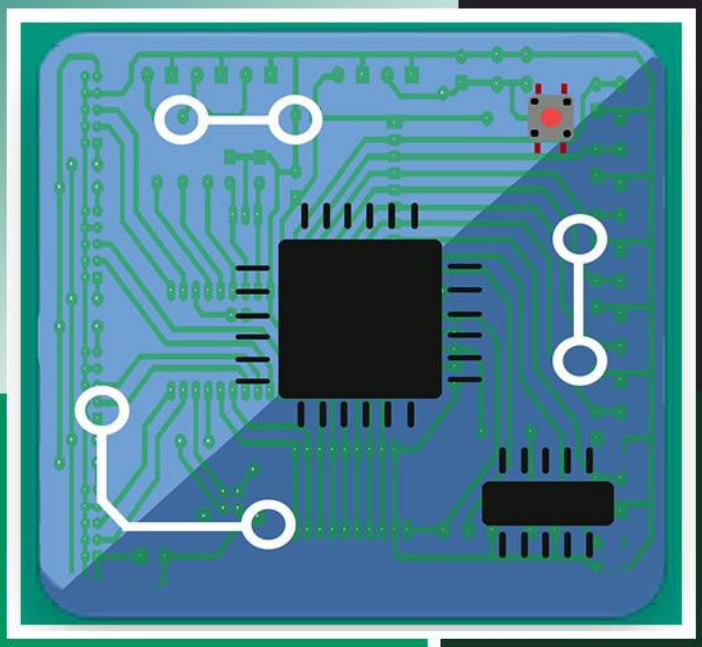
Radu Dănescu
Mircea Paul Mureșan
Răzvan Itu



Proiectare cu Microprocesoare

Indrumător de laborator

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

UTPRESS
Cluj-Napoca, 2018
ISBN 978-606-737-336-3

Editura U.T. PRESS
Str. Observatorului nr. 34
C.P. 42, O.P. 2, 400775 Cluj-Napoca
Tel.:0264-401.999
e-mail: utpress@biblio.utcluj.ro
http://biblioteca.utcluj.ro/editura

Director: Ing. Călin D. Câmpean

Recenzia: Conf.dr.ing. Florin Oniga
Prof.dr.ing. Sergiu Nedevschi

ISBN 978-606-737-336-3

CUPRINS

Introducere

Microprocesorul este elementul central al unui sistem de calcul modern, el încorporând pe un singur circuit integrat toate funcțiile unei unități centrale de procesare (CPU). CPU are ca sarcină principală execuția unui program, care este o secvență de instrucțiuni în cod mașină stocate într-o memorie. Execuția unei instrucțiuni are de obicei patru etape: citirea instructiunii (*fetch*), decodificarea ei (*decode*), execuția aritmetico-logică (*execute*), și scrierea rezultatelor (*write back*).

Deși dispun de o complexitate ridicată, microprocesoarele nu pot funcționa în lipsa unor sisteme esențiale, precum memoriile în care sunt stocate instrucțiunile și datele, interfețele de intrare și ieșire a datelor, sau alte dispozitive periferice pentru primire și transmitere a datelor, conectate toate la microprocesor prin intermediul magistralelor. Împreună cu microprocesorul, aceste componente alcătuiesc sistemele cu microprocesor. Un exemplu de sistem cu microprocesor este placa de bază a unui PC.

Spre deosebire de microprocesoare, care au doar rolul de execuție al instrucțiunilor, microcontrollerele dispun în mod suplimentar și de memorie pentru program și date, de porturi de intrare/ieșire (GPIO - General Purpose Input/Output), și de alte componente precum temporizatoare, interfețe de comunicare, convertoare, etc. Microcontrollerul este deci un sistem cu microprocesor inclus pe un circuit integrat, care poate fi folosit pentru a controla diverse procese sau dispozitive. Datorita dimensiunii reduse și al costului de productie scăzut, microcontrollerele sunt utilizate adesea în sisteme de tip "embedded", unde consumul redus de resurse are un rol important.

Disciplina Proiectare cu Microprocesoare, din programa anului III Calculatoare și Tehnologia Informației, are ca obiectiv familiarizarea studenților cu caracteristicile sistemelor cu microprocesor. De asemenea, prin această disciplină intenționăm să le dăm studenților (și tuturor persoanelor interesate) cunoștințele și deprinderile practice necesare pentru a putea proiecta și implementa propriile lor sisteme cu microprocesor sau cu microcontroller.

Acest îndrumător de laborator conține materiale pentru familiarizarea pas cu pas a studenților cu microcontrollerele AVR pe 8 biți, cu plăcile de dezvoltare Arduino Mega și Arduino Uno, cu utilizarea porturilor de intrare/ieșire și a interfețelor seriale, precum și cu utilizarea multiplelor module periferice pentru introducere și afișare a datelor, utilizarea senzorilor și a actuatorilor. Pentru scrierea și depanarea programelor, studenții vor fi familiarizați cu mediile de dezvoltare Arduino IDE, Atmel Studio și Processing.

Ne dorim ca aceste lucrări de laborator, care conțin explicații și exemple detaliate, să ajute studenții în asimilarea noțiunilor teoretice și practice ale disciplinei, în vederea obținerii unui rezultat bun la examen, dar și să le ofere deprinderi practice care să-i inspire să-și transpună în realitate propriile lor idei originale.

AUTORII

I. Laborator 1 – Introducere în utilizarea plăcilor Arduino

Familia de unelte de dezvoltare Arduino include plăci cu microcontroller, accesorii şi componente software open source, care permit utilizatorilor să realizeze proiecte folosind o abordare unificată, de nivel înalt, care se doreşte a fi independentă de microcontrollerul folosit. Plăcile Arduino sunt echipate în principal cu microcontrollere Atmel AVR, dar există şi plăci echipate cu microcontrollere de tip ARM, sau din familia x86. În afara plăcilor Arduino oficiale, există o gamă largă de clone, de obicei cu preț redus (si performanţă discutabilă), precum XDruino, Freeduino, etc.

Placa pe care o vom folosi pentru activitatea de laborator este Arduino Mega 2560, bazată pe microcontrollerul Atmel AVR ATMega2560, pe 8 biți. Placa dispune de 54 de pini digitali pentru intrare/ieşire, şi 16 pini pentru preluare de semnale analogice. Unii pini pot avea funcţie multiplă, constituind semnale pentru diferite interfeţe de comunicaţie (UART, SPI, I2C). Frecvența microcontrollerului este de 16 MHz. Placa se alimentează prin cablul USB cu care se conectează la calculator, sau poate fi alimentată cu o sursă de tensiune continuă, de 7 … 12 V, care poate furniza o intensitate a curentului de minim 0.25 A. A doua opţiune este necesară când placa trebuie să alimenteze periferice mari consumatoare, precum motoare, shield-uri GSM. Etc.

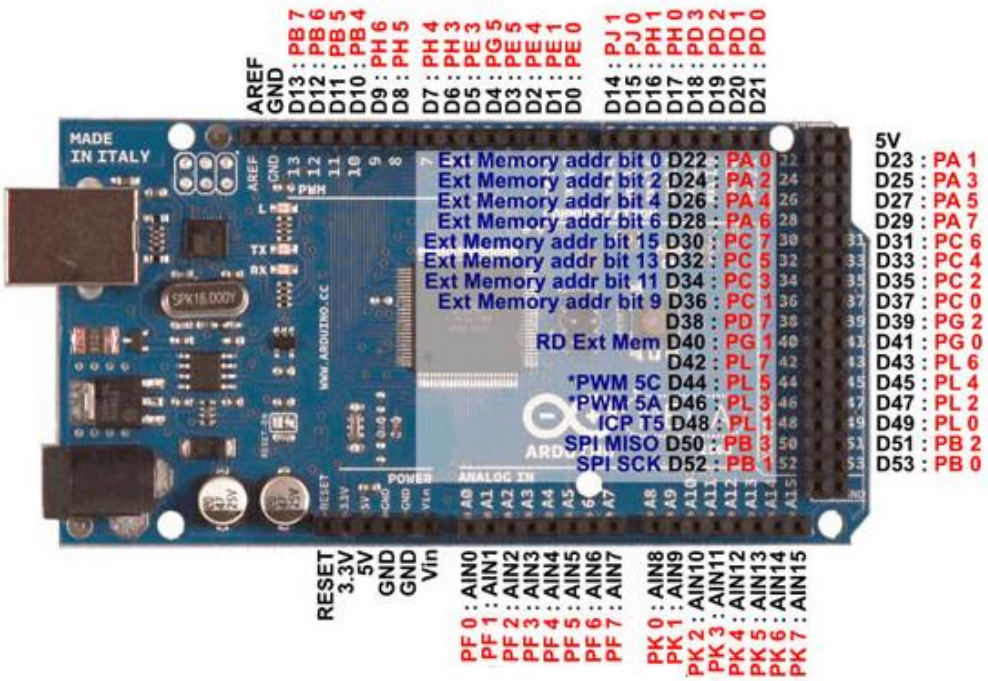


Fig.I.1 Pinii plăcii Arduino Mega (sursa: http://electrobist.com/product/arduino-mega-2560-r3/)

1.1. Primul exemplu: Utilizarea unei placi de dezvoltare Arduino

Pentru o mai bună desfășurare a activității, placa de dezvoltare a fost fixată pe un suport de plastic, împreună cu o placă de prototipizare (“breadboard”).

Pentru început, vom încărca pe placă exemplul cel mai elementar, “Blink”, disponibil în directorul de instalare al mediului de dezvoltare Arduino (de obicei C:\Program Files\Arduino\examples\01.Basics\Blink). Pentru acest lucru, copiați directorul “Blink” în directorul vostru de lucru (**pe care îl veți crea în D:\Studenti\Grupa30xxx ! În orice altă locație directorul dvs. va fi ŞTERS periodic**). Verificați ca după copiere directorul și fișierul din interiorul acestuia, blink.ino, să nu fie cu atributele “Read only”.

Regulă: fiecare proiect Arduino, chiar dacă are doar un singur fișier sursă, trebuie plasat într-un director cu același nume ca fișierul sursă.

După copierea directorului, lansați în execuție mediul Arduino, executând dublu-click pe fișierul sursa **blink.ino**. Fereastra deschisă ar trebui să arate așa:

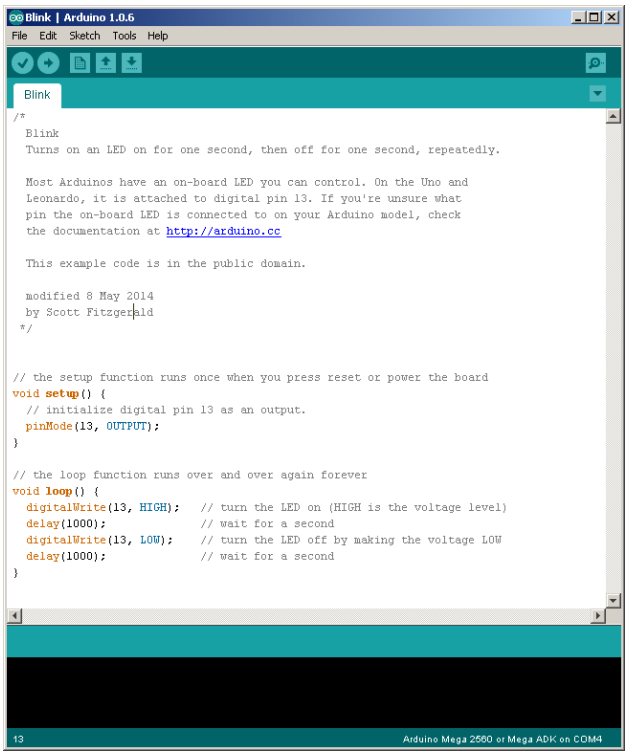


Fig.I.2. Fereastra principală a mediului de dezvoltare Arduino

Dacă mediul de dezvoltare nu se deschide, este posibil ca asocierea dintre mediu și tipul de fișier .ino să nu se fi putut realiza cu succes. În acest caz, deschideți mediul de dezvoltare din Start/Programs și alegeți din meniu opțiunea File → Open.

În acest moment, puteți conecta la PC placa Arduino Mega 2560, prin cablul USB. Folosiți, dacă se poate, portul USB din panoul frontal al calculatorului, deoarece veți fi nevoiți să deconectați și să re-conectați cablul de mai multe ori în timpul unui laborator.

După conectare, este posibil ca sistemul să ceară instalarea unui driver. Indicați sistemului de operare calea spre driver-ul Arduino, de obicei "C:\Program Files\Arduino\drivers". Dacă nu vă descurcaţi, sau aveți probleme cu drepturile utilizator, apelați la ajutorul cadrului didactic.

Dacă instalarea driverului a decurs cu succes și placa Arduino este în funcțiune (este aprins un LED verde pe placă), atunci puteți merge mai departe.

Înainte de a încerca să programăm placa, trebuie să ne asigurăm că mediul este configurat în mod corect. **Trebuie aleasă placa corectă, din meniul Tools → Board, conform imaginii de mai jos**:

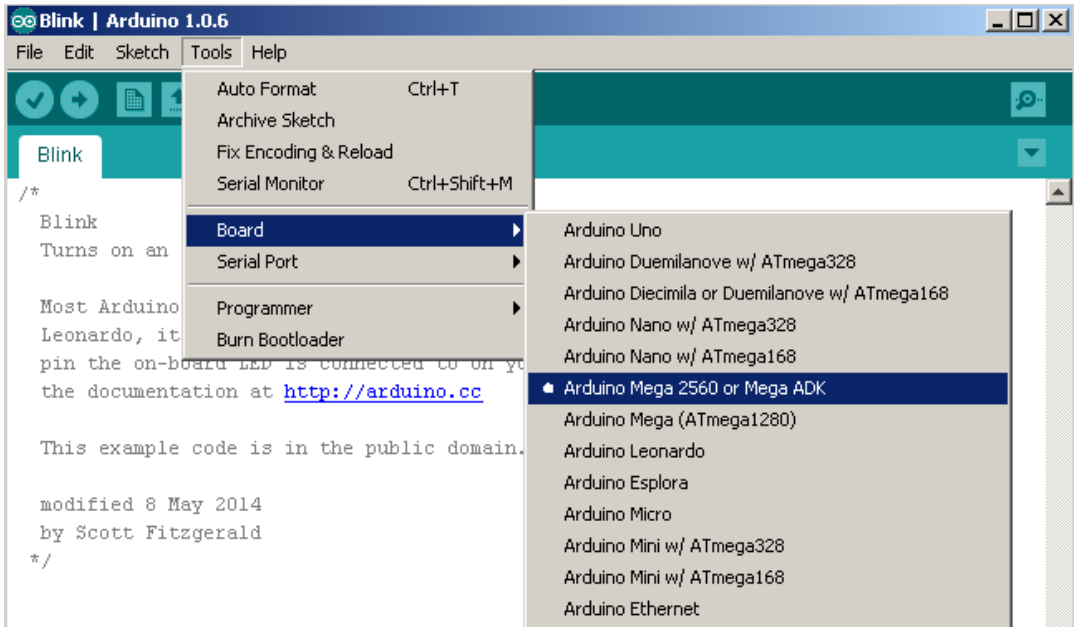


Fig.I.3. Selecția tipului plăcii de dezvoltare

Dacă folosiţi o altă placă, precum Arduino Uno, trebuie să alegeţi setările corespunzătoare.

De asemenea, trebuie configurat portul serial prin care se face comunicarea cu placa. Conexiunea USB va instala un port serial virtual, cu un nume ce începe cu COM, urmat de o cifră. Calculatoarele pe care lucrați au unul sau doua porturi seriale pe placa de bază, numite COM1 sau COM2. Porturile seriale virtuale primesc un număr mai mare, de la 3 în sus.

Configurați portul serial din meniul Tools → Serial Port, ca în figura de mai jos:

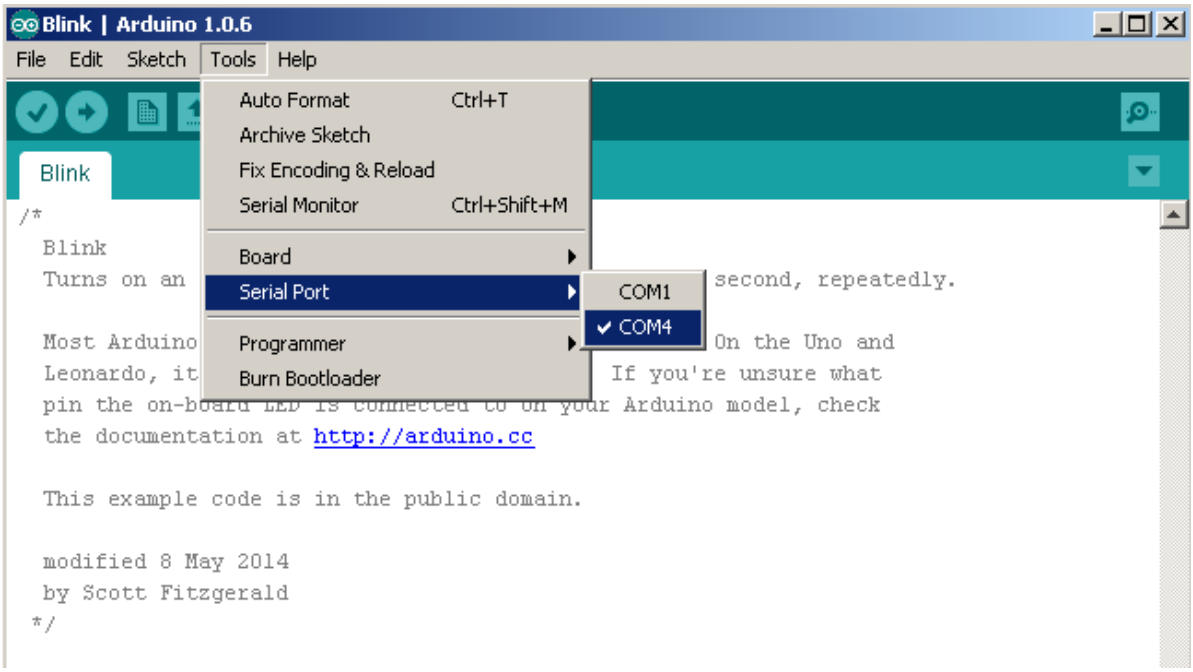


Fig.I.4. Selecția portului serial

După configurare, puteți compila și programa primul exemplu pe placă, apăsând butonul "Upload", cel indicat în figura de mai jos. Dacă toţi paşii au fost executaţi corect, programul va rula pe placă, aprinzând şi stingând led-ul conectat la pin-ul digital 13.

Fig.I.5. Butonul pentru încărcare

1.2. Al doilea exemplu: Pini digitali de intrare și folosirea interfeței seriale

Ca dispozitiv elementar de intrare, vom utiliza blocul de butoane vizibil în figura de mai jos:

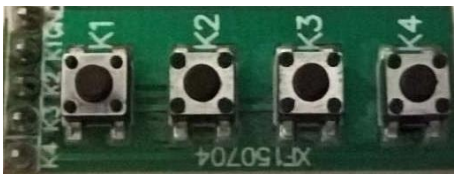


Fig.I.6. Blocul de butoane

Blocul de butoane are 5 pini:
- GND (masa)
- 4 pini de date (K1, K2...K4), indicând starea butoanelor (logic 0 = buton apăsat)

Numele corespunzator apare atât în dreptul fiecărui pin, cât și în dreptul fiecărui buton.

Schemele generale de conectare a unui buton la un microcontroller sunt prezentate mai jos. Prima schemă ilustrează folosirea unui rezistor Pull-Down, pe când în a doua schemă se folosește un rezistor de Pull-Up.

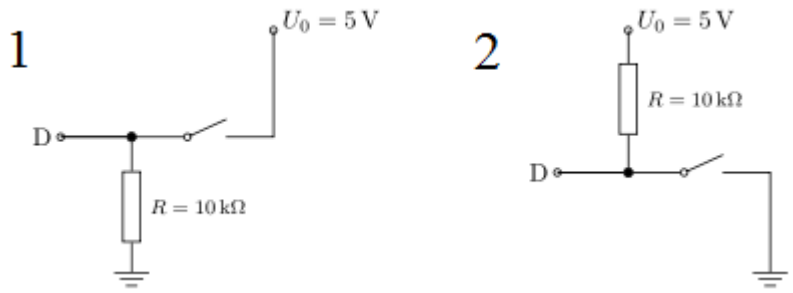


Fig.I.7, Fig.I.8 – Folosirea rezisențelor pull-down și pull-up (sursa: https://de.wikibooks.org/wiki/Mikrocontroller/_Digitale_und_analoge_Signale)

Ne vom îndrepta atenția asupra folosirii rezistoarelor de tip Pull-Up, întrucât sunt mai comune decât cele de tip Pull-Down. În principiu, funcționarea lor este similară, diferența fiind că rezistorii Pull-Up sunt conectați la 5V (sau VCC), pe când rezistorii de tip Pull-Down sunt conectați la masă (GND). Când se folosește un rezistor de tip Pull-Up, valoarea citită la pinul de intrare D, când butonul nu e apăsat, va fi 1 logic sau HIGH. Cu alte cuvinte, o cantitate mică de curent se scurge între VCC și pinul de intrare (nu GND), așadar acest pin de intrare citește o valoare aproape de VCC. În momentul în care apăsăm butonul, pinul de intrare se conectează direct la ground. Curentul trece prin rezistor și ajunge la ground făcând pinul de intrare să citească valoarea 0 sau LOW. Dacă nu am folosi rezistența de Pull-Up, butonul ar conecta VCC la GND, lucru care ar rezulta într-un scurt circuit (lucru pe care nu îl dorim).
Blocul de butoane utilizat nu are rezistențe, așadar pentru utilizarea lui avem 2 opțiuni:
- Lipim rezistențe auxiliare fiecărui buton
- Folosim rezistențele de tip Pull-Up / Pull-Down prezente pe microcontroller

În prezenta lucrare vom folosi rezistențele oferite de placa de dezvoltare Arduino. Acestea sunt activate folosind opțiunea INPUT_PULLUP în stadiul de configurare a pinilor.

Pentru ieșire, vom utiliza interfața serială, putând astfel monitoriza ieșirile plăcii direct de pe calculator. **Toate conexiunile între un dispozitiv periferic, componenta, Bread-Board și placa Arduino le faceți doar cu cablul USB decuplat de la calculator!**

Pentru conectarea blocului de butoane, vom folosi placa de prototipizare (“Bread-Board”). Vom conecta cei patru pini de date la pinii digitali 4, 5, 6, 7 si GND. Pentru a evita situația în care firele se pot desprinde la orice mică mișcare, ducând la funcționare greșită sau

la defecțiuni (montaj de tip "păianjen"), vom folosi placa de prototipizare pentru a rigidiza montajul.

Placa de prototipizare are conectorii conectaţi electric în grupuri de câte 5, pe jumătăţi de coloană, cum arată figura de mai jos:

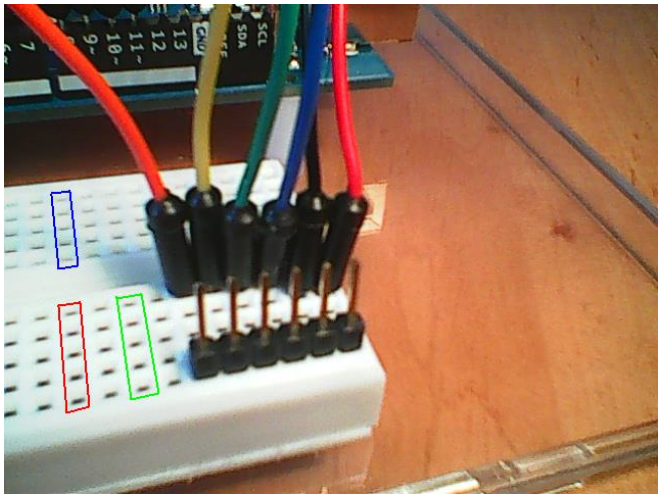


Fig.I.9. Conexiunile electrice ale unei plăci de prototipizare

Plasaţi blocul de butoane în breadboard şi apăsaţi, astfel încât pinii blocului de butoane să intre în găurile breadbordului. Introduceţi apoi fire, pe fiecare semi-coloană în care aveţi pini ai blocului de butoane, ca în figura de mai jos. Firul negru din figură reprezintă masa (GND), iar celelalte fire, de culori diferite, corespund butoanelor.

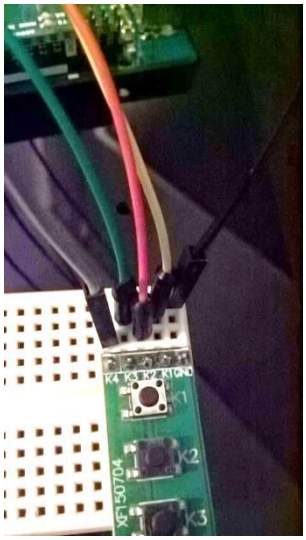


Fig.I.10. Conectarea butoanelor la placa de prototipizare

Capetele libere ale firelor le veți conecta la placa Arduino, astfel:

- Firele de semnal le conectați la pinii digitali 4, 5, 6, 7.
- Firul negru la GND.

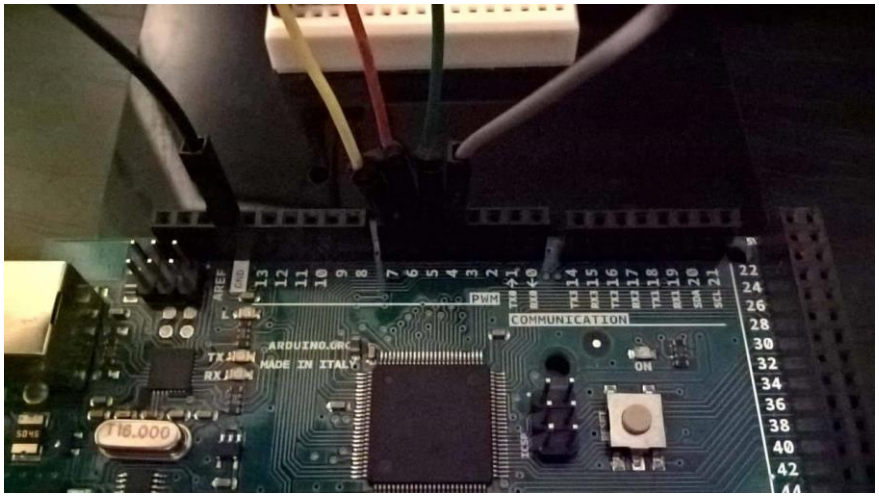


Fig.I.11. Conectarea firelor la placa Arduino

În acest moment, montajul fizic este gata. În continuare, deschidem mediul Arduino şi creăm un program nou (*File → New*), ce va conține următorul cod:

```
// Citirea stării butoanelor conectate la pinii 4, 5, 6, 7
// afișarea prin interfața serială
//  se transmite un număr care are ultimele 4 cifre starea
//butoanelor apăsate
// variabile pentru starea butoanelor
int b1;
int b2;
int b3;
int b4;

// variabila pentru compunerea numărului de transmis
int stat = 0;

void setup() {
    // configurare pini pentru butoane, intrare
    pinMode(4, INPUT_PULLUP);
    pinMode(5, INPUT_PULLUP);
    pinMode(6, INPUT_PULLUP);
    pinMode(7, INPUT_PULLUP);
    // activare comunicatie serială
    Serial.begin(9600);
}

void loop() {
    // citire stare butoane
    b1 = digitalRead(4);
    b2 = digitalRead(5);
    b3 = digitalRead(6);
    b4 = digitalRead(7);
    // compunere rezultat
```

```
    stat = 10000 + b4 * 1000 + b3 * 100 + b2 * 10 + b1;
    // transmisie
    Serial.println(stat);
    // așteptare 0.5 sec
    delay(500);
}
```

Conectați placa Arduino şi apăsați butonul "Upload". Pentru vizualizarea rezultatului, deschideți utilitarul "Serial Monitor", din meniul Tools → Serial Monitor. În fereastra utilitarului va fi afişat, la fiecare 0.5 secunde, un număr de 5 cifre (1XXXX), ultimele patru cifre fiind starea butoanelor, ca în figura de mai jos. Când apăsați pe un buton, starea lui se transformă din 1 în 0.

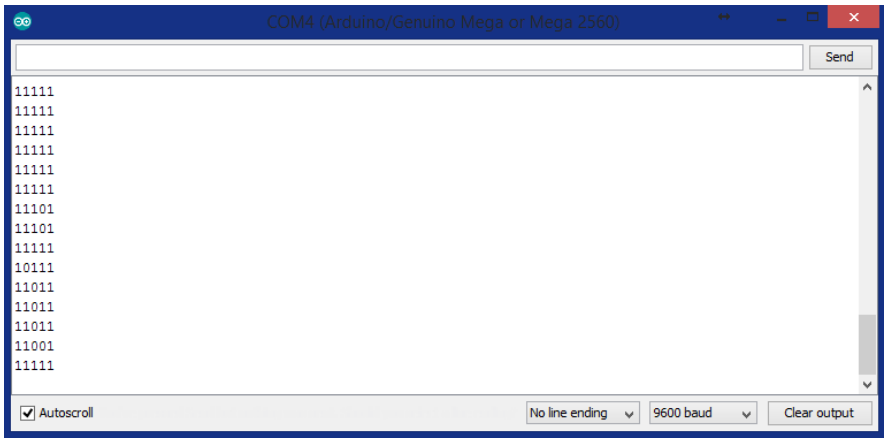


Fig.I.12. Datele transmise, vizualizate în Serial Monitor

Atenție! Serial Monitor trebuie închis înainte de a deconecta placa Arduino de la calculator. În caz contrar, este posibil ca portul serial virtual să rămână blocat şi comunicarea ulterioară cu placa să nu mai fie posibilă, decât după restartarea calculatorului !

1.3. Al treilea exemplu: Folosirea blocului de LED-uri pentru ieşire. Folosirea Porturilor.

În al treilea exemplu, vom utiliza pentru ieşire un bloc de leduri. Acesta are 7 pini, dintre care unul este de alimentare (GND) şi 6 sunt semnalele pentru cele 6 led-uri (valoare logică 1 = led aprins). Observați că ledurile au atașate rezistențe pentru a le proteja să nu fie arse.

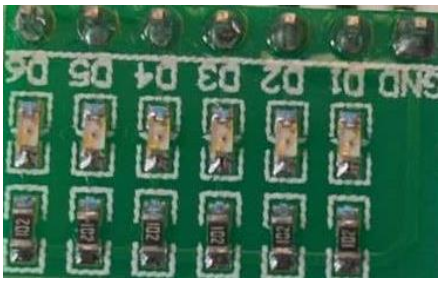


Fig.I.13. Blocul de LED-uri

Pentru comanda mai rapidă a acestor led-uri, vom folosi porturile microcontrollerului ATMega 2560.

Primul pas constă în pregătirea celor 7 poziții pe breadboard, corespunzătoare celor 7 pini ai blocului de led-uri (D6…D1, GND). Vom folosi 7 fire, pe care le vom conecta ca în figura de mai jos.

Vom introduce blocul de led-uri direct în breadboard, ca în figura de mai jos. **Apăsaţi ferm, fără a lovi.**

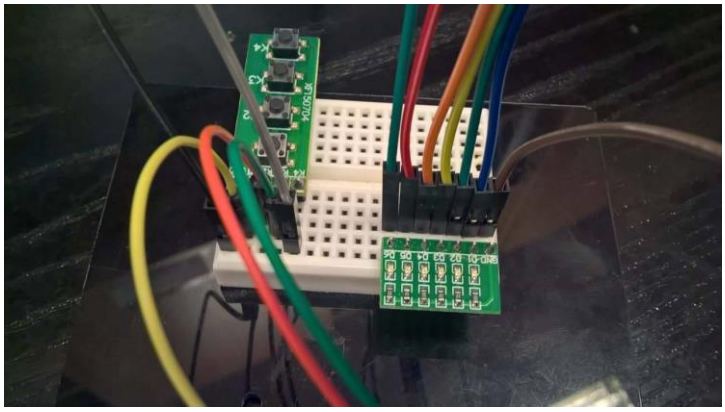


Fig.I.14. Conectarea LED-urilor la placa de prototipizare

Firele de semnal le vom conecta la pinii digitali 22, 23, 24, 25, 26, 27 ai plăcii Arduino, corespunzători biților din portul A (PA5, PA4, PA3, PA2, PA1 si PA0), ca în figura de mai jos:

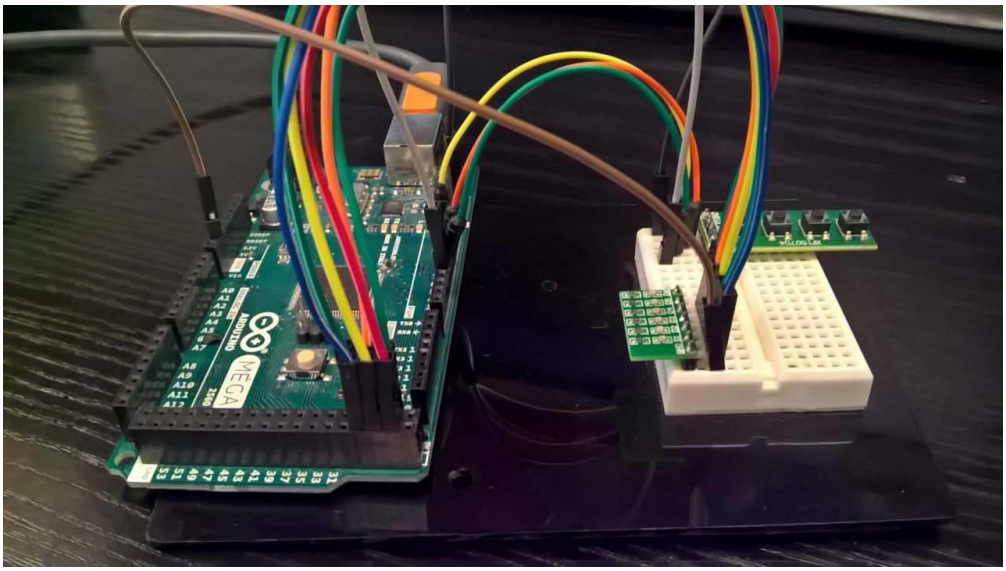


Fig.I.15. Conectarea întregului sistem

Corespondenţa dintre pini şi porturi este descrisă în documentul "ATmega2560-Arduino Pin Mapping", ce poate fi vizualizat aici: http://arduino.cc/en/Hacking/PinMapping2560 , sau în prima figură a acestui document. Pentru portul A, avem:

PA7 (AD7)	Digital pin 29
PA6 (AD6)	Digital pin 28
PA5 (AD5)	Digital pin 27
PA4 (AD4)	Digital pin 26
PA3 (AD3)	Digital pin 25
PA2 (AD2)	Digital pin 24
PA1 (AD1)	Digital pin 23
PA0 (AD0)	Digital pin 22

Fig.I.16. Corespondența dintre pinii digitali și porturile AVR

Pentru pornirea și opriera ledurilor se folosește montajul cu butoane realizat în exercițiul precedent.

Notă: În viitor, când vom avea mai multe componente, vom încerca să identificăm firele cu semnale comune între componente. Acestea vor fi grupate pe breadboard (puse pe aceeași coloană). În exemplul nostru semnalele de ground ale celor doua module puteau fi grupate, putând astfel să folosim un singur fir de masă.

