Universitatea Tehnica a Moldovei
Facultatea Calculatoare, Informatica sI Microelectronica
Departamentul Informatica si Ingineria Sistemelor

Disciplina:
Bazele Transmiterii de Date

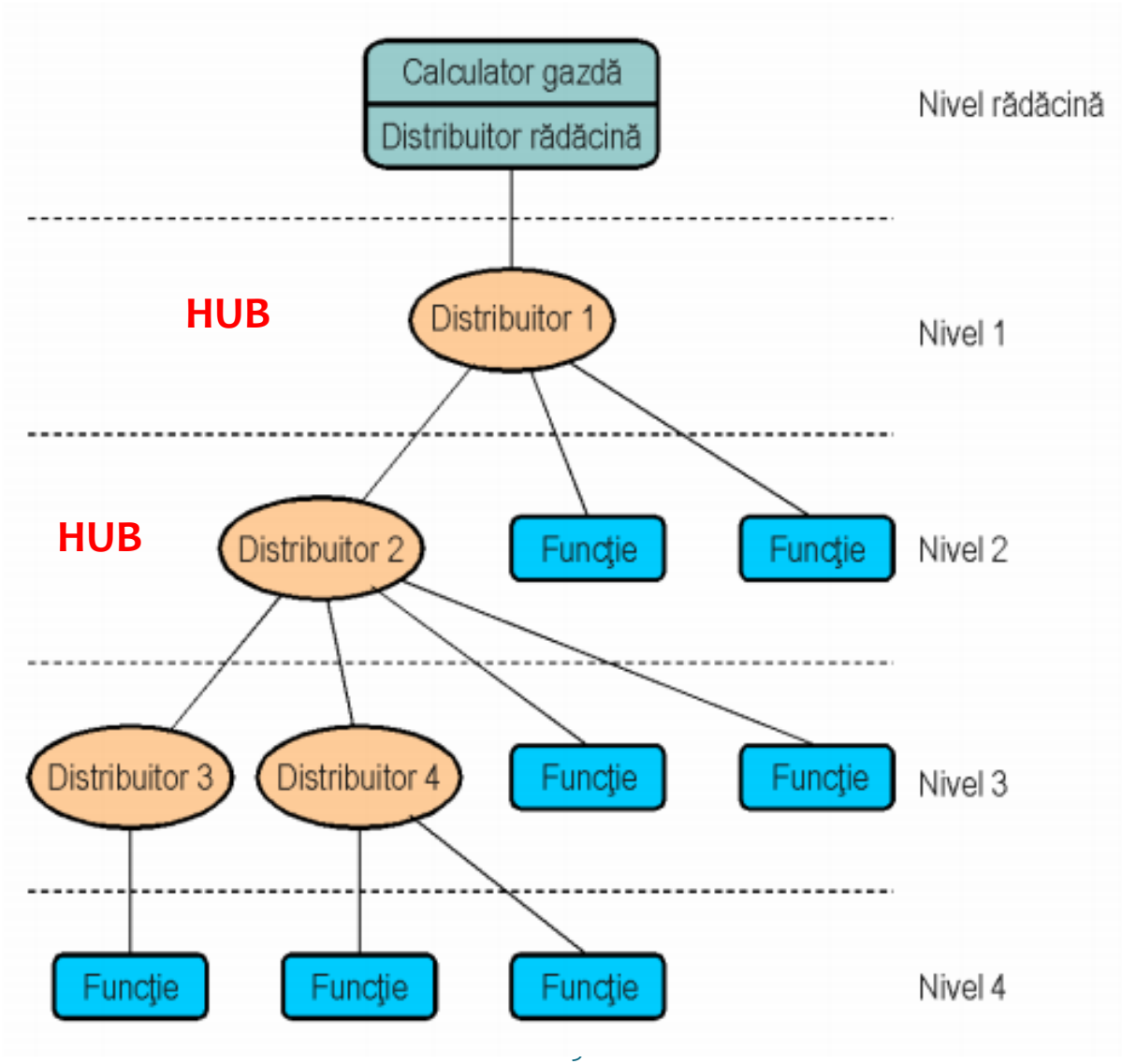
Tema Nr. 6.2. Interfete si Protocoale de Comunicare.

Titular de curs:
Conf.univ.,dr. V. Ababii

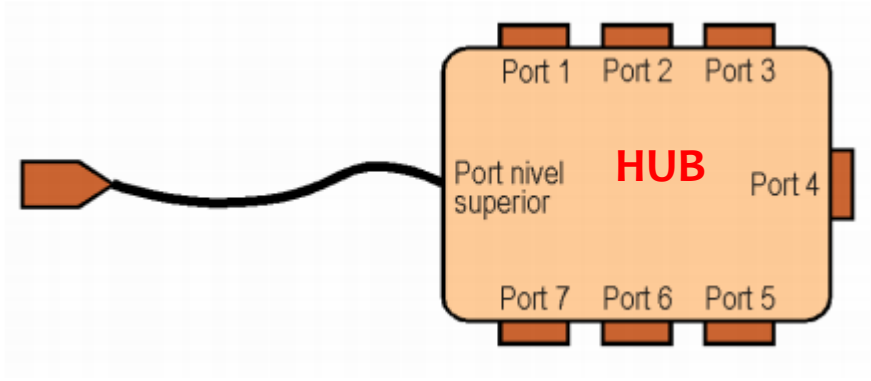
Subiecte abordate:

- Controlorul I8255. Structura. Principiul de funcționare. Programarea şi metode de utilizare la organizarea schimbului de date în cod paralel.
- Controlorul I8251. Structura. Principiul de funcționare. Programarea şi metode de utilizare la organizarea schimbului de date în cod secvențial.
- Interfața UART/USART
- Interfața COM.
- Magistrala USB.
- Interfețe RS: RS-232, RS-422, RS-423, RS-485.
- Interfața Bluetooth.
- Interfața IrDA.
- Interfața I2C.
- Interfața SPI.
- Interfața CAN.
- Interfața Ethernet.
- Controloare specializate pentru implementarea interfeței Ethernet. Protocoale de comunicare.

Interfata USB.



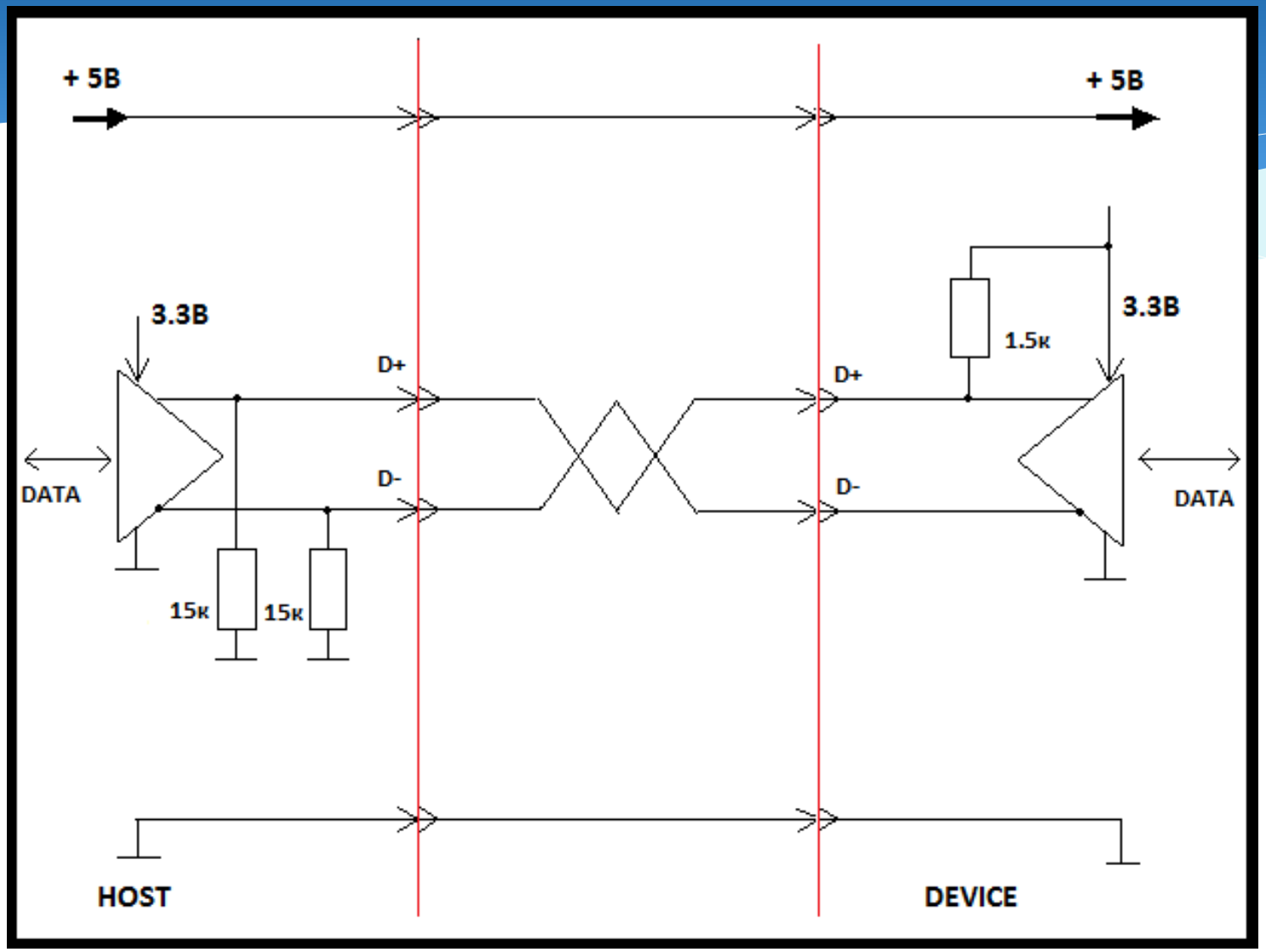
Interfata USB.



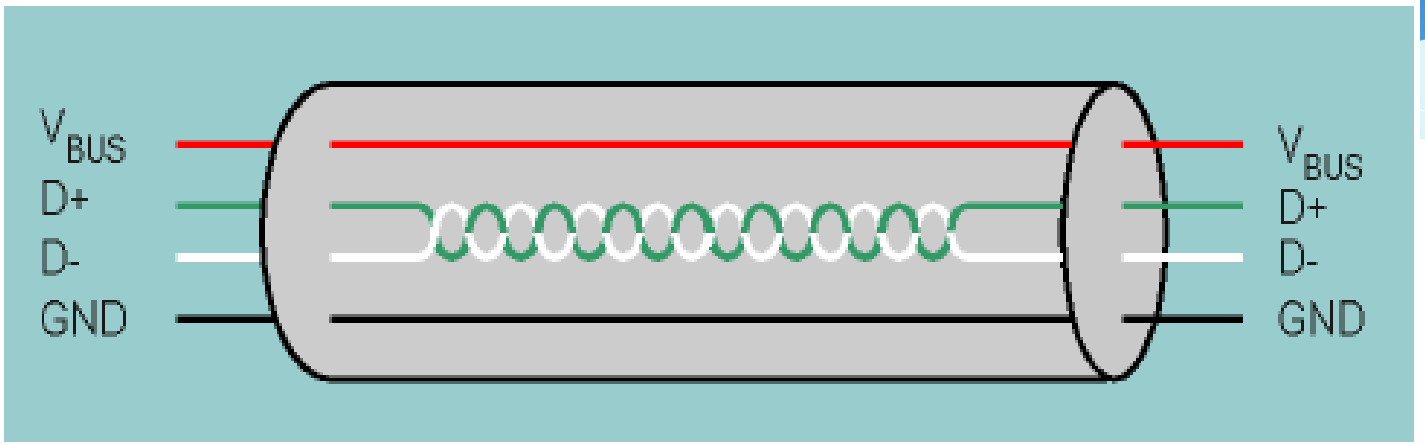
Standarde USB.

