Laborator 5 – Interfețe de comunicație

Electronica embedded se refera la interconectarea de circuite (procesoare sau alte circuite integrate) cu scopul de a crea un sistem simbiotic. Pentru ca aceste circuite să-şi poată transfera informații trebuie sa conțină o modalitate de comunicare comună. Deşi exista sute de modalități de comunicare, aceste modalități se împart în doua categorii: modalități **seriale si paralele**.

Interfețele paralele transferă mai mulți biți în același timp. De obicei au nevoie de magistrale de date (bus) care transmit pe 8, 16 sau mai multe linii. Datele transmise si recepționate sunt fluxuri masive de 1 si 0. În figura 1, observăm o magistrală de date cu lățimea de 8 biți, controlată de un semnal de ceas şi transmite câte un byte la fiecare puls al ceasului (se folosesc 9 fire).

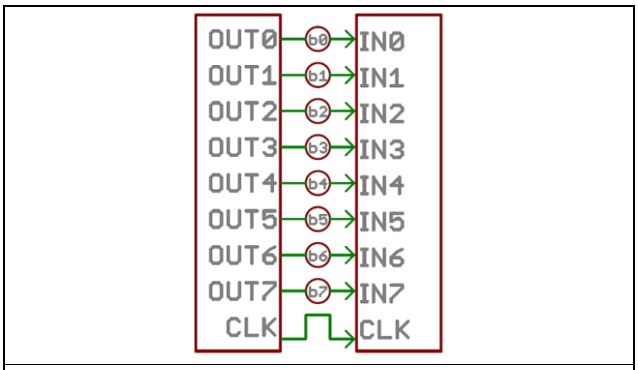

Fig. 1. Transmisie paralelă

Interfețele seriale trimit informația bit cu bit. Aceste interfețe pot opera doar pe un singur fir si de obicei nu necesita mai mult de 4 fire (minim 1, maxim 4).

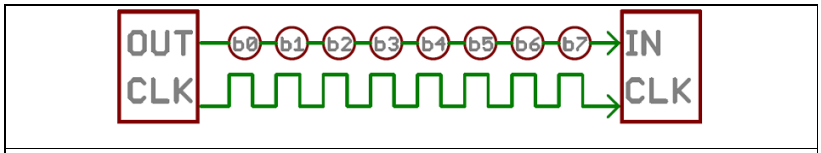

Fig 2. Transmise serială

Mai sus, în figura 2, se poate observa un exemplu de interfața care transmite cate un bit la fiecare impuls de ceas (aici doar 2 fire sunt folosite). Deşi protocoalele de comunicție paralelă au beneficiile lor, acestea necesita un număr mare de pini din partea platformei de dezvoltare pe care o folosesc si astfel, având în vedere ca numărul de pini de pe Arduino UNO/ Mega e redus ne vom concentra pe interfețe de comunicație serială.

O alta modalitate de clasificare a interfețelor de comunicație este după modalitatea de comunicație: **sincronă** sau **asincronă**. O interfața de comunicație sincronă folosește un semnal de ceas unic la ambele capete ale comunicației (emițător şi receptor). Aceasta modalitate de comunicație este de multe ori mai rapidă, cu toate astea are nevoie de cel puțin un fir in plus intre dispozitivele care comunica (pentru transmiterea semnalului de ceas). Exemple de astfel de comunicații sunt SPI şi I2C. Transferul asincron se refera la faptul ca

datele sunt transferate fără suportul unui semnal de ceas extern. In acest fel se elimina firul de ceas, dar o atenție sporita trebuie acordata sincronizării datelor transferate.

Regulile transferului serial asincron

Transferul serial asincron conține un număr de mecanisme care asigură transferul robust şi fără erori. Acest mecanism, care a fost construit pentru a evita semnalul de ceas extern conține:

- Rata de transfer (baud rate)
- Pachetul de date (data frame)
- Biții de date – caracter (data chunk)
- Biții de sincronizare (synchronization bits)
- Biții de paritate (parity bits)
- Linia de date (**care in stare inactiva este la nivel logic „1”**)

Partea critică este asigurarea ca ambele dispozitive care folosesc magistrala serială să folosească acelaşi protocol.