După realizarea montajului fizic, creaţi un nou proiect Arduino şi adăugaţi în el următorul cod:

```
// Citirea stării butoanelor conectate la pinii 4,5,6,7
// afișarea pe LED-uri conectate la PORTA

// variabile pentru starea butoanelor
int b1;
int b2;
int b3;
int b4;

// variabila pentru compunerea numărului de transmis
unsigned char stat = 0;

void setup() {
    // configurare pini pentru butoane, intrare
    pinMode(4, INPUT_PULLUP);
    pinMode(5, INPUT_PULLUP);
    pinMode(6, INPUT_PULLUP);
    pinMode(7, INPUT_PULLUP);
    // activare PORTA , ca ieșire,
    DDRA = 0b11111111;
}

void loop() {
    // citire stare butoane
```

```
    b1 = digitalRead(4);
    b2 = digitalRead(5);
    b3 = digitalRead(6);
    b4 = digitalRead(7);
    // compunere rezultat
    // fiecare LED e controlat de 1 buton, unele butoane sunt
    //duplicate
    stat = (b4<<5) | (b3<<4) | (b4<<3) | (b3<<2) | (b2<<1) |
    b1;
    // afișare pe LED-uri, legate la port a
    PORTA = stat;
    // așteptare 50 ms
    delay(50);
}
```

1.4. Lucru individual:

1. Calculați valoarea rezistenței de Pull-Up, în cazul în care nu am folosi opțiunea INPUT_PULLUP din Arduino, ci o rezistență externă și am dori să limităm curentul la 1 mA.
2. Rulați exemplele 1 și 2.
3. Modificați exemplul 2, pentru ca placa să transmită la calculator informații diferite în funcție de butonul apăsat. De exemplu, la apăsarea unui buton puteți transmite numărul de milisecunde de când a fost pornit programul, apelând funcția millis(), la apăsarea unui alt buton puteți transmite numărul de secunde (millis()/1000), la apăsarea altui buton un mesaj text, etc.
4. Rulați exemplul 3.
5. Modificați exemplul 3 pentru a afișa pe led-uri o animație (aprindere din 2 în 2, baleiere, etc) care se modifică la apăsarea unui buton.

Referințe :

1. Materiale suplimentare: http://users.utcluj.ro/~rdanescu/teaching_pmp.html
2. „Maparea pinilor la Arduino Mega 2560”: http://arduino.cc/en/Hacking/PinMapping2560

II. Laborator 2 – Aplicaţii cu module simple de intrare şi ieşire

2.1. Afișorul cu 2x7 segmente

Modulul PmodSSD (Seven Segment Display) din figura de mai jos oferă posibilitatea afișării a două caractere. Fiecare segment este un LED, care se aprinde dacă există combinaţia corectă de tensiuni pe anodul si catodul acestuia (anodul HIGH, catodul LOW).

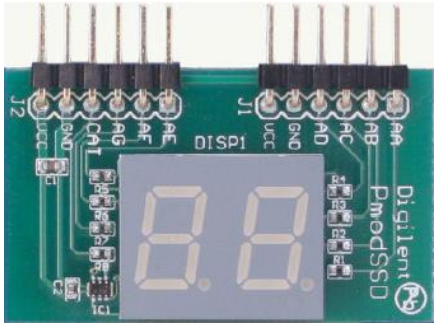


Fig.II.1. Afișaz cu 7 segmente (SSD)

Pentru a economisi numărul de pini de semnal necesari, cele două cifre nu pot fi aprinse simultan. Selecţia dintre cele două cifre o face semnalul CAT (C pe schema de mai jos). Dacă acest semnal este 0, se aprind segmentele cifrei unităţilor (cifra din dreapta), iar dacă semnalul CAT este 1, se aprind segmentele cifrei din stânga.

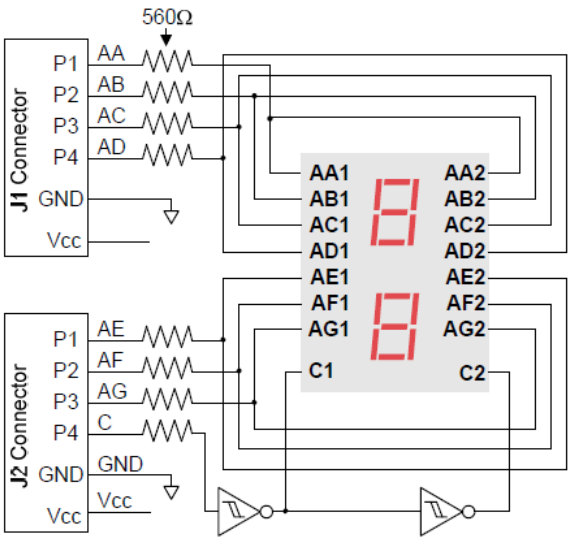


Fig.II.2. Structura internă a SSD (sursa: https://store.digilentinc.com/pmod-ssd-seven-segment-display/)

Semnalele notate cu litere AA…AC corespund anozilor LED-urilor segment.
Corespondenţa acestor semnale este indicată de figura de mai jos:

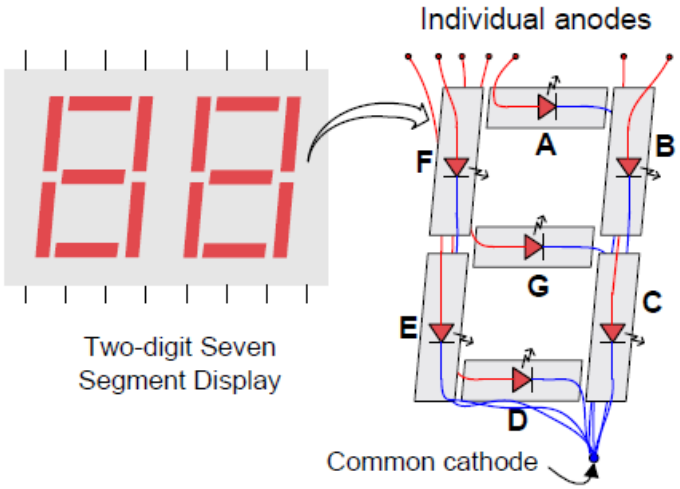


Fig.II.3. Structura internă a unei cifre (sursa: https://store.digilentinc.com/pmod-ssd-seven-segment-display/)

De exemplu, pentru afişarea cifrei 3 va fi nevoie ca segmentele să primească următoarele nivele logice:

G	F	E	D	C	B	A
1	0	0	1	1	1	1

Fig.II.4. Exemplu de activare a biților pentru afișarea cifrei ‘3’

Pentru afișarea unui număr de două cifre, trebuie comutat foarte rapid între cele două blocuri, folosind semnalul CAT. Următoarea diagramă de timp ilustrează procesul:

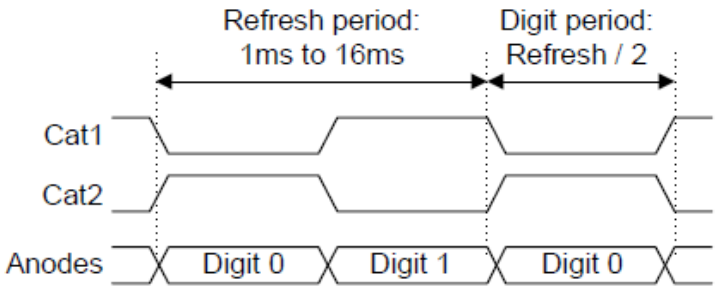


Fig.II.5. Diagrama de timp pentru ciclul de baleiere al SSD (sursa: https://store.digilentinc.com/pmod-ssd-seven-segment-display/)

Pentru realizarea funcţionalităţii acestei diagrame de timp, se foloseşte următorul pseudocod:

```
Loop:
   AA…AG = cod_cifra_unitati
   CAT=0
   Delay()
   AA…AG=cod_cifra_zeci
   CAT=1
   Delay()
Goto loop
```

2.2. Utilizarea porturilor microcontrollerului pentru Intrare/Ieşire

Arduino, prin funcţiile sale digitalRead şi digitalWrite, ascunde mecanismul prin care microcontrollerul face aceste operaţii. Mai mult, aceste funcţii introduc întârzieri, care pot fi semnificative atunci când e nevoie de transferul datelor pe mai mulţi biţi.

Orice microcontroller este conectat cu exteriorul prin porturi de intrare/ieşire. AVR ATmega 2560 este un microcontroller pe 8 biţi, deci şi porturile lui au 8 biţi. Fiecare port are asociate trei registre (x va fi înlocuit cu A, B, C, D, …, în funcţie de portul folosit):

– DDRx – registrul de direcţie
– PINx – registrul pentru date de intrare
– PORTx – registrul pentru date de ieşire

Registrul de direcţie DDRx

DDRx (Data Direction Register) configurează direcţia datelor pe pinii portului (dacă un bit din port va fi folosit pentru intrare sau pentru ieşire). Un 0 scris pe un bit din DDRx face ca pinul corespunzător din port să fie pin de intrare, iar un bit setat la 1 face ca pinul corespunzător să fie pin de ieşire.

Exemple:

- Pentru a configura toţi pinii portului A ca intrare:
  DDRA = 0b00000000;
- Pentru a configura toţi pinii portului A ca ieşire:
  DDRA = 0b11111111;
- Pentru a configura jumătatea inferioară a portului B ca ieşire, şi jumătatea superioară ca intrare:
  DDRB = 0b00001111;

Registrul de intrare PINx

PINx (Port IN) se foloseşte pentru citirea datelor de la pini configuraţi ca intrare. Pentru a se putea citi date, aceşti pini trebuie configuraţi ca intrare, setând biţii din DDRx la zero.
Exemplu: citirea datelor din portul A
DDRA = 0;

char a = PINA;

Registrul de ieşire PORTx

Registrul PORTx este folosit pentru a transmite date de la microcontroller la perifericele conectate la pinii portului x. Pentru ca datele să fie vizibile la ieşire, biţii corespunzători din registrul de direcţie DDRx trebuie să fie configuraţi cu valoarea 1.
Exemplu: aprinderea din 2 in 2 a 8 led-uri conectate la portul A
DDRA = 0xFF
PORTA = 0b10101010

2.3. Conectarea afișorului 7segmente la placa Arduino Mega 2560

Montajul se face cu placa deconectată de la PC. Primul pas este introducerea afişorului în breadboard, apăsând ferm, dar fără a produce lovituri. Apoi vom conecta firele de alimentare, în semi-coloanele corespunzătoare VCC şi GND pentru conectorul al doilea (J2) al afișorului. Celălalt capăt al firelor de alimentare îl vom conecta la pinii 5V si GND ai plăcii Arduino. Nu este nevoie să aplicăm tensiune şi la conectorul J1.

Pentru pinii de semnal, corespunzători anozilor afișorului AA…AG şi pentru catodul CAT, vom conecta pe rând fire, pe care le vom conecta la placa Arduino la pinii digitali 22, 23, … 29, corespunzători biților 0… 7 ai portului A.

Montajul integral:

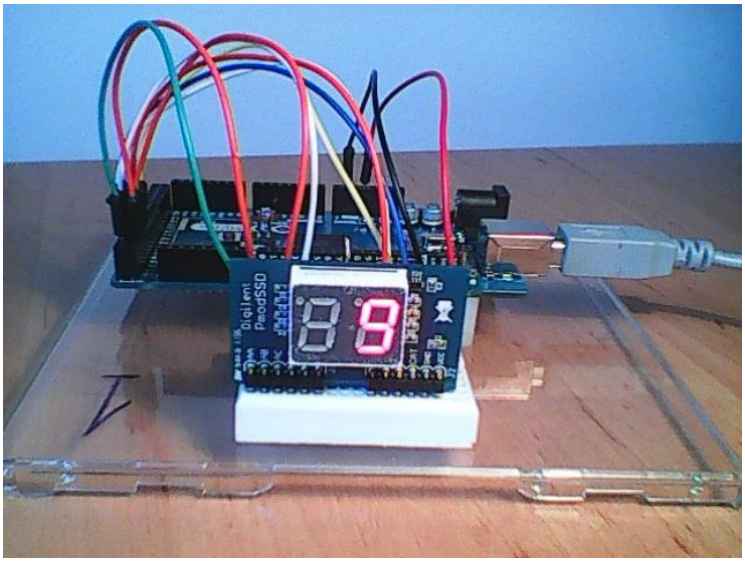


Fig.II.6. Montajul Arduino-SSD

Detaliu cu firele conectate în spatele afișorului:

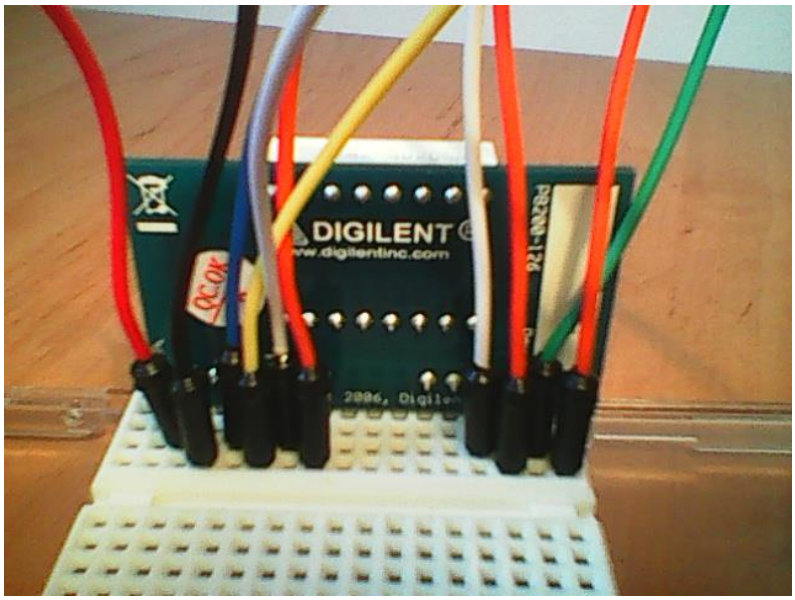


Fig.II.7. Conectarea firelor și a SSD pe placa de prototipizare

Conectarea pinilor de semnal:

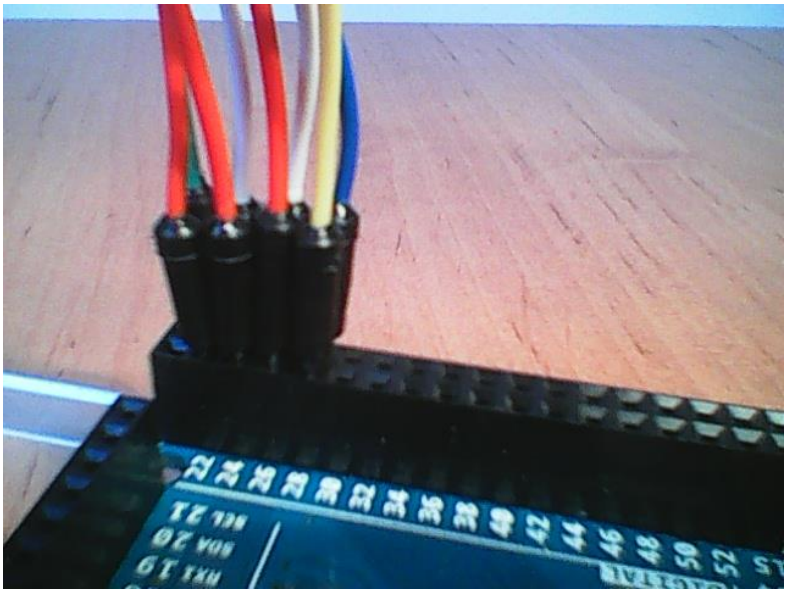


Fig.II.8. Conectarea firelor SSD la Arduino

Atenţie! Primii doi pini de pe conectorul plăcii Arduino sunt pini de alimentare! Pinii digitali încep cu a doua pereche! Mare atenţie la poziţia acestora!

Creați un nou proiect (sketch) Arduino şi introduceți următorul cod:

```
// Afișare pe SSD
// conectat la PORTA

// Tabelă de valori (LUT) cu codurile pentru fiecare cifră de
//la 0 la 9. Fiecare bit corespunde unui LED, 1 înseamnă LED
//aprins, 0 înseamnă LED stins.

const unsigned char ssdlut[] = {0b00111111, 0b00000110,
0b01011011, 0b01001111, 0b01100110, 0b01101101, 0b01111101,
0b00000111, 0b01111111, 0b01101111};
// dimensiunea LUT-ului
const int lutsize = 10;

int cpos = 0; // poziția curentă
int cdigit = 0; // prima cifră din cele două
unsigned char outvalue = 0;

void setup() {
    // activare PORTA ca ieșire
    DDRA = 0b11111111;
}

void loop() {

    outvalue = cdigit>0 ? 0x80 : 0;
    // care catod îl aprindem ? (00000000 sau 10000000)
    // catodul este legat la bitul 7 din portul A, prin
    //această operație punem
    // bitul 7 pe 1 sau pe 0, alternativ, urmând ca ceilalți
    //biți să fie atașați
    // prin OR logic, în linia de cod următoare

    PORTA = (ssdlut[cpos] | outvalue); // facem OR între
    valoarea din LUT și catodul selectat

    cpos++; // incrementăm poziția curentă

    if (cpos>=lutsize) { // dacă am ajuns la capăt
        cpos = 0; // revenim la zero
        cdigit^=1; // dacă cifra anterioară a fost 0, acum e
        //1, dacă a fost 1, acum e zero (^ = XOR)
    }

    // așteptare 0.5 sec
    delay(500);
}
```

Acest program va afișa cifre de la 0 la 9 pe primul element, apoi același lucru pe al doilea element.

2.4. Shield-ul pentru învăţare – Arduino Learning Shield

Shield-urile sunt PCB-uri (plăci de circuit imprimat – Printed Circuit Board) care pot fi plasate deasupra plăcilor Arduino, extinzându-le astfel capabilitățile. Există o varietate foarte mare de astfel de shield-uri, precum: XBee shield, SD-Shield, H-Bridge shield, etc. Pe PCB-urile Shield-urilor, anumite fire sunt trase către baretele de pini care urmează sa fie introduse în Arduino. Așadar, trebuie să fim atenți la pinii folosiți de componenta electronică, pentru a o putea mapa corect în program și pentru a nu avea conflicte în utilizarea acestor pini.

În această lucrare de laborator vom utiliza un shield de învăţare, prezentat în figura de mai jos. Acesta dispune de următoarele resurse:

- 1 afişor cu 7 segmente, cu patru cifre
- 3 butoane (+ buton de reset)
- 4 leduri
- 1 potenţiometru pentru generare de semnal analogic
- 1 generator de sunet (buzzer)

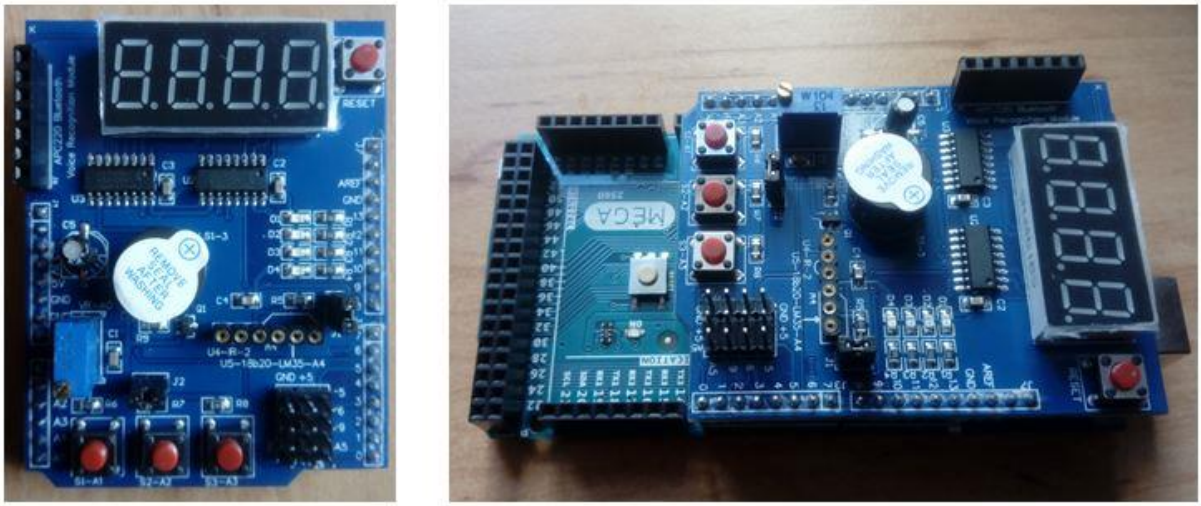


Fig.II.9, Fig.II.10 – Shield-ul pentru învățare și Arduino

Schema electrică a acestui shield se poate descărca de aici:
https://ardushop.ro/en/index.php?controller=attachment&id_attachment=40

Resursele shield-ului pot fi accesate astfel:

Butoanele sunt conectate la pinii A1, A2 si A3. Aceste butoane dispun de rezistenţe Pull-Up pe shield, astfel încât în starea liberă pe pini va fi citit nivelul logic 1, iar la apăsarea butonului se va citi nivelul logic 0. Nu este nevoie de activarea rezistorilor Pull-Up interni.

Led-urile sunt conectate cu anodul la VCC și **catozii la pinii 10, 11, 12 și 13**. Astfel, pentru aprinderea unui LED trebuie scris nivelul logic 0 pe pinul corespunzător.

Afişorul 4x7 segmente este organizat în modul anod comun şi catozi separaţi pentru fiecare segment de cifră, spre deosebire de PMod SSD, unde segmentele au catodul comun și anozii separați. Astfel, pentru aprinderea unei cifre trebuie setat anodul cifrei pe nivelul logic 1 şi catozii segmentelor care trebuie aprinse pe zero.

Pentru comanda celor patru cifre, vom avea deci nevoie de 4 semnale pentru anozi şi opt semnale pentru catozi (7 segmente şi punctul). Aceste semnale nu sunt disponibile direct pentru utilizator, ci sunt conectate la doi regiştri de deplasare, conform schemei din figura de mai jos. Regiştrii de deplasare sunt legaţi în serie, ieşirea registrului de anozi fiind conectată la intrarea registrului pentru catozi. Astfel, pentru scrierea unei configuraţii complete (selecţie cifră activă + selecţia segmentelor care vor fi aprinse), este nevoie **doar de trei semnale**:

- SDI – serial data in, conectat la **pinul 8**
- SFTCLK – shift clock, semnalul de ceas pe care se vor prelua datele, conectat la **pinul 7**
- LCHCLK – latch clock pin, semnalul care va permite introducerea datelor, conectat la **pinul 4**.

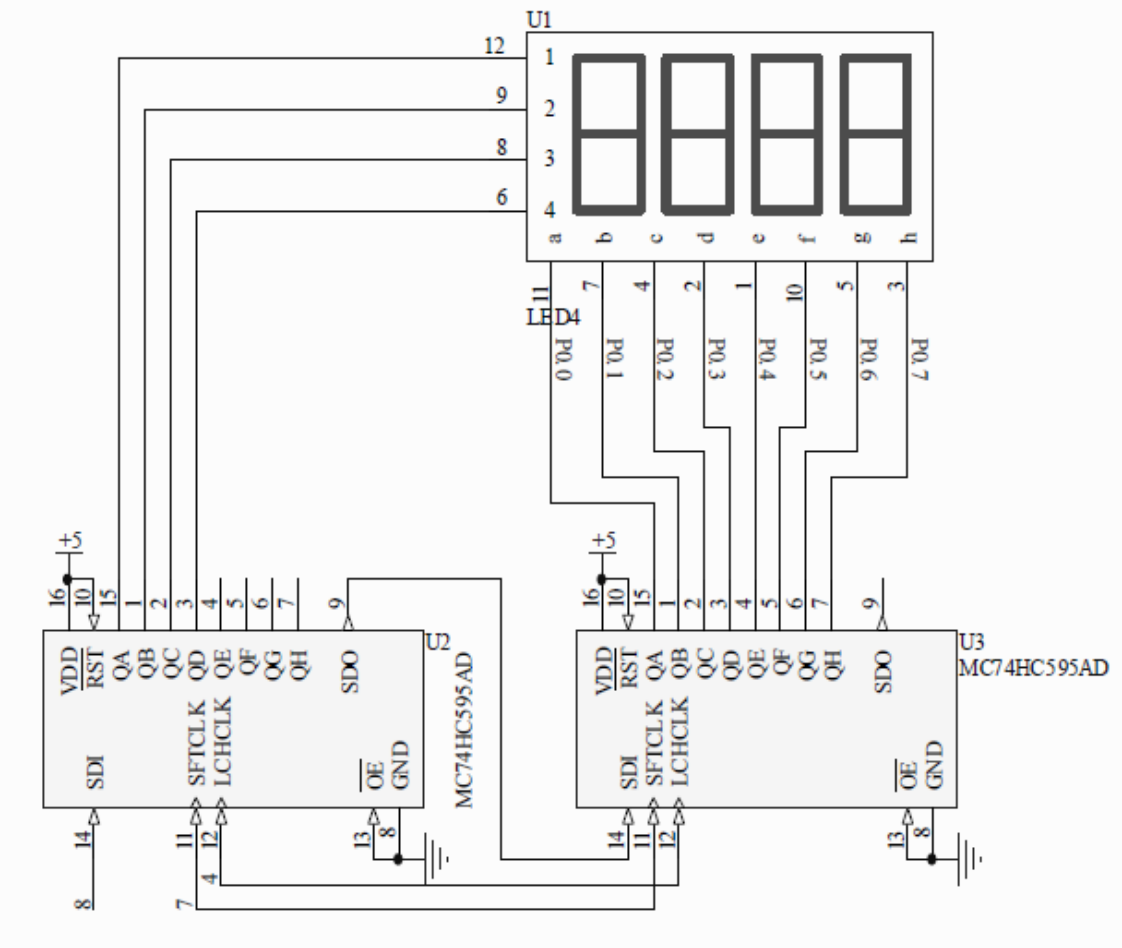


Fig.II.11. Structura internă a sistemului SSD al shield-ului de învăţare (sursa: https://ardushop.ro/en/index.php?controller=attachment&id_attachment=40)

Pentru scrierea unei cifre, trebuie ca prima dată să se transmită serial 8 biţi conţinând configuraţia catozilor, iar apoi 8 biţi pentru configuraţia anozilor (din care doar 1 bit trebuie să fie 1 pentru cifra activă).

Următorul exemplu ilustrează utilizarea afişorului 4x7 segmente:

```
int latchPin = 4;
int clockPin =7;
```

```
int dataPin = 8; // Pinii SSD

const unsigned char ssdlut[] = {0b00111111, 0b00000110,
0b01011011, 0b01001111, 0b01100110, 0b01101101, 0b01111101,
0b00000111, 0b01111111, 0b01101111};
const unsigned char anodelut[] = {0b00000001, 0b00000010,
0b00000100, 0b00001000};

const unsigned char digits[] = {1,2,3,4}; // Numărul afişat va
//fi 1234. Modificaţi aici pt alt număr

void setup ()
{

    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    pinMode(dataPin,OUTPUT); // Cei trei pini pentru registrii
   // de deplasare, configuraţi ca ieşire

}

void loop()
{

    for(char i=0; i<=3; i++) // pentru fiecare din cele 4
    //cifre
    {

        unsigned char digit = digits[i];  // cifra curentă
        unsigned char cathodes = ~ssdlut[digit]; // catozii
        //cifrei curente, vom nega valoarea din LUT

        digitalWrite(latchPin,LOW); // vom activa semnalul
        //latch pentru a permite scrierea
        shiftOut(dataPin,clockPin,MSBFIRST, cathodes); //
        //serializăm octetul anozilor
        shiftOut(dataPin,clockPin,MSBFIRST, anodelut [i] );
        // serializăm octetul anozilor
        digitalWrite(latchPin,HIGH); // dezactivăm semnalul
        //latch
        delay(2);   // aşteptare
    }

}
```

Analiza codului de mai sus:

Utilizăm acelaşi LUT ca în exemplul precedent, dar aici vom aplica o operaţie de negare pe biţi, pentru a ilustra faptul că acum se activează catozii, care necesită semnal 0.

Arduino oferă funcţia shiftOut, care transmite un octet în mod serial, pe pinul dataPin, generând şi un semnal de ceas pe pinul clockPin. Această funcţie poate fi utilizată cu oricare

doi pini digitali. Există două moduri de a serializa un octet: începând cu bitul 7 (MSBFIRST), sau începând cu bitul 0 (LSBFIRST). În acest caz, e nevoie să se aleagă varianta MSBFIRST, deoarece e important ca biții catozilor să corespundă cu segmentele cifrelor.

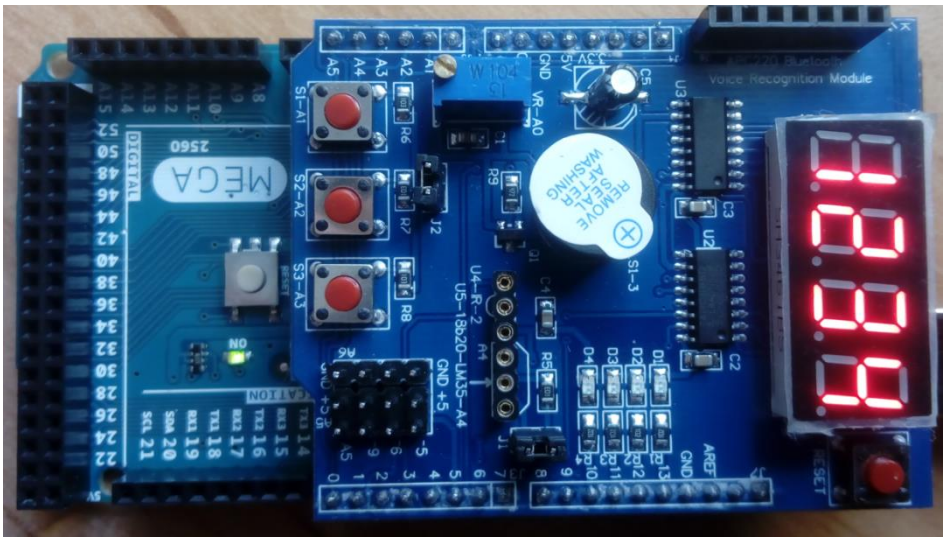


Fig.II.12. Rezultatul programului

2.5. Lucru individual

1. Realizaţi montajul şi rulaţi primul exemplu din acest document.
2. Completaţi LUT-ul cu valori pentru numere hexazecimal (A, B, C, D, E). Testaţi programul cu numere de acest fel.
3. Adaptaţi programul pentru afişarea oricărui număr de la 0 la 99 pe afişor, în format decimal. Ţineţi cont că numărul nu este acelaşi lucru cu reprezentarea lui ca cifre separate pe jumătăţi de octet (acest lucru este însă valabil în cazul numerelor hexazecimale). De exemplu, numărul 16 este reprezentat în binar ca 00010000, care va duce la afişarea numărului ‘10’. Pentru afişarea corectă a numerelor zecimale, trebuie făcuţi următorii paşi:
- aflaţi cifra zecilor, câtul împărţirii cu 10
- aflaţi cifra unităţilor, restul împărţirii cu 10

CZ = numar div 10
CU = numar mod 10 = numar – (CZ*10)

4. Rulaţi al doilea exemplu, care utilizează Learning Shield.
5. Modificaţi exemplul 2, pentru a afişa orice număr de 4 cifre, în format zecimal. Numărul va fi dat ca **int**, nu ca cifre separate. Faceţi ca numărul să se incrementeze periodic.
6. Utilizaţi butoanele de pe Learning Shield. Folosiţi un buton pentru incrementarea numărului afişat şi alt buton pentru decrementare.
7. Folosiţi afişorul de 4 cifre şi butoanele (Learning Shield), pentru a realiza un cronometru ce numără secundele (2 cifre) şi sutimile de secundă (2 cifre). Folosiţi un buton pentru start, unul pentru stop şi unul pentru reset. **Indicaţie: folosiţi funcţia millis() .**

III. Laborator 3 – Utilizarea afișorului LCD și a sistemului de întreruperi

3.1. Utilizarea shield-ului LCD

Shield-urile sunt PCB-uri (plăci de circuit imprimat – Printed Circuit Board) care pot fi plasate deasupra plăcilor Arduino, extinzându-le astfel capabilitățile. Există o varietate foarte mare de astfel de shield-uri, precum: XBee shield, SD-Shield, H-Bridge shield, etc. Pe PCB-urile Shield-urilor, anumite fire sunt trase către baretele de pini care urmează sa fie introduse în Arduino. Așadar, trebuie să fim atenți la pinii folosiți de componenta electronică, pentru a o putea mapa corect în program și pentru a nu avea conflicte în utilizarea acestor pini. Pinii folosiți sunt în general specificați în foaia tehnică a produsului. Figura de mai jos conține câteva exemple de shield-uri (shield LCD, un shield SD card și un shield de recunoaștere vocală EasyVR).

Fig.III.1. Shield-uri diferite

Dat fiind faptul că plăcile Arduino au avut un succes foarte mare pe piață, mulți dintre competitorii platformei de dezvoltare au folosit același factor de formă când au proiectat plăcile lor. Acest lucru este avantajos întrucât și alte plăci, cu alte procesoare, pot folosi *anumite* shield-uri Arduino. În figura de mai jos puteți observa alte plăci de dezvoltare care au un design similar cu cel al plăcilor Arduino (placa din stânga este un Fez Domino cu un procesor ARM, care se programează în .Net, iar placa din dreapta este un Chip Kit care conține un procesor de tip PIC și este programabilă în c++).

Fig.III.2. Multiple plăci de dezvoltare cu aspect similar cu Arduino

În acest laborator vom folosi shield-ul LCD, care conține un afișor cu cristale lichide și un potențiometru pentru reglarea intensității luminii.

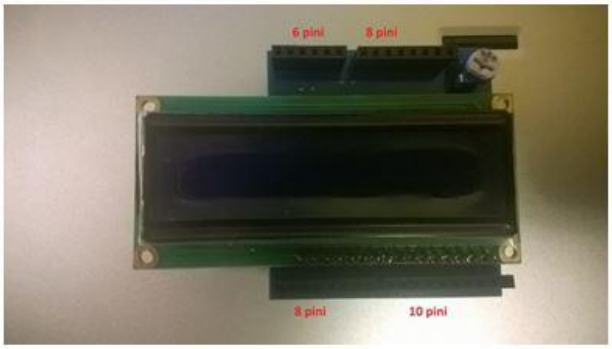


Fig.III.3, Fig.III.4 – Shield-ul LCD

Shield-ul este amplasat deasupra plăcii Arduino Mega, astfel încât baretele mai lungi (8, respectiv 10 pini) să fie în dreptul pinilor digitali, iar cele scurte în dreptul pinilor analogici, ca în figura de mai jos.

NU scoateți shield-ul de pe placa Arduino (exista riscul să îndoiţi pinii acestuia) !!!

Fig.III.5. Montarea shield-ului LCD la Arduino

LCD-ul utilizează pinii digitali de la 2 la 7, astfel: pinul digital 7 - RS; pinul digital 6 - EN; pinul digital 5 - DB4; pinul digital 4 - DB5; pinul digital 3 - DB6; pinul digital 2 - DB7 (explicația semnalelor pe pagina următoare).

Shield-ul este bazat pe controllerul clasic, care se folosește la LCD-uri, Hitachi HD44780. LCD-urile care folosesc cel mult 80 de caractere distincte și cel mult 4 linii de afișare au nevoie de un singur controller. LCD-urile care au mai mult de 80 de caractere au nevoie de

două controlere. Imaginea cu pinout-ul LCD-ului şi semnificația pinilor o găsiți în figura și tabelul de pe pagina următoare.