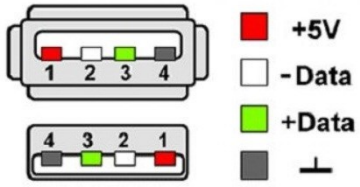
USB 1.0 *Low Speed* - apărut în 1996, a fost mai mult un prototip pentru USB 1.1, capabil de transferuri de date la viteze de până la 1,5 Mbps.
USB 1.1 *Full Speed* - apărut în 1998, cu viteze de transfer de 12 Mbps.
USB 2.0 *Hi-Speed* - apărut în aprilie 2000 și suportă viteze de transfer teoretice de maxim 480 Mbps. Este compatibil cu USB 1.0 și USB 1.1.
USB 3.0 *SuperSpeed* - a intrat pe piață în noiembrie 2008 și oferă viteze de transfer teoretice de până la 5 Gbps.
USB 3.1 *SuperSpeed+* - a apărut în iulie 2013. Este capabil de transferuri de date la viteze maxime teoretice de 10 Gbps, dublu față de USB.
USB 3.2 - lansat în august 2017, introduce două noi moduri *SuperSpeed +* de transfer prin conectorul USB-C cu rate de 10 Gbit/s și 20 Gbit/s.
USB4 - este succesorul USB 3.2, se bazează pe specificația protocolului Thunderbolt 3 și a fost lansat pe 29 august 2019. Suportă un debit de 40 Gbit/s.

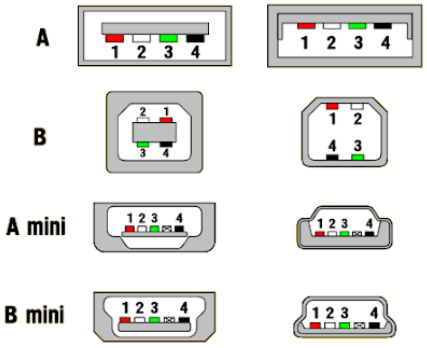
Interconectarea dispozitivelor USB.

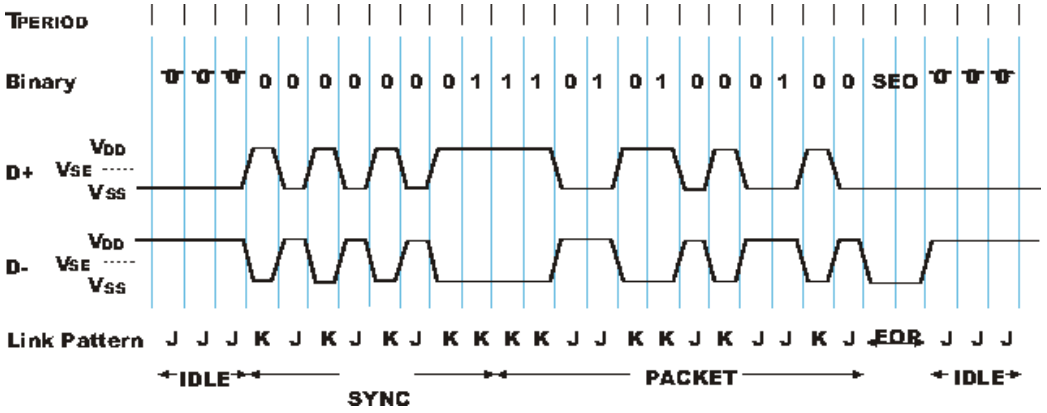


Cablul si conectoare USB.

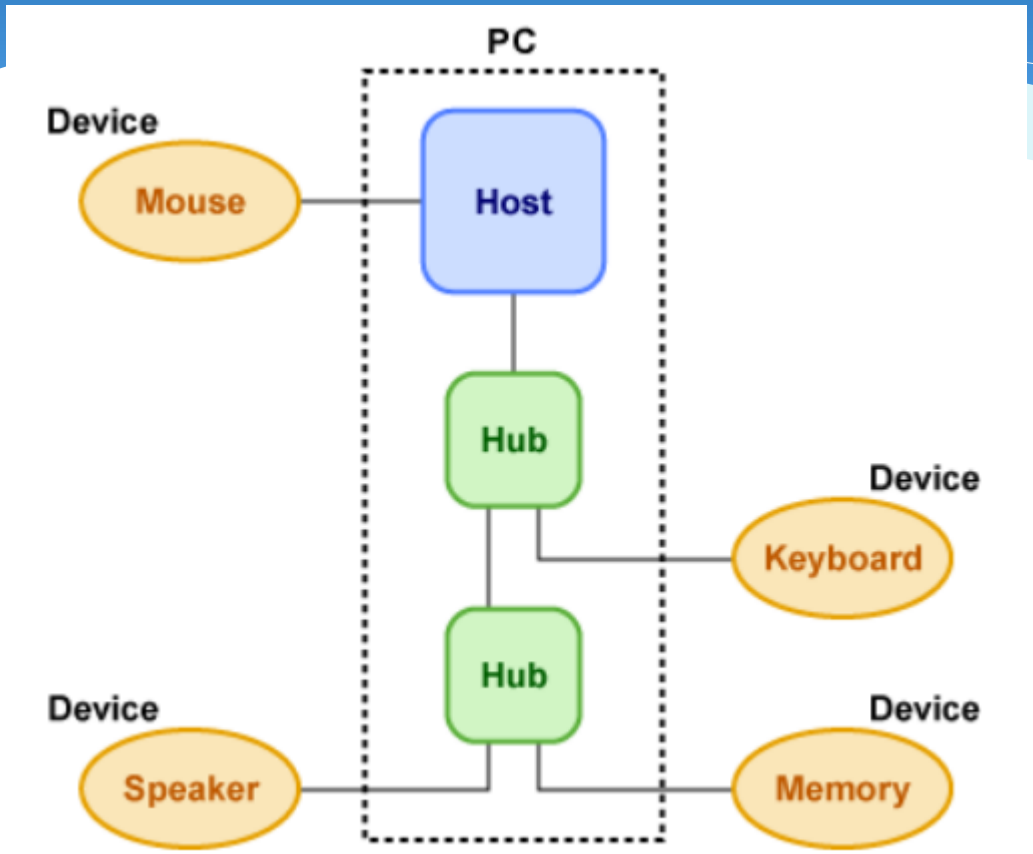


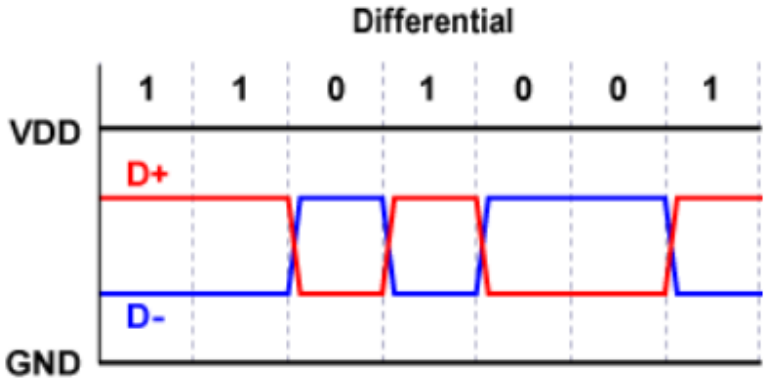




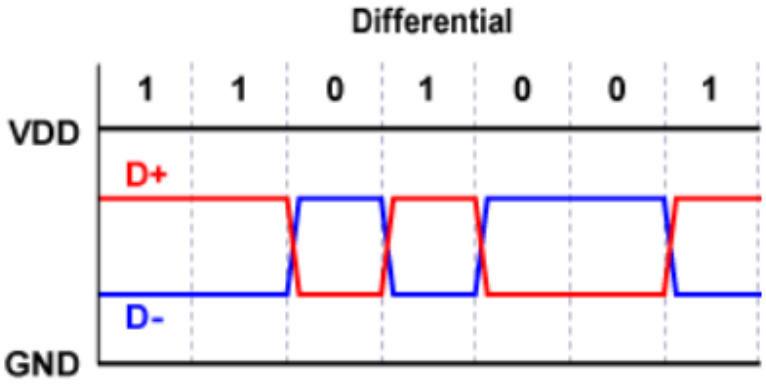


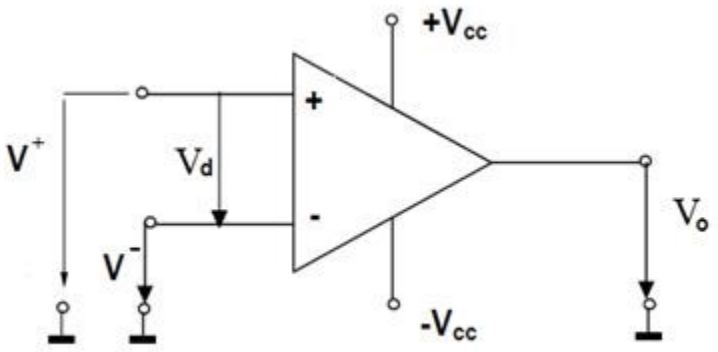
Comunicarea in USB.





Codificarea a in USB.

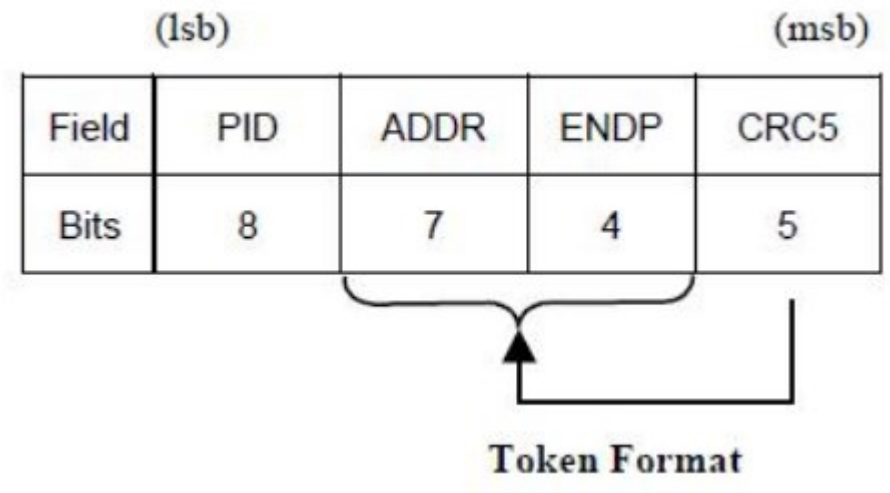




Vo=(V+)-(V-)

Formatul dateor in USB.

Setup packet



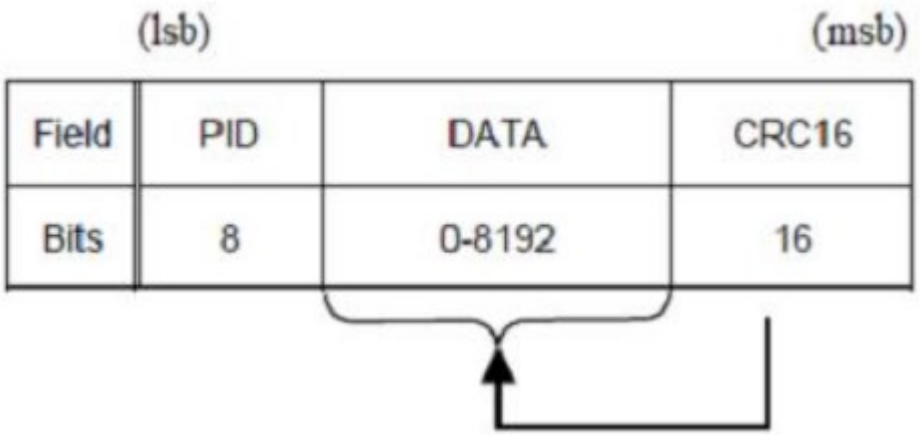
Token Format

Packet ID

The address field specifies. Being 7 = 127 devices.