Baud Rate

Baud Rate-ul ne spune cât de rapid sunt transmise datele pe linia serială. Aceasta mărime este exprimată in stari pe secunda (de obicei o stare este 1 sau 0, deci un bit, dar exista interfete care pot avea mai mult de doua stari, si atunci baud rate nu este acelasi lucru cu biti pe secunda). Exista mai multe baud rate-uri standard precum 1200, 2400, 4800, 19200, 38400, 57600, sau115200.

Pachetul de date (Data Frame)

Fiecare bloc (de obicei un octet) de date, care urmează sa fie transmis este trimis într-un pachet (frame) de biți. Pachetele sunt create adăugând biți de sincronizare sau paritate datelor care urmează sa fie transmise. Figura 3 ilustrează aspectul pachetului de date.

Fig. 3. Pachet - Data Frame

Data chunk

Partea cea mai importanta a fiecărui pachet o reprezintă datele pe care le conține pachetul. Acest pachet mai este numit si data chunk (bucata de date), întrucât dimensiunea bucății nu este întotdeauna fixă. Îl vom denumi in continuare ”**caracter**” de date. Cantitatea de informație din fiecare pachet poate fi intre 5 si 9 biți (datele standard sunt pe 8 biți). După ce se decide cât trebuie sa fie dimensiunea datelor, ambele dispozitive care comunică trebuie sa fie de acord cu **endianness-ul (care bit este transferat primul, cel mai semnificativ (msb) sau cel mai putin semnificativ (lsb))**

Biți de sincronizare

Biții de sincronizare sunt biți speciali care sunt transferați cu fiecare caracter de date. Aceștia sunt biții de **start** si de **stop**; ei marchează începutul si finalul unui pachet. Tot timpul bitul de start este unul singur iar biții de stop sunt fie unu fie doi. Bitul de start este tot timpul indicat de o linie de date inactive care trece de la 1 la 0 pe când biții/bitul de stop va merge înapoi la starea inactiva ținând linia la nivelul 1 logic.

Biți de paritate

Biții de paritate asigură un tip rudimentar de control al erorii. Paritatea poate fi „impară" (odd) sau „pară"(even). Pentru a produce bitul de paritate toți biții din caracterul de date sunt compuși cu operatorul „sau exclusiv" și paritatea rezultatului ne spune daca bitul este setat sau nu. Paritatea este o măsura de verificare opțională, care nu este prea folosita. Este util să folosim biții de paritate atunci când transmitem date in medii cu zgomote.

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

Fig. 4. Compunerea bitului de paritate

O magistrală serială asincronă conține doar 2 fire – unul pentru trimiterea datelor, iar celălalt pentru recepția lor. Așadar componentele care doresc sa comunice serial vor trebui sa aibă 2 pini: pinul de recepție (**RX**), si pinul de transmisie (**TX**), așa cum se poate observa și în figura 5.

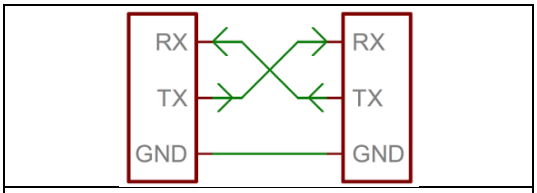

Fig. 5. Schema de conexiune serială

Toate plăcile Arduino conțin cel puțin un port **Serial** (cunoscut ca și UAART sau USART). Comunicarea serială se poate realiza prin pinii 0 (RX) si 1(TX), dar și prin USB (interfața USB comunică cu microcontrollerul prin pinii RX0 și TX0). Din acest motiv, pinii digitali 0 și 1 nu trebuie niciodată utilizați pentru aplicații utilizator, pentru că pierderea controlului asupra lor înseamnă pierderea controlului asupra programării plăcii.

Placa Arduino Mega conține 3 porturi seriale adiționale **Serial1** pe pinii 19 (RX) și 18 (TX), **Serial2** pe pinii 17 (RX) și 16 (TX), **Serial3** pe pinii 15 (RX) și 14 (TX).

In primul exemplu din laborator vom face o comunicație serială între Arduino și PC si vom afișa pe LCD mesajul transmis de pe PC. Pentru acest exemplu vom folosi LCD shield-ul montat pe placa de dezvoltare.

