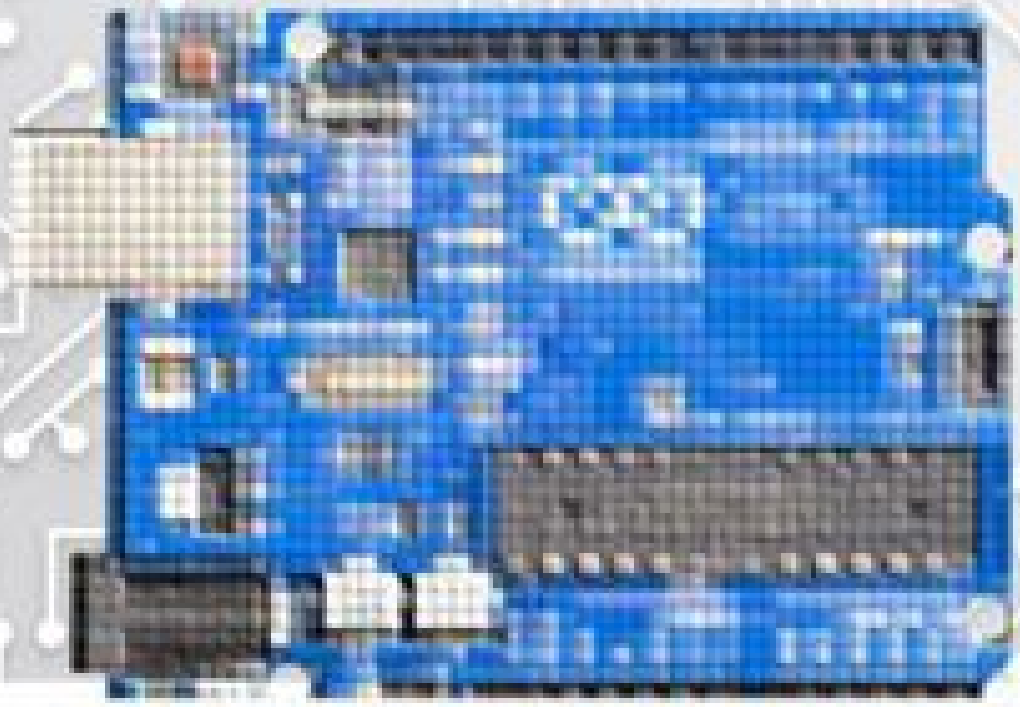
Sebastian Petru SABOU

ÎNDRUMĂTOR LABORATOR MICROCONTROLERE - ARDUINO



U.T.PRESS
CLUJ-NAPOCA,2018
ISBN 978-606-737-341-7

Sebastian Petru SABOU

Îndrumător laborator microcontrolere

ARDUINO

U.T. PRESS
CLUJ-NAPOCA, 2018
ISBN 978-606-737-341-7

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

ISBN 978-606-737-341-7

CUPRINS

Introducere - ARDUINO

Arduino este o companie open-source care produce atât partea de hardware (sistemele de dezvoltare) cât şi partea de software necesară compilării şi programării sistemelor de dezvoltare. Pe lângă componenta de design, producţie şi comercializare este inclusă şi o comunitate uriaşă care se ocupă cu crearea, realizarea, implementarea multor proiecte, de ajutor reciproc pentru depanarea hardware şi software a multor proiecte, de crearea unor componente (librării) cu funcţii suplimentare sau pentru diverse componente noi.

Platformele Arduino se bazează pe sisteme de dezvoltare cu microcontrolere (sunt suportate mai multe tipuri de microcontrolere), acestea sunt construite astfel încât să pună la dispoziţie utilizatorului pini (conectori) de intrare şi ieşire, care pot fi interfaţaţi cu o multitudine de senzori şi alte componente, fie prin conexiuni directe, fie prin intermediul unor module de extensie care poartă de numirea de shield-uri. Platformele asigura alimentarea, programarea şi pot asigura şi comunicaţia, prin intermediul interfeţei USB. Pot fi alimentate şi din surse de alimentare externe, majoritatea având conector de alimentare pentru acest scop.

Pentru programare şi dezvoltarea de programe este asigurat un mediu de dezvoltare integrat (IDE), bazat pe *Processing*, asigurând suport pentru limbajul C şi C++.

Arduino este şi o platformă, un sistem de dezvoltare, utilizat pentru a programa o serie de procesoare din familia Atmel. Se utilizează procesoare de tip SoC (System on Chip), specificaţiile fiecărei variante de platforme fiind disponibile la adresa https://www.arduino.cc/en/Products/Compare .

În cadrul laboratorului se vor utiliza platformele Arduino Uno. Acestea utilizează microcontrolerele Atmega328P, au o arhitectură pe 8 biţi, sunt alimentate la o tensiune de 5V şi o frecvenţă a oscilatorului de 16MHz, memorie SRAM de 2kB, EEPROM 1kB şi o memorie flash pentru programe de 32kB din care 0,5kB este utilizată de către bootloader (programul care asigură comunicaţia cu calculatorul, care are şi rolul de a realiza transferul şi programarea software-ului de la calculator). Ca şi interfaţă are 14 pini digitali (pini de intrare-ieşire, nivele logice, din care 6 pot fi ieşiri cu PWM) şi 6 pini analogici (pot fi utilizaţi ca şi intrări pentru convertorul analog-digital pe 10 biţi, încorporat). Alimentarea microcontrolerului se realizează la 5V dar sistemul de dezvoltare poate fi alimentat în intervalul maxim 6-20V, recomandat 7-12V (din experienţa practică nu se recomandă depaşirea unei tensiuni de 9V datorită disipării termice foarte reduse a stabilizatorului de tensiune).

Comunicaţia cu calculatorul se realizează prin intermediul interfeţei USB, se realizează un port serial virtual care permite programarea sistemului de dezvoltare dar şi comunicaţia serială a eventualelor programe cu calculatorul (care nu mai este condiţionat de existenţa unui port serial realizat cu circuite dedicate). Unele sisteme au implementat on-board această comunicaţie, altele necesită un circuit extern, pentru a asigura conversia USB-serial. Toate pot fi programate prin intermediul unui port ICSP dedicat, acest port se utilizează îndeosebi când din diferite motive este necesară reprogramarea programului de iniţializare bootloader.

Arduino are o platformă hardware open-source: referințele de design pentru Arduino sunt distribuite sub licența Creative Commons Attribution Share-Alike 2.5 și sunt disponibile pe situl Arduino. Schemele și fișierele de producție sunt și ele disponibile. Codul sursă pentru IDE este disponibil sub GNU General Public License, version 2.[3]

1. Mediul de lucru Arduino IDE

Pentru dezvoltarea programelor şi pentru programarea platformelor Arduino se utilizează mediul de programare Arduino IDE. Acesta se poate descărca de la adresa https://www.arduino.cc/en/Main/Software şi este distribuit cu titlu gratuit. Există variante pentru cele mai uzuale sisteme de operare: Windows, Mac OS şi Linux. O altă alternativă este crearea de programe utilizând platformele software care sunt online https://create.arduino.cc/ .

Presupunând că sistemele de operare care sunt, uzual, instalate pe calculatoarele din laborator sunt cele de tip Windows, în continuare se va face referire la utilizarea acestora.

După descărcarea şi instalarea mediului de porgramare Arduino IDE se rulează aplicaţia şi se obţine imaginea din figura 1.1.

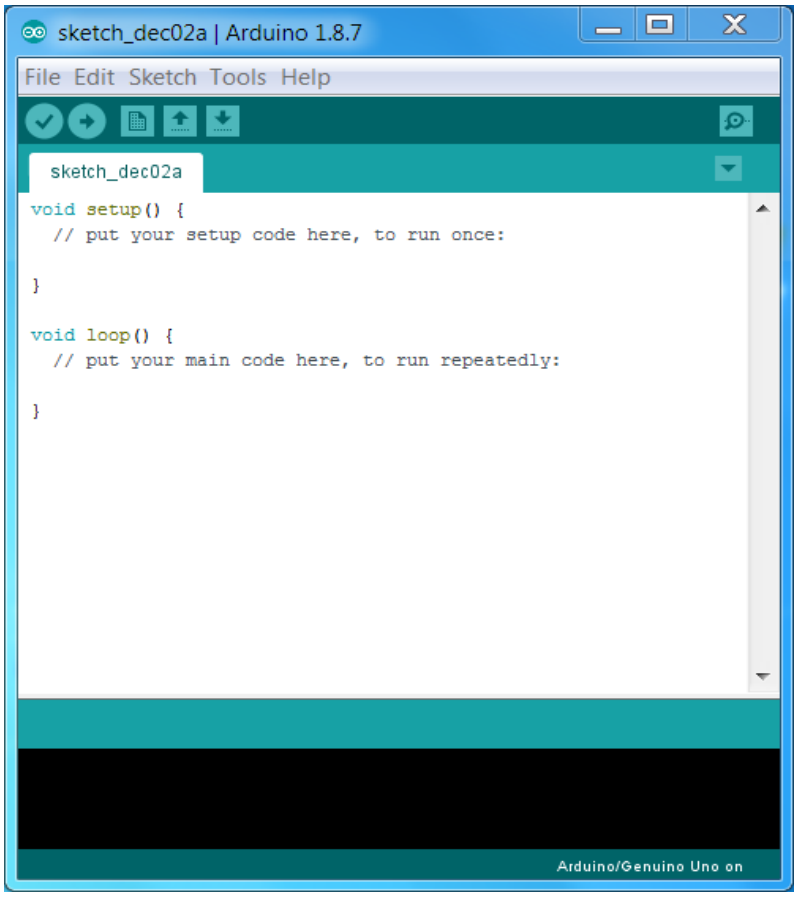


Fig. 1.1 Ecranul de start Arduino IDE

Arhitectura unui program pentru Arduino

Programele scrise pentru platformele Arduino poartă denumirea de „sketch” . Acestea sunt alcătuite din două funcţii importante:

```
void setup () {
// instrucţiunile de aici se execută o singură dată, la pornire
// se utilizează pentru setări şi iniţializări
}

void loop() {
// instrucţiunile de aici se execută în buclă, de la ultima instrucţiune se reia bucla
// cu prima instrucţiune din această funcţie, aici e programul principal
// se execută în mod repetat până la oprirea alimentării cu energie a sistemului
}
```

După cum au fost descrise pe scurt, prima funcţie este utilizată pentru iniţializarea variabilelor, a comunicaţiei seriale (de exemplu baud rate), pentru declararea pinilor ca fiind de ieşire sau de intrare, diverse funcţii pentru iniţializarea unor senzori, etc. Această funcţie este executată la pornirea (imediat după alimentarea sau după reset) sistemului de dezvoltare Arduino Uno, fiind rulată o singură dată. A doua funcţie (loop();), este funcţia similară cu „main()” din limbajul de programare „C”, este rulată în buclă, adică la sfârşitul execuţiei ultimei instrucţiuni din cadrul funcţiei este continuată cu execuţia primei instrucţiuni din această funcţie, totul fiind un ciclu care se repetă continuu.

Instalare Arduino IDE

Instalarea mediului de programare Arduino IDE începe prin descărcarea programului de la adresa https://www.arduino.cc/en/Main/Software a versiunii dorite. Sunt disponibile si versiuni mai vechi precum si versiuni care necesită instalarea sau doar dezarhivarea şi rularea programului. Aceasta ultimă variantă se utilizează în situaţiile în care se utilizează sistemul de operare Windows la care nu se cunoaşte sau nu se doreşte instalarea programului (sau utilizatorul nu are drepturile necesare instalării).

Mediul de programare Arduino IDE poate fi instalat pe cele mai uzuale sisteme de operare: Windows, Linux, Mac OS. În continuare se va presupune instalarea sau utilizarea lui folosind sistemul de operare Windows.

În momentul descărcării programului este posibilă şi donarea unei sume modice, necesare pentru finanţarea acestui proiect, donaţia nu este o condiţie obligatorie, se poate descărca programul şi fără acest act de caritate.

După instalare sau după prima rulare (în cazul variantei care nu necesită instalare), programul va crea în folderul *Documents* un folder *Arduino* unde va salva bibliotecille de funcţii care vor fi instalate ulterior precum şi programele nou create vor fi salvate acolo în mod implicit (dacă nu se specifică altă cale).

La rularea Arduino IDE va apărea ecranul de start din figura 1.1. Dupa lansarea în execuţie a programului va fi necesară configurarea lui. De menţionat că aceasta configurare, a portului de comunicaţie, va trebui repetată de fiecare dată când se schimbă portul USB din calculator la care este conectată platforma Arduino, de fiecare dată când se va utiliza alt sistem Arduino şi la fiecare pornire a calculatorului. Este

necesar acest lucru deoarece, teoretic, la fiecare conectare a Arduino la calculator, sistemul de operare va atribui acestei conexiuni virtuale seriale un nume, acest nume poate să difere dacă se schimba sistemul sau dacă trece un anumit timp în care Arduino nu a fost conectat la calculator.

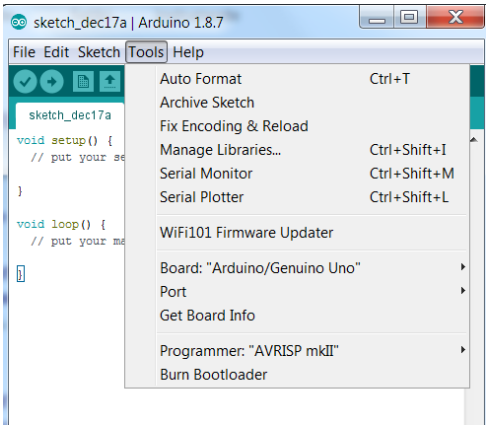


Figura 1.2 Configurare Arduino IDE

În figura 1.2 se poate observa meniul în care se pot face setările menţionate anterior, în meniul *Board* se alege platforma Arduino pentru care se va realiza programul sau care va fi programat iar în meniul *Port* se alege portul serial prin

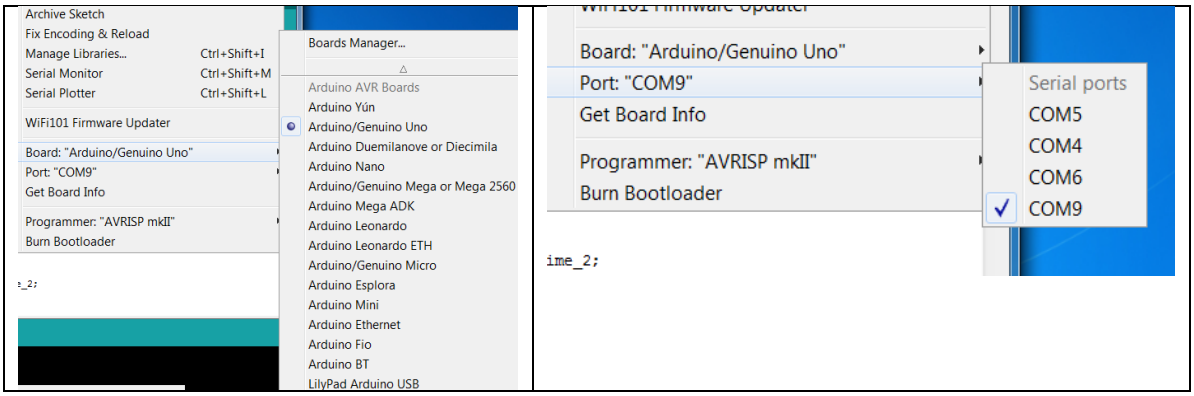


Figura 1.3 Configurarea tipului de Arduino precum şi a portului de comunicaţie

intermediul căruia calculatorul va comunica cu platforma (Figura 1.3).

După finalizarea acestor configurări, se poate trece la următoarea etapă, ce de scriere a programului pentru microcontroler. Pentru început este recomandată utilizarea exemplelor, pentru a descoperi fiecare facilitată oferită de către Arduino, exemplele se afla în meniul *File – Examples*, fiind grupate pe categorii.

Este de reţinut faptul ca fiecare bibliotecă de funcţii care este adăugată mediului de programare, în mod uzual, este însoţită şi de programe exemple care să faciliteze înţelegerea modului de lucru cu respectivele funcţii.

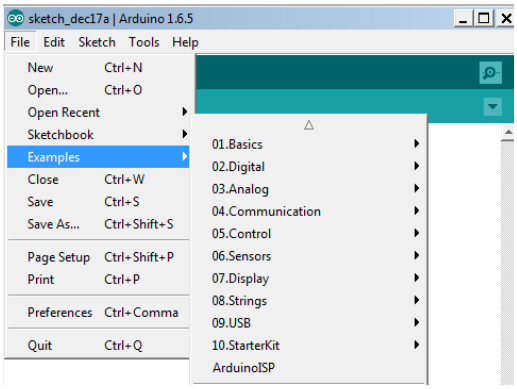


Figura 1.4 Exemple de programe din Arduino IDE

Se pot deschide, pentru început, programe din *01. Basics*, pentru a verifica compilarea şi programarea platformei Arduino Uno. Unul din primele programe care se testează, furnizând şi un feedback vizual, este *Blink*. După deschiderea fişierului urmează compilarea şi programarea. Acestea se pot realiza apelând la meniuri sau prin intermediul a două butoane, prezentate în figura 1.5

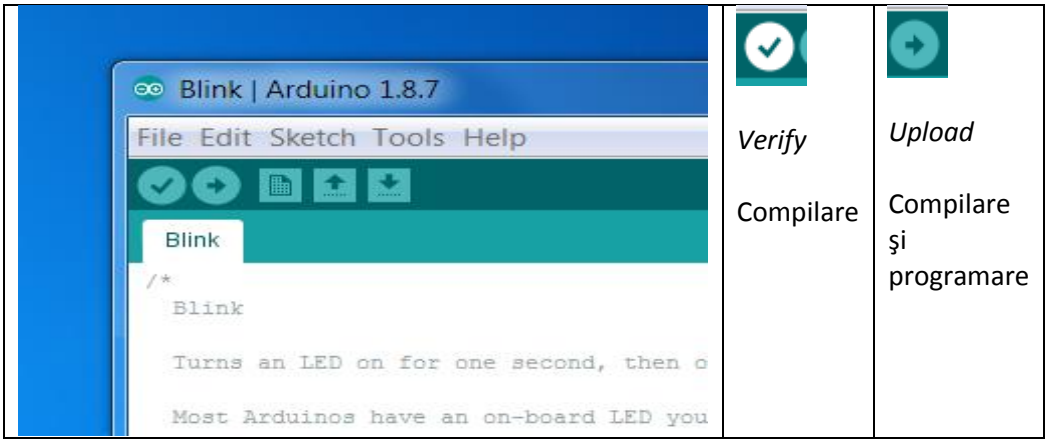


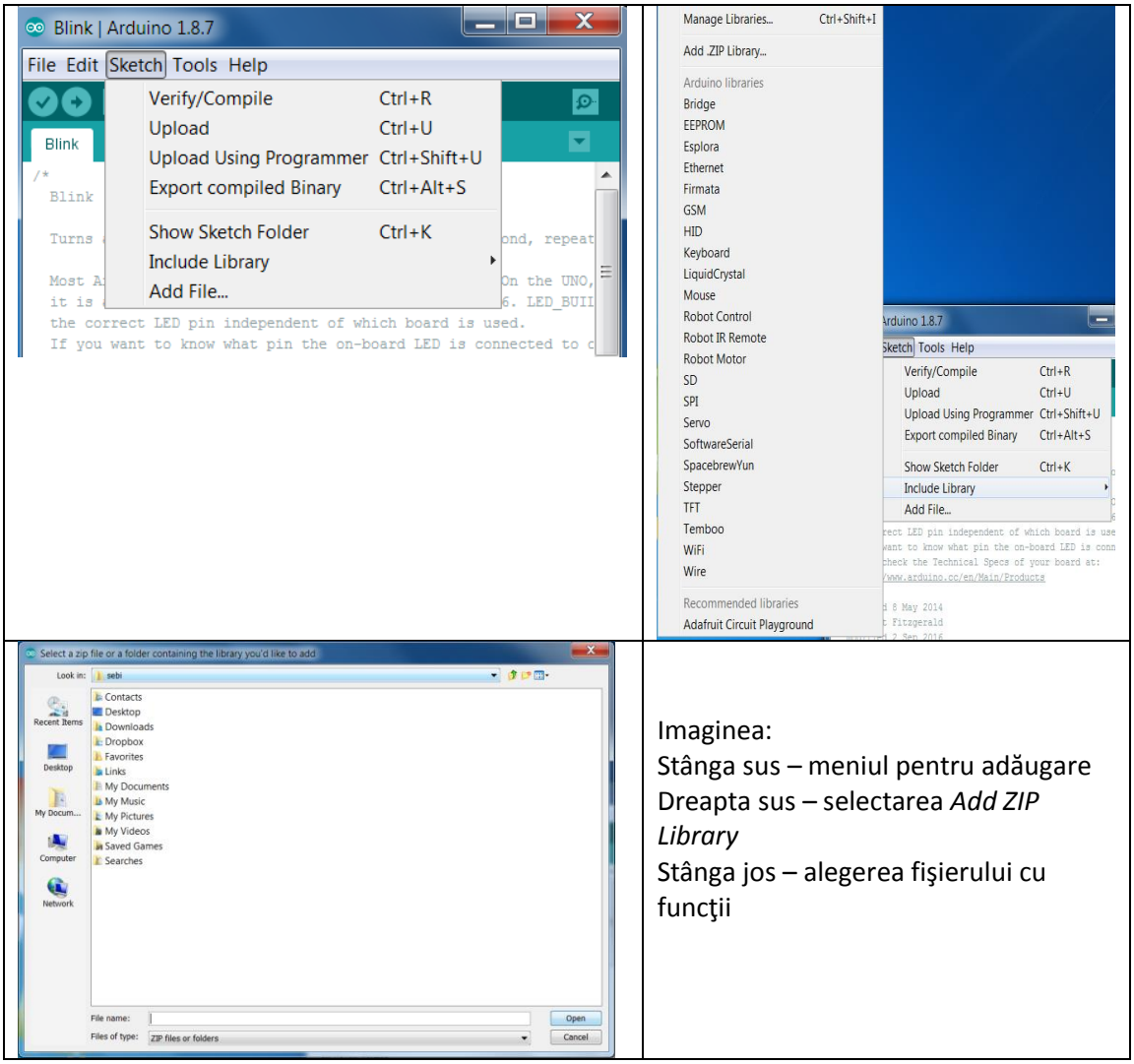
Figura 1.5 Localizarea şi semnificaţia butoanelor de compilare şi programare

Diferenţa dintre cele două butoane (sau opţiuni) este aceea că butonul de *Verify* realizează doar o compilare a programului pentru a descoperi dacă există erori de sintaxă sau de utilizare a unor funcţii, pe când butonul de *Upload* realizează o compilare şi, în caz de succes, urmată de o programare a platformei cu programul rezultat.