Cyclic Redundancy Checks

Data packet



End of packet

Convertorul UART in USB pentru Arduino.

UART şi USB:

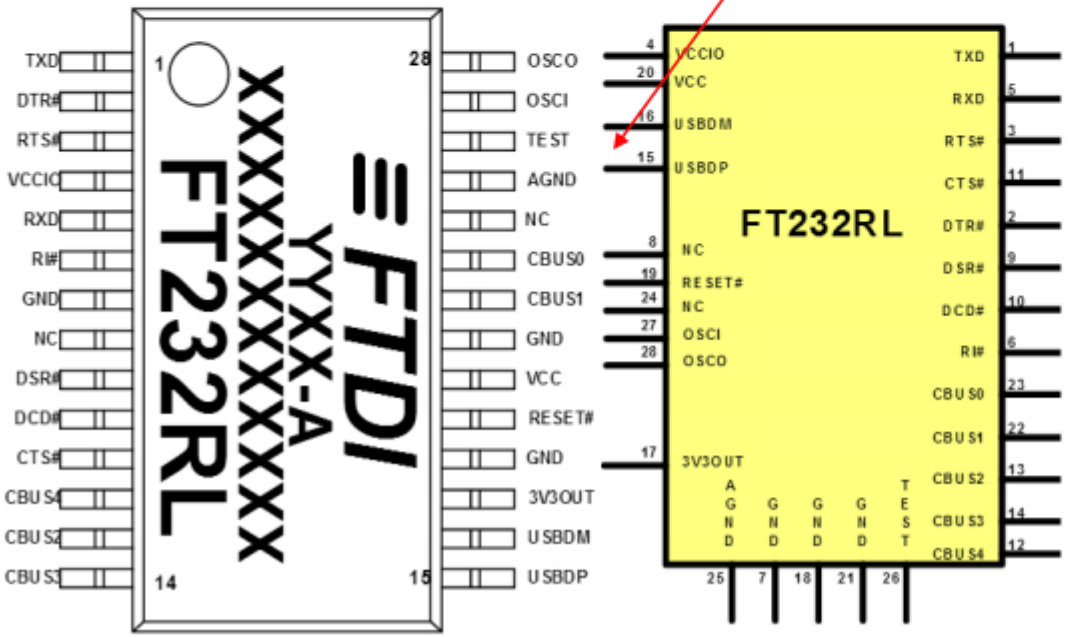
Folosirea unui adaptor FTDI (Future Technology Devices International Ltd)

Arduino Mega foloseşte the FT232RL chip

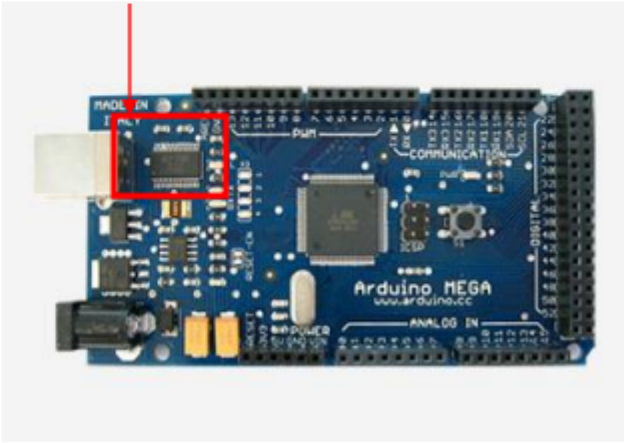
- Vizibil ca un port COM virtual la PC
- Conversie bi-direcţională între USB şi UART

UART Tx şi Rx

USB D+ şi D-



FTDI 232 RL



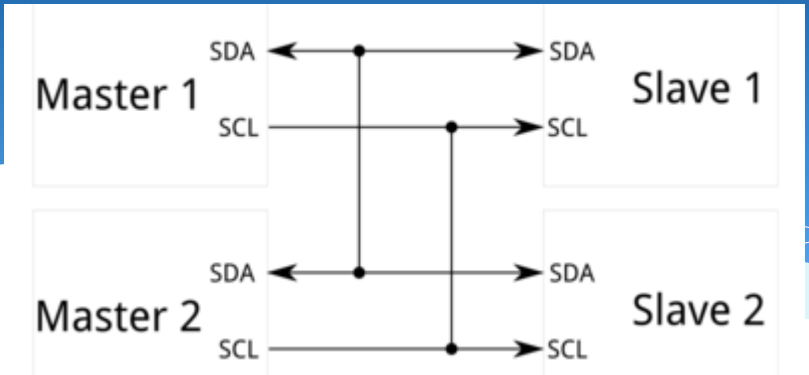
Interfata I2C.

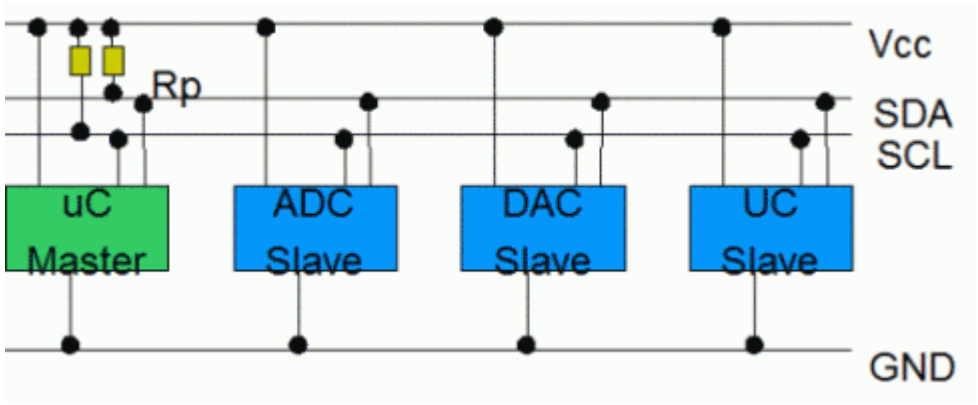
Protocolul Inter Integrated Circuit (I2C) este un protocol creat pentru a permite mai multor circuite integrate “slave” sa comunice cu unul sau mai mute cipuri “master”. Acest tip de comunicare poate fi folosit doar pe distante mici de comunicare si asemenea protocolului UART are nevoie doar de 2 fire de semnal pentru a trimite/primi informatii.

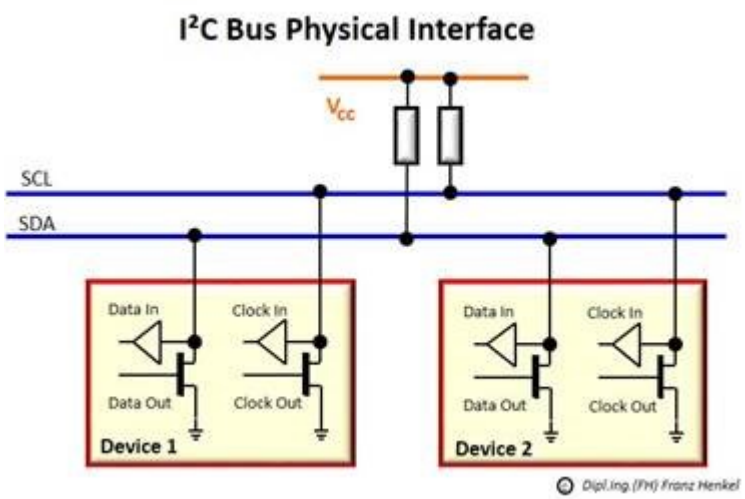
Protocolul I2C a fost dezvoltat in 1982 de *Philips* pentru diferite cipuri *Philips*. Specificatiile originale permiteau comunicarea doar pe 100kHz si doar pe 7 biti de adresa, limitand numarul dispozitivelor conectate la bus, la 112. In 1992 spatiul de adrese a fost extins la 10 biti si comunicarea se realiza pe 500kHz. In prezent exista trei moduri aditionale: *fast plus (1MHz)*, *high-speed (3.4MHz)* si *ultra-fast (5MHz)*.
In plus, in 1995 *Intel* a introdus o noua varianta de I2C, numita *"System Management Bus" (SMBus)*, care este mult mai bine controlata si realizata cu intentia de a maximiza predictabilitatea comunicarii dintre suporturile IC si placile de baza ale PC-urilor. Cea mai notabila diferenta dintre SMBus si I2C este ca prima limiteaza viteza de la 10kHz la 100kHz, iar a doua poate suporta dispozitive de la 0kHz la 5MHz

I2C necesita doua fire care si comunicarea seriala asincrona, cu diferenta ca acest tip de comunicare poate suporta pana la 1008 dispozitive de tip "slave". Mai mult decat atat, in comunicarea de tip I2C pot exista mai multe dispozitive de tip "master" la un bus, lucru nepermis in comunicarea *SPI*.
Rata datelor nu este foarte buna, fiind asemanatoare cu cea de la *portul serial*; cele mai multe dintre *I2C-uri* comunica cu o rata de transmisie cuprinsa intre 100kHz si 400kHz.
In ceea ce priveste implementarea, aceasta este mai complexa decat in cazul implementarii *SPI*, dar mult mai usoara decat in cazul *comunicarii asincrone, cu UART*.

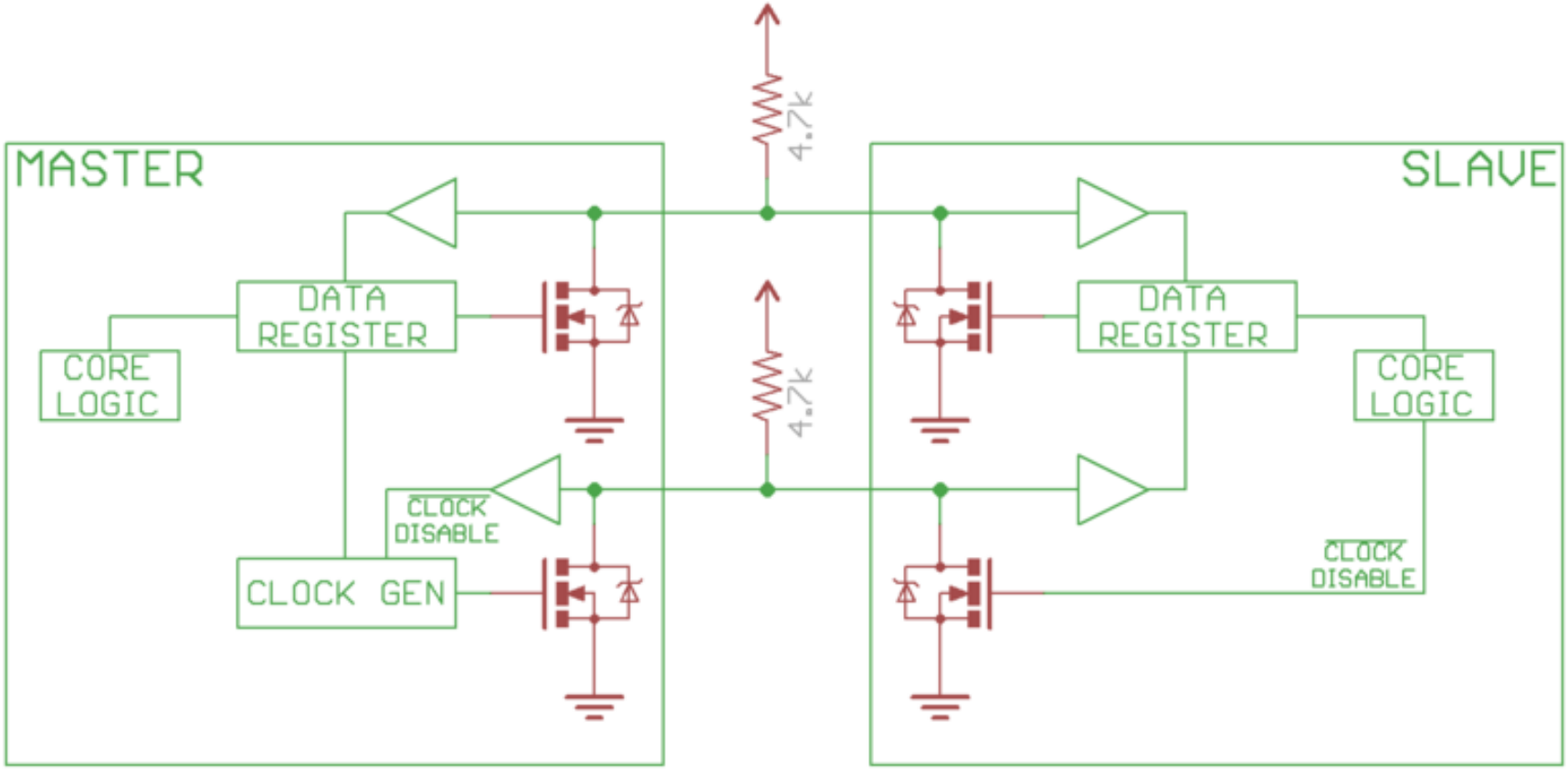
Interfata I2C.