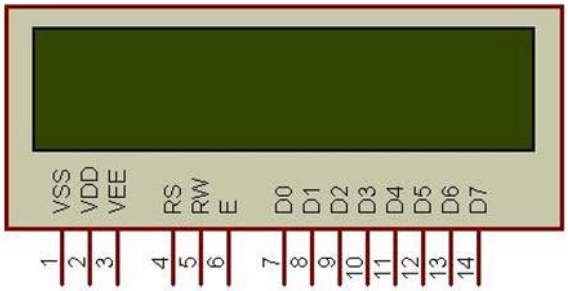


Fig.III.6. Pinii LCD-ului (sursa: https://www.pantechsolutions.net/interface-cards-tutorials/user-manual-for-lcd-interface-card)

Pin Nr	**Nume**	**Descriere**
1	**VSS**	Power supply (GND)
2	**VCC**	Power supply (+5V)
3	**VEE**	Contrast adjust
4	**RS**	Register Select 0 = Instruction input 1 = Data input
5	**R/W**	0 = Write to LCD module 1 = Read from LCD module
6	**EN**	Enable signal
7	**D0**	Data bus line 0 (LSB)
8	**D1**	Data bus line 1
9	**D2**	Data bus line 2
10	**D3**	Data bus line 3
11	**D4**	Data bus line 4
12	**D5**	Data bus line 5
13	**D6**	Data bus line 6
14	**D7**	Data bus line 7(MSB)

Fig.III.7. Semnificația pinilor LCD

Controllerul HD44780 conține două registre pe 8 biți: registrul de date și registrul de instrucțiuni. Registrul de instrucțiuni e un registru prin care LCD primește comenzi (shift, clear etc). Registrul de date este folosit pentru a acumula datele care vor fi afișate pe display. Când semnalul Enable al LCD-ului este activat, datele de pe pinii de date sunt puse în registrul de date, apoi mutate în DDRAM (memoria de afișaj, Display Data RAM) și afișate pe LCD. Registrul de date nu este folosit doar pentru trimiterea datelor către DDRAM ci și către CGRAM, memoria care stochează caracterele create de către utilizator (Character Generator RAM).

Display Data Ram (DDRAM) stochează datele de afișare, reprezentate ca și caractere de 8 biți. Capacitatea extinsă a memoriei este de 80 X 8 biți, sau 80 de caractere. Memoria rămasă liberă poate fi folosită ca un RAM generic. Pe LCD-ul nostru se afișează doar 2x16 caractere, deoarece aceasta este dimensiunea afișorului, dar controllerul poate stoca 80.

Forma grafică efectivă a unui caracter afișat pe LCD este dată de conținutul memoriei CGROM (Character Generator Read Only Memory). Această memorie conține matricele de puncte pentru fiecare caracter (5x8 puncte sau 5x10 puncte – depinde de dispozitiv).

Fig.III.8. Codul caracterelor pentru caractere formate din 5 x 8 puncte (sursa: https://www.sparkfun.com/datasheets/LCD/HD44780.pdf)

Aspectul caracterelor de la 0x00 la 0x07 poate fi definit de utilizator. Se va specifica pentru fiecare caracter o matrice de 8 octeți, câte unul pentru fiecare rând. Cei mai puțini semnificativi 5 biți din fiecare rând vor specifica care pixeli vor fi aprinși și care nu (vezi exemplul de mai jos).

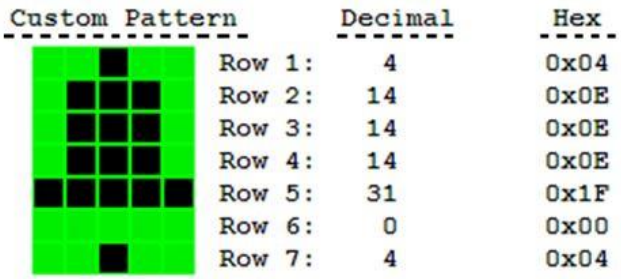


Fig.III.9. Un exemplu de model creat de utilizator (source: https://omerk.github.io/lcdchargen/)

Afișoarele LCD pot comunica cu microcontrollerul în două moduri: pe 8 biți și pe 4 biți. De obicei se preferă conectarea pe 4 biți, pentru că sunt mai putini biți de interfațat și, în consecință, rămân mai mulți pini pentru alte aplicații. *Shield-ul LCD pe care îl veți utiliza este gata configurat pentru a folosi comunicarea pe 4 biți*. Acest mod folosește doar 7 pini de pe placa Arduino; modul pe 8 biţi ar fi folosit 11;

Exemplu: afișarea șirurilor de caractere și a valorilor numerice pe LCD

În primul exemplu vom afișa un șir de caractere pe prima linie a LCD-ului și numărul de secunde care au trecut de la începerea programului.

```
//include biblioteca pentru lucrul cu LCD
#include <LiquidCrystal.h>
/* LCD RS pinul digital 7
 * LCD Enable pinul 6
 * LCD D4 pinul 5
 * LCD D5 pinul 4
 * LCD D6 pinul 3
 * LCD D7 pinul 2
Al șaptelea pin este pinul de la potențiometrul de reglare a
iluminării  */
//inițializează lcd-ul la valorile stabilite ale pinilor
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
unsigned long time;

void setup()
{
    //setează numărul de rânduri și coloane ale LCD-ului
    lcd.begin(16, 2);
}

void loop()
{
    //luăm numărul de secunde trecute de la reset (sau
    //pornirea programului)
    time = millis() / 1000;
    //setăm cursorul la coloana 0 rândul 1
    lcd.setCursor(0, 0);
    //scriem un text
    lcd.print("Hello Children");
    //setăm cursorul la mijlocul liniei a doua
    lcd.setCursor(7, 1);
    //scriem timpul
    lcd.print(time);
}
```

În figura următoare puteți vedea rezultatul:

Fig.III.10. Rezultatul afișării pe LCD

Exemplu: generarea de caractere utilizator
Acest exemplu ilustrează crearea caracterelor de către utilizator și folosirea lor.

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);

//  Masca  pentru  primul  caracter,  fiecare  linie  de  biţi
//reprezintă o linie a caracterului
byte happy[8] = {
    B00000,
    B11011,
    B11011,
    B00000,
    B00000,
    B10001,
    B01110,
};
// Masca pentru al doilea caracter
byte sad[8] = {
    B00000,
    B11011,
    B11011,
    B00000,
    B00000,
    B01110,
    B10001,
};
void setup() {
    lcd.begin(16, 2);
    // cele două caractere sunt stocate în CGROM, zona
    //utilizator, poziţiile 0 şi 1
    lcd.createChar(0, happy);
    lcd.createChar(1, sad);
    // Afişare prima linie, un text standard urmat de primul
    //caracter utilizator
    lcd.setCursor(0, 0);
    lcd.print("Happy ");
```

```
    lcd.write(byte(0)); // Observaţi diferenta dintre print şi
    //write
    /* când referiţi caracterul „0” trebuie să faceţi un cast
    la byte. Altfel compilatorul va semnala o eroare. Excepţie
    este cazul în care referiţi o variabilă:
        byte zero = 0;
        lcd.write(zero);
    */

    // Afişare pe a doua linie
    lcd.setCursor(0, 1);
    lcd.print("Sad ");
    lcd.write(1); // când referiţi caractere diferite de „0”
    //nu mai este necesar cast-ul;
}

// Funcţia loop rămâne nefolosită, puteţi să o folosiţi pentru
//a extinde funcţionalitatea
void loop()
{
}
```

3.2. Folosirea sistemului de întreruperi

Întreruperile sunt evenimente care necesită atenția imediată a microcontrollerului. În momentul în care se întâmplă un eveniment care are ca efect declanșarea unei întreruperi, microcontrollerul oprește programul pe care îl execută și începe să se ocupe de întrerupere, executând o ISR (Interrupt Service Routine), o procedură atașata întreruperii respective. Pentru ca microcontrollerul să răspundă la cereri de întreruperi, trebuie activat bitul care controlează sistemul global de întreruperi (Global Intrerupt Enable) și bitul corespunzător întreruperii. Următoarele lucruri sunt esențiale pentru ca sistemul de întreruperi să poată fi folosit în mod corect:

- Întreruperea trebuie să fie activată prin bitul ei corespunzător.
- Bitul Global corespunzător întreruperilor (bitul I) din SREG trebuie și el activat.
- Stiva trebuie inițializată. Acest lucru se realizează în mod automat dacă se lucrează în mediul Arduino.
- Fiecare rutină se finalizează cu instrucțiunea RETI. În acel moment microcontrollerul știe să se întoarcă la execuția programului întrerupt. La Arduino, instrucțiunea este plasată în mod automat de compilator.

Întreruperile sunt de două feluri: interne și externe. Întreruperile interne sunt asociate cu perifericele microcontrollerului (Timer/Counter, Analog comparator etc.) Întreruperile externe sunt declanșate de pini externi (de exemplu, la acești pini se pot atașa butoane).

Fiecare AVR are o listă de întreruperi, care include tipul de eveniment care va declanșa întreruperea. În momentul în care întreruperile sunt activate și unul dintre aceste evenimente se întâmplă, procesorul va realiza un salt în memoria program la o anumită locație/adresă (referită de vectorul întreruperii, adresă pe care o va găsi în lista/tabela de întreruperi). Scriind o procedură ISR și apoi făcând o legătură spre această procedură la adresa întreruperii

corespunzătoare, putem să determinăm sistemul să realizeze o acțiune specifică în momentul în care un anumit eveniment se produce.

În primul exemplu vom folosi o întrerupere declanșată de butoane (externă) și vom afișa pe LCD un mesaj corespunzător butoanelor. În momentul în care butonul este apăsat se declanșează cererea de întrerupere și programul afișează un mesaj pe LCD. Pentru acest exemplu aveți nevoie de placa Arduino Mega, un afișor LCD, un bloc de butoane PModBtn și de bread board. ATmega 2560 are 6 pini de întrerupere. Pentru a vedea lista de pini și întreruperile corespunzătoare accesați: http://arduino.cc/en/Hacking/PinMapping2560.

Vom conecta butoanele la pinii corespunzători întreruperilor INT0 și INT1, adică pinii digitali 20 și 21. În figura următoare se observă o imagine a conexiunilor necesare, iar mai jos codul corespunzător programului.

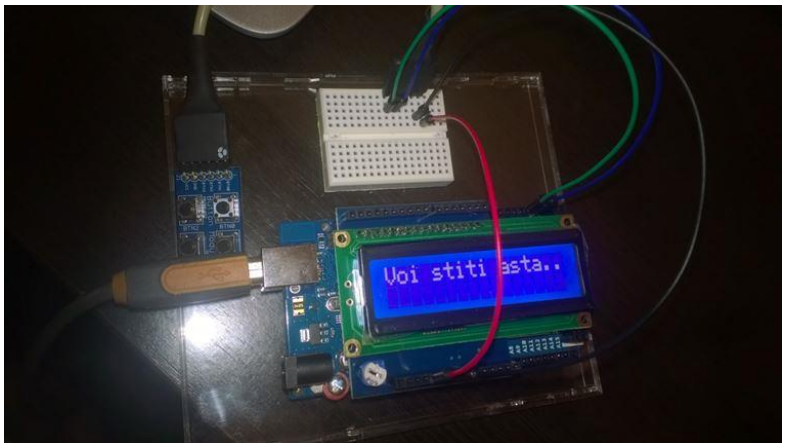


Fig.III.11. Conexiunile necesare primului exemplu

```
//includem headerul responsabil pentru operații cu întreruperi
pentru avr
#include "avr/interrupt.h"
//include biblioteca de manipulat LCD
#include <LiquidCrystal.h>
//inițializează lcd-ul la valorile stabilite ale pinilor
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
volatile int buttonVariable; //variabila publica ce poate fi
//modificata de o intrerupere

void setup(void)
{
    buttonVariable = 0; //initializăm variabila shared între
    //ISR și programul principal

    //setează numărul de rânduri și coloane ale LCD-ului
    lcd.begin(16, 2);
    lcd.print("Incepe experimentul");
    delay(1000);//facem o scurtă pauză pentru a vizualiza
    //mesajul de pe ecran
```

```
    //setăm pinul 21 ca și pin de intrare; pinul pe care se
    //află întreruperea INT0
    pinMode(21 ,INPUT);
    //setăm pinul 20 ca și pin de intrare; pinul pe care se
    //afla întreruperea INT1
    pinMode(20, INPUT);

    pinMode(13, OUTPUT); //setăm pinul 13 ca și iesire
    digitalWrite(13, HIGH); //aprindem led-ul ataşat la pinul
    //13

    delay(1000);

    EIMSK |= (1 << INT0); //activăm punctual întreruperea INT0
    EIMSK |= (1 << INT1); //activăm punctual întreruperea INT1

    EICRA |= (1 << ISC01); //activăm întreruperea 0 pentru
    //front descrescător.
    EICRA |= (1 << ISC11); //ca și mai sus, pentru
    //întreruperea 1
    sei(); //activăm întreruperile la nivel global

    digitalWrite(13, LOW); // stingem led-ul ataşat la pinul
    //13

    lcd.clear(); //ștergem ecranul LCD
}

void loop()
{
    //dacă a fost executată o ISR trebuie să ștergem ecranul
    //și să afișăm iar mesajul
    // principal
    if(buttonVariable == 1)
    {
        lcd.clear(); //se șterge ecranul LCD
        buttonVariable = 0; // variabila este re-inițializată
    }

    delay(1000);
    lcd.setCursor(0,0); //setăm cursorul
    lcd.print("Liniste…"); //afișăm un mesaj
}

//Rutina pentru tratarea întreruperii atașată la INT0
ISR(INT0_vect)
{
    digitalWrite(13, !digitalRead(13)); //schimbă starea pin
    //13
    lcd.setCursor(0,0); //poziționăm cursorul stânga sus
```

```
    lcd.print("Intrerupem"); //afişăm mesaj
    lcd.setCursor(0,1);
    lcd.print("ptr stirea zilei");
    buttonVariable = 1;
}

//Rutina pentru tratarea întreruperii ataşată la INT0
ISR(INT1_vect)
{
    digitalWrite(13, !digitalRead(13));
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Stirea Doi");
    buttonVariable = 1;
}
```

Este bine ca o procedură de tip ISR să poată fi executată cât mai repede, deoarece programul principal este oprit în aceste momente, aşteptând terminarea procedurii.

Ţineţi minte că delay() şi millis() nu funcţionează „pe parcursul" unei rutine de întrerupere. Aceste funcţii folosesc şi ele sistemul de întreruperi, iar acesta este dezactivat atunci când suntem într-o procedură de tip ISR. În general, în timpul execuţiei unei ISR sistemul nu poate răspunde la alte cereri. Din acest motiv, este recomandat ca aceste proceduri să poată fi executate într-un timp foarte scurt.

Dacă se doreşte modificarea unei variabile în interiorul unei proceduri ISR şi doriţi ca valoarea modificată a acestei variabile să fie vizibilă în tot programul, declaraţi această variabilă ca **volatile**. Prin această declaraţie, compilatorul ştie că variabila poate fi modificată în orice moment, renunţând la eventuale optimizări şi plasând variabila în memoria RAM.

Arduino ne permite să utilizăm sistemul de întreruperi şi fără a avea cunoştinţe despre mecanismul specific microcontrollerului, punând la dispoziţie anumite funcţii generice. Prima funcţie pe care o vom prezenta este **attachIntrerrupt().** Funcţia aceasta are rolul de a ataşa o funcţie ISR la o întrerupere externă, înlocuind orice funcţie precedentă care a fost ataşată anterior şi de a activa întreruperea. Sintaxa este: **attachInterrupt(interrupt, ISR, mode).** Primul parametru al funcţiei attachInterrupt reprezintă numărul întreruperii.

Din păcate, parametru **interrupt** nu corespunde cu numărul întreruperii externe din microcontrollerul ATMega2560 şi nici cu pinul digital al plăcii. Este un simplu număr de ordine. Tabelul de mai jos ne arată diferenţa dintre parametrul **interrupt**, întreruperea externă a ATMega2560, pinul de pe chip şi pinul digital de pe placă.

attachInterrupt	Name	Pin on chip	Pin on board
		(TQFP)	
0	INT4	6	D2
1	INT5	7	D3
2	INT0	43	D21
3	INT1	44	D20
4	INT2	45	D19
5	INT3	46	D18

Fig.III.12. Numărul întreruperii vs. Pini (sursa: http://www.gammon.com.au/interrupts)

Din acest motiv, se recomandă utilizarea funcţiei **digitalPinToInterrupt**(pin), care va returna numărul întreruperii atașată pinului digital (dacă aceasta există).

Al doilea parametru al funcției **attachInterrupt** este procedura de tratare a întreruperii (ISR). Al treilea parametru este condiția de declanșare a întreruperii (front crescător: **RISING**, descrescător: **FALLING,** pe nivel 0: **LOW,** sau nivel 1: **HIGH**, sau la orice schimbare de nivel pe pin: **CHANGE**).

Funcția **noInterrupts()** dezactivează întreruperile. Acestea pot fi reactivate cu funcția **interupts()**.

Funcția **detachInterrupt()** dezactivează întreruperea cu numărul specificat ca parametru Sintaxa este detachInterrupt(*interrupt*), interrupt având aceeaşi semnificaţie ca în cazul funcției attachInterrupt().

Exemplul anterior a fost implementat folosind registre de configurare ale AVR, fiind astfel dependent de specificațiile microcontrollerului. În continuare vom prezenta un exemplu cu funcționalitate similară, realizat cu ajutorul funcțiilor specifice Arduino.

```
#include <LiquidCrystal.h>

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

void setup()
{
    buttonVariable = 0;
    lcd.begin(16,2);
    lcd.print("A inceput");
    lcd.setCursor(0,1);
    lcd.print("din nou");
    delay(1000);

    // cei doi pini de întrerupere, 21 și 20, declarați ca
    //intrare și rezistențe Pull-Up active
    pinMode(21 ,INPUT);
    pinMode(21 ,INPUT);
    digitalWrite(20, HIGH);
    digitalWrite(21, HIGH);
    // atașăm pinilor 21 și 21, corespunzători INT1 și INT0,
    //funcţii ISR
    attachInterrupt(digitalPinToInterrupt(20), functieUnu,
    RISING);
    attachInterrupt(digitalPinToInterrupt(21), functieDoi,
    CHANGE);
}

void loop()
{
```

```
    //aici sunt taskuri care se execută în mod normal cînd se
    //rulează programul
    lcd.print("Programul principal");
    delay(1000);
}

//prima procedură ISR
void functieDoi()
{
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Functia Doi");
}

//a doua procedură ISR
void functieUnu()
{
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Functia Unu");
}
```

3.3. Lucru individual

1. Rulați exemplele prezentate
2. Folosind generatorul de caractere, creați o animație care indică trecerea timpului (un ceas cu braț care se rotește, o clepsidră, etc)
3. Folosind întreruperile, creați un cronometru. Unul din butoane va porni/opri cronometru, iar celălalt buton va avea rol de reset.
4. Folosiți animația de la punctul 2 împreună cu cronometrul creat. Animația va rula cât timp cronometrul merge și se va opri când acesta e oprit.
5. Bonus: folosiţi o placă pentru a genera semnale de diferite frecvenţe şi cu diferiţi factori de umplere (timpul cât semnalul este 1), folosind modificarea stării unui pin (prin digitalWrite) la intervale de timp controlate prin delay() sau millis(). Folosiţi întreruperile externe de pe altă placă pentru a determina caracteristicile acestui semnal şi a le afişa pe LCD.

Referinţe:

1.https://www.arduino.cc/en/Reference/LiquidCrystal
2.https://www.arduino.cc/en/Reference/AttachInterrupt

IV. Laborator 4 – Utilizarea temporizatoarelor

4.1. Întreruperi bazate pe temporizatoare

În afara întreruperilor externe, studiate în laboratorul precedent, există şi întreruperi interne, care nu sunt cauzate de evenimente pe pinii exteriori, ci de componente hardware incluse în microcontroller. În această categorie sunt incluse întreruperile generate de temporizatoare (timers).

Prin utilizarea acestor întreruperi, puteți genera acțiuni la intervale precise de timp, fără a utiliza funcții de tip *delay* sau *millis*. Întreruperile de temporizatoare funcționează asincron, lucru care permite ca programul să execute bucla principală şi doar când s-a ajuns la un anumit prag de timp să execute o rutină specifică. Temporizatorul incrementează un registru numit counter register, iar în momentul în care se atinge valoarea maximă posibilă în acest registru, se setează un bit (flag) ce indică depăşirea acestei valori (overflow). Acest flag poate fi verificat manual, sau poate declanşa o cerere de întrerupere. La execuţia procedurii de tratare a întreruperii (ISR), acest flag va fi pus din nou la zero.

Fiecare temporizator are nevoie de o sursă de semnal de ceas (clock). În general se alege ca sursă oscilatorul plăcii, frecvenţa acestuia putând fi divizată printr-un numărător auxiliar (prescaler).

Arduino Mega are 5 temporizatoare (2 pe 8 biți și 3 pe 16 biți) care pot fi folosite. La Arduino UNO diferența este că există doar un timer pe 16 biți). Pentru a folosi aceste temporizatoare, va trebui să setăm valori specifice în registrele de configurare. Două dintre aceste registre sunt TCCRxA şi TCCRxB, unde x este numărul temporizatorului. TCCR vine de la Timer Counter Control Register. Pentru pornirea temporizatorului, cei mai importanţi biţi sunt cei trei biţi care aleg divizarea frecvenţei de intrare, reglând implicit prin aceasta viteza de numărare. În figurile de mai jos se observă structura registrelor A şi B şi posibile configurări pentru biţii de selecţie a frecvenţei.

TCCR1A – Timer/Counter 1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
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	

TCCR1B – Timer/Counter 1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig.IV.1. Registre de control pentru Timer 1 (sursa: Atmega2560 datasheet)

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Fig.IV.2. Selecția semnalului de ceas pentru Timer1 (sursa: Atmega2560 datasheet)

Implicit cei trei biţi sunt 0, ceea ce înseamnă că temporizatorul nu are semnal de ceas şi deci este oprit. O configurare validă pentru frecvenţa de intrare înseamnă pornirea temporizatorului. Dacă sursa de ceas selectată este de tip extern, temporizatorul va funcţiona doar dacă o astfel de sursă este conectată la placă (ceea ce de obicei nu e cazul, deci astfel de setări se vor evita).

Un sumar al celor mai importante registre pentru lucrul cu temporizatoarele:

- TCCRx - Timer/Counter Control Register. Aici se poate selecta sursa de ceas.
- TCNTx - Timer/Counter Register. Valoarea numărată este stocată aici (valoarea de moment a temporizatorului)
- OCRx - Output Compare Register – Valoare scrisă de utilizator, cu care se compară TCNTx, pentru a genera diferite unde sau evenimente.
- ICRx - Input Capture Register (doar pentru numărătoare pe 16 biţi) – folosit pentru a măsura timpul dintre evenimente externe.
- TIMSKx - Timer/Counter Interrupt Mask Register. Pentru activarea sau dezactivarea întreruperilor bazate pe temporizator.
- TIFRx - Timer/Counter Interrupt Flag Register. Indică prezenţa unei cereri de întrerupere.

În exemplul următor vom incrementa o variabilă în momentul în care temporizatorul TIMER1 va face overflow. Valoarea variabilei incrementate va fi afişată pe LCD.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <LiquidCrystal.h>

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

void setup()
{
    myVar = 0;
    //inițializarea primului numărător
```

```
    cli(); //facem disable la întreruperile globale pentru a
    //face
    //modificările corespunzătoare timerelor
    TCCR1A = 0; // SETĂM TCCR1A și B la 0
    TCCR1B = 0; // timer este setat în Normal mode (WGMx3:0 =
    //0)

    lcd.begin(16, 2);
    lcd.print("Timere");

    //facem enable la întrerupere de overflow pentru timerul 1
    TIMSK1 = (1 << TOIE1); //timer overflow întrerupt enable
    //for timer 1
    //setăm temporizatorul să ruleze o frecvență divizată cu
    //1024
    //DE MENȚIONAT CĂ FRECVENȚA PROCESORULUI e de 16 MHZ și
    //timer 1
    //este un temporizator de 16 biți
    //cu un prescaller de 1024  avem incrementare o data la
    //t = 1024 / (16 * 10^6) secunde
    //temporizatorul va da overflow la fiecare (t * 2^16) =
    //4.194 secunde

    TCCR1B |= (1 << CS10);
    TCCR1B |= (1 << CS12);

    //activăm întreruperile globale
    sei();
}

void loop()
{
    lcd.setCursor(0,1);
    lcd.print(myVar);
    lcd.setCursor(5, 1);
    lcd.print(TCNT1);
}

ISR(TIMER1_OVF_vect)
{
    myVar = myVar + 1;
}
```

Pentru a face ca întreruperea de temporizator să se declanşeze la un anumit moment dorit de utilizator şi nu doar când capacitatea registrului de numărare este depăşită, vom folosi un alt mod de declanşare a întreruperilor numit CTC (clear on timer compare match). În acest mod de lucru, valoarea registrului de numărare TCNTx se va compara cu registrul OCRx scris de utilizator şi la egalitate TCNTx va lua din nou valoarea zero.

Pentru a obţine o perioadă T între cererile de întrerupere, trebuie să scriem valoarea OCRx rezultată prin aplicarea următoarei ecuaţii:

OCRx + 1 = T / (P / (16 * 10^6)) **(1)**

În ecuaţia 1, T reprezintă timpul dorit între evenimente (de exemplu, 1 secundă), iar P reprezintă factorul de diviziune a frecvenţei (de exemplu 1024). Se aplică +1 la valoarea OCR deoarece resetarea valorii temporizatorului în momentul în care s-a atins valoarea dorită durează un ciclu de ceas.

Aplicând ecuaţia pentru o perioadă T de 1 secundă, vom obţine pentru OCRx valoarea 15624.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <LiquidCrystal.h>

LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
volatile int myVar;
void setup()
{
    // initialize Timer1
    cli();          // facem disable la întreruperile globale
    TCCR1A = 0;     // setăm TCCR1A și B la 0
    TCCR1B = 0;
    lcd.begin(16, 2);
    lcd.print("Timere cu CTC");
    // setăm registrul cu valoarea căruia vom compara TCNT
    OCR1A = 15624;
    // activăm modul CTC:
    TCCR1B |= (1 << WGM12);
    // divizăm ceasul plăcii cu 1024:
    TCCR1B |= (1 << CS10);
    TCCR1B |= (1 << CS12);
    // facem enable la întreruperea la comparare prin setarea
    //bitului
    // corespunzător din mască
    TIMSK1 |= (1 << OCIE1A);
    // validăm sistemul global de întreruperi
    sei();
}

void loop()
{
    lcd.setCursor(0,1);
    lcd.print(myVar);
    lcd.setCursor(5, 1);
    lcd.print(TCNT1);
}

ISR(TIMER1_COMPA_vect)
{
    myVar = myVar + 1;
}
```

Spre deosebire de exemplul anterior, în care incrementam variabila doar la overflow (la aproximativ 4 secunde), în acest exemplu avem mai mult control asupra perioadei de incrementare a variabilei noastre; variabila se incrementează la fiecare secundă.

În continuare vom vedea o metodă mai simplă pentru utilizarea temporizatoarelor, folosind biblioteca TimerOne. Exemplul de mai jos va activa o rutină la fiecare secundă. Biblioteca Timer One poate fi descărcată de pe site-ul Arduino, de la adresa http://playground.arduino.cc/Code/Timer1 , unde sunt date şi explicaţii detaliate, precum şi instrucţiunile de instalare.

```
#include <TimerOne.h>
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);

volatile int myVar;

void setup(void)
{
    Timer1.initialize(1000000); //se iniţializează intervalul
    //de timp la care temporizatorul va
    // declanşa evenimente (în microsecunde, 1000000
    //microsecunde = 1 secundă)

    Timer1.attachInterrupt(ShowMessage); // funcţia
    //ShowMessage se va apela la intevalul
    // stabilit
}

void ShowMessage(void)
{
    lcd.setCursor(0,0);
    lcd.print(myVar);
    myVar++;
}

void loop(void)
{
}
```

4.2. Generarea de tonuri de frecvenţă dată

Pentru generarea de tonuri, Arduino pune la dispoziţia utilizatorului funcţia **tone**, care generează un puls dreptunghiular de frecvenţă specificată şi factor de umplere 50%. La apelul acestei funcţii trebuie specificată durata semnalului, altfel tonul va continua să sune până la apelarea funcţiei **noTone()**. Nu este posibil să generaţi tonuri mai joase de 31 Hz. Daca doriţi să generaţi tonuri pe mai mulţi pini, va trebui să apelaţi funcţia **noTone()** înainte de a genera tonul pe un alt pin.
Sintaxa funcţiei este următoarea:

tone(pinul, frecventa, durata)

sau
tone(pin, frecventa)

Pentru a putea rula exemplul următor, realizaţi următoarea conexiune: conectaţi **pinul roşu** (de semnal) de la difuzor la **pinul digital 8 de pe placa Arduino**. **Pinul negru** al difuzorului trebuie conectat la masă (**GND**).

În continuare, creaţi un fişier nou numit **pitches.h**. Pentru a crea un fişier nou pentru un proiect, apăsaţi butonul din partea dreapta sus a ferestrei mediului de dezvoltare şi apăsaţi pe butonul "New Tab", ca în figura următoare.

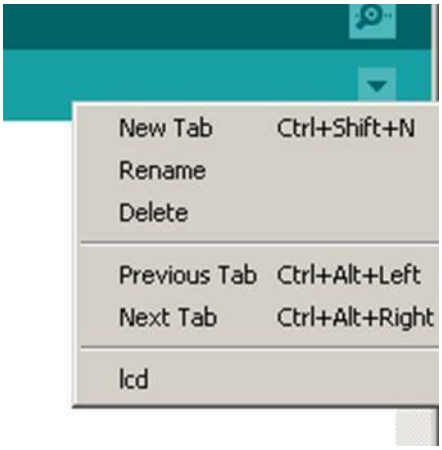


Fig.IV.3. Crearea unui fişier nou

Numiţi acest fişier pitches.h şi introduceţi în el tonurile definite în **anexa A** a acestui document. Salvaţi documentul şi reveniţi la tab-ul principal, unde veţi introduce următorul program:

```
//includem fișierul cu definițiile pentru tonuri
#include "pitches.h"

// melodia ca lista de note
int melody[] = {
    NOTE_C4,  NOTE_G3,NOTE_G3,  NOTE_A3,  NOTE_G3,0,  NOTE_B3,
NOTE_C4};

//introducem durata pentru fiecare notă din melodie
int noteDurations[] = {4, 8, 8, 4,4,4,4,4 };

void setup() {

    //pentru fiecare notă din vectorul melody
    for (int thisNote = 0; thisNote < 8; thisNote++) {
        //calculăm durata de afișare a notei
        int noteDuration = 1000/noteDurations[thisNote];
        //apelăm funcția de tone pentru difuzorul atașat la
        //pinul 8 și durata specificată
```

```
        tone(8, melody[thisNote],noteDuration);
        int pauseBetweenNotes = noteDuration * 1.30;
        delay(pauseBetweenNotes);
        noTone(8);
    }
}

void loop()
{
}
```

4.3. Generare de semnale PWM cu Arduino

PWM (pulse width modulation, modulaţie prin lăţimea pulsului) este o metodă de a obţine un semnal analogic prin intermediul unui semnal digital, alternând periodic ieşirea între nivelul logic 1 şi nivelul logic 0. Fracţiunea de perioadă cât semnalul este activ (1 logic) se numeşte factor de umplere, sau, în engleză, duty cycle. Cu Arduino se poate genera PWM în trei moduri: fie folosind direct modurile de lucru PWM ale temporizatoarelor, fie folosind funcţia **analogWrite,** fie variind prin program durata cât un pin este 1 logic (software PWM).

În continuare vom folosi funcţia **analogWrite (factor_de_umplere)**. Valorile posibile pentru argumentul funcţiei analogWrite sunt de 0, care reprezintă 0 Volţi, la 255, echivalentul Vcc (semnal permanent 1). Factorul de umplere de 50% se realizează pentru argumentul 127. În figura următoare se pot observa mai multe exemple de PWM.

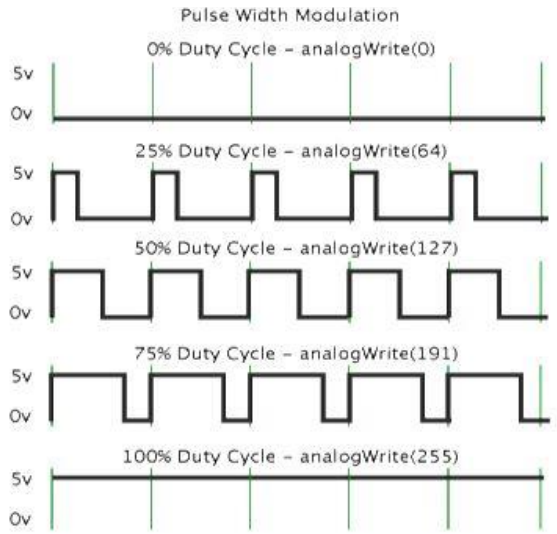


Fig.IV.4. Exemple PWM cu factor de umplere diferit (sursa: https://www.arduino.cc/en/Tutorial/PWM)

În exemplul care va urma vom genera un semnal PWM pentru generatorul de sunete piezoelectric şi un PWM, cu acelaşi factor de umplere, pentru LED-ul conectat pe placă. Pinul de semnal al difuzorului (firul roşu) se conectează la pinul digital 8, iar firul negru se conectează la GND.

