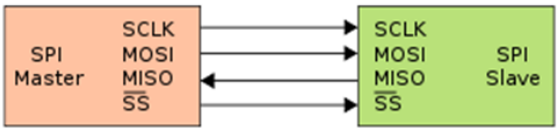


Fig.1 Configurație SPI pentru un master și un slave

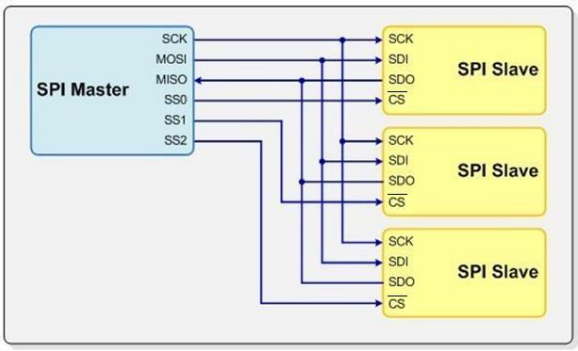


Fig.2 Configurație SPI tipică pentru un master și mai multe slave-uri

Configurație Multislave
Mai multe dispozitive slave pot fi folosite cu un singur master SPI. Dispozitivele de tip slave pot fi conectate în modul obișnuit sau în modul lanț.
Configurație SPI tipică:
În modul de conectare obișnuit într-un sistem cu SPI, există un semnal SS individual pentru fiecare slave deservit de master (Fig. 3). Odată ce semnalul SS este activat (pe LOW) de către master, semnalul de ceas și datele de pe liniile MOSI / MISO sunt disponibile pentru slave-ul selectat. Dacă sunt activate mai multe semnale SS deodată, datele de pe linia MISO sunt corupte, deoarece nu există nicio modalitate prin care masterul să identifice care slave transmite datele. Astfel, într-o configurație SPI tipică, masterful activează un singur slave la un moment dat.
Pe măsură ce crește numărul de dispozitive slave, crește și numărul de linii SS ce trebuie controlate de master, limitând astfel numărul de slave-uri care pot fi folosite într-un sistem cu SPI. Există diferite tehnici care pot fi utilizate pentru a crește numărul de dispozitive slave în modul regulat; de exemplu, folosind un circuit mux pentru a genera un semnal /CS.