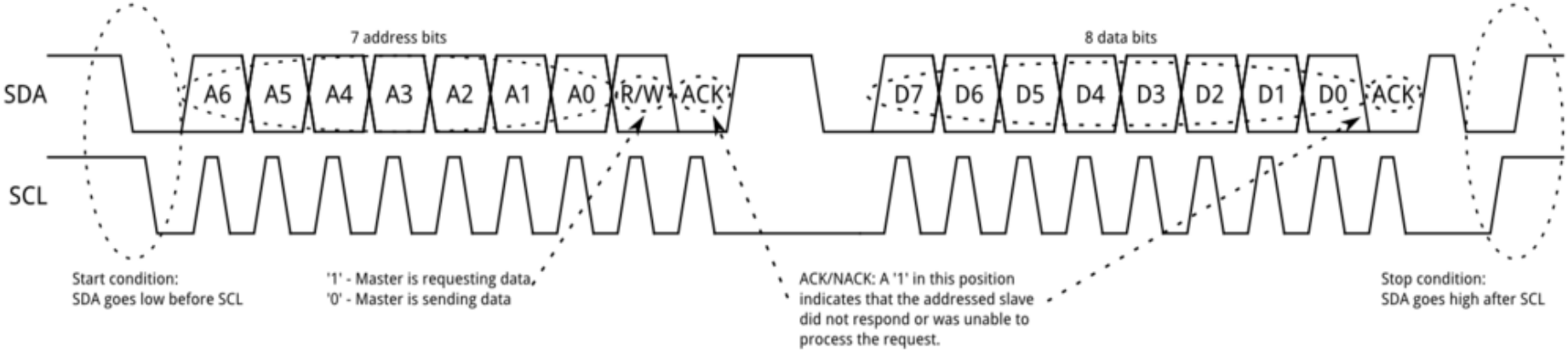


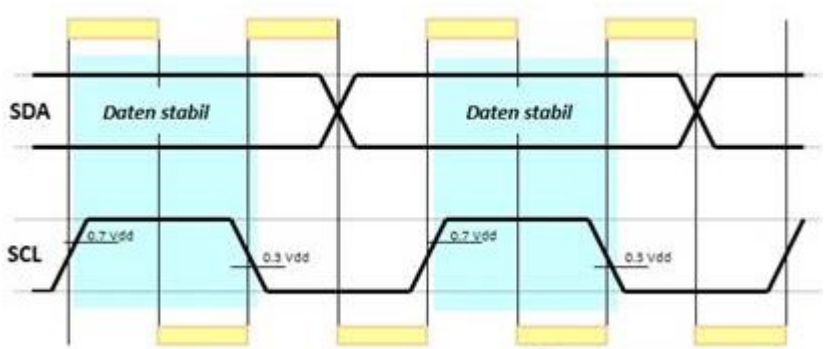


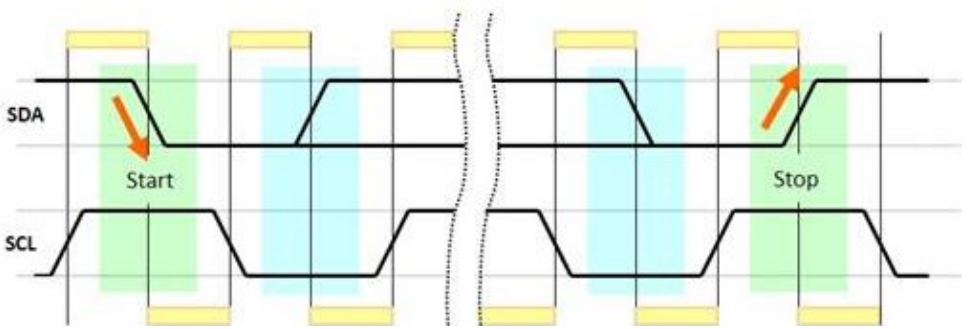
Interfata I2C.



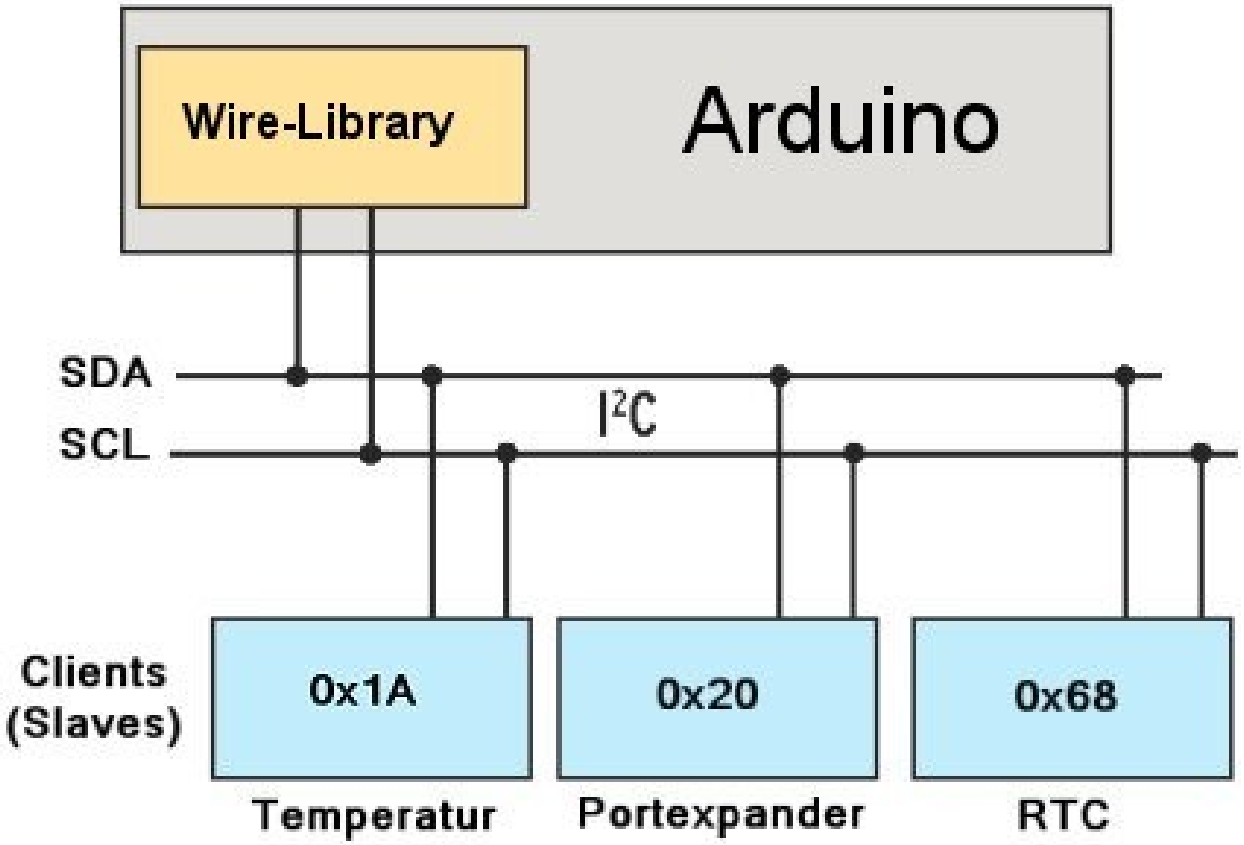
Interfata I2C.







Interfata I2C.



Programarea comunicare dintre un masret si un slave I2C.

```
#include <Wire.h>  /// Master
void setup()
{
 Wire.begin();
}
void loop()
{
 Wire.beginTransmission(1);
 Wire.write('H');
 Wire.endTransmission();

 delay(500);

 Wire.beginTransmission(1);
 Wire.write('L');
 Wire.endTransmission();

 delay(500);

 Wire.beginTransmission(2);
 Wire.write('H');
 Wire.endTransmission();

 delay(500);

 Wire.beginTransmission(2);
 Wire.write('L');
 Wire.endTransmission();

 delay(500);
}
```

```
#include <Wire.h>  /// Slave
const byte slaveId = 1;
void setup()
{
 Wire.begin(slaveId);
 Wire.onReceive(receiveEvent);

 pinMode(13,OUTPUT);
 digitalWrite(13,LOW);
}
void loop()
{
}
void receiveEvent(int howMany)
{
 char inChar;

 while(Wire.available() > 0)
 {
  inChar = Wire.read();

  if (inChar == 'H')
  {
   digitalWrite(13, HIGH);
  }
  else if (inChar == 'L')
  {
   digitalWrite(13, LOW);
  }
 }
}
```

Programarea comunicare dintre un masret si 2 slave I2C.

```
#include <Wire.h>  /// Master
void setup()
{
 Wire.begin();
}
void loop()
{
 Wire.beginTransmission(1);
 Wire.write('H');
 Wire.endTransmission();

 delay(500);

 Wire.beginTransmission(1);
 Wire.write('L');
 Wire.endTransmission();

 delay(500);

 Wire.beginTransmission(2);
 Wire.write('H');
 Wire.endTransmission();

 delay(500);

 Wire.beginTransmission(2);
 Wire.write('L');
 Wire.endTransmission();

 delay(500);
}
```

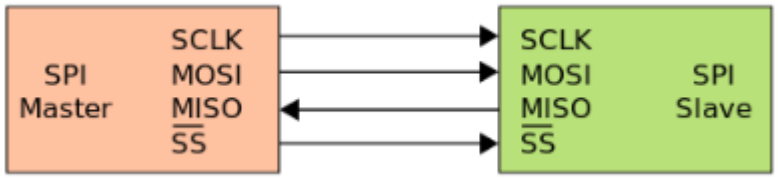
```
#include <Wire.h>  /// Slave 1
const byte slaveId = 1;
void setup()
{
 Wire.begin(slaveId);
 Wire.onReceive(receiveEvent);
 Serial.begin(9600);

 pinMode(13,OUTPUT);
 digitalWrite(13,LOW);
}
void loop()
{
}
void receiveEvent(int howMany)
{
 char inChar;
  while(Wire.available() > 0)
 {
  inChar = Wire.read();
   if (inChar == 'H')
  {
    digitalWrite(13, HIGH);
  }
  else if (inChar == 'L')
  {
    digitalWrite(13, LOW);
  }
  else
  {
    Serial.println("Different from H and L");
 }
}
```

```
#include <Wire.h>  /// Slave 2
const byte slaveId = 2;
void setup()
{
 Wire.begin(slaveId);
 Wire.onReceive(receiveEvent);
 Serial.begin(9600);
}
void loop()
{
}
void receiveEvent(int howMany)
{
 char inChar;

 while(Wire.available() > 0)
 {
  inChar = Wire.read();

  Serial.println(inChar);
 }
}
```

https://profs.info.uaic.ro/~arduino/index.php/Comunicare_I2C

Interfata SPI.