```
int buzerPin = 8; //pinul la care atașăm generatorul de sunete
int puls = 0;    // factorul de umplere, inițial 0
int pas = 10;// pasul de incrementare al factorului de umplere
int ledPin = 13; //ledul de pe placă

void setup()  {
    // declararea pinilor ca ieșire
    pinMode(buzerPin, OUTPUT);
    pinMode(ledPin, OUTPUT);
}

void loop()  {
    // setăm factorul de umplere al buzerului și al ledului
    analogWrite(buzerPin, puls);
    analogWrite(ledPin, puls);
    // modificăm pwm-ul pentru următoarea iterație
    puls = puls + pas;

    // schimbăm direcția la capetele intervalului: din
    //incrementare devine decrementare și invers
    if (puls <= 0 || puls >= 255) {
        pas = -pas ;
    }
    // un mic delay pentru a vedea efectul
    delay(30);
}
```

4.4. Lucru Individual

1. Implementați toate exemplele din laborator. Întrebați cadrul didactic pentru orice nedumerire legată de conceptele din laborator sau conectivitatea cu placa.
2. Folosind întreruperile, realizaţi rularea melodiei în fundal, în mod repetat, în timp ce programul principal va rula o animaţie pe afişorul LCD. Animaţia poate fi simplă (gen: deplasarea unui caracter pe ecran). Viteza animaţiei va varia în timp, pentru a demonstra independenţa melodiei de programul principal.
3. Folosind blocul de butoane, realizaţi un mini-pian cu patru note. Asiguraţi-vă că sunetul generat se va opri la ridicarea butonului.
4. Folosind PWM şi un bloc de led-uri, realizaţi o animaţie prin care intensitatea fiecărui LED este variată în mod continuu.

Referințe:

1.Datasheet ATMega 2560: http://www.atmel.com/images/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf
2. Biblioteca Timer 1 http://playground.arduino.cc/Code/Timer1
3. Funcția analogWrite https://www.arduino.cc/en/Reference/AnalogWrite
4. Generarea de tonuri: https://www.arduino.cc/en/Reference/Tone

Anexa A

(conținutul fișierului pitches.h, from: https://www.arduino.cc/en/Tutorial/toneMelody)

```
#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
```

```
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
```

V. Laborator 5 – Interfețe de comunicație

Electronica embedded se referă la interconectarea de circuite (procesoare sau alte circuite integrate) cu scopul de a crea un sistem dedicat. Pentru ca aceste circuite să-și poată transfera informații trebuie să conțină o modalitate de comunicare comună. Deși există sute de modalități de comunicare, aceste modalități se împart în doua categorii: modalități **seriale și 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 și recepționate sunt fluxuri masive de 1 și 0. În figura V.1, observăm o magistrală de date cu lățimea de 8 biți, controlată de un semnal de ceas și transmite câte un byte la fiecare puls al ceasului (se folosesc 9 fire).

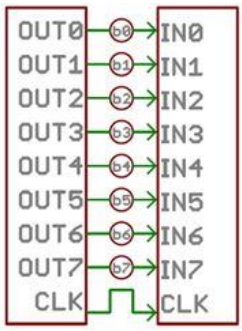


Fig.V.1. Transmisie paralelă (sursa: https://learn.sparkfun.com/tutorials/serial-communication/all)

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

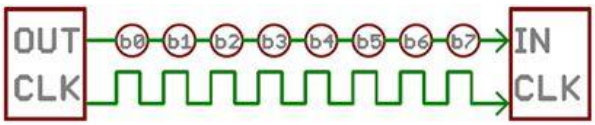


Fig.V.2. Transmisie serială (sursa: https://learn.sparkfun.com/tutorials/serial-communication/all)

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

O altă modalitate de clasificare a interfețelor de comunicație este după modalitatea de comunicație: **sincronă** sau **asincronă**. O interfață de comunicație sincronă folosește un semnal de ceas unic la ambele capete ale comunicației (emițător și receptor). Această modalitate de comunicație este de multe ori mai rapidă, însă, cu toate acestea, ea are nevoie de cel puțin un fir în plus între dispozitivele care comunică (pentru transmiterea semnalului de ceas). Exemple de astfel de comunicații sunt SPI și I2C. Transferul asincron se referă la faptul că datele sunt transferate fără

suportul unui semnal de ceas extern. În acest fel se elimină firul de ceas, dar o atenție sporită trebuie acordată 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 în stare inactivă 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 în stări pe secundă (de obicei o stare este 1 sau 0, deci un bit, dar există interfețe care pot avea mai mult de două stări și atunci *baud rate* nu este același lucru cu biți pe secundă). Există mai multe *baud rate*-uri standard, precum 1200, 2400, 4800, 19200, 38400, 57600, sau 115200.

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 biţi. Pachetele sunt create adăugând biți de sincronizare sau paritate datelor care urmează să fie transmise. Figura V.3 ilustrează aspectul pachetului de date:

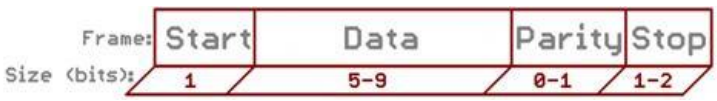


Fig.V.3. Pachet - Data Frame (sursa: https://learn.sparkfun.com/tutorials/serial-communication/all)

Data chunk

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

Biţi de sincronizare

Biții de sincronizare sunt biți speciali care sunt transferați cu fiecare caracter de date. Aceștia sunt biții de **start** și de **stop**; ei marchează începutul si finalul unui pachet. Tot timpul bitul de start este unul singur iar biții de stop sunt fie unul, 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 inactivă, ținând linia la nivelul 1 logic.

Biţi 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 dacă bitul este setat sau nu. Paritatea este o măsura de verificare opțională, care nu este prea folosită. Este util să folosim biții de paritate atunci când transmitem date în 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.V.4. Compunerea bitului de paritate (source: https://learn.sparkfun.com/tutorials/serial-communication/all)

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

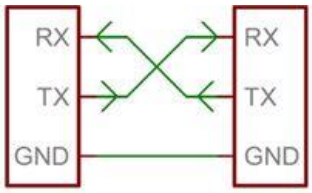


Fig.V.5. Schema de conexiune serială (source: https://learn.sparkfun.com/tutorials/serial-communication/all)

Toate plăcile Arduino conțin cel puțin un port **Serial** (cunoscut ca și UART sau USART). Comunicarea serială se poate realiza prin pinii 0 (RX) şi 1(TX), dar şi prin USB (interfaţa USB comunică cu microcontrollerul prin pinii RX0 şi TX0). Din acest motiv, pinii digitali 0 şi 1 nu trebuie niciodată utilizaţi pentru aplicaţii 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).

În 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. Există o varietate de funcții pentru manipularea datelor seriale. În laboratoarele precedente, am folosit o comunicație serială între Arduino și computer, când afișam starea butoanelor apăsate. Funcțiile/metodele cele mai uzuale pentru manipularea serială a datelor sunt prezentate în Figura V.6 (https://www.arduino.cc/en/Reference/Serial).

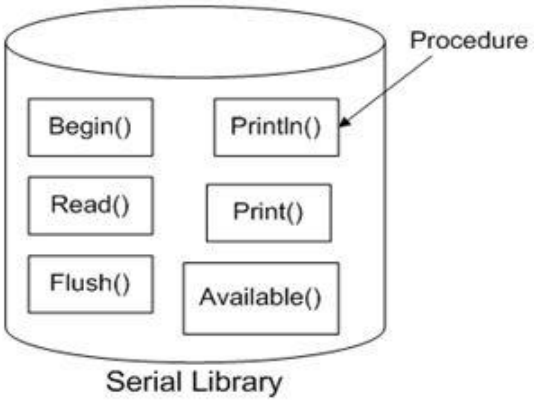


Fig.V.6. Funcții pentru comunicația serială asincronă (sursa: https://www.arduino.cc/en/Reference/Serial)

Funcțiile **print** și **println** ale clasei Serial trimit date pe portul serial. Diferența este că println() adaugă un caracter rând nou (‘\n’) și un caracter „carriage return” (‘\r’) la finalul mesajului transmis. Pentru numere transmise puteți specifica și un format de transmitere a datelor (HEX, DEC etc.).

Funcția **begin()** setează viteza de comunicație în biți/secundă (baud rate). Pentru comunicația cu computerul se folosesc, în general, următoarele viteze: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, sau 115200. Se mai poate adăuga un parametru opțional pentru configurarea pachetului de date: câți biți sunt, paritatea și numărul biților de stop. Implicit (dacă nu se specifică parametrul de configurare opţional) sunt setate următoarele valori: **8 biți de date, no parity, one stop bit.**

Funcția **read()** citește datele venite prin interfața serială. Sintaxa este următoarea:

IncomingByte = Serial.read();

Funcția **write()** trimite un octet sau o succesiune de octeți. Pentru a trimite totuși numere se recomandă folosirea funcției **print()**.

Instrucțiunea **flush()** așteaptă ca transmiterea serială de date să se finalizeze.

Funcția **available()** întoarce numărul de octeți care pot fi citiți de la portul serial. Aceste date au ajuns deja și sunt stocate în bufferul de recepție serială.

O funcție utilă pe care o vom folosi este **serialEvent().** Funcția este definită de utilizator și va fi apelată în mod automat în momentul în care apar date pentru a fi citite.

În exemplul de jos se citesc date venite pe interfața serială (de la PC prin serial monitor) și se afișează pe LCD.

```
//includem librăria de LCD
#include <LiquidCrystal.h>
String inputString = ""; // creăm un string care să ne țină datele
//care vin pe serial
// condiție pentru verificare dacă stringul este complet (s-a
//apăsat enter)
boolean stringComplete = false;
//inițializăm obiectul de tip lcd (vezi exemplul 1 pentru
//explicații asupra pinilor)
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);

void setup() {
    // inițializare interfață serială
    Serial.begin(9600); // format frame serial implicit
    // inițializare și setare lcd
    lcd.begin(16, 2);
    // rezervăm 200 de octeti pentru șirul de preluare a datelor
    //de intrare
    inputString.reserve(200);
}

void loop() {
    // afișăm stringul când primim new line
    if (stringComplete) {
        // setăm cursorul la coloana și rândul 0
        lcd.setCursor(0, 0);
        lcd.print(inputString);
        Serial.println(inputString);
        // golim șirul
        inputString = "";
        // resetăm contorul care verifică dacă stringul este sau
        //nu complet
        stringComplete = false;
    }
}

/*
SerialEvent este apelată de fiecare dată când date noi ajung pe
portul RX
Această funcție rulează de fiecare dată când rulează și loop. Deci
dacă am pune un delay în loop ne-ar întârzia și afișarea
rezltatului.
*/
void serialEvent() {
    while (Serial.available()) {
        // luăm byte-ul nou venit:
        // îl citim cu serial.read
        char inChar = (char)Serial.read();
```

```
        // verificăm dacă nu e cumva new line și dacă nu este, îl
        //adăugăm în inputString
        // nu adăugăm new line în input string întrucât ne va
        //afișa un caracter în plus pe lcd
        if (inChar != '\n')
        inputString += inChar;
        // dacă caracterul care vine este new line, setăm flagul
        // în așa fel încât loop-ul principal poate face ceva în
        //legătură cu asta
        if (inChar == '\n') {
            stringComplete = true;
        }
    }
}
```

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

Fig.V.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" să comunice cu unul sau mai mute cipuri "master". Acest tip de comunicare a fost intenționat pentru a fi folosit doar pe distanțe mici de comunicare și asemenea protocolului UART sau RS232 are nevoie doar de 2 fire de semnal pentru a trimite/primi informații.

Spre deosebire de UART, I2C permite comunicarea între mai mult de două dispozitive. Unul dintre dispozitive va fi *master*, iar el va comunica cu un *slave*, dar ulterior rolurile se pot schimba.

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

Spre deosebire de alte metode de comunicație serială, magistrala I2C este de tip "open drain", ceea ce înseamnă că pot trage o anumită linie de semnal în 0 logic, dar nu o pot conduce spre 1 logic. Așadar, se elimină problema de „bus contention", unde un dispozitiv încearcă să tragă una dintre linii în 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".

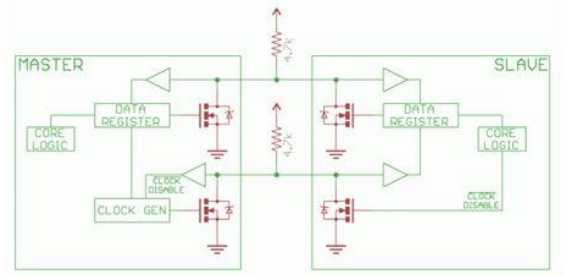


Fig.V.8. Schema de conectare I2C. Se remarcă folosirea a 2 rezistențe pe liniile de semnal (sursa: https://learn.sparkfun.com/tutorials/i2c/all)

Selecția rezistențelor variază cu dispozitivele care folosesc busul, dar o regula bună este de a începe cu rezistențe de 4.7k și scăderea lor dacă e necesar.

Descrierea protocolului:

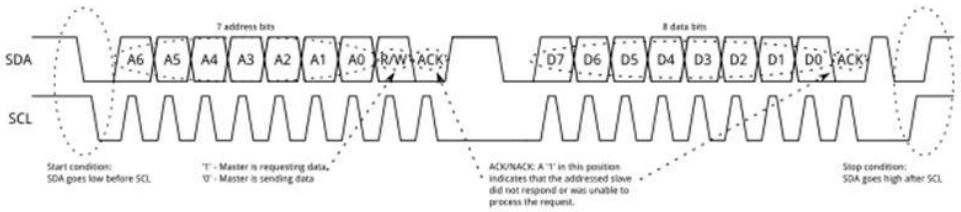


Fig.V.9. Protocolul de transmisie I2C (sursa: https://learn.sparkfun.com/tutorials/i2c/all)

Mesajele sunt sparte în 2 tipuri de cadre (frames): cadre de adresă, unde *master*-ul 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 și sunt eșantionate când SCL ajunge HIGH. Timpul între 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 adresă, 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 câștigă arbitrajul și implicit controlul busului.

Cadrul de adresă (Address Frame)

Cadrul de adresă 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 adresă). După ce primii 8 biți ai 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, fie nu a știut cum să 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 să fie transmise**. ***Masterul* va continua să 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, *master*-ul 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 V.10 de mai jos:

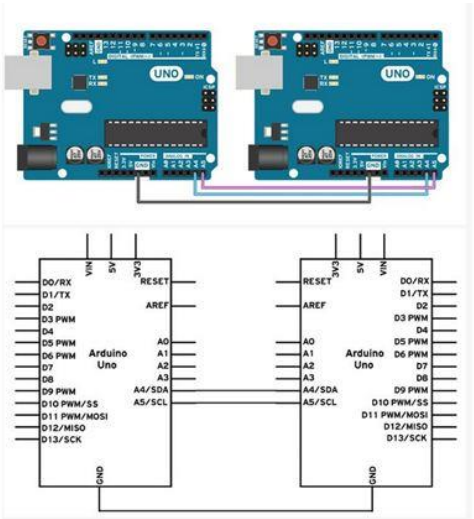


Fig.V.10. Conectarea a 2 plăci Arduino Uno (sursa: https://www.arduino.cc/en/Tutorial/MasterWriter)

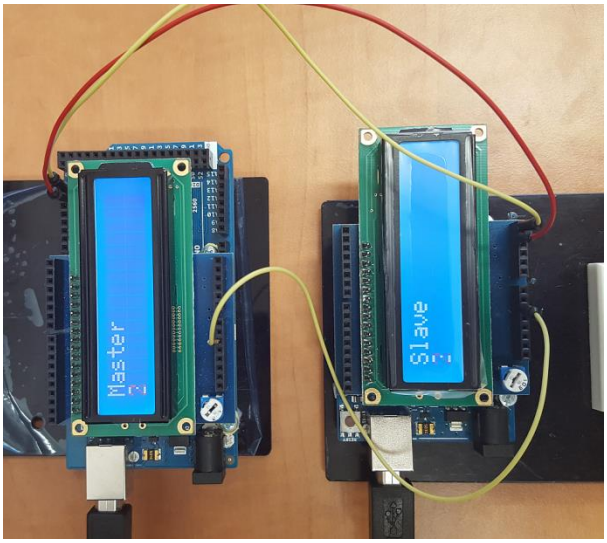


Fig.V.11. Conectarea unei plăci Arduino Mega cu o placă Arduino Uno

În cazul în care folosiți Arduino UNO/dumilanove/mini conectați pinii A4 și A5 ai unei plăci la exact aceiași pini la placa a doua. De asemenea, pinii de GND vor trebui legați împreună.

Nu legați și tensiunile împreună 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).

În tabelul de mai jos avem locația pinilor I2C pe diferite plăci 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

Fig.V.12. Localizarea semnalelor I2C la diferite plăci Arduino

Codul pentru dispozitivul *slave*:

```
#include <LiquidCrystal.h>

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

int x = 0;
```

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

void setup() {
    // Pornim busul I2C ca și slave la adresa 9
    Wire.begin(9);
    // Atașăm o funcție care să se declanșeze atunci când primim
    //ceva
    Wire.onReceive(receiveEvent);

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

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

void loop() {
    lcd.setCursor(0,1); // afișare caracter recepționat
    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);  // afișare caracter transmis pe lcd
    //master
    lcd.print(x);

    x++; // incrementăm x
    if (x > 5) x = 0; // resetăm x odată ce ajunge la 6
```

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

5.1. Activități practice

1. Testați exemplele din laborator. Întrebați cadrul didactic pentru orice nelămurire aveți legată de modul de conexiune a firelor/componentelor și de transmitere a datelor.

2. Realizați un sistem de comunicație între două PC-uri folosind plăci Arduino. Plăcile vor fi conectate la PC prin USB și între 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. Realizaţi o conexiune ca la exemplul 2, dar bi-direcţională. Pentru transmisiunea de la *slave* la *master* studiaţi https://www.arduino.cc/en/Tutorial/MasterReader .

4. Realizaţi o reţea cu un *master* şi doi *slave*. *Master*-ul va fi conectat la PC şi va primi prin interfaţa serială mesaje de genul

   a. s1-hello
   b. s2-goodbye

În funcţie 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.

VI. Laborator 6 - Folosirea limbajului de asamblare AVR

6.1. Limbajul de asamblare și limbajul C

De ce să folosim limbajul de asamblare?

Codul scris în limbaj de asamblare se traduce direct în instrucțiuni AVR care vor fi executate de microcontroller, fără cod suplimentar adăugat de compilator sau de mediu. Deși compilatoarele C/C++ performante pot uneori să producă un cod mașină foarte eficient din punctul de vedere al vitezei sau al necesarului de memorie, codul în limbaj de asamblare dă programatorului controlul absolut asupra codului binar rezultat. De cele mai multe ori codul în asamblare este mai mic, mai rapid și mai previzibil din punctul de vedere al necesarului de memorie și de timp, astfel încât este mai ușor de depanat.

Directive asamblor

Directivele pentru asamblor nu sunt parte a limbajului de asamblare (acesta este format din mnemonicile pentru instrucțiunile care sunt executate direct de către microprocesorul țintă), dar ele dau indicații compilatorului limbajului de asamblare (asamblorului) în procesul de generare a codului mașină.
Exemple de utilizare:

- Ajustarea unei locații în memoria program
- Definirea de macro-uri
- Inițializarea memoriei

Exemple:

.byte	Rezervă un octet pentru o variabilă
.comm	Declară un simbol comun
.data	Specifică începutul secțiunii de date a unui program
.ifdef	Condiționează includerea codului ce urmează în program dacă condiția este satisfăcută
.else	Condiționează includerea codului ce urmează în program dacă condiția nu este satisfăcută
.include	Include fișiere adiționale
#include	Include fișiere adiționale
.file	Începutul unui fișier logic
.text	Compilează ce urmează după această directivă, definește secțiunea de cod
.global	Definește sau declară o variabilă globală, sau o subrutină. De multe ori este folosită împreună cu variabilele specificate prin .comm, dacă variabila este folosită în mai multe fișiere
.extern	Tratează simbolurile nedefinite ca externe
.space	Alocă spațiu în memorie (pentru un șir)
.equ	Definește constante folosite în program
.set	Definește variabile locale folosite în program

Fig.VI.1. Exemplu de directive asamblor

Noțiuni de bază despre funcții

Dacă dorim să combinăm limbajul C/C++ cu limbajul de asamblare, putem să facem acest lucru folosind funcții (proceduri). Funcțiile trebuie declarate (prototipul funcției) și descrise (corpul funcției). Funcțiile pot să aibă sau să nu aibă intrări și ieșiri. În general, dacă o variabilă este pasată ca argument unei funcții, această variabilă trebuie să rămână neschimbată în timpul execuției funcției. Variabilele globale pot să fie schimbate în interiorul unei funcții.

Convențiile de apelare determină unde sunt stocate argumentele și valorile returnate de funcție. Când o funcție este apelată, trebuie să știe unde să caute argumentele și unde să stocheze valoarea returnată.

Apelul unei funcții prin CALL pune adresa de retur pe stivă și argumentele și valorile returnate sunt pasate conform convenției de apel (în registre, pe stivă, etc). Dacă funcția folosește variabile globale, acestea trebuie declarate ca atare și inițializate cu valori corecte.

După ce funcția este executată, se execută instrucțiunea RET, care va scoate adresa de retur de pe stivă.

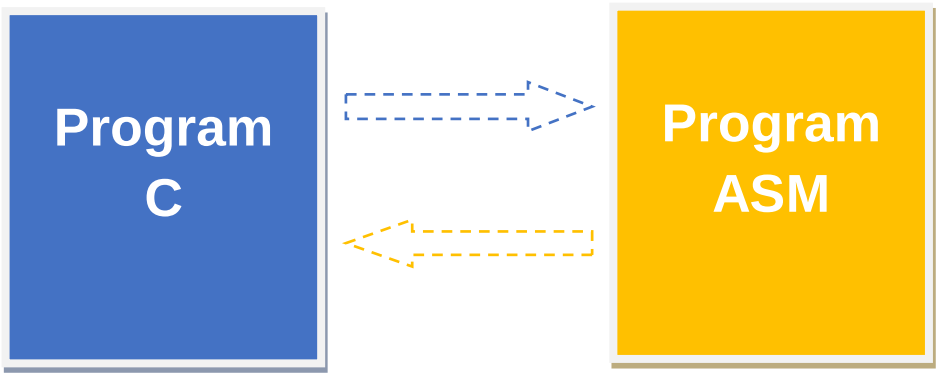


Fig.VI.2. Argumentele sunt transmise folosind convenția GCC.
Funcțiile C pot returna doar o singură valoare.

Funcțiile și stiva

Când o funcție este apelată, un context de activare este memorat (push) pe stivă. La revenirea din stivă, acest context este scos (pop) de pe stivă. Contextul de activare este format din toate datele necesare a fi memorate pentru a putea să continuăm programul la revenirea din funcție. De obicei contextul este format din variabile locale (de obicei stocate în registre) și adresa de revenire.

Stiva este o zonă de memorie SRAM adresată de pointerul de stivă SP, de 16 biți. La microcontrollerele AVR, pointerul de stivă este decrementat atunci când datele sunt memorate pe stivă, astfel că el trebuie inițializat cu cea mai mare adresă disponibilă (la AtMega2560 aceasta este 0x21FF).

Folosirea variabilelor globale

Variabilele globale sunt accesibile oriunde în cod, indiferent că este vorba de secțiuni de cod C sau de asamblare.

Variabilele globale pot fi declarate în fișierul .ino, dar este recomandat ca ele să fie declarate într-un fișier header. Variabilele globale trebuie să aibă echivalente în fișierul cu cod de asamblare, .S (declarate cu directivele asamblor .comm și .global).

Exemplu de declarare a variabilelor globale în fișierul .ino sau în header:

```
extern "C" int8_t var8b;
extern "C" int16_t var16b;
extern "C" uint32_t var32b;
```

Declararea acestor variabile în codul de asamblare (fișierul .S):

```
.data
.comm var8b, 1
.global var8b
.comm var16b, 2
.global var16b
.comm var32b, 4
.global var32b
```

În limbajul C (Arduino) o variabilă globală poate fi folosită ca atare, dar în limbajul de asamblare ea trebuie accesată la nivel de octet.

Exemplu:
Arduino:

```
void setup()
{
    longvar = 0xAABBCCDD;
    func1();
}
```

Cod asamblare:

```
.align 2
.comm longvar, 4
.global longvar
.text
.global func1 ;
func1:
lds r18, longvar
lds r19, longvar+1
lds r20, longvar+2
lds r21, longvar+3
...
ret
```

Când se folosesc variabile pe mai mult de un octet, compilatorul C/C++ de obicei se așteaptă ca ele să fie aliniate în memorie. De exemplu, o variabilă pe doi octeți trebuie să înceapă de la o adresă multiplu de 2, iar o variabilă pe patru octeți (precum longvar declarată mai sus) trebuie să înceapă de la o adresă multiplu de 4. Directiva asamblor .align, cu argumentul o putere a lui 2 (2^2 = 4 în exemplul de mai sus), asigură această aliniere.

Convenția de apelare a parametrilor

Instrucțiunile call (două cuvinte, salt la distanță) și rcall (un cuvânt, salt scurt) schimbă valoarea registrului PC pentru a conține adresa apelată. O instrucțiune de apel va salva adresa de retur pe stivă. Instrucțiunea ret scoate adresa de retur de pe stivă și o va pune în PC. Parametrii funcției se vor stoca în registrele r25 ... r8, primul octet fiind stocat în r24. Dacă funcția are mai mulți parametri, care nu mai încap în aceste registre, parametrii sunt plasați pe stivă înainte de execuția apelului. Codul din interiorul funcției va citi parametrii de pe stivă, iar codul apelant va trebui să elimine de pe stivă acești parametri după ce procedura este executată.

Prototipul funcției

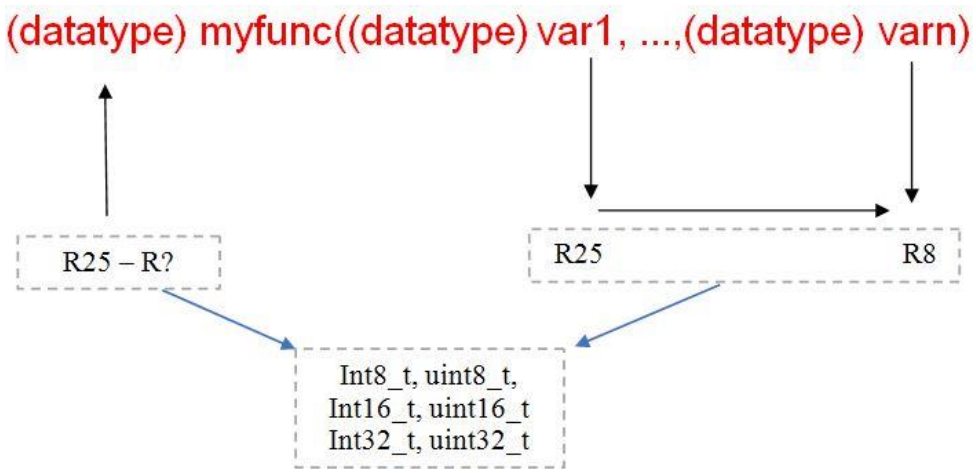


Fig.VI.3. Prototipul funcției – transmiterea parametrilor și returnarea rezultatului folosind registre

Pentru a accesa parametrii din cadrul stivei (contextul de activare, contextul local), trebuie să copiați pointerul de stivă SP în registrul pointer Y:

```
in r28, SPL
in r29, SPH
```

Dacă avem o funcție cu 11 parametri de un octet fiecare, primii 9 vor fi stocați în registrele cu număr par de la r24 până la r8, iar ultimii doi vor fi salvați pe stivă. Dacă folosim pointerul Y pentru accesarea acestor valori, trebuie să îl salvăm și pe el pe stivă, astfel că începutul funcției va arăta astfel:

```
push r28
push r29
in r28, SPL
in r29, SPH
```

Parametrii 10 și 11 se vor accesa astfel:

```
ldd r7, Y+5
ldd r7, Y+6
```

Folosirea registrelor în interiorul funcțiilor

Cele 32 de registre ale microcontrollerului AVR au roluri diferite, alocate de compilatorul gcc. Dacă proiectul vostru conține doar cod asm pur, puteți folosi registrele cum doriți. Dacă combinați codul asm cu cod C, trebuie să respectați regulile din tabelul următor:

0x00	R0	Registru "liber", conținutul poate fi modificat oricând fără a fi necesară refacerea.
0x01	R1	Trebuie să conțină întotdeauna valoarea 0, nu schimbați.
0x02	R2	Trebuie lăsate neschimbate de o funcție, sau salvate și restaurate înainte de revenirea din funcție.
…	…	
0x0D	R13	
0x0E	R14	
0x0F	R15	
0x10	R16	
0x11	R17	
0x12	R18	R18 ... R27 sunt disponibile pentru utilizarea liberă în funcții. Schimbarea valorii lor în funcții este așteptată.
…	…	
0x1A(XL)	R26	Pointerul X se poate folosi liber, nu trebuie salvat.
0x1B(XH)	R27	
0x1C(YL)	R28	Pointerul de cadru local Y poate fi utilizat în interiorul funcțiilor, dar trebuie salvat și restaurat înainte de retur.
0x1D(YH)	R29	
0x1E(ZL)	R30	Pointerul Z se poate folosi liber, nu trebuie salvat.
0x1F(ZH)	R31	

Fig.VI.4. Folosirea registrelor în interiorul funcțiilor

Valori returnate

Valorile returnate ale funcțiilor sunt stocate în registrele r25-r28, în funcție de dimensiunea valorii returnate (dimensiunea maximă este de 8 octeți). Dacă valoarea returnată este de 1 octet, ea se va plasa în r24, r25 rămânând 0, pentru valori pozitive, sau 255, pentru valori negative.

Ieșire declarată	**Locația valorii returnate**
Byte, Boolean, int8_t, uint8_t	r24 (r25 = 00,FF)
int, uint, short, char, unsigned char int16_t, uint16_t	r25:r24
long, ulong, int32_t, uint23_t	r25:r22

Fig.VI.5. Tipul valorilor returnate de funcții și registrele utilizate de acestea

Unelte de dezvoltare

Pentru dezvoltarea de cod în limbaj de asamblare, care poate fi rulat pe plăcile Arduino Mega, avem următoarele opțiuni:

1. Folosirea Arduino IDE

   a. Asamblare "Inline"– mici părți de cod în limbaj de asamblare inserat în interiorul codului C++
   b. Fișiere sursă în limbaj de asamblare, conținând funcții apelate din fișierul .ino principal

2. Folosirea IDE-ului Atmel Studio

În această lucrare de laborator vom folosi variantele 1.b și 2.

6.2. Folosirea Arduino IDE

a. Un simplu blink

Combinarea codului în limbaj de asamblare cu codul C folosind Ardino IDE nu pune probleme deosebite. În primul nostru exemplu, vom replica funcționalitatea primului nostru program Arduno, "Blink", folosind funcții scrise în limbaj de asamblare pentru manipularea biților porturilor.

Deschideți mediul de dezvoltare Arduino și copiați următorul cod:

```
extern "C" void setpin();
extern "C" void turnon();
extern "C" void turnoff();

void setup() {
    setpin();
}

void loop() {
    turnon();
    delay(1000);
    turnsoff(0);
    delay(1000);
}
```

În exemplul de mai sus, funcțiile setpin (care va configura pinul 13 ca ieșire), turnon (care va aprinde LEDul) și turnoff (care va stinge LEDul) vor fi implementate în limbaj de asamblare.

Pentru a include codul în limbaj de asamblare, creați un fișier .S prin apăsarea butonului cu pictograma săgeată aflat în zona dreapta sus a ferestrei IDE, numiți-l cum doriți, dar nu uitați să adăugați extensia ".S". Folosiți „S" mare, nu „s" mic, pentru că în caz contrar, compilatorul nu va procesa fișierul.

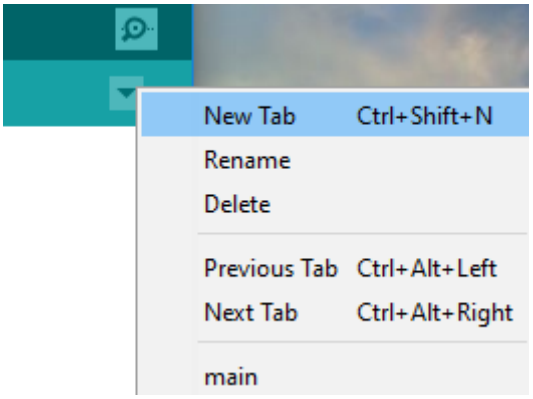


Fig.VI.6. Crearea unui fișier nou

De exemplu, putem numi acest fișier asm_functions.S. Copiați fragmentul de mai jos în fișierul nou creat. Fișierele .ino și .S trebuie să fie în același director.

```
#include "avr/io.h"

.global setpin

setpin:
    sbi _SFR_IO_ADDR(DDRB), 7 ; setează bitul 7 al pe 1 - ieșire
    ret

.global turnon
turnon:
    sbi _SFR_IO_ADDR(PORTB), 7  ; setează bitul 7 al PORTB pe 1
    ret

.global turnoff
turnoff:
    cbi _SFR_IO_ADDR(PORTB), 7  ; setează bitul 7 al PORTB pe 0
    ret
```

Codul de mai sus manipulează valoarea bitului 7 al portului B, care este conectat la pinul digital 13 al plăcii Arduino Mega (vedeți https://www.arduino.cc/en/Hacking/PinMapping2560 pentru alte corespondențe ai pinilor la porturi). Prima dată pinul trebuie configurat ca ieșire prin scrierea unui '1' pe poziția corespunzătoare din DDRB, apoi valoarea lui va fi configurată prin schimbarea valorii bitului din PORTB).

Atenție: compilatorul Arduino presupune că fiecare nume de port/simbol se referă la adresa din spațiul adreselor de memorie de date și nu la adresa din spațiul I/O.
De exemplu, pentru portul B avem două adrese: 0x05 în spațiul de adrese I/O și 0x25 în spațiul de adrese din memorie. Pentru a impune compilatorului folosirea spațiului de adrese I/O, folosiți macro-ul _SFR_IO_ADDR.

PORTB – Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
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	

Fig.VI.7. Adresarea portului B (sursa: datasheetul atmega2560)

Compilați programul și încărcați-l pe placă. Programul ar trebui să execute operația de clipire (blink).

b. *Folosirea unei funcții cu parametri*

În continuare, vom încerca să obținem același comportament ca în exemplul anterior, dar în loc de două funcții, una pentru aprindere și alta pentru stingere, vom utiliza o singură funcție și vom pasa starea LED-ului ca parametru.

Codul C++ va fi schimbat astfel:

```
extern "C" void setpin();
extern "C" char turnspecified(char c);

void setup() {
    setpin();
}

void loop() {
    turnspecified(1);
    delay(1000);
    turnspecified(0);
    delay(1000);
}
```

Iar codul de asamblare astfel:

```
#include "avr/io.h"

.global setpin

setpin:
    sbi _SFR_IO_ADDR(DDRB), 7 ; setează bitul 7 din DDRB to 1 –
    ;ieșire
    ret

.global turnspecified
turnspecified:
    tst r24 ; r24 este parametrul funcției, se testează pentru
    ;zero
```

```
    breq set0  ; daca e zero, salt la setarea pinului pe zero
    sbi _SFR_IO_ADDR(PORTB), 7  ; în caz contrar, se setează pe 1
    rjmp finish
set0:
    cbi _SFR_IO_ADDR(PORTB), 7     ; setare pe zero
finish:
    ret
```

Parametrii unei funcții sunt transmiși în registrele r25 ... r8. Un parametru pe 8 biți se transmite în r24, un parametru pe 16 biți în r25:r24, etc.

c. *Folosirea interfeței seriale în limbaj de asamblare*

Următorul exemplu va afișa un mesaj prin interfața serială (Serial). Mesajul va fi stocat în memoria program, ca un șir de caractere terminat cu 0.

Codul Arduino C++ este:

```
extern "C" void Serial_Setup();
extern "C" void Print_Hello();

void setup() {
    Serial_Setup();
}

void loop() {
    Print_Hello();
    delay(500);
}
```

Codul în limbaj de asamblare este:

```
#include "avr/io.h"
.global Serial_Setup
Serial_Setup:

    ; Configurare parametri pentru interfața serială 0
    clr r0
    sts   UCSR0A, r0
    ldi   r24, 1<<RXEN0 | 1 << TXEN0  ; activare Rx și Tx
    sts   UCSR0B, r24
    ldi   r24, 1 << UCSZ00 | 1 << UCSZ01 ; asincron, fără
    ;paritate, 1 bit de stop, 8 biți
    sts   UBRR0H, r0
    ldi   r24, 103
    sts   UBRR0L, r24
    ret

.global Print_Hello
```

```
Print_Hello:

    ; se încarcă adresa de început a șirului în pointerul Z
    ldi ZL, lo8(the_message)          ; r30
    ldi ZH, hi8(the_message)          ; r31
    lpm r18, Z+          ; Citește primul caracter din șir în r18

Loop:
    lds   r17, UCSR0A
    sbrs  r17, UDRE0 ; verifică dacă se pot transmite date (UDR e
    ;gol)
    rjmp  Loop
    sts   UDR0, r18       ; se transmit datele din r18
    lpm r18, Z+           ; se preia următorul caracter din memorie
    tst r18               ; verifică dacă e zero - final de șir
    brne Loop
    ret

the_message:         ; mesajul propriu zis, urmat de LF, CR și 0
    .ascii "Assembly is fun"
    .byte 10, 13,0
```

Prima funcție în limbaj de asambare configurează parametrii interfeței seriale UART0 (interfața Serial din Arduino), iar a doua funcție transmite un mesaj stocat în memoria program prin această interfață. Deschideți Serial Monitor pentru a vedea mesajul transmis.

Verificați în fișa tehnică a AVR ATMega2560 (http://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf) și în paginile cursului 6, pentru a înțelege setările și operațiile interfeței UART.

d. Folosirea șirurilor C în funcții în limbaj de asamblare

În acest exemplu vom scrie o funcție în limbaj de asamblare care va aduna elementele unui șir declarat în C. Funcția va fi apelată din codul Arduino C++. Creați următoarele fișiere, cu conținutul specificat mai jos:

Fișierul .ino

```
#include "external_functions.h"

void setup() {
    compute();
    uint8_t val = result;
    Serial.begin(9600);
    Serial.println(val);
}
void loop() { }
```

Fișierul external_functions.h

```
#include <stdint.h>
extern "C" uint8_t result;
extern "C" void compute(void);
extern "C" uint8_t myarray[10]={1, 30, 3, 4, 5, 6, 7, 8, 10, 11};
```

Fișierul arsum.S

```
.file "arsum.S"
.data
.comm result, 1
.global result

.text
.global compute

compute:
    ldi r30, lo8(myarray)
    ldi r31, hi8(myarray)
    ldi r18, 0
    ldi r21, 0

looptest:
    ld r22, z+
    add r21, r22
    inc r18
    cpi r18, 10
    brlo looptest

out:
    sts result, r21
    ret
```

6.3. Utilizarea Atmel Studio

Configurarea mediului Atmel Studio

Atmel Studio este o platformă integrată de dezvoltare (IDP) pentru scrierea și depanarea de aplicații pentru multiple microcontrollere (incluzând AVR). Mediul Atmel Studio 7 vă oferă posibilitatea de a dezvolta și depana aplicații scrise în C/C++ sau în limbaj de asamblare. De asemenea, pune împreună utilitare pentru programare și depanare pentru diferite dispozitive AVR.