O facilitate utilizată la fiecare utilizare a unei componente noi sau a unui modul nou, este adăugarea de biblioteci (sau librării) care conţin funcţii care permit conectarea, accesul la date, comenzi sau alte comenzi care permit utilizarea noilor componente. Astfel, se realizează un acces la nivel mai înalt la aceste componente, putându-le utiliza fără a cunoaşte foarte detaliat modul de comunicare sau de transmitere a datelor, lista de comenzi care pot fi executate de către aceste componente, etc.

Adăugarea acestor biblioteci de funcţii se realizează doar o singură dată, acestea rămânând salvate în folderul *Documents/Arduino*. De asemenea e de menţionat faptul că aceste biblioteci de funcţii conţin (în marea lor majoritate) şi exemple de utilizare a acestor funcţii. Aceste exemple vor fi listate în meniul *File-Examples-Numele bibliotecii*.

Adăugarea de noi biblioteci se realizează urmând paşii descrişi în figura 1.6.



Imaginea:
Stânga sus – meniul pentru adăugare
Dreapta sus – selectarea *Add ZIP Library*
Stânga jos – alegerea fişierului cu funcţii

Figura 1.6 Selecţia meniurilor şi opţiunile necesare includerii de noi biblioteci

Configuraţi Arduino IDE pentru a permite conectarea la platforma Arduino Uno, din cadrul laboratorului, selectaţi programul *Blink* din exemple, compilaţi şi pe urmă compilaţi şi programaţi Arduino Uno. Care este rezutatul ? (din punct de vedere vizual)

Adăugaţi o bibliotecă, deschideţi exemplul (sau unul din exemple) care este conţinut de către biblioteca adaugată.

2. Intrări – Ieşiri digitale

Informaţii teoretice preluate din sursa [2]

Pinii microcontrolerului Atmega328 pot fi configuraţi ca şi intrări sau ieşiri digitale, adică cu nivele logice. Chiar şi pinii care sunt consideraţi ca fiind analogici pot fi utilizaţi ca şi intrari sau ieşiri cu nivele logice.

Înainte de a fi utilizaţi, în cadrul unui program, este necesară configurarea pinilor ca fiind de intrare sau de ieşire. În cazul platformei Arduino, implicit (la pornirea sistemului) pinii digitali sunt configuraţi ca fiind de intrare, adică sunt în starea de înaltă impedanţă. Este de menţionat faptul că daca se utilizează pinii astfel configuraţi şi nu au conectate alte componente care să le stabilească un nivel logic, există posibilitatea ca nivelele logice ale pinilor să fie influenţate de către mediul ambiant şi să generez modificări ale nivelelor logice „citite", rezistenţa lor echivalentă de intrare fiind de ordinul 100 Mohm. Prin urmare este necesară stabilirea unui nivel logic stabil prin intermediul unor componente electronice externe, pentru a fi siguri că nu se obţin valori influenţate de către mediu. Dacă pinii respectivi nu sunt utilizaţi în cadrul programului ca fiind intrări (adică nu sunt „citiţi" de către program) atunci nu e necesară conectarea altor componente la acei pini.

La iniţializare toţi pinii sunt in starea de înaltă impedanţă şi din motive de consum de energie electrică, având o rezistenţă electrică echivalentă de valoare mare practic curentul prin circuitul electric al respectivului pin este de valoare foarte mică şi implicit rezultă un consum redus de energie electrică.

Atunci când sunt utilizaţi ca şi pini de intrare este recomandată utilizarea unor rezistenţe externe, de valori de ordinul 10k, conectate la VCC sau la GND („pull-up" sau „pull-down").

Microcontrolerul Atmega328 din componenţa platformei Arduino Uno are integrate rezistenţe electrice de „pull-up", care pot fi activate utilizând parametrul INPUT_PULLUP în cadrul instrucţiunii de setare a tipului de pin (de intrare sau de ieşire). Activarea acestor rezistenţe interne de pull-up are ca efect, oarecum, inversarea comportamentului pinilor de intrare, în sensul în care dacă nu este activ senzorul de la intrare (de exemplu un comutator) se va citi valoarea HIGH iar când senzorul este activat se va citi valoarea LOW. Uzual valoarea rezistenţelor interne de pull-up este de ordinul 20k-50k dar diferă şi funcţie de tipul microcontrolerului. Conectând un led la GND şi la un pin digital (având activă rezistenţa de pull-up) este foarte probabil ca ledul să lumineze suficient cât să fie vizibil.

În cazul în care se activează rezistenţele de pull-up, senzorii sau comutatoarele care se conectează la intrare ar trebui să fie active (să comute la activarea lor) pe starea logică de LOW.

Un alt efect al utilizării rezistenţelor interne de pull-up, datorită faptului că registrele care configurează activarea lor sunt aceleaşi care controlează şi starea logică a ieşirilor pinilor, când sunt configuraţi ca ieşiri, având activate rezistenţele interne de pull-up în momentul în care se configurează pinul ca fiind de ieşire, starea pinului va fi HIGH ! Acest comportament este valabil şi în sens invers, având setat pinul de ieşire pe nivelul HIGH şi comutând respectivul pin ca fiind de intrare se va activa automat rezistenţa internă de pull-up !

Utilizarea pinilor ca ieşire digitală implică utilizarea parametrului OUTPUT în instrucţiunea pinMode(). Pinul configurat ca fiind de ieşire poate furniza curent de maxim 40mA, fie în starea LOW fie in starea HIGH. Ca urmare pot alimenta sau comanda diverse circuite electrice pentru care este suficientă valoarea maximă

suportata de circuitul intern al pinului respectiv. Uzual se pot comanda direct leduri (cu utilizarea unei rezistente de limitare a curentului) dar nu se pot comanda direct relee sau motoare care necesită curenţi mai mari pentru funcţionarea lor.

Scurtcircuitarea pinilor între ei sau conectarea la GND sau VCC poate duce inclusiv la distrugerea integratului deşi de obicei are ca efect distrugerea circuitului intern corespunzător acelui pin.

Unul din cele mai simple programe utilizate pentru iniţierea utilizării mediului de programare Arduino IDE precum şi pentru învăţarea programării platformei Arduino Uno este programul “Blink”, are ca efect comanda unui LED realizând efectul de clipire.

Schema electrică este prezentată în figura 2.1.

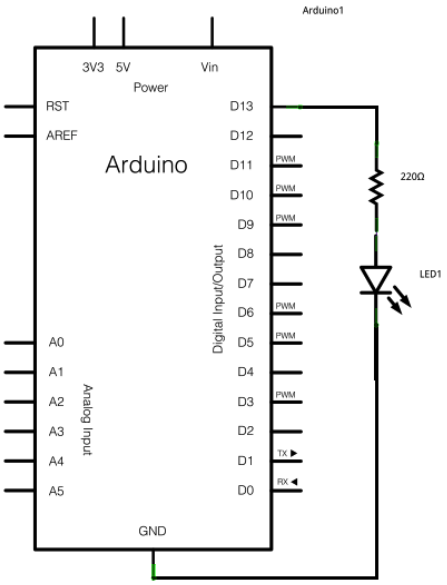


Figura 2.1 Schema electrică de conectare a LED-ului „onboard” [2]

Este utilizată o diodă LED care este gata lipită pe cablajul platformei Arduino Uno, fiind conectată la pinul D13, prin intermediul unei rezistenţe de 220 ohmi. Un nivel logic de 5V la pinul D13 are ca efect aprinderea LED-ului.

Programul utilizat se găseşte în Arduino IDE la File/Examples/01.Basics/Blink .

3. Modulația în durata a impulsurilor - PWM

Informaţii preluate şi adaptate de la [20]

Termenul de PWM vine din limba engleza de la Pulse Witdh Modulation si inseamna ca avem un semnal modulat în lăţimea impulsurilor de comandă. Ca sa fim mai expliciţi putem spune că un astfel de semnal PWM constă în codarea informaţiei în lăţimea impulsului obţinut. Factorul de umplere al unui semnal PWM se calculează cu relaţia $D(fu)=T_i/T$, unde T_i este durata impulsului şi T perioada semnalului.

Deci, putem astfel să observăm că fiecare procent al unui astfel de semnal reprezintă o valoare importantă în aplicaţia pe care o dorim să o implementăm cu un astfel de semnal spre deosebire de semnalul TTL care poate sa aiba doar două stări. (high, low).

Semnalele PWM sunt semnale de comandă a unor tranzistoare de putere, folosite în cadrul unor convertoare în comutatie.

Un modulator PWM are rolul de a comanda un comutator şi este o parte importantă şi complexă a unui regulator de tensiune în comuţatie.

Principiul de realizare a unui astfel de modulator PWM constă în alcătuirea unei scheme electronice care să conţină:

- generator în dinte de fierastrău
- amplificator de eroare
- comparator

O diagramă simplificată pentru realizarea semnalului PWM este prezentată în figura următoare.

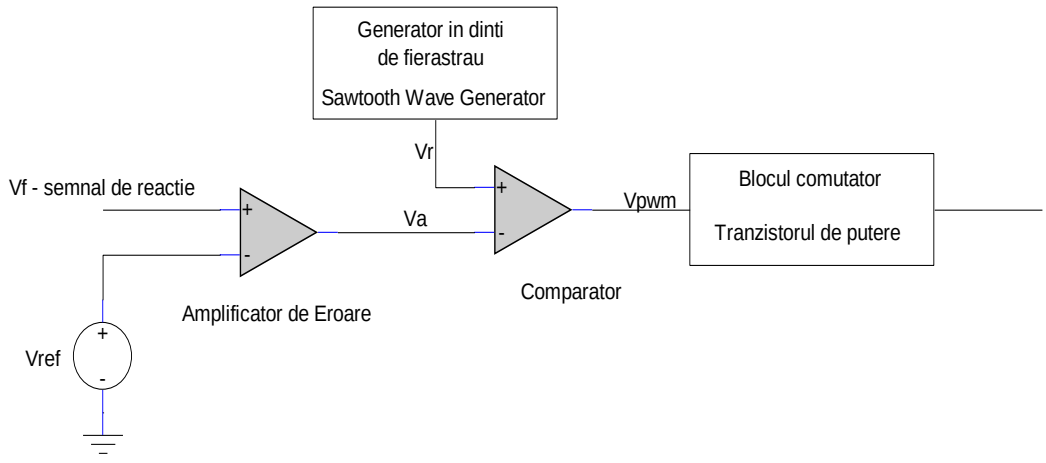


Figura 3.1 Circuit simplificat pentru realizarea semnalului PWM

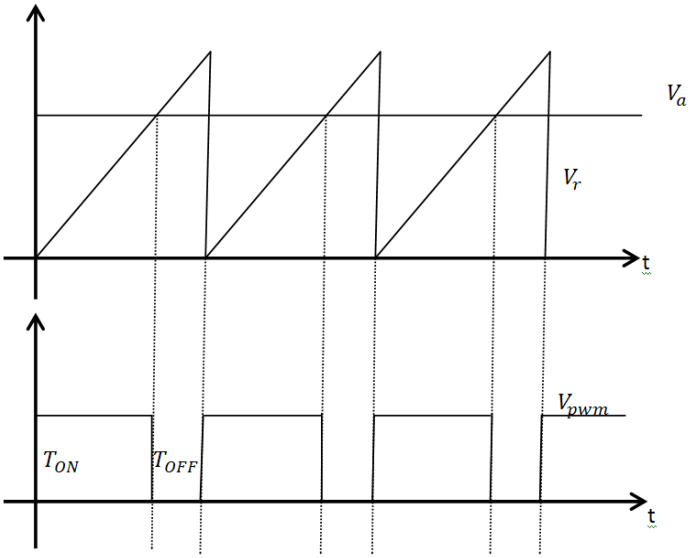


Figura 3.2 Formele de undă pentru circuitul 3.1

Dupa cum se poate observa si in figura 3.1 acest modulator PWM constă dintr-un generator în dinte de fierastrău (saw-tooth generator), un amplificator de eroare şi un comparator.

Frecvenţa generatorului este setata de valoarea constantei de timp RC.

Amplificatorul de eroare compară tensiunea de referinţă şi semnalul de reacţie. Semnalul de reacţie este obtinutţ de obicei, printr-o divizare a tensiunii de ieşire. Dacă consideram ca V_f este semnalul de reactie si V_{Ref} este tensiunea de referinta si $V_f = \beta V_0$, deoarece $V_f = V_{Ref}$, $V_0 = V_{Ref}/\beta$.

Tensiunea de la iesirea amplificatorului de eroare este comparata cu valoarea semnalului in dinte de fierastrau. Daca iesirea acestuia este mai mare decat valoarea dintelui de fierastrau atunci la iesirea comparatorului vom avea ‘1’ logic adica T_{on}. Daca iesirea amplificatorului este mai mica decat valoarea dintelui de fierastrau atunci la iesirea comparatorului vom avea ‘0’ logic adica $T_{off.}$

Daca tensiunea de iesire tinde sa creasca, atunci tensiunea de reactie va creste peste tensiunea de referinta, astfel tensiunea de iesire a amplificatorului de eroare va scadea rezultand astfel o durata mai mica pentru care la iesirea comparatorului vom acea ‘1’ logic. Daca tensiunea de iesire scade atunci la iesire comparatorului vom avea o durata mai mare de ‘1’ logic. Aceasta modificare a latimi impulsurilor in functie de tensiunea de iesire este datorata factorului de umplere (duty-cicle).

În cazul în care tensiunea de ieşire este constantă aceasta este menţinută de reacţia negativa la valoarea dorita.

Altfel descris, PWM (**P**ulse **W**idth **M**odulation) este o tehnică folosită pentru a varia în mod controlat tensiunea dată unui dispozitiv electronic. Această metodă schimbă foarte rapid tensiunea oferită dispozitivului respectiv din ON în OFF și invers (treceri rapide din HIGH (5V de exemplu) in LOW (0V). Perioada de timp corespunzătoare valorii ON dintr-un ciclu ON-OFF se numește factor de umplere (*duty cycle*) și reprezintă, în medie, ce tensiune va primi dispozitivul electronic. Astfel, se pot controla circuite analogice din domeniul digital. Practic, asta înseamnă că un LED acționat astfel se va putea aprinde / stinge gradual, iar în cazul unui motor acesta se va învârti mai repede sau mai încet.

În cazul unui motor, căruia i se aplică un semnal PWM cu factor de umplere de 0%, viteza de rotație a acestuia va fi egală cu 0rpm. Un factor de umplere de 100% va duce la o turație maximă a acestuia.

1. Principiul de funcționare

Factorul de umplere se exprimă în procente și reprezintă cât la sută din perioada unui semnal acesta va fi pe nivelul ON. În Figura 1 se pot observa semnale PWM cu factori de umplere diferiți. Astfel, se poate deduce foarte ușor formula pentru a obține valoarea factorului de umplere (D):

D = t_on / (t_on + t_off) * 100 = pulse_width / period * 100

Astfel, tensiunea medie care ajunge la dispozitiv este dată de relația: D * Vcc.

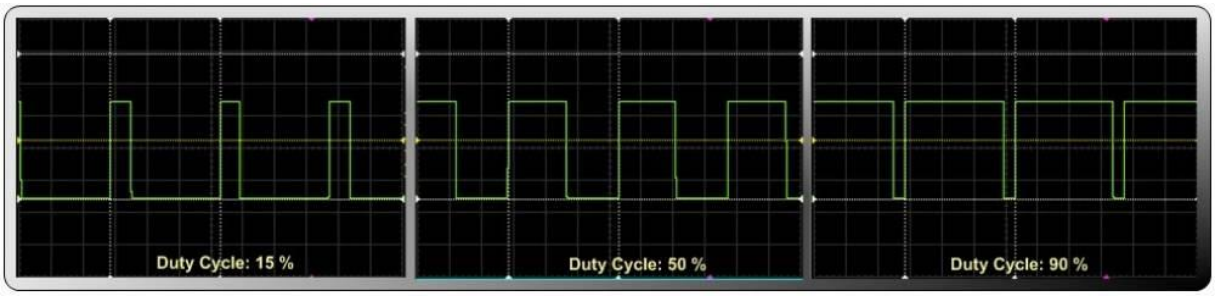


Figura 3.3 Semnal PWM cu diferiți factori de umplere

Modularea folosește variația factorului de umplere a unei forme de undă dreptunghiulară pentru a genera la ieșire o tensiune analogică. Considerând o formă de undă dreptunghiulară *f(t)* cu o valoare minimă *ymin* și o valoare maximă *ymax* și factorul de umplere *D* (ca în figura figure 1) valoarea medie a formei de undă e dată de relația:

$$\bar{y} = \frac{1}{T}\int_0^T f(t)dt$$

cum *f(t)* este o formă de undă dreptunghiulară valoarea sa maximă se atinge pentru 0<t<D*T.

Multe circuite digitale pot genera semnale PWM. Majoritatea microcontrolerelor oferă această facilitate, pe care o implementează folosind un numărător care este incrementat periodic (conectat direct sau indirect la o unitate de ceas) și care este resetat la sfârșitul fiecărei perioade a PWM-ului. Când valoarea numărătorului este

mai mare decât valoarea de referință, ieșirea PWM (output-ul) trece din starea HIGH în starea LOW (sau invers).

În cazul microcontrolerului ATmega328, semnalele de comandă PWM sunt generate cu ajutorul timerelor sau pot fi generate doar prin intermediul unui program, în acest ultim caz frecvenţa semnalului fiind mai mică (datorită modului de generare).

În figura 3.4 este prezentată schema electrică de conectare a unui motor de curent continuu prin intermediul căreia se poate comanda motorul în două regimuri, în primull caz se poate comanda doar în modul pornit-oprit (on, off), fără să se poată modifica turaţia acestuia, iar în al doilea caz se poate comanda astfel încât turaţia motorului să poată fi modificată cu un număr de 256 de trepte. Factorul de umplere al semnalului PWM fiind definit prin intermediul unui registru pe 8 biţi se poate deduce că numărul maxim de trepte care poate fi generat este de 256. În cazul schemei din figura 3.4 este prevăzut un buton de pornire, buton conectat la pinul D2, care atunci când este apăsat va aduce pinul D2 la nivelul logic high, urmând ca în cuprinsul programului să fie citit şi interpretat acest nivel logic de la intrarea D2.

Un exemplu de program generic care generează semnal PWM este următorul:

```
int PWM_out_pin = 9; // trebuie să fie unul din pinii 3, 5, 6, 9, 10, or 11in cazul Arduino Uno
void setup() {
        pinMode(PWM_out_pin, OUTPUT); //pin de iesire PWM
}
void loop() {
        byte PWM_out_level; //variabila care va reprezenta factorul de umplere
        PWM_out_level = ... // calculul factorului de umplere – se poate calcula functie de algoritmul
dorit
        analogWrite( PWM_out_pin, PWM_out_level); //se comanda iesire, generarea semnalului
PWM
}
```

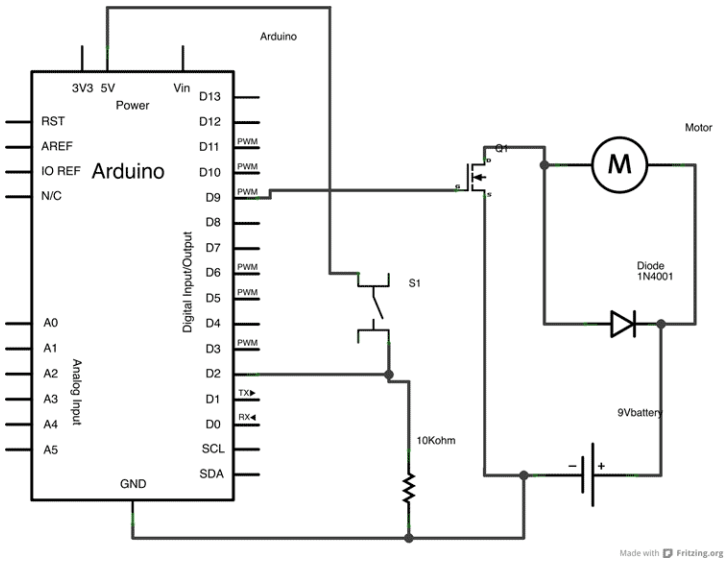


Figura 3.4 Schema de comandă PWM a unui motor de curent continuu

Să se scrie programul care să genereze semnalul PWM conform schemei de conectare din figura 3.4, pe baza codului generic prezentat anterior.

Să se modifice programul astfel încât să se ia în calcul şi butonul S1.

Semnalul PWM poate fi utilizat şi în cazul în care se doreşte modificarea turaţiei unui motor dar şi posibilitatea schimbării sensului de rotaţie a acestuia. În acest scop se utilizează circuitul din figura 3.6 care conţine circuitul integrat L293D, acesta fiind o punte H integrată.