Interfața seriala SPI *(Serial Peripheral Interface)* este o interfața sincronă standard de mare viteză, ce operează în mod full duplex. Numele ei a fost dat de Motorola. Ea e folosită ca sistem de magistrală serială sincronă pentru transmiterea de date, unde circuitele digitale pot să fie interconectate pe principiul master-slave. Aici, modul master/slave înseamnă că dispozitivul (circuitul) digital master inițiază cuvântul de date. Mai multe dispozitive (circuite) digitale slave sunt permise cu *slave select individual*, adică cu selectare individuală.



Serial Peripheral Interface (SPI)

- Comunicare seriala sincrona
- Mod de functionare full duplex
- Configurare Master sau Slave
- Frecventa variabila
- Se poate folosi pentru conexiune intre placi

SPI-ul are patru semnale logice specifice.

- **SCLK** - *Ceas serial* (ieșire din master).
- **MOSI/SIMO** - *Master Output, Slave Input* (ieșire master, intrare slave).
- **MISO/SOMI** - *Master Input, Slave Output* (intrare master, iesire slave).
- **SS** - *Slave Select* (active low, ieșire din master).

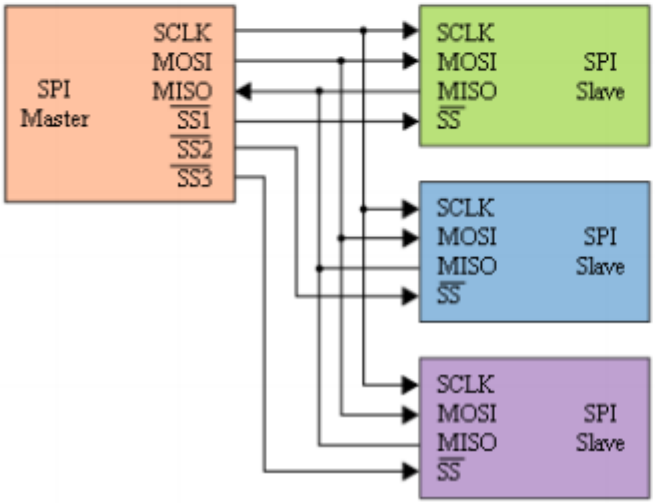
https://profs.info.uaic.ro/~arduino/index.php/Comunicare_I2C

Interfata SPI.

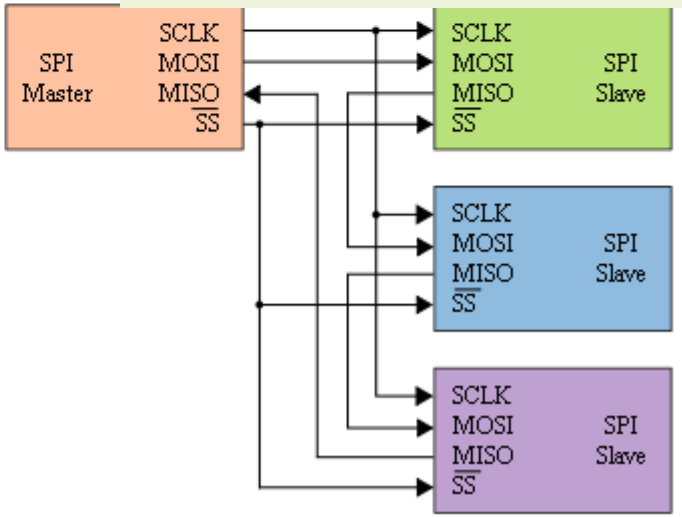
Utilizarea semnalului SS

- Pentru un dispozitiv “slave” **SS** este semnal de intrare
  - SS cu valoare 0 inseamna activarea dispozitivului slave. O tranzitie din 0 in 1 inseamna resetarea ciclului de transfer (marcheaza sfarsitul unui pachet)
  - SS cu valoare 1 – dispozitiv slave inactiv
- Pentru un dispozivit “master” **SS** poate fi:
  - Iesire – prin el activeaza 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 sau “daisy chain”

Accesare selectiva - paralela

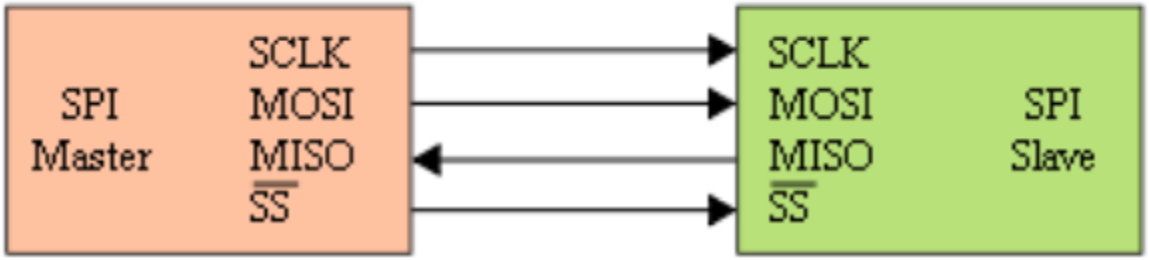


Accesare cu acces in serie



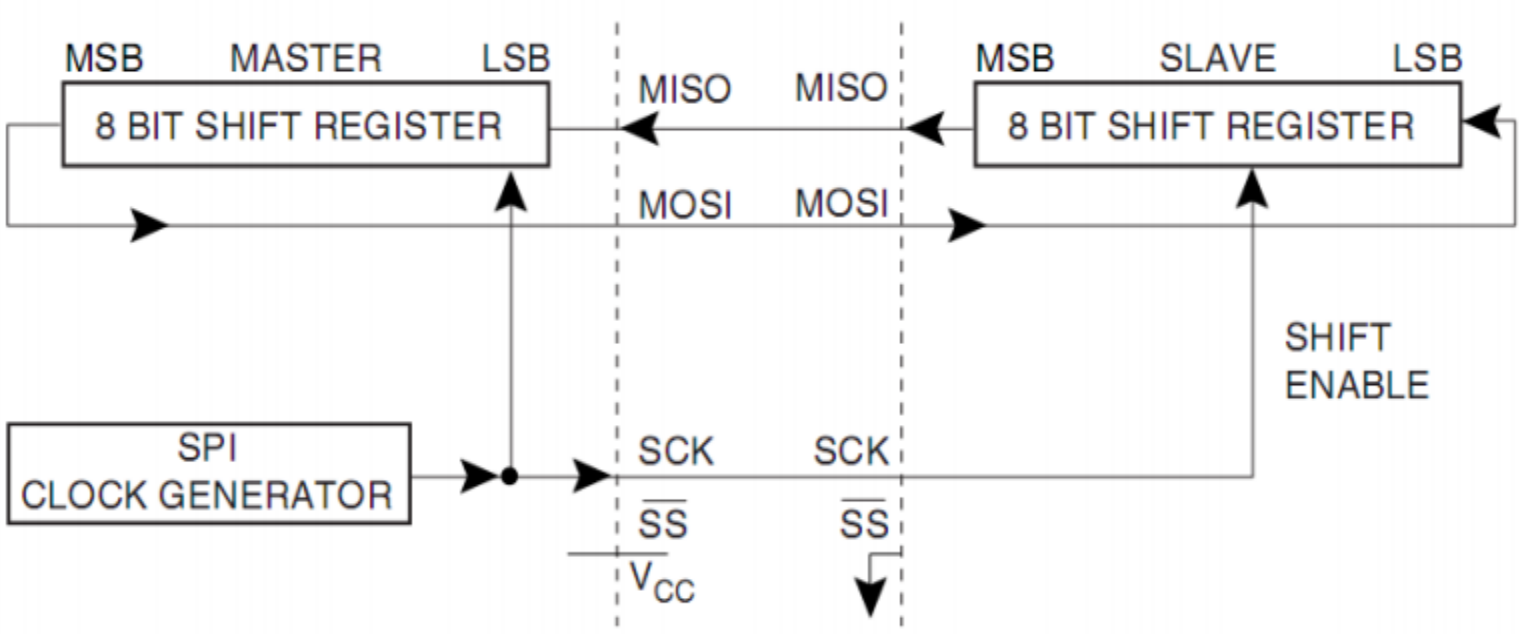
Modul de functionare. Interfata SPI.

- Master initiaza comunicatia prin activarea SS
- Master genereaza semnalul de ceas SCLK
- Pe fiecare perioada de ceas un bit se transmite de la master la slave, si un bit de la slave la master
- Dupa fiecare pachet de date (8, 16 biti,…) SS este dezactivat, pentru sincronizarea transmisiei



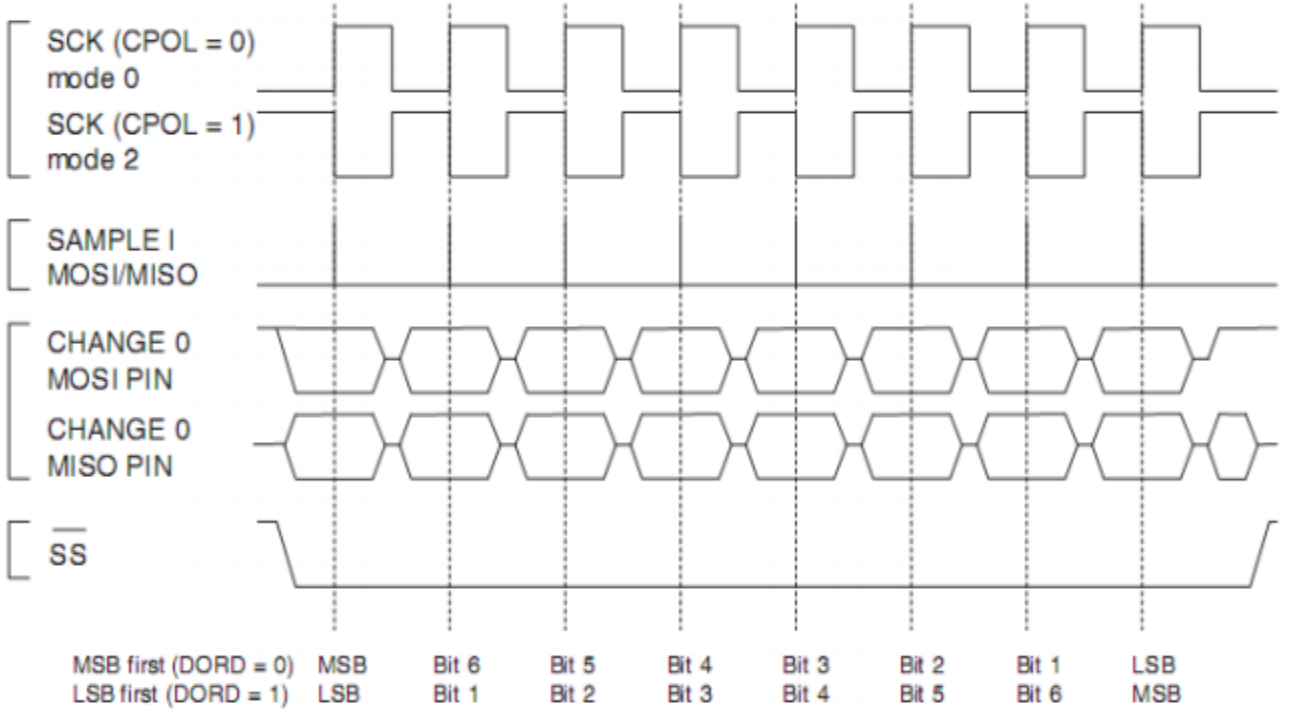
Modul de functionare. Interfata SPI.