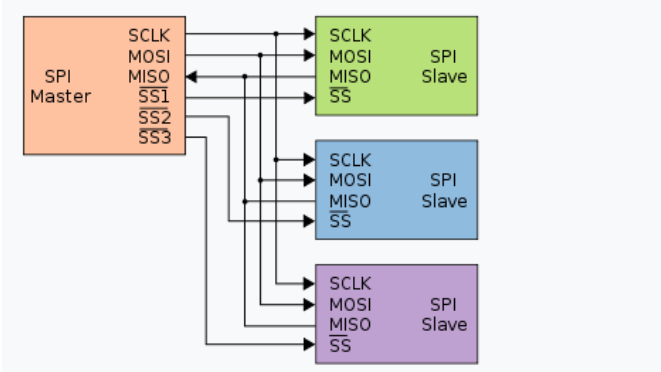


Fig. 3 Bus SPI tipic: un master și 3 slave-uri independente

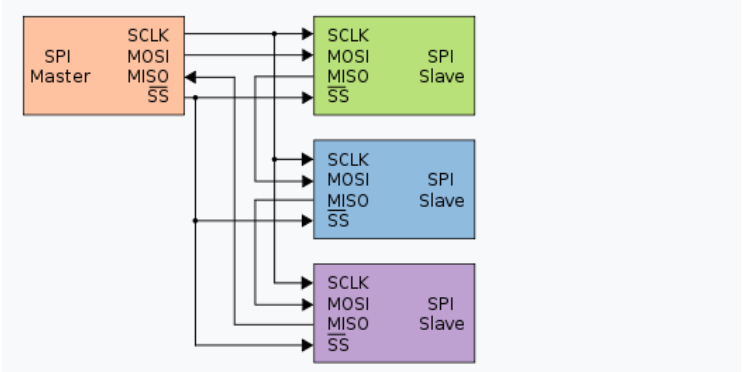


Fig. 4 Configurație Daisy-chain: un master și 3 slave-uri conectate în lanț

Configuratie de lant de tip Daisy:
Unele produse care implementează SPI pot fi conectate într-o configurație de lanț de tip Daisy chain (Fig. 4), prima ieșire slave fiind conectată la a doua intrare slave ș.a.m.d. Portul SPI al fiecărui slave este conceput pentru a trimite (în timpul celui de-al doilea grup de impulsuri de ceas) o copie exactă a datelor primite în timpul primului grup de impulsuri de ceas. Întregul lanț acționează ca un registru de deplasare: înlănțuirea de tip Daisy se face adesea cu registre de shiftare pentru a furniza un banc de intrări sau ieșiri prin SPI. Fiecare slave copiază intrarea la ieșire în următorul ciclu de ceas până când linia SS activă (în LOW) devine HIGH (se dezactivează).

Astfel, datele sunt trimise în cascadă pe linie, de la primul slave până la ultimul slave din serie, iar acesta din urmă poate folosi apoi linia MISO pentru a trimite date către dispozitivul principal, deci către master. O astfel de configurație necesită doar o singură linie SS de la master, și nu o linie SS separată pentru fiecare slave în parte (ca în mod tipic).
În această configurație, pe măsură ce datele sunt propagate de la un slave la altul, numărul de cicluri de ceas necesare pentru transmiterea datelor este proporțional cu poziția slaveului în lanțul de tip Daisy. De exemplu, într-un sistem pe 8 biți, sunt necesare 24 de impulsuri de ceas pentru ca datele să fie disponibile pe al treilea slave, comparativ cu doar opt impulsuri de ceas în modul SPI obișnuit. Modul Daisy-chain nu este neapărat acceptat de toate dispozitivele SPI. (se recomandă consultarea fișei tehnice a produsului)

Interfațarea SPI: Pentru a începe comunicarea SPI, masterul trebuie să trimită semnalul de ceas și să selecteze slave-ul activând semnalul SS. De obicei, semnalul pt selectarea cipului este un semnal activ LOW (/CS); prin urmare, masterul trebuie să trimită 0 logic pe acest pin SS, pentru a selecta slave-ul.
Busul SPI specifică 4 semnale de interfață:

- SCLK: Serial Clock (ieșire de la Master)
- MOSI: Master Out Slave In (datele ies din master și intră în slave)
- MISO: Master In Slave Out (datele ies din slave și intră în master)
- SS: Slave Select (semnal activ în general pe LOW, ieșire de la master)

MOSI și MISO sunt liniile de date: MOSI transmite date de la master la slave și MISO transmite date de la slave la master.
Semnalul MOSI de la master se conectează cu semnalul MOSI de la slave. Semnalul MISO de la master se conectează cu semnalul MISO de la slave. Semnalul SS (Slave Select) are aceeași funcționalitate ca și semnalul /CS (activare circuit) (Fig. 1).

Desfășurare lucrare: Pentru început vom analiza un program în care se dă transmisia serială sincronă din figura de mai jos, cu formele de undă obţinute pe osciloscopul digital din UnoArduSim.

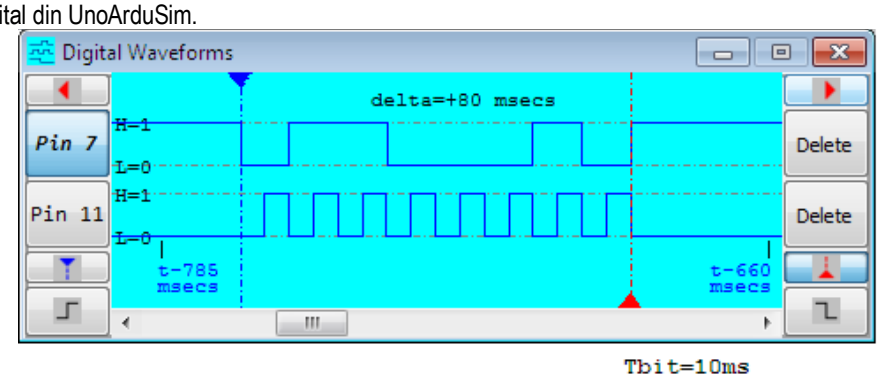
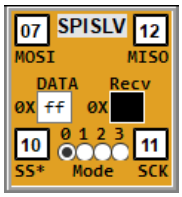