Prima dată trebuie să configurăm mediul pentru a putea încărca codul scris cu Atmel Studio pe plăcile Arduino Mega.

1. Deschideți Atmel Studio 7
2. Deschideți meniul **Tools**
3. Selectați **External Tools**

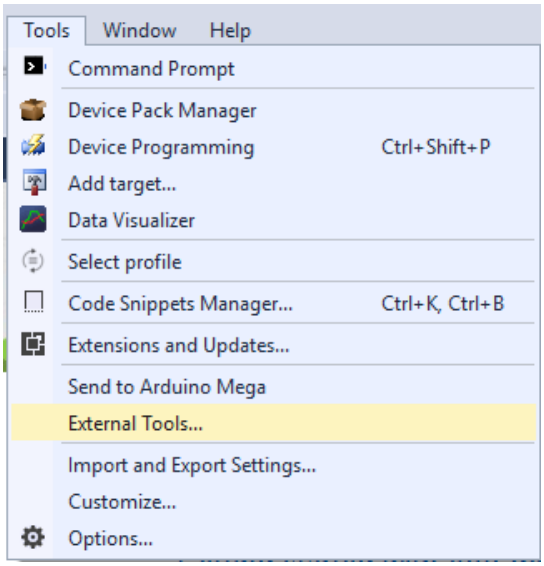


Fig.VI.8. Meniul external tools

4. Apăsați butonul Add din fereastra care se deschide

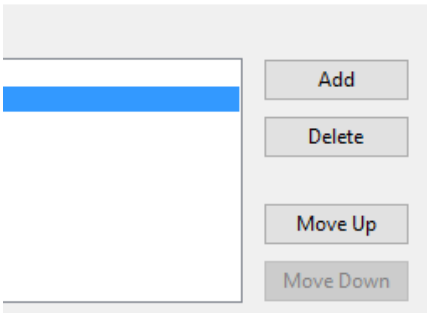


Fig.VI.9. Crearea unei noi unelte externe

5. Se va crea o nouă opțiune. Completați următoarele date în căsuțele de text corespunzătoare:

Title: Send to Arduino Mega
Command : C:\Program Files (x86)\Arduino\hardware\tools\avr\bin\avrdude.exe
Arguments: -v -C"C:\Program Files (x86)\Arduino\hardware\tools\avr\etc\avrdude.conf" -p atmega2560 -c wiring -P COM5 -b 115200 -D -U flash:w:$(TargetDir)$(TargetName).hex:i

Observați că acest proces presupune că uneltele Arduino sunt instalate în directorul C:\Program Files (x86). Dacă ați instalat Arduino în altă parte, modificați cu calea corectă. Dacă ați instalat Arduino ca o aplicație Windows 10, dezinstalați-l și instalați-l normal, deoarece în caz contrar nu îl veți putea folosi cu Atmel Studio.

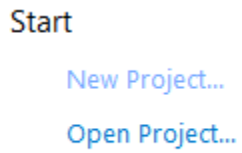
În șirul de argumente, portul COM este scris ca o constantă. Verificați portul plăcii Arduino Mega atașate calculatorului dvs și înlocuiți textul marcat cu roșu cu numele corect al portului.

Bifați opțiunea "Use Output Window", apăsați Apply și apoi Ok. După ce ați parcurs acești pași, opțiunea "Send to Arduino Mega" va fi disponibilă în meniul Tools.

Crearea unui proiect dintr-o schiță Arduino

Atmel Studio ne permite să transformăm un proiect Ardino (sketch) într-un proiect Atmel Studio. Trebuie parcurși următorii pași:

1. În Atmel Studio, apăsați pe new project, din meniul File.



2. Alegeți să creați un proiect dintr-o schiță Arduino și completați numele, locația noului proiect și numele soluției.

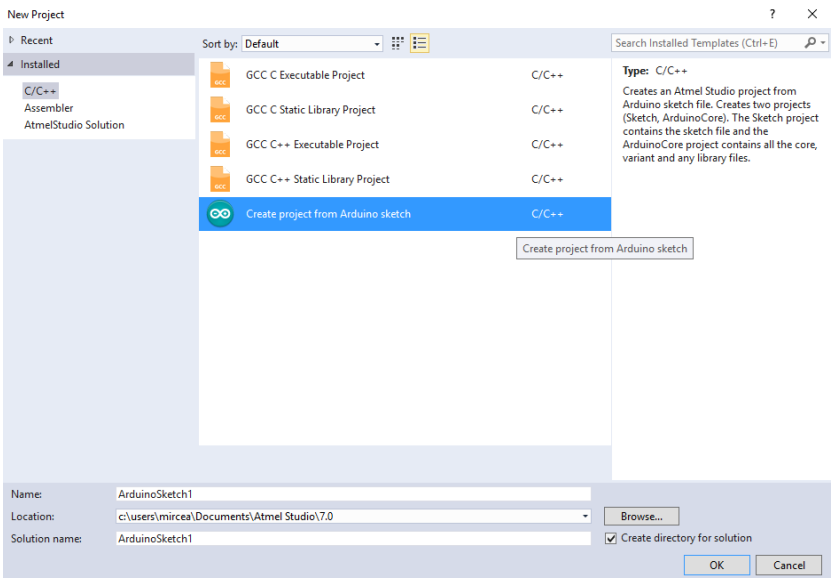


Fig.VI.10. Crearea unui proiect dintr-o schiță Arduino

3. În fereastra nou deschisă, indicați calea spre schița Arduino existentă, selectați calea spre mediul Arduino, selectați placa și microcontrollerul și apăsați OK.

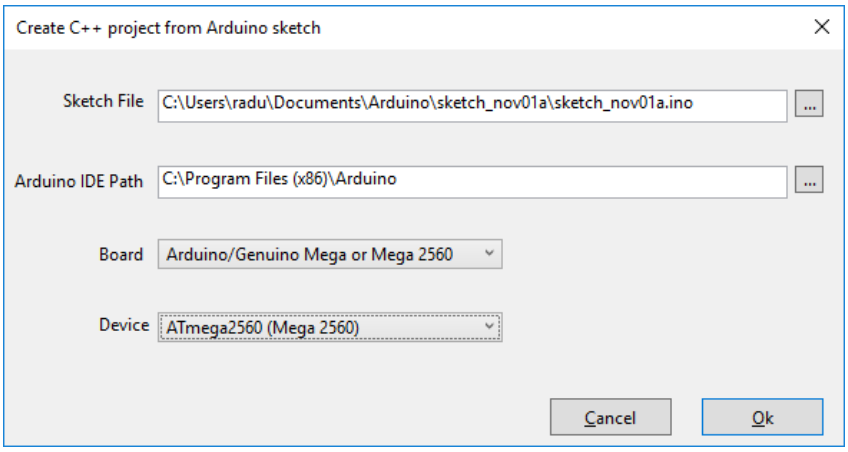


Fig.VI.11. Configurarea noului proiect

În acest exemplu vom utiliza al doilea exemplu de tip blink, cu cod în asamblare și în C++, care folosește o funcție în limbaj de asamblare pentru a seta starea LED-ului de la pinul 13.

După importarea schiței, arborele soluției arată ca în imaginea de mai jos:

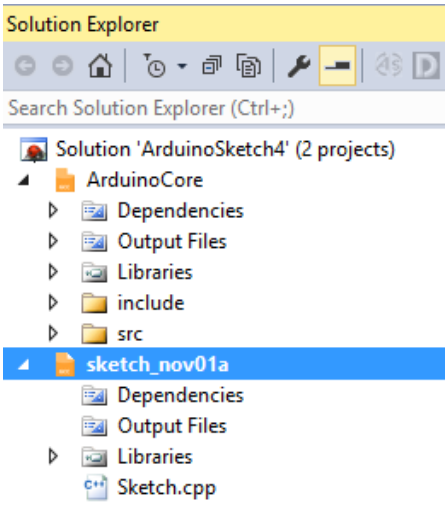


Fig.VI.12. Arborele soluției pentru noul proiect

Dacă vom compila programul, observăm că vom obține mai multe erori:

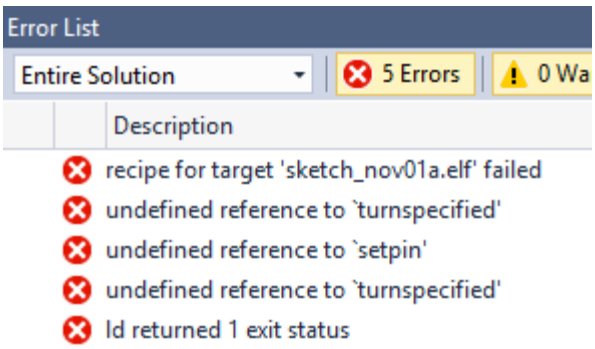


Fig.VI.13. Erori de compilare

Acest lucru se întâmplă deoarece Atmel Studio nu adaugă în mod automat referința către o bibliotecă externă. Pentru a rezolva această problemă, dați click dreapta pe numele proiectului și selectați Add Existing Item din meniu.

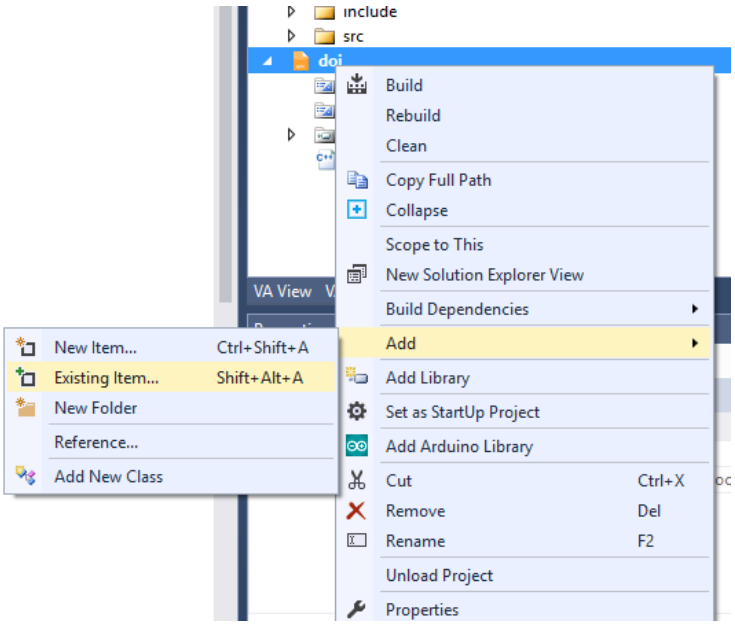


Fig.VI.14. Adăugarea dependențelor lipsă

Indicați locația fișierului în limbaj de asamblare asm_functions.S, din directorul schiței Arduino și adăugați acest fișier la soluție.

Re-compilați soluția și de data aceasta ar trebui să obțineți o compilare reușită.

Pentru a încărca programul pe placă, deschideți meniul „Tools” și selectați „Send to Arduino Mega”. Dacă ați parcurs pașii anteriori în mod corect, programul ar trebui să se încarce fără probleme. Veți vedea mesajul de mai jos, dacă programul este încărcat corect:

```
Reading | ################################################## | 100% 0.13s

avrdude.exe: verifying ...
avrdude.exe: 882 bytes of flash verified

avrdude.exe: safemode: lfuse reads as FF
avrdude.exe: safemode: hfuse reads as D0
avrdude.exe: safemode: efuse reads as FF
avrdude.exe: safemode: Fuses OK (E:FF, H:D0, L:FF)

avrdude.exe done.  Thank you.
```

Fig.VI.15. Compilare finalizată cu succes

Depanarea folosind Atmel Studio

O caracteristică puternică a Atmel Studio este depanarea bazată pe simulare. Acest mecanism de depanare ne permite să analizăm comportamentul programului, să observăm starea registrelor, a porturilor și a memoriei, chiar dacă nu avem la dispoziție o placă.

Pentru a configura mediul pentru depanare, selectați opțiunea **Debug** din caseta de tip roll down, și ATmega2560 din bara de unelte, ca în figura de mai jos:

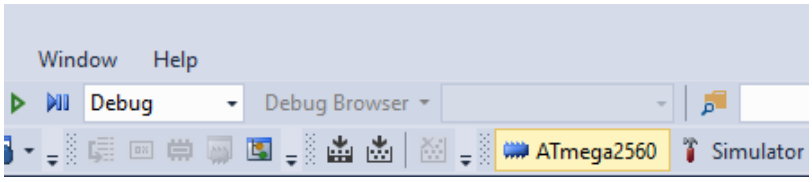


Fig.VI.16. Configurarea dispozitivului țintă pentru depanare

La apăsarea butonului cu opțiunea ATmega2560 se va deschide o nouă fereastră. Alegeți opțiunea **Tool** și de aici selectați pentru **Select debugger / programmer** varianta **Simulator**.

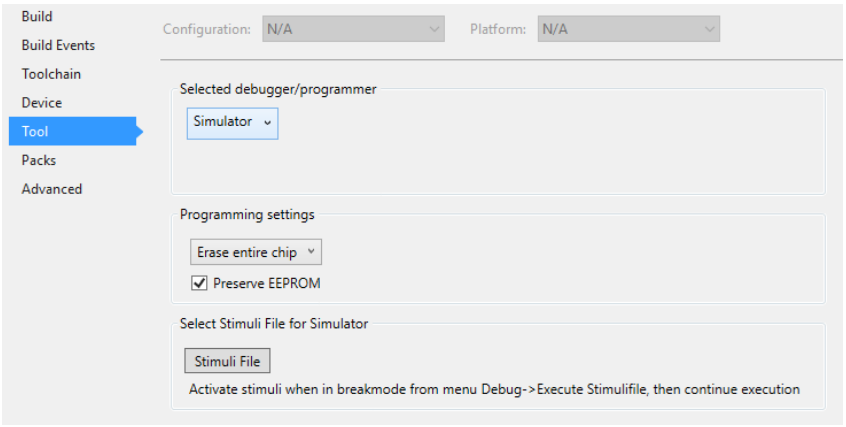


Fig.VI.17. Configurarea opțiunilor de depanare

După configurarea opțiunilor, reveniți în fereastra principală a programului cu tab-ul, selectând schița programului. Putem folosi depanatorul pentru a analiza comportamentul programului blink pe care l-am importat deja din Arduino. Pentru depanare, se recomandă să comentați apelurile de funcție delay, pentru că ele iau foarte mult timp în simulare.

Puteți configura puncte de oprire (breakpoint) prin apăsarea cu mouse-ul a zonei gri din stânga liniei de cod unde doriți oprirea, sau din meniul Debug selectând opțiunea **Toggle Breakpoint** (sau apăsând F9). Puteți pune breakpoint în fișierul c++ sau în fișierul cu cod în asamblare.

Pentru pornirea depanării, apăsați butonul  de lângă meniul de selecție a configurațiilor, sau apăsați Alt+F5, și apoi apăsați o dată F5. Pentru a observa registrele AVR de intrare/ieșire (porturi), din meniul **Debug** selectați Windows, apoi **I/O**. Din același meniu **Debug/Windows** puteți selecta să vizualizați alte informații precum Registers (afișează conținutul celor 32 de registre), Memory (conținutul memoriei program, a memoriei de date, a EEPROM), Disassembly, etc.

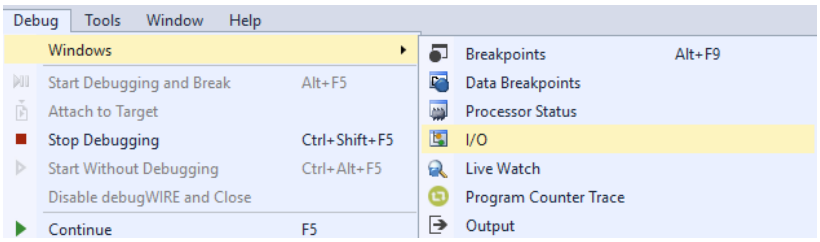


Fig.VI.18. Selectarea ferestrei de afișare a informațiilor despre I/O

Din fereastra I/O, selectați PORTB. Veți vedea, la fiecare pas al programului, conținutul registrelor asociate, DDRB, PORTB, și pinii de intrare PINB.

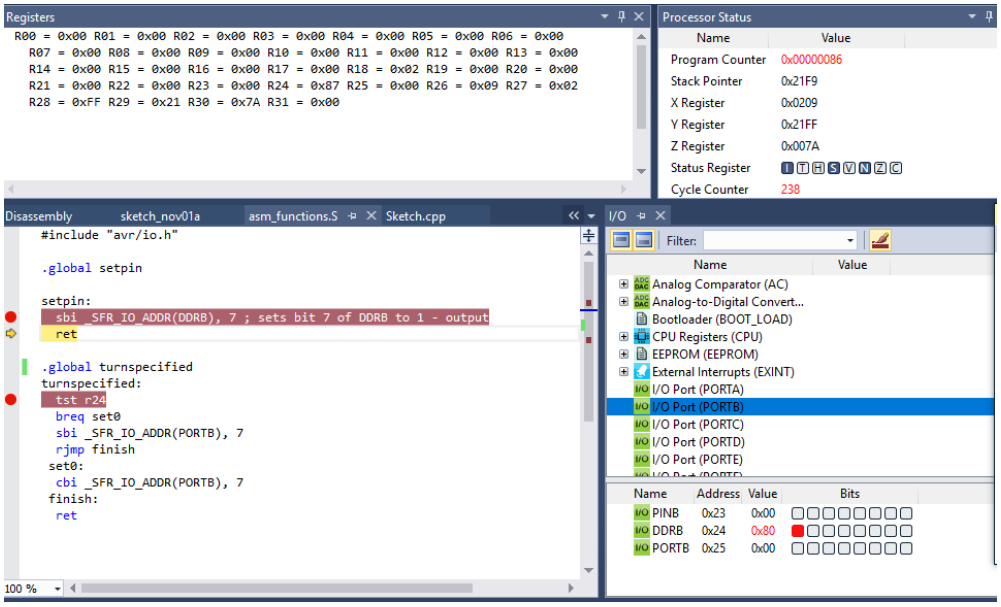


Fig.VI.19. Afișarea stării portului B în fereastra I/O

Crearea de proiecte folosind doar limbajul de asamblare

Atmel Studio vă permite să scrieți programe în care să utilizați doar limbajul de asamblare. Din meniul **File**, selectați **New Project**. La deschiderea ferestrei New Project, selectați din panoul din stânga opțiunea "**Assembler**", ca în figura de mai jos. Dați un nume proiectului dvs și apăsați OK.

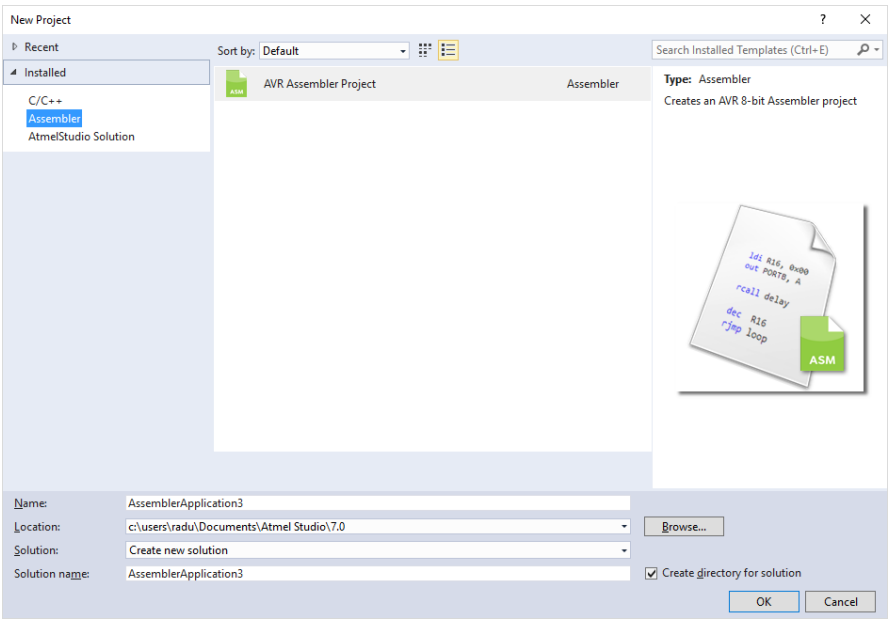


Fig.VI.20. Crearea unui proiect nou, în limbaj de asamblare

Mediul va genera un fișier main.asm, cu un cod machetă asm. Înlocuiți acest cod cu cel de mai jos, care va transmite mesajul “Assembly is fun” pe interfață.

```
; Program principal
main:
    rcall asm_setup

main_loop:
    rcall asm_loop
    rjmp main_loop


asm_setup:
; Inițializare interfață serială
    clr r0
    sts   UCSR0A, r0
    ldi   r24, 1<<RXEN0 | 1 << TXEN0    ; activare Rx & Tx
    sts   UCSR0B, r24
    ldi   r24, 1 << UCSZ00 | 1 << UCSZ01 ; asincron, fără
    paritate, 1 bit stop, 8 biți
    sts   UBRR0H, r0
    ldi   r24, 103
    sts   UBRR0L, r24
    ret

asm_loop:
    ; tipărește mesaj și așteaptă
    rcall Print_Hello
    rcall wait
    ret


Print_Hello:

    ; Se încarcă adresa de început a șirului
    ldi ZL, LOW(2*array)          ; r30
    ldi ZH, HIGH(2*array)         ; r31
    lpm r16, Z+                 ; Citire caracter indicat de
    ;pointerul Z din memoria program

Loop:
    lds   r17, UCSR0A
    sbrs  r17, UDRE0 ; verifică dacă bufferul e gol pt transmisie
    rjmp  Loop
    sts   UDR0, r16           ; trimite datele din r16
    lpm r16, Z+               ; următorul caracter
    tst r16              ; verificare final de șir - 0
    brne Loop
    ret
```

```
; funcție simplă pentru așteptare aproximativ 1 secundă, prin
;numărare
wait:

    ldi  R17, 0x53
    LOOP0:  ldi  R18, 0xFB
    LOOP1:  ldi  R19, 0xFF
    LOOP2:  dec  R19
    brne LOOP2
    dec  R18
    brne LOOP1
    dec  R17
    brne LOOP0
    ret

; șirul de transmis, stocat în memoria program
array:
.db "Assembly is fun",13,10,0
```

Compilați soluția folosind meniul **Build** și încărcați pe Arduino Mega. Pentru monitorizarea mesajelor transmise pe interfața serială, puteți folosi Serial Monitor de la Arduino, sau puteți utiliza terminalul Atmel Studio. Pentru acest lucru, selectați din meniul **Tools** opțiunea **Data Visualizer**. Din panoul din stânga, alegeți **Visualization/Terminal**, iar din panoul central selectați portul serial al plăcii și apăsați Connect. Terminalul ar trebui să se deschidă și să afișeze mesajul din figura de mai jos:

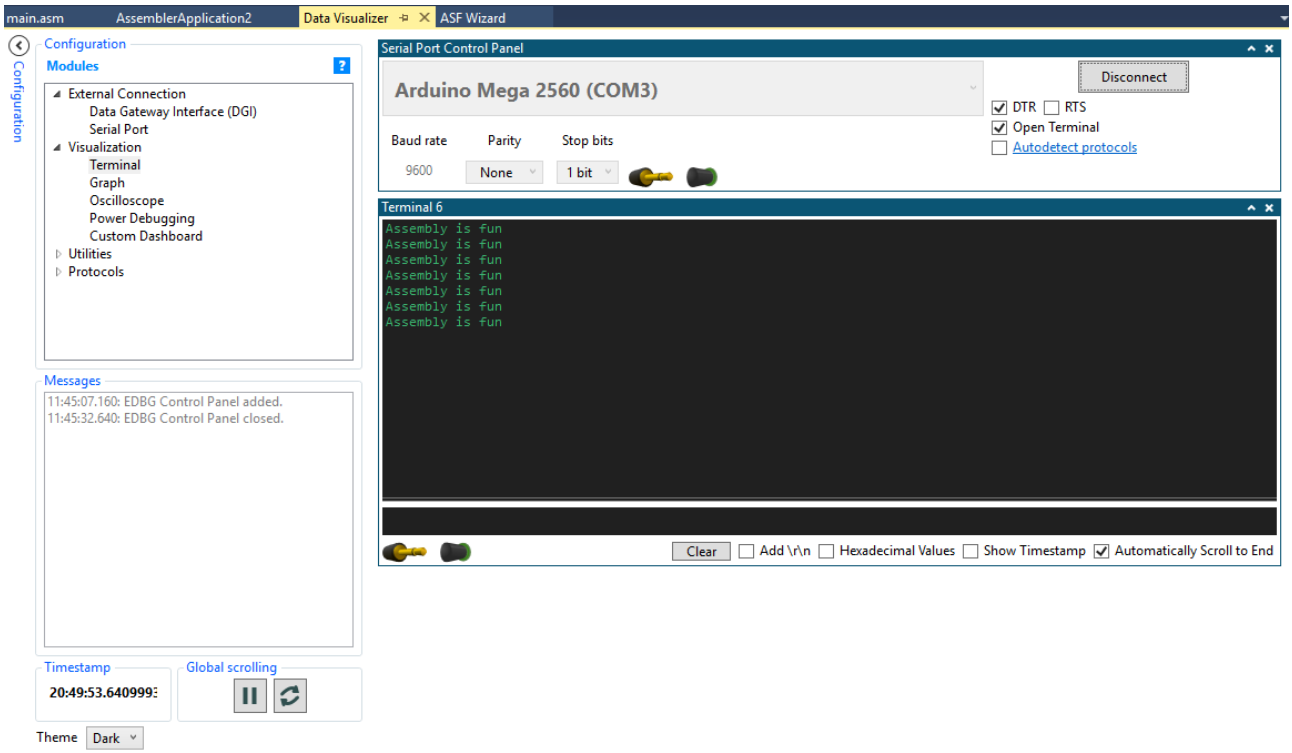


Fig.VI.21. Terminalul Atmel Studio

Puteți folosi depanatorul (simulatorul) pentru a analiza pas cu pas execuția programelor în limbaj de asamblare. Pașii care trebuie efectuați sunt aceiași ca în cazul proiectelor C/C++.

6.4. Lucru individual:

1. Implementați exemplele din această lucrare de laborator. Folosiți depanatorul cât mai des, pentru a analiza comportamentul programului. Puteți vedea mesajul în memoria program flash?
2. Folosind fișa tehnică și informațiile din cursul 6, scrieți un document care explică setările și funcționarea interfeței seriale, așa cum este ea folosită în exemplul al treilea.
3. Scrieți o funcție în asamblare care va returna o valoare (valorile returnate se transmit începând cu registrele r25:r24). Scrieți programul .ino care apelează această funcție și folosește valoarea returnată. *Indiciu: puteți citi un port.*
4. Modificați exemplul care scrie mesajul "Arduino is fun" pentru a afișa orice mesaj (șir de caractere, terminat cu zero) declarat în programul C++. *Indiciu: șirul de caractere este în memoria de date.*
5. Analizați codul scris în asamblare pentru comunicarea prin interfața serială din proiectul Arduino, și comparați-l cu codul din proiectul ce folosește doar limbaj de asamblare. Descrieți diferențele și asemănările.

Referințe:

1. https://docslide.us/documents/lecture-12-5600350816ac8.html
2. https://forum.arduino.cc/index.php?topic=490065.0
3. https://www.youtube.com/watch?v=8yAOTUY9t10

VII. Laborator 7 – Procesarea semnalelor analogice

Un semnal analogic este o tensiune variabilă în timp, de obicei ieşirea unui senzor care monitorizează mediul înconjurător. Un astfel de semnal poate fi procesat şi interpretat de un microcontroller care foloseşte un convertor analog-digital (ADC), care este un dispozitiv ce converteşte tensiunea într-un număr care poate fi „înţeles" de microcontroller.

Pinii care pot fi folosiți împreună cu senzori analogici sunt pinii care au un ‚A' în faţa numelui lor pe placă. În Figura VII.1 se observă pinii analogici de pe Arduino Mega încercuiți cu roşu, iar în Figura VII.2 se poate observa locația pinilor analogici de pe Arduino UNO.

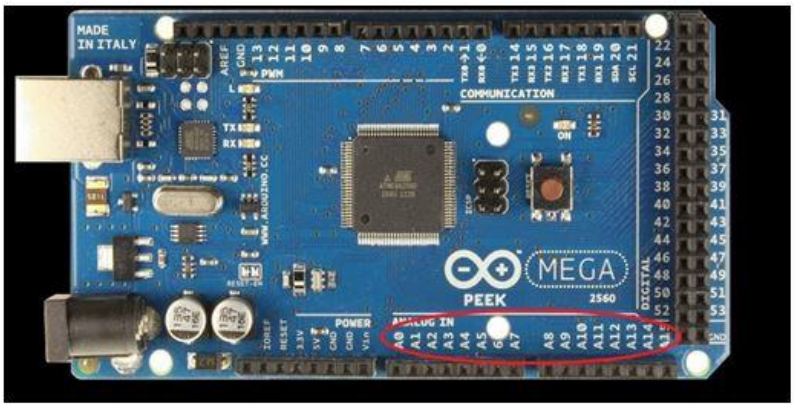


Fig.VII.1. Pinii Analogici din Arduino Mega. (source: https://store.arduino.cc/arduino-mega-2560-rev3)

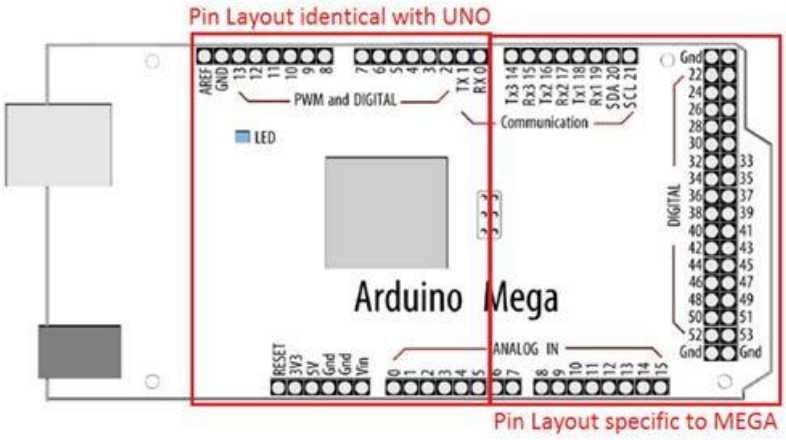


Fig.VII.2. Arduino UNO şi Mega pin layout

ADC-urile pot varia mult între diferite tipuri de microcontrollere. Spre exemplu, pe Arduino Mega avem ADC-uri care au o precizie de 10 biţi. Acest lucru înseamnă că aceste convertoare pot detecta până la 1024 valori. Există şi adc-uri care au rezoluţie de 8 sau 16 biţi. ADC-ul întoarce o valoare fracţionară. Asta înseamnă ca ADC-ul consideră 5V ca 1023 şi orice valoare mai mică decât 5V va fi construită o fracție intre 5V si 1023 (2).

$$\frac{5}{1023} = \frac{\textit{Citirea de la ADC}}{\textit{Valoare tensiune masurata}} \qquad (2)$$

Pinii analogici de pe Arduino pot fi folosiţi şi ca pini de tip I/O general (GPIO), fiind echipaţi inclusiv cu rezistenţe PULL-UP, ei având aceleaşi funcţionalităţi ca şi pinii digitali, în cazul în care cei oferiţi de placă nu sunt suficienţi. Sintaxa arduino pentru activarea acestor rezistori este similară cu cea de la pinii digitali: digitalWrite(A0, HIGH);//pinul A0 fiind setat ca input. Pentru citirea unei valori de la un senzor se foloseşte comanda analogRead().

!!Atenţie
Comanda analogRead() nu va funcţiona corect dacă pinul de pe care încercați să citiţi a fost setat ca pin de ieşire.

În momentul în care realizăm o citire, dacă executăm rapid o comutare între pozițiile pe care dorim să le citim, se vor introduce zgomote în citirea semnalului. Se recomandă folosirea unui mic delay() înaintea citirii unei valori analogice de pe alt pin.

O altă funcţie importantă legată de utilizarea senzorilor analogici este „analogReference()”. Pentru a măsura o tensiune analogică trebuie să existe o tensiune de referinţă la care să o raportam. Funcţia „analogReference()” setează tensiunea maximă cu care să efectuăm măsurătoarea.

Configuraţii posibile pentru această referinţă sunt :

- **DEFAULT** – foloseşte tensiunea de referinţă a plăcii (5V pentru plăcile Arduino care folosesc tensiune de 5V, UNO and MEGA)

- INTERNAL – setează o tensiune de referinţă de 1.1 V. Poate fi folosită pe plăcile care conţin ATMEGA 328 (UNO), dar nu poate fi folosită pe ATMEGA2560

- INTERNAL1V1 – tensiune de referinţă de 1.1 V folosită pe plăcile MEGA

- INTERNAL2V56 – tensiune de referinţă de 2.56 V, valabilă doar pe plăcile MEGA

- EXTERNAL – tensiune aplicată pinului AREF. Această tensiune este între 0 şi 5V

Note:
1. După schimbarea tensiunii de referinţă, primele citiri cu analogRead() pot fi eronate!!!
2. Nu folosiţi o tensiune de referinţă externă negativă (<0V) sau mai mare de 5V pe pinul AREF! Dacă folosiţi o tensiune externă de referinţă, configuraţi referinţa ca externă apelând analogReference() înainte de a apela funcţia analogRead(), în caz contrar putând cauza un scurtcircuit prin conectarea referinţei interne cu cea externă !!!

Exemplul 1: citirea unui potenţiometru

În exemplul următor vom citi valoarea de la un potențiometru liniar. Valoarea citită va fi afișată pe LCD. Circuitul pentru acest exemplu este ilustrat în Figura VII.3 (legaţi pinii VCC şi GND ai potenţiometrului la +5V şi GND de pe placă şi semnalul de ieşire la un pin analogic – A1).