În acest exemplu informația este transmisă de la computer la Arduino și afișata pe LCD. Exista o varietate de funcții pentru manipularea datelor seriale. In laboratoarele precedente am folosit o comunicație serială între Arduino si computer, când afișam starea butoanelor apăsate. Funcțiile/metodele cele mai uzuale pentru manipularea seriala a datelor sunt prezentate in figura 6 (https://www.arduino.cc/en/Reference/Serial).

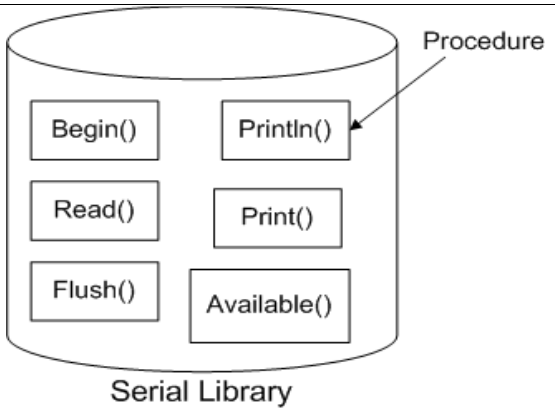

Fig. 6. Funcţii pentru comunicaţia serială asincronă

Funcţiile **print** şi **println** ale clasei Serial trimit date pe portul serial. Diferenţa este ca prinln() adaugă un caracter rând nou (‘\n’) si un caracter „carriage return” (‘\r’) la finalul mesajului transmis. Pentru numere transmise puteţi specifica si un format de transmitere a datelor (HEX, DEC etc.).

Funcţia **begin()** setează viteza de comunicaţie, in biţi/secunda (baud rate). Pentru comunicaţia cu computerul se folosesc în general următoarele viteze (300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200). Se mai poate adaugă un parametru opţional pentru configurarea pachetului de date: câţi biţi sunt, paritatea si numărul biţii de stop. Implicit (daca nu se specifica parametrul de configurare opţional) sunt setate următoarele valori: **8 biti de date, no parity, one stop bit.**

Funcţia **read()** citeşte datele venite prin interfaţa seriala. Sintaxa este următoare: **IncomingByte = Serial.read();**

Funcţia **write()** trimite un octet sau o succesiune de octeţi. Pentru a trimite totuşi numere se recomanda folosirea funcţiei **print()**.

Instrucţiunea **flush()** aşteaptă ca transmiterea serială de date să se finalizeze.

Funcţia **available()** întoarce numărul de octeţi care pot fi citiţi de la portul serial. Aceste date au ajuns deja şi sunt stocate în bufferul de recepţie serială.

O funcţie utila pe care o vom folosi este **serialEvent().** Funcţia este definita de utilizator, şi va fi apelată în mod automat în momentul în care apar date pentru a fi citite.

In exemplul de jos se citesc date venite pe interfaţa serială (de la PC prin serial monitor) şi se afişează pe LCD.

```
//includem libraria de LCD
#include <LiquidCrystal.h>
String inputString = "";         // cream un string care sa ne tina datele care vin pe serial
// conditie pentru verificare daca stringul este complet (s-a apasat enter)
```

```
boolean stringComplete = false;
//initializam obiectul de tip lcd (vezi exemplul 1 pentru explicatii asupra piniilor)
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);

void setup() {
  // initializare interfata seriala
  Serial.begin(9600); // format frame serial implicit
  // initializare si setare lcd
   lcd.begin(16, 2);
  // rezervam 200 de octeti pentru sirul de preluare a datelor de intrare
  inputString.reserve(200);
}

void loop() {
  // afisam stringul cand primim new line
  if (stringComplete) {
 // setam cursorul la coloana si randul 0
    lcd.setCursor(0, 0);
    lcd.print(inputString);
    Serial.println(inputString);
    // golim sirul
    inputString = "";
  // resetam contorul care verifica daca stringul este sau nu complet
    stringComplete = false;
  }
}

/*
 SerialEvent este apelata de fiecare data cand date noi ajung pe portul RX
Aceasta functie ruleaza de fiecare data cand ruleaza si loop. Deci daca am pune un delay in
loop ne-ar intarzia si afisarea rezltatului.
*/
void serialEvent() {
  while (Serial.available()) {
    // luam byte-ul nou venit:
    // il citim cu serial.read
    char inChar = (char)Serial.read();
    // verificam daca nu e cumva new line si daca nu e il adaugam in inputString
    // nu adaugam new line in input string intrucat ne va afisa un character in plus pe lcd
    If (inChar != '\n')
    inputString += inChar;
    // daca caracterul care vine este new line setam flagul
    // in asa fel incat loop-ul principal poate face ceva in legatura cu asta
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
```