- Ambii parteneri au cate un registru de deplasare intern, iesirile si intrarile fiind conectate prin MISO/MOSI
- Ambii registri au acelasi ceas, SCLK
- Cei doi registri formeaza impreuna un registru de rotatie
- Dupa un numar de perioade de ceas egal cu dimensiunea unui registru, Master si Slave fac schimb de date

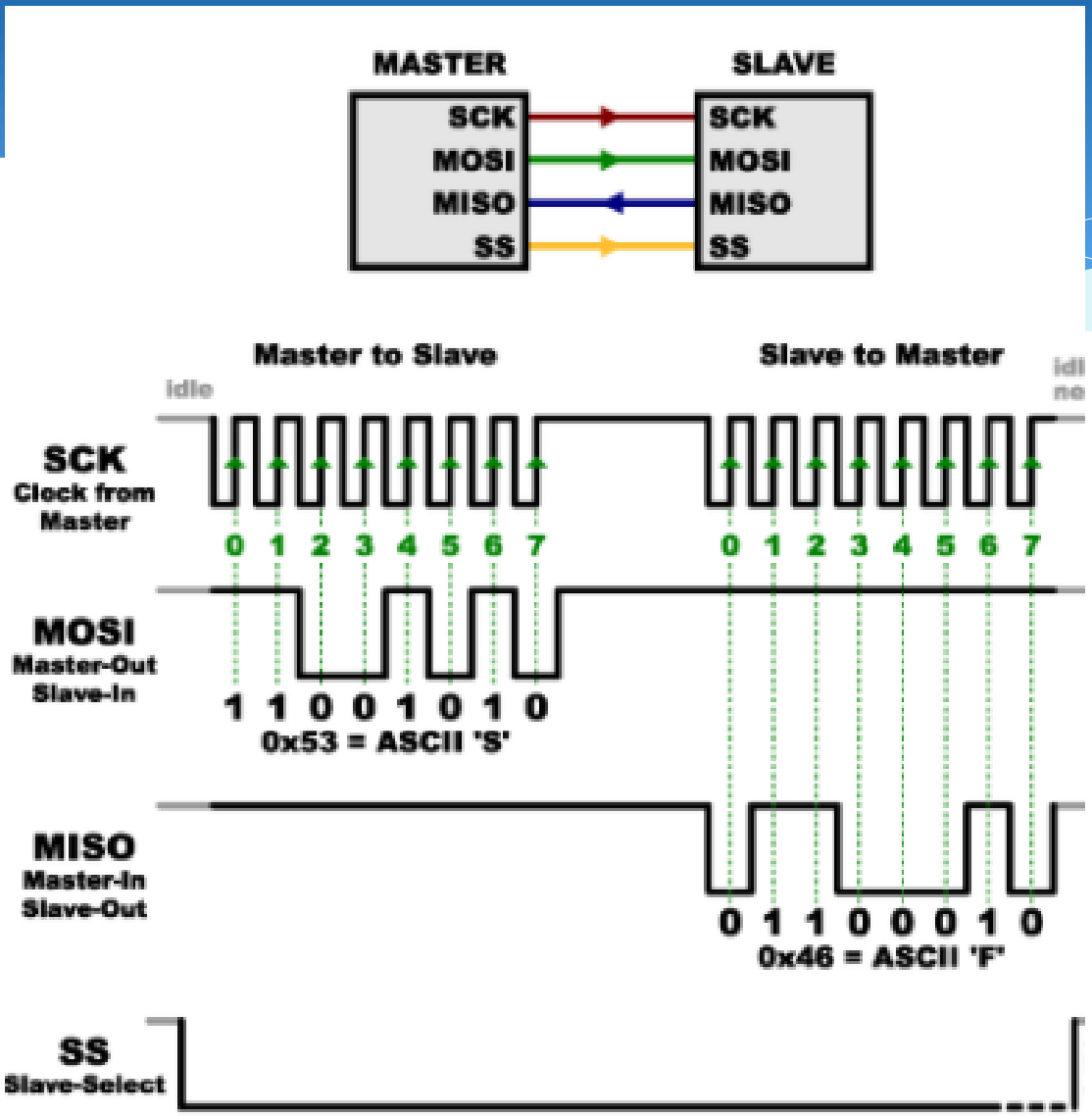


Sincronizarea datelor. Interfata SPI.

- Deplasarea (shiftare) datelor si preluarea lor se fac pe fronturi opuse
- CPOL – clock polarity – primul front e crescator sau descrescator
- CPHA – clock phase
- Pentru CPHA = 0
  - Pe primul front se face preluarea datelor
  - Pe al doilea front se face stabilizarea (deplasarea)



Sincronizarea datelor. Interfata SPI.



Exemple de programare. Interfata SPI.

```
#include <SPI.h>
#include <SoftSPI.h>

// Create a new SPI port with:
// Pin 2 = MOSI,
// Pin 3 = MISO,
// Pin 4 = SCK
SoftSPI mySPI(2, 3, 4);

void setup() {
  mySPI.begin();
  Serial.begin(9600);
  }

void loop() {
  static uint8_t v = 0;

  Serial.print("Sending value: ");
  Serial.print(v, HEX);
  uint8_t in = mySPI.transfer(v);
  Serial.print(" Got value: ");
  Serial.print(in, HEX);
  Serial.println(v == in ? " PASS" : " FAIL");
  delay(1000);
  v++;
}
```

```
// inslude the SPI library:
#include <SPI.h>
// set pin 10 as the slave select for the digital pot:
const int slaveSelectPin = 10;
void setup() {
  // set the slaveSelectPin as an output:
  pinMode(slaveSelectPin, OUTPUT);
  // initialize SPI:
  SPI.begin(); }
void loop() {
  // go through the six channels of the digital pot:
  for (int channel = 0; channel < 6; channel++) {
    // change the resistance on this channel from min to max:
    for (int level = 0; level < 255; level++) {
      digitalPotWrite(channel, level);
      delay(10);   }
    // wait a second at the top:
    delay(100);
    // change the resistance on this channel from max to min:
    for (int level = 0; level < 255; level++) {
      digitalPotWrite(channel, 255 - level);
      delay(10);
    } } }
void digitalPotWrite(int address, int value) {
  // take the SS pin low to select the chip:
  digitalWrite(slaveSelectPin, LOW);
  delay(100);
  //  send in the address and value via SPI:
  SPI.transfer(address);
  SPI.transfer(value);
  delay(100);
  // take the SS pin high to de-select the chip:
  digitalWrite(slaveSelectPin, HIGH);
}
```

Interfata CAN.

Magistrala CAN (Controller Area Network) este o magistrală serială utilizată în industria de automobile, cu scopul de a asigura comunicarea între mai multe microcontrolere fără utilizarea unui calculator-gazdă. Dezvoltată inițial de către firma Bosch, în anul 1983, specificația a fost lansată oficial în anul 1986 (CAN 1.2) și standardizată sub denumirea de ISO 11898. Ulterior, mai mulți producători de semiconductoare (Intel, Philips, Infineon, Texas Instruments, Motorola) au implementat periferice pe bază de CAN. În septembrie 1991, Bosch lansează versiunea a 2-a a specificației (CAN 2.0).

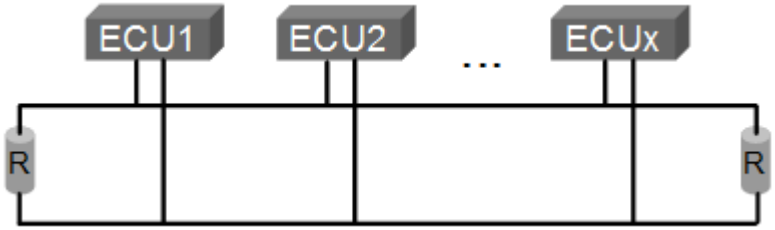
Pe lângă industria de automobile (sisteme de frânare, o gamă largă de senzori, lampi de semnalizare, controlul automat al ușilor) protocolul CAN a început să fie utilizat cu succes și în alte ramuri ale electronicii industriale (echipamente medicale, războaie de țesut).

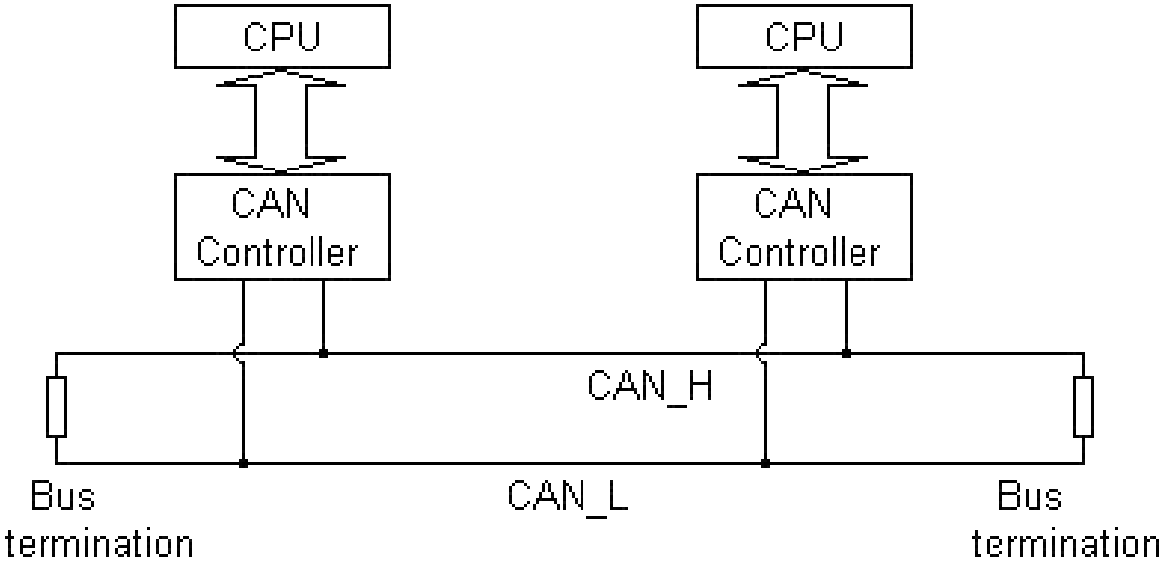
Principial, diferențele dintre versiunea 1.2 și 2.0 a standardului, constau în domeniul de adresare a nodurilor, care a fost extins în noua versiune. Mai exact, CAN 1.2 definește doar un singur tip de mesaj (mesaj standard) având lungimea câmpul de identificare a nodului (Id) de 11 biți, pe când versiunea CAN 2.0 mai introduce, pe lângă tipul de mesaj definit anterior și un mesaj cu lungimea Id-ul de 29 de biți numit mesaj extins.

Interfata CAN.