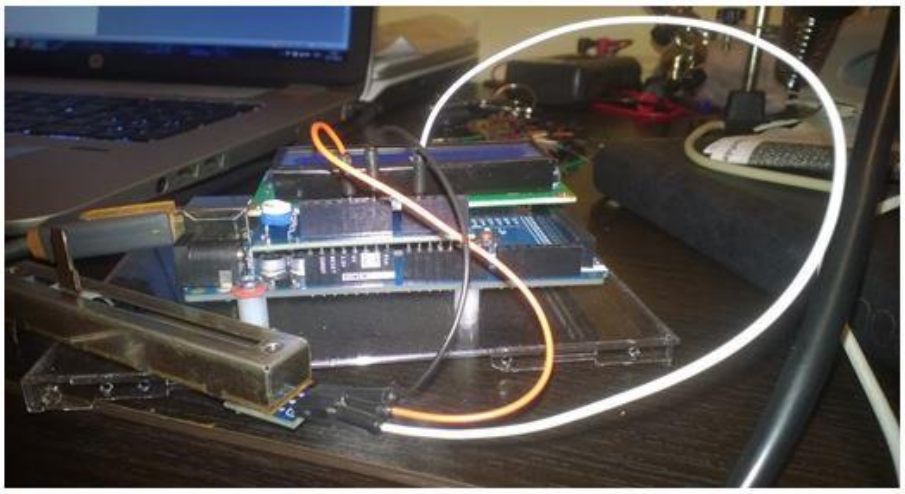


Fig.VII.3. Conexiunile pentru exemplul cu senzorii analogici

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
void setup()
{
    analogReference(DEFAULT); //setarea tensiunii de referință la
    //tensiunea default
    lcd.begin(16, 2); //inițializarea LCD ului
    lcd.setCursor(0,0);
    lcd.print("Cititi senzor");
    pinMode(A1, INPUT); // setarea pinului analogic A1 ca și pin
    //de intrare
}
void loop()
{
    int val = analogRead(A1); //citirea valorii analogice
    lcd.setCursor(0,1);
    lcd.print(val);
}
```

Notă: Dacă nu puteţi folosi potenţiometrul liniar (nu sunt destule disponibile), puteţi folosi **Learning Shield**. Acest Shield are un potenţiometru rotativ conectat la pinul de intrare analogică **A0**. Dacă folosiţi Learning Shield, nu puteţi folosi LCD Shield, dar puteţi afişa numărul citit pe SSD (vedeţi Lab 2), sau puteţi să folosiţi interfaţa Serial.

Exemplul 2: Citirea temperaturii:

Senzor de temperatură folosind LM50 http://www.ti.com/lit/ds/symlink/lm50.pdf

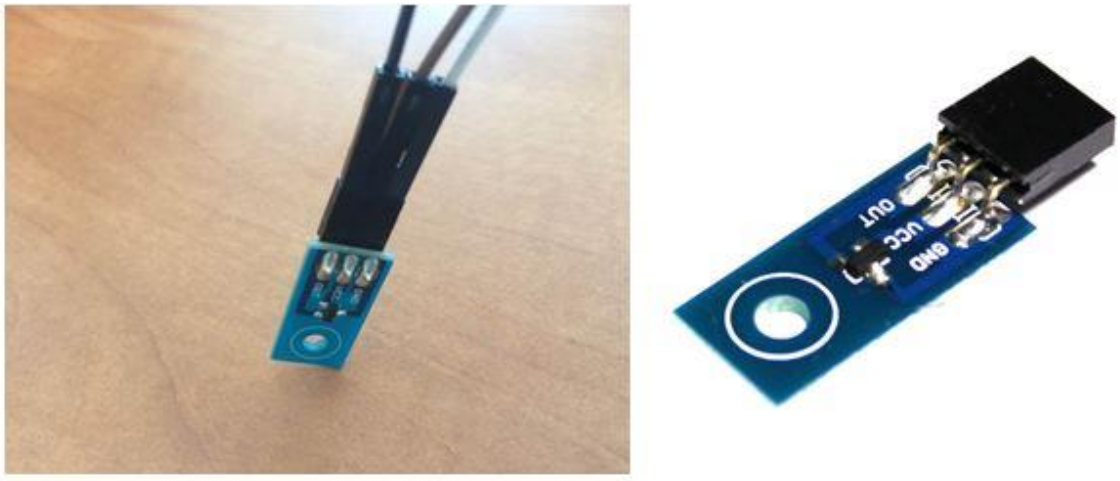


Fig.VII.4. Senzorul de temperatură utilizat

Caracteristici:

• Ieşire liniară +10.0 mV/°C = 0.01V/°C
• Domeniu de temperaturi −40°C ... +125°C
• Deplasament constant +500 mV pentru citirea temperaturilor negative

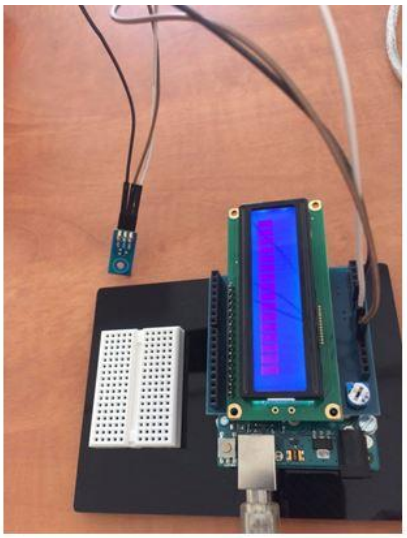


Fig.VII.5. Montajul pe care îl aveţi de realizat pentru exemplul 2

Examplul 2 – Citire temperatură de la senzor - face media a 10 citiri consecutive şi trimite către PC. Conectaţi senzorul (conectaţi ieşirea senzorului la pinul analogic A0, ca în Figura VII.5) şi rulaţi codul de mai jos.

```
float resolutionADC = .0049 ; // rezoluția implicită (pentru
//referința 5V) = 0.049 [V] / unitate
float resolutionSensor = .01 ; // rezoluție senzor = 0.01V/°C
```

```
void setup() {
    Serial.begin(9600);
}
void loop(){
    Serial.print("Temp [C]: ");
    float temp = readTempInCelsius(10, 0); // citeste temperatura
    //de 10 ori, face media
    Serial.println(temp); // afisare
    delay(200);
}
float readTempInCelsius(int count, int pin) {
     // citește temperatura de count ori de pe pinul analogic pin
    float sumTemp = 0;
    for (int i =0; i < count; i++) {
        int reading = analogRead(pin);
        float voltage = reading * resolutionADC;
        float tempCelsius = (voltage - 0.5) / resolutionSensor ;
        // scade deplasament, convertește în grade C
        sumTemp = sumTemp + tempCelsius; // suma temperaturilor
    }
    return sumTemp / (float)count; // media returnată
}
```

Utilizarea ADC-urilor folosind registre AVR

Microcontroller-ul Atmega 2560 conține un singur ADC pe 10 biți cu o rată de eșantionare maximă de 15kS/s la rezoluția maximă. Metoda de eșantionare folosită este aproximarea succesivă. Atmega 2560 oferă posibilitatea de a selecta dintre 16 pini analogici. Un anumit pin analogic este selectat printr-un proces de multiplexare. Măsurători diferenţiale ale tensiunii pot fi făcute folosind patru canale diferenţiale independente. O schemă simplificată a subsistemelor ADC-ului este prezentată in Figura VII.6 :

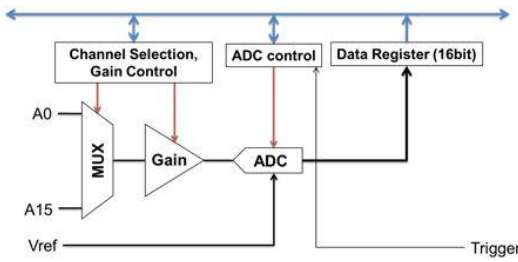


Fig.VII.6. Vedere simplificată a ADC.(sursa: https://bennthomsen.wordpress.com/arduino/peripherals/analogue-input/)

ADC-ul poate fi operat în următoarele 3 moduri:

1. Single Conversion: conversie începută scriind un 1 logic în bitul de începere conversie al ADC-ului
2. Triggered conversion: conversia e începută în momentul în care avem un front crescător

pentru un declanşator (trigger) ales.

3. Free running: Următoarea conversie începe imediat după ce conversia precedentă a fost terminată.

Înainte de a folosi ADC e necesar să selectăm tensiunea de referinţă şi frecvenţa ceasului ADC-ului. Opţiunile pentru tensiunea de referinţă sunt ilustrate în tabelul din Figura VII.7 şi sunt setate utilizând biţii 6 şi 7 din registrul ADMUX, aşa cum e ilustrat în Figura VII.8.

REFS1	REFS0	Voltage Reference Selection[1]
0	0	AREF, Internal V_{REF} turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Internal 1.1V Voltage Reference with external capacitor at AREF pin
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Fig.VII.7. Tensiunea de referinţă ADC. (sursa: datasheetul atmega2560)

Pentru a seta tensiunea de referinţă la VCC, în program trebuie scris ADMUX = (1<<REFS0).

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
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	

Fig.VII.8. Registrul de multiplexare ADMUX. (sursa: atmega 2560)

- **Bit 5 – ADLAR – ADC Left Adjust Result** – Dacă este '1', rezultatul conversiei se aliniază la stânga.
- **Bits 4:0 – MUX4:0 – Analog Channel and Gain Selection Bits** – Aceşti biţi selectează canalul de intrare în convertor.

Prescalerul controlează frecvenţa de lucru a ADC-ului (de obicei între 50 şi 200 kHz). Procesul de conversie necesită 13-14 cicluri de ceas. Opţiunile de divizare sunt prezentate în tabelul din Figura VII.9 de mai jos:

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Fig.VII.9. Valorile de divizare ale ceasului pentru conversie. (sursa: datasheetul atmega 2560)

Aceşti biţi sunt setaţi în registrul Adc Control and Status Register (ADCSRA) precum se vede în Figura VII.10 de mai jos:

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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	

Fig.VII.10. Registrul ADCSRA. (sursa: datasheetul atmega 2560)

Semnificaţia biţilor din registrul ADCSRA:

- **Bit 7 – ADEN – ADC Enable** – Aşa cum îi sugerează şi numele, acest bit activează funcţionalitatea ADC. Dacă acest bit nu este setat, operaţiile de conversie nu ar putea avea loc, iar pinii de intrare se comportă ca pini GPIO.

- **Bit 6 – ADSC – ADC Start Conversion** – La scrierea acestui bit cu valoarea 1 începe o conversie analog/digitală. Acest bit rămâne '1' atâta timp cât conversia e în lucru, după care redevine 0. În mod normal, operaţiile de conversie ADC au nevoie de 13 pulsuri de ceas, dar prima conversie are nevoie de 25 de cicli, pentru iniţializare.

- **Bit 5 – ADATE – ADC Auto Trigger Enable** – Setând acest bit la '1' activează modul de auto-declanşare pentru ADC. Sursa de declanşare se va selecta folosind biţi ADTS din registrul ADCSRB.

- **Bit 4 – ADIF – ADC Interrupt Flag** – În momentul în care o conversie s-a finalizat şi regiştrii s-au actualizat, această valoare devine '1'. Acest bit este de regulă folosit pentru a verifica dacă conversia s-a finalizat sau dacă este încă în curs de desfăşurare.

- **Bit 3 – ADIE – ADC Interrupt Enable** – Când acest bit este setat la '1', se activează întreruperea la finalizarea unei conversii ADC.

- **Bits 2:0 – ADPS2:0 – ADC Prescaler Select Bits** – Aceştia sunt biţii de selecţie a frecvenţei detaliaţi mai sus.

ADCL si ADCH – Regiştrii de date ADC

Aceşti regiştri stochează rezultatul conversiei. Deoarece valoarea numerică rezultată este un număr pe 10 biţi, un singur registru de 8 biţi nu este suficient, astfel încât se folosesc doi, ADCL şi ADCH (octetul superior şi octetul inferior). Împreună, ei pot fi referiţi ca ADCW, iar compilatorul va genera codul pentru înlocuirea ADCW cu perechea ADCH – ADCL.

În funcţie de valoarea bitului ADLAR din ADMUX, cei 10 biţi vor fi aranjaţi majoritar în ADCH (aliniere stânga) sau în ADCL (aliniere dreapta).

În Figura VII.11 sunt ilustraţi atât cei doi regiştri, cât şi efectul setării bitului ADLAR:

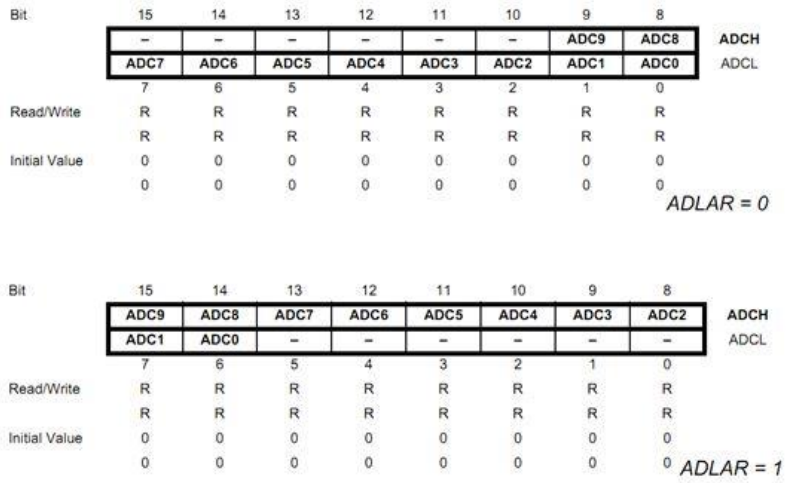


Fig.VII.11. Regiştrii ADCL, ADCH şi efectul bitului ADLAR. (sursa: datasheetul atmega 2560)

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
void setup()
{
    //16MHz/128 = 125kHz ceasul ADC

    ADCSRA |= ((1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0));
    ADMUX |= (1<<REFS0);//Setează tensiunea de referință la
    //Vcc (5v)
    ADCSRA |= (1<<ADEN);        //Activează ADC
    ADCSRA |= (1<<ADSC);
}

void loop()
{
    int val = read_adc(0); //citirea valorii analogice
    lcd.setCursor(0,1);
    lcd.print(val);
}


uint16_t read_adc(uint8_t channel)
{
    ADMUX &= 0xE0;            //șterge biții MUX0-4
    ADMUX |= channel&0x07;    //Setează biții în MUX0-2 pentru noul
    //canal din care va trebui să citim
    ADCSRB = channel&(1<<3); //Setează MUX5
    ADCSRA |= (1<<ADSC);       //începe conversia
    while(ADCSRA & (1<<ADSC));  //Așteaptă până când conversia se
    //termină

    return ADCW;
}
```

7.1. Lucru individual

1. Implementaţi exemplele din laborator. Întrebaţi cadrul didactic în legătură cu orice nelămurire aveţi legată de conectare.
2. Folosind un senzor de lumină, implementaţi o funcţionalitate de tipul lumină de noapte: când nivelul de lumina ambientală scade, creşteţi luminozitatea unui LED.
3. Folosind funcţia micros(), comparaţi viteza de conversie ADC a funcţiei Arduino analogRead cu viteza funcției read_adc() dată ca exemplu. Modificaţi valoarea frecvenţei de conversie, folosind biţii ADPS şi observaţi efectul asupra timpului măsurat.
4. Măsuraţi temperatura folosind senzorul analogic şi senzorul digital din laboratorul anterior. Folosind senzorul digital ca referinţă, realizaţi o procedură de calibrare automată (calculul diferenţei dintre senzori), care va rula în mod continuu atâta timp cât un buton va fi menţinut apăsat. Când butonul de calibrare este eliberat, sistemul va folosi deplasamentul calculat pentru a corecta valoarea analogică şi va afişa cele două valori împreună pe LCD.

VIII. Laborator 8 - Arduino şi aplicaţii WiFi - IoT

IoT - Internet of Things se referă la conectarea diferitelor dispozitive electronice inteligente (inclusiv microcontrollere, senzori) la Internet. În această lucrare de laborator vom utiliza Arduino împreună cu un modul ESP8266 WiFi module. Cu acest dispozitiv, vom controla, dintr-o pagină web găzduită pe Arduino, starea unui LED de pe placă şi vom vizualiza diferite informaţii.
ESP 8266 este un modul autonom WiFi care poate fi programat folosind mediul de dezvoltare Arduino. Programarea acestui dispozitiv poate fi realizată folosind un programator FTDI, sau o placă Arduino cu cipul ATMega328 detaşat.

În acest laborator, vom configura esp8266 prin interfaţa lui serială, folosind comenzi AT.

Conectarea modulului WiFi

Vom conecta modulul ESP la Arduino folosind interfaţa UART Serial1. Conectaţi pinul RX al ESP8266 la pinul **18 (TX1)** Arduino şi pinul TX al ESP8266 la pinul **19 (RX1)**. Conectaţi pinul GND la un pin GND al Arduino şi pinul VCC/3V3 la pinul Arduino de **3.3 V**. Conectaţi şi pinul EN al ESP8266 tot la **3.3 V**.

Consultaţi schema de mai jos pentru realizarea conexiunilor:

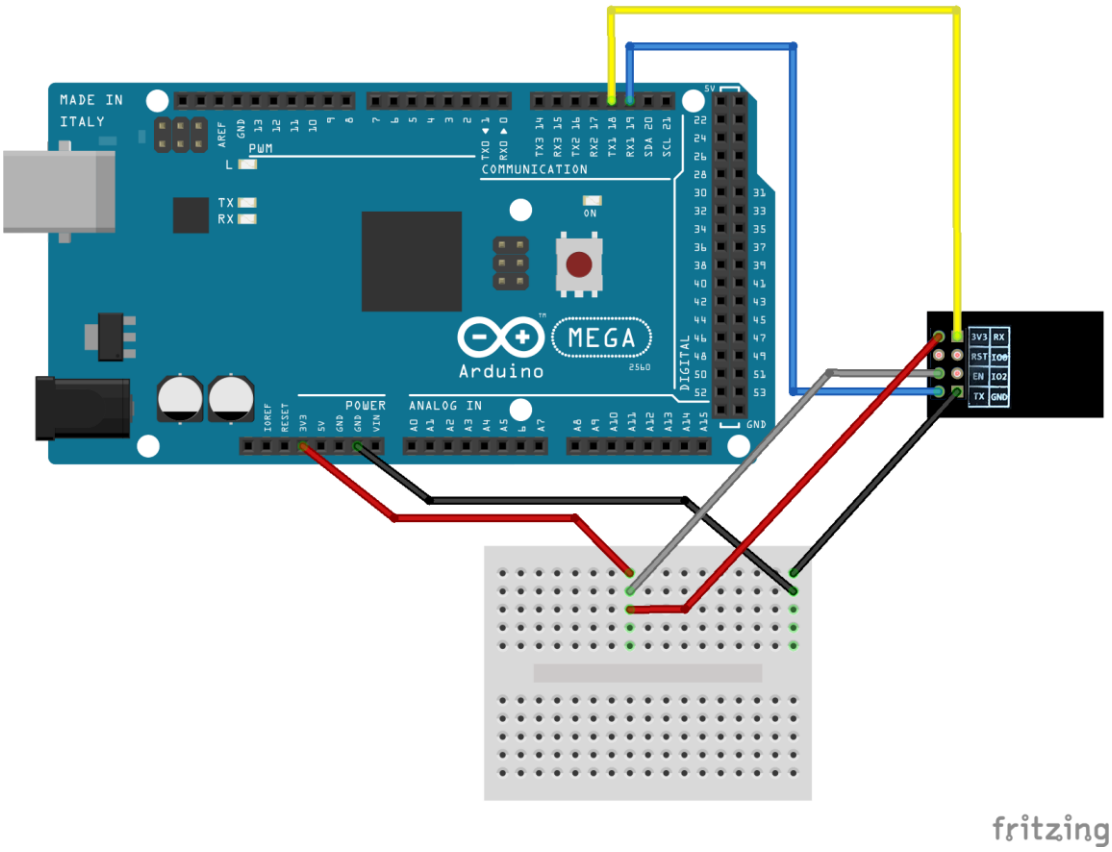


Fig.VIII.1. Conectarea modulului wifi la Arduino prin breadboard

Programul Arduino trebuie să genereze comenzi AT pentru resetarea modulului WiFi (*"AT+RST"*). Următorul pas este configurarea acestui modul ca punct de acces WiFi (*"AT+CWMODE=2"*). După acest pas, se citeşte adresa IP a modulului, 192.168.4.1, folosind comanda: *"AT+CIFSR",* care va tipări şi adresa MAC.

Pentru a obţine informaţia de conectare (SSID), vom rula comanda (*"AT+CWSAP?"*): se va returna numele reţelei şi parola (implicit nu există parolă), şi apoi putem configura sistemul să accepte conexiuni multiple (*"AT+CIPMUX=1"*) şi vom porni serverul web pe portul 80 (*"AT+CIPSERVER=1,80"*).

Fiecare comandă AT trebuie terminată cu *carriage return* şi *newline* (*"\r\n"*).

Mai multe informaţii privind comenzile AT ale ESP8266 pot fi găsite aici: https://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf şi aici: https://github.com/espressif/esp8266_at/wiki.

Comenzile sunt trimise folosind interfaţa serială iniţializată din programul Arduino ("**Serial1**"), folosind metoda "*print*". Răspunsul modulului la comandă este citit şi salvat într-un string şi apoi este afişat pe serial monitor, folosind conexiunea dintre Arduino şi PC, "**Serial**". Consultaţi funcţia "**sendData()"** din exemplul de mai jos.

În bucla principală verificăm dacă datele sunt disponibile pe interfaţa Serial1 şi verificăm dacă sunt date de pe reţea (vor include substring-ul *"+IPD"*). Prima dată trebuie să citim identificatorul conexiunii, deoarece acesta este necesar când transmitem date folosind comanda: *"AT+CIPSEND"*.

O pagină web este construită sub forma unui string şi trimisă la modulul ESP8266. Pagina include text de afişat, două butoane pentru preluarea comenzilor de la utilizator şi apoi alt text de afişat, ce include date generate de Arduino. După ce se transmite comanda AT pentru transmiterea paginii web, se închide conexiunea folosind: *"AT+CIPCLOSE"*.

Mecanismul pentru controlul led-ului pe Arduino se bazează pe folosirea a două butoane în pagina web, fiecare indicând un URL diferit. Primul buton va avea ataşat URL-ul "*/l0*" , iar al doilea "*/l1*". Prin apăsarea acestor butoane, pagina web încearcă să redirecţioneze către aceste adrese şi va genera o cerere către server. Pe Arduino, vom primi această cerere citind datele transmise de ESP8266 (în funcţia "*sendData()*") şi vom verifica dacă conţine substring-urile "*/l0*" sau "*/l1*" (ex: "*response.indexOf("/l0") != -1*").

Datele de la Arduino sunt afişate pe pagina web adăugând rezultatul funcţiei "*readSensor()*" la string-ul paginii web şi trimiterea lui folosind comenzi AT către modulul WiFi. În exemplul de mai jos, funcţia va afişa de fapt rezultatul apelării funcţiei *millis*().

ESP 8266 cod exemplu:

```
#define DEBUG true

void setup() {
    Serial.begin(115200);
    Serial1.begin(115200);
    pinMode(LED_BUILTIN, OUTPUT);
    digitalWrite(LED_BUILTIN, LOW);
    sendData("AT+RST\r\n", 2000, false); // resetare modul
    sendData("AT+CWMODE=2\r\n", 1000, false); // configurare ca
    //access point
    sendData("AT+CIFSR\r\n", 1000, DEBUG); // citeşte adresa IP
    sendData("AT+CWSAP?\r\n", 2000, DEBUG); // citeşte informaţia
```

```
    //SSID (nume rețea)
    sendData("AT+CIPMUX=1\r\n", 1000, false); // configurare
    //conexiuni multiple
    sendData("AT+CIPSERVER=1,80\r\n", 1000, false); // pornire
    //server pe port 80
}

void loop() {
    if (Serial1.available()) {
        if (Serial1.find("+IPD,")) {
            delay(500);
            int connectionId = Serial1.read() - 48; // functia
            //read() returnează valori zecimale ASCII
            // si caracterul '0' are codul ASCII 48
            String webpage = "<h1>Hello World!</h1><a
            href=\"/l0\"><button>ON</button></a>";
            String cipSend = "AT+CIPSEND=";
            cipSend += connectionId;
            cipSend += ",";
            webpage += "<a href=\"/l1\"><button>OFF</button></a>";

            if (readSensor() > 0) {
                webpage += "<h2>Millis:</h2>";
                webpage += readSensor();

            }

            cipSend += webpage.length();
            cipSend += "\r\n";
            sendData(cipSend, 100, DEBUG);
            sendData(webpage, 150, DEBUG);

            String closeCommand = "AT+CIPCLOSE=";
            closeCommand += connectionId; //se adaugă
            //identificatorul conexiunii
            closeCommand += "\r\n";
            sendData(closeCommand, 300, DEBUG);
        }
    }
}

String sendData(String command, const int timeout, boolean debug)
{
    String response = "";
    Serial1.print(command); // trimite comanda la esp8266
    long int time = millis();
    while ((time + timeout) > millis()) {
        while (Serial1.available()) {
            char c = Serial1.read(); // citește caracter următor
            response += c;
        }
    }
```

```
    if (response.indexOf("/l0") != -1) {
        digitalWrite(LED_BUILTIN, HIGH);
    }
    if (response.indexOf("/l1") != -1) {
        digitalWrite(LED_BUILTIN, LOW);
    }
    if (debug) {
        Serial.print(response);
    }
    return response;
}


unsigned long readSensor() {
    return millis();
}
```

Rularea programului

Încărcaţi codul pe Arduino. Asiguraţi-vă că modulul ESP8266 este alimentat la 3.3V şi că pinul EN este conectat tot la 3.3V.
Nu conectaţi aceşti pini la 5V, pentru că veţi distruge adaptorul !

Un dispozitiv client convenabil pentru accesarea reţelei este telefonul dvs. Scanaţi după reţele WiFi şi găsiţi numele reţelei care se potriveşte cu cel afişat pe Serial Monitor.

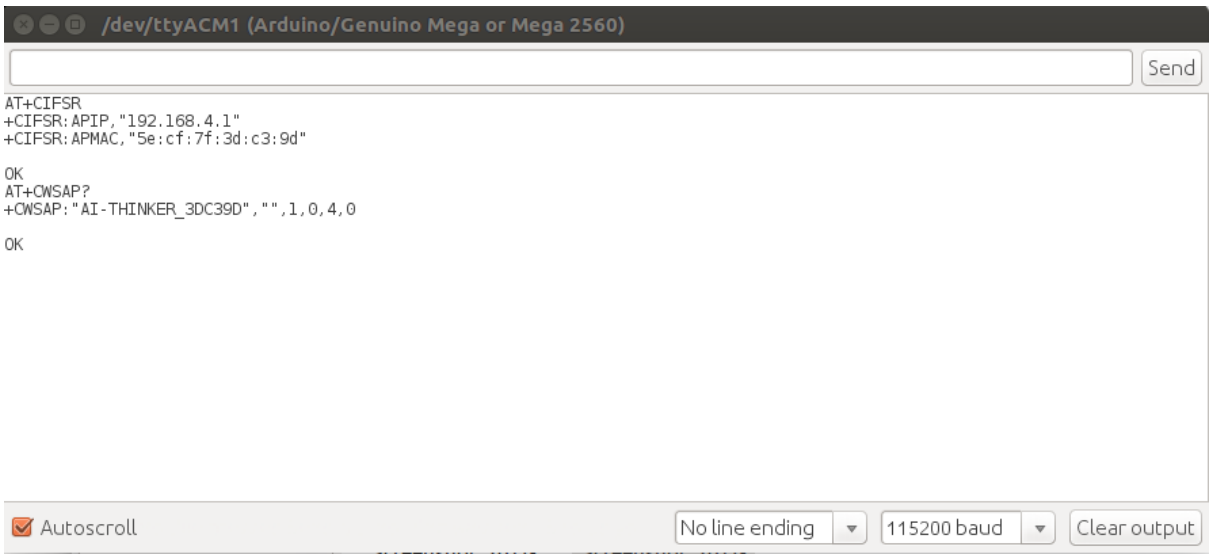


Fig.VIII.2. Căutarea numelui rețelei

Nu uitaţi să schimbaţi **baud rate-ul din serial monitor la 115200!** După cum se observă, numele reţelei din figura de mai sus este “AT-THINKER_3DC39D”, fără parolă. Acelaşi nume trebuie să fie vizibil pe dispozitivul dvs. mobil:

Fig.VIII.3. Numele modulului ca access point wifi

Fig.VIII.4. Pagina web

Notă: Fiecare adaptor are numele lui unic de rețea. Conectați-vă la rețeaua adaptorului vostru propriu, nu la rețelele colegilor !

După ce vă conectaţi la reţeaua WiFi, deschideţi pe telefonul mobil un browser web şi scrieţi adresa serverului web: **192.168.4.1**. Rezultatul ar trebui să fie cel din imaginea de mai sus, partea dreaptă (Fig.VIII.4).

8.1. Lucru individual:

1. Rulaţi exemplul. Asiguraţi-vă că modulul WiFi este corect conectat şi alimentat la 3.3V.

2. Modificaţi exemplul pentru a afişa date suplimentare în browser-ul web. Conectaţi un senzor la Arduino şi apoi, folosind butoanele de pe pagina web, selectaţi informaţia de afişat (millis() sau datele de la senzor). Asiguraţi-vă că pagina web informează clar utilizatorul ce fel de informaţie este afişată.

3. Folosind manualul modulului WiFi, schimbaţi setările reţelei WiFi: schimbaţi numele reţelei (SSID), adăugaţi o parolă şi un mod de criptare.

IX. Laborator 9 - Utilizare motoare si servomotoare. Robotul experimental.

Această lucrare de laborator prezintă utilizarea motoarelor de curent continuu (DC) şi utilizarea servo-motoarelor.

9.1. Motoare DC

Fig.IX.1. Motor DC cu reductor de turație 1:48 (sursa: https://ardushop.ro/en/electronics/64-dc-motor-3v-6v-gear-148.html)

Motoarele de curent continuu (DC motors) clasice convertesc energia electrică în lucru mecanic. Viteza de rotaţie a unui motor este proporţională cu tensiunea de alimentare de la bornele acestuia, iar direcţia de rotaţie depinde de polaritate (conectarea celor 2 fire de alimentare ale motorului la +Vcc şi Gnd, sau vice-versa). Motoarele au cutie de viteze (reductor de turaţie) cu raport de 1:48, ceea ce înseamnă că pentru o rotaţie completă a axului extern se efectuează de fapt 48 de rotaţii ale motorului electric. Folosirea unui reductor are avantajul că măreşte forţa de acţionare, cu costul vitezei.

Datorită faptului că motoarele necesită o intensitate a curentului semnificativă pentru a produce mişcare, ele nu pot fi conectate direct la ieşirile (pinii) unui microcontroller. Se impune separarea semnalelor de comandă de circuitul de putere şi acest lucru se realizează prin folosirea punţilor H ("H bridges"). Punțile H sunt circuite care conțin 4 comutatoare (de obicei tranzistori), numerotate S1, S2, S3 și S4 (Figura IX.2).

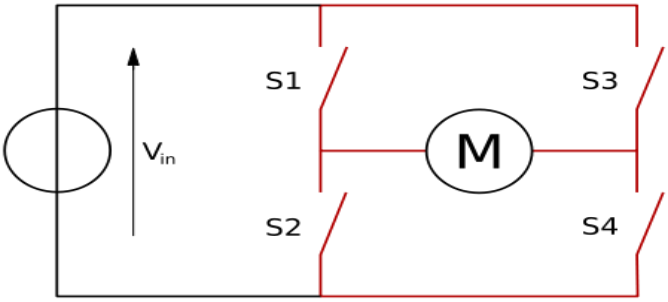


Fig.IX.2. Punte H: S1-S4 sunt comutatoarele, iar M reprezintă motorul (sursa: https://en.wikipedia.org/wiki/H_bridge)

Denumirea de punte „H” vine de la aspectul schemei din figura de mai sus. Porțile din stânga sus (S1) şi dreapta jos (S4) sunt de obicei conectate la un semnal de control comun (“A”), în timp ce porțile din dreapta sus (S3) şi stânga jos (S2) sunt conectate la un alt semnal de control comun, (“B”). Semnalele A şi B sunt exclusive, activarea unuia cauzând rotaţia motorului într-un anume sens. Activarea ambelor semnale în acelaşi timp va scurtcircuita sursa de alimentare.

Cele două stări permise ale comutatoarelor unei punţi H sunt ilustrate mai jos în Figura IX.3:

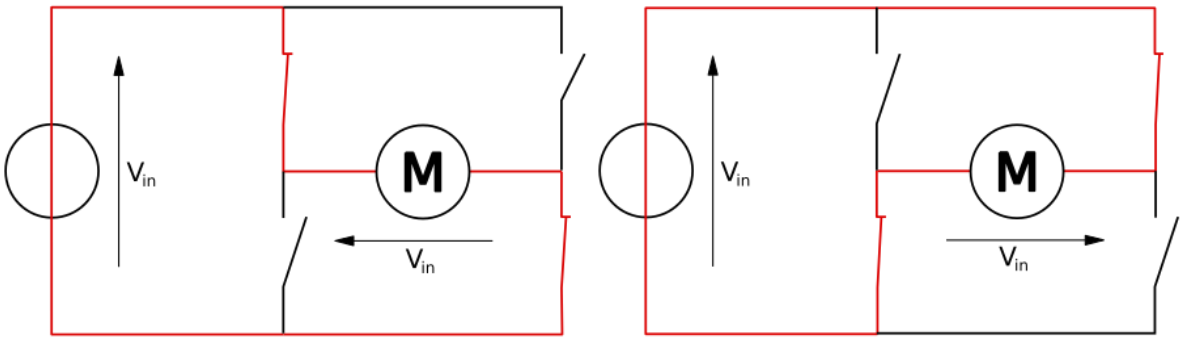


Fig.IX.3. Stările posibile ale comutatoarelor (sursa: https://en.wikipedia.org/wiki/H_bridge)

Prin deschiderea comutatoarelor S1 şi S4, motorul se va roti într-o direcție, iar dacă vom deschide comutatoarele S2 şi S3 motorul se va roti în direcţia opusă.

Pentru această lucrare de laborator vom folosi puntea dual H L298N Dual H-Bridge, care este capabilă să acţioneze două motoare DC în acelaşi timp.

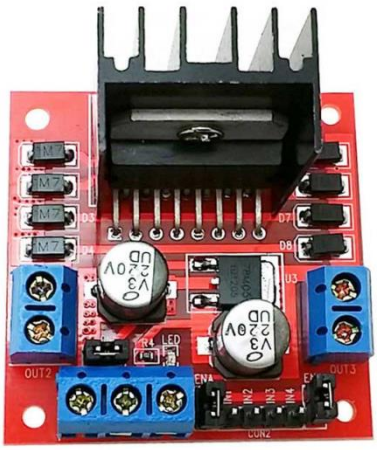


Fig.IX.4. L298N Dual H-Bridge (sursa: https://ardushop.ro/en/electronics/84-dual-h-bridge-for-dc-and-stepper-motors.html?search_query=L298&results=2)

Specificaţiile circuitului:

- Tensiunea de alimentare pentru acţionarea motoarelor (pinul marcat +12V) Vs: 5~35V; dacă dorim să alimentăm din aceeaşi sursă şi placa Arduino, trebuie să ataşăm o tensiune între 7 şi 35V, pentru a permite regulatorului integrat să genereze tensiunea de 5 V pe pinul +5V.
- Curent maxim pentru circuitul de alimentare motoare: 2A

- Tensiune pentru alimentarea circuitelor logice (pinul marcat +5V) Vss: 5 - 7V (poate fi conectat la Arduino + 5V, pentru alimentarea acestuia)
- Curent maxim pentru circuitul logic 36mA
- Nivele ale semnalelor de control: logic 0, $-0.3 \leq Vin \leq 1.5V$, logic 1, $2.3V \leq Vin \leq Vss$
- Putere maximă: 20W

Schema bloc este prezentată în Figura IX.5:

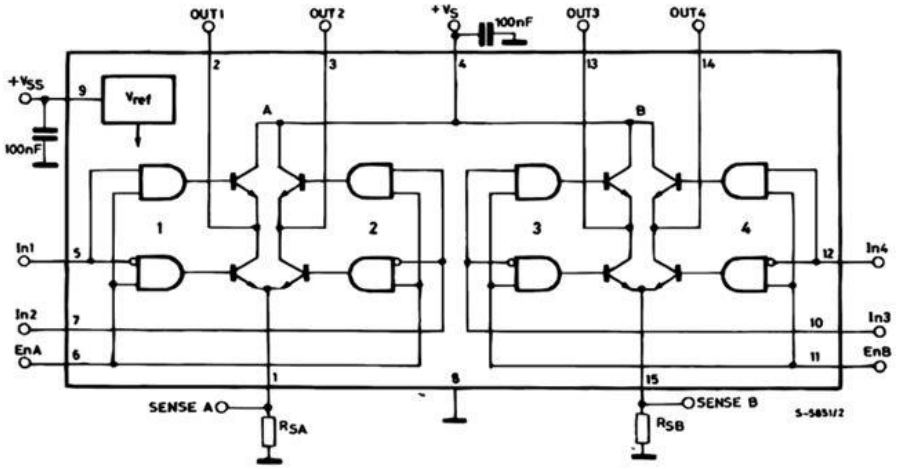


Fig.IX.5. Schema bloc a circuitului driver L298N (sursa: L298 datasheet)

Fiecare motor are trei pini de control. Astfel, primul motor este controlat de pinii EnA, In1 și In2, iar motorul al doilea de pinii EnB, In3 și In4. Pinii En sunt conectaţi la nivelul logic 1 prin jumperi, deci prin program vom controla doar pinii In. Sunt disponibile următoarele combinaţii:

In1	**In2**	**Efect**
0	0	Motor 1 oprit (frână)
0	1	Motor 1 pornit – înainte
1	0	Motor 1 pornit – înapoi
1	1	Motor 1 oprit (frână)

In3	**In4**	**Efect**
0	0	Motor 2 oprit (frână)
0	1	Motor 2 pornit – înainte
1	0	Motor 2 pornit – înapoi
1	1	Motor 2 oprit (frână)

Fig.IX.6. Comenzile pentru motor

Motoarele pot fi acţionate în acelaşi timp. Nu schimbaţi direcţia de rotaţie a motorului fără a-l opri înainte pentru câteva milisecunde.