Pentru transmiterea de date către Arduino folosiți programul Serial Monitor, deschis din meniul Tools.

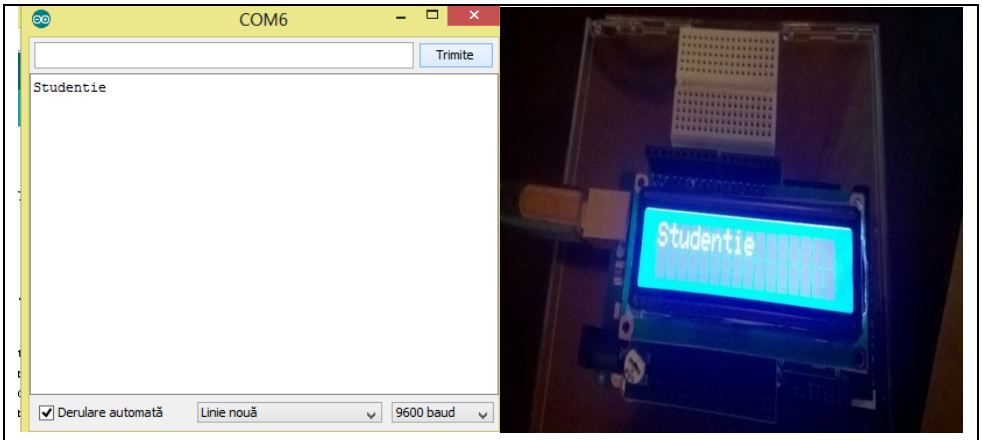

Fig. 7. Transmitere date prin intermediul Serial Monitor

Protocolul Inter-Integrated Circuit (I2C)

Protocolul **Inter Integrated Circuit (I2C)** e un protocol care a fost creat pentru a permite mai multe circuite integrate "slave" sa comunice cu unul sau mai mute cipuri "master". Acest tip de comunicare a fost intenționat pentru a fi folosit doar pe distante mici de comunicare si asemenea protocolului UART sau RS232 are nevoie doar de 2 fire de semnal pentru a trimite/primii informații.

Întrucât porturile seriale sunt asincrone, componentele care le folosesc trebuie sa se înțeleagă din timp referitor la rata de transfer a datelor. De asemenea, componentele care comunică trebuie sa aibă semnale de ceas cu rate similare, altfel datele primite vor fi amestecate. Sa nu uităm ca la fiecare transfer serial se transmit 10 biți într-un frame daca am avea 8 biți de date, ceea ce însemna o încetinire a ratei de transmisie. Un dezavantaj major al comunicațiilor seriale constă in faptul ca transferul se realizează doar între 2 componente (deși este posibilă conectarea mai multor dispozitive la un singur port serial, numită si „bus contention", unde 2 dispozitive încearcă să se conecteze la aceeași linie de transmisie – există însă posibilitatea distrugerii componentelor si de obicei se rezolva cu costuri adiționale).

In cazul comunicației I2C exista posibilitatea extinderii la un număr mai mare de componente master. Ratele de transmisie in cazul transmiterii seriale sunt limitate la un set de baud rate-uri cunoscute pe când in cazul I2C rata de transfer este mult mai mare.

Fiecare bus I2C este compus din 2 semnale: SCL si SDA. SCL reprezintă semnalul de ceas iar SDA semnalul de date. Semnalul de ceas este întotdeauna generat de busul master curent. (Unele componente slave vor forța ceasul la nivelul low uneori pentru a sugera masterului sa introducă un o întârziere (delay) în transmiterea de date – acest lucru se mai numește si „clock stretching").