Fig. 5. Forme de undă obţinute cu osciloscopul digital cu 2 canale din simulator pentru problema propusă

In exemplul din figură se transmite caracterul '**b**', având cod ASCII 01100010b, cu MSB transmis primul (în stânga); Pentru configurare, s-a ales pinul 7 pentru semnalul de date (MOSI) şi pinul 11 pentru semnalul de ceas (SCLK), iar perioada de ceas este de 8us. Se cere să se implementeze în setup() o astfel de transmisie, folosind funcţiile Arduino delayMicroseconds() şi digitalWrite(), transmiţând caracterul 'b' şi folosind viteza de transmisie 100 biţi pe secundă. Acest program simulează de fapt o interfaţă serială de tip SPI. Programul se va rula având osciloscopul deschis (click stânga pe pinii 7 şi 11). Imediat după lansarea în execuţie a programului, se urmăresc formele de undă şi se opreşte programul, surprinzând pe ecran forma digitală a caracterului 'b' = 01100010b (aşa cum se observă în Fig.5).

Soluție: din UnoArduSim se alege componenta **Configurable SPI Slave ('SPISLV') și se va configura astfel:**
Tbit = 1/v => Tbit [usec] = 1.000.000 / v = 1.000.000 /100 = 10.000 usec = 10 msec
(e corect, întrucât pe figură se obs că în 80 msec se transmit 8 biți ai caracterului => un bit se transmite în 10 msec). Pentru a forma semnalul de ceas, avem nevoie de semnal în High pentru o perioadă Tbit/2 şi semnal în Low pentru o perioadă Tbit/2 (a se vedea funcția **pulsCeas()**).
Functia **linie_in_repaus() pune pe pinul de date High si pe pinul de ceas Low.**
S-a folosit modul de lucru 1 al interfeţei



Functia loop() va fi vidă, astfel că se va implementa aplicaţia în funcția Setup, pentru a rula o singură dată. Veţi da comanda "Accept" de fiecare dată când vreți să re-rulați aplicaţia. Pentru a vizualiza osciloscopul digital, dați click pe pinii doriți (7 și 11) și se va deschide "Digital Waveform".

```
#define X 1
#define Y 7
#define V 100*X

int pinDate = Y;
int pinCeas = (X+Y)%11 + 3;// Z
int Tbit_us = 1000000/V;// b/s --> us(microsecs)
char carX = 'a' + X%3; // doar A, B sau C

void setup(){
    int bitActual;
    pinMode(pinDate, OUTPUT);
```

```
    pinMode(pinCeas, OUTPUT);
// stare init. linii de semnal

linie_in_repaus();

delay(100);                 // pt. o forma de unda clara la validare


// bucla de serializare car.(octet) de TX MSB 1st
for (int j=7; j>=0; j--){                   //bitActual = carX & (1<<j); // ori asa, ori cu linia de mai jos
        bitActual=bitRead(carX, j);
    if (bitActual == 0)
                digitalWrite(pinDate, LOW);
    else
                digitalWrite(pinDate, HIGH);
    pulsCeas();
                  }  //  end for()
linie_in_repaus();
}  // setup()

void loop(){ }      // functia loop e goala

void linie_in_repaus(){
   digitalWrite(pinDate, HIGH);
   digitalWrite(pinCeas, LOW);
}

void pulsCeas(){
    delayMicroseconds(Tbit_us/2); // front urcator, ceas la mijloc de bit
    digitalWrite(pinCeas, HIGH);
    delayMicroseconds(Tbit_us/2); // front cazator, ceas la mijloc de bit
    digitalWrite(pinCeas, LOW);
}
```

Aplicatia 2: modificati aplicatia 1, astfel: **Adaugati semnalul Slave Select pe pinul 9**
Indicatii:
se adauga ca si variabila: **int SlaveSel_pin = 9;**

In program: **pinMode(SlaveSel_pin, OUTPUT);**
// activare SlaveSelect
digitalWrite(SlaveSel_pin, LOW);

si la iesire din Setup: **digitalWrite(SlaveSel_pin, HIGH);**

Aplicatia 3:

```
#define X 1
#define Y 7
#define V 100*X

int pinDate = Y;
int pinCeas = (X+Y)%11 + 3;// Z
int Tbit_us = 1000000/V;// b/s --> us(microsecs)
char carX = 'a' + X%3; // doar A, B sau C
int SlaveSel_pin = 9;

void setup(){
            int bitActual;
            pinMode(pinDate, OUTPUT);
            pinMode(pinCeas, OUTPUT);
            pinMode(SlaveSel_pin, OUTPUT);
            // activare SlaveSelect
            digitalWrite(SlaveSel_pin, LOW);
            // stare init. linii de semnal
            linie_in_repaus();
            delay(100); // pt. o forma de unda clara la validare
            // bucla de serializare car.(octet) de TX MSB 1st
            for(int j=7; j>=0; j--){
                        //bitActual = carX & (1<<j); // ori asa, ori cu linia de mai jos, e OK
                        bitActual=bitRead(carX, j);
                        if(bitActual == 0)
                                    digitalWrite(pinDate, LOW);
                        else
                                    digitalWrite(pinDate, HIGH);
```

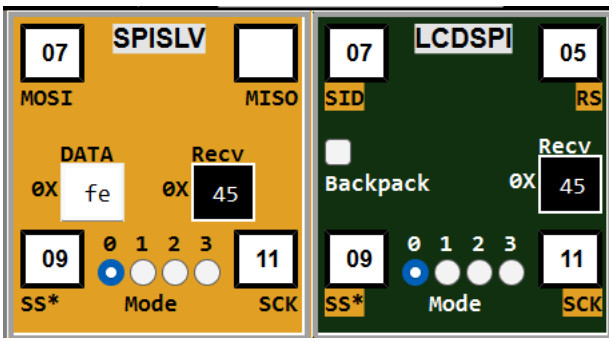
```
                pulsCeas();
        }//for()
        linie_in_repaus();
        // Dezactivare SlaveSelect
        digitalWrite(SlaveSel_pin, HIGH);
}// setup()

void loop(){ }

void linie_in_repaus(){
        digitalWrite(pinDate, HIGH);
        digitalWrite(pinCeas, LOW);
}

void pulsCeas(){
        delayMicroseconds(Tbit_us/2); // front urc. ceas la mijloc de bit
        digitalWrite(pinCeas, HIGH);
        delayMicroseconds(Tbit_us/2); // front caz. ceas la mijloc de bit
        digitalWrite(pinCeas, LOW);
}
```

Aplicatia 4: Conectarea unui LCD prin SPI:




```
/*
functia trimite atat la SPISLV cat si la SPILCD un sir de caractere, "bcd EFg"

1. Adaugat SPISLV
alegem Slave Select <-- pin9
schimbam MOSI <-- pin7
SCK <-- 11
stergem casuta MISO(neconectat)
Mode <-- lasam cum era default, Mode 0 (ZERO)

2. Conectam si un LCD pe SPI la acelasi MOSI, SS si SCK
RS <-- pin5
Help/IDE --> Text LCD SPI ('LCDSPI')
"...
This 'I/O' device emulates a 1, 2 or 4-line character-LCD,
in one of two modes:
a) backpack (Adafruit -SPI port expander)(if check-box "Backpack" = ON)
b) no backpack, native mode integrated SPI interface , <-- asta o alegem
Pin 'SID' is serial data in, i.e. MOSI <-- pin 7
and 'RS'(Reg. Selct.) is the command/data i.e. ~c/d pin., <-- alegem pin 5
You can only write to LCD,
commands and DDRAM data,
not read back from LCD.
Double-click on LCDSPI peripherial to set the the display (16x2, etc.)
.."
Setam mod SPI Mode0 si pt. LCD, asa era implementarea "bit-banged"
CPOL=0--starea de repaus al ceasului este LOW(polaritate pozitiva),
CPHA =0 bitii de date se citesc la frontul urcator al ceasului(pe frontul pozitiv)
```

Enunt:Se da transmisia seriala sincrona din figura, forme de unda pe osciloscopul digital UnoArduSim.In exemplul din figura se transmite caracterul 'b', avand cod ASCII 01100010b, MSB 1-st, s-a ales pinul 7 pt. semnalul de date, si 11 pt. cel de ceas, iar perioada de ceas este de 8us. Implementati in setup() o astfel de transmisie, folosind functiile Arduino delayMicroseconds() si digitalWrite(), transmitand caracterul X, datele pe pinul Y, ceasul pe Z si viteza de transmisie V biti pe sec. X, Y, Z si V sunt datele personalizate ale enuntului problemei.

```
Functia loop() va fi vida.*/

#define X 1 // din tabelul de coduri-student
```

```
#define Y 7
#define V 100*X

int pinDate = Y;
int pinCeas = (X+Y)%11 + 3;// Z
int Tbit_us = 1000000/V;// b/s --> us(microsecs)

// adaugate in noi.2020
char carX[]="bcd EFg";// GLOB var
int SlaveSel_pin = 9;// necesar pt. SPISLV

int RS_pin = 5; // ~C/D pt. LCDSPI

void setup(){
pinMode(pinDate, OUTPUT);
pinMode(pinCeas, OUTPUT);
pinMode(SlaveSel_pin, OUTPUT);
pinMode( RS_pin, OUTPUT);
digitalWrite(SlaveSel_pin, LOW); // activare SlaveSelect
linie_in_repaus(); // stare init. linii de semnal if. SPI
LCD_init(); // seteaza poz. de scriere(cursor) pe "Coltul stg.-a sus"

for(int i=0; i<strlen(carX); i++)
SPI_wr_byte(carX[i]);

// scriem ceva si in linia a doua. Sirul inversat de la jumatate incolo
digitalWrite(RS_pin, LOW); // LCD-command follows
SPI_wr_byte(0b10000000 | 0x40+8); delayMicroseconds(40);// set cursor to line 2 col 8
digitalWrite(RS_pin, HIGH); // LCD-data follows
for(int i=strlen(carX)-1; i>=0;i--)
SPI_wr_byte(carX[i]);

linie_in_repaus();
digitalWrite(SlaveSel_pin, HIGH); // Dezactivare SlaveSelect, ...gata
}// setup()

void loop(){ }

void linie_in_repaus(){
digitalWrite(pinDate, HIGH);
digitalWrite(pinCeas, LOW);
}

void pulsCeas(){
delayMicroseconds(Tbit_us/2); // front urc. ceas la mijloc de bit
digitalWrite(pinCeas, HIGH);
delayMicroseconds(Tbit_us/2); // front caz. ceas la mijloc de bit
digitalWrite(pinCeas, LOW);
}

void LCD_init(){
/* For 8-bit data-if.-mode, this is done as follows:
1. Wait more than 15 msecs after power is applied.
2. Write 0x030 to LCD and wait 5 msecs for the "Function-set" instr. to complete
3. Write 0x03X to LCD and wait 160 usecs for instr. to complete(X -- command param.-bits)
4. Write 0x03X AGAIN to LCD and wait 160 usecs or Poll the Busy Flag
5. Set the Operating Characteristics of the LCD
o Write 0x010 to turn off the Display-shift
o Write 0x001 to Clear the LCD
o Write "Set Cursor Move Direction" Setting Cursor Behavior Bits
o Write "Enable Display/Cursor/Crs.blk" & enable Display, Optional Cursor and Crs.-blink

The HD44780 instruction set is shown below:
4 5 14 13 12 11 10 9 8 7 Pins
R/S R/W D7 D6 D5 D4 D3 D2 D1 D0 Instruction/Description

0 0 0 0 0 0 0 0 0 1 Clear Display. It needs t>1.6ms to complete. Sets DDRAM addr.=0
0 0 0 0 0 0 0 0 1 * Return Cursor and LCD to Home Position(if any was shifted)
0 0 0 0 0 0 0 1 ID S Set Cursor Move Direction
0 0 0 0 0 0 1 D C B Enable Display/Cursor(Display, Cursor & Curs.blink ON/OFF cotrol)
0 0 0 0 0 1 SC RL * * Move Cursor/Shift Display(Cursor or Display-shift)
0 0 0 0 1 DL N F * * Set Interface Length ("Function-set")
0 0 0 1 A A A A A A Move Cursor into CGRAM(Set CGRAM Addr.)
0 0 1 A A A A A A A Move Cursor to Display posn(set DDRAM Addr.)
0 1 BF * * * * * * * Poll the "Busy Flag"
1 0 D D D D D D D D Write a Character(8bits) to the Display at the Current Cursor Position
1 1 D D D D D D D D Read the Character on the Display at the Current Cursor Position
```

The bit descriptions for the different commands are:
"*" - Not Used/Ignored. This bit can be either "1" or "0"
Set Cursor Move Direction:
ID - Increment the Cursor after Each Byte Written to Display if Set, decr. if clr'd
S - Shift Display when Byte Written to Display if S-bit is set. No shift if clr'd
Enable Display/Cursor
D - Turn Display On(1)/Off(0)
C - Turn Cursor On(1)/Off(0)
B - Cursor Blink On(1)/Off(0)
Move Cursor/Shift Display - command
SC - Shift Display if On(1) / shift the Cursor if Off(0)
RL - Direction of Shift Right(1)/Left(0)
Set Interface Length
DL - Set Data Interface Length 8(1)/4(0)
N - Number of Display Lines 1(0)/2(1)
F - Character Font 5x10(1)/5x7(0)
Poll the "Busy Flag"
BF - This bit is set while the LCD is processing
Move Cursor to CGRAM/Display
A - Address
Read/Write ASCII to the Display
D – Data

```
*/ // 1. Wait t>15ms after power on
delay(15); // ms
// 2. Write 0x30 to LCD and wait 5 msecs
digitalWrite(RS_pin, LOW);                  // ~C/D=0 means LCD-command;
SPI_wr_byte(0x30);                          // 0 0 1 DL N F * * Set data I.f. width=8b, nbr of lines=1
delay(5); // ms
// 3. Write 0x03X to LCD and wait 160 usecs for instruction to complete
// 4. Write 0x03X AGAIN to LCD and wait 160 usecs (or Poll the Busy Flag)
for(int i=0; i<2; i++){                     // (Function-set command) - same command, two times
      SPI_wr_byte(0b00111000);        // 0 0 1 DL N F * * Set 8bit-data I.F., 2 LCD rowsSPI_wr_byte(0x30);
     delayMicroseconds(160);
   }
// 5. Set the remaining Operating Characteristics of the LCD

/*o Write 0x10 to turn the Display-shifting off */
SPI_wr_byte(0x10);                  // 0 0 0 0 1 SC RL * * Move Cursor/Shift Display
delayMicroseconds(40);

/*o Write 0x01 to Clear the Display */
SPI_wr_byte(0b00000001);//Clear Display. Icludes also set DDRAM addr. to zero
delay(2);                                   // 1ms was not enough here! Command exec. needs at least 1.6ms

/*o Write "Set Cursor Move Direction" Setting Cursor Behavior Bits*/
SPI_wr_byte(0b00000110);// 0 0 0 0 0 1 ID S //Set Cursor Move Direction Incr/~Decr, no displ.-sh
delayMicroseconds(40);              // as of datasheets

/*o Write "Enable Display/Cursor" Enable Display and Optional Cursor */
SPI_wr_byte(0b00001111);        //0 0 0 0 1 D C B Enable Display, Cursor and Crs.blk
delayMicroseconds(40);

// c-da set DDRAM-Addr. to Line 0 col 0 (Adr. de 7-biti <-- ZERO)
SPI_wr_byte(0x80);
delayMicroseconds(40);

// Reg. Sel LCD: <-- Date
digitalWrite(RS_pin, HIGH); delayMicroseconds(40);
}

void SPI_wr_byte(char x){
int bitActual;
   // bucla de serializare car.(octet) de TX MSB 1st
for(int j=7; j>=0; j--){
                  //bitActual = x & (1<<j); // ori asa, ori cu linia de mai jos, e OK
              bitActual=bitRead(x, j);
              if (bitActual == 0)
                digitalWrite(pinDate, LOW);
             else
                digitalWrite(pinDate, HIGH);
            pulsCeas();
                  }//for()
}
```