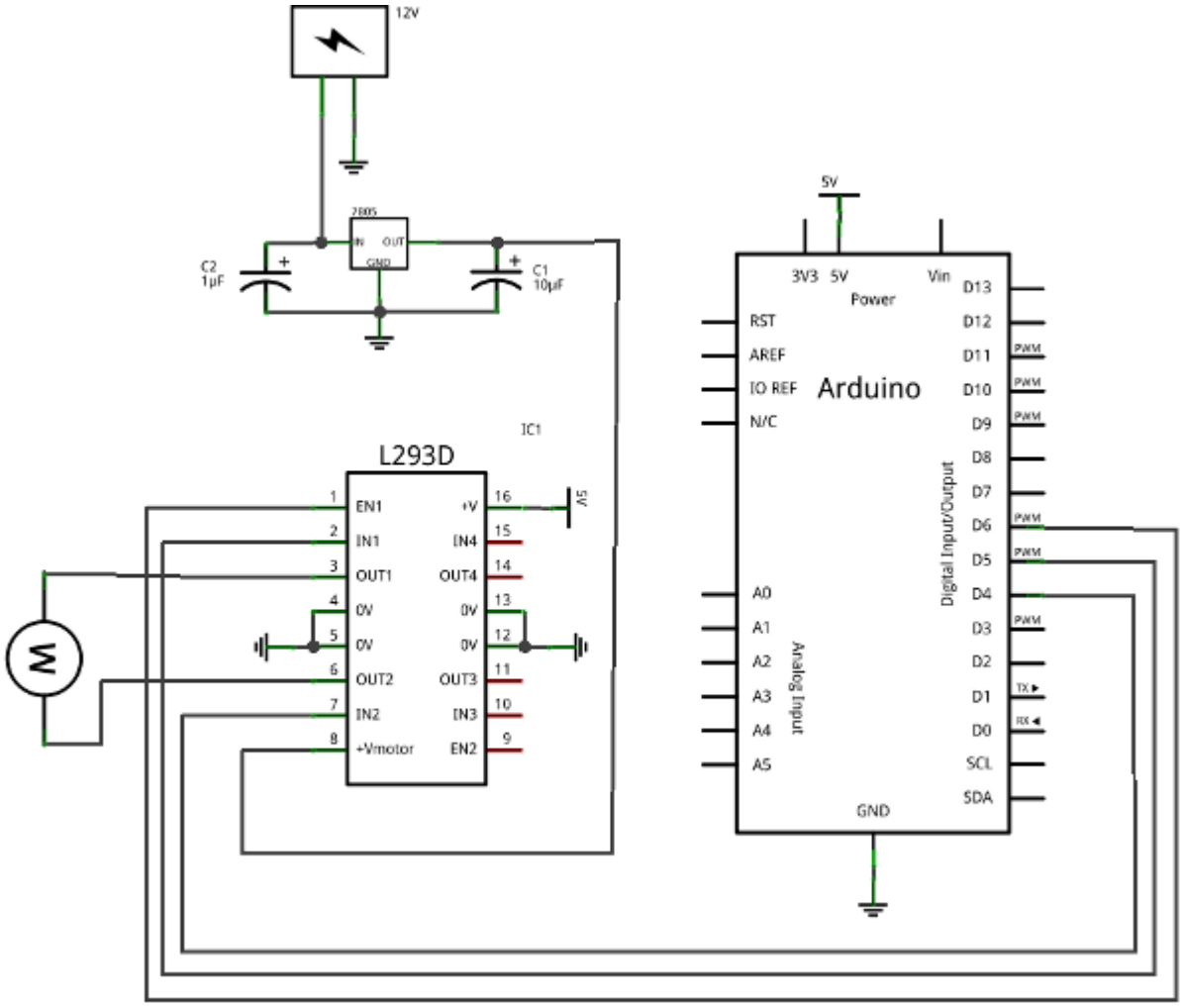


Figura 3.6 Schema electrică de conectare a unui motor cc în punte H

Exemplu de program utilizat pentru circuitul de la figura 3.6.

```
//functie pentru comanda motorului
void motor(int pornit)
{
if(pornit==1) {//verifica daca motorul trebuie pornit
if(directie==0)//verifica in ce directie trebuie rotit motorul
        {
        analogWrite(VIT, 0);//opreste motorul
        }}
//seteaza rotire la stanga
digitalWrite(DIRA,LOW);
```

```
digitalWrite(DIRB,HIGH);
//stabileste noua viteza
analogWrite(VIT, viteza);

}

else//daca directia setata este dreapta
{
analogWrite(VIT, 0);
}
//roteste motorul la dreapta
digitalWrite(DIRA,HIGH);
digitalWrite(DIRB,LOW);
analogWrite(VIT, viteza); //seteaza viteza noua
}
}
else //daca comanda este de oprire
{ analogWrite(VIT, 0);//opreste motorul
digitalWrite(DIRA,LOW);
digitalWrite(DIRB,LOW);}
}
```

4. Comunicaţia serială - USART

Sistemul de dezvoltare Arduino Uno permite realizarea comunicaţiilor seriale având implementate circuite periferice pentru următoarele tipuri de comunicaţii:

- comunicaţie serială USART
- comunicaţie serială I2C (TWI)
- comunicaţie serială SPI

Aceste tipuri se pot extinde utilizând prin utilizarea de programe adecvate, astfel se pot utiliza şi alte medii de comunicare (radio, ethernet, infraroşu…) cu diverse variante de transmitere a datelor (serial, paralel), având implementate sau nu protocoale de detecţie şi (sau) corecţie a erorilor.

În acest capitol va fi abordată comunicaţia serială utilizând USART. Aceasta se realizează din punct de vedere al intrărilor-ieşirilor prin utilizarea pinilor RX şi TX. Pentru a se asigura comunicaţia între două echipamente este necesară conectarea lor conform figurii 4.1

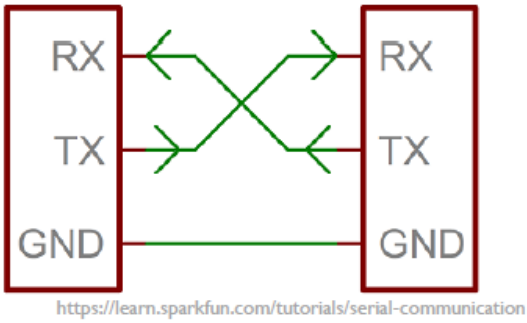


Figura 4.1 Conectarea pinilor USART

Prin pinul TX se transmit octeţi, prin RX se primesc octeţi. Se poate observa că este necesară conectarea TX la RX, respectiv RX la TX, respectiv fiecare pin de iesire la pinul de intrare al celuilalt sistem.

Comunicarea serială USART este realizabilă doar între două echipamente (este bidirecțională), nu sunt asigurate condiţiile din punct de vedere electric, de utilizare a mai multor echipamente conectate la aceeaşi magistrală serială.

În figura 4.2 se poate vedea schema electrică de conectare a doua sisteme Arduino Uno care pot să comunice între ele.

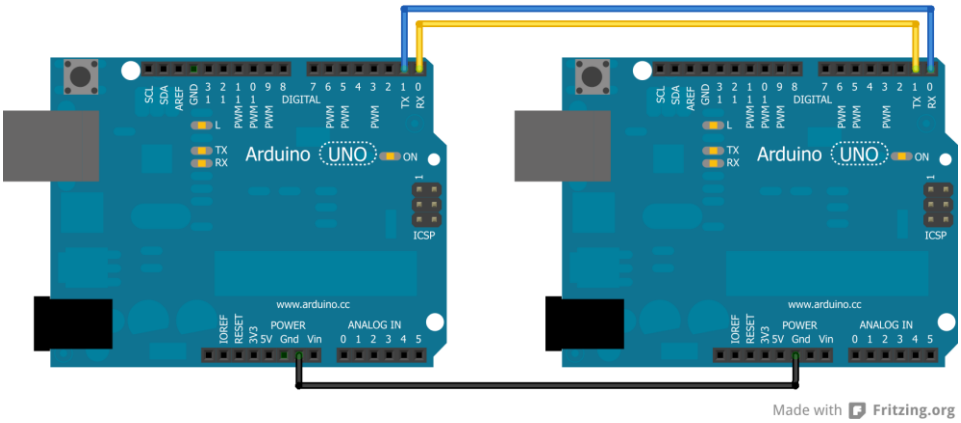


Figura 4.2 Conectarea a două sisteme Arduino Uno

Nivelele logice utilizate sunt cele TTL (în cazul Arduino Uno, prezentat mai sus) iar distanța maximă (lungimea conectorilor) este de ordinul zecilor de centimetri. Este de menționat faptul că pinii RX și TX (pinii 0 și 1) sunt utilizați și pentru comunicația prin intermediul portului USB, respectiv sunt utilizați și pentru programarea sistemului de dezvoltare. Acest fapt impune o atenție sporită la utilizarea acestor porturi, includerea acestora în circuite care pot menține sau modifica nivelul de tensiune poate duce la imposibilitatea programării platformei Arduino Uno. De asemenea, în cazul circuitului din figura 4.2 de exemplu, dacă unul din sisteme este programat deja și încearcă să transmită pe portul serial date și se dorește programarea celuilalt sistem Arduino, suprapunerea celor două sisteme de comunicație simultană (programarea și respectiv comunicația de la un sistem) poate duce la erori în cadrul procesului de programare.

Această interfață este utilizată și pentru programarea sistemului de dezvoltare, având preprogramat un mic program, care poartă denumirea de bootloader, care asigură comunicația cu calculatorul (cu Arduino IDE) și prin intermediul căruia se pot încărca programele pe platforma Arduino IDE.

Parametri de comunicație cei mai importanți sunt:
- Formatul datelor pe 8 biți, număr biți date și bit de paritate
- Rata de transfer (baud rate): 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, sau 115200
-

Funcții cele mai utilizate pentru comunicația serială USART, sunt:

-*Serial.begin(viteza)* -are rolul de a iniţializa biblioteca Serial. Parametrul viteza este dat în biţi pe secundă

-*Serial.read()* – citeşte date de pe portul serial

-*Serial.print()* – tipareşte date pe portul serial

-*Serial.println()*- tipareşte datele pe o linie nouă

Arduino Uno utilizează pentru comunicația serială, mai exact pentru comunicația implicită, componente hardware care sunt integrate în microcontroler. În cazul în care este necesară utilizarea mai multor interfețe seriale, se pot obține utilizând biblioteci specifice, obținând interfețe seriale software. Limitarea acestora constă în viteza de transfer (baud rate) care este mai mică față de cel realizat hardware.

Un exemplu de utilizare a interfeței seriale este prezentat în cele ce urmează. Acesta recepționează un caracter de la calculator și trimite înapoi caracterul recepționat pe portul serial.

```
byte byteRead; //variabila de memorare a caracterului receptionat

void setup() {
  Serial.begin(9600); //comunicatia serial, baud rate 9600
}

void loop() {
  if (Serial.available()) {       //se verifica daca exista date receptionate
    byteRead = Serial.read(); //in caz afirmativ citeste octetul receptionat
    Serial.write(byteRead);  //transmite octetul receptionat inapoi pe interfata seriala
  }
}
```

SoftwareSerial este tot o modalitate de comunicare seriala, doar ca in loc de a utiliza pinii RX si TX ai microcontroller-ului (care sunt dedicati pentru o astfel de comunicare), utilizam o librarie software care emuleaza comunicarea folosind (aproape) oricare doi pini digitali ai placii Arduino. Avantajul imediat este faptul ca in acest mod putem conecta placa Arduino cu mult mai multe dispozitive seriale. [25]

```
#include <SoftwareSerial.h>
SoftwareSerial mySerial(10, 11); // RX, TX
void setup() {
 Serial.begin(9600);
 Serial.println("Goodnight moon!");
 mySerial.begin(4800);
 mySerial.println("Hello, world?");
}
void loop() {
 if (mySerial.available()) {
 Serial.write(mySerial.read());
 }
 if (Serial.available()) {
 mySerial.write(Serial.read());
 }
}
```

Un exemplu de program care utilizează transmisia serială între două sisteme de dezvoltare Arduino Uno, folosind schema din figura 4.2, care transpune programul Blink (din exemple) în varianta Remote Blink.

```
//Transmițător
void setup() {
Serial.begin(9600);
}
void loop() {
Serial.print('H');
```

```
//Receptor
void setup() {
Serial.begin(9600);
pinMode(13, OUTPUT);
}
void loop() {
```

```
delay(1000);
Serial.print('L');
delay(1000);
}
```

```
if (Serial.available() > 0) {
incomingByte= Serial.read();
if (incomingByte== 'H') {
        digitalWrite(13, HIGH);
}
if (incomingByte== 'L') {
digitalWrite(13, LOW);
}}}
```

5. Măsurarea semnalelor analogice

Un semnal analogic este un semnal continuu in amplitudine si timp, poate avea orice valoare in orice moment de timp, cum se poate observa in graficul din partea stanga a figurii 1. Un semnal digital, pe de alta parte, este discret si in amplitudine si in timp, poate avea un numar finit de valori, cum se poate observa in graficul din partea dreapta a figurii 5.1.

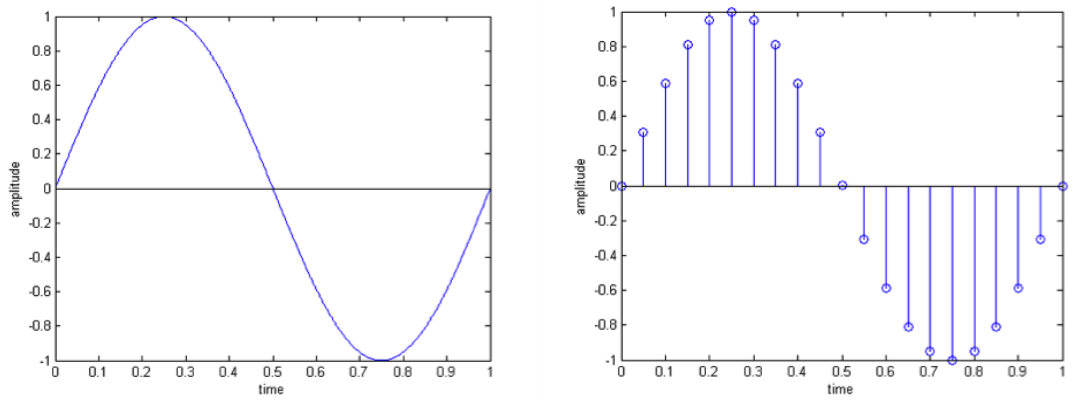


Figura 5.1: Semnal analogic si discret

Un calculator este capabil sa lucreze doar cu semnale discrete (digitale). In acelasi timp, aproape toate semnalele din lumea reala sunt semnale continue (analogice).

Pentru a putea măsura semnalele analogice într-un sistem de calcul digital, acestea trebuie convertite în valori numerice discrete. Un *convertor analog – digital* (ADC) este un circuit electronic care convertește o tensiune analogică de la intrare într-o valoare digitală.

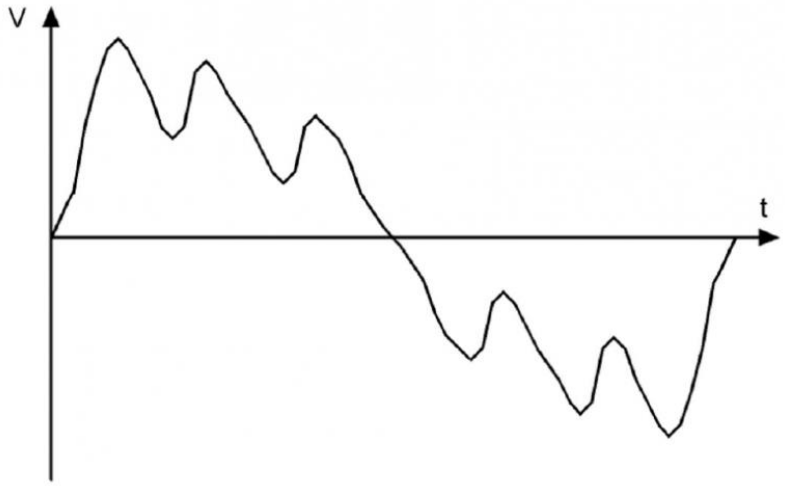


Fig. 5.2: Reprezentarea unui semnal analogic

Convertorul analog-digital (prescurtat ADC – analog digital converter) are nevoie ca semnalul analogic sa fie intr-un anumit interval de valori. Acest domeniu de valori poate fi diferit pentru fiecare circuit de conversie, este specificat in foaia de catalog.

In cazul convertorului integrat in microcontrolerul ATMEGA328, care se gaseste in placile de dezvoltare Arduino Uno, tensiunea minima de masurare este 0V iar cea maxima este 5V.

Dupa cum se stie, calculatoarele opereaza utilizand numere reprezentate binar, aceasta insemna ca si valoarea obtinuta in urma conversiei trebuie sa fie in acelasi sistem binar de reprezentare.

O caracteristică importantă a unui ADC o constituie **rezoluția** acestuia. Rezoluția indică numărul de valori discrete pe care convertorul poate să le furnizeze la ieșirea sa în intervalul de măsură. Deoarece rezultatele conversiei sunt stocate intern sub formă binară, rezoluția unui convertor analog-digital este exprimată în biți.

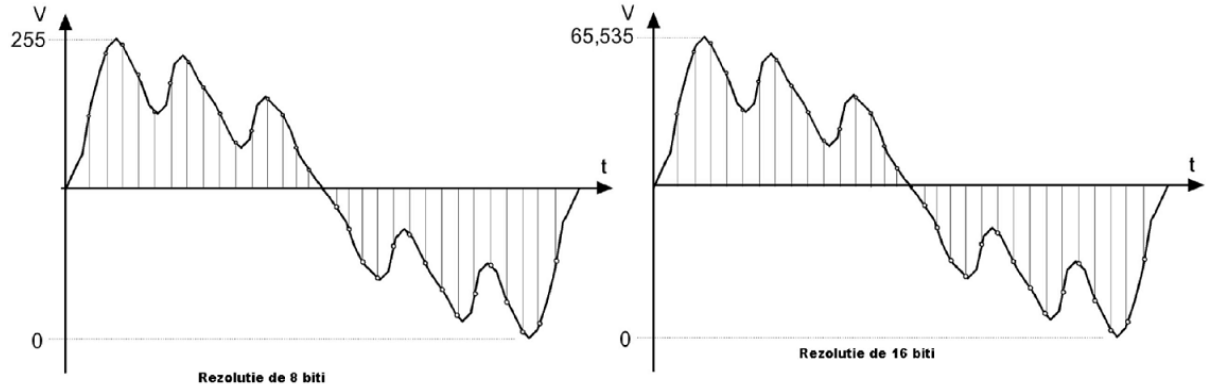


Fig.5.3: Același semnal analogic eșantionat la rezoluții diferite

De exemplu, dacă rezoluția unui convertor este de 10 biți atunci el poate furniza $2^{10} = 1024$ valori diferite la ieșire. Dacă gama de măsurare este de 0-5V, rezoluția de măsurare va fi: $\frac{5V-0V}{1024} = 0.0048V = 4.8mV$.

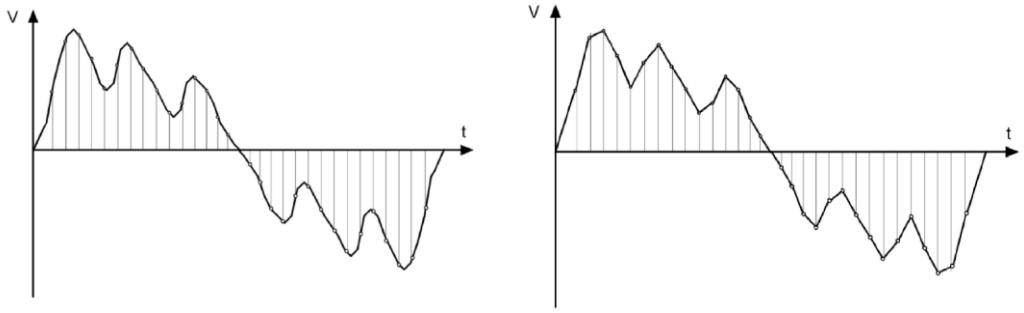


Fig. 5.4: Semnalul analogic refăcut în urma conversiei inverse (DAC)

Care este însa rata minimă de eșantionare pentru a reproduce fără pierderi un semnal de o frecvența data?
Teorema lui Nyquist spune că o rată de eșantionare de minim două ori mai mare decât frecvența semnalului măsurat este necesară pentru acest lucru, teorema aplicându-se și pentru un semnal compus dintr-un intreg spectru de frecvențe, cum ar fi vocea umană sau o melodie. Limitele maxime ale auzului uman sunt 20Hz – 20kHz dar frecvențele obișnuite pentru voce sunt în gama 20-4000Hz, de aceea centralele telefonice folosesc o rată de eșantionare a semnalului de 8000Hz. Rezultatul este o reproducere inteligibilă a vocii umane, suficientă pentru transmiterea de informații într-o convorbire obișnuită. Pentru reproducerea fidelă a spectrului audibil se recurge la rate mai mari de eșantionare. De exemplu, înregistrarea pe un CD are o rată de eșantionare de 44100Hz ceea ce este mai mult decât suficient pntru reproducerea fidelă a tuturor frecvențelor audibile.

In figura 5 se poate observa diferenta care poate sa apara daca un semnal este esantionat cu o frecventa mai mica fata de cea obtinuta conform teoremei lui Nyquist. Se poate observa ca

exista posibilitatea ca in urma reconstituirii semnalului analogic din cel discret sa se obtina un cu semnal care este diferit de cel initial.

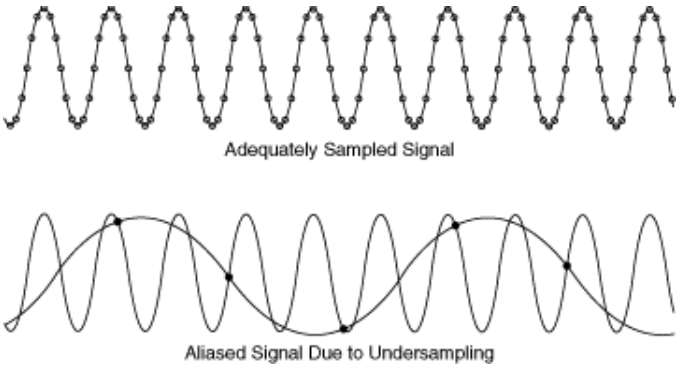


Fig. 5.5 Sus- semnal esantionat corect
Jos – semnal esantionat cu o frecventa mai mica fata de de cea corecta

In funcție de modul în care se execută conversia, convertoarele analog-digitale pot fi de mai multe **tipuri** :

- ADC paralel (Flash)
- ADC cu aproximare succesivă
- ADC cu integrare (single-slope, dual-slope);
- ADC Sigma-delta (delta-sigma, 1-bit ADC sau ADC cu oversampling).

Convertorul ADC al Atmega328

Convertorul analog-digital inclus în microcontrollerul Atmega328 este un ADC cu aproximări succesive. Are o rezoluție de pana la **10 biți** și poate măsura orice tensiune din gama 0-5V de pe **opt intrări analogice** multiplexate. Dacă semnalul de la intrare este prea mic în amplitudine, convertorul are facilitatea de preamplificare a acestuia in două setări, de 10x sau de 200x.

Acest convertor poate fi controlat printr-un registru de stare și control (ADCSRA) și un registru cu biți de selecție pentru multiplexoare (ADMUX). În primul dintre ele, putem stabili când să se efectueze conversia, dacă să se genereze întrerupere la finalul unei conversii etc. Folosind registrul pentru multiplexoare se alege ce canal va genera input pentru convertor si tensiunea de referință. De asemenea, în afară de aceste două registre mai avem registrul ADC în care este scris rezultatul conversiei (ADC).
Registrele necesare pentru configurarea convertorului sunt următoarele:

ADMUX - ADC Multiplexer Selection Register

Mai multe detalii puteți să găsiți la secțiunea 23.9 - Register description din datasheet.

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. 5: Registrul ADMUX

- REFS1 și REFS0 selectează referința de tensiune, adică valoarea maximă care poate fi convertită. Utilizatorul poate alege dintre două referințe interne, de 2.5V și 5V sau o referință externă (orice valoare între 0V si 5V) care trebuie furnizată pe pinul AVcc.
- ADLAR selectează ordinea biților rezultatului conversiei (LSB-first sau MSB-first).
- MUX4..0 selectează intrarea de pe care se face conversia. Se poate alege și un mod de functionare diferențial, caz în care se va măsura diferența de tensiune între cei doi pini aleși.

ADCSRA - ADC Control and Status Register A

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. 5.6: Registrul ADCSRA

- ADEN este bitul de Enable și activează convertorul.
- ADSC: scrierea acestui bit pe 1 va genera automat o conversie de pe intrarea selectată.
- ADATE este bitul de Enable pentru trigger-mode în care o conversie este generată pe frontul pozitiv al unui semnal furnizat extern de către utilizator.
- ADIF este bitul de întrerupere și va fi setat automat la terminarea unei conversii.
- ADIE este bitul de Enable pentru întrerupere. Aceasta va fi generată automat la fiecare conversie dacă bitul respectiv este 1.
- ADPS2..0 selectează rata de eșantionare pentru convertor. Aceasta poate fi de maxim fclk/2.

ADC - ADC Data Register

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Fig. 5.7: Registrul de date ADC

În funcție de valoarea bitului ADLAR, rezultatul conversiei va fi pus în registrul ADC aliniat la stânga sau dreapta.

Relația dintre valoarea din registrul ADC și tensiunea măsurată este următoarea: $ADC = \frac{V_{in} \cdot 1024}{V_{ref}}$, unde *Vin* este tensiunea măsurată iar *Vref* este tensiunea aleasă ca referință.

Exemple de utilizare

- Inițializare ADC: canal 1 (PA1):

```
ADMUX = 0;
ADMUX |= (1 << MUX0);        // ADC1 - channel 1
ADMUX |= (1 << REFS0);       // AVCC with external capacitor at AREF pin
ADCSRA = 0;
ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); //set clock prescaler at 128
ADCSRA |= (1 << ADEN);       // enable ADC
```

- Citire ADC prin polling:

```
ADCSRA |= (1 << ADSC);        //start conversion
loop_until_bit_is_set(ADCSRA, ADIF);  //wait until conversion is done: until ADIF from ADCSRA is set to 1
unsigned int value = ADC;
```

Materiale:

- Arduino Uno Board
- cablu USB A-B
- fire de conexiuni

Procedura:

- Mersul lucrarii
  - Se verifica sa nu fie alimentata placa Arduino
  - Conectati cu ajutorul unui fir de conexiune tensiunea 3,3V la pinul A0

Figura 5.8

- Scrieti codul
  - Utilizati codul de mai jos ca si un model pentru a face conversia la A0:

Listing 5.1

```
////////////////////////////////////////////////////////////////////////////////
//  AnalogRead la Serial
//  Author: Sabou Sebastian
////////////////////////////////////////////////////////////////////////////////

// initializare  pin 0 al Arduino ca si pin intrare
int aIn = A0;
void setup() {
        //comunicatia seriala, 9600 baud
```

```
        Serial.begin(9600);
}

void loop() {
        // citeste intrarea analogica de la  pin 0:
        int sensorValue = analogRead(aIn);

        // tipareste valoarea obtinuta in urma conversiei la portul serial
        Serial.println(sensorValue);
        delay(10);
        // intirziere pentru a putea observa modificarile la portul serial
}
```

- Majoritatea liniilor de cod au fost utilizate in lucrarea de laborator referitoare la comunicatia seriala
- Partea nou de cod este:

  ```
  int sensorValue = analogRead(aIn);
  ```

- Aceasta comanda are ca efect conversia tensiunii de la intrarea memorata in variabila *aln*. De remarcat ca variabila memoreaza valoarea A0 desi *aln* este de tip intreg si prin urmare A0 nu este, in genera, tratat ca un integer. Este o exceptie, A indica, in acest caz, ca este o intrare Analogica.
- Valoarea tensiunii de la pinul A0 este convertita de catre ADC intr-un numar din intervalul 0-1023 si este stocata in variabila *sensorValue*.
- Dupa scrierea codului se procedeaza la incarcarea fisierului compilat (verificati sa fie selectata corect varianta de placa – Arduino Uno si portul de comunicatie sa fie cel corect)
- Compilati si incarcati fisierul executabil in microcontroler
  - Deschideti monitorul serial din IDE Arduino
  - Verificati daca este selectata valorea de 9600baud pentru comunicatie
  - Observati valoarea afisata
  - Lasati monitorul serial deschis
- Modificati valoarea tensiunii de la intrare prin conectarea firului de conexiune la +5V, 3.3V, GND (0V).
- Convertiti valoarea afisata pe terminalul serial in tensiunea analogica corespunzatoare (practic obtineti valoarea tensiunii de la intrare) cu ajutorul formulei:

$$Voltage = (ADC\ output)\frac{V_{\mathrm{REF+}}}{2^n}$$

  - unde *n* este numarul de biti ai convertorului ADC. V_{REF+} =5V in cazul Arduino si numarul de biti este 10

Modificati programul sa utilizeze alta intrare (A1, …A5).
Modificati programul astfel incat sa utilizeze doua intrari pentru conversie si sa afiseze valorile respective.

Realizati circuitul de mai jos, utilizand 5V si GND de pe placa Arduino.

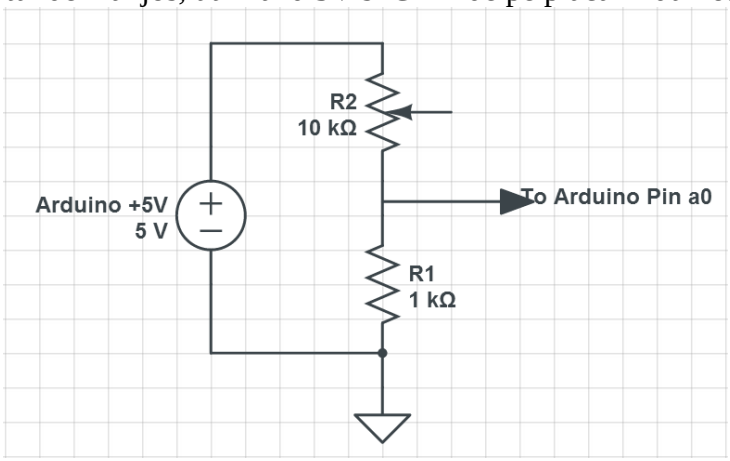


Figura 5.9 Măsurare tensiune continuă

Programul care preia rezultatul conversiei ADC si converteste acest rezultat in valoarea tensiunii corespunzatoare de la intrare este prezentat mai jos. Verificati corectitudinea afisarii cu tensiunea de la pinul A0, masurand aceasta tensiune cu ajutorul unui voltmetru electronic.

Realizati cel putin 10 masuratori diferite, modificand potentiometrul, si calculati eroarea convertorului ADC. (Diferenta dintre rezultatul obtinut in urma conversiei si valoarea masurata cu ajutorul voltmetrului electronic)

Calculati procentual eroarea obtinuta mai sus.

```
//////////////////////////////////////////////////////////////////////
// AnalogRead - Potentiometru
// Author: Sabou Sebastian
//////////////////////////////////////////////////////////////////////
int potentiometer = A0; // Se declara pinul la care e conectat potentiometrul
float voltageReading = 0;

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

void loop(){
  // Se citeste rezultatul conversiei
  float raw = analogRead(potentiometer);

  // Se converteste rezultatul in valoarea tensiunii de la intrare
  voltageReading = (raw/1023) * 5;

  // Se afiseaza valoarea tensiunii
  Serial.print("Tensiune potentiometru: ");
  Serial.println(voltageReading);
```

```
  // Se seteaza frecventa cu care se face conversia la 4Hz
  delay(250);
}
```

6. Dispozitive de afișare - LCD

Informaţii preluate şi adaptate de la [17], [18] ,[19]

Ecranul LCD pe care il vom utiliza este dedicat afişării textului. Ca şi caracteristici, el poate afişa un text format din 32 de caractere, cate 16 pe fiecare dintre cele două rânduri disponibile. Fiecare caracter este de fapt o matrice de 5x8 puncte. Pentru a putea comanda LCD-ul este utilizat un controller de tip HD44780 care are două registre pe 8 biţi:

IR (Instruction Register) - tine minte coduri reprezentand instructiuni precum stergerea ecranului sau mutarea curorului la o anumita pozitie.

DR (Data Register) - memoreaza remporar informatiile ce pot fi stocate fie in DDRAM sau in CGRAM.

Alegerea intre cele doua registre se realizează prin intermediul unui selector de registru care, in cazul ecranului nostru, este conectat la pinul 8.

Ecranul contine doua tipuri de memorii:
DDRAM (Display Data RAM) - memoreaza codurile ascii ale caracterelor ce vor fi afisate si poate memora pana la 80 de litere. Adresele sunt de la 00 pana la 0F pentru primul rand si de la 40 pana la 4F pentru al doilea rand.
CGRAM - in aceasta memorie este stocata forma caracterelor. Scriind in ceasta memorie, un programator poate sa isi defineasca propriile caractere (din pacate puteti defini doar 8 caractere).
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).

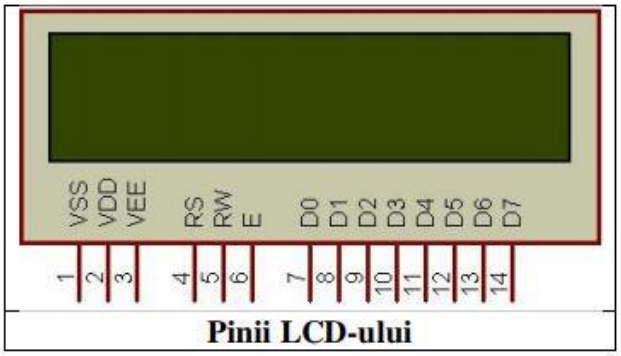


Figura 6.1 Configuraţia pinilor pentru afişajul LCD

Pentru comunicaţia cu modulul LCD, se utilizează mai mulţi pini, transferul datelor fiind realizat în mod paralel a celor 8 biţi. Controlerul afisajului (HD44780) pemite şi transferul datelor utilizând doar 4 biţi pentru date, octetul fiind transferat în doi paşi. Astfel se poate realiza o economie a pinilor utilizaţi pentru comanda LCD.

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)
Semnificația pinilor LCD		

Figura 6.2 Semnificaţia pinilor afişajului 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 biţi, sau 80 de caractere.

Codul caracterelor pentru caractere formate din 5 x 8 puncte

Figura 6.3 Codurile ASCII care pot fi utilizate pentru afişare

Memoria rămasă liberă poate fi folosita 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).

Aspectul caracterelor de la 0x00 la 0x07 poate fi definit de utilizator. Se va specifica pentru fiecare caracter o matrice de 8 octeți, cate unul pentru fiecare rând. Cei mai putini semnificativi 5 biți din fiecare rând vor specifica care pixeli vor fi aprinși și care nu (vezi figura 6.4) [18].

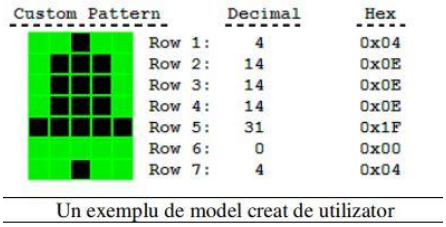


Figura 6.4 Realizarea de caractere personalizate

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.

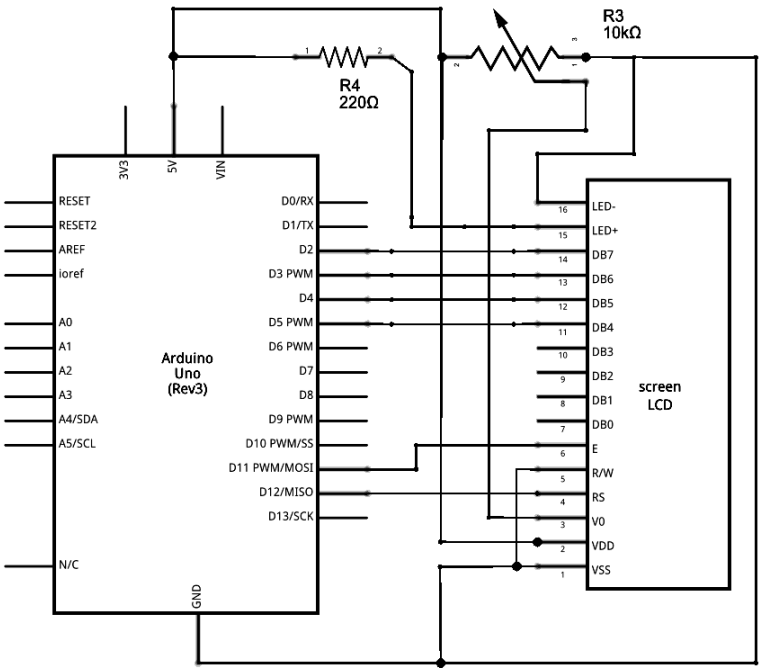


Figura 6.5 Schema electrică de conectare a unui LCD

Exemplu de program pentru afişarea pe ecranul LCD

```
//include biblioteca pentru lucrul cu LCD
#include <LiquidCrystal.h>
/* LCD RS pinul digital 12
 * LCD Enable pinul 11
 * LCD D4 pinul 5
 * LCD D5 pinul 4
 * LCD D6 pinul 3
 * LCD D7 pinul 2
Al șaptelea pin este pinul de la potentiometrul de reglare a iluminarii */
 //inițializează lcd-ul la valorile stabilite ale pinilor
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
unsigned long time;
void setup()
{
 //seteaza numarul de randuri si coloane ale LCD-ului
 lcd.begin(16, 2);
}
void loop()
{
 //luam numarul de secunde trecute de la reset (sau pornirea programului)
 time = millis() / 1000;
 //setam cursorul la coloana 0 randul 1
 lcd.setCursor(0, 0);
 //scriem un text
 lcd.print("hello, world!");
 //setam cursorul la mijlocul liniei a doua
 lcd.setCursor(0, 1);
 //scriem timpul
 lcd.print(time);
}
```

Figura 6.6 Rezultatul rulării programului

Modificați programul anterior astfel încât să actualizeze doar caracterele care sunt modificate pentru timpul afișat.

Exemplu: generarea de caractere utilizator (Figura 6.4)
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, poziŃiile 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)); // ObservaŃi diferenta dintre print şi write
/* cand referiti caracterul „0” trebuie sa faceti un cast la byte. Altfel compilatorul va
semnala o eroare. Exceptie este cazul in care referii o variabila:
byte zero = 0;
lcd.write(zero);
*/
// Afisare pe a doua linie
lcd.setCursor(0, 1);
lcd.print("Sad ");
lcd.write(1); // cand referiti caractere diferite de „0” nu mai este necesar cast-ul;
}
// Functia loop ramane nefolosita, puteti să o folositi pentru a extinde funcționalitatea
void loop()
{
}
```

7. Protocolul de comunicaţie I2C

Protocolul Inter Integrated Circuit (I2C) este un protocol creat pentru a permite mai multor circuite integrate “slave” să comunice cu unul sau mai mute circuite integrate “master”. Acest tip de comunicare poate fi folosit doar pe distanțe mici de comunicare și asemenea protocolului UART are nevoie doar de 2 fire de semnal pentru a trimite/primii informații.
Protocolul I2C a fost dezvoltat în 1982 de către firma *Philips* pentru diferite circuite integrate *Philips*. Specificațiile originale permiteau comunicarea doar cu frecvența de 100kHz și doar pe 7 biți de adresă, limitând numărul dispozitivelor conectate la bus, la 112. În 1992 spațiul de adrese a fost extins la 10 biți și comunicarea se realiza pe 500kHz. În prezent există trei moduri adiționale: *fast plus (1MHz)*, *high-speed (3.4MHz)* si *ultra-fast (5MHz)*.
In plus, in 1995 *Intel* a introdus o nouă variantă de I2C, numită *"System Management Bus" (SMBus)*, care este mult mai bine controlată şi realizată cu intenția de a maximiza predictibilitatea comunicării dintre suporturile IC și plăcile de baza ale PC-urilor. Cea mai notabilă diferență dintre SMBus si I2C este aceea că prima limitează viteza de comunicaţie de la 10kHz la 100kHz, iar a doua poate suporta dispozitive cu o frecvenţă de la 0kHz la 5MHz.

Figura 7.1 Comunicaţia serială între două dispozitive

Un dezavantaj al comunicaţiei seriale asincrone este acela că, comunicarea este permisă doar între două dispozitive. Chiar daca este posibil să conectăm mai multe dispozitive la un singur port serial, apar conflicte de bus ("bus contention") deoarece două dispozitive tind să conducă aceeași linie în același timp şi dispozitivele se pot defecta.
În cele din urmă, rata de transfer a datelor reprezintă o problema. Teoretic nu este nici o limită în *comunicarea seriala asincrona*, dar practic în *comunicarea UART*, dispozitivele suportă o rata de transfer fixă şi cea mai mare rată este de aproximativ 230400 biți/secunda.

Dezavantajele comunicaţiei seriale SPI.

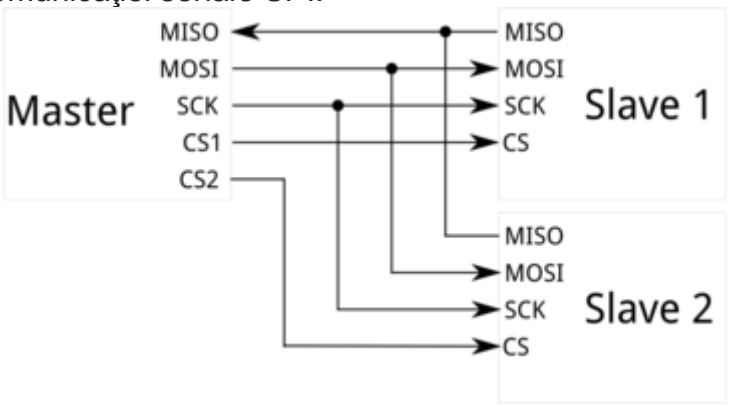


Figura 7.2 Comunicaţia serială SPI

Cel mai mare minus pe care-l are protocolul *SPI* este numărul mare de pini de care este nevoie. Pentru conectarea unui singur "master" cu un singur "slave" la un bus SPI, este nevoie de 4 linii, iar fiecare "slave" pe care-l conectam după, necesita un circuit adițional ce selectează un pin I/O de pe dispozitivul folosit ca "master". Prin urmare, în acest tip de comunicare este realizabilă conectarea a mai multor dispozitive "slave" la un singur dispozitiv "master", dar cu toate acestea, din cauza numărului mare de pini utilizați, este nevoie ca circuitul „master” să utilizeze mai multe porturi de ieşire pentru comanda circuitelor „slave”
SPI este totuși bun din cauza implementării ușoare si a ratei de transmitere a datelor (transmitere si recepționare simultana) prin conexiunea full-duplex, suportând o rata de transmisie de 10 milioane de biți/secunda.

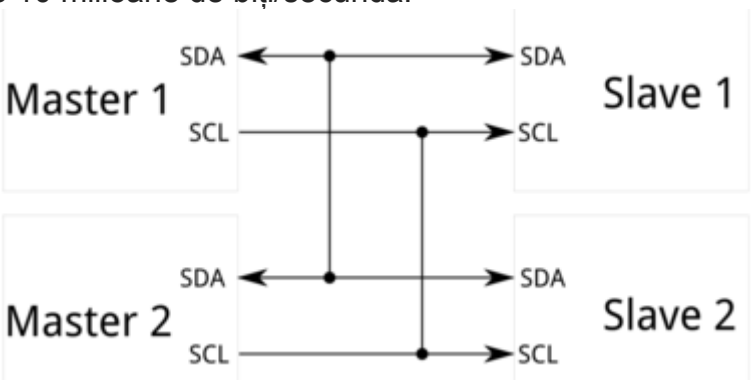


Figura 7.3 Conectarea mai multor dispozitive I2C

I2C necesita două legături, similar cu comunicarea serială asincronă, cu diferența ca acest tip de comunicare poate suporta pana la 1008 dispozitive de tip "slave". Mai mult decât atât, in comunicarea de tip I2C pot exista mai multe dispozitive de tip "master" conectate la un bus, lucru nepermis în comunicaţia *SPI*.
Rata de transfer a datelor nu este foarte bună, fiind asemănătoare cu cea de la *portul serial*; cele mai multe dintre dispozitivele *I2C-uri* comunica cu o rată de transmisie cuprinsă între 100kHz şi 400kHz.
În ceea ce privește implementarea, aceasta este mai complexă decât în cazul implementării *SPI*, dar mult mai ușoara decât in cazul *comunicării asincrone, cu UART*.

Fiecare bus *I2C* este compus din doua semnale: *SCL* (semnalul de ceas) si *SDA* (semnalul de date). Semnalul de ceas este întotdeauna generat de bus-ul masterul curent. Fiecare bus *I2C* poate suporta până la 112 dispozitive, iar toate dispozitivele trebuie să aibă aceeaşi conexiune *GND*.
Spre deosebire de alte metode de comunicare, precum *UART*, magistrala *I2C* este de tip “open drain”, ceea ce duce la faptul că fiecare dispozitiv conectat la magistrală modifică starea logică a magistralei doar în sensul nivelului logic „low”, nici un dispozitiv nu va încerca „fixarea” unui nivel logic „high”. Așadar, se elimină problema de "bus contention”, unde un dispozitiv încearcă sa „tragă” una dintre linii in starea "high” in timp ce altul o aduce in "low”, eliminând posibilitatea de a distruge componente. Fiecare linie de semnal are un rezistor pull-up pe ea, pentru a putea readuce semnalul pe "high” când nici un alt dispozitiv nu cere „low”.

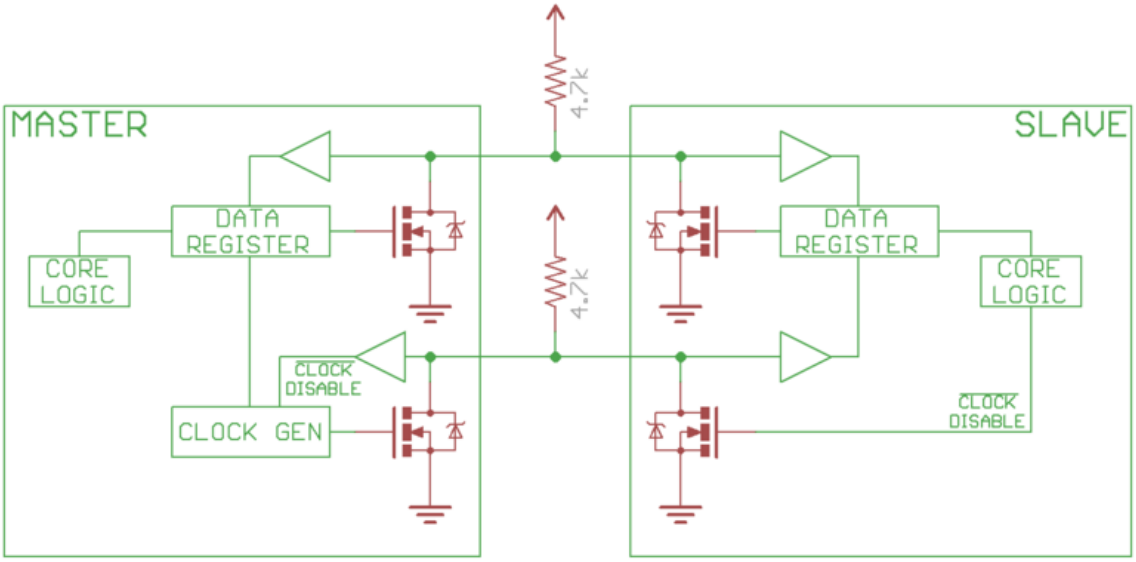


Figura 7.4 Circuitul electric de interfaţă în cazul I2C

În figura 7.4 se remarcă folosirea a două rezistenţe pe liniile de semnal. Selecția rezistentelor variază odată cu dispozitivele care folosesc busul, dar o regula buna este de a începe cu rezistente de 4.7k si cu scăderea valorilor lor dacă e necesar.

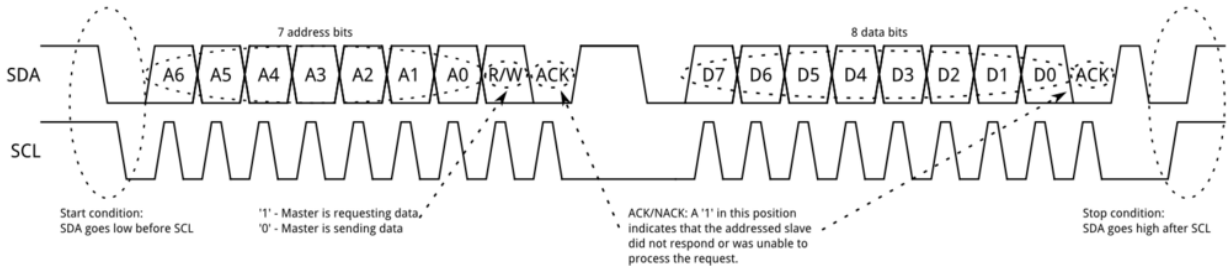


Figura 7.5 Transmisia datelor în cazul protocolului I2C

Mesajele transmise sunt împărțite in doua tipuri de cadre (frames): cadre de date (conțin mesaje pe 8 biți direcționate de la dispozitivul de tip "master" la cel de tip "slave" şi invers) şi cadre de adresă (conține adresa dispozitivul de tip "slave", la care dispozitivul de tip "master" trimite mesajul).

Datele sunt puse pe linia SDA după ce SCL ajunge la nivel "low" şi sunt eșantionate când SCL ajunge "high". Timpul între nivelul de ceas si operațiile de citire/scriere este definit de dispozitivele conectate pe magistrala şi va fi diferit, funcţie de circuitele integrate utilizate.

Condiția de start

Pentru a inița cadrul de adresa, dispozitivul "master" lasă SCL "high" si pune SDA pe "low", astfel toate dispozitivele "slave" sunt pregătite pentru a recepționa date.

Daca doua sau mai multe dispozitive "master" doresc sa-si asume bus-ul la un moment dat, primul dispozitiv care ajunge de la "high" la "low" primul, va prelua controlul bus-ului.

Cadrul de adresa

Cadrul de adresa este mereu primul atunci când o noua comunicare începe. Mai întâi se vor trimite sincron biții adresei (primul fiind cel mai semnificativ), apoi se va transmite un semnal de tip R/W pe biți. Bitul 9 al cadrului este reprezentat de bitul NACK/ACK. Acesta este cazul pentru ambele cadre (de adrese si de date).

După ce primii 8 biți ai cadrului sunt transmiși, dispozitivului receptor ii este dat controlul asupra SDA-ului. Daca acest dispozitiv nu schimba in 0 logic linia SDA înainte de al nouălea puls de ceas, se poate deduce ca acesta nu a primit datele, ori nu a știut sa interpreteze mesajul recepționat. In acest caz, schimbul de date se oprește si rămâne la latitudinea dispozitivului "master" cum va proceda mai departe.

Cadrul de date

După ce cadrul de adresa a fost trimis, datele/mesajele pot începe a fi transmise. Dispozitivul "master" va continua sa genereze impulsuri de ceas la un interval regulat, iar datele vor fi plasate pe SDA, fie de dispozitivul "master" fie de cel "slave", in funcție de biții R/W transmiși in cadrul anterior. Numărul de cadre de date este arbitrar.

Condiția de oprire

De îndată ce toate cadrele au fost trimise, dispozitivul "master" va genera o condiție de oprire. Condițiile de oprire sunt definite de tranziția de la "low" la "high" (0 -> 1) pe SDA, după o tranziție 0 -> 1 pe SCL cu SCL pe "high".

În timpul operației de scriere, valoarea din SDA nu ar trebuie sa se schimbe când SCL este "high" pentru a evita condițiile false de oprire.

În cazul Arduino UNO, pinii I2C pentru Arduino sunt pinul analogic 4 (care este pinul SDA), si pinul analogic 5 (care este pinul SCL). Pentru Arduino Mega, pinul digital 20 este pinul SDA si pinul digital 21 este pinul SCL. In cazul Arduino Leonardo, cei doi pini SDA si SCL sunt distincți, marcați ca atare pe placa. Pentru a folosi un dispozitiv I2C, vei conecta pinul SDA al dispozitivului cu pinul SDA al Arduino, pinul SCL al dispozitivului cu pinul SCL al Arduino. In plus, in funcție de dispozitiv, s-ar putea sa fie necesar sa conectezi si pinul de adresa la GND sau la VCC (in cele mai multe cazuri, pinul de adresa este deja conectat, dar nu întotdeauna).

Comunicarea dintre trei sisteme de dezvoltare Arduino Uno, utilizând protocolul I2C. Programul utilizat este prezentat mai jos, pentru fiecare din cele trei sisteme (unul master şi două sisteme de tip slave).

Master

```
#include <Wire.h>
void setup()
{
  Wire.begin();
}
void loop()
{

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

 delay(500);


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

 delay(500);


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

 delay(500);


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

 delay(500);
}
```

Slave1

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

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

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

   if (inChar == 'H')
   {
     digitalWrite(13, HIGH);
   }
   else if (inChar == 'L')
   {
     digitalWrite(13, LOW);
   }
   else
   {
     Serial.println("Different from H and
L");
   }
}
```

Slave2

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

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

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

O situaţie des întâlnită de utilizarea a magistralei I2C, în cazul sistemelor Arduino, este aceea de conectare a unui dispozitiv de afişare de tip LCD alfanumeric. Pentru reducerea conexiunilor electrice şi utilizarea pinilor disponibili de la Arduino pentru alte dispozitive, sunt motivele principale de utilizare a acestui circuit adaptor.
În figura 7.5 este prezentată imaginea acestui circuit conectat la LCD, iar în figura 7.6 este prezentată schema electrică de conectare.

Figura 7.5 Sistemul I2C care permite conectarea ecranului LCD la Arduino

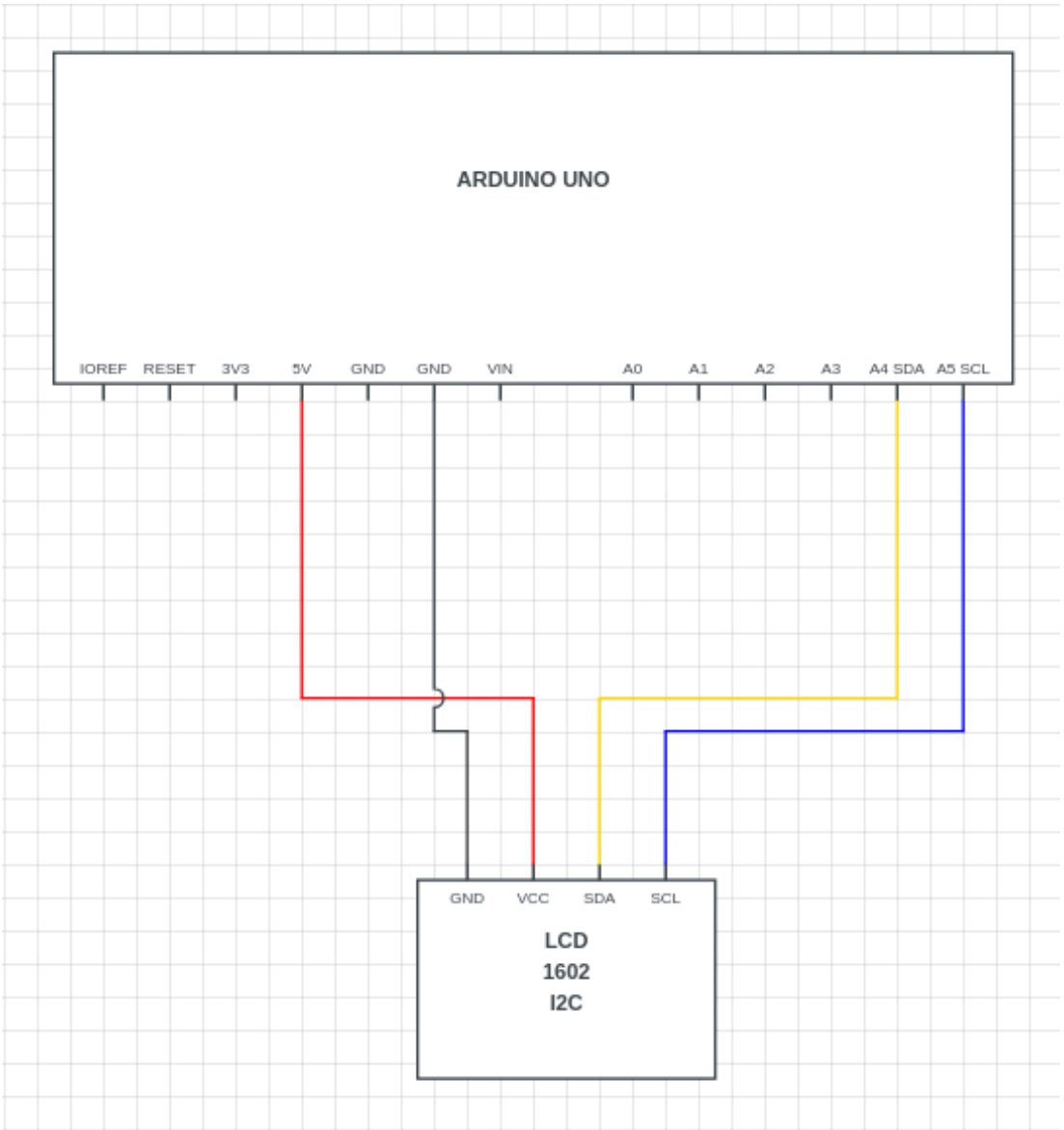


Figura 7.6 Schema electrică de conectare LCD I2C

Se conectează pinul 1 (SDA) la pinul analogic 4 al Arduino, pinul 2 (SCL) la pinul 5 analogic al Arduino, pinul 3 (VCC) la pinul 5V al Arduino si pinul 4 (GND) la pinul GND al Arduino.
Din meniul Examples, încarcă exemplul "HelloWorld_i2c" (sau încarcă codul de test de mai jos).

```
#include "Wire.h"
#include "LiquidCrystal.h"

// Connect via i2c, default address #0 (A0-A2 not jumpered)
LiquidCrystal lcd(0);

void setup() {
  // set up the LCD's number of rows and columns:
   lcd.begin(16, 2);

  lcd.setBacklight(HIGH);
  lcd.print("hello, world 0 !");
  lcd.setCursor(0, 1);
  lcd.print("hello, world 1 !");

}

void loop() {

}
```

8. Protocolul de comunicație SPI

Comunicarea SPI este de tip Master/Slave, circuitul master fiind de obicei un microcontrolerul ce controleaza unul sau mai multe circuite periferice. SPI a fost dezvoltată de Motorola pentru a furniza o interfaţă simplă, de cost redus între microcontrolere și dispozitive periferice. Fiind o intefață serială sincronă, transmisiile sunt sincronizate cu un semnal de ceas care sunt furnizate de către master (dispozitivul principal) receptorului (dispozitivul secundar), care și el la rândul său utilizează semnalul de ceas la achiziția fluxului de biți.

Pentru comunicarea SPI sunt utilizaţi o serie de pini dedicaţi şi care au roluri clar definite:

- MISO (Master In Slave Out) - este de fapt pinul 12 al Arduino-ului. Aşa cum sugerează şi denumirea, aceasta este linia utilizată de către slave pentru a trimite date catre master.
- MOSI (Master Out Slave In) - pinul 11 al Arduino, este utilizat pentru ca masterul să poata trimite date către slave.
- SCK (Serial Clock) - pentru ca cele două dispozitive să poată comunica sincron (după cum spuneam mai sus, comunicarea este una sincronă), este nevoie ca circuitul master să genereze frecvenţa care va reprezenta referinţa în ceea ce priveşte „ritmul" comunicaţiei. Circuitul slave va recepţiona această frecvenţă şi va reacţiona începând comunicaţia cu circuitul master. SCK in cazul nostru este implementat de pinul 13 al Arduino.
- SS (Slave Select) - Este utilizat de către master pentru a selecta diverse circuite slave. Deşi în acest caz este implementat utilizând pinul 10, acest lucru nu este obligatoriu putând fi utilizat oricare alt pin digital. În comunicaţia SPI, dacă pinul SS este setat pe valoarea LOW atunci circuitul master doreşte să comunice cu acel dispozitiv, practic se activează recepţia datelor pentru acel circuit. Ca să oprească comunicarea cu acel dispozitiv, se setează pinul SS pe high. Ca şi exemplu, daca am conectat doua dispozitive care comunică pe SPI, acestea vor utiliza aceiaşi pini 11,12,13 cu semnificatiile de mai sus dar vor utiliza doi pini SS diferiţi - să zicem 9 si 10 (la 10 sa presupunem ca avem conectat un shield ethernet iar la 9 avem conectat un cititor de carduri MicroSD). Pentru a selecta dispozitivul ce are conectat pinul SS la 9 va trebui sa setăm 10 cu high şi 9 cu low (în aceasta ordine) - în cazul nostru am selectat cardul MicroSD şi comunicaţia va fi facută cu el în timp ce shieldul ethernet va fi ignorat. Pentru a selecta dispozitivul ce are portul SS conectat la pinul 10, se va seta 9 cu high şi apoi 10 cu low - în cazul nostru se va opri comunicaţia cu cititorul de carduri şi se va comunica cu shieldul ethernet.

Cu o conexiune SPI avem mereu un dispozitiv principal care controlează dispozitivele periferice. Teoretic, sunt 3 linii comune pentru toate dispozitivele:

- MISO (Master In Slave Out)- linia secudară pentru trimiterea datelor la cea princi-pală.
- MOSI (Master Out Slave In)- linia principală pentru trimiterea datelor la dispozitivele periferice.
- SCK (Serial Clock)- impulsurile ceasului care sincronizează transmiterea datelor generate de dispozitivul principal.

Există o linie specifică fiecarui dispozitiv pentru unele procesoare:

- SS (Slave Select)- pin-ul fiecărui dispozitiv prin care dispozitivul principal poate activa sau dezactiva dispozitive specifice.

Când pin-ul SS al unui dispozitiv este activ, are loc comunicarea cu dispozitivul principal.
Când este inactivat, ignoră dispozitivul principal. Acesta permite distribuirea acelorași căi de transmisii MISO, MOSI și CLK mai multor dispozitive SPI.
Pentru a scrie codul unui nou dispozitiv SPI trebuie să ținem cont de următoarele:

- Viteza maximă folosită pentru un dispozitiv SPI este controlată de primul parametru din setările SPI. Viteza este mai mică sau egală decât cea a unui cip de 15 MHz care utilizează cu setările SPI.
- Modificarea datelor este controlată atât de al doilea parametru al setărilor SPI cât și de MSB (Most Significant Bit) sau LSB (Least Significant Bit). Majoritatea cipurilor SPI utilizează prima comandă de date.
- Al treilea parametru al setărilor SPI controlează impulsurile de ceas și permit punerea în așteptare a ceasului când este oprit sau pornit.

SPI Standard poate implementa mai multe dispozitive, fiecare întru-un mod diferit, adică trebuie să acorzi o atenție mai specială pentru fișa de date a dispozitivului când scri un cod.
Unele periferice care au interfețe SPI sunt în măsură să transfere mai mulți octeți deodată, în care se pot transfera un flux continuu de date de la dispozitivul principal către cele secundare. Acest tip de transfer permite ca semnalul SS să rămână activ pe toată durata transferului. Unele periferice secundare permit conectarea unui lanț de priorități. De exemplu, dacă dispozitivul principal trimite trei octeți prin interfața SPI, atunci primul octet va fi transferat la primul dispozitiv secundar A, timp în care B și C rămân liberi, apoi la transferul următorului octet, primul octetul din A merge la dispozitivul secundar B, iar noul octet rămâne în A. Și în final ultimul octet nou se transferă în A, cel de-al doilea ,care a fost în A, merge în B, iar ce a fost în B merge în dispozitivul secundar C.
Această conectare în lanț nu este funcțională la orice dispozitiv periferic SPI.
In general sunt 4 moduri de transmisie. Aceste moduri controlează tot timpul datele modificate la intrare sau ieșire, la creșterea sau scăderea semnalului de ceas (faza de ceas), când ceasul este în repaus și când este pornit sau oprit (polaritate de ceas).
Cele 4 moduri combină polaritățile și fazele conform acestui tabel (Tabelul 3.3):

Mod	Polaritatea ceasului (CPOL)	Faza de ceas (CPHA)
SPI_MODE0	0	0
SPI_MODE1	0	1
SPI_MODE2	1	0
SPI_MODE3	1	1

Tabelul celor 4 moduri a ceasului

Toate plăcile de bază AVR au un pin SS care este folositor atunci când se comportă ca și un dispozitiv secundar controlat de un dispozitiv principal extern. De când această librărie suportă doar modul principal acest pin ar trebui să fie setat mereu ca ieșire, altfel interfața SPI poate fi pusă automat în modul secundar de către componente, interpretând libraria inoperativă. Este posibilă folosirea oricărui contact SS pentru dispozitive.
Ca exemplu, placa de Arduino Ethernet folosește pinul 4 pentru a controla conexiunea SPI cu slotul cardului extern și pinul 10 pentru a controla conexiunea cu controlorul Ethernet.[4]

Pentru comunicaţia utilizând protocolul SPI se va utiliza biblioteca SPI.h, unde sunt funcţiile care se pot utiliza pentru această comunicaţie.

Figura 8.1 Comunicaţia SPI cu un singur circuit slave [12]

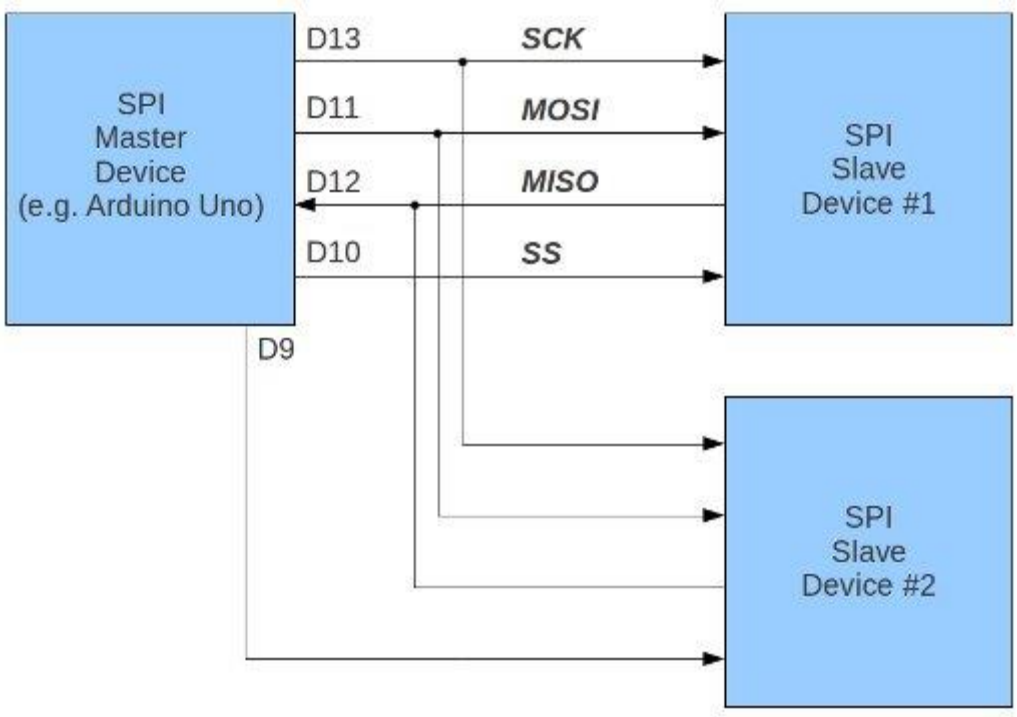


Figura 8.2 Comunicaţia SPI cu două circuite cu rol de slave [12]

În figurile 9.1 şi 9.2 sunt prezentate schemele de conectare a unui circuit master cu unul, respectiv două circuite slave. Se poate observa că la fiecare circuit slave adăugat, circuitul master trebuie să aloce un pin pentru activarea acestuia, cu cât mai multe circuite de tip slave sunt conectate la magistrală cu atât mai mulţi pini ai

circuitului master vor fi dedicaţi acestei comunicaţii, fiind indisponibilizaţi pentru alte utilizări.

În figura 8.3 este prezentată schema de conectare a două sisteme Arduino Uno care comunică între ele folosind magistrala SPI.

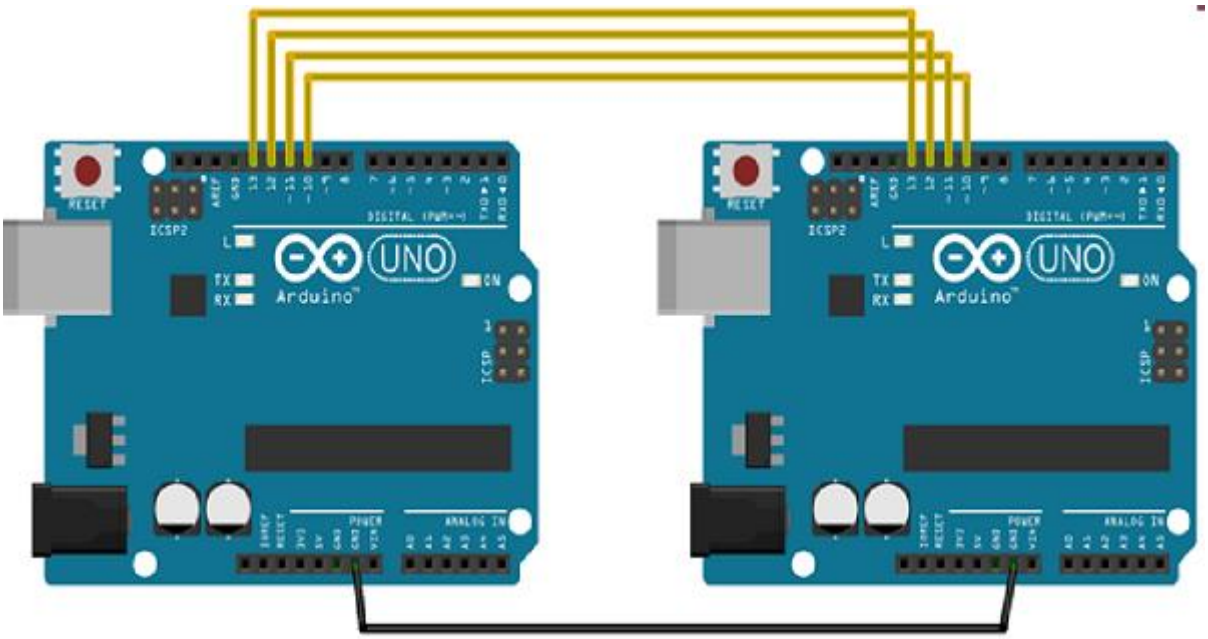


Figura 8.3 Comunicaţia a două sisteme Arduino Uno utilizând SPI

Programul utilizat pentru această comunicaţie este prezentat mai jos, unul este pentru microcontrolerul care are rolul de master şi un program este pentru al doilea circuit cu rolul de slave.

Exemplu de program - master

```
#include <SPI.h>
void setup (void) {
  Serial.begin(115200); //comunicaţia serială cu PC-ul va fi la 115200 baud
  digitalWrite(SS, HIGH); // dezactivare – deselectare circuit slave
  SPI.begin ();
  SPI.setClockDivider(SPI_CLOCK_DIV8);//divizorul semnalului de ceas 8
}

void loop (void) {
  char c;
  digitalWrite(SS, LOW); // selectarea circuitului slave
  // trimiterea sirului de caractere de test
  for (const char * p = "Hello, world!\r" ; c = *p; p++) {
    SPI.transfer (c);
    Serial.print(c);
  }
  digitalWrite(SS, HIGH); // dezactivare circuit slave
  delay(2000);
}
```

Exemplu de program pentru circuitul cu rol de slave

```
#include <SPI.h>
char buff [50];
volatile byte indx;
volatile boolean process;

void setup (void) {
  Serial.begin (115200);
  pinMode(MISO, OUTPUT); // pinul de transmisie setat ca si iesire
  SPCR |= _BV(SPE); // setarea SPI cu rol de slave
  indx = 0; // golirea buferului
  process = false;
  SPI.attachInterrupt(); // activarea intreruperii pentru SPI
}
ISR (SPI_STC_vect) // rutina pentru intreruperea SPI
{
  byte c = SPDR; // citire octet din registru de date SPI
  if (indx < sizeof buff) {
    buff [indx++] = c; // salvare date la urmatorul index in bufer
    if (c == '\r') //verificare sfarsit cuvant
    process = true;
  }
}

void loop (void) {
  if (process) {
    process = false; //resetare variabila  process
    Serial.println (buff); //tiparirea sirului de caractere la terminalul serial
    indx= 0; //resetare index
  }
}
```

9. Dispozitive de intrare - Tastatura

Pentru realizarea unor programe care să fie cât mai interactive este necesară utilizarea unor dispozitive de intrare prin intermediul cărora să poată fi introduse sau modificate anumite variabile. Înclusiv navigarea printr-un meniu este mult mai facilă prin utilizarea unor butoane.

Una din cele mai rapide metode de utilizare a unor butoane pentru sistemele de dezvoltare Arduino, este conectarea acestora prin intermediul unor rezistențe (connfigurate ca divizor de tensiune reglabil) la convertorul analog digital.

În figura 9.1 este prezentată schema de conectare și în figura 9.2 se prezintă implementarea acestei variante.

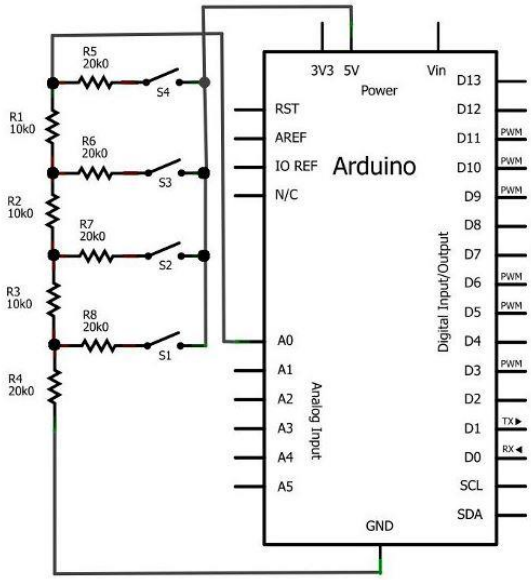


Figura 9.1 Tastatura analogică

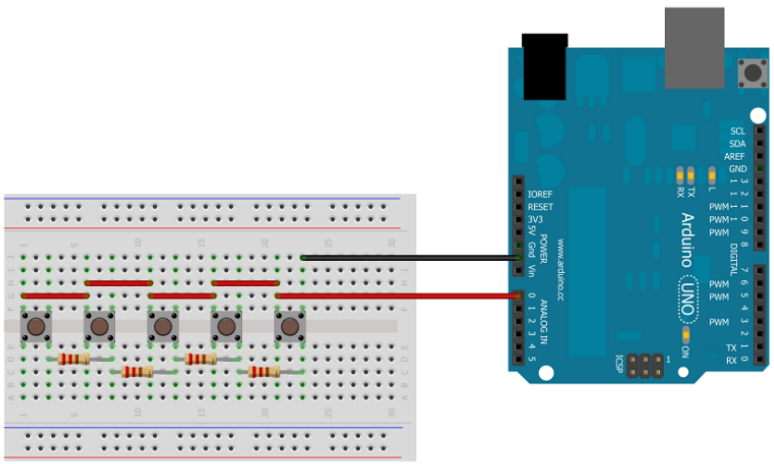


Figura 9.2 Implementarea practică a tastaturii analogice

Programul pentru utilizarea tastaturii analogice este prezentat mai jos:
//preluat din sursa [26]

```
void setup() {
//activeaza rezisstenta de pull up pentru portul
digitalWrite(A0, HIGH);
// initializare port serial
Serial.begin(9600);
}

void loop() {
// citeste valoarea de pe portul analogic
int analogic = analogRead(A0);
delay(200);//asteapta putin pentru revenirea butonului

if (!(analogic>=1010&&analogic<=1023)){//incepe verificarea doar daca s-a apasat
un buton
if (analogic>=185&&analogic<=200)//185-200 este intervalul valorilor citite pe portul
analogic pentru butonul 1
{
// tipareste valoarea si mesaj
Serial.println("Ai apasat butonul:1"); }

if (analogic>=150&&analogic<=170)//150-170 este intervalul valorilor citite pe portul
analogic pentru butonul 2
{
// tipareste valoarea si mesaj
Serial.println("Ai apasat butonul:2"); }

if (analogic>=100&&analogic<=149)//100-149 este intervalul valorilor citite pe portul
analogic pentru butonul 3
{
// tipareste valoarea si mesaj
Serial.println("Ai apasat butonul:3"); }

if (analogic>=69&&analogic<=101)//69-101 este intervalul valorilor citite pe portul
analogic pentru butonul 4
{
// tipareste valoarea si mesaj
Serial.println("Ai apasat butonul:4"); }

if (analogic>=0&&analogic<=25)//0-25 este intervalul valorilor citite pe portul analogic
pentru butonul 5
{
// tipareste valoarea si mesaj
Serial.println("Ai apasat butonul:5"); }

}
}
```

O variantă mai complexă este utilizarea unor butoane organizate matricial, cum este prezentată în figura 9.3

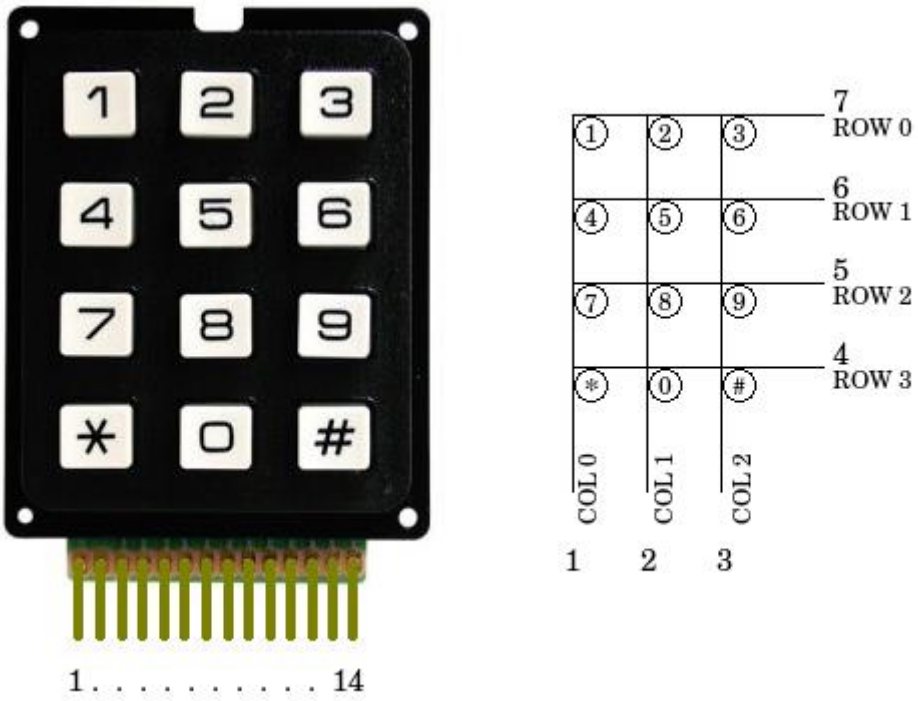


Figura 9.3 Tastatura cu organizare matricială a butoanelor

Pentru relizarea programului se folosește librăria ***Keypad*** .Exemplul de program care utilizează această variantă de tastatură este prezentat mai jos. [27]

```
/*  Keypadtest.pde
 *
 *  Demonstrate the simplest use of the  keypad library.
 *
 *  The first step is to connect your keypad to the
 *  Arduino  using the pin numbers listed below in
 *  rowPins[] and colPins[]. If you want to use different
 *  pins then  you  can  change  the  numbers below to
 *  match your setup.
 *
 */
#include <Keypad.h>

const byte ROWS = 4; // Four rows
const byte COLS = 3; // Three columns
// Define the Keymap
char keys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'#','0','*'}
};
// Connect keypad ROW0, ROW1, ROW2 and ROW3 to these Arduino pins.
byte rowPins[ROWS] = { 9, 8, 7, 6 };
// Connect keypad COL0, COL1 and COL2 to these Arduino pins.
byte colPins[COLS] = { 12, 11, 10 };
```

```
// Create the Keypad
Keypad kpd = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

#define ledpin 13

void setup()
{
  pinMode(ledpin,OUTPUT);
  digitalWrite(ledpin, HIGH);
  Serial.begin(9600);
}

void loop()
{
  char key = kpd.getKey();
  if(key)  // Check for a valid key.
  {
    switch (key)
    {
      case '*':
        digitalWrite(ledpin, LOW);
        break;
      case '#':
        digitalWrite(ledpin, HIGH);
        break;
      default:
        Serial.println(key);
    }
  }
}
```

10. Servomotoare

În literatura sub denumirea de servomotoare se cuprind motoarele electrice executate special pentru a fi utilizate în sistemele automate de poziţionare şi care în general sunt de puteri reduse (până la puteri de ordinul câtorva [kW]).

Pentru puteri mai mari se folosesc motoarele electrice convenţionale, care sunt elemente de execuţie mai lente, cu constante de timp mai mari dar şi cu randament mai bun. Servomotoarele sunt motoare electrice speciale, de curent continuu sau curent alternativ cu viteză de rotaţie reglabilă într-o gamă largă în ambele sensuri având ca scop deplasarea într-un timp prescris a unui sistem mecanic (sarcina) de-a lungul unei traiectorii date, realizând totodată şi poziţionarea acestuia la sfârşitul cursei cu o anumită precizie.

Sistemele de reglare automată moderne impun servomotoarelor următoarele performanţe:

1. gamă largă de modificare a vitezei în ambele sensuri;
2. funcţionare stabilă la viteză foarte mică;
3. constante de timp cât mai reduse;
4. fiabilitate şi robusteţe ridicate;
5. raport cuplu/moment de inerţie cât mai mare;
6. suprasarcină dinamică admisibilă mare;
7. caracteristici de reglare liniare.

Servomotoarele electrice se folosesc în cele mai diverse aplicaţii cum ar fi acţionarea roboţilor industriali universali, a maşinilor unelte cu comandă numerică, a perifericelor de calculator, în acţionarea imprimantelor rapide, în tehnica aerospaţială, instalaţii medicale etc. În aplicaţiile enumerate, cuplul dezvoltat de servomotoare variază într-o plajă largă de valori, 0,1 ÷ 100 [Nm], cu puteri nominale ce variază în intervalul 100 [W] şi 20 [kW].

Conform principiului lor de funcţionare, servomotoarele electrice pot fi clasificate în: servomotoare de curent continuu, servomotoare asincrone şi servomotoare sincrone, în această ultimă categorie fiind incluse atât servomotoarele de curent continuu fără perii cât şi servomotoarele pas cu pas.

Servomotoarele de curent continuu se caracterizează prin posibilitatea de reglare a vitezei în limite largi, 1:10.000 şi chiar mai mult, prin intermediul unei părţi de comandă electronică relativ simplă.

Servomotoarele de curent continuu au caracteristici mecanice şi de reglaj practic liniare, cuplu de supraîncărcare mare, greutate specifică mică, moment de inerţie redus etc. Dezavantajele sunt legate de colector, fenomene de comutaţie, uzură şi scânteiere.

Servomotoarele asincrone, în prezent răspândite în tot mai mare măsură, elimină dezavantajele servomotoarelor de curent continuu legate de sistemul colector-perii, fiind de asemenea atractive prin robusteţea, simplitatea şi preţul lor. Există însă şi o serie de dezavantaje legate de randament, factor de putere, greutate şi nu în ultimul rând procedee de comandă mai complicate decât cele ale servomotorului de curent continuu.

Spre deosebire de motoarele DC, care produc rotaţie continuă cât timp 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.

Figura 10.1 Imaginea servomotorului utilizat

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 10.2 ilustrează diferite tipuri de scheme de culori.

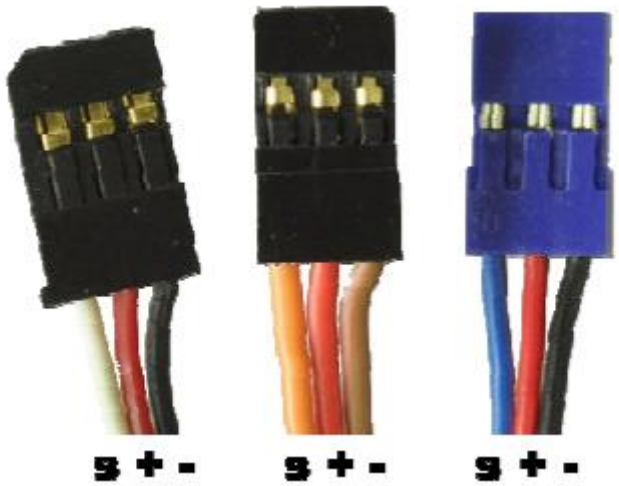


Figura 10.2 Diferite variante ale conectorului servomotorului

Semnalul de comandă seamănă foarte bine cu PWM, deosebirea fiind în felul de utilizare a impulsului, mai exact nu contează raportul ON/OFF (duttz cycle) ci doar durata impulsului.

În figura de mai jos este prezentat grafic semnalul de comandă al acestui servomecanism.

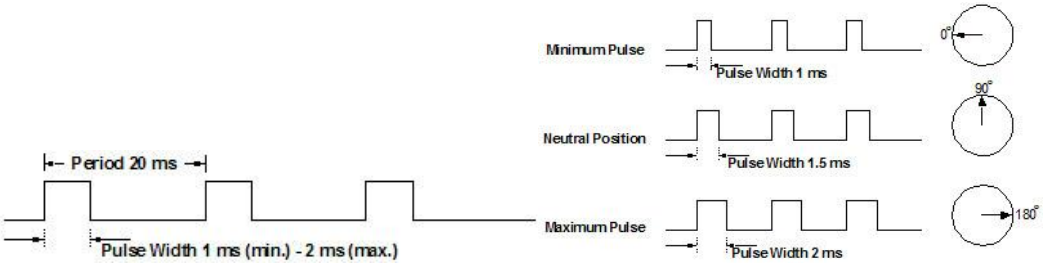


Figura 10.3 Semnale comandă servomecanism

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() – verifica daca 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 maxima intr-o directie; 180: viteza maxima in direcţia opusă; ≈ 90: oprit)

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

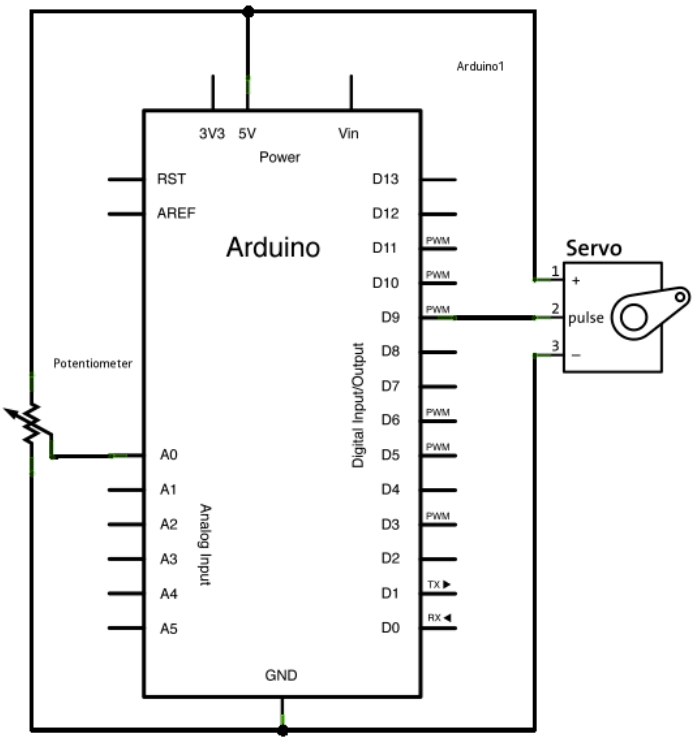


Figura 10.4 Schema electrică de conectare a componentelor pentru testarea servomotorului

```
#include <Servo.h>   //includerea bibliotecii functiilor pentru servo
Servo myServo;       //declararea myServo
int const potPin = A0; //definirea pinului A0 , cursorul potentiometrului
int potVal;          //rezultatul conversiei analog-numerice
int unghi;           //variabila pentru unghiul de rotatie al servo-ului

void setup() {
        myServo.attach(9);
        Serial.begin(9600);
}

void loop() {
potVal = analogRead(potPin); //se converteste tensiunea de la cursorul potentiometrului
Serial.print("potVal: ");
Serial.print(potVal);              //se afiseaza rezultatul conversiei la portul serial
angle = map(potVal, 0, 1023, 0, 179); //se scaleaza valoarea tensiunii din domeniul
                                      //0-1023 in 0-179 …adica din tensiune in unghiul
                                      // de rotire
Serial.print(", unghi: ");
Serial.println(unghi);             //se afiseaza unghiul calculat
myServo.write(unghi);              //se comanda rotirea servo-ului cu unghiul calculat
delay(15);
}
```

11. Măsurarea distanţelor prin metoda ultrasunetelor

Viteza sunetului

Într-un gaz ideal viteza sunetului este în funcţie de temperatură.Din fericire pe pământ comportamentul aerului este foarte aproape de comportamentul unui gaz ideal, dacă temperatura şi presiunea se menţin la un nivel standard aflat la nivelul mării.

De aceea viteza sunetului "c" în cazul unui gaz ideal în cazul nostru este:

(11.1)

unde:

$$c = \sqrt{\gamma * R} * T$$

c = viteza sunetului în metri pe secundă
γ = constanta de temperatură, pentru aer uscat γ = 1.4
R = constanta gazului, pentru aer uscat R = 286.9 N*m / (kg*K)
T = temperatura absolută (Kelvin), unde 0°C = 273.16 K
Pentru exemplu, viteza sunetului la temperatura unei camere (22°C) este de :

$$c = \sqrt{1.4 * (22 + 273.16)} * 286.9 = 344.31 \quad \text{Metri/secundă} \qquad (11.2)$$

Viteza sunetului depinde şi de tipul gazului.De exemplu atmosfera de pe Marte este compusă din 95.3% dioxid de carbon (CO2) atunci constanta de temperatură γ=1.29, iar constanta gazului R=188.9 N*m / (kg*K) atunci avem:

$$c = \sqrt{1.29 * (295.16)} * 188.9 = 268 \quad \text{Metri/secundă} \qquad (11.3)$$

Detecţia prin comparare cu o tensiune de prag

Majoritatea sistemelor de măsură cu ultrasunete se bazează pe determinarea timpului de zbor (TOF = intervalul de timp dintre momentul emisiei şi momentul recepţiei unui puls sonor).

Momentul emisiei, t_E, fiind cunoscut, rămâne de estimat momentul recepţiei, t_R, pentru a putea determina TOF.

$$TOF = t_R - t_E \qquad (11.4)$$

Metoda folosită pentru determinarea t_R constă în compararea semnalului recepţionat, *x*, cu o valoare de prag, V_P, aşa cum se prezintă în figura 3.1. Momentul, t_{RP}, în care *x* depăşeşte pentru prima dată valoarea de prag se consideră a fi t_R.

După cum rezultă din figura 1.1 această metodă oferă un estimat deplasat pentru *TOF*. Deplasarea:

$$\varepsilon = t_{RP} - t_R \qquad (11.5)$$

depinde de V_P, dar şi de forma semnalului recepţionat.
Pentru că deplasarea să fie 0 ar trebui îndeplinită cel puţin una din următoarele două condiţii:
Condiţia 1, V_P = 0 nu se poate realiza din cauza zgomotului propriu al receptorului.
Condiţia 2 , timpul de creştere al semnalului recepţionat să fie 0, nu se poate realiza din cauza răspunsului în domeniul timp al sistemului de traductoare electroacustice (emiţător - receptor).

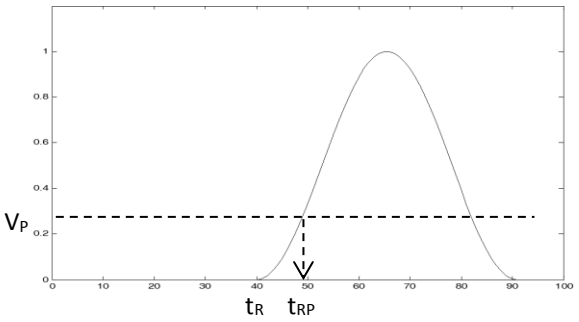


Figura 11.1. Măsurarea TOF prin metoda comparării cu o tensiune de prag.

Nivelul de presiune sonoră şi sensibilitatea

Nivelul de presiune sonoră (SPL) caracterizează performanţa acustică a traductorului emiţător în timp ce sensibilitatea (engl). sensitivity) caracterizează performanţa electrică a traductorului receptor.

Dacă notăm cu *P* presiunea sonoră într-un punct şi cu P_{REF} o valoare de presiune sonoră de referinţă, nivelul presiunii sonore se poate exprima în dB cu relaţia:

$$SPL = 20\log\frac{P}{P_{REF}} \tag{11.6}$$

În acustică se obişnuieşte a se lua drept valoare de referinţă pentru presiunea sonoră corespunzătoare pragului de audibilitate:

$$P_{REF} = 0.0002\times10^{-6}\,bar \tag{11.7}$$

Conform [4], pentru o tensiune de excitaţie de 40kHz şi 10V_{ef} la distanţa de 30cm faţă de traductor nivelul presiunii sonore va fi de minimum 120dB. Din (11.6) şi (11.7), valoarea absolută a presiunii sonore, va fi în aceste condiţii:

$$P = 0.0002\,bar \tag{11.8}$$

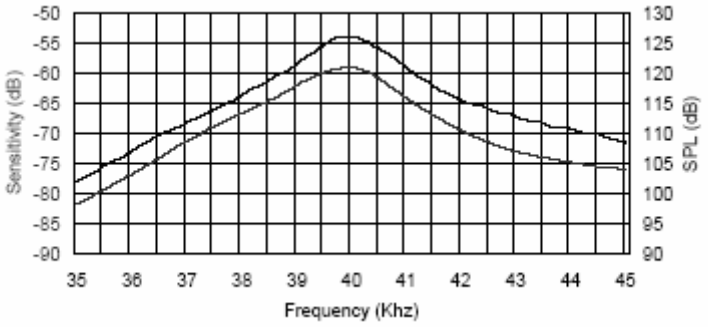


Fig 11.2. Dependenta de frecventa a sensibilităţii

În figura 11.2 se poate urmări dependenţa de frecvenţă a SPL pentru MA 400 ST (curba din partea de sus a graficului). Se poate remarca faptul că în jurul frecvenţei centrale (40±0,5kHz), nivelul presiunii sonore este de peste 125dB, în timp ce pentru o abatere faţă de f_C de ±2,5kHz, aceasta se reduce deja la 115dB. Pentru un randament bun de conversie a energiei electrice în energie acuşţică este important că traductorul să fie excitat la o frecvenţă cât mai apropiată de frecvenţa centrală.

Nivelul presiunii sonore va fi cu atât mai mare cu cât valoarea efectivă a tensiunii de excitaţie va fi mai mare. Într-adevăr, dacă notăm cu I intensitatea sonoră avem:

$$P \propto \sqrt{I} \quad \wedge \quad I \propto {V_{ef}}^2 \quad \Rightarrow \quad P \propto V_{ef} \tag{11.9}$$

Dependenţa dintre SPL şi distanţă nu este la fel de simplă. Atenuarea sunetului în aer depinde de umiditate şi temperatură şi poate fi determinată pe baza unor relaţii empirice, cum ar fi relaţia (1.10), dedusă pe baza rezultatelor experimentale de Knudsen şi Harris:

$$\alpha_A = \left(\frac{f}{1000}\right)^{3/2} \times \frac{0.283}{20+\Phi_t} \qquad (11.10)$$

unde: α_t este coeficientul de atenuare în dB/m
f este frecvenţa în Hz
$\Phi_t = \Phi_{20}(1+0{,}067\Delta t)$
Φ_{20} este umiditatea relativă la 20^0C în %
Δt este diferenţa de temperatură faţă de 20^0C.

A doua curbă (ce din partea de jos) din figura 11.3 reprezintă graficul variaţiei sensibilităţii în funcţie de frecvenţă.

Sensibilitatea este un parametru caracteristic traductoarelor receptoare (MA 400 SR). Ea se defineşte la frecvenţa centrală şi este amplitudinea tensiunii de mers în gol la bornele traductorului pentru o presiune acustică dată. În [4] este specificată valoarea -65dB la 40kHz şi valorile de referinţă de 1V pentru tensiune şi 1µbar pentru presiune.

Această informaţie se interpretează în felul următor : tensiunea de ieşire, în gol, la bornele MA 400 SR pentru 1µbar presiune acustică la intrare va fi dată de relaţia:

$$\frac{20\log\left(\frac{v_0}{1V}\right)}{1\mu bar} = \frac{-65\,dB}{1\mu bar} \quad \Rightarrow \quad 20\log\frac{v_0}{1V} = -65dB \quad \Rightarrow \quad v_0 = 0.562\,mV\ la\ 1\mu bar \qquad (11.11)$$

Pe baza relaţiilor (1.10) şi (1.11) se poate deduce valoarea tensiunii de mers în gol a receptorului la distanţa de 30cm faţă de emiţător, acesta din urmă fiind excitat la 40kHz cu o tensiune de $10V_{ef}$. În aceste condiţii presiunea exercitată asupra receptorului este de 200µbar (vezi relaţia 1.9) şi prin urmare:

$$\frac{20\log\left(\frac{v_0}{1V}\right)}{200\mu bar} = \frac{-65}{200} \quad \Rightarrow \quad v_0 = 10^{\frac{-65}{20\cdot 200}} \quad \Rightarrow \quad v_0 = 0.963V \quad la\ 200\mu bar \qquad (11.12)$$

Această valoare a fost confirmată de rezultate experimentale .

Unghiul de emisie / recepţie

Unghiul de emisie pentru traductoarele MA 400 ST este parametrul care caracterizează traductorul din punctul de vedere al variaţiei SPL în funcţie de abaterea faţă de normala dusă în centrul traductorului.

Considerând ca valoare de referinţă (0dB) nivelul presiunii într-un punct situat la distanţa *x* pe normala dusă în centrul traductorului, în conformitate cu datele de catalog, un punct situat la aceeaşi distanţă *x* faţă de traductor dar pe o dreaptă care face cu normala un unghi de ± 30^0 va avea SPL cu 6dB mai mic. Acesta este sensul în care, în catalog, este specificat un unghi de emisie de 60^0 la -6dB.

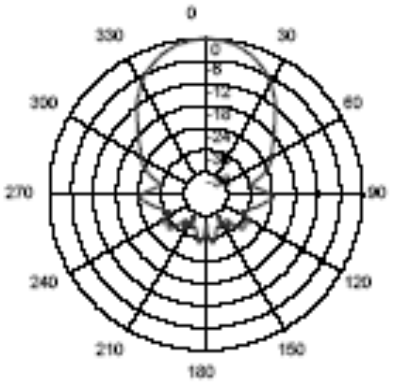


Figura 11.3. Unghiul de emisie / recepţie pentru traductoarele MA400 ST / R

Unghiul de recepţie, pentru traductoarele MA 400 SR, caracterizează traductorul din punctul de vedere al variaţiei tensiunii de ieşire în funcţie de abaterea sursei faţă de normala dusă în centrul traductorului. Considerând ca valoare de referinţă (0dB) nivelul de răspuns la o sursă situată la distanţa *x* pe normala dusă în centrul traductorului, în conformitate cu datele de catalog, o sursă situată la aceeaşi distanţă *x* dar pe o dreaptă care face cu normala un unghi de ± 30^0 va produce un nivel de

ieşire cu 6dB mai mic. Acesta este sensul în care în catalog este specificat un unghi de recepţie de 60^0 la -6dB. Graficul variaţiei SPL respectiv al sensibilităţii în funcţie de unghi este prezentat în figura 1.3.

Influenţa temperaturii asupra parametrilor traductoarelor

Temperatura influenţează valorile SPL, sensibilitatea şi frecvenţa centrală. Modul de variaţie al acestor parametrii se prezintă în figura 1.4.

După cum se poate vedea în figura 1.4.a, influenţa temperaturii asupra sensibilităţii receptorului nu este semnificativă. Se constată o scădere a sensibilităţii cu 2 – 3dB pe tot domeniul de temperaturi (-30^0C … $+80^0$C) de funcţionare admis [4]. Tendinţa de scădere este mai accentuată la temperaturi mai mari decât 50^0. În cazul utilizării în aer liber atingerea acestui prag este improbabilă.

În cazul MA 400 ST, după cum se vede din graficul 1.4.b, SPL este practic constant în intervalul de temperaturi 30^0…80^0C. În afara acestui interval, se constată o scădere a nivelului presiunii sonore o dată cu scăderea temperaturii. În intervalul de temperaturi de funcţionare admis, (-30^0C … $+80^0$C), diferenţa de nivel nu depăşeşte 3dB.

Temperatura în fluenţează de asemenea şi valoarea frecvenţei centrale a traductoarelor. Atât pentru MA 400 ST cât şi pentru MA 400 SR, frecvenţa centrală scăde cu

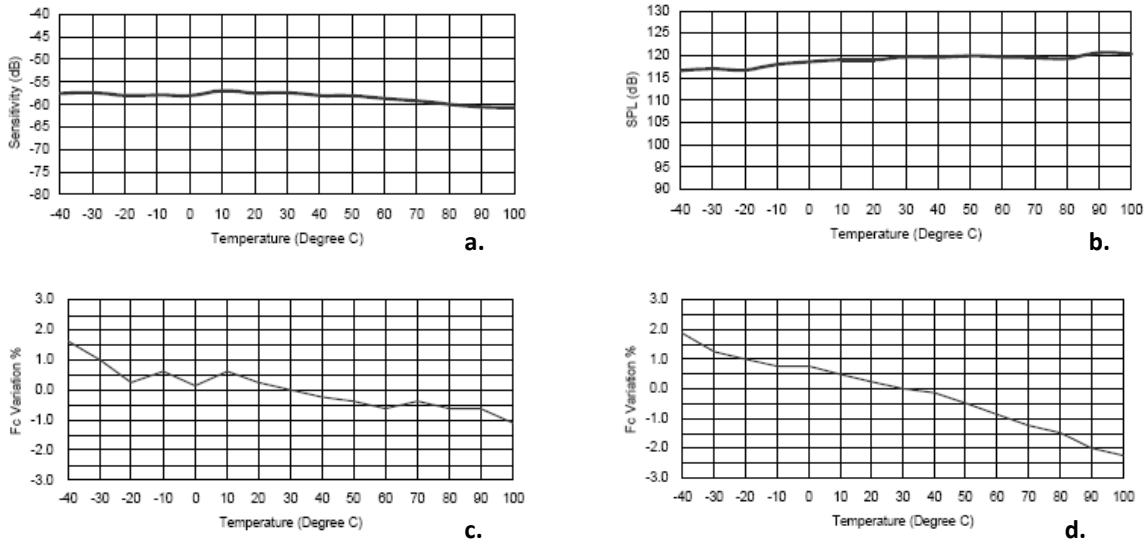


Figura 11.4. Influenţa temperaturii asupra: **a.** sensibilităţii MA 400 SR **b.** frecvenţei centrale MA 400 SR **c.** SPL MA 400 ST **d.** frecvenţei centrale MA 400 ST

creşterea temperaturii. Variaţia relativă maximă a frecvenţei nu este mai mare de +1,5%…-0,5% pentru receptor respectiv +2%...-2,5% pentru traductorul emiţător. În intervalul de temperaturi de uţilizare recomandat de producător [4], această variaţie se încădrează în limitele +1%…-0,5% pentru MA 400 SR, respectiv +1,5%…-1,5% pentru MA 400 ST.

Pe baza celor prezentate se pot trage următoarele concluzii cu privire la utilizarea traductoarelor din seria MA400 la măsurarea nivelului prin metoda puls-ecou.

1. Traductoarele din seria MA400 sunt destinate să lucreze în aer, într-un mediu normal, necontaminat (ne-coroziv). Frecvenţa lor centrală este de 40kHz iar banda la -6dB este de ±2kHz pentru traductoarele MA400ST respectiv ±2,5kHz pentru traductoarele MA400SR. Sunt prin urmare traductoare de bandă îngustă. Din punctul de vedere al utilizării acest lucru este important deoarece ele pot fi excitate cu (pot genera) semnale de frecvenţă constantă şi foarte apropiată de frecvenţa centrală. Este prin urmare exclusă posibilitatea utilizării unor semnale speciale (secvenţe codate) de bandă largă pentru excitarea acestor traductoare.
2. Banda mai largă a traductoarelor MA400SR impune folosirea lor ca receptoare în timp ce banda mai îngustă a traductoarelor MA400ST le recomandă ca emiţătoare. Altfel, efectul piezoelectric fiind reversibil, oricare dintre cele două traductoare poate fi folosit pe post de

emiţător sau receptor, mai mult, un singur traductor poate fi folosit alternativ pentru emisie şi recepţie. Utilizarea lor conform destinaţiei date de fabricant, (T – emiţător, R - receptor) conduce, aşa cum s-a demonstrat experimental în primul referat la cel mai bun raport de conversie electric – acustic – electric.

3. Nivelul presiunii sonore care se obţine le 30cm de emiţător, în condiţiile în care acesta este excitat pe frecvenţa centrală cu o tensiune de $10V_{ef}$ este de 120dB. Din acest parametru de catalog se poate calcula care ar fi nivelul presiunii sonore la altă valoare a tensiunii de excitaţie (aşa cum s-a demonstrat în paragraful 1 există o relaţie de proporţionalitate între valoarea efectivă a tensiunii de excitaţie şi presiune). Din păcate această informaţie nu este suficientă pentru a determina nivelul presiunii sonore la altă distanţă faţă de emiţător. Mai ales în condiţiile în care propagarea are loc în spaţii închise şi presiunea într-un punct este dată de suma mai multor reflexii defazate, acest nivel nu poate fi dedus din datele de catalog. Determinările experimentale reprezintă singura soluţie.

4. Sensibilitatea receptorului este un parametru foarte important de care depinde în ultimă instanţă raza maximă de acţiune a sistemului. Nivelul de -65dB dat de catalog se raportează la o tensiune de referinţă de 1V şi la o presiune de referinţă de 1µbar. Această dublă raportare face ca parametrul să fie greu de utilizat deoarece nu se cunoşte distanţa la care emiţătorul (cel puţin în anumite condiţii) este capabil să asigure această presiune. Prin urmare acest parametru este util numai pentru compararea diferitelor tipuri de receptoare între ele. Raza de acţiune a unui sistem de emisie recepţie în anumite condiţii date se poate determina cu precizie numai experimental.

5. Unghiul de emisie (recepţie) al acestor traductoare este larg (aproximativ 60^0 la -6dB) ceea ce face ca în cazul utilizării în spaţii închise unda emisă respectiv recepţionată să fie compusă din foarte multe vibraţii de aceeaşi frecvenţă dar de amplitudini şi faze diferite. Pe lângă dezavantajele evidente pe care ecourile suprapuse le produc din punctul de vedere al detecţiei, acest fenomen prezintă şi avantaje: în conformitate cu legea numerelor mari, dacă numărul reflexiilor este suficient de mare, amplitudinea şi faza vibraţiilor care se însumează poate fi considerată aleatoare, cu distribuţia normală, şi în consecinţă probabilitatea de recepţie a unui ecou este aceeaşi în orice punct de pe suprafaţa unei secţiuni transversale a tubului. Aceasta face ca poziţionarea traductoarelor să nu fie critică.

Pentru laborator se va utiliza un modul HC-SR04, modul care conţine şi emiţătorul de ultrasunete şi partea de recepţie (microfonul), circuitul fiind proiectat să lucreze cu semnale logice TTL, astfel se poate conecta direct la pinii platformei Arduino

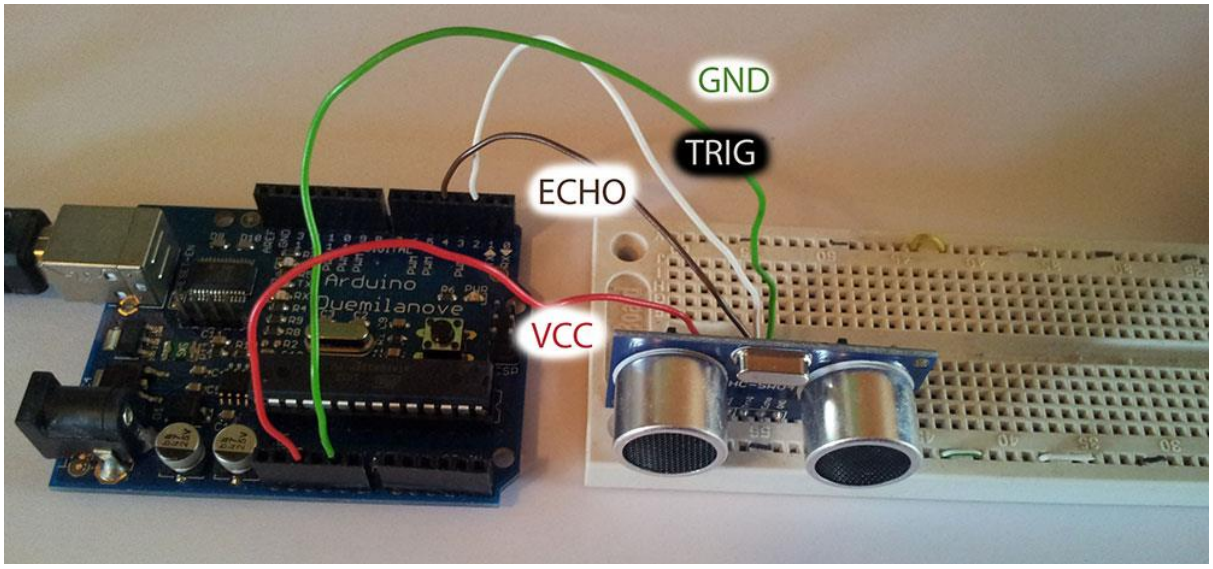


Figura 11.5 Conectarea modulului ultrasunete la Arduino Uno

CC de la senzor se cupleaza la +5V de pe sistemul Arduino Uno.
TRIG de la senzor se cuplează la unul din pinii digitali de pe sistemul Arduino Uno (în cod este setat pinul 2)
ECHO de la senzor se cuplează la un alt pin digital de pe sistemul Arduino Uno (în cod este setat pinul 3)
GND de la senzor se cuplează la unul din pinii GND de la Arduino Uno.
Programul de testare este descris mai jos, furnizarea valorilor măsurate şi a distanţei calculate fiind realizate prin intermediul comunicaţiei seriale cu calculatorul.
Se vor realiza conexiunile descrise anterior urmând a fi copiat programul de mai jos în Arduino IDE, se va compila şi încărca programul rezultat şi se vor urmării valorile rezultate, testînd pentru diferite distanţe. Domeniul de măsură este de 10cm – 400cm, experimental rezultând o precizie mai mare în intervalul 30cm – 300cm.

```
/*
* Senzor: HC-SR04
* Pini senzor  | Conectare la platforma Arduino
* VCC la 5V
* TRIG la Digital pin 2
* ECHO la Digital pin 3
* GND la GND
Acest program realizeaza masurarea distantei pina la cel mai apropiat obiect si
returneaza valoarea distantei
*/
/*
A fost preluat si adaptat de la:
 Original code for Ping))) example was created by David A. Mellis
   Adapted for HC-SR04 by Tautvidas Sipavicius
   This example code is in the public domain.
 */
// https://www.tautvidas.com/blog/2012/08/distance-sensing-with-ultrasonic-sensor-and-arduino/
//definirea pinilor
const int trigPin = 2;
```

```
const int echoPin = 3;

void setup() {
// se seteaza pinii de intrare si de iesire
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  // se initializeaza comunicatia seriala
  Serial.begin(9600);
}

void loop()
{
  //variabilele pentru durata impulsului
  // si pentru rezultatul distantei in centimetri
  long duration, cm;

//Procesul de masurare este pornit prin generarea unui impuls HIGH cu o durata
//de minim 10 microsecunde
//pentru a fi asigurata o forma corecta a impulsului, pinul de Trigger este adus in starea LOW
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

//masurarea timpului de la emiterea impulsului pina la receptionarea ecoului
//durata este masurata in microsecunde

//pulseIn() este o fucntie definita in Arduino Ide  care permite masurarea duratei unui impuls
//masoara cu acuratete mai mare impulsuri cu o durata mai mare de 10usec pina la 3 minute
//returneaza valoarea in microsecunde

  duration = pulseIn(echoPin, HIGH);

  // conversia timp in distanta
  cm = microsecondsToCentimeters(duration);

  Serial.print(cm);
  Serial.print("cm");
  Serial.println();

  delay(100);
}


long microsecondsToCentimeters(long microseconds)
{
//viteza sunetului este de 340m/s sau 29 microsecunde / 1cm
//impulsul parcurge distanta de masurat de doua ori, dus si intors
//prin urmare durata pina la aparitia ecoului presupune distanta
//de la emitator – obiect – receptor, adica se va masura dublul distantei
//pina la obiect
  return microseconds / 29 / 2;
}
 //
```

Se vor realiza măsurători pentru mai multe distanţe (pentru referinţă se va utiliza o ruletă) iar ulterior se va schimba durata impulsului semnalului ultrasonor şi se vor compara rezultatele (pentru o aceeaşi distanţa, practic se va fixa senzorul într-o poziţie, urmând a se schimba doar programul din sistemul Arduino Uno).

12. Comunicaţia radio – nRf24L01

Modului de transmisie nRF24L01 este un cip de transmisie radio de putere mică care lucrează la o frecvenţă de 2.4 GHz potrivit pentru aplicațiile fără fir.

Modulul wireless nRF24L01 de bază este produs de compania Nordic în Norvegia și este unul dintre cele mai recente module wireless de înaltă performanță. nRF24L01 este un cip de transmisie de date cu un design profesional care poate crește banda PA de mare putere și LNA, RF care schimbă filtrul de bandă compusă în întregime.

Necesită o alimentare de 3.3V, antena încorporată poate transmite date pâna la 100m, iar aceasta se poate extinde pana la 1km cu ajutorul unei antene suplimentare.

nRF24L01 poate fi operat și configurat cu ajutorul interfeței periferice seriale SPI. Registrele pot fi accesate prin SPI, conținând toate configurațiile regiștrilor din nRF24L01 și este accesibil tuturor modurilor de operare ale cipului. Parametrii pot fi configurați de către utilizator ca și canal de frecvență și suportă o rată de transmisie a datelor de 250 kbps, 1 Mbps și 2 Mbps. Rata ridicată de transmisie a datelor combinată cu două moduri de economisiere a energiei face ca nRF24L01 să fie potrivit pentru modelele de putere foarte scazută.

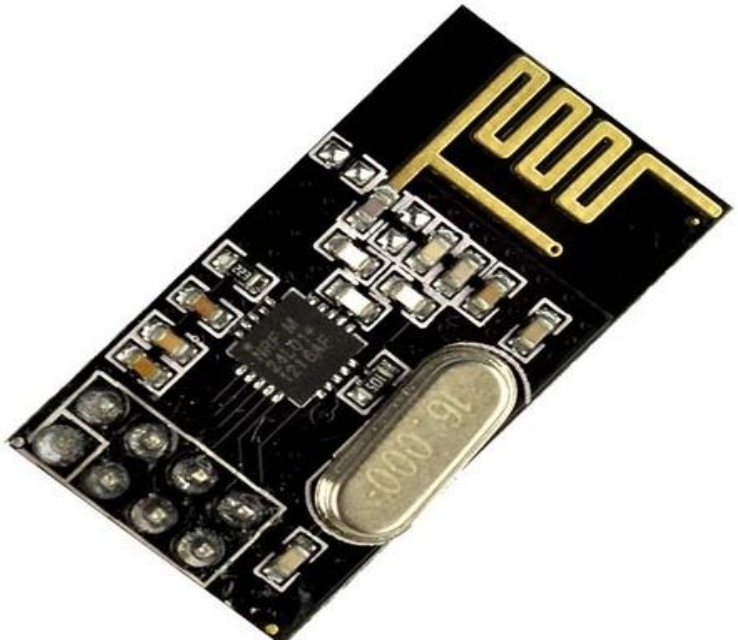


Figura 12.1 Modulul de transmisie nRF24L01

Modulul nRF24L01 are în componența sa un dispozitiv care controlează tranzițiile dintre modurile de operare ale cipului. Acest dispozitiv preia valorile intărilor definite de utilizator și semnalele interne.

Modulul nRF24L01 permite configurarea modurilor power down, de repaus, Rx și Tx.

Modul power-down este dezactivat folosindu-se un consum scăzut de curent. Toate valorile registrelor sunt disponibile și SPI este păstrat activ, activând modificareaa configurației și încărcarea și descărcarea registrelor de date.

Modul de repaus micșorează consumul obișnuit de curent menținându-se durate de porniri scurte. În acest mod doar o parte a cristalului oscilator este activ. Schimbarea la modurile active se produce doar atunci când CE este setat la maxim sau la minin, nRF24L01 revine la modul de repaus de la Rx și Tx.

Modul Rx este un mod activ unde nRF24L01 este utilizat ca receptor. În acest mod receptorul demodulează semnalalele de la canalul RF, prezentând constant datele demodulate protocolului lățimii de bandă, unde acesta caută constant un pachet valid. Dacă se găsește un pachet valid, conținutul pachetului este introdus într-un spațiu liber în Rx FIFO. Dacă Rx FIFO este plin pachetul primit este eliminat.

Received Power Detector (RPD) este un semnal setat la maxim când un semnal RF mai puternic de -64dBm este detectat în interiorul receptorului canalului de frecvență. Semnalul RPD intern este filtrat înainte să fie introdus în registrul RPD. Semnalul RF trebuie să fie prezent pentru cel puțin 40 us înainte ca RPD să fie setat la maxim.

Modul Tx este un mod activ de transmitere a pachetelor. nRF24L01 rămâne în modul Tx pâna când se finalizează transmiterea pachetului. Dacă CE=0, nRF24L01 revine la modul de repaus. Dacă CE=1 statusul lui Tx FIFO determină următoarea acțiune. Dacă Tx FIFO nu este gol, nRF24L01 rămâne în modul Tx și transmite următorul pachet. Dacă Tx FIFO este gol, nRF24L01 intră în modul repaus. PLL operează liber în modul Tx. Este important să nu ținem niciodată transmițătorul nRF24L01 în modul Tx mai mult de 4 ms odată.

Rata de transmisie a datelor a lui nRF24L01 dă o sensibilitate mai bună de recepție decât cantitatea de date care poate fi procesată. Transmițătorul și receptorul trrebuie să fie programate cu aceeași rată de transmisie de date prin aer pentru a comunica cele 2 module.

Frecvența canalului RF determină centrul canalului folosit de nRF24L01. Canalul ocupă o lățime de bandă mai mică de 1MHz la 250Kbps și 1Mbps la o lățime de bandă mai mică de 2 MHz. [7]

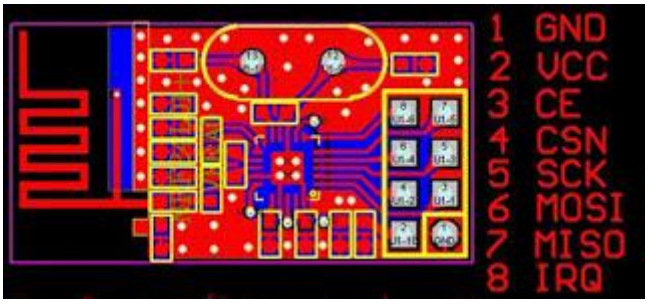


Figura 12.2 Dispunerea pinilor nRF24L01

Pin	Nume	Funcție	Descriere
1	CE	DI	Pin de activare a modului Rx si Tx
2	CSN	DI	Pin selecție SPI
3	SCK	DI	Ceas SPI
4	MOSI	DI	Intrare date Slave SPI
5	MISO	DO	Iesire date Slave SPI
6	Irq	DO	Pin intrerupere
7	VDD	PWR	Alimentare (+1.9V- +3.6V)
8	XC2	PWR	Pamântare
9	XC1	AO	Pinul 2 a cristalului
10	VDD_PA	AI	Pinul 1 a cristalului
11	ANT1	PWRO	Ieșire de alimentare
12	ANT2	RF	Interfata antenei 1
13	VSS	RF	Interfata antenei 2
14	VDD	PWR	Pamantare
15	IREF	PWR	Alimentare
16	VSS	AI	Curent de referință
17	SVDD	PWR	Pamantare
18	DVDD	PWR	Alimentare (+1.9V- +3.6V)
19	VSS	PWRO	Iesire digitala interna
20		PWR	Pamantare

Descrierea pinii modulului nRF24L01

În figura 12.3 şi 12.4 sunt prezentate schemele electrice a două sisteme Arduino Uno care comunică între ele utilizând aceste module radio. În figura 12.3 este prezentat modulul master iar în figura 12.4 este prezentat modulul slave.

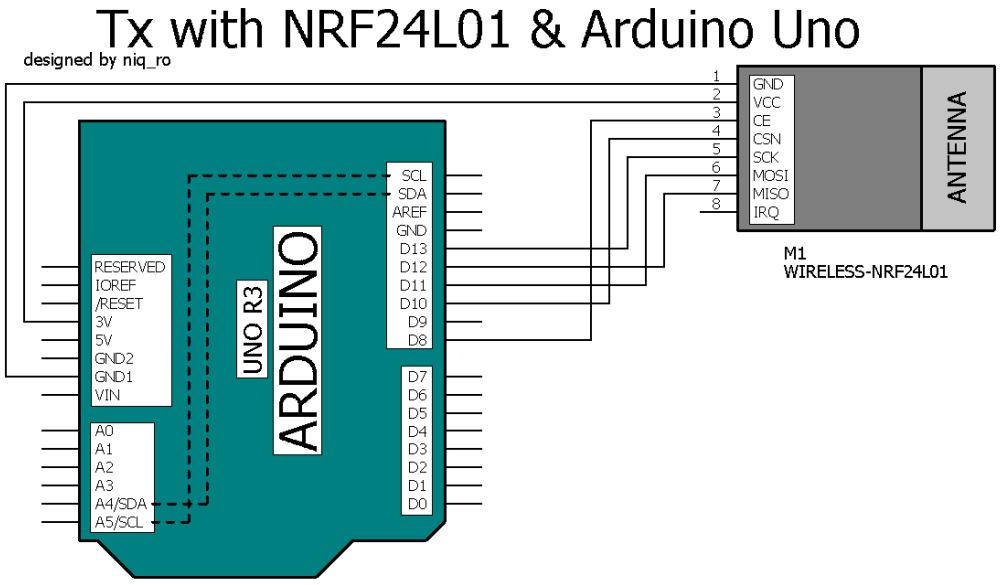


Figura 12.3 Schema de conectare a nRF24L01 la „emiţător”

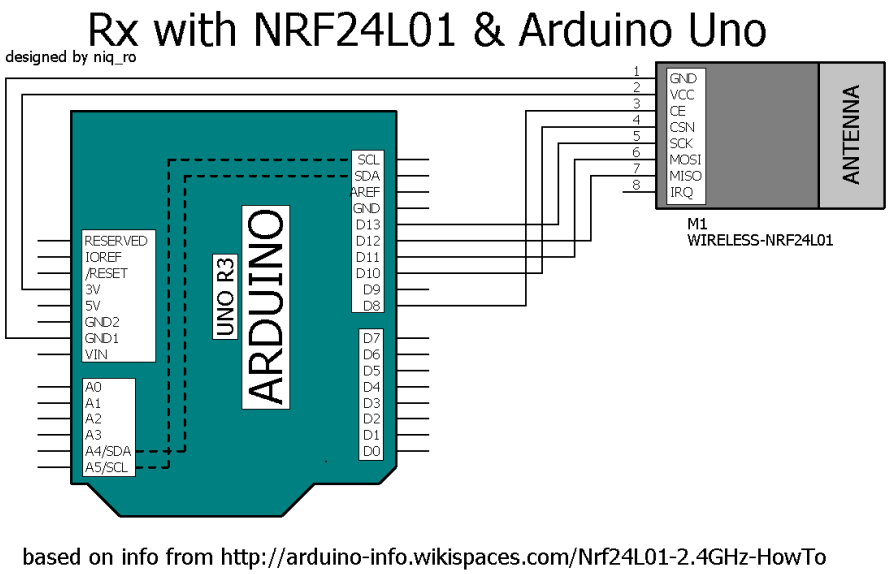


Figura 12.4 Schema de conectare a nRF24L01 la „receptor”

Deoarece, în cazul acestor module radio, există numeroase biblioteci de funcţii care utilizează aceste module, este recomandată să se ia ca punct de plecare în scrierea programului, unul din exemplele care sunt incluse în biblioteca de funcţii respectivă. Chiar dacă sunt biblioteci care au acelaşi nume sau funcţiile utilizate au aceleaşi denumiri, exisată şanse mari ca ele să fie diferite în ceea ce priveşte parametrii utilizaţi, facilităţile pe care le oferă, deci e nevoie de atenţie sporită în utilizarea acestora.

Pentru utilizarea acestor module se va utiliza biblioteca RadioHead (http://www.airspayce.com/mikem/arduino/RadioHead/RadioHead-1.46.zip)

Programele utilizate (se regasesc in exemplelel din biblioteca) sunt prezentate mai jos:

```
// nrf24_reliable_datagram_server.pde
// -*- mode: C++ -*-
// Example sketch showing how to create a simple addressed, reliable messaging server
// with the RHReliableDatagram class, using the RH_NRF24 driver to control a NRF24 radio.
// It is designed to work with the other example nrf24_reliable_datagram_client
// Tested on Uno with Sparkfun WRL-00691 NRF24L01 module
// Tested on Teensy with Sparkfun WRL-00691 NRF24L01 module
// Tested on Anarduino Mini (http://www.anarduino.com/mini/) with RFM73 module
// Tested on Arduino Mega with Sparkfun WRL-00691 NRF25L01 module

#include <RHReliableDatagram.h>
#include <RH_NRF24.h>
#include <SPI.h>

#define CLIENT_ADDRESS 1
```

```
#define SERVER_ADDRESS 2

// Singleton instance of the radio driver
RH_NRF24 driver;
// RH_NRF24 driver(8, 7);   // For RFM73 on Anarduino Mini

// Class to manage message delivery and receipt, using the driver declared above
RHReliableDatagram manager(driver, SERVER_ADDRESS);

void setup()
{
  Serial.begin(9600);
  if (!manager.init())
    Serial.println("init failed");
  // Defaults after init are 2.402 GHz (channel 2), 2Mbps, 0dBm
}

uint8_t data[] = "And hello back to you";
// Dont put this on the stack:
uint8_t buf[RH_NRF24_MAX_MESSAGE_LEN];

void loop()
{
  if (manager.available())
  {
    // Wait for a message addressed to us from the client
    uint8_t len = sizeof(buf);
    uint8_t from;
    if (manager.recvfromAck(buf, &len, &from))
    {
      Serial.print("got request from : 0x");
      Serial.print(from, HEX);
      Serial.print(": ");
      Serial.println((char*)buf);

      // Send a reply back to the originator client
      if (!manager.sendtoWait(data, sizeof(data), from))
        Serial.println("sendtoWait failed");
    }
  }
}
```

```
// nrf24_reliable_datagram_client.pde
// -*- mode: C++ -*-
// Example sketch showing how to create a simple addressed, reliable messaging client
// with the RHReliableDatagram class, using the RH_NRF24 driver to control a NRF24 radio.
// It is designed to work with the other example nrf24_reliable_datagram_server
// Tested on Uno with Sparkfun WRL-00691 NRF24L01 module
// Tested on Teensy with Sparkfun WRL-00691 NRF24L01 module
// Tested on Anarduino Mini (http://www.anarduino.com/mini/) with RFM73 module
// Tested on Arduino Mega with Sparkfun WRL-00691 NRF25L01 module

#include <RHReliableDatagram.h>
```

```
#include <RH_NRF24.h>
#include <SPI.h>

#define CLIENT_ADDRESS 1
#define SERVER_ADDRESS 2

// Singleton instance of the radio driver
RH_NRF24 driver;
// RH_NRF24 driver(8, 7);   // For RFM73 on Anarduino Mini

// Class to manage message delivery and receipt, using the driver declared above
RHReliableDatagram manager(driver, CLIENT_ADDRESS);

void setup()
{
  Serial.begin(9600);
  if (!manager.init())
    Serial.println("init failed");
  // Defaults after init are 2.402 GHz (channel 2), 2Mbps, 0dBm
}

uint8_t data[] = "Hello World!";
// Dont put this on the stack:
uint8_t buf[RH_NRF24_MAX_MESSAGE_LEN];

void loop()
{
  Serial.println("Sending to nrf24_reliable_datagram_server");

  // Send a message to manager_server
  if (manager.sendtoWait(data, sizeof(data), SERVER_ADDRESS))
  {
    // Now wait for a reply from the server
    uint8_t len = sizeof(buf);
    uint8_t from;
    if (manager.recvfromAckTimeout(buf, &len, 2000, &from))
    {
      Serial.print("got reply from : 0x");
      Serial.print(from, HEX);
      Serial.print(": ");
      Serial.println((char*)buf);
    }
    else
    {
      Serial.println("No reply, is nrf24_reliable_datagram_server running?");
    }
  }
  else
    Serial.println("sendtoWait failed");
  delay(500);
}
```

Bibliografie

[1] https://www.arduino.cc/en/Tutorial/Blink

[2] https://www.arduino.cc/en/Tutorial/DigitalPins

[3] https://github.com/arduino/Arduino

[4] http://users.utcluj.ro/~rdanescu/pmp-lab05.pdf Interfete de comunicatie

[5] https://profs.info.uaic.ro/~vcosmin/pagini/resurse_arduino/Cursuri_2016/8/Arduino_8.pdf

[6] https://www.arduino.cc/en/Reference/Wire pt i2c

[7] https://www.arduino.cc/en/Tutorial/MasterReader i2c master

[8] https://www.arduino.cc/en/Tutorial/MasterWriter i2c

[9] https://howtomechatronics.com/tutorials/arduino/how-i2c-communication-works-and-how-to-use-it-with-arduino/

[10] https://www.arduino.cc/en/Reference/SPI

[11] https://www.arduino.cc/en/Tutorial/SPITransaction

[12] https://tronixstuff.com/2011/05/13/tutorial-arduino-and-the-spi-bus/

[13] http://playground.arduino.cc/Main/KeypadTutorial

[14] https://www.arduino.cc/en/Tutorial/PWM

[15] https://www.arduino.cc/en/Tutorial/SecretsOfArduinoPWM

[16] https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/

[17] https://www.arduino.cc/en/Tutorial/HelloWorld?from=Tutorial.LiquidCrystal

[18] http://users.utcluj.ro/~rdanescu/pmp-lab03.pdf

[19] https://profs.info.uaic.ro/~arduino/index.php/Laboratorul_5

[20] http://mce.utcluj.ro/lucrari/PWM.doc

[21] http://cs.curs.pub.ro/wiki/pm/lab/lab3

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

[23] https://www.tutorialspoint.com/arduino/arduino_serial_peripheral_interface.htm

[24] https://www.arduino.cc/en/Tutorial/SPIEEPROM

[25] https://www.arduino.cc/en/Reference/SoftwareSerial

[26] http://www.roroid.ro/5-butoane-pe-o-intrare-analogica/

[27] https://playground.arduino.cc/Main/KeypadTutorial