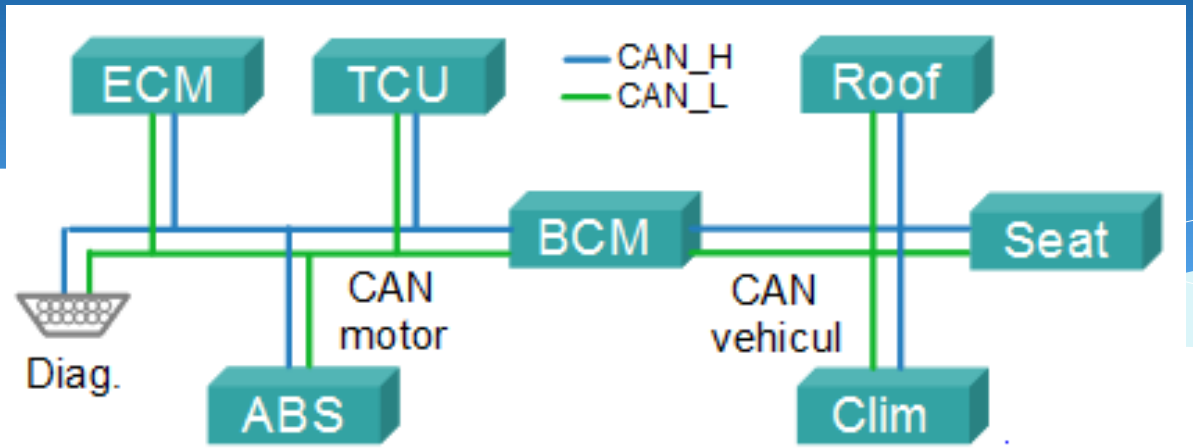
Specificația de CAN definește mai multe nivele:
nivelul fizic - descrie modul de transmitere a semnalului pe magistrală (reprezentare unui bit, nivele de transmisie a semnalelor, aspecte legate de mediul de transmisie)
nivelul transfer - descrie tipurile de mesaje trimise/recepționate de un nod de la nivelul sau superior (obiect); tot în grija acestui nivel țin și aspectele legate de durata unui bit, sincronizare, formatul mesajelor, tehnici de arbitrare, confirmare, detecție de erori precum și mecanisme de restrângere a perturbațiilor
nivelul obiect - se ocupă cu aspecte ce țin de filtrarea și manipularea mesajelor nivelul aplicație
Tipurile de cadre
Transferul de mesaje se manifestă și totodată este controlat prin patru tipuri diferite de cadre:
Cadrul de date (Data Frame) - transportă date de la transmițător la receptor
Cadru de solicitare (Remote Frame)- cadru de solicitare a unui cadru de date
Cadrul de eroare (Error Frame) - transmis de fiecare nod la detecția unei erori pe magistrală
Cadru de supraîncărcare (Overload Frame) - solicită un timp suplimentar între cadrul anterior și cel următor

Nivelul fizic al protocolului CAN.

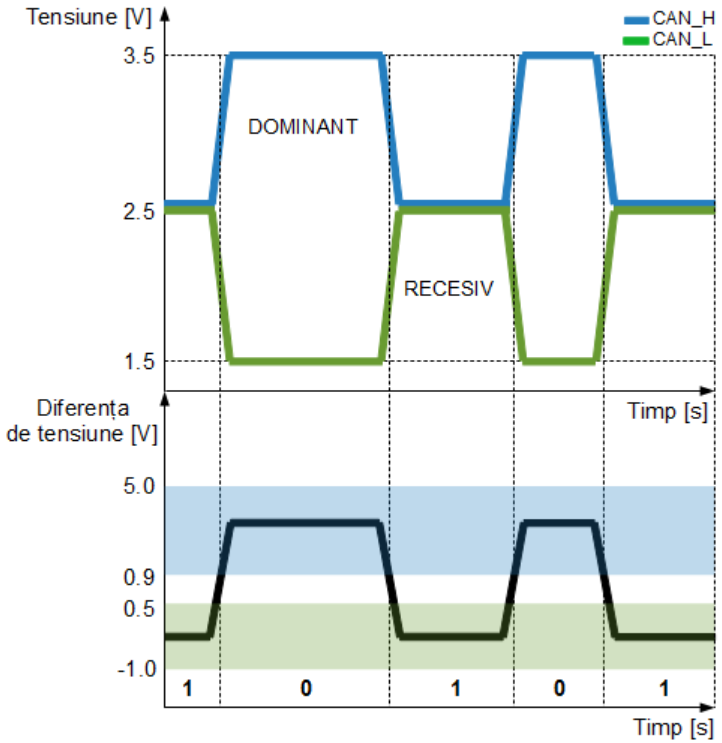




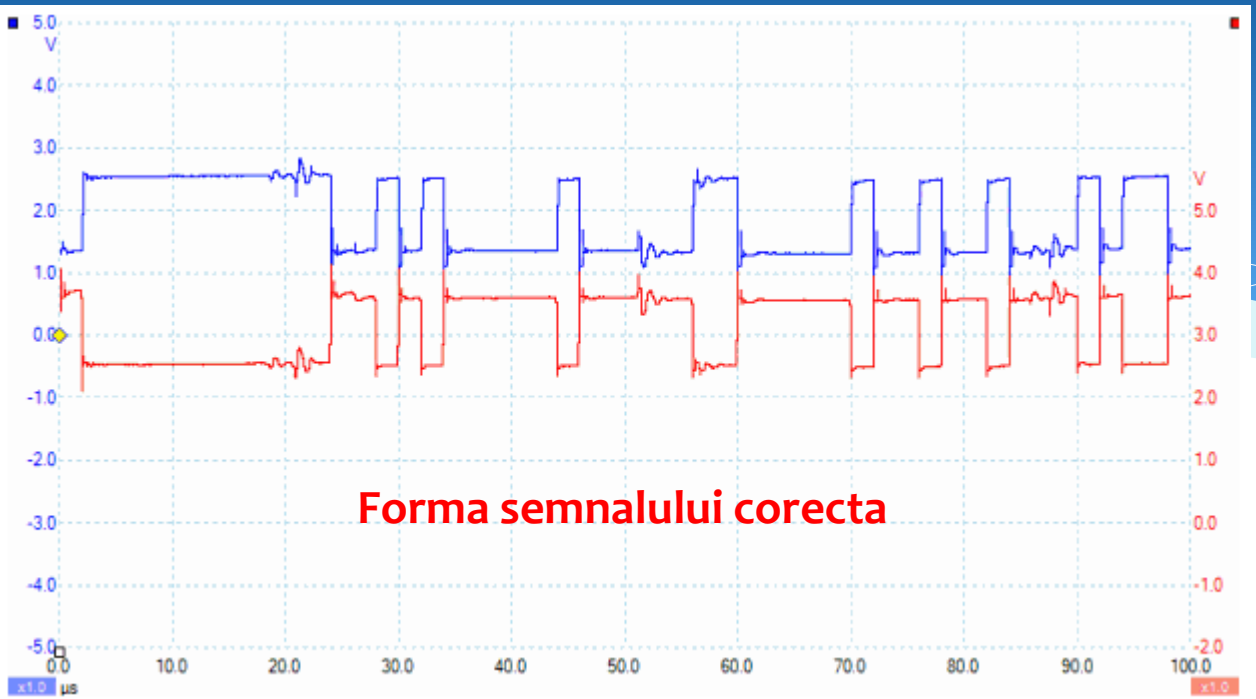
Exemplu de retea CAN.



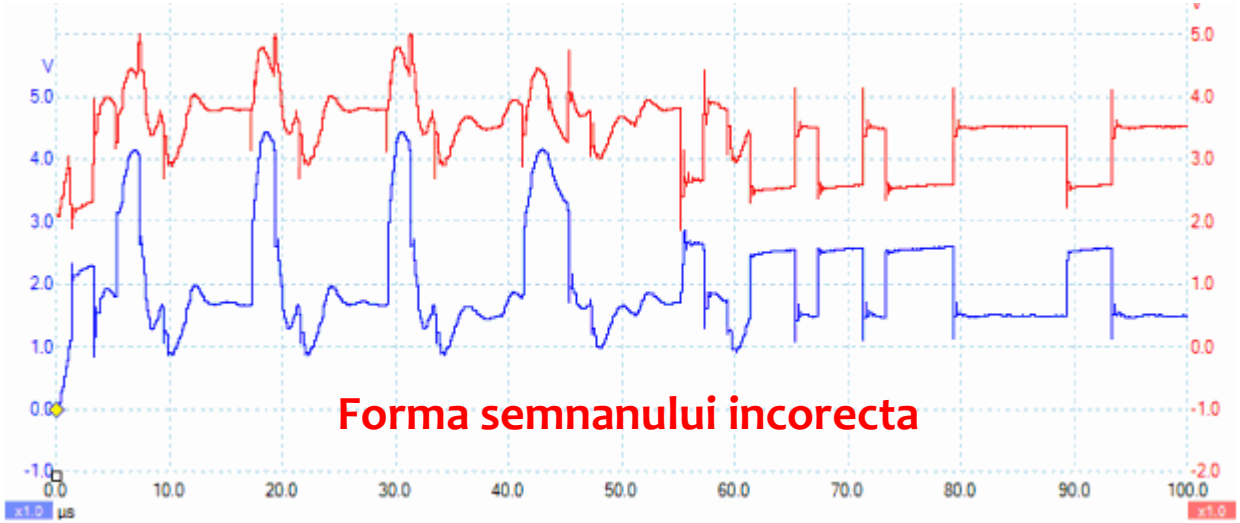
ECM (Engine Control Module) – calculatorul de injecție (motor)
TCU (Transmission Control Unit) – calculatorul transmisiei automate
ABS (Anti-lock Braking System) – calculatorul sistemului de frânare
BCM (Body Control Module) – calculatorul de habitaclu
Roof (Plafon) – calculatorul pentru controlul trapei
Seat (Scaun) – calculatorul pentru controlul scaunelor
Clim (climatizare) – calculatorul pentru controlul climatizării
Diag. (diagnostic) – conectorul de diagnosticare



Fiabilitatea retelei CAN.



Forma semnalului corecta



Forma semnanului incorecta

Interfata IrDA.

Tehnologia IrDA (Infrared Data Association) definește un set de standarde al tehnologiei fără fir bazat pe comunicații prin infraroșu de transmitere și recepție de date pe distanță mică. IrDA a fost fondată la 28 iunie 1993 de un grup format din 50 de companii cu scopul de a standardiza comunicațiile în infraroșu.

Tehnologia IrDA este implementată în dispozitive portabile precum smartphone, laptop, PDA, camere video, periferice, telecomenzi, aparate medicale și industriale, auto etc. Primul standard, bazat pe portul serial RS-232 a fost aprobat în 1994. Acest standard folosește specificațiile portului serial, aceeași structură de date dar și limitele de viteză. În 1995 a fost aprobat un nou standard cu limita de viteză la 1Mbps.

În principal, tehnologia IrDA utilizează modulația ASK (Amplitude-shift keying) care este o formă de modulație a amplitudinii unui semnal analogic cu un semnal modulator reprezentat de un flux de biți.
Caracteristicile principale ale acestui tip de comunicații wireless, sunt transferul de date securizat, și rata de eroare foarte redusă, ceea ce îl face foarte eficient.
Dispozitivele IrDa folosesc LED-uri infraroșii pentru a emite radiație infraroșie care este focalizată într-o rază îngustă. Raza este modulată, pornită sau oprită, pentru a codifica datele. Receptorul folosește o fotodiodă pentru a converti radiația infraroșie în curent electric.
Pentru ca dispozitivele să comunice, trebuie să fie așezate în linie unul cu celălat pe direcția de transmisie a luminii de infraroșu.
Suita de protocoale IrDA are rate de transfer de până la 16 Mbps pe o distanță în jur de aproximativ 1m, iar la distanța de 5m rata de transfer scade foarte mult, ajungând în jurul valorii de 75 kbps.

Interfata IrDA.

Infrared Data Association

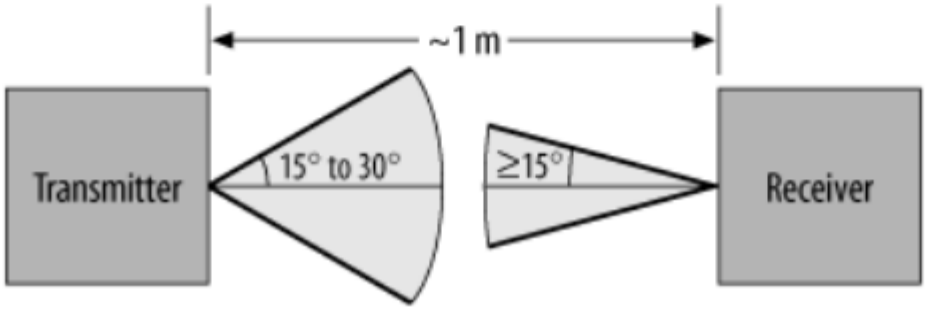
Principalul obiectiv este de a furniza comunicaţii pe distante scurte intre 2 dispozitive

Comunicatie 1 la 1

Standardul initial (v1.0) suporta viteze de transfer intre 2400 si 15200 bps, pe distante de aproximativ 1m.

Comunicatia initiala se realizeaza cu viteza de 9600 bps si dispozitivele negociaza o rata de comunicatie pentru transfer, mai mare sau mai mica (depinzand de capabilitatile fiecarui dispozitiv)

Standardul a fost extins sa suporte viteze de comunicatie mai mari, de 1.152 Mbps si 4 Mbps.



Codificare IrDA.

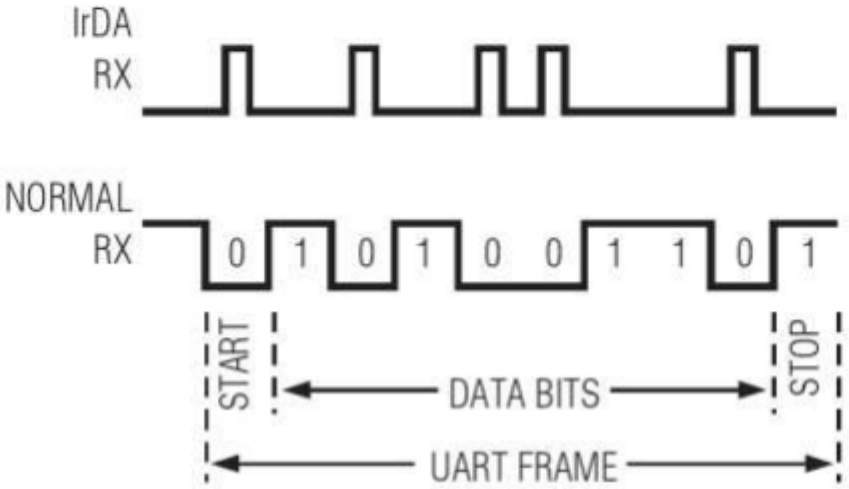
IrDA foloseste o schema de codare Return-to-Zero (RZ).
In codarea RZ, un cadru este format dintr-un interval de transmisie care este divizat in subintervale reprezentand biti individuali.
Un zero logic este reprezentat de un impuls cu durata de 3/16 din latimea intervalului pentru un bit
Un unu logic este reprezentat prin absenta unui puls

Impulsul este mai scurt, pentru a economisi energie
Transmiterea unui sir de zerouri => impulsuri de sincronizare

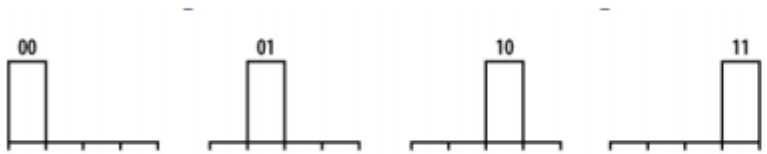
Codificare IrDA.

Pentru orice baud, pulsul de lumina poate fi ingust de 1.7 microsecunde (pt “0“)

1.7 us == 3/16 din perioada unui bit la 115200

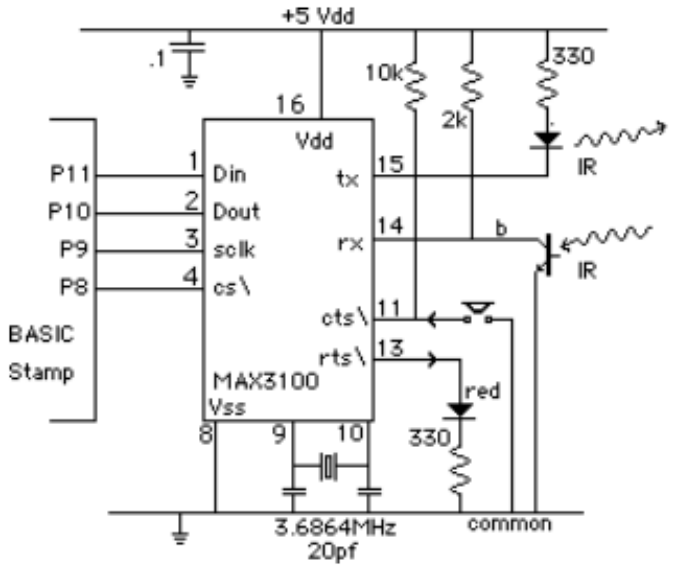


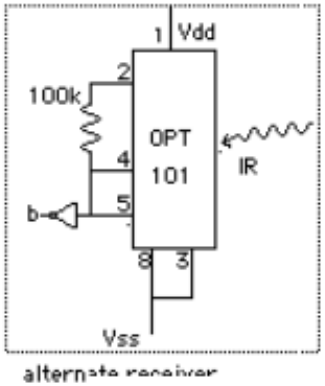
Codificare IrDA.



- La viteze de 4 Mbps se folosete PPM (Pulse Position Modulation):
- pozitia pulsului variaza (4 coduri)

Scheme de conectare IrDA.

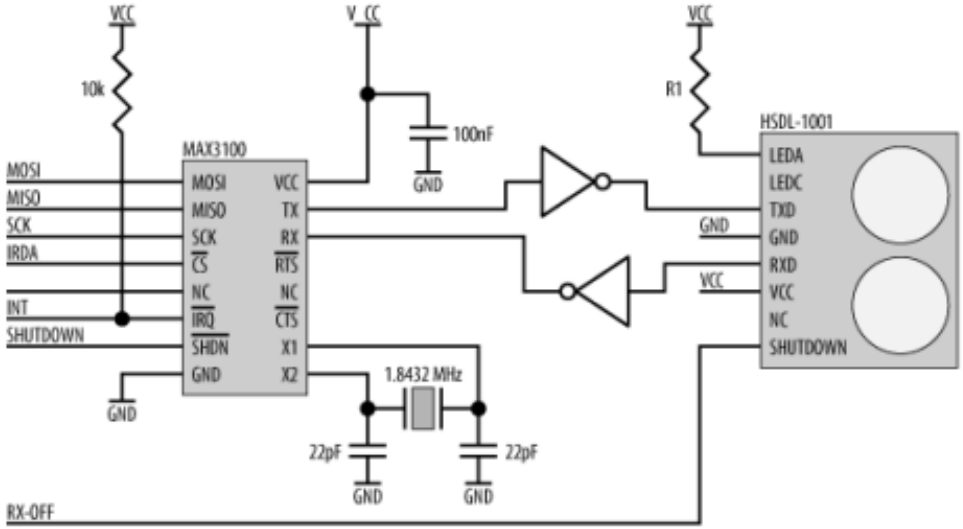




Q: 1.8432 MHz
(multiplu de 115200 =115200x16, pentru a realiza impulsul de 3/16)

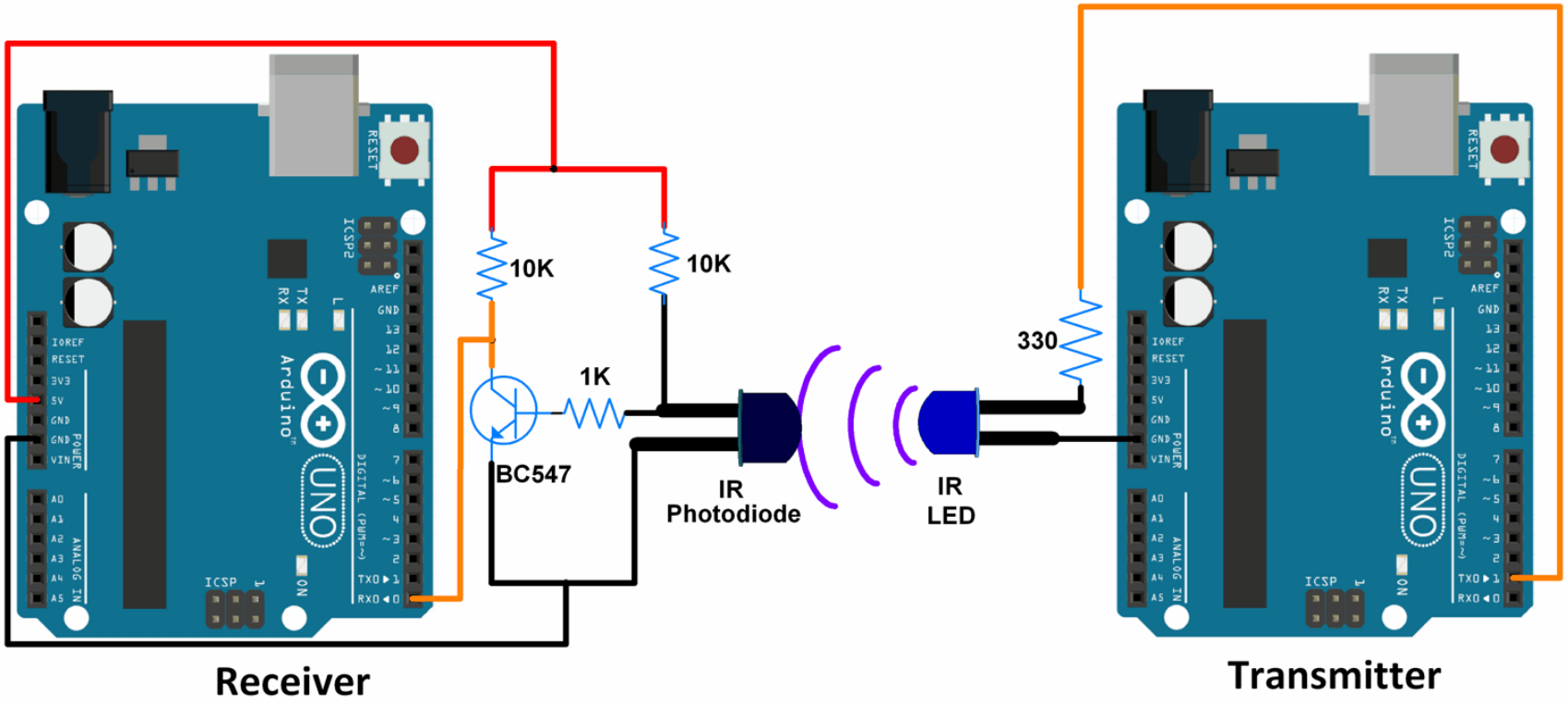
R1 – reglaj curent LED IR

V=2.1, I=250mA



Pentru MAX3100 interfata de comunicatie cu uC e SPI,

Scheme de conectare Arduino IrDA.



Programarea conectarea in Arduino IrDA.

Transmite IrDA

```
void setup() {
  Serial.begin(9600);            /* Define
baud rate for serial communication */
}

void loop() {
  int count;
  for(count = 0; count<100; count++)
  {
    Serial.println(count);
    delay(1000);
  }
}
```

Receptie IrDA

```
void setup() {
  Serial.begin(9600);            /* Define baud
rate for serial communication */
}

void loop() {
if(Serial.available())           /* If data is
available on serial port */
  {
    Serial.print(char(Serial.read()));      /* Print
character received on to the serial monitor */
  }
}
```

Interfata Bluetooth.