Spre deosebire de alte metode de comunicație serială, magistrala I2C este de tip “open drain”, ceea ce înseamnă ca pot trage o anumita linie de semnal în 0 logic dar nu o pot conduce spre 1 logic. Așadar, se elimina problema de „bus contention”, unde un dispozitiv încearcă să tragă una dintre linii in starea „high” în timp ce altul o aduce în „low”, eliminând posibilitatea de a distruge componente. Fiecare linie de semnal are un pull-up rezistor pe ea, pentru a putea restaura semnalul pe „high”, când nici un alt dispozitiv nu cere „low”.

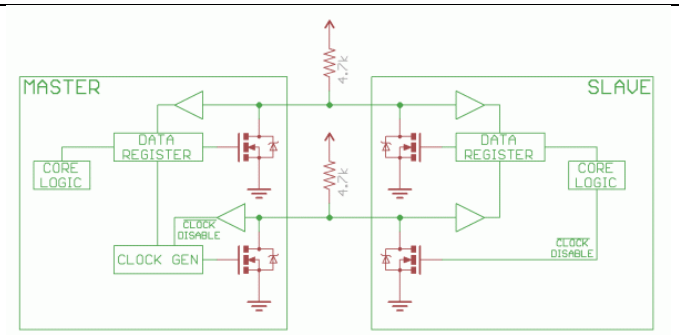

Fig. 8. Schema de conectare I2C. Se remarca folosirea a 2 rezistente pe liniile de semnal.

Selecția rezistentelor variază cu dispozitivele care folosesc busul, dar o regula bună este de a începe cu rezistente de 4.7k si scăderea lor daca e necesar.

Descrierea protocolului

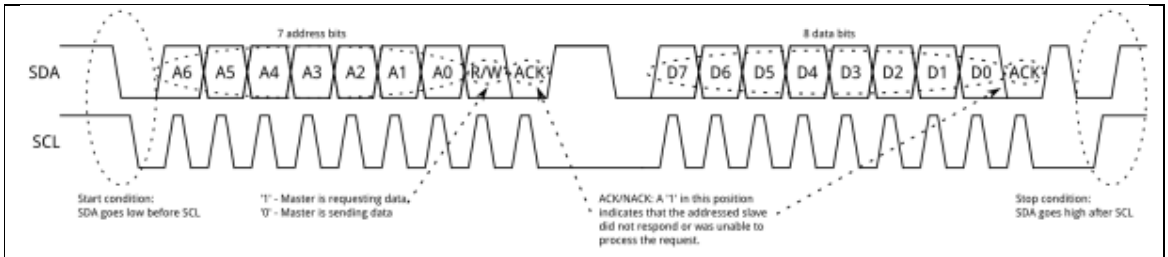

Fig. 9. Protocolul de transmisie I2C.

Mesajele sunt sparte în 2 tipuri de cadre (frames): cadre de adresa, unde masterul indică slave-ul la care mesajul va fi trimis și unul sau mai multe cadre de date care conțin mesaje pe 8 biți pasate de la master la slave sau viceversa. Datele sunt puse pe linia SDA după ce SCL ajunge la nivel low, si sunt eșantionate când SCL ajunge HIGH. Timpul intre nivelul de ceas și operațiile de citire/scriere este definit de dispozitivele conectate pe magistrală și va varia de la cip la cip.

Condiția de start (Start Condition)

Pentru a inița cadrul de adresa, dispozitivul master lasă SCL high și trage SDA low. Acest lucru pregătește toate dispozitivele slave întrucât o transmisie este pe cale să înceapă. Dacă două dispozitive master doresc să își asume busul la un moment dat, dispozitivul care trage la nivel low SDA primul câștiga arbitrajul și implicit controlul busului.

Cadrul de adresa (Address Frame)

Cadrul de adresa este întotdeauna primul în noua comunicație. Mai întâi se trimit sincron biții adresei, primul bit fiind cel mai semnificativ, urmat de un semnal de R/W pe biți, indicând dacă aceasta este o operație de citire (1) sau de scriere (0). Bitul 9 al cadrului este bitul NACK / ACK. Acesta este cazul pentru toate cadrele (date sau adresa). După ce primii 8 biți ale cadrului sunt trimiși, dispozitivului receptor îi este dat controlul asupra SDA. Dacă dispozitivul de recepție nu trage linia SDA în 0 logic înainte de al 9-lea puls de ceas, se poate deduce că dispozitivul receptor, fie nu au primit datele sau nu a știut cum sa interpreteze mesajul. În acest caz, schimbul se oprește, și tine de master să decidă cum să procedeze mai departe.

Cadrele de date (Data Frames)

După ce cadrul adresă a fost trimis, datele pot începe sa fie transmise. Masterul va continua sa genereze impulsuri de ceas, la un interval regulat, iar datele vor fi plasate pe SDA, fie de master fie de slave, în funcție de starea biților R/W (care indică dacă o operație este citire sau scriere). Numărul de cadre de date este arbitrar.

Condiția de oprire (Stop condition)

De îndată ce toate cadrele au fost trimise, masterul va genera o condiție de stop. Condițiile de stop sunt definite de tranziții low-> high (0-> 1) pe SDA, după o tranziție 0->1 pe SCL cu SCL rămânând pe high. În timpul operațiilor de scriere valoarea din SDA nu ar trebui să se schimbe când SCL e high pentru a evita condițiile de stop false.

Exemplu

Pentru a testa modul de funcționare a protocolului I2C realizați schema din figura 10 de mai jos.

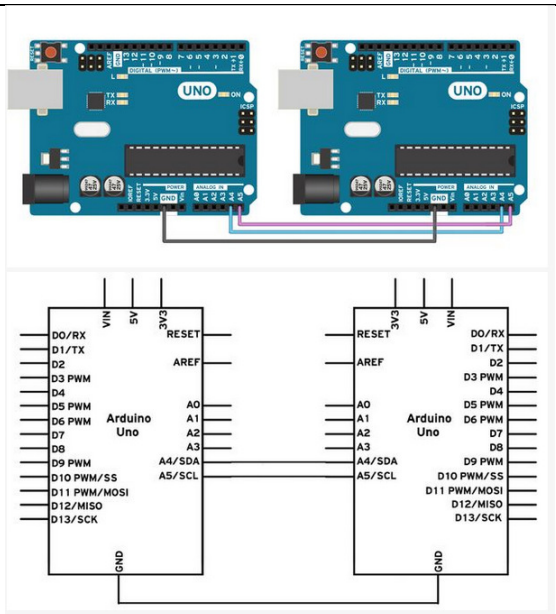

Fig. 10. Conectarea a 2 placi Arduino Uno

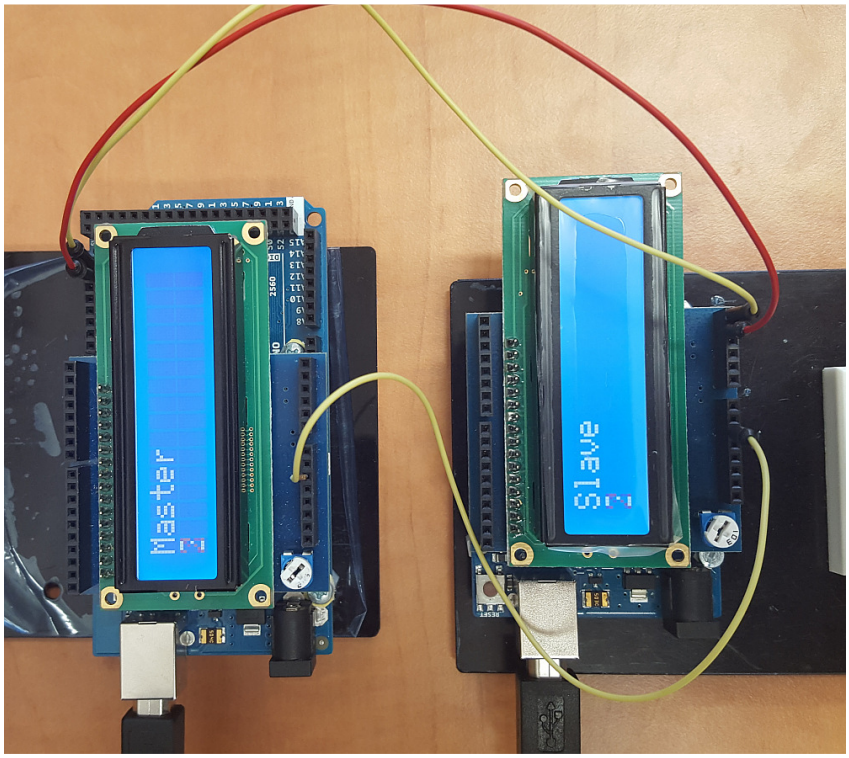

Fig. 11. Conectarea unei placi Arduino Mega cu o placa Arduino Uno

In cazul în care folosiți Arduino UNO/dumilanove/mini conectați pinii A4 si A5 ai unei placi la exact aceiași pini la placa a doua. De asemenea pinii de GND vor trebui legați împreuna.

Nu legați si tensiunile împreuna, și asigurați-vă că plăcile folosesc aceleași tensiuni de la sursa de alimentare.

Pentru a scrie codul vom folosi biblioteca Wire din mediul Arduino (https://www.arduino.cc/en/Reference/Wire).

In tabelul de mai jos avem locația pinilor I2C pe diferite placi Arduino.

Board	**I2C**
UNO, Ethernet	A4 (SDA), A5 (SCL)
MEGA 2560	20 (SDA), 21 (SCL)
Leonardo	2 (SDA), 3 (SCL)
Due	20 (SDA), 21 (SCL), SDA1, SCL1

Codul pentru dispozitivul slave:

```
#include <LiquidCrystal.h>

// Includem biblioteca necesara pentru I2C
#include <Wire.h>
```

```
int x = 0;

LiquidCrystal lcd(7, 6, 5, 4, 3, 2);

void setup() {
  // Pornim busul I2C ca si slave la adresa 9
  Wire.begin(9);
  // Atasam o functie care sa se declanseze atunci cand primim ceva
  Wire.onReceive(receiveEvent);

  lcd.begin(16,2);
  lcd.print("Slave");
}

void receiveEvent(int bytes) {
  x = Wire.read();    // citim un character din I2C
}

void loop() {
  lcd.setCursor(0,1); // afisare caracter receptionat
  lcd.print(x);
}
```

Codul pentru master:

```
#include <LiquidCrystal.h>

// Includem biblioteca wire pentru I2C
#include <Wire.h>

LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
int x = 0;

void setup() {
  // Dechidem magistrala I2C ca master
  Wire.begin();
  lcd.begin(16,2);
  lcd.print("Master");
}

void loop() {
  Wire.beginTransmission(9); // transmitem spre device #9
  Wire.write(x);             // trimitem x
  Wire.endTransmission();    // oprim transmisia

  lcd.setCursor(0,1);  // afisare caracter transmis pe lcd master
  lcd.print(x);

  x++; // incrementam x
  if (x > 5) x = 0; // resetam x odata ce ajunge la 6
```

```
  delay(500);
}
```

Activități practice

1. Testați exemplele din laborator. Întrebați cadrul didactic pentru orice nelămurire aveți legata de modul de conexiune a firelor/componentelor si de transmitere a datelor.
2. Realizați un sistem de comunicație între doua PC-uri folosind plăci Arduino. Plăcile vor fi conectate la PC prin USB, si intre ele prin I2C. Textul scris în serial monitor la PC-ul legat la placa I2C master va apărea în serial monitor la PC-ul conectat la placa I2C slave.
3. Realizați o conexiune ca la exemplul 2, dar bi-direcțională. Pentru transmisiunea de la slave la master studiați https://www.arduino.cc/en/Tutorial/MasterReader
4. Realizați o rețea cu un master şi doi slave. Masterul va fi conectat la PC, şi va primi prin interfața serială mesaje de genul
    a. s1-hello
    b. s2-goodbye

    În funcție de numărul de după litera s, mesajul de după cratimă se va transmite slave-ului 1 sau 2. Plăcile slave for afişa mesajul destinat lor (şi numai acest mesaj) pe lcd.