Turaţia unui motor este dată de tensiunea aplicată acestuia. Deoarece din microcontroller putem să generăm doar semnale de 0 sau 5 V, pentru varierea turaţiei unui motor puteţi folosi un semnal de tip PWM, aplicat pe pinii In1, In2, In3 sau In4.

9.2. Servo-motoare

Spre deosebire de motoarele DC, care produc rotaţie continuă atâta timp cât sunt conectate la o sursă de tensiune, motoarele servo sunt folosite pentru a obţine rotaţii parţiale, stabile şi controlate, pentru efectuarea unor operaţii cu amplitudine mică dar cu precizie ridicată: acţionare mecanism de închidere-deschidere, poziţionare senzori, efectuarea unor gesturi, etc.

Fig.IX.7. Motor Servo (sursa: https://www.indiamart.com/proddetail/sg90-9g-servo-motor-14077708788.html)

Motoarele servo au 3 fire, iar culoarea acestora variază în funcție de producător. Culoarea roșie desemnează de obicei Vcc (5V), în timp ce GND este de obicei negru sau maro. Pe lângă aceste două fire de alimentare, există un al treilea, firul de comandă, care este de obicei galben, portocaliu sau alb. Figura IX.8 ilustrează diferite tipuri de scheme de culori.

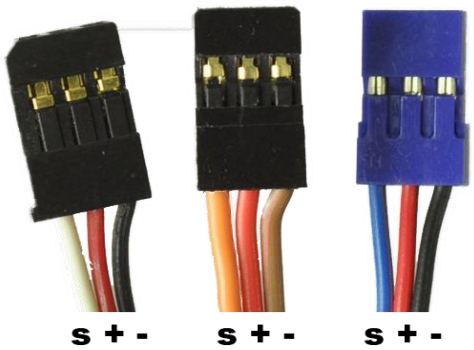


Fig.IX.8. Tipuri de scheme de culori folosite la servo-motoare (sursa: http://www.pitch-play.nl/tips-and-tricks/servo-wires/)

Motorul servo nu va executa (de obicei!) o rotaţie completă, ci va devia de la poziţia de echilibru cu un unghi controlat de tensiunea aplicată pinului de semnal. Folosind un semnal PWM pe acest pin, vom avea control asupra unghiului de rotaţie al motorului.

Cel mai simplu mod de a controla motoarele de tip servo este prin folosirea bibliotecii Servo. Folosirea acestei biblioteci permite controlarea a până la 48 de motoare pe placa Arduino Mega. Dacă se folosesc mai mult de 12 motoare, biblioteca va dezactiva funcţia PWM pe pinii 11 şi 12. La Arduino Uno, această bibliotecă va dezactiva PWM pe pinii 9 şi 10, indiferent de câte motoare se folosesc.

Metodele puse la dispoziţie de biblioteca Servo sunt prezentate mai jos:

servo.attach(pin) / servo.attach(pin, min, max) – atașează obiectul Servo la pini
- servo: un obiect instanță a clasei Servo
- pin: numărul pinului digital unde va fi atașat semnalul pentru motorul Servo
- min (opțional): lățimea pulsului, în microsecunde, corespunzătoare unghiului minim (0 grade) al motorului servo (implicit 544)
- max (opțional): lățimea pulsului, în microsecunde, corespunzătoare unghiului maxim (180 grade) al motorului servo (implicit 2400)

servo.detach() – detașează obiectul de tip Servo de la pin.
boolean val servo.attached() – verifică dacă obiectul de tip Servo este atașat unui pin. Returnează adevărat sau fals.

servo.write (angle) – scrie o valoare (0 .. 180) către servo, controlând mișcarea:
- Pentru Servo standard ⇒ setează unghiul axului [grade] cauzând motorul să se orienteze în direcția specificată.
- Servo cu rotație continuă ⇒ configurează viteza de rotaţie (0: viteza maximă într-o direcție; 180: viteza maximă în direcţia opusă; ≈ 90: oprit)

int val servo.read() – citește unghiul curent al motorului servo, configurat la ultimul apel al metodei **write**().

9.3. Robotul experimental

Pentru desfăşurarea activităţilor de proiect, au fost asamblaţi roboţi experimentali, având următoarele componente:

1. Placă microcontroller compatibilă Arduino Uno
2. Driver motoare L298N Dual H-Bridge
3. 2x Motor DC
4. 1 Motor Servo
5. Carcasă baterii 4xAA (R6)
6. 2 roţi conectate la motoare, 1 roată suport
7. Suport plexiglas
8. Două plăci de prototipizare (Breadboard)
9. 1 senzor sonar (neconectat)

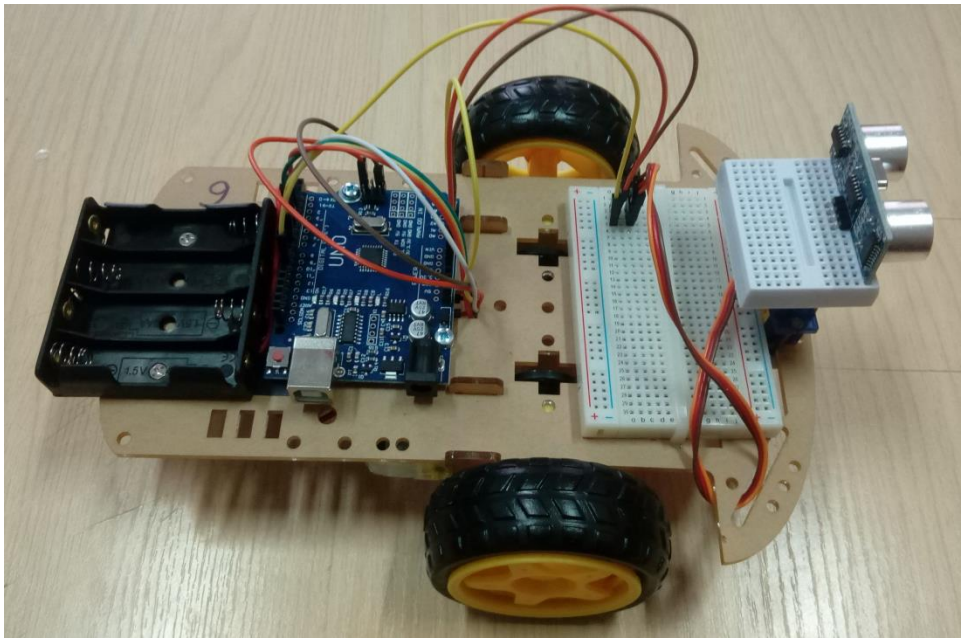


Fig.IX.9. Robotul experimental – vedere de sus – elementele de control

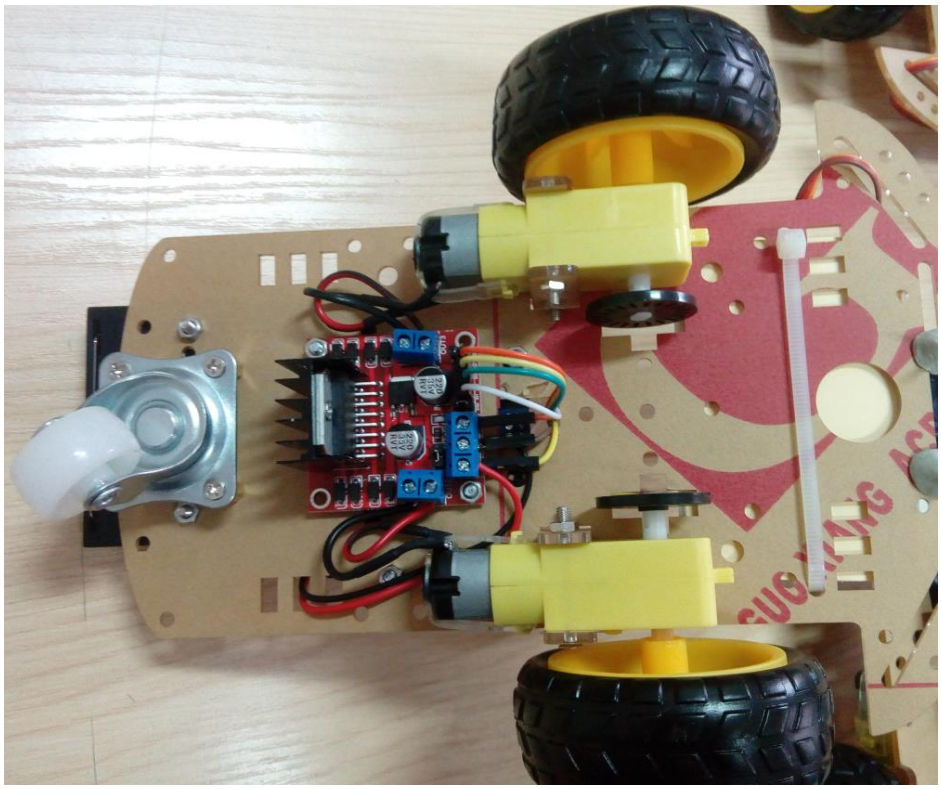


Fig.IX.10. Robotul experimental – vedere de jos – circuitul de putere

Schema conexiunilor dintre elementele robotului este prezentată în Figura IX.11:

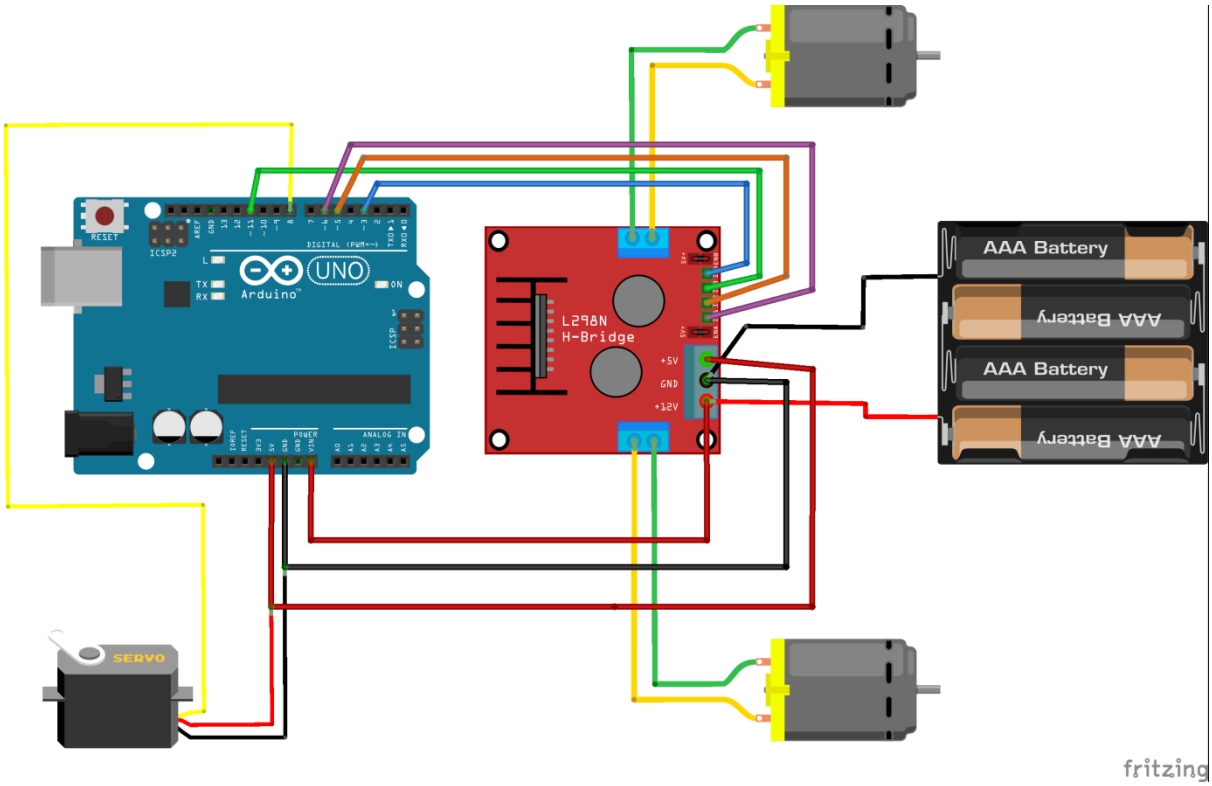


Fig.IX.11. Schema electrică a robotului

Câteva caracteristici importante:

- Alimentarea motoarelor se poate face de la baterii (soluţie neoptimă, conform specificaţiilor L298N, dar care oferă mobilitate), sau de la o sursă externă conectată la mufa jack a plăcii Arduino. Pinul Vin al Arduino este conectat la mufa jack şi este, de asemenea, conectat la pinul +12V al L298N.
- Nu folosiţi sursa externă împreună cu bateriile, pentru că ele vor fi expuse tensiunii acestei surse!
- Ieşirea +5V a L298N este conectată la alimentarea +5V de pe placa Arduino. Pentru economisirea pinilor, a fost utilizat conectorul ICSP.
- Pinii de control ai motoarelor au fost conectaţi la pinii 3 şi 11 (Motor 1) şi 5 şi 6 (Motor 2), pini capabili de generare semnal PWM.
- Controlul motorului Servo este conectat la pinul 8.

Elemente de precauţie:

- Nu lăsaţi robotul să pornească motoarele cât timp este conectat cu cablul de programare la PC. În programul exemplu, care va fi prezentat mai jos, există un timp de avertizare (LED-ul de pe pinul 13 va pâlpâi la început mai rar, apoi mai des, înainte de pornirea motoarelor). Programaţi robotul, apoi deconectaţi cablul USB şi conectaţi sursa externă sau baterii.
- Verificaţi polaritatea sursei externe de alimentare (electrodul pozitiv să fie în centrul mufei) şi tensiunea acesteia (recomandare: 7.5V).
- Nu lăsaţi robotul să pornească motoarele cât timp este cu roţile pe masă! Se va mişca brusc şi poate cauza rupere de cabluri sau poate cădea de pe masă. Ţineţi robotul în mână sau montaţi un

suport care să ţină roţile depărtate de masă. Când doriţi să testaţi modul de deplasare al robotului, puneţi-l pe podea şi lăsaţi spaţiu suficient în jurul lui.

9.4. Program exemplu

Următorul cod este deja programat în microcontrollerul robotului. El va testa toate motoarele, după execuţia codului de avertizare.

```
#include <Servo.h>

// Pinii motor 1
#define mpin00 5
#define mpin01 6

// Pinii motor 2
#define mpin10 3
#define mpin11 11

Servo srv;

void setup() {
    // configurarea pinilor motor ca iesire, initial valoare 0
    digitalWrite(mpin00, 0);
    digitalWrite(mpin01, 0);
    digitalWrite(mpin10, 0);
    digitalWrite(mpin11, 0);

    pinMode (mpin00, OUTPUT);
    pinMode (mpin01, OUTPUT);
    pinMode (mpin10, OUTPUT);
    pinMode (mpin11, OUTPUT);

    // pin LED
    pinMode(13, OUTPUT);
}

// Funcţie pentru controlul unui motor
// Intrare: pinii m1 şi m2, direcţia şi viteza
void StartMotor (int m1, int m2, int forward, int speed)
{

    if (speed==0) // oprire
    {
        digitalWrite(m1, 0);
        digitalWrite(m2, 0);
    }
    else
    {
        if (forward)
        {
            digitalWrite(m2, 0);
```

```
            analogWrite(m1, speed); // folosire PWM
        }
        else
        {
            digitalWrite(m1, 0);
            analogWrite(m2, speed);
        }
    }
}

// Funcție de siguranță
// Execută oprire motoare, urmată de delay
void delayStopped(int ms)
{
    StartMotor (mpin00, mpin01, 0, 0);
    StartMotor (mpin10, mpin11, 0, 0);
    delay(ms);
}

// Utilizare servo
// Poziționare în trei unghiuri
// La final, rămâne în mijloc (90 grade)
void playWithServo(int pin)
{
    srv.attach(pin);
    srv.write(0);
    delay(1000);
    srv.write(180);
    delay(1000);
    srv.write(90);
    delay(1000);
    srv.detach();

}

void loop() {

    // Cod avertizare
    // Blink lent
    for (int i=0; i<10; i++)
    {
        digitalWrite(13, 1);
        delay(200);
        digitalWrite(13, 0);
        delay(200);
    }

    // Blink rapid. Scoateți cablul USB!!!!
    for (int i=0; i<10; i++)
    {
        digitalWrite(13, 1);
        delay(100);
```

```
        digitalWrite(13, 0);
        delay(100);
    }

    digitalWrite(13, 1);

    // Pornirea motorului Servo
    playWithServo(8);

    // Acum se pornesc motoarele DC
    StartMotor (mpin00, mpin01, 0, 128);
    StartMotor (mpin10, mpin11, 0, 128);

    delay (500); // Cât timp e motorul pornit
    delayStopped(500); // Cât timp e oprit

    StartMotor (mpin00, mpin01, 1, 128);
    StartMotor (mpin10, mpin11, 1, 128);

    delay (500);
    delayStopped(500);

    StartMotor (mpin00, mpin01, 0, 128);
    StartMotor (mpin10, mpin11, 1, 128);

    delay (500);
    delayStopped(500);

    StartMotor (mpin00, mpin01, 1, 128);
    StartMotor (mpin10, mpin11, 0, 128);

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

9.5. Desfăşurarea activităţilor de proiect

1. Fiecare echipă de studenţi (1 sau 2) va prelua un robot. Cadrul didactic va scrie în fişa de prezenţă numărul robotului. În săptămânile următoare, studenţii vor folosi aceeaşi roboţi, cu excepţia cazurilor de forţă majoră.
2. Se vor verifica conexiunile, să corespundă cu documentaţia. Se va rula programul de test şi se va urmări ca toate motoarele să funcţioneze corect (fiecare motor trebuie să efectueze rotaţii în ambele sensuri). Dacă există defecte, se vor identifica şi se va încerca remedierea lor.

Punctul 2 este obligatoriu pentru fiecare şedinţă de laborator!

3. Echipele vor adăuga funcţionalităţi robotului, prin modificarea programului şi prin adăugarea de componente suplimentare. Evitaţi modificările permanente, lăsaţi roboţii aşa cum i-aţi găsit.
4. Fiecare echipă trebuie să îşi păstreze codul program confidenţial. Este responsabilitatea fiecărui student să se asigure că nu rămâne cod pe calculator şi că realizările sunt salvate la loc sigur.

5. În ultima şedinţă de laborator, se va evalua proiectul din punctul de vedere al funcţionalităţii, al complexităţii şi al originalităţii. Studenţii vor preda şi o scurtă documentaţie (5-10 pagini).

Fiecare componentă defectată se va înlocui de către persoana vinovată !

Idei pentru funcţii suplimentare (se pot cumula):

- Implementarea unui control precis al motoarelor pentru a asigura o deplasare în linie dreaptă. Implică măsurarea turaţiei şi reglarea pulsului PWM pentru a asigura turaţie egală celor două motoare.
- Detecţia şi ocolirea obstacolelor. Implică utilizarea senzorului sonar împreună cu motorul servo pentru a determina distanţa şi unghiul până la obstacole sau spre spaţiul liber.
- Crearea unei hărţi a mediului înconjurător. Implică detecţia obstacolelor şi evidenţa mişcării proprii, pentru a poziţiona obstacolele pe hartă.
- Utilizarea memoriei EEPROM pentru a memora un traseu parcurs anterior.
- Control la distanţă, folosind telefonul mobil. Implică utilizarea modulului WiFi.
- Comunicare între roboţi prin gesturi.
- Urmărire robot leader.

Este posibil ca funcţiile suplimentare să necesite achiziţionarea unor piese. Dacă aceste piese sunt achiziţionate de către studenţi, ele rămân în proprietatea acestora. Se recomandă achiziţionarea individuală a cel puţin 4 baterii AA alcaline.

Recomandare: puteţi folosi avantajele plăcilor Mega fără a desface plăcile UNO de pe robot. Conectaţi plăcile prin I2C şi folosiţi placa UNO pentru acţionarea motoarelor, iar placa Mega pentru celelalte funcţii. Conectaţi împreună pinii 5V şi GND ai celor două plăci, pentru alimentare simultană.

Referinţe suplimentare:

1. Senzor fotoelectric pentru utilizarea împreună cu roata perforată, pentru măsurarea turaţiei:
https://ardushop.ro/ro/home/146-senzor-de-intrerupere-infrarosu.html?search_query=fotoelectric&results=1
sau
https://www.robofun.ro/senzor-ir-break-beam-led-3mm

2. Senzorul de tip sonar pentru determinarea distanţelor:
https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

3. Utilizarea senzorilor de distanţă Sharp:
http://www.robotshop.com/letsmakerobots/files/IR_SHARP.doc

4. Utilizarea memoriei EEPROM integrată AVR:
https://www.arduino.cc/en/Reference/EEPROM

Anexa 1 – Folosirea mediului Processing

1. Introducere în Processing

Pentru o mai bună vizualizare a datelor generate de un microcontroller, date care sunt communicate către PC folosind interfaţa serială UART, se pot folosi diferite unelte software. În categoria acestor unelte găsim programul Processing (https://processing.org/) . Folosind Processing putem crea interfețe grafice pentru a vizualiza datele sub formă de grafice sau tabele.

Processing este o soluție Open Source, la fel ca și Arduino, având interfața grafică foarte asemănătoare.

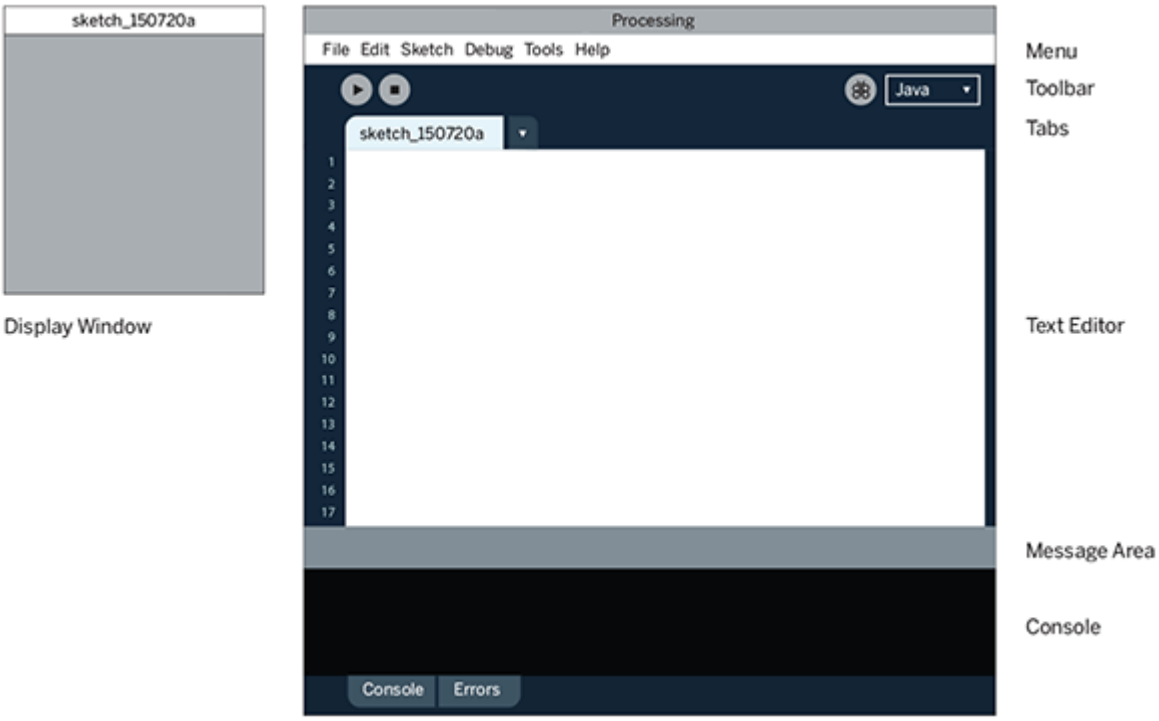


Fig.A1.1. Mediul de dezvoltare IDE

În fereastra text editor vom scrie programele propriu-zise. Zonele “Message Area” și “Console” vor oferi feedback pe masură ce programul este compilat sau executat.

Asemănător programelor Arduino, programele Processing vor avea o parte de inițializare și o parte de cod care va rula în buclă. Inițializarea se face cu “setup()”, iar tot ce este scris în funcția “draw()” reprezintă partea de cod care va fi executată în buclă (echivalentul funcției “loop()” din Arduino).

Exemplul 1 - Desenare linie și elipsă simplă în Processing:

```
void setup() {
    size(500, 300);
}

void draw() {
    line(15, 15, 15, 50);
    ellipse(150, 150, 80, 80);
}
```

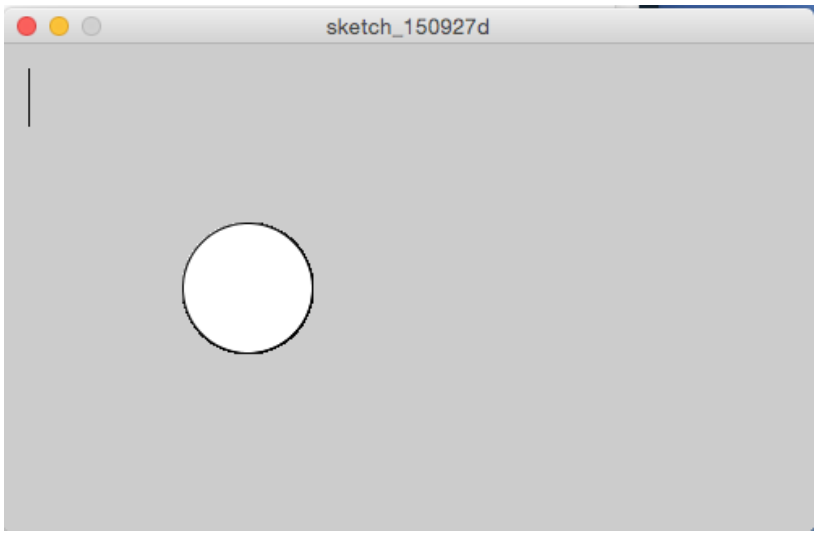


Fig.A1.2. Desenarea primitivelor grafice in processing

În "setup()" vom defini mărimea ferestrei, în acest exemplu ea fiind de 500 pixeli lățime, respectiv 300 pixeli înălțime. Funcția "draw()" va conține comenzi grafice cu care putem desena în fereastră diferite elemente geometrice: elipsă, dreptunghi, linii, pătrat, etc. Culoarea de fundal a ferestrei poate fi definită tot în "setup()":

```
void setup() {
    size(500, 300);
    background(0, 0, 0);
}
```

Culorile sunt definite prin valori cuprinse între 0-255 pentru fiecare canal de culoare (Red, Green, Blue - RGB), așadar pentru o fereastră cu fundal alb vom scrie background(255, 255, 255), care este echivalent și cu background(255). Se pot folosi și culori specificate în formatul web: background(#ffffff).

Sistemul de coordonate în Processing își are originea în colțul din stânga sus. O linie este definită între două puncte A(x1, y1) și B(x2, y2).

Un dreptunghi este definit în felul următor: *rect(x, y, width, heigth)*, unde (x, y) reprezintă colțul din stânga sus al dreptunghiului.

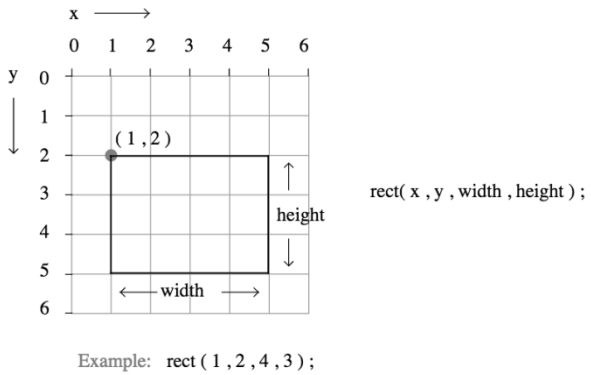


Fig.A1.3. Coordonate pentru desenarea unui patrat.
(sursa: https://processing.org/tutorials/drawing/)

Dreptunghiul se poate defini și ținând cont de punctul central, dacă se specifică *rectMode(CENTER)* înainte de desenare.

O alta metodă de specificare a dreptunghiului poate fi și prin specificarea colțurilor stânga sus / dreapta jos, folosind modul *rectMode(CORNERS)*. Pentru mai multe exemple puteți consulta documentația Processing: https://processing.org/tutorials/drawing/.

2. Folosirea datelor de la senzori. Exemplificare cu Processing

Datele preluate de Arduino de la senzori vor putea fi prelucrate și vizualizate mai ușor folosind mediul Processing. Vom folosi exemplul de citire a temperaturii din laboratorul 7 (**pe care îl veți modifica pentru a transmite doar informația numerică pe portul serial!**) și vom scrie pe interfața serială valoarea temperaturii. În Processing vom citi valorile transmise pe interfața serială și le vom folosi pentru a afișa un grafic al evoluției temperaturii.

Exemplul 2 - Citire date de pe interfața serială folosind Processing:

```
import processing.serial.*;

Serial myPort;
int xPos = 1;       // indexul axei X a graficului
int lastxPos=1;
int lastheight=0;

void setup() {
    size(400, 300);
    println(Serial.list());
    int lastport = Serial.list().length;
    String portName = Serial.list()[lastport-1]; // ultimul port
    //serial din listă
```

```
    myPort = new Serial(this, portName, 9600);
    myPort.bufferUntil('\n');

    background(0);
}

void draw() {
    String inString = myPort.readStringUntil('\n');
    if (inString != null) {
        inString = trim(inString);
        float inByte = float(inString);
        println(inString);
        inByte = map(inByte, -5, 45, 0, height); // mapare
        //temperatură la dimensiunile ferestrei

        stroke(231, 76, 60);// culoare linie (componente de
        //culoare RGB)
        strokeWeight(2);        // grosime linie
        line(lastxPos, lastheight, xPos, height - inByte); //
        //desenare linie
        lastxPos= xPos;
        lastheight= int(height-inByte);
        xPos++;
    }
}
```

Pentru citirea datelor este necesară utilizarea bibliotecii Serial, inclusă în Processing. Funcția "Serial.list()" va afișa o listă a tuturor interfețelor seriale conectate la computer și ne va ajuta pentru alegerea portului corect în metoda "setup()".

Inițializarea portului serial pentru citire se face astfel:

```
myPort = new Serial(this, portName, 9600);
```

Vom citi datele până la întâlnirea caracterului "newline":

```
myPort.bufferUntil('\n');
```

Se citește câte un singur string o dată de pe serial folosind:

```
myPort.readStringUntil('\n');
```

În cazul în care avem date, vom converti string-ul citit la tipul float și apoi desenăm pe ecran linii pentru a genera un grafic al evoluției temperaturii. Va rezulta un ecran similar cu cel din figura de mai jos:

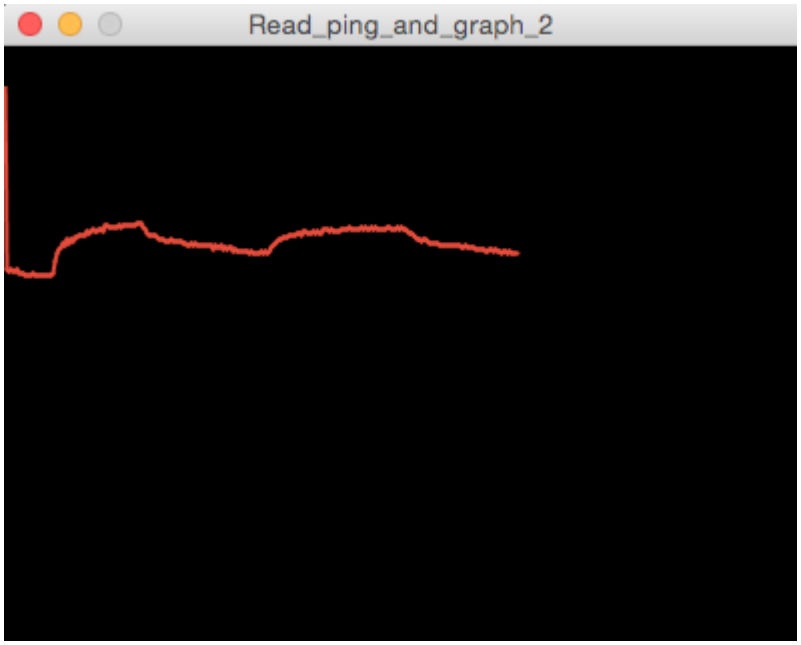


Fig.A1.4. Desenarea unui grafic din datele senzoriale

Liniile sunt desenate unind coordonatele curente cu cele precedente, pentru a avea un grafic continuu.

3. Scriere date în Processing

Biblioteca pentru comunicarea serială inclusă în mediul Processing oferă, pe lângă funcționalitatea de citire a datelor de pe interfața serială, și posibilitatea de scriere. Astfel, folosind funcția *write* se pot trimite caractere.

```
import processing.serial.*;

Serial myPort;

// Afișare porturi seriale disponibile:
printArray(Serial.list());

myPort = new  Serial(this,  Serial.list()[Serial.list().length-1],
9600);

// Scrie "A" pe portul serial
myPort.write(65);
```

Sursa: https://processing.org/reference/libraries/serial/Serial_write_.html

Există posibilitatea de a transmite un șir de mai multe caractere, astfel:

```
myPort.write("Hello");
```

4. Citirea datelor transmise de Processing în Arduino

Datele trimise cu Processing vor putea fi citite în Arduino folosind interfața serială (cu biblioteca "Serial"). Citirea unui șir de caractere se face în modul următor:

```
string dataFromSerial = Serial.readStringUntil('\n');
```

Exemplul 3: citire date în Arduino – codul pentru Arduino:

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
String inputString = "";         // string pentru datele de intrare
boolean stringComplete = false;  // indică string complet

void setup() {
    Serial.begin(9600);
    lcd.begin(16, 2);
    lcd.print("Read Processing:");    // afișare mesaj pe LCD
    inputString.reserve(16); // rezervă 16 bytes pentru
    //inputString
}

void loop() {
    if (stringComplete) {
        lcd.setCursor(0, 1);
        lcd.print(inputString);
        inputString = "";
        stringComplete = false;
    }
}

void serialEvent() {
    while (Serial.available()) {
        inputString = Serial.readStringUntil('\n');
        stringComplete = true;
  }
}
```

Codul de mai sus utilizează shield-ul LCD pentru a afișa mesajul citit de pe interfața serială.

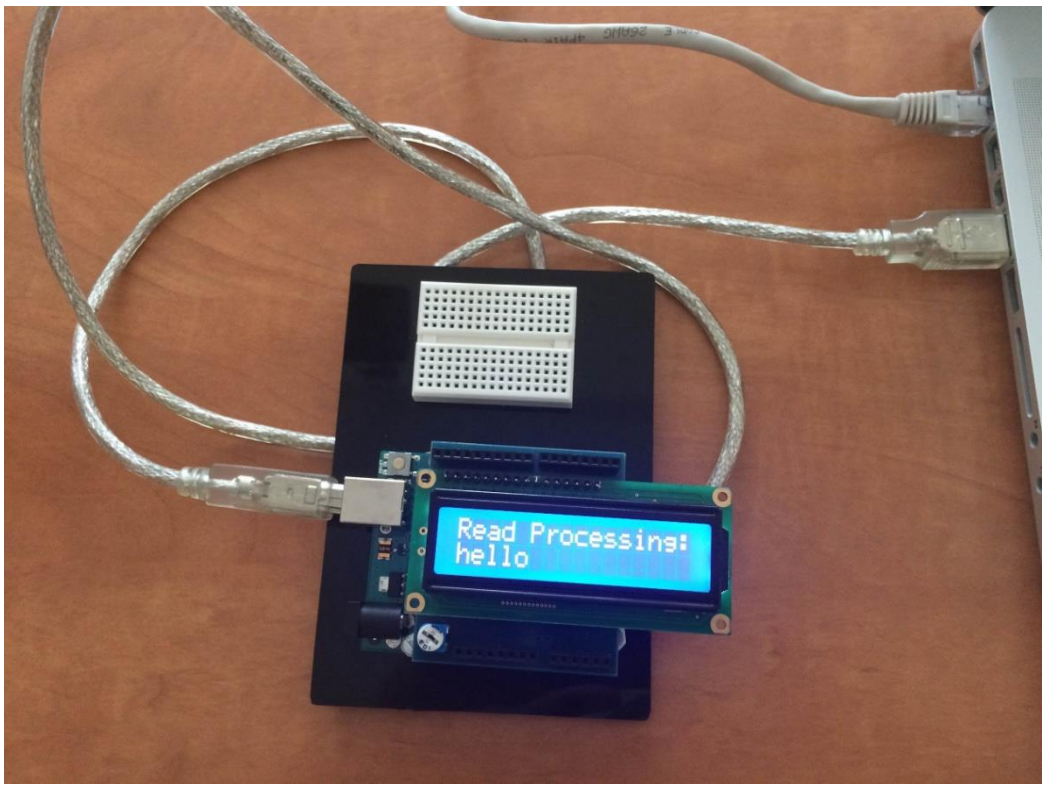


Fig.A1.5. Exemplu de citire șir de caractere transmis de Processing folosind interfața serial

În Processing vom crea o interfață grafică simplă, cu un singur buton.

Fig.A1.6. Interfața grafică minimală în Processing, cu un singur buton

Exemplul 3 - Codul pentru Processing:

```
import processing.serial.*;

int rectX, rectY;
int rWidth = 120;
int rHeight = 40;
```

```
color rectColor;
color rectHighlight;
boolean rectOver = false;


int c = 0;

Serial myPort;

void setup() {
    size(600, 400); // mărimea ferestrei
    rectColor = color(0); // culoarea de bază a butonului
    rectHighlight = color(51); // culoarea butonului la trecerea
    //mouse-ului

    // poziția butonului
    rectX = width/2-rWidth-10;
    rectY = height/2-rWidth/2;

    printArray(Serial.list());
    myPort = new Serial(this, Serial.list()[Serial.list().length-
1], 9600);
}

void draw() {
    update(mouseX, mouseY); // pentru schimbarea culorii butonului
    //la trecerea mouse-ului
    // schimbă culoare buton la trecerea mouse-ului
    if (rectOver) {
        fill(rectHighlight);
    }
    else {
        fill(rectColor);
    }

    // desenează conturul butonului
    stroke(255);
    rect(rectX, rectY, rWidth, rHeight);

    fill(255);
    text("SEND", rectX + rHeight + 5, rectY + rHeight/2 + 5);
}

void update(int x, int y) {
    rectOver = overRect(rectX, rectY, rWidth, rHeight);
}

// acțiunea la apăsarea butonului de mouse
void mousePressed() {
    if (rectOver) {
        myPort.write("hello"+c);
        c++;
    }
```

```
}

// returnează true dacă mouse-ul este deasupra butonului
boolean overRect(int x, int y, int width, int height) {
    if (mouseX >= x && mouseX <= x+width &&
    mouseY >= y && mouseY <= y+height) {
        return true;
    }
    else {
        return false;
    }
}
```

Lucru individual:

1. Rulați exemplele din lucrarea de laborator.
2. Modificați exemplul 2 pentru a reseta graficul odată ce liniile ajung în capătul ecranului.
3. **Opțional, în loc de punctul 2:** modificați exemplul 2 pentru ca graficul să se deplaseze în mod continuu spre stânga, atunci când se umple.
4. Adăugați la temperatura afișată pe grafic informații sub formă de text. Folosiți funcția **text (text_de_afisat, x, y)**, unde x este coordonata orizontală și y coordonata verticală. Afișați informația textuală la un interval de timp care să asigure că textele afișate nu se vor suprapune.
5. Modificați exemplul 3 (cod Arduino și cod Processing) pentru a controla starea unor LED-uri folosind interfața grafică. Conectați la Arduino blocul de led-uri. În Processing, realizați patru butoane în interfața grafică. La apăsarea unui buton din interfața programului Processing, se va schimba starea led-ului corespunzător conectat la Arduino.
6. Folosiți funcțiile pentru "keyboard" din Processing pentru a citi tastele apăsate. Afișați tasta apăsată în consola Processing, dar și pe LCD al Arduino.

Referințe:

1.https://www.arduino.cc/en/Reference/AnalogRead
2.https://www.arduino.cc/en/Reference/AnalogReference
3.https://processing.org/
4.https://processing.org/examples/keyboard.html

Anexa 2 - Robotic Operating System (ROS) și Arduino

Ce este ROS și ce putem face cu el

ROS nu este un adevărat sistem de operare, ci o colecție de unelte de dezvoltare software proiectate în jurul unei arhitecturi de comunicare între procese și mașini. Acest middleware robotic are un limbaj de programare și o arhitectură independentă de hardware, ce oferă servicii precum abstractizarea hardware, control de nivel jos al dispozitivului, transmiterea de mesaje între procese, managementul pachetelor și altele. ROS nu este proiectat pentru a înlocui un sistem de operare real, ci pentru a funcționa împreună cu unul. Deoarece ROS folosește multe biblioteci open source, cel mai bun suport software pentru el se găsește pe sistemele de operare bazate pe Linux.

Robotic Operating System poate fi utilizat în aplicații unde este necesar calculul distribuit, sau reutilizabilitatea și testarea rapidă. De exemplu, într-o aplicație de calcul distribuit mulți roboți diferiți pot să depindă de un software care rulează pe calculatoare diferite și pornește multiple procese. ROS este o alegere bună atunci când cineva dorește să reutilizeze un cod care rulează pe un robot, de exemplu funcții de generare de hărți, sau de planificare a traseului.

Organizarea ROS

Sistemul de fișiere ROS este organizat pe două nivele:

- Pachete
  Acestea sunt cele mai de jos nivele de organizare și sunt proiectate pentru o singură funcționalitate. Există și un fișier numit *manifest,* care este responsabil pentru descrierea pachetului și pentru definirea dependințelor între pachete.

- Stiva
  Acestea sunt colecții de pachete care formează biblioteci de nivel înalt. Există un fișier *manifest* și în stivă, cu același scop ca cel din pachet.

Platforma de comunicare ROS

Nodurile ROS sunt responsabile pentru calcule specifice. Ele reprezintă procesele distribuite în rețeaua ROS, care pot oferi sau pot lua date din rețea. Exemplu de sarcini ale nodurilor:

- Controlul roților robotului
- Captură de imagini de la camere
- Vizualizare grafică a datelor din sistem

Nucleul ROS este compus din trei programe care sunt necesare pentru a rula restul componentelor:

1. **ROS master**
   Master-ul ROS este centrul ROS și se comportă ca un server DNS. Master-ul ROS este un centru RPC centralizat care negociază conexiunile de comunicație, înregistrează și caută nume pentru resursele grafului ROS.
   El stochează teme și informații despre înregistrarea serviciilor pentru nodurile ROS și face apeluri inverse către noduri când informația de înregistrare se schimbă. Nodul master permite de asemenea nodurilor să facă conexiuni în mod dinamic pe măsură ce noi noduri sunt pornite.
2. **Serverul de parametri**
   Rulează ca parte a masterului ROS și este responsabil pentru stocarea parametrilor persistenți de configurare sau a altor date arbitrare. Cât timp nucleul rulează, datele sunt stocate în serverul de parametri. Serverul nu este proiectat pentru înaltă performanță și de

aceea este recomandat pentru stocarea parametrilor de configurare. Convenția de denumiri a ROS previne conflictul de date, prin folosirea spațiilor de nume (namespace).

3. **Rosout**
   Este o simplă ieșire, bazată pe rețea, pentru mesaje ce pot fi citite de operatorul uman.

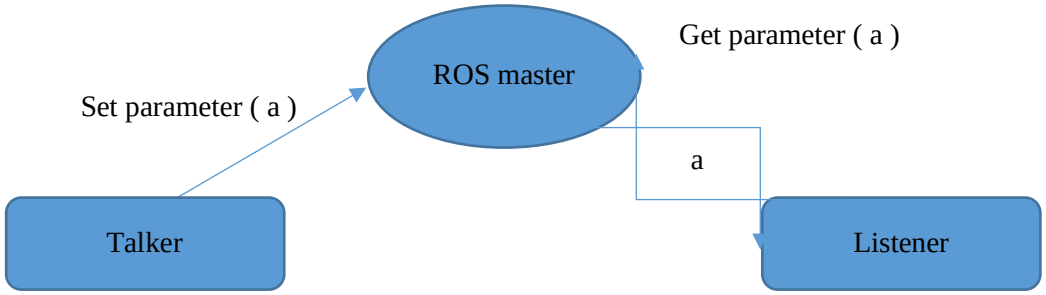


Fig.A2.1. Comunicare in ROS

Mesaje
Nodurile comunică unul cu altul prin transmiterea mesajelor, care sunt structuri de date ce includ un câmp care specifică tipul. Mesajele pot fi direcționate ori prin utilizarea temelor ori prin servicii. Tipurile de date standard (sau primitive) sunt: int, float, array[], etc.

Teme și servicii
Temele și serviciile sunt metode de comunicare între noduri. Serviciile reacționează la o interogare făcută de la terminal sau de un nod și transmit răspunsul dat de un alt nod, ce oferă respectivul serviciu. Temele necesită o abonare la un nod care va difuza informația. O altă diferență dintre teme și servicii este că temele sunt torente de comunicație asincronă cu mai multe surse și destinații, pe când serviciile sunt funcții de comunicare cu o singură sursă și mai mulți destinatari, sincrone.

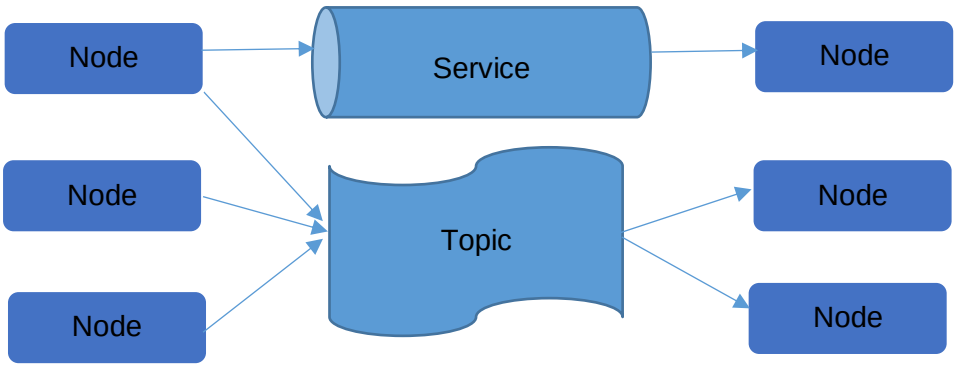


Fig.A2.2. Servicii si topice in ROS

Figură inspirată din „ROS Tutorial book, Robotics Operating System“, Antonio Marin-Hernandez, October 31, 2014

Modelul publicare-abonare este o paradigmă flexibilă unde cei care publică și cei care recepționează nu sunt conștienți unii de existența celorlalți. Un singur nod poate publica și se poate abona la mai multe teme. Un nod transmite un mesaj prin publicarea lui pe o anume temă și un nod care are nevoie de anumite date specifice trebuie să se aboneze la o temă specifică. Tipul temei este definit de tipurile de mesaje pe care nodul le transmite. Pentru „jocul“ publicare-abonare nu este nevoie de o ordine anume a execuției.

Serviciile nu pot folosi paradigma publicare-abonare. Ele implementează o funcționalitate cerință-răspuns (prin utilizarea unei perechi de structuri de mesaje, unul pentru cerință și unul pentru răspuns). În general este un nod care oferă un serviciu cu un nume specific. Nodul client „consumă“ serviciul prin transmiterea unui mesaj cerere și așteptarea unui răspuns.

În următoarele exemple vom vedea trei programe simple care folosesc rosserial. Versiunea de ROS folosită este ROS Indigo Igloo (a 8-a distribuție oficială ROS). Microcontrollerul folosit este atmega 2560. Instalarea mediului de dezvoltare nu va fi detaliată aici, dar se pot găsi instrucțiunile pe Internet.

Ghid de configurare a mediului ROS

Pentru configurarea mediului ROS putem să urmăm instrucțiunile de pe site-ul oficial ROS, sau putem utiliza o mașină virtuală cu ROS preinstalat. Vom utiliza versiunea preinstalată ROS indigo, dar încurajăm cititorii să urmeze și ghidul de instalare pas cu pas și să exploreze noi versiuni ale mediului ROS.

Linkul pentru descărcarea mașinilor virtuale pe 32 și pe 64 de biți se găsește în finalul Anexei 2, la referinta cu numărul 7.

Hipervizorul folosit este Virtual Box. Pentru hipervizor este esențial să se descarce și să se instaleze pachetul de extensii pentru versiunea utilizată, astfel încât să se poată accesa porturile USB din interiorul mașinii virtuale.

Odată ce VirtualBox este instalat, trebuie făcute anumite modificări la configurarea mașinii virtuale. Apăsați butonul “settings” și navigați la secțiunea USB. Bifați opțiunea “Enable USB controller” și opțiunea “Enable USB 2.0 (EHCI) Controller”.

Un alt pas important de făcut, dacă mașina gazdă rulează Linux, este acela de a adăuga utilizatorii virtual box la grupul de utilizatori ai mașinii gazdă. Acest lucru se realizează cu comanda:

sudo adduser $USER vboxusers

Pentru a verifica dacă placa Arduino este recunoscută de mașina gazdă (sub Linux), deschideți un terminal și tastați lsusb. Dacă placa este recunoscută, veți vedea în listă un dispozitiv ce conține numele Arduino.

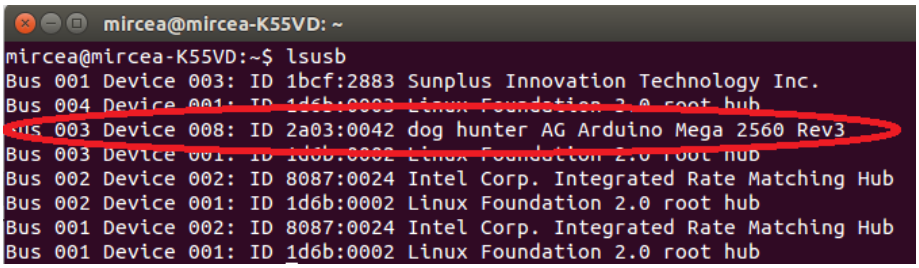


Fig.A2.3. Portul USB conectat la placa Arduino

După ce toate opțiunile de mai sus au fost bifate, utilizatorii virtual box au fost incluși în grupul utilizatorilor mașinii gazdă și placa Arduino este recunoscută de calculatorul gazdă, adăugați un nou

filtru USB cu toate câmpurile setate la valorile dispozitivului USB atașat la PC-ul gazdă. Pașii descriși mai sus sunt vizualizați grafic în figura următoare:

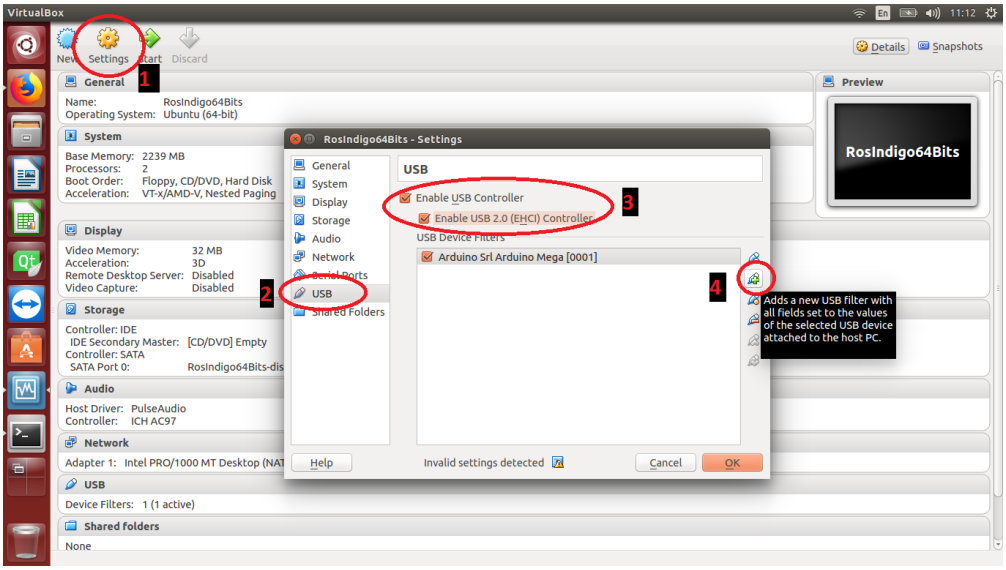


Fig.A2.4. Pasi necesari vizualizarii placii Arduino in mașina virtuala cu ROS

Acum puteți porni mașina virtuală apăsând butonul de start (săgeata verde). Numele utilizator implicit și parola implicită sunt "viki".

După pornirea mașinii virtuale, trebuie să instalați mediul de programare Arduino. Folosind pachetul rosserial_arduino, puteți utiliza ROS împreună cu Arduino. Acest pachet funcționează pe baza interfeței UART a Arduino și oferă posibilitatea de comunicare ROS prin considerarea Arduino ca nod ROS. La fel ca la instalarea oricărei biblioteci, biblioteca ros_lib trebuie copiată în directorul bibliotecilor Arduino pentru a putea fi accesată de IDE-ul Arduino.
Pentru a instala biblioteca în mașina virtuală ROS, trebuie să tastați următoarele comenzi:

sudo apt-get install ros-indigo-rosserial-arduino
sudo apt-get install ros-indigo-rosserial

Astfel se crează biblioteca ros_lib, care trebuie copiată în mediul Arduino. Trebuie să ștergeți libraries/ros_lib, deoarece existența acestuia cauzează erori. Directorul "sketchbook" este directorul unde Arduino sub Linux salvează sketck-urile (proiectele utilizator).

cd <sketchbook>/libraries
rm -rf ros_lib
rosrun rosserial_arduino make_libraries.py .

După parcurgerea acestor pași, ros_lib va apărea în secțiunea exemple a Arduino IDE.

Exemplul 1: Publicare
În acest exemplu vom crea un program de publicare pe microcontrollerul atmega 2560 și vom arăta mesajele transmise de microcontroller, pe o anume temă, într-o fereastră consolă. Pentru acest prim exemplu, avem nevoie de o placă Arduino și de un cablu USB pentru conectarea ei cu calculatorul ce rulează ROS.

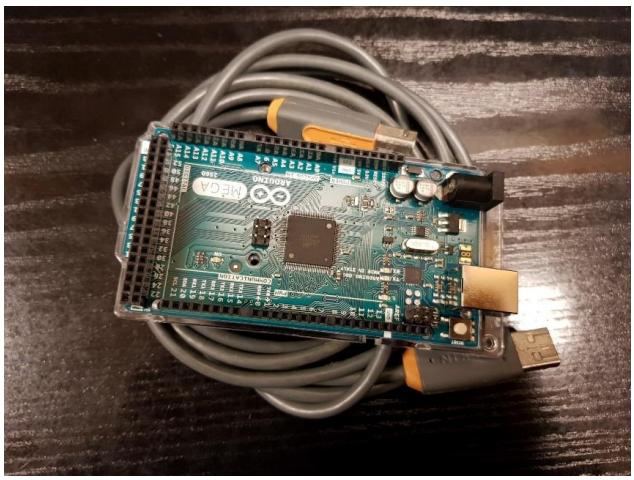


Fig.A2.5. Placa Arduino si cablul USB

```
// Pentru folosirea bibliotecii rosserial, includeți ros.h înainte
//de orice alt header.
#include <ros.h>
#include <std_msgs/String.h>

//instanțierea handle-ului de nod, ce permite programului să
//creeze publicatii și //abonamente. Acest handle va trata
//problema comunicației seriale
ros::NodeHandle nodeHandle;
std_msgs::String msg;

// Trebuie să instanțiem publicațiile și abonamentele pe care le
//utilizăm.
// Aici instanțiem o publicație cu numele “warning”. Al doilea
//parametru este referința
// la obiectul de tip mesaj folosit pentru publicare

ros::Publisher pub(“warning”, &msg);

char warning[26] = “This is your last warning!";

void setup() {
    // inițializare handle nod ROS
    nodeHandle.initNode();
    // anunță orice temă ce va fi publicată
    nodeHandle.advertise(pub);
}
void loop()
{
    // nodul publică mesajul "This is your last warning!"
    msg.data= warning;
    pub.publish(&msg);
    // apelare funcție spinOnce(), care va trata toate funcțiile
    //callback de comunicare ROS
    nodeHandle.spinOnce();
    delay(1000);
```

```
}
```

În primele două linii de cod vom include antetele necesare programului nostru. Antetul ros.h trebuie să fie întotdeauna pe prima linie. După aceea, vom crea handle-ul de nod. Cu acest handle vom crea publicații și abonamente pentru nodul nostru (ros::NodeHandle nodeHandle). Apoi vom crea o publicație cu numele temei "warning". În funcția setup vom anunța tema pe care vrem să publicăm. Funcția initNode va inițializa handle-ul de nod. În funcția loop publicația va publica mesajul.

Pentru testare, vom lansa trei terminale pe mașina noastră Ubuntu. În primul terminal vom lansa nucleul ROS, roscore (comanda roscore). Apoi vom rula aplicația client rosserial, care va transmite mesajele la celelalte noduri ROS. În final, vom afișa mesajele publicate pe tema aleasă cu comanda **rostopic echo warning**.

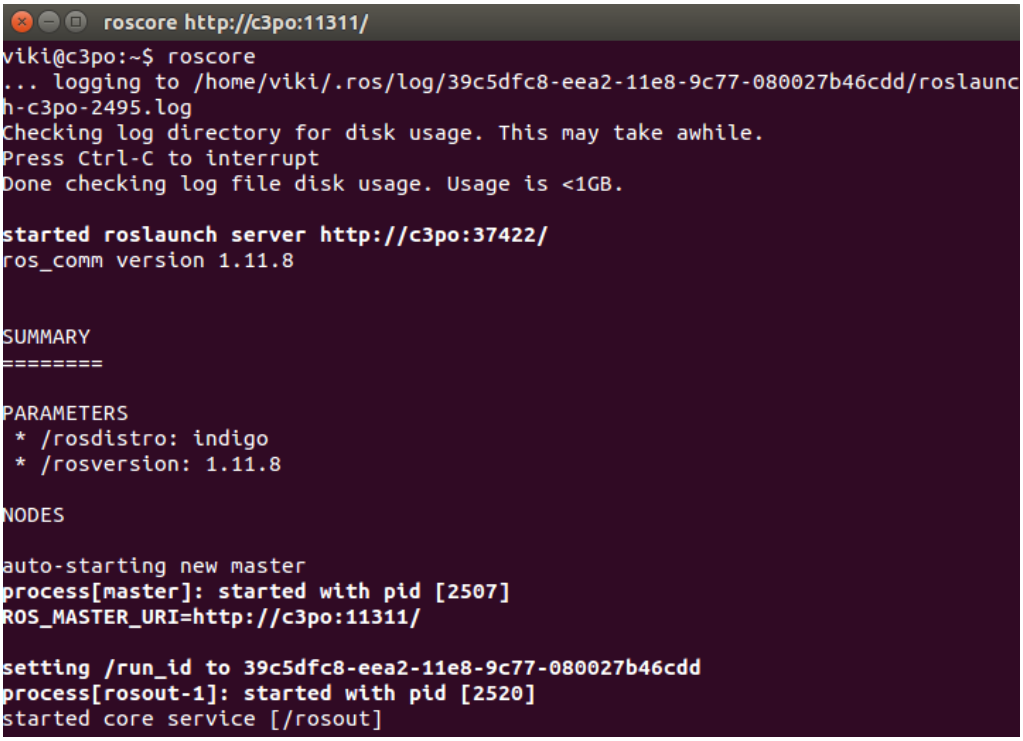


Fig.A2.6. Rularea comenzii roscore in terminalul linux

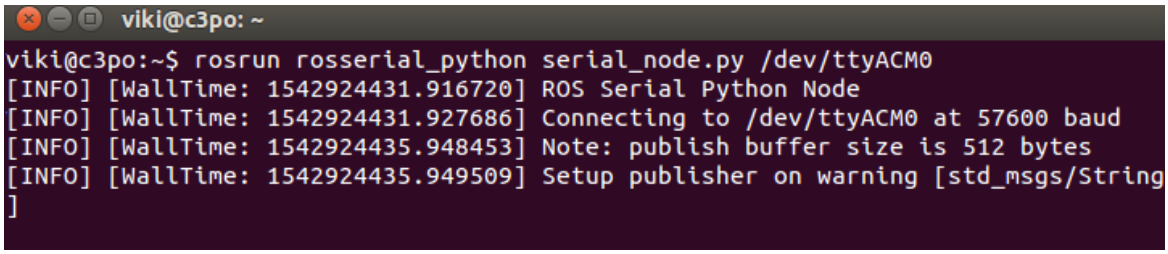


Fig.A2.7. Rularea unui client in ROS

```
viki@c3po:~$ rostopic echo warning
data: This is your last warning!
---
data: This is your last warning!
---
data: This is your last warning!
---
data: This is your last warning!
---
data: This is your last warning!
---
data: This is your last warning!
---
data: This is your last warning!
---
data: This is your last warning!
---
```

Fig.A2.8. Afisarea mesajului dintr-un topic

Exemplul 2: Abonare

În acest exemplu, care va exemplifica abonarea, vom scrie un program care va schimba starea unui led de fiecare dată când va primi un mesaj de la o publicație. Acest exemplu poate fi găsit pe site-ul web oficial Arduino ROS și în exemplele bibliotecii Arduino ros_lib.

```
#include <ros.h>
#include <std_msgs/Empty.h>

ros::NodeHandle nh;

// Vom crea o funcție callback pentru abonatul nostru. Această
//funcție primește ca
// argument o referință constantă la mesaj. În funcția
//callbackFunction, tipul mesajului
// este  std_msgs::Empty și numele mesajului este  toggle_msg.
void callbackFunction( const std_msgs::Empty& toggle_msg)
{
    digitalWrite(13, HIGH-digitalRead(13));
}

// Aici vom instanția un abonament (Subscriber) cu numele temei
//"interschimbare”,
// și tipul std_msgs::Empty. La subscriber, trebuie aplicat
//template-ul mesajului.
// Argumentele sunt tema la care se abonează și funcția callback
//utilizată
ros::Subscriber<std_msgs::Empty> sub(“interschimbare”,
&callbackFunction);

void setup()
{
    pinMode(13, OUTPUT);
    nh.initNode();
    // ne abonăm la temele pe care le vom asculta
```

```
    nh.subscribe(sub);
}

void loop()
{
    nh.spinOnce();
    delay(1);
}
```

Explicațiile pentru handle-ul nodului și pentru antetele incluse rămân valabile. P funcție callback este o funcție care este transmisă ca argument unei alte funcții și care va fi automat apelată când un eveniment este declanșat. Această funcție poate fi apelată de la un nivel software mai coborât. Această funcție callback trebuie să primească o referință constantă la un mesaj ca argument.

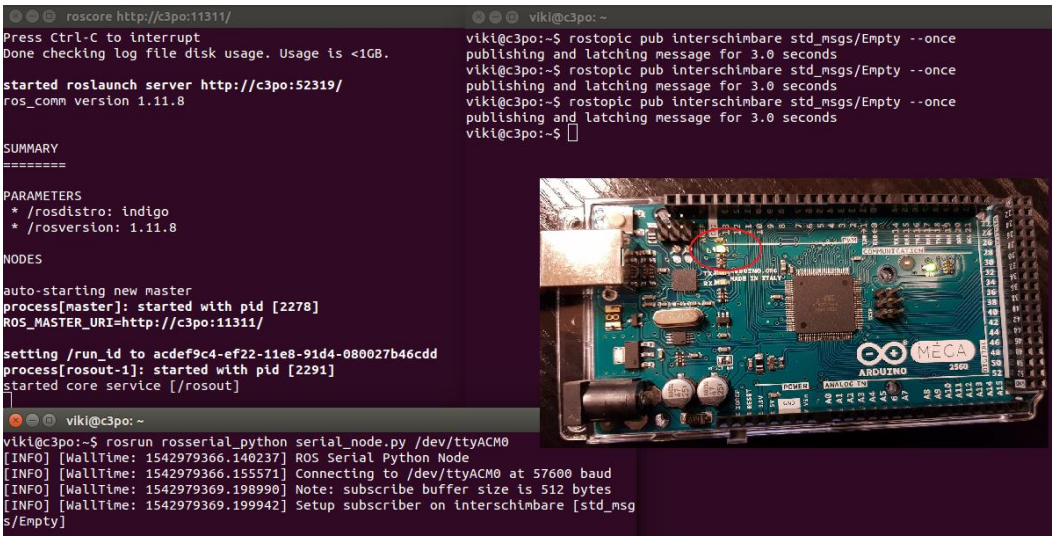


Fig.A2.9. Comenzile si rezulatul exemplului cu abonatul

Exemplul 3: Publicație IR

În acest ultim exemplu vom atașa un senzor de distanță Sharp în domeniul infraroșu la microcontrollerul nostru și vom vizualiza distanțele pe un grafic folosind rqt_plot. Pentru acest exemplu avem nevoie de un senzor Sharp IR, o placă Arduino și un cablu USB. Acesta este un exemplu preluat din ros_lib și modificat.

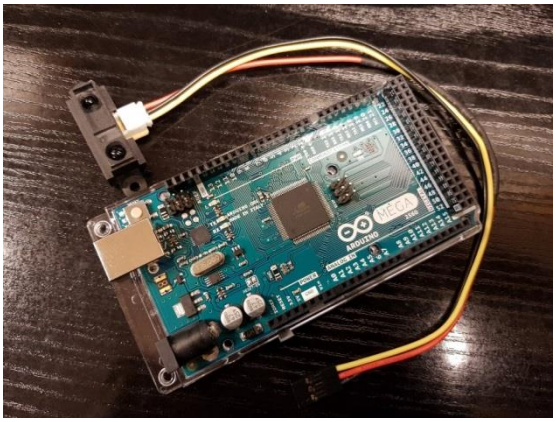


Fig.A2.10. Placa Arduino si senzorul de distanta infra rosu folosite in exemplul cu publicatia

```
// includere headere și inițializare noduri și mesaje
#include <ros.h>
#include <ros/time.h>
#include <sensor_msgs/Range.h>

ros::NodeHandle node;
sensor_msgs::Range msg;
ros::Publisher pub_range( “Sharp", &range_msg);
const int analog_pin = 0;
unsigned long range_timer;

//funcție care va citi distanța de pe senzorul Sharp conectat la
//un pin analogic pin_num
// și apoi o filtrează, făcând media pe mai multe măsurători
float getRange(int pin_num)
{
    int count = 10;
    int sum = 0;
    for (int i = 0; i<count; i++)
    {
        float volts = analogRead(pin_num) * ((float) 5 / 1024);
        float distance = 65 * pow(volts, -1.10);
        sum = sum + distance;
        delay(5);
    }
    sum = (int) (sum/count);
    return (sum -1)/100;
}
// variabila globală pentru stringul frame id. trebuie să fie
//globală, ca să fie disponibilă
// atâta timp cât mesajul este folosit
char frameid[] = "/ir_ranger";

void setup()
{
    // inițializare handle nod
    node.initNode();
    node.advertise(pub_range);
    // completăm câmpurile mesajului
    range_msg.radiation_type = sensor_msgs::Range::INFRARED;
    range_msg.header.frame_id = frameid;
    // aici sunt caracteristicile senzorului
    range_msg.field_of_view = 0.8;
    range_msg.min_range = 1;
    range_msg.max_range = 8;
}

void loop()
{
    // publicăm citirile la fiecare 50 ms
    if ( (millis()-range_timer) > 50)
    {
```

```
        range_msg.range = getRange(analog_pin);
        range_msg.header.stamp = node.now();
        pub_range.publish(&range_msg);
        range_timer = millis();
    }
    node.spinOnce();
}
```

Aplicația se rulează ca în exemplele anterioare:

1. roscore
2. rosrun rosserial_python serial_node.py _port:=/dev/ttyACM0

Vom afișa datele în format grafic folosind comanda rqt_plot:

rqt_plot Sharp/rang

Dacă nu este nici un obiect în fața senzorului, acesta va transmite valori fluctuante, iar dacă obiectul există vom observa valori continue în intervalul 0...1 (valorile se vor scala în acest interval). Variind distanța de la obiect la senzor, vom obține graficul de mai jos:

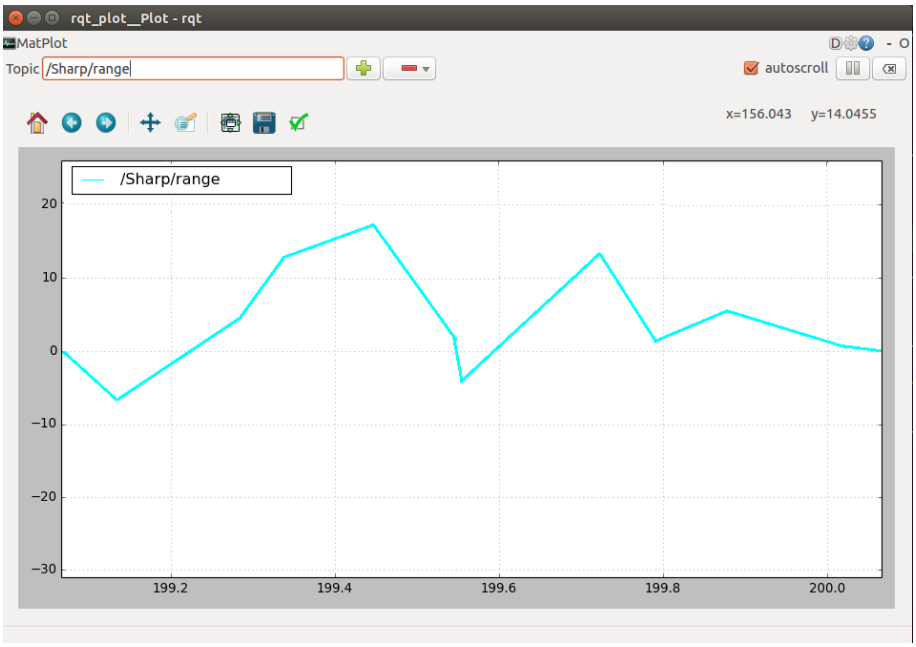


Fig.A2.11. Grafic desenat folosind informatiile venite de la sensorul infrarosu

Lucru individual

1. Testați și rulați exemplele date

2. Pentru mai multe exemple, accesați http://wiki.ros.org/ROS/Tutorials

3. Implementați un sistem de detecție a intrușilor procesând informația de la senzorul IR. Detectorul va fi activat și dezactivat dintr-un program publicație și va publica un mesaj de alarmă la detecția intrusului.

Bibliografie:

1. JasonM. O'Kane, "A Gentle Introduction to ROS"
2. Jonathan Bohren, "ROS Crash-Course (Part I) ", The Johns Hopkins University
3. Murilo Fernandes Martins, "PhD", Department of Electrical Engineering, FEI University Centre
4. Action and Perception (RAP) Group, "Robotics Operation System"**,** Universidad Veracruzana, Research Center on Artificial Intelligence LAAS-CNRSRobotics
5. "ROS 0-60A Comprehensive tutorial ofthe Robot Operating System"
6.http://wiki.ros.org/
7.https://nootrix.com/downloads/?fbclid=IwAR18YLs05vOQ5OG4DZcDIMK5oSDASWb1BtGlfhanjlKwICXBOaL3KGGt8p4#RosVM
8.http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup