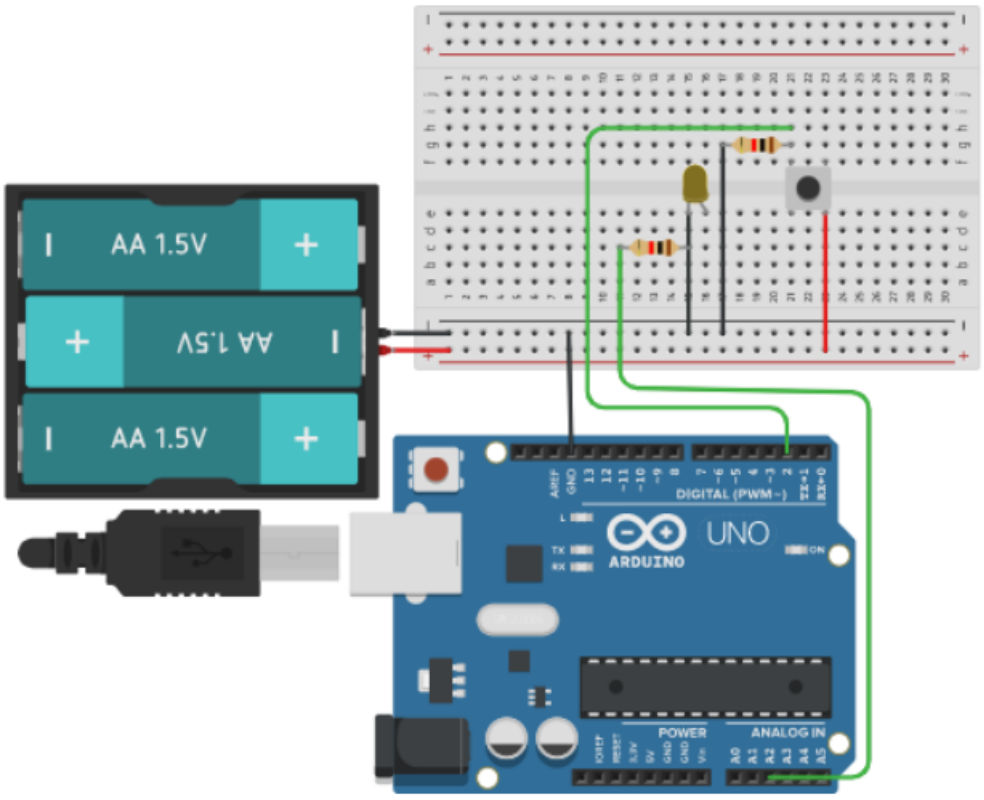
Sisteme încorporate: fundamentele utilizării sistemelor cu microcontrolere

http://rovislab.com

Gigel Măceșanu, Tiberiu Cociaș, Sorin Grigorescu



Editura Universității Transilvania din Brașov

2019

Prefaţă

Evoluţia tehnologică din domeniul sistemelor integrate a făcut posibilă apariția multor module care facilitează dezvoltarea de aplicații care au la bază microcontrolere. Acest manual suport de curs facilitează înțelegerea componentelor hardware și software care pot fi interconectate pentru dezvoltarea diverselor tipuri de aplicații. Facilitarea dezvoltării aplicațiilor este susținută și de utilizarea platformei integrate Arduino, care reprezintă unul dintre cele mai utilizate module de acest fel, atât în mediul academic cât și în cel privat sau industrial.

Lucrarea de față se adresează acelor persoane care sunt interesate de domeniul sistemelor încorporate, care utilizează pentru procesarea datelor un microcontroler. Partea teoretică a materialului este orientată spre două direcții. Prima direcție constituie explicarea componentelor electronice și a modului de interconectare a acestora. Sunt introduse noțiuni despre partea hardware care poate fi utilizată în astfel de aplicații. Componenta software reprezintă cea de a doua direcție. Aceasta este detaliată astfel încât să se creeze imaginea de ansamblu a modului de funcționare al tuturor modulelor importante care intră în componența unui microcontroler. Pentru o mai buna înțelegere, fiecare capitol este însoțit de o aplicație practică care să fundamenteze cunoștințele teoretice prezentate.

Prima parte a cărții își propune să facă o recapitulare a cunoştinţelor de programare în limbajul C, întrucât toate celelalte capitole au la bază acest limbaj pentru dezvoltarea aplicațiilor. Capitolul al doilea prezintă mecanismul de transfer al datelor, către microcontroler sau în exteriorul acestuia, atunci când sunt folosiți pinii generali de date. În capitolul al treilea este prezentat sistemul de întreruperi, urmând ca în capitolul al patrulea să se prezinte cum funcționează modulul de timp timer al unui microcontroler. Având în vedere numărul mare de senzori analogici, este necesară prezentarea în capitolul cinci a modulului de conversie analog numerică. Comunicația serială este prezentată în capitolul șase, pentru transferul de date utilizând un canal specific de comunicație. Ultimul capitol este dedicat

mai multor exemple practice, care prin diversitate și dificultate încearcă să acopere subiecte de interes, care pot fi tratate cu un astfel de sistem cu microcontroler.

Autorii aduc pe aceasta cale mulțumiri domnilor profesori Claudiu Pozna și Sorin Moraru, referenții științifici ai prezentei lucrări, care, prin citirea atentă a manuscrisului, a observaţiilor și sugestiilor făcute, au contribuit la finalizarea acestei lucrări în forma în care autorii o prezintă cititorilor.

Autorii doresc să mulțumească membrilor și studenților grupului de cercetare *Robotics, Vision and Control Laboratory* (ROVIS) `http://rovislab.com`, din care fac parte și autorii cărții, pentru sprijinul acordat pe parcursul ultimilor ani.

Braşov, Octombrie 2019 Autorii

1. Introducere

Introducere în limbajul C
Arduino Software (IDE)
Arduino Hardware

Dezvoltarea aplicaţiilor care au la bază un microcontroler a crescut în ultimii ani, în special datorită introducerii de noi platforme integrate. Una dintre acestea este și platforma cu microcontroler *Arduino*. Aceasta permite dezvoltarea de aplicații software pentru microcontrolere într-un mod mult mai simplu decât în variantele clasice. Acest lucru se datorează în primul rând integrării de funcționalități specifice, care permit achiziția de informații din mediul exterior și reacția corespunzătoare a platformei.

Arduino este platformă electronică gratuită (eng. open-source) bazată pe un mecanism software și hardware ușor de utilizat. Plăcile de dezvoltare de acest fel permit citirea de date de la diferite tipuri de intrări (de ex.: senzori de lumină, butoane, informații de pe internet) și le transformă în mărimi de ieșire, care pot controla motoare, aprinde LED-uri sau publica date pe internet.

Locul unde a fost dezvoltat pentru prima oară *Arduino*, în 2003, este *Ivrea Interaction Design Institute* (în Italia). Scopul a fost acela de a se realiza prototipuri, cât mai repede și mai ușor, de către studenți fără cunoștințe de programare sau electronică. Odată câștigată popularitatea, placile de dezvoltare Arduino au început sa fie adaptate și utilizate în diferite produse sau servicii: Internet of Things (IoT), imprimante 3D, sisteme embedded complexe.

1.1 Introducere în limbajul C

Programarea microcontroler poate fi realizată în diferite limbaje, cel mai raspandit fiind *Limbajul C*. O extensie a acestui limbaj este *Embedded C*, care este orientat spre platformele cu microcontroler. Acesta conține facilități complexe de accesare a blocurilor de memorie sau de manipulare a mărimilor de intrare/ieșire. În continuare se vor prenta informaţii utile pentru înțelegerea și implementarea lucrărilor de laborator prezentate în acest îndrumar.

1.1.1 Directive preprocesor

Aceste directive încep cu caracterul # și sunt interpretate înainte de a se compila programul propriu-zis (compus din instrucţiuni și declaraţii). În mod frecvent sunt întâlnite două directive [9]:

- directiva # *define*: permite folosirea constantelor simbolice în programe. La compilare, fiecare apariţie a simbolului este înlocuită, caracter cu caracter, cu valoarea sa. O astfel de directivă nu se termină cu ';';
- directiva # *include*: permite includerea unui fişier sursă în alt fişier sursă. Aceste fișiere sunt de obicei fișiere numite *antet* (eng. *header*), ce reunesc declaraţii de funcţii standard.

1.1.2 Tipuri de date

Principalele tipuri de date în limbajul C sunt împărțite în trei categorii:

- tipuri de bază: sunt tipuri de date aritmetice și pot fi clasificate în tipuri de date întregi și tipuri de date reale (în virgulă mobilă);
- tipuri derivate: acestea includ: tip pointer, tip tablou, tip structură, tip uniune și tip funcție;
- tipul *void*: reprezintă lipsa unui tip de date.

În tabelul 1.1 sunt prezentate tipurile de date întregi, dimensiunea ocupată în memorie și intervalul de valori pe care îl poate lua [8].

Tab. 1.1 Date de tip întreg.

Tip	Valori	Dimensiune în memorie
char	-128 ÷ 127	1 byte
unsigned char	0 ÷ 255	1 byte
signed char	-128 ÷ 127	1 byte
int	-32768 ÷ 32767	2 bytes
unsigned int	0 ÷ 65535	2 bytes
long	-2,147,483,648 ÷ 2,147,483,647	4 bytes

Detalii legate de tipurile de date reale, împreuna cu dimensiunea ocupată în memorie și cu valorile posibile sunt prezentate în tabelul 1.2. Ambele reprezentări au aceeași gamă de valori pentru plăcile de dezvoltare utilizate în cadrul laboratorului.

Tab. 1.2 Date de tip real.

Tip	Valori	Dimensiune în memorie
float	1.1E-38 ÷ 3.4E+38	4 byte
double	2.2E-308 ÷ 1.7E+308	8 byte

1.1.3 Variabile și constante

Variabila este un identificator asociat cu o locaţie de memorie în care se stochează valori care pot fi modificate pe parcursul existenței ei. Fiecare variabilă are un tip asociat, care

determină dimensiunea ocupată în memorie, valorile pe care le poate lua și setul de operații care se poate aplica asupra variabilei.

Numele variabilei poate fi compus din litere, cifre și caracterul "_". Numele poate începe cu literă sau "_". Limbajul C face distincție între majuscule și minuscule. Tipul de date pe care o variabilă poate să îl preia a fost prezentat în secțiunea anterioară.

În următorul program sunt prezentate o serie implementări:

```
1 int    i, j, k;
2 char   c, ch;
3 float  f, salariu;
4 double d;
```

În exemplul anterior au fost declarate și definite variabilele *i*, *j*, *k* ceea ce a condus la crearea de către compilator a variabilelor *i*, *j*, *k* de tip *int*.

De asemenea, variabilele pot fi și inițializate (se atribuie o valoare inițială) la declararea lor. De exemplu:

```
1 int d = 3, f = 5; //definire si initializare d si f
2 char x = 'x';     // variabila x primeste valoarea 'x'.
```

Constantele [9] reprezintă valori fixe (numerice, caractere sau şiruri de caractere), care nu pot fi modificate în timpul execuției programului. Tipul constantei este determinat de compilator pe baza valorii şi sintaxei utilizate, putând avea oricare dintre tipurile predefinite de date. Constantele rămân în memorie pe toată durata execuției programului.

În limbajul C sunt două moduri de a scrie o constantă:

- utilizând "#define";
- utilizând cuvântul cheie *const*.

În următorul exemplu sunt exemplificate ambele variante:

```
1 //varianta 1 utilizind #define
2 #define LUNGIME 10
3 #define INALTIME 78
4
5 //sau folosind cuvantul cheie const
6 const int LUNGIME = 10;
7 const int INALTIME = 78;
```

La variabilele prezentate anterior se poate adăuga și un identificator care să prezinte scopul (vizibilitatea) cât și durata de existență a lor. Dintre cei mai utilizați identificatori, din limbajul C, se amintește [8]:

- *register*: permite definirea de variabile locale care pot fi stocate în regiștrii, în loc de memoria RAM (depinde și de configurația HW). Astfel, variabila are dimensiunea maximă egală cu dimensiunea registrului. Se utilizează pentru variabile care necesită acces rapid (de exemplu numărătoare);

– *static*: variabilele statice sunt create o singură dată atunci când codul specific funcției în care au fost declarate este încărcat în memorie şi nu sunt distruse decât atunci când acest cod este eliminat din memorie (o astfel de variabilă rămâne în memorie atâta timp cât rulează programul);
– *extern*: permite crearea unei referințe globale, între toate fișierele program. Astfel, variabilele de acest tip sunt vizibile în toate fișierele programului.

1.1.4 Operatori și expresii

Operatorii sunt simboluri care, aplicate unor operanzi (date - constante, variabile, expresii), au ca efect realizarea unor operații matematice sau logice, care returnează un rezultat. O succesiune validă de operanzi şi operatori formează o *expresie*. Următoarele tipuri de operatori sunt disponibili în limbajul C: aritmetici, relaționali, logici, la nivel de bit, de atribuire.

În tabelul 2.1 sunt prezentați unii dintre operatorii utilizați în exemplele din cadrul acestui îndrumar.

Tab. 1.3 Operatori ai limbajului C.

Operator	Descriere	Exemplu
$+, -, *, \backslash$	Operatori matematici de bază	A=1, B=3: A $\star$ B = 3
$++, --$	Incrementare sau decrementare	A=4: A++ = 5
==	Verifică dacă valorile operanzilor sunt egale sau nu. Dacă da, condiția e adevărată	A=1, B=3: (A == B) este fals
! =	Verifică dacă valorile operanzilor sunt egale sau nu. Dacă nu, condiția e adevărată	A=1, B=3: (A != B) este adevarat
>, <, >=, <=	Verifică dacă valoarea este: mai mare, mai mică, mai mare și egală și mai mică și egală	A=5, B=3: (A >= B) este adevarat
&&, ||, !	Operator: ȘI logic, SAU logic și negație	A=1, B=0: (A || B) este adevarat
$\sim$	Operatorul complement pe bit: inversează reprezentarea sirului de biți	A=1: $\sim$A=0
&, |, $\wedge$, <<, >>	Operatori pe biți (ȘI, SAU, SAU exclusiv, șiftare la stânga, șiftare la dreapta)	A=60, A<< 2 = 240 în binar 11110000
+ =, − =, ∗ =, / =	Operatori de atribuire	C + = A este echivalent cu C = C + A

Pe lângă operatorii prezentați anterior se mai pot enumera și cei din tabelul 1.4:

1.1.5 Instrucțiuni

O instrucțiune reprezintă o porțiune de cod care odată executată permite efectuarea unei acțiuni: atribuire, selecție, iterație, etc. Instrucţiunile utilizează în general conceptul

Tab. 1.4 Operatori ai limbajului C.

Operator	Descriere	Exemplu
$sizeof()$	Returnează dimensiunea unei variabile	int A: $sizeof(A)$ este 2
&	Returnează adresa unei variabile	&A returnează adresa variabilei
$*$	Pointer la o variabilă	$*A$
?:	Dacă, condiția este adevarată ? atunci valoarea X : daca nu valoarea Y	if (A>B)? C=5 : C=10;

de adevărat și fals. Cele mai utilizate instrucțiuni sunt cele de decizie (condiționale), de selecție și cele repetitive.

Instrucțiuni de decizie

Acest tip de instrucțiuni sunt introduse utilizându-se cuvântul cheie *if*, astfel avem următoarele sintaxe generale:

Algoritmul 1.1 Varianta 1

```
1 if (expresie)
2     Instructiune; //Daca expresie este adevarata se executa
        ↪ Instructiune
```

Algoritmul 1.2 Varianta 2

```
1 if (expresie)
2     Instructiune1; //Daca expresie este adevarata se executa
        ↪ Instructiune1
3 else
4     Instructiune2; //Daca expresie nu este adevarata se executa
        ↪ Instructiune2
```

Următorul exemplu prezintă utilizarea celor două variante:

Algoritmul 1.3 Utilizare instrucțiuni de decizie

```
1 int nVarsta = 5;
2 int nContor = 1;
3 if (nVarsta > 3)
4     nContor++; //Expresia este adevarata, deci se executa linia
        ↪ nContor++
5 else
6     nContor--; //Conditia a fost adevarata, deci nu se executa
        ↪ aceasta linie
```

Instrucțiuni de selecție

Pentru a se realiza o selecție multiplă (în situația în care există mai multe cazuri posibile), se poate folosi o succesiune de instrucțiuni *if* (incluse unele în altele) sau se poate folosi o

instrucțiune de tipul *switch*. Aceasta din urmă permite introducerea unei enumerații a cazurilor posibile și o expresie de selecție. Forma generală este:

Algoritmul 1.4 Instrucțiunea *switch* - sintaxă generală

```
switch(expresie){
case c1:
    instructiune1;
    break;
case c2:
    instructiune2;
    break;
...
default:
    instructiune_default;
    break;
```

La început se evaluează *expresiea*, iar dacă se găsește în ceea ce precede cuvântul *case* expresia respectivă, se execută codul corespunzător până la linia *break*. Dacă nu este regăsită expresie în nici un case, atunci se execută *default*. Această variantă implicită, *default*, nu este obligatorie, dar este indicată întrucât poate trata situații neașteptate.

Utilizarea instrucțiunii switch este aratată în exemplul următor:

Algoritmul 1.5 Exemplu de utilizare a instrucțiunii *switch*

```
char operatie;
int nNr1, nNr2, rezultat;
.....
switch(operatie){
case '+':
    rezultat = nNr1 + nNr2;
    break;
case '-':
    rezultat = nNr1 - nNr2;
    break;
case '*':
    rezultat = nNr1 * nNr2;
    break;
```

Instrucțiuni repetitive

În limbajul C există trei tipuri de instrucțiuni repetitive: *for*, *while* și *do-while*.

Instrucțiunea *while* este una repetitivă, cu o condiție inițială și cu un număr necunoscut de etape. Sintaxa generală este:

Algoritmul 1.6 Instrucțiunea *while* - sintaxă generală

```
1 while (expresie)
2     Instructiune; //Atata timp cat expresie este adevarata se
      ↪ executa Instructiune
```

Expresia trebuie să conțină o variabila care este modificată (actualizată) interiorul buclei *while*. În acest fel, expresia poate să devină falsă și bucla să se oprească. De exemplu:

Algoritmul 1.7 Instrucțiunea *while* - exemplu

```
1 int nConditie = 1;
2 while (nConditie < 10)
3     nConditie++; //Atata timp cat expresie este mai mica de 10 se
      ↪ executa incrementarea
```

O buclă cu condiția de forma *while*(1) este o buclă infinită.

Instrucțiunea *for* este de asemenea una repetitivă, cu condiție inițială și cu un număr cunoscut de pași. Sintaxa sa generală este:

Algoritmul 1.8 Instrucțiunea *for* - sintaxă generală

```
1 for (expresie1; expresie2; expresie3)
2     Instructiune;
```

Instrucțiunea se execută în următoarele etape:

- expresie1: este evaluată la intrarea în buclă;
- expresie2: este evaluată înaintea fiecărei iterații şi reprezintă condiția de execuție a instrucțiunii; valoarea logică falsă a ei provoacă ieşirea din instrucțiune;
- expresie3: se evaluează la sfârșitul fiecărei iterații, pentru a se actualiza parametrii instrucțiunii.

Un exemplu de utilizare este următorul:

Algoritmul 1.9 Instrucțiunea *for* - exemplu

```
1 int i, nRezultat;
2 for (i = 0; i < 10; i++)
3     nRezultat++; //se repeta operatia de incermentare pana i = 9;
```

Instrucțiunea *do-while* este utilizată atunci când se dorește utilizarea unei condiții finale, iar ciclul să se execute cel puțin o dată. Acest lucru este diferit față de instrucțiunile repetitive *while* sau *for*. Sintaxa generală este:

Algoritmul 1.10 Instrucțiunea *do-while* - sintaxă generală

```
1 do{
2     Instructiune;
3 }while (expresie);
```

Se execută *instrucțiune* apoi este evaluată *expresie*. În funcție de aceasta se mai execută sau nu *instrucțiune*. Un exemplu de utilizare este următorul:

Algoritmul 1.11 Instrucțiunea *do-while* - exemplu

```
1 int nVar = 0, i = 10;
2
3 do{
4     nVar++; //Se executa aceaste instructiuni pana i ajunge la
        ↪ valoarea 0
5     i--;
6 }while (i != 0);
```

1.1.6 Tablouri de date

Tablourile de date reprezintă liste care pot stoca colecții fixe de elemente de același fel. Un tablou de date poate fi văzut și ca o colecție de variabile de același fel. Accesul la un element al unui tablou se face folosind unul sau mai mulți indici. În memorie, tablourile ocupă locații contigue. De asemenea, tablourile pot avea una sau mai multe dimensiuni.

Sintaxa de declarare a unui tablou de date în C, cu un anumit tip de date conținut este:

Algoritmul 1.12 Tablou unidimensional de date - sintaxă generală

```
1 tip nume_tablou_date [dimensiune];
```

Declararea acestui tablou uni-dimensional cuprinde dimensiunea sa, care trebuie să fie de tip întreg (mai mare decât zero), iar tipul este oricare dintre tipurile de date acceptate de limbajul C și prezentate în secțiunea § 1.1.2. De exemplu, pentru a se declara un tablou numit DateT cu 10 elemente de tip float, se poate folosi:

Algoritmul 1.13 Tablou unidimensional de date - exemplu

```
1 float DateT[10]; //declarare tablou unidimensional cu 10 elemente
    ↪ de tipul float
```

Declararea unui tablou de date se poate face și cu inițializare, după cum se arată în următorul exemplu.

```
1 float DateT[10]; //declarare tablou fara initializare
2 int DataI[3]={5,7,12}; //declarare tablou cu initializare
```

În cadrul tablourilor de date se mai pot folosi tablouri bidimensionale dar și declararea fără a se specifica numărul de elemente (atunci când se realizează inițializarea elementelor tabloului). Următorul exemplu ilustrează acest lucru.

Algoritmul 1.14 Tablou bidimensional de date - exemplu

```
1 float DateTI[10][3]; //declarare tablou bidimensional cu 10*3
    ↪ elemente, cu date de tip float
```

```
2 float vect[ ]={1.5, 3.2, 7.1, -2.5, 6}; //declarare fara a se
    ↪ preciza dimensiunea
```

Tablourile unidimensionale care conțin elemente de tip char sunt folosite pentru memorarea şirurilor de caractere. Pentru a marca sfârşitul unui şir de caractere, după ultimul caracter se adaugă un octet ('\0') numit *terminator de şir*. Declararea unui şir de caractere se face la fel ca la declararea tablourilor unidimensionale, cu date de tip char.

Algoritmul 1.15 Tablou bidimensional de date - exemplu

```
1 char numeSir[6]; //declarare sir de caractere fara initializare
2 char numeSirNou[3] = {'I', 'R', '\0'}; //declarare sir cu
    ↪ initializare
3 char anotimp[5]= "Vara";
```

1.1.7 Funcții

O funcție reprezintă un grup de declarații care împreună pot realiza o anumită sarcină. Limbajul C trebuie să conțină cel puțin o funcție, iar aceasta se numește *main*().

Utilitatea introducerii funcțiilor provine de la posibilitatea împărțirii codului după anumite funcționalități, alese de fiecare programator după necesitate. În mod logic, aceste funcții trebuie să realizeze o sarcină specifică.

O funcție are doua componente:

- *declarația* funcției: prin aceasta se informează compilatorul despre numele funcției, tipul de dată returnat și parametrii;
- *definiția* funcției: reprezintă corpul efectiv al funcției, în care se implementează funcționalitatea acesteia.

Sintaxa generală a unei funcții definite în C este:

Algoritmul 1.16 Funcțue - sintaxă generală

```
1 tip_returnat nume_functie(lista_parametri)
2 {
3     corpul functiei
4     .....
5 }
```

Funcțiile în limbajul C pot să conțină unele dintre următoarele:

- tip_returnat: este un tip oarecare de dată şi reprezintă tipul rezultatului returnat de funcție. Dacă nu este specificat, implicit tipul rezultatului returnat este int. Pentru funcțiile care nu returnează rezultat trebuie să se specifice tipul void;
- nume funcție: este un identificator unic al funcției;
- parametrii funcției: reprezintă enumerarea declarațiilor parametrilor sub forma: tip_parametru nume_parametru. Tipul parametrului poate fi orice tip valid de date;

– corpul funcției: conține o colecție de instrucțiuni care definesc scopul efectiv al unei funcții.

Următorul exemplu ilustrează definiția unei funcții care calculează maximul dintre două numere, primite ca și parametru. Funcție returnează valoarea maximă dintre cele două.

Algoritmul 1.17 Implementare funcție care returnează valoarea maximă

```
1  int calculeazaMaxim(int nNr1, int nNr2)
2  {
3      //se declara o variabila locala
4      int nResult;
5      //se verifica care numar este mai mare
6      if (nNr1 > nNr2)
7          nResult = nNr1;
8      else
9          nResult = nNr2;
10     //se returneaza numarul mai mare
11     return nResult;
12 }
```

1.2 Arduino Software (IDE)

Mediul de dezvoltare integrat Arduino (sau Arduino IDE) conține un editor de text în care se poate dezvolta codul, o zonă în care sunt afișate mesaje, o consolă și o zonă cu diferite meniuri. Acesta are capacitatea de a se conecta la partea hardware a modulului de dezvoltare Arduino, atât pentru a se încărca programele dezvoltate de utilizator, cât și pentru a se comunica cu acestea.

1.2.1 Instalare Arduino IDE

Editorul în care se vor scrie toate aplicațiile prezentate în acest îndrumar se poate descărca de la adresa: https://www.arduino.cc/en/Main/Software. Se alege varianta "Windows ZIP file for non admin install". Următorul pas presupune extragerea arhivei într-un director dorit și instalarea *arduino.exe*. După ce acest pas a fost efectuat, este necesară conectarea unui dispozitiv Arduino la unul dintre porturile USB ale calculatorului. După detectarea de către PC a plăcii Arduino, se poate selecta instalarea de drivere de pe PC, prin indicarea adresei unde aţi făcut dezarhivarea. Din acest director se alege directorul *drivers*.

1.2.2 Utilizare Arduino IDE

Rularea executabilului instalat la pasul anterior va conduce la deschiderea unui editor precum cel din figura 1.1.

Programele scrise în Arduino IDE sunt numite **sketches**. Acestea sunt scrise în editorul Arduino și salvate cu extensia *.ino*.

În cadrul editorului de text, pot fi identificate 4 zone importante. Zona 1 corespunde

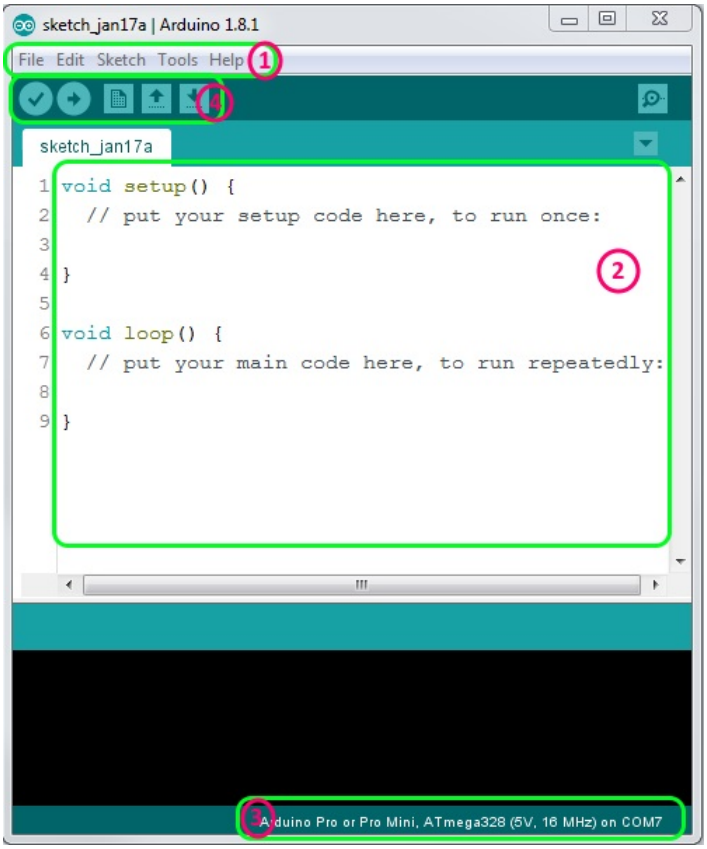


Fig. 1.1 Editorul de text pentru Arduino.

principalelor meniuri pe care le are IDE-ul: File, Edit, Sketch, Tools, Help. Funcționalitățile acestor meniuri sunt următoarele:

- *File*:
  - *New*: permite crearea de noi instanțe ale editorului de texte;
  - *Open*: Permite deschiderea unui program (sketch) existent;
  - *Exemple*: Conține toate exemplele disponibile la instalare în Arduino IDE, cât și exemplele provenite din librăriile instalate ulterior;
  - *Preferences*: Deschide o fereastră în care se pot particulariza anumite funcționalități ale IDE-ului.
- *Edit*: Conține comenzi de transfer de text, comentare/decomentare, cât și funcționalități de căutare.
- *Sketch*:
  - *Verify/Compile*: Verifică codul scris de erori de compilare. De asemenea, realizează un raport cu memoria utilizată de codul și variabilele din programul principal, raportate la microcontrolerul ales;
  - *Upload*: Realizează compilarea și încărcarea pe placa selectată a fisirului binar obținut, utilizîndu-se un port ales de utilizator (selectabil dintr-un alt meniu);
  - *Include Library*: Permite includerea în proiectul curent a unei librări externe. Librăria poate să fie preinstalată sau poate fi instalată, având arhiva acesteia;
  - *Add File*: Permite adăugarea unui fișier sursă la proiectul curent.

- *Tools*:
  - *Auto Format*: Formatează codul scris (spațiere, indentare);
  - *Serial Monitor*: Deschide o fereastră ce permite monitorizarea datelor transferate între placa de dezvoltare și portul de comunicație ales;
  - *Serial Plotter*: Realizează graficul datelor transmise pe canalul de comunicații serial într-o manieră real-time. Se pot realiza vizuălizări simultane pentru mai multe variabile disponibile pe modulul de comunicație serial;
  - *Board*: Permite selectarea plăcii de dezvoltare (o listă completă a acestora se poate găsi aici: https://www.arduino.cc/en/Guide/Environment-boards);
  - *Port*: Acest meniu conține toate dispozitivele seriale (reale sau virtuale) de pe PC-ul considerat.
- *Help*: În acest meniu se pot găsi numeroase documente care se instalează odată cu Arduino IDE.

Zona 2: Reprezintă spațiul de scriere al codului pentru placa de dezvoltare considerată (aplicația propriu zisă). La crearea unui nou program sunt adăugate în mod implicit două secțiuni:

- *setup*: este o secțiune care este rulată doar o singură dată, când este alimentată sau resetată placa de dezvoltare;
- *loop*: este o buclă infinită, executată ciclic atâta timp cât placa este alimentată.

Codul C corespunzător celor două secțiuni, la crearea unei noi instanțe Arduino este prezentat în algoritmul 1.18:

Algoritmul 1.18 Configurația inițială a unui program Arduino

```
1 void setup() {
2   // put your setup code here, to run once:
3 }
4
5 void loop() {
6   // put your main code here, to run repeatedly:
7 }
```

Zona 3: Conține informații legate de placa de dezvoltare utilizată și despre portul pe care se realizează comunicația cu aceasta (se poate modifica din meniul Tools/Port). Ultima zonă, zona 4, conține două comenzi, regăsite și în meniul *Sketch*: Verify și upload.

1.2.3 Simularea modulelor Arduino

Un prim pas în dezvoltarea aplicațiilor folosind Arduino presupune o etapă de testare primară sau validare. Această etapă poate fi realizată prin implementarea fizică a unei scheme electrice simplificată a proiectului, sau folosindu-se un simulator pentru Arduino. Astfel de simulatoare pentru Arduino sunt disponibile pe piață, atât pentru persoane cu experiență, cât și pentru începători. Testarea aplicațiilor în aceste simulatoare realizându-se fără grija eventualelor probleme de neconformitate cu anumite specificații tehnice.

Platformele de simulare pentru Arduino sunt ideale pentru programatorii și proiectanții care doresc să învețe elemente de bază ale circuitelor electrice. Astfel, cu ajutorul unui simulator pentru Arduino se poate învăța programare orientată Hardware (HW), fără a avea distruge componente sau echipamente atunci când se se fac greșeli.

Un alt avantaj al realizării de simulări pentru Arduino este faptul că permite realizarea de operații de debug, linie cu linie. Astfel, programatorul poate să descopere eventualele probleme sau funcționări incorecte în codul elaborat.

În cadrul îndrumarului se va utiliza simulatorul (emularea) Autodesk TINKERCAD [3]. Principalele componente conținute sunt prezentate în figura 1.2:

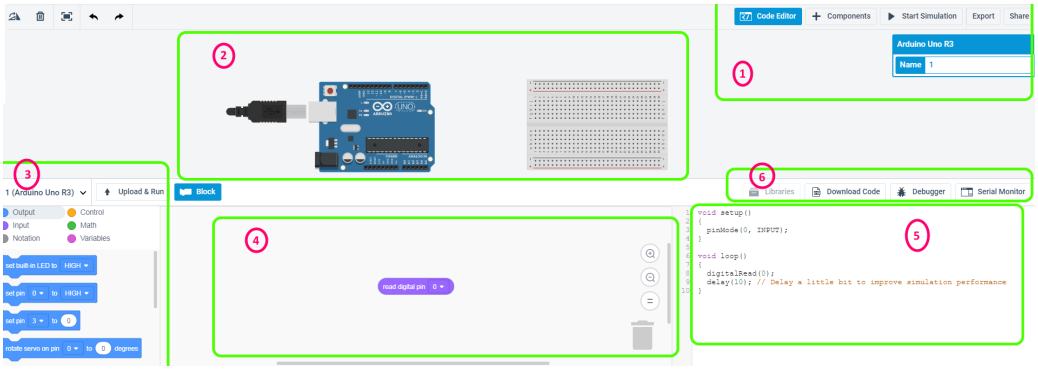


Fig. 1.2 Tinkercad: Simulator pentru Arduino.

Zonele importante pentru realizarea unei simulări sunt următoarele:

- zona 1: corespunde comenzilor legate de activarea/dezactivarea zonelor de scriere cod și listă componente. Pe lângă acestea se găsesc trei butoane: pornirea simulării, exportul modelului dezvoltat (a configuraţiei curente) și partajare (share) cu alte persoane, utilizându-se o adresă de email;
- zona 2: Corespunde zonei de elaborare a schemei HW, ce urmează a fi simulată și al cărei cod poate fi editat în zona 5;
- zona 3: Corespunde unor module predefinite, care pot fi manevrate cu "drag & drop" în zona 4. Odată poziționate în zona 4, se va putea observa și codul corespunzător modulelor, în zona 5;
- zona 4: utilizată pentru a crea o schemă logică, cu module predefinite. Codul corespunzător se găsește în zona 5;
- zona 5: zona de elaborare cod. Aici se poate observa codul corespunzător modulelor adăugate în zona 4. În cazul în care se dorește editarea directă a codului, fără blocuri predefinite, se dezactivează butonul *Block* din zona 3;
- zona 6: Corespunde modulelor de debug pentru codul elaborat sau de monitorizare a modulului de comunicație serială.

Pentru realizarea unei simulări este necesară înregistrarea pe pagina simulatorului Tinkercad [3] ca utilizator nou. Apoi, prin alegerea meniului "Circuits" și selectarea butonului "Create new Circuit" se poate ajunge la meniul prezentat mai sus. Pașii următori presupun alegerea de componente din zona corespunzătoare, editarea codului, pentru modulul Arduino, și încărcarea acestuia în HW. Pasul final este acela de rulare a simulării, prin

apăsarea butonului "Start Simulation".

1.3 Arduino Hardware

Plăcile de dezvoltare Arduino sunt extrem de variate, având posibilitatea de a se alege diferite funcționalități și procesoare. Toate plăcile de dezvoltare conțin un set de intrări digitale sau analogice care pot fi interfațate cu diferite plăci de extensie pentru Arduino (numite în lb. engleză *shields*) sau circuite conexe. Modulele de comunicație serială (pentru unele modele sunt prezente interfețe USB) sunt utilizate, în cadrul plăcilor de dezvoltare, pentru programarea modulelor Arduino.

Din punct de vedere hardware, multe dintre plăcile de dezvoltare Arduino conțin microcontrolere Atmel [2] pe 8 biți (ATmega8, ATmega168, ATmega328, ATmega1280, ATmega2560) cu diferite variante de echipare, din punct de vedere al memoriei flash, pinilor sau funcționalităților. Mare parte dintre plăci funcționează la 5 volți și au un oscilator cu cuarț (rezonator ceramic) de 16Mhz. Informații legate de arhitectura microcontrolerelor ATmega sunt disponibile în secțiunea 2.9 din cartea [5].

Microcontrolerele de pe modulele Arduino sunt pre-programate cu *boot loader*[4], astfel încât să se simplifice încărcarea programelor dezvoltate de programator pe memoria flash de pe cip. Plăcile de dezvoltare Arduino sunt programate utilizându-se interfața USB, având integrat un convertor USB-serial (cum este FTDI FT232 [6]).

1.3.1 Arduino UNO

Modelul UNO de la Arduino reprezintă una dintre cele mai populare plăci de dezvoltare pentru cei care doresc să învețe electronică și programare pe microcontrolere. Această placă de dezvoltare este considerată cea mai robustă, utilizată și bine documentată placă dintre toate cele disponibile, din familia Arduino. Acest model este cel de referință pentru modelele Arduino, este primul din seria placilor de dezvoltare Arduino cu conexiune USB.

Arduino UNO conține un microcontroler din familia ATmega328P [7] și are următoarele caracteristici principale:

- 14 pini digitali de intrare/ieșire (6 pot fi utilizați ca ieșiri PWM);
- 6 intrări analogice;
- oscilator cu cuarț de 16MHz;
- conexiune USB, alimentare jack;
- lucrează la 5 V, poate fi alimentat între 5-12V;
- 32KB de memorie flash (0.5KB pentru bootloader), 2KB SRAM, 1KB EEPROM;

În figura 1.3 se prezintă configurația tuturor pinilor și funcţiile speciale pe care le poate avea fiecare dintre aceștia.

Unii dintre pinii prezenți pe placă au funcții speciale. Dintre aceştia amintim:

- Comunicație serială: pinii 0 (RX) și 1 (TX). Utilizați pentru a transmite și a recepționa date într-o manieră serializată. Aceștia sunt conectați la pinii corespunzători adaptorului USB-TTL;
- Întreruperi externe: pinii 2 și 3. Aceștia pot fi setați să genereze o cerere de întrerupere

atunci când apar tranziții de la o stare la alta a pinicilor;
- PWM: pinii 3, 5, 6, 9, 10. Utilizați pentru a furniza un semnal de 8 biți, modulat în frecvență;
- comunicație SPI: pinul 10 (SS), 11(MOSI), 12 (MISO), 13 (SCK). Acești pini suportă comunicația SPI;
- LED 13: pin conectat la un led;
- Interfață I2C: pinii 18 (SDA) și 19 (SCL) pot fi folosiți pentru a transmite sau recepționa date;
- Intrări analogice: notate cu A0-A5, sunt conectate la un modul ce permite convertirea unui semnal analogic în unul digital.

Plăcile de dezvoltare Arduino UNO conțin un fuzibil resetabil, care protejează portul USB al calculatorului de scurtcircuite sau sarcini mai mari de curent. Chiar dacă multe dintre porturile calculatoarelor au astfel de protecții, acesta aduce extra protecție portului USB. Dacă un curent mai mare de 500 mA este aplicat pe port, fuzibilul o sa oprească portul/conexiunea până când scurtul sau supraîncărcarea este eliminată.

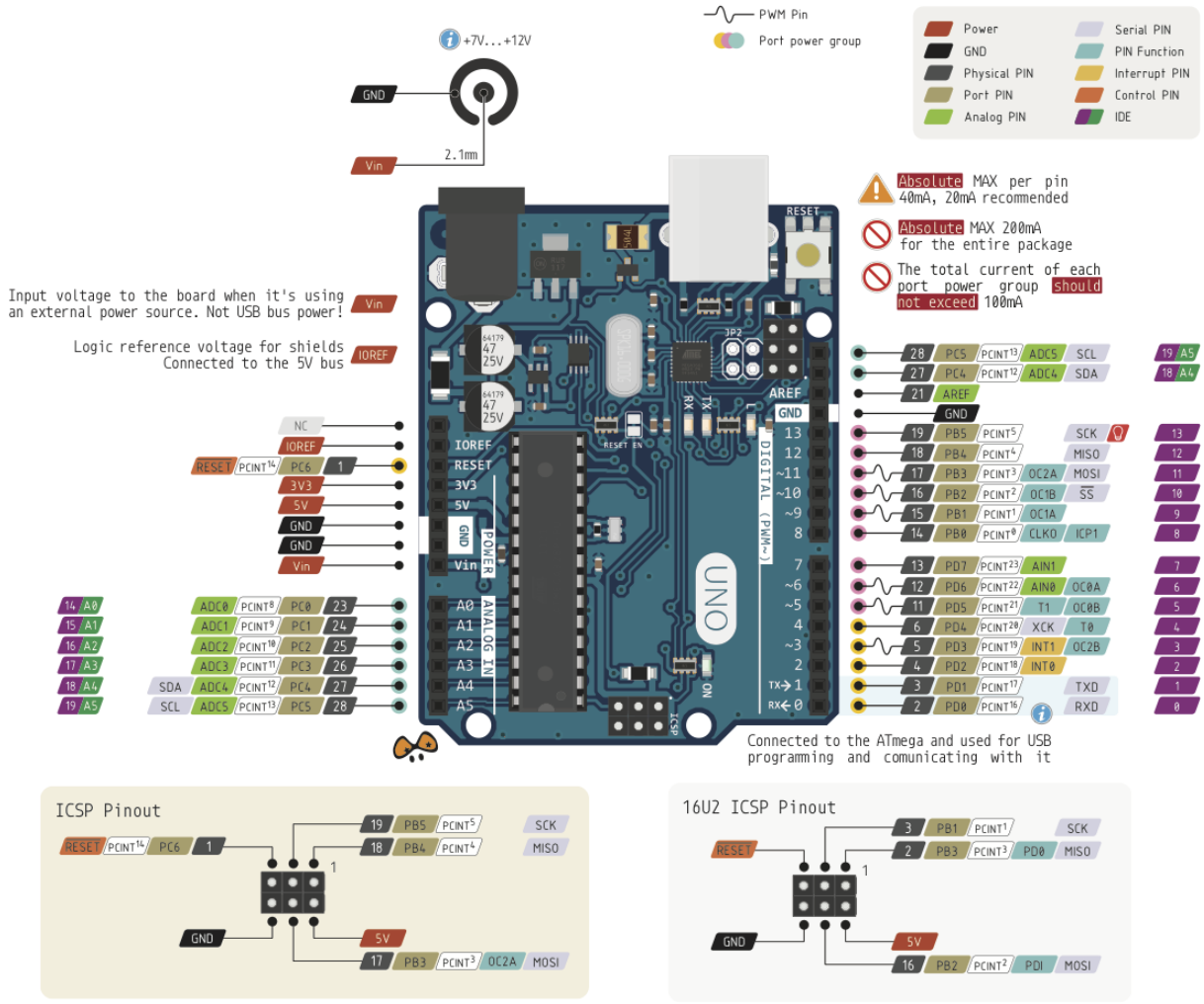


Fig. 1.3 Configurația pinilor pentru Arduino UNO.

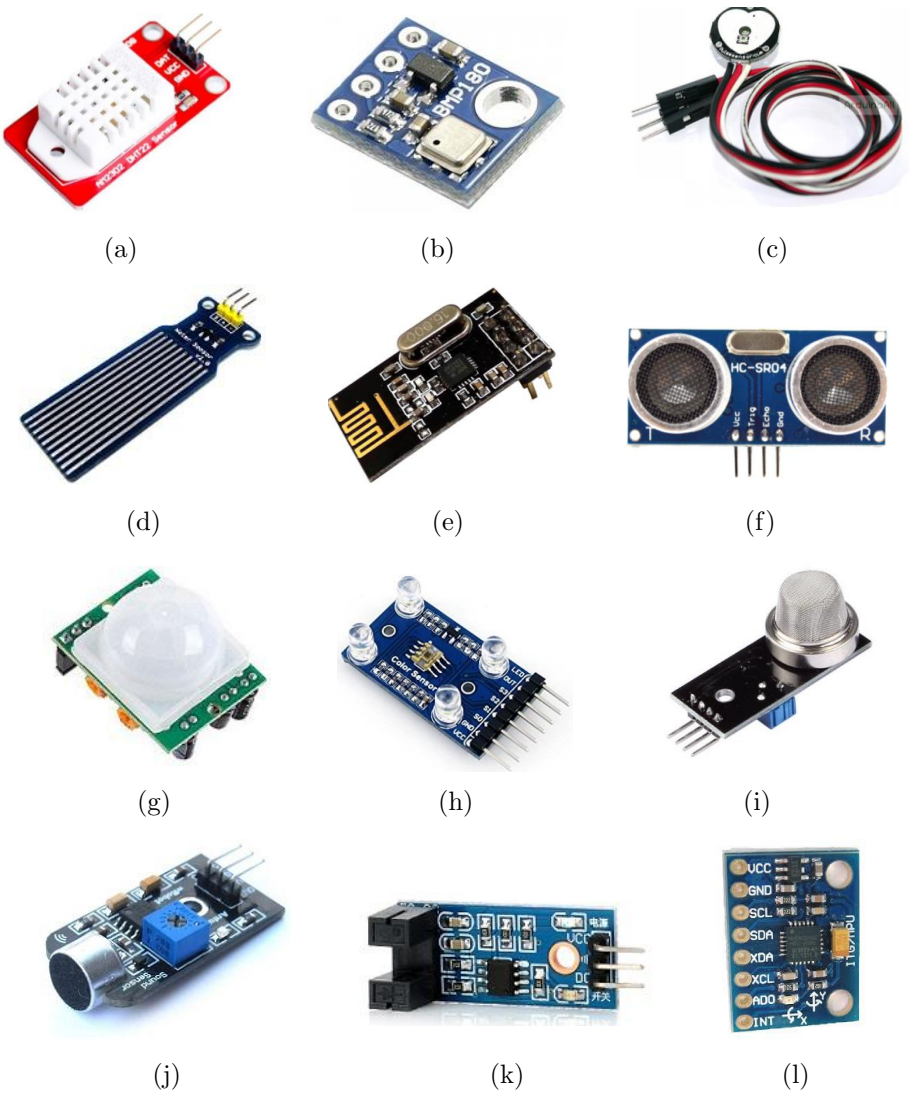
1.3.2 Senzori compatibili cu modulele Arduino

În cadrul diferitelor aplicații embeded, plăcile de dezvoltare Arduino pot fi utilizate cu ușurință împreună cu diferiți senzori digitali sau analogici precum:

- senzor de temperatură (figura 1.4(a)): permite măsurarea temperaturii și a umidității. Pentru măsurarea temperaturii se pot folosi senzori din familia: DHT11, DHT22, LM35 sau TMP36;
- senzor barometric (figura 1.4(b)): permite determinarea valorilor presiunii atmosferice, a temperaturii cât și a altitudinii. Din această familie de senzori fac parte BMP180 sau BMP280;
- senzor de puls (figura 1.4(c)): permite determinarea ritmului cardiac. Poate fi utilizat pentru monitorizarea ritmului cardiac;
- senzor de ploaie (figura 1.4(d)): acest senzor poate fi utilizat pentru determinarea prezenței apei pe suprafața senzorului;
- senzor wireless (figura 1.4(e)): Utilizat pentru a permite transferul serial de date pe WIFI. Modulul permite realizarea de conexiuni simple pe TCP/IP. Un modul de acest fel este senzorul ESP8266;
- senzor cu ultrasunete (figura 1.4(f)): permite măsurarea distanțelor utilizănd viteza de propagare a sunetelor. Cei mai populari senzori de distantă cu ultrasunete sunt HC-SR04 și Ping;
- senzor de prezență (figura 1.4(g)): permite detectarea mișcării, cum este cazul detectării unei persoane în interiorul sau exteriorul unei anumite zone. Un astfel de senzor este PIR Hc-SR501;
- senzor de culoare (figura 1.4(h)): Senzorii din această categorie detectează culoarea și furnizează la exterior un semnal a cărui frecvență este proporțională cu intensitatea luminoasă. Un astfel de senzor este TCS230;
- senzor de gaz (figura 1.4(i)): permite detecția concentrației de GPL, butan, metan, alcool, propan, hidrogen sau fum. Un astfel de senzor este CH4;
- senzor detecție sunete (figura 1.4(j)): poate fi utilizat pentru detecția de sunete din mediul înconjurător, având o sensibilitate ajustabilă. Un astfel de senzor este KY-038;
- senzor detecție viteză (figura 1.4(k)): Este utilizat în combinație cu un disc cu fante. Acest sistem, format din senzor și disc, permite monitorizarea vitezei unui motor (contorizarea fantelor). Sistemul este compus dintr-un element care furnizează lumină (un LED) și un element sensibil la lumină (de exemplu un fototranzistor). Un senzor de acest fel este LM393.
- senzor cu giroscop și accelerometru (figura 1.4(l)): Un astfel de senzor este compus din două module capabile să măsoare orientarea senzorului față de polul N, cât și viteza unghiulară. Tot pe același senzor este un modul capabil să determine accelerațiile. Un astfel de modul este MPU-6050.

Un alt modul care poate fi adăugat senzorilor prezentați anterior, este un modul de control (nu este un senzor) pentru motoare de curent continuu. Un astfel de modul, cunoscut și sub

denumirea de *driver de motoare* este L298H. Acest modul permite controlul simultan a două motoare. Are la bază circuitul integrat L298 (compus dintr-o punte H). Controlul se poate realiza în ambele direcții, iar curenții maximi sunt de 2A, pentru fiecare motor. Acest driver este ideal pentru aplicații robotice, întrucât necesită puține linii de comandă pentru fiecare motor.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

Fig. 1.4 Senzori compatibili cu module Arduino: (a) senzor de temperatura; (b) senzor barometric; (c) senzor de puls; (d) senzor de ploaie; (e) senzor wireless; (f) senzor cu ultrasunete; (g) senzor de prezență; (h) senzor de culoare; (i) senzor de gaz; (j) senzor pentru sunete; (k) senzor pentru măsurarea vitezei, (l) senzor cu giroscop și accelerometru.

1.4 Aplicaţii propuse

Utilizându-se compilatorul de C disponibil online la adresa: www.onlinegdb.com, să se implementeze următoarele exemple:

1. Să se implementeze un program care realizează inversiunea a două numere citite de la tastatură. Rezultatul se va afișa în consolă.

2. Să se implementeze o funcție ce returnează valoare 1 dacă numărul primit ca parametru

este par și valoarea 0 dacă numărul este impar.

3. Să se implementeze o funcție ce returnează cel mai mare număr dintre cele 3 primite ca și parametrii de intrare ai funcției.

4. Să se implementeze un minicalculator pentru operațiile de bază: adunare, scădere, înmulţire și împărţire.

5. Să se implementeze o funcție care să returneze starea bitului de pe poziţia n, primit ca parametru al funcției.

2. Portul digital de intrare-ieșire

Regiștrii importanți
Portul digital de intrare
Portul digital de ieșire
Aplicații cu porturi digitale de intrare-ieșire

Porturile digitale de intrare-ieșire, referite în continuare ca porturi I/O, reprezintă capacitatea unui microcontroler de a transmite sau monitoriza stări logice în circuitul electric în care acesta este folosit. În general, un microcontroler deține 2 sau 3 porturi I/O care pot fi conectate direct la circuitul electric din care face parte, bineînțeles cu respectarea specificaţiilor electrice ale microcontrolerului. În cadrul laboratorului se va folosi Microcontrolerul Atmega328P. Acesta deține 28 de pini metalici. Dintre aceștia, 23 pot fi folosiți ca pini I/O, restul de 5 pini fiind pini cu funcţii specifice: alimentare (GND și VCC), respectiv referință (AREF). Fiecare pin din cei 23 reprezintă un singur canal de legătură cu exteriorul.

Din punct de vedere software, un port I/O este reprezentat prin intermediul unui byte, adică este format din 8 biți. Fiecare bit al fiecărui port I/O este conectat la un pin metalic al integratului microcontrolerului. Acesta poate fi folosit ca ieșire, ca intrare sau ca intrare/ieșire. Din considerente de optimizare, pe lângă funcția de intrare/ieșire digitală, un pin metalic poate îndeplini funcționalități precum: întrerupere, comunicație serială, convertor analog-digital, etc (vezi figura 2.1). Aceste funcționalități nu pot fi folosite în paralel pe același pin. Este de datoria utilizatorului să asigure că celelalte funționalități paralele sunt dezactivate atunci când pinul este folosit într-un anumit mod. De exemplu, pentru a folosi pinul 2 al microcontrolerului Atmega328P, utilizatorul trebuie să asigure dezactivarea modulului de comunicație serială, deoarece pinul 0 al Portului D (PD0) mai poate îndeplini și funcția de recepție semnal serial (RxD).

Termenul *digital* se referă la faptul că un microcontroler folosește cele 2 nivele logice 1, respectiv 0 pentru a defini o stare. Circuitul electric în care microcontrolerul este utilizat are la bază o tensiune electrică, care, în general, este de 5V sau 3.3V. Așadar, un pin al microcontrolerului poate avea tensiunea de 5V sau 0V (GND). Microcontrolerul digitalizează această valoare prin maparea ei la una dintre cele două stări logice (1, respectiv 0). Așadar, din perspectiva microcontrolerului, orice tensiune peste valoarea de 2V va fi mapată la starea

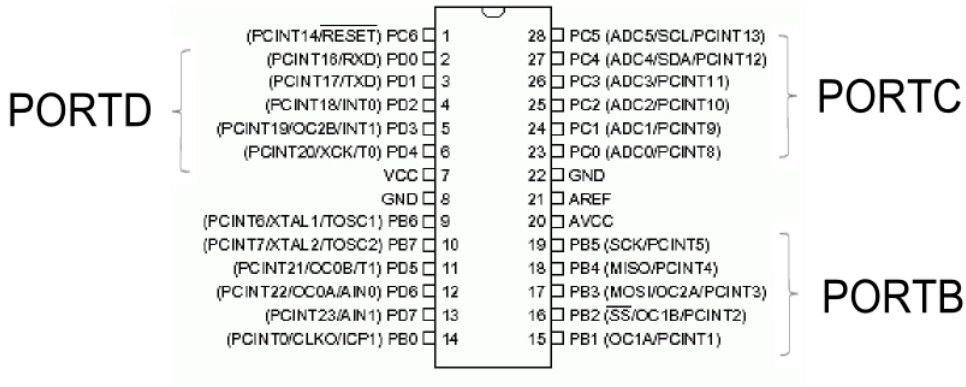


Fig. 2.1 Porturile I/O a microcontrolului Atmega328P.

logică 1 (eng. *high*) pentru tehnologia TTL, respectiv orice tensiune sub valoare de 0.8V (GND) va fi mapată către starea logică 0 (eng. *low*). Pentru orice tensiune între cele 2 praguri menționate, pinul microcontrolerului se va afla în starea de înaltă impedanță.

2.1 Regiștrii portului de intrare-ieșire

Un microcontroler deține multiple resurse. Porturile I/O sunt un exemplu. Fiecare resursă poate fi utilizată în mai multe feluri și cu diferiți parametrii, viteze, etc.. Regiștri sunt elementele ce permit configurarea funcționalităților resurselor unui microcontroler. Intuitiv, un registru poate fi comparat cu un panou cu butoane (vezi figura 2.2).

Fig. 2.2 Panou cu butoane de comandă.

Un buton poate avea două stari: deschis (0 logic) sau închis (1 logic). Închis înseamnă că funcționalitatea la care este atașat butonul este activă. Mai multe butoane cu funcționalități similare pot fi grupate și formează un *registru*. Grupurile se pot forma din multipli de 8 butoane. Un registru este foarte asemănător cu o adresă de memorie, diferența fiind aceea că o memorie stochează un număr, în timp ce un registru stochează o configurație a unei resurse a microcontrolerului. De exemplu, dacă un registru ar arăta ca cel din figura 2.3, microcontrolerul ar fi configurat să activeze pe pini 0, 6 și 7 ai unui port de ieșire starea logică 1, fapt ce ar fi determinat aprindere unor LED-uri. Pentru că utilizăm un sistem digital, un buton este echivalentul unui bit. Practic cele 8 butoane vor constitui 1 byte de configurare, adică un registru.

Configurarea unui port de intrare/ieșire se realizează prin intermediul următorilor regrstrii (vezi figura 2.4):

- PORTx - reprezintă registrul în care se setează starea (1 sau 0) pinilor de ieșire;
- DDRx - reprezintă registrul în care se configurează sensul piniilor portului ca fiind de intrare sau ieșire;

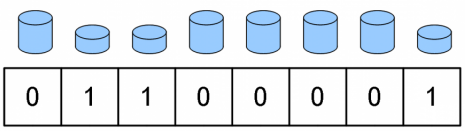


Fig. 2.3 Registru de 8 biți.

– PINx - reprezintă registrul din care se citește starea (1 sau 0) pinilor de intrare.

Pentru că un microcontroler poate avea mai multe porturi, la sfârșitul denumirii fiecărui registru de tip port se adaugă o literă ce identifică unic acel port. De exemplu microcontrolerul Atmega328P are 3 porturi I/O identificate prin literel B, C și D (PORTB, PORTC și PORTD). Așadar, registrul portului D în care se va face configurarea direcției pinilor portului va fi denumit DDRD.

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
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	

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
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	

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Fig. 2.4 Regiștrii de configurare ai portului D.

2.2 Portul digital de ieșire

Este folosit de microcontroler pentru a transmite semnale logice către circuitul electric în care acesta este utilizat. Un pin al portului digital de ieșire configurat să furnizeze starea logică 1 asigură existența unei tensiuni de 5 volți și un curent de maxim 40 mA. Aceste caracteristici sunt suficiente pentru a aprinde un LED sau pentru a controla numeroși senzori, însă insuficiente pentru a alimenta un releu sau diverse motare de curent continuu. Scurcicuitarea sau suprasolicitarea pot conduce la defectarea pinilor portului de ieșire. Totuși, această situație nu va afecta restul funcționalităților microcontrloerului, ele putând fi folosite în continuare.

Utilizarea portului digital I/O ca port de ieșire necesită configurarea stării logice 1 pentru fiecare bit din registrul DDRx ce se dorește a fi folosit ca ieșire. Stabilirea unei tensiuni (0V sau 5V) pe pinii de ieșire ai portului microcontrolerului se configurează prin intermediul

registrului PORT. Astfel, configurarea stării logice 1 a unui bit al acestui registru determină aplicarea tensiunii de 5V la pinul de ieșire aferent.

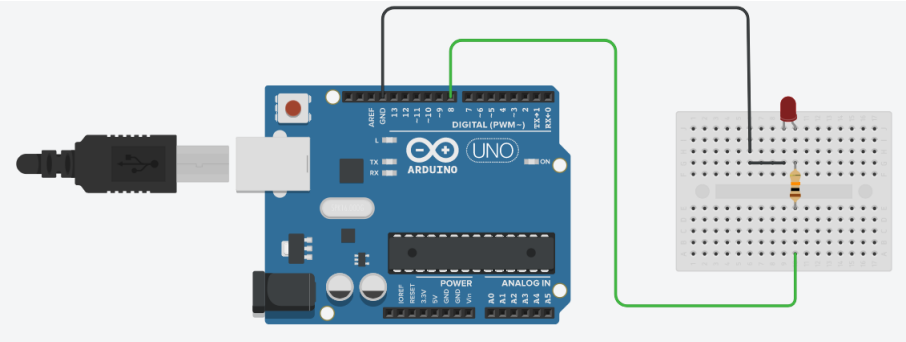


Fig. 2.5 Schema electrică pentru aprinderea unui LED.

2.2.1 Exemplu de aplicaţie

Se consideră schema electrică formata dintr-o placă de dezvoltare Arduino UNO, un LED și un rezistor de 10 KOhmi (vezi figura 2.5). Montajul a fost realizat folosind simulatorul disponibil pe pagina de web *https://www.tinkercad.com*. Obiectivul este acela de a aprinde, respectiv stinge, LED-ul la un interval de 1000 milisecunde.

Programul scris în IDE-ul Arduion 1.8.5 este următorul:

Algoritmul 2.1 Exemplu de utilizare a portului de ieșire al unui microcontroler pentru a aprinde, respectiv a stinge un LED

```
1  void setup()
2  {
3      // stabileste directia pinului 8 al placii Arduino ca pin de
       ↪ iesire
4      pinMode(8, OUTPUT);
5  }
6
7  void loop()
8  {
9      // aplica 1 logic pe  pinul 8 pentru a aprinde LED-ului placii
       ↪  Arduino
10     digitalWrite(8, HIGH);
11     // asteapta 1000 milisecunde(s)
12     delay(1000);
13     // aplica 0 logic pe  pinul 8 pentru a stinge LED-ul placii
       ↪ Arduino
14     digitalWrite(8, LOW);
15     //  Asteapta 1000 milisecunde(s)
16     delay(1000);
17 }
```

În programul anterior au fost realizate următoarele setări:

- linia 4: configurarea pinului 8 ca iesirea (care este conectat la bitul 0 al portului B al microcontrolerului);
- linia 10: configurarea stării logice 1 pentru bitul 0 al portului B;
- linia 14: configurarea stării logice 0 pentru bitul 0 al portului B.

Programul prezentat mai sus conține funcții ce îl ajută pe programator să scrie cod într-o manieră intuitivă, corelată cu placa de dezvoltare Arduino UNO. De exemplu, linia de cod 9 determină setarea tensiunii de 5V pe pinul numărul 8 al plăcii de dezvoltare. Funcţia *digitalWrite* se va ocupa mai departe de legătura acestuia cu pinul microcontrolerului. Practic, programatorul nici nu trebuie să știe la ce pin al microcontrolerului este conectat pinul 8 al plăcii de dezvoltare. Aceeaşi funcționalitate a algoritmului anterior poate fi rescrisă, fără a utiliza aceste funcții, după cum urmează:

Algoritmul 2.2 Exemplu de utilizare a portului de ieșire pentru a aprinde un LED utilizând explicit registrele microcontrolerului.

```
1  #define DDB0            0
2  #define PORTB0          0
3
4  void setup()
5  {
6      // stabileste directia bitului 0 al portului B ca pin de
       ↪ iesire
7      // bitul 0 al portului B este conectat la pinul 8 respectiv
       ↪ LED-ul placii Arduino
8      DDRB = 1 << DDB0;
9  }
10
11 void loop()
12 {
13     // aplica 1 logic pe  pinul 8 ceea ce determine aprinderea LED
       ↪ -ului placii Arduino
14     PORTB = 1 << PORTB0;
15     // asteapta 1000 milisecunde(s)
16     delay(1000);
17     // aplica 0 logic pe  pinul 8 ceea ce determine aprinderea LED
       ↪ -ului placii Arduino
18     PORTB = 0 << PORTB0;
19     //  asteapta 1000 milisecunde(s)
20     delay(1000);
21 }
```

În programul anterior au fost realizate următoarele setări:

– linia 1: redefinirea pinului 0 din registrul DDRB, sub numele de *DDB0*;
– linia 2: redefinirea pinului 0 din registrul PORTB, sub numele de *PORTB0*;
– linia 8: configurarea lui DDB0 (bitul 0) ca pin de ieșire în registrul de direcție DDRB;
– linia 14: configurarea stării logice 1 pentru PORTB0 (bitul 0) în registrul PORTB;
– linia 18: configurarea stării logice 0 pentru PORTB0 (bitul 0) în registrul PORTB.

2.3 Portul digital de intrare

Este folosit de microcontroler pentru a monitoriza/citi semnale din circuitul electric în care acesta este utilizat. Cel mai des, pinii porturilor de intrare sunt folosiți pentru a evalua starea unor butoane (închis/deschis). Două registre sunt importante în acest sens: registrul DDR și registrul PIN. Configurând valoarea 0 pe un bit al registrului DDR al unui port, îi spunem microcontrolerului că dorim să utilizam acel pin, ca pin de intrare. Citirea stări/valorii logice a pinului portului se realizează prin intermediul registrului PIN corespunzător (vezi figura 2.4).

2.3.1 Exemplu de aplicaţie

Se consideră schema electrică formată dintr-o placă Arduino UNO, un LED, un rezistor și un buton fără reținere (vezi figura 2.6). Montajul a fost realizat folosind simulatorul disponibil pe pagina de web *https://www.tinkercad.com*. Obiectivul este acela de a aprinde, respectiv a stinge un LED la fiecare apăsare a butonului.

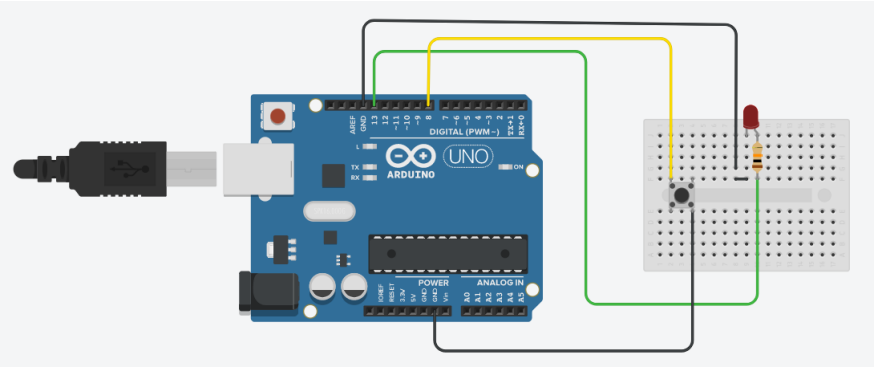


Fig. 2.6 Schema electrică pentru citirea stării unui buton.

Citirea stării butonului se face pe pinul numărul 8, de pe placa de dezvoltare. Conform cablajului circuitului plăcii de dezvoltare, pinul 8 este conectat la bit-ul 0 al portului B al microcontrolerului. Astfel, când butonul este apăsat, starea logică a pin-ului este 0 (0V). Când acesta este relaxat, starea logică a pinului este necunoscută. Practic, pe firul galben din figura 2.6 nu avem nici 5V și nici 0V (stare de înaltă impedanță). Pentru a evita această situație se va folosi un rezistor care are rolul de a menține o stare logică cunoscută atunci când butonul nu este apăsat, de exemplu 1 logic dacă rezistorul este conectat la 5V. Acest rezistor poartă denumirea de rezistor de *pull-up*. Valoarea rezistorului trebuie să fie de aproximativ 20 KOhmi. Microcontrolerul Atmega328P are încapsulat în integrat-ul lui un astfel de rezistor pentru fiecare pin al fiecărui port de intrare. Acești rezistori pot fi activați prin configurarea stării logice 1 în bitul corespunzător al registrului PORT (vezi

figura următoare).

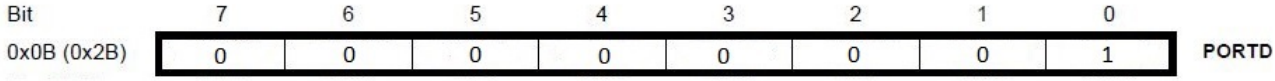


Fig. 2.7 Activarea rezistorului de pull-up pentru pinul 0 al portului D.

Programul scris în IDE-ul Arduino 1.8.5 este următorul:

Algoritmul 2.3 Exemplu de utilizare a portului de intrare pentru a citi starea unui buton și a aprinde, respectiv a stinge un LED

```
1  // pin-ul conectat la buton
2  const byte pinButon = 8;
3  // pin-ul conectat la LED
4  const byte pinLED = 13;
5
6  // starea curenta a LED-ului
7  byte stareLED = LOW;
8
9  void setup()
10 {
11     // stabileste directie pin ca intrare
12     pinMode(pinButon, INPUT);
13     // stabileste directie pin ca iesire
14     pinMode(pinLED, OUTPUT);
15
16     // activeaza registru de pull-up
17     digitalWrite(pinButon, HIGH);
18     // setare stare initiala LED
19     digitalWrite(pinLED, stareLED);
20 }
21
22 void loop()
23 {
24     // se verifica daca butonul a fost apasat,
25     // adica daca pin-ul e in starea LOW
26     if (digitalRead(pinButon) == LOW)
27     {
28         // asteptam ca butonul sa nu mai fie apasat
29         // inainte de a schimba starea, adica asteptam
30         // atat timp cat pin-ul e in starea LOW
31         while (digitalRead(pinButon) == LOW)
```

```
32          {
33              // asteptare...
34          }
35
36          // basculare stare LED
37          stareLED = !stareLED;
38          // setare stare LED
39          digitalWrite(pinLED, stareLED);
40      }
41  }
```

În programul anterior au fost realizate următoarele setări:

- linia 2: definirea variabilei ce configurează legătura dintre placa de dezvoltare, pe care se află conectat LED-ul, și pinul 5 al portului B al microcontrolerului;
- linia 4: definirea variabilei ce configurează legătura dintre placa de dezvoltare și pinul 0 al portului B al microcontrolerului;
- linia 7: variablila pentru a stoca starea LED-ului;
- linia 12: configurarea bitului 5 al registrului DDRB, ca pin de ieşire;
- linia 14: configurarea bitului 0 al registrului DDRB, ca pin de intrare;
- linia 17: configurarea stării logice 1 pentru bitul 0 al registrului PORTB, pentru a activa rezistorul de pull-up;
- linia 19: configurarea stării inițiale a LED-ului.

Programul anterior poate fi rescris, fără a utiliza funcții specifice Arduino, după cum urmează:

Algoritmul 2.4 Exemplu de utilizare a portului de intrare pentru a citi starea unui buton și a aprinde, respectiv a stinge un LED, utilizând explicit registrele microcontrolerului.

```
1  #define PIN0     0
2  #define PIN5     5
3
4  byte stareLED = 0;
5
6  void setup()
7  {
8      // stabileste directiile celor doi pini,
9      // buton (PIN0) pe input si LED (PIN5) pe output
10     DDRB = (1 << PIN5) | (0 << PIN0);
11
12     // activeaza registru pull-up si setare
13     // stare initiala pentru LED
14     PORTB = (stareLED << PIN5) | (1 << PIN0);
15 }
```

```
16
17 void loop()
18 {
19     // se verifica daca butonul a fost apasat,
20     // adica daca pin-ul e in starea LOW (0)
21     if ((PINB & (1 << PIN0)) == 0)
22     {
23         // asteptam ca butonul sa nu mai fie apasat
24         // inainte de a schimba starea, adica asteptam
25         // atat timp cat pin-ul e in starea LOW (0)
26         while ((PINB & (1 << PIN0)) == 0)
27         {
28             // asteptare...
29         }
30
31         stareLED = !stareLED; // basculare stare LED
32         // resetare stare LED anterioara
33         PORTB &= ~(1 << PIN5);
34         // setare stare LED actuala
35         PORTB |= stareLED << PIN5;
36     }
37 }
```

2.4 Aplicații cu porturi digitale de intrare-ieșire

În cadrul acestui sub-capitol vor fi prezentate trei dintre cele mai utile moduri de folosire a porturilor digitale de intrare / ieșire ale unui microcontroler. Pentru fiecare dintre acestea se va prezenta: schema electrică, programul software aferent și o serie de explicații utile.

2.4.1 Utilizarea porturilor digitale de ieșire pentru comanda afişajelor cu 7 segmente

Afișarea valorii temperaturii, afișarea orei, respectiv a minutului unui ceas digital, afișarea valorii unui numărător, sunt doar câteva aplicații în care un microcontroler trebuie să transmită informații către utilizator, prin intermediul un dispozitiv de afișare. O soluție este afișajul cu 7 segmente. Acesta poate fi controlat prin intermediul porturilor digitale de ieșire. Practic, 8 leduri sunt așezate pentru a descrie vizual o valoare numerică între 0 și 9 (vezi figura următoare).

Modul de conectare al unui afișaj cu 7 segmente la un microcontroler, precum și modul de configurare al registrului PORT pentru a afișa valoarea 5, este prezentat în figura următoare:

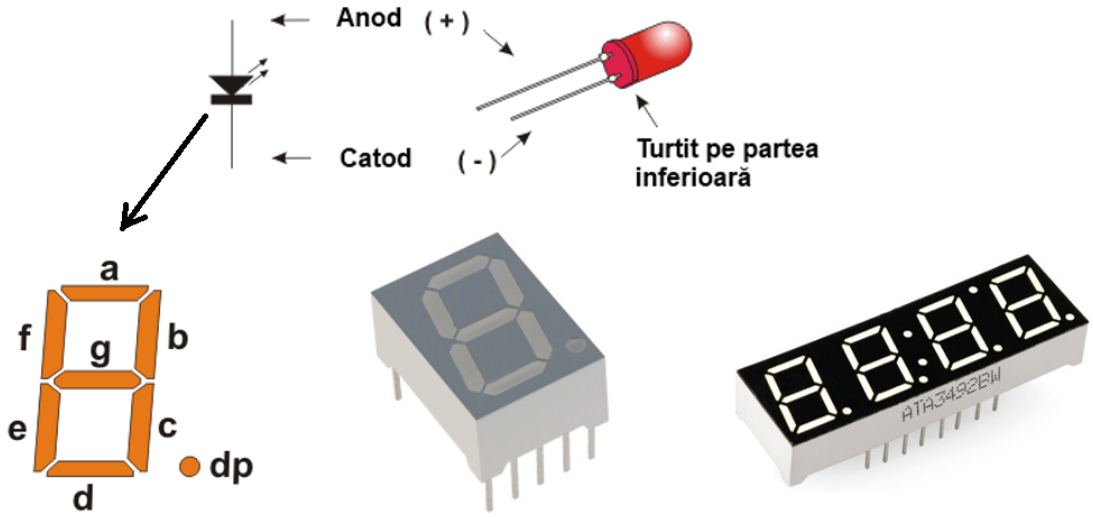


Fig. 2.8 Afișaj cu 7 segmente. (sus): o diodă LED. (jos): stânga: adnotarea segmenelor; (jos) mijloc: afișaj pentru o singură valoare; (jos) dreapta: afișaj pentru 4 valori.

2.4.1.1 Exemplu de aplicaţie

Se propune o aplicație care să contorizeze număul de pasageri ai unui avion. Îmbarcarea unui pasager va fi simulată prin apăsarea unui buton care va incrementa numărul de pasageri. Omolog, debarcarea va fi simulată prin apăsarea unui al doilea buton care va decrementa numărul de pasageri. Resetarea numărului de pasageri se va face la apăsarea celui de-al treilea buton. Se consideră că, într-un avion pot încăpea aproximativ 200 de pasageri. Pentru a afișa o astfel de valoare vor fi necesare 3 module cu 7 segmente. Astfel. pentru a controla toate aceste 7 segmente vor fi necesari 3x7 = 21 pini de ieșire. Majoritatea microcontrolerelor nu beneficiază de aşa mulți pini de ieșire, motiv pentru care toate cele trei module cu 7 segmente vor fi conectate în paralel la un singur port de ieșire. Practic, va fi ca o magistrală pe care se vor transmite valori, urmând ca fiecare afișaj să fie activat, pe rand, prin intermediul unui tranzistor care va controla catodul modulului cu 7 segmente. Datorită frecvenței mari de activare, respectiv dezactivare, secvențială a fiecărui modul, vizul se va avea impresia că ambele sunt aprinse (active) în același timp. Schema electrică, prezentată în figura 2.10, necesită următoarele componente electrice:

- 1x arduino Uno R3;
- 13 resistori x 1k ohm;
- 3x butoane fără reținere;
- 3x afișaje cu 7 segmente - catod comun;
- 3x tranzistoare de tip PNP.

Algoritmul 2.5 Utilizarea afișajului cu 7 segmente.

```
1 // numarul de afisoare cu 7 segmente
2 const byte N = 3;
3
```

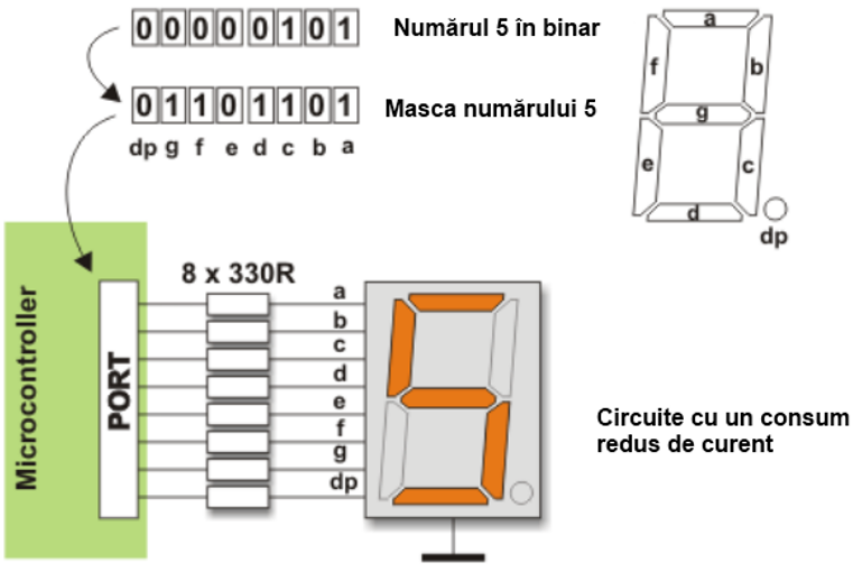


Fig. 2.9 Afișaj cu 7 segmente configurat să afișeze valoarea decimală 5.

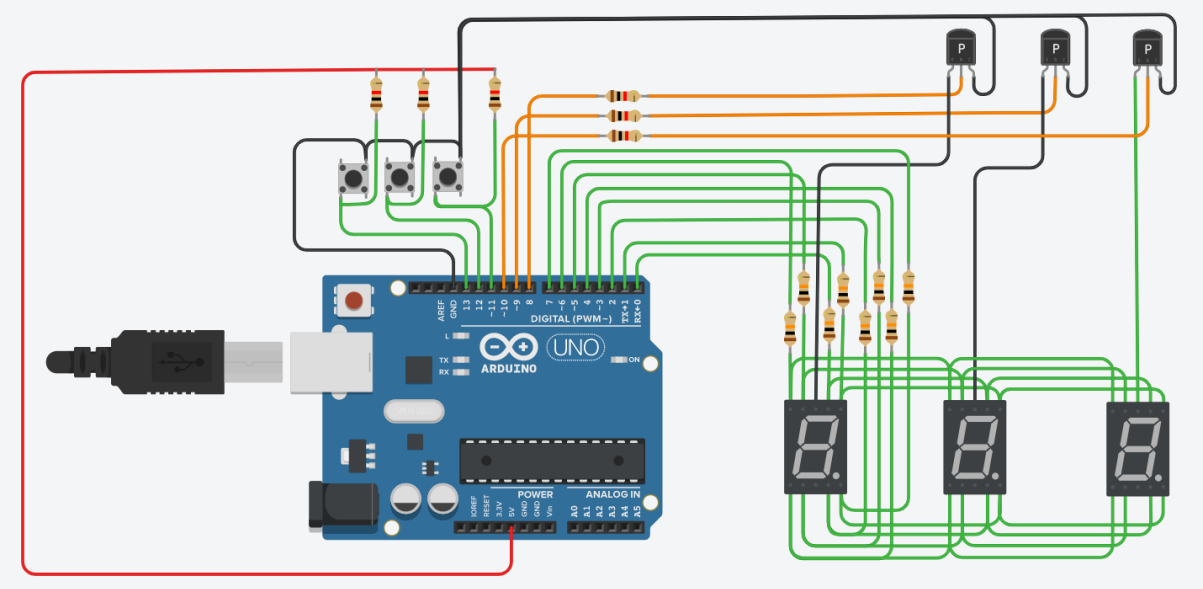


Fig. 2.10 Schema electrică pentru contorizarea numărului de pasageri ai unui avion.

```
4  // pin buton incrementare
5  const byte pinInc = 5;
6  // pin buton decrementare
7  const byte pinDec = 4;
8  // pin resetare numar pasageri
9  const byte pinReset = 3;
10
11 // pini afisaje
12 const byte pinsDisplay[N] = { 0, 1, 2 };
13
14 // starea pinilor pentru fiecare numar
```

```
15 const byte starePortNumere[10] =
16 {
17     // gfedcba (abcdefg invers)
18     0b00111111, // 0
19     0b00000110, // 1
20     0b01011011, // 2
21     0b01001111, // 3
22     0b01100110, // 4
23     0b01101101, // 5
24     0b01111101, // 6
25     0b00000111, // 7
26     0b01111111, // 8
27     0b01101111  // 9
28 };
29
30 int contor = 0;
31
32 void setup()
33 {
34     // setare tot portul D ca iesire
35     DDRD = 0b11111111;
36     // setare pini intrare/iesire port B
37     DDRB = 0b00000111;
38 }
39
40 // functie care verifica daca butonul
41 // de pe un anumit pin a fost apasat, adica
42 // daca starea lui e LOW (0); in plus, functia
43 // asteapta pana cand butonul nu mai e apasat
44 bool buton_apasat(byte pin_buton)
45 {
46     // verificare daca starea butonului e LOW (0)
47     if ((PINB & (1 << pin_buton)) == 0)
48     {
49         // asteptare cat timp butonul e LOW (0)
50         while ((PINB & (1 << pin_buton)) == 0)
51         {
52             // asteptare...
53         }
54
55         // butonul este apasat, pentru ca este LOW
```

```
56          return true;
57      }
58
59      // butonul nu este apasat
60      return false;
61  }
62
63  void loop()
64  {
65      if (buton_apasat(pinInc))
66      {
67          ++contor;
68      }
69
70      if (buton_apasat(pinDec))
71      {
72          --contor;
73      }
74
75      if (buton_apasat(pinReset))
76      {
77          contor = 0;
78      }
79
80      // pentru fiecare afisaj, afisam numarul
81      // respectiv, si activam/dezactivam ceea
82      // ce e necesar
83      for (byte i = 0; i < N; ++i)
84      {
85          // primul  afisaj, afisare sute
86          if (i == 0)
87          {
88              PORTD = starePortNumere[(contor / 100) % 10];
89          }
90          // al doilea afisaj, afisare zeci
91          else if (i == 1)
92          {
93              PORTD = starePortNumere[(contor / 10) % 10];
94          }
95          // al treilea afisaj, afisare unitati
96          else if (i == 2)
```

```
97          {
98              PORTD = starePortNumere[contor % 10];
99          }
100
101         // dezactivare afisaje
102         PORTB |= 0b00000111;
103         // activare doar cel curent
104         PORTB &= ~(1 << pinsDisplay[i]);
105
106         delay(20);
107     }
108 }
```

2.4.1.2 Aplicație propusă

Să se implementeze o aplicație software pentru microcontrolerul 328 care să permită controlul unei treceri de pietoni semaforizate. Culoarea verde a semaforului pietonilor este solicitată la cerere, prin apăsarea unui buton fără reținere. Culoarea verde a semaforului pietonilor trebuie sa dureze 30 secunde, timp în care semaforul vehiculelor au culoarea roșie. Atât semaforul pentru vehicule cât și semaforul pentru pietoni trebuie să prezinte un contor al timpului rămas până la schimbarea culorii. Simularea aplicației poate fi realizată utilizând pagina web *https://www.tinkercad.com*.

2.4.2 Utilizarea porturilor digitale I/O pentru comanda afișajelor LCD

Cel mai comod şi mai răspândit mod pentru afişarea caracterelor alfanumerice (consacrate sau definite de utilizator) este prin utilizarea dispozitivelor de afişare cu cristale lichide (LCD), în format matricial. Un astfel de dispozitiv este constituit dintr-o matrice de dimenisune fixă (de exemplu 8x8 sau 7x8 celule lichide) care devin opace, sau își schimbă culoarea, sub influența unui curent sau câmp electric. Unul dintre cele mai întâlnite afișaje LCD grupează pe 2 rânduri câte 16 astfel de matrici. Acest afișaj LCD este suficient pentru majoritatea aplicațiilor complexe care implică descrierea vizuală a unor informații. Pentru a putea realiza o astfel de aplicaţie, managementul datelor afişate este asigurat de un microcontroler, conectat la un LCD prin intermediul unui port de intrare-ieșire.

2.4.2.1 Modul de conectare al unui afişaj cu cristale lichide

Un afișaj LCD poate primii date/informații de la un microcontroler utilizând 4 sau 8 linii de comunicație (bus de date). Astfel, pentru cazul cu 8 linii (biți) de comunicare, afişajul are nevoie de: o tensiune de alimentare de +5V, GND, 8 linii I/O și 3 linii de control. Pentru cazul cu 4 linii de comunicare sunt necesare: o tensiune de alimentare de +5V, GND, 4 linii I/O și 3 linii de control. Când afişajul LCD nu este pornit liniile de date sunt TRI-STATE, ceea ce înseamnă că ele sunt în stare de înaltă impedanţă (ca şi cum ar

fi deconectate) şi astfel nu interferează cu funcționarea microcontrolerului. Microcontrolerul realizează controlul LCD-ului cu ajutorul a 3 linii de control, acestea având următoarea semnificaţie:

- linia *Enable* (E) permite accesul la afişaj prin intermediul liniilor R/W şi RS. Când această linie este LOW, LCD-ul este dezactivat şi ignoră semnalele de la R/W şi RS. Când linia (E) este HIGH, LCD-ul verifică starea celor două linii de control şi răspunde corespunzător;
- linia *Read/Write* (R/W) stabileşte direcţia datelor dintre LCD şi microcontroler. Când linia este LOW, datele sunt scrise în LCD. Când linia este HIGH, datele sunt citite de la LCD;
- linia *Register Select* (RS), este utilizată de LCD pentru interpretarea tipului de date, de pe liniile de date. Când este LOW, o instrucţiune este scrisă în LCD, iar când este HIGH un caracter este scris în LCD.

Starea logică în care se găseşte fiecare linie de control este următoarea:

Tab. 2.1 Semnificația stării logice a liniilor de control ale unui LCD.

E:	0 – Accesul la LCD dezactivat	1 – Accesul la LCD activat
R/W:	0 – Scrie date în LCD	1 – Citeşte date din LCD
RS:	0 – Instrucţiuni	1 – Caracter

Procedura pentru scrierea unui caracter pe afișajul LCD-ului este următorul:

- se seteaza bitul R/W pe valoarea logică 0;
- se seteaza bitul de RS pe valoarea logică 0 sau 1, în funcţie de utilizarea unei instrucțiuni sau de utilizarea datelor;
- se trimite date către liniile de date (dacă se execută o scriere);
- se setează linia E pe valoarea logică 1;
- se citesc date de la liniile de date (dacă se execută o citire).

Procedura pentru citirea datelor se realizează la fel, cu excepţia setării bitului R/W pe valoarea logică 1. Pentru a genera un caracter special se utilizează reprezentarea din figura 2.11. Afişajul LCD mai conţine o memorie CGRAM (Character Generator RAM). Această memorie este rezervată pentru caracterele definite de utilizator. Datele din CGRAM sunt reprezentate sub formă de caractere bitmap, de 8 biți. Pentru a afişa caracterul bitmap pe LCD, trebuie setată adresa CGRAM la punctul de start (de obicei 0) şi apoi scrise datele în afişaj. Pentru a-i ajuta pe utilizatori, IDE-ul Arduino conține o librărie care definește deja funcții specifice pentru utilizarea facilă a afișajelor de tip LCD. Aceste funcții pot fi utilizate odată cu apelarea directivei:

```
1    #include <LiquidCrystal.h>
```

în aplicația software. Descrierea completă poate fi urmărită la adresa web: *https://www.arduino.cc/en/Reference/LiquidCrystal*.

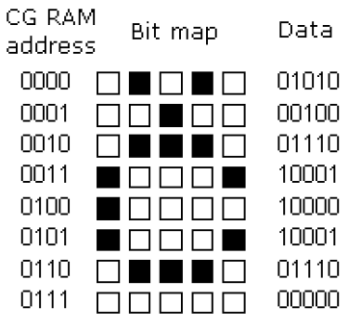


Fig. 2.11 Modul de formare al unui caracter al unui afișaj LCD.

2.4.2.2 Exemplu de aplicație

Să se elaboreze și simuleze, utilizându-se Tinkercard, un program care să afișeze un text și o valoare pe un afișaj de tip LCD. Schema electrică a aplicației propuse este prezentată în figura 2.12.

Componentele utilizate în schemă sunt:

- 1x LCD 16x2;
- 1x arduino Uno R3;
- 1x potențiometru 500 ohmi;
- 1x rezistor 1 kohm.

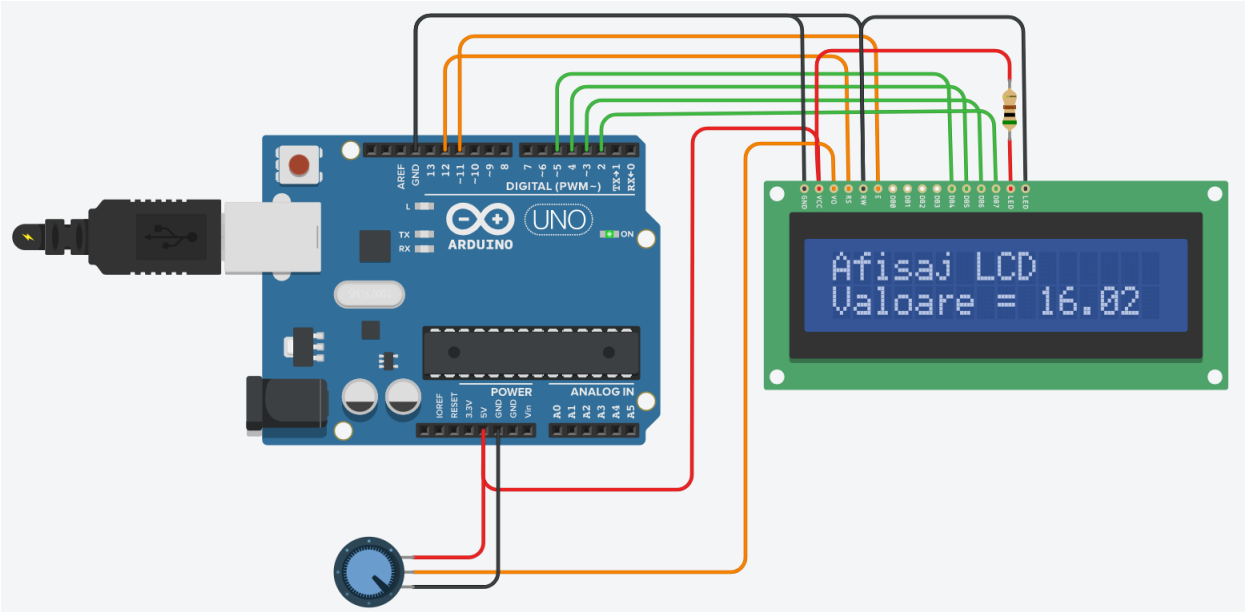


Fig. 2.12 Afișarea de caractere pe LCD.

Programul este următorul:

Algoritmul 2.6 Utilizarea afișajului LCD.

```
1 // include libraria specifica:
2 #include <LiquidCrystal.h>
3
```

```
4  /*
5    Conectarea pinilor la placa Arduino UNO:
6   * LCD RS la pinul digital 12
7   * LCD Enable la pinul digital 11
8   * LCD D4 la pinul digital 5
9   * LCD D5 la pinul digital 4
10  * LCD D6 la pinul digital 3
11  * LCD D7 la pinul digital 2
12  * LCD R/W la pinul de masa
13  * LCD VSS la pinul de masa
14  * LCD VCC la pinul 5V
15  * VO la pinul de iesire al potentiometrului
16 */
17
18 // asociaza pinii LCD-ului catre pinii placii Arduino UNO
19 const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
20 LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
21
22 void setup() {
23   // defineste nr. de lini si coloane ale LCD-ului:
24   lcd.begin(16, 2);
25   // Afiseaza un text pe LCD.
26   lcd.print("Afisaj LCD");
27 }
28
29 void loop() {
30   // seteaza cursorul pe linia 1 coloana 0
31   // atentie, linia 1 este pe cel de-al doilea rand, deoarece
       ↪ numaratoarea incepe de la 0
32   lcd.setCursor(0, 1);
33   // printeaza un text
34   lcd.print("Valoare = ");
35   lcd.setCursor(10, 1);
36   // printeaza o valoare
37   lcd.print(16.02);
38 }
```

2.4.3 Utilizarea porturilor digitale I/O pentru controlul unei tastaturi hexa-numerice

Tastatura numerică reprezintă un mod practic și facil pentru a transmite valori unui microcontroler. Un astfel de dispozitiv este construit dintr-o matrice de butoane, uzual având dimensiunea de 4x4. Aceasta permite utilizarea unui număr decimal cuprins între 0 și 9 precum și definirea a 6 butoane cu funcții speciale. Funcțiile speciale pot fi folosite după cum necesită fiecare aplicație în parte.

2.4.3.1 Modul de funcționare al unei tastaturi hexa-numerice

O tastatură hexa-numerică este compusă din 16 butoane, fără reținere, așezate sub forma unei matrici. Astfel, se definesc 8 conexiuni: 4 pentru rânduri, respectiv 4 pentru coloane. Schema electrică precum și imaginea cu o astfel de tastatură sunt prezentate în figura 2.13.

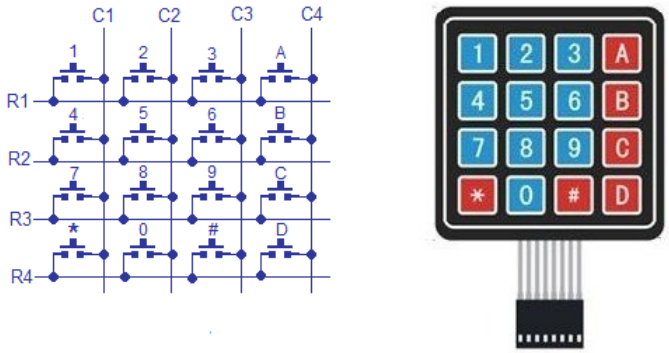


Fig. 2.13 Tastatură numerică (hexazecimală): matrice de butoane (stânga) și forma ei fizică (dreapta).

Procedura de identificare a butonului apăsat este următoarea. În primul pas se scanează coloanele. Pentru acest lucru, fiecare dintre rânduri este menținut pe rand la starea logică 0, iar celelalte rânduri sunt menținute la starea logică 1. Citirea stăriilogice a fiecărei coloane este realizată de microcontroler. Dacă o anumită coloană este găsită cu valoarea 0, acest lucru înseamnă că butonul care se află pe coloana și rândul respectiv este în scurt-circuit, adică apăsat. Procesul se repetă într-o buclă pentru a capta o nouă apăsare. Relativ la figura 2.13, dacă rândul 1 este 0 și coloana 1 este găsită, în timpul scanării, tot cu valoarea 0, atunci butonul 1 este apăsat. Pentru a evita implementarea manuală a metodei anterior aminite, se recomandă utilizarea librăriei *Keypad* a cărui cod sursă se află la adresa: *https://github.com/Chris–A/Keypad*

2.4.3.2 Exemplu de aplicație

Utilizând libraria menționată anterior, să se elaboreze și simuleze, utilizând Tinkercard, un program care să citească o parolă de la o tastatură hexa-numerică, iar dacă aceasta este identică cu valoarea *1602*, să aprindă un led verde. Pe toată perioada în care parola nu este corectă, respectiv nu toate valorile au fost introduse, să se aprindă un led roșu. Pentru a șterge valorile introduse până în momentul de față se va utiliza tasta $\star$. Resetarea parolei se face cu tasta #. Schema electrică a aplicației propuse este prezentată în figura 2.14

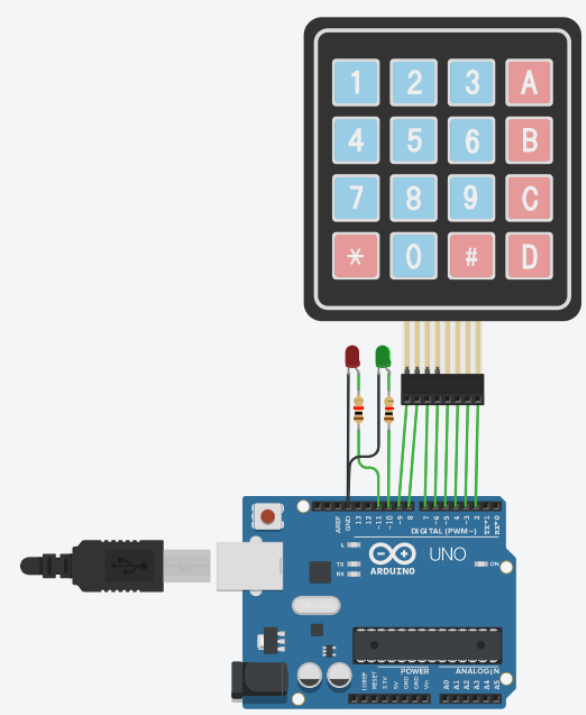


Fig. 2.14 Schema electrică pentru utilizarea tastaturii numerice.

Componentele utilizate în schemă sunt:

- 1x arduino UNO;
- 1x hex keyboard;
- 2x leduri, unul rosu și unul verde;
- 2x rezistor de 1 kohm.

Programul care realizează cerințele impuse este următorul:

Algoritmul 2.7 Utilizarea tastaturii hexa-numerice.

```
1  #include <Keypad.h>
2
3  #define DDRB0    0
4  #define DDRB1    1
5  #define DDRB2    2
6  #define DDRB3    3
7
8  #define PORTB2   2
9  #define PORTB3   3
10
11 #define DIM_PAROLA   4
12
13 #define RANDURI 4
14 #define COLOANE 4
15
16 byte contor = 0;
```

```
17
18 char valoareTasta[RANDURI][COLOANE] = {
19     { '1', '2', '3', 'A' },
20     { '4', '5', '6', 'B' },
21     { '7', '8', '9', 'C' },
22     { '*', '0', '#', 'D' }
23 };
24
25 char parolaCorecta[DIM_PAROLA] = { '1', '6', '0', '2' };
26 char parolaIntrodusa[DIM_PAROLA] = { '0', '0', '0', '0' };
27
28 byte piniRanduri[RANDURI] = { 9, 8, 7, 6 };
29 byte piniColoane[COLOANE] = { 5, 4, 3, 2 };
30
31 Keypad tastatura(
32   makeKeymap(valoareTasta),
33   piniRanduri, piniColoane,
34   RANDURI, COLOANE);
35
36 void setup()
37 {
38     // seteaza tot portul D ca port de intrare
39     DDRD = 0x00;
40     // seteaza pini iesire/intrare in portul B
41     DDRB = (0 << DDRB0) | (0 << DDRB1) | (1 << DDRB2) | (1 <<
       ↪ DDRB3);
42
43     PORTB |= 1 << PORTB3; // aprinde led rosu
44 }
45
46 bool compara_parole()
47 {
48     // pentru fiecare caracter din cele doua
49     // siruri de caractere, daca caracterele
50     // de pe acceasi pozitie nu sunt egale,
51     // atunci parolele nu sunt identice
52     for (byte i = 0; i < DIM_PAROLA; ++i)
53     {
54         if (parolaCorecta[i] != parolaIntrodusa[i])
55         {
56             return false;
```

```
57          }
58      }
59
60      return true;
61  }
62
63  void loop()
64  {
65      const char tastaApasata = tastatura.getKey();
66      if (tastaApasata != 0)
67      {
68          if (tastaApasata == '#')
69          {
70              // resetare parola
71              for (byte i = 0; i < DIM_PAROLA; ++i)
72              {
73                  parolaIntrodusa[i] = '0';
74              }
75          }
76          else
77          {
78              // retine caracterul introdus la pozitia
79              // curenta, si incrementeaza pozitia
80              parolaIntrodusa[contor] = tastaApasata;
81              ++contor;
82
83              // parola are maxim DIM_PAROLA caractere,
84              // deci trebuie resetat la 0 contorul daca
85              // depaseste valoarea respectiva
86              if (contor >= DIM_PAROLA)
87              {
88                  contor = 0;
89              }
90
91              // verifica daca parolele sunt identice
92              if (compara_parole())
93              {
94                  // stinge led rosu
95                  PORTB &= ~(1 << PORTB3);
96                  // aprinde led verde
97                  PORTB |= (1 << PORTB2);
```

```
98                      // asteapta 1 secunda
99                      delay(1000);
100                     // aprinde led rosu
101                     PORTB |= (1 << PORTB3);
102                     // stinge led verde
103                     PORTB &= ~(1 << PORTB2);
104                 }
105             }
106         }
107 }
```

2.4.3.3 Aplicație propusă

Să se implementeze o aplicație software pentru microcontroler care să rezolve operații matematice simple (înmulțire, adunare, scădere, împărțire), utilizând valori zecimale introduse de la o tastatură numerică. Considerându-se tastatura prezentată în figura 2.13, tasta A se va folosi pentru a realiza operația de adunare între două valori, B pentru operația de scădere, C pentru operația de înmulțire, respectiv D pentru operația de împărțire. Tasta $\star$ va avea funcție de ștergere (resetare) a datelor citite de la tastatura, respectiv # va avea funcția de afișare rezultat (=). Vizualizarea valorilor citite de la tastatură, cât și a rezultatelor, se va realiza pe un afișaj LCD 16x2.

3. Sistemul de întreruperi

Sistemul de întreruperi externe al micocontrolerului la ATmega328
Exemplu de utilizare a sistemului de întreruperi
Aplicație propusă

Datorită existenței unei singure unități aritmetice și logice, microcontrolerul poate executa instrucțiuni într-o manieră secvențială. Astfel, codul mașină va rula sub forma unei structuri secvențiale. Totuși, există situații neprevăzute în care trebuie se execute o altă activitate față de cea care rulează în momentul respectiv. Acest tip de eveniment poate fi executat cu ajutorul sistemului de întreruperi. În acest sens, fiecărui eveniment i se asociază o întrerupere. Apariția evenimentului respectiv va provoca executarea unei rutine de cod special scrisă de programator (rutine de deservire a întreruperii).

Întreruperile sunt des utilizate în corelare cu îndeplinirea unei condiții (interioare sau exterioare microcontrolerului). Spre deosebire de sistemul de intrare oferit de modulul PORT, întreruperile permit executarea codului indiferent de momentul apariției evenimentului. Întreruperile oferă flexibilitate în scrierea și rularea programelor. Cererea de întrerupere apare la nivel hardware, deci nu este necesară coordonarea rutinei de către programator, microcontrolerul fiind cel care gestionează această cerere.

În cadrul unui sistem cu microcontroler pot exista mai multe surse de întrerupere. Cele mai des întâlnite sunt întreruperile externe, cele provocate de timere sau întreruperile provocate de module de comunicație. Placa de dezvoltare Arduino UNO, ce are în componența sa un microcontroler ATmega328P, are 26 de surse de întrerupere. Dintre acestea, întreruperile prezentate în tabelul 3.1 prezintă interes în cadrul laboratoarelor.

Drept studiu de caz, lucrarea de fata se va concentra asupra utilizarii intreruperilor externe. Placa de dezvoltare Arduino UNO, ce are la baza microcontrolerul ATmega328, detine doi pini pe care se pot genera evenimente externe: INT0 și INT1 (pinii 2 și 3 de pe placa Arduino). Întreruperile de pe acești doi pini pot fi generate pe frontul crescător sau descrescător al semnalului de intrare sau de nivelul de LOW al acestuia. Cererea de întrerupere este implementată în hardware. O altă cerere de întrerupere poate fi dată de schimbarea valorii unui pin din portul MC-ului. Pinii afectați sunt cei prezentați în tabela vectorilor de întrerupere.

Tab. 3.1 Tipuri de întreruperi disponibile pentru microcontrolerul ATmega328P.

Vector de întrerupere	Tip întrerupere	Sursă	Nume vector
2	Întrerupere externă	INT0	INT0_vect
3	Întrerupere externă	INT1	INT1_vect
4	Întrerupere externă la nivel de port, D8-D13	PCINT0	PCINT0_vect
5	Întrerupere externă la nivel de port, A0-A5	PCINT1	PCINT1_vect
6	Întrerupere externă la nivel de port, D0-D7	PCINT2	PCINT2_vect
10	Timer/Counter2 overflow	TIMER2_OVF	TIMER2_OVF_vect
14	Timer/Counter1 overflow	TIMER1_OVF	TIMER1_OVF_vect
17	Timer/Counter0 overflow	TIMER0_OVF	TIMER2_OVF_vect
18	Transfer complet pe SPI	SPI STC	SPI_STC_vect
19	USART Rx recepție completă	USART_RX	USART_RX_vect
21	USART Tx transfer complet	USART_TX	USART_TX_vect
22	Conversie ADC completă	ADC	ADC_vect
25	Transfer complet pe I2C	TWI	TWI_vect

Pentru cazul întreruperii externe, registrele EICRA, EIMSK, EIFR și SREG sunt importante. În tabelul 3.1 sunt prezentații biții registrului EICRA (E*xternal* I*nterrupt* C*ontrol* R*egister* A). Acest registru este responsabil pentru controlul întreruperii. Biții acestui registru sunt următorii:

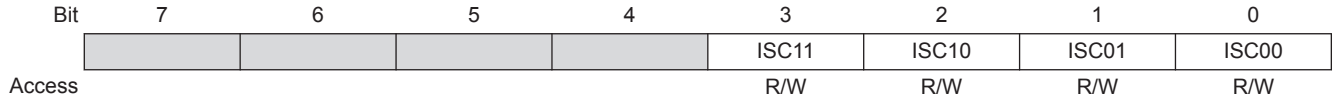


Fig. 3.1 Registrul EICRA.

– Biții 3:2 – ISC1n: Acesti biți permit controlul sensului întreruperii 1 [n=1:0]. O întrerupere externă pe pinul INT1 este funcțională dacă bitul 7 din registrul SREG este activat și masca corespunzătoare întreruperii este activată. Modurile prin care se poate genera o cerere de întrerupere pentru această sursă externă sunt date in funcție de valorile biților 3:2 astfel:
  - 00: valoarea de Low a pinului INT1;
  - 01: orice schimbare a valorii logice la pinul INT1;
  - 10: schimbarea din High în Low a valorii pinului INT1;
  - 11: schimbarea din Low în High a valorii pinului INT1.
– Biții 1:0 – ISC0n: Acesti biți permit controlul sensului întreruperii 0 [n=1:0]. O întrerupere externă pe pinul INT0 este funcțională dacă bitul 7 din registrul SREG este activat și masca corespunzătoare întreruperii este activată. Modurile prin care se poate genera o cerere de întrerupere pentru această sursă externă sunt date in funcție

de valorile biților 3:2 astfel:

- 00: valoarea de Low a pinului INT0;
- 01: orice schimbare a valorii logice la pinul INT0;
- 10: schimbarea din High în Low a valorii pinului INT0;
- 11: schimbarea din Low în High a valorii pinului INT0.

În tabelul 3.2 sunt prezentații biții registrului EIMSK (*External Interrupt Mask Register*). Acest registru este utilizat pentru a activa sau dezactiva masca unuia dintre pinii INT0 și INT1, pentru generarea de întreruperi. Biții acestui registru sunt următorii:

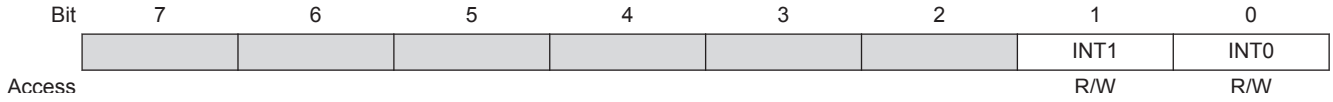


Fig. 3.2 Registrul EIMSK.

– Bitul 1 - INT1: Activare întrerupere externă 1. În momentul în care bitul INT1 este setat și bitul 7 din registrul de stare SREG este setat, întreruperea pe pin este activată. Generarea întreruperii are în vedere setările efectuate în registrul EICRA (tabelul 3.1);
– Bitul 0 - INT0: Activare întrerupere externă 0. În momentul în care bitul INT0 este setat și bitul 7 din registrul de stare SREG este setat, întreruperea pe pin este activată. Generarea întreruperii are în vedere setările efectuate în registrul EICRA (tabelul 3.1);

Registrul EIFR (*External Interrupt Flag Register*) este responsabil pentru înregistrarea apariției unei întreruperi. La apariția unei întreruperi externe biții din acest registru vor fi setati automat de catre microcontrler. Următorii doi biți fac parte din acest registru (figura 3.3:

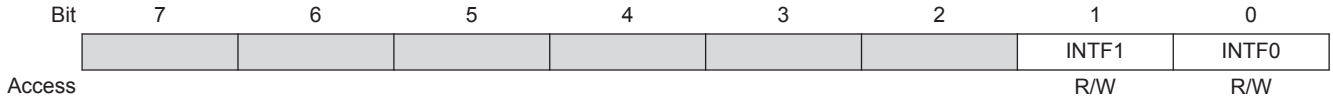


Fig. 3.3 Registrul EIFR.

– Bitul 1 - INTF1: Când o cerere de întrerupere este generată pe pinul INT1, bitul INTF1 este setat în mod automat;
– Bitul 0 - INTF0: Când o cerere de întrerupere este generată pe pinul INT0, bitul INTF0 este setat în mod automat;

Tratarea unei întreruperi se face în interiorul funcției ISR(). Aceasta este o funcție standard de apel a vectorului de întrerupere pentru microprocesoare ATmega. Această funcție poate avea doar un parametru și nu returnează nimic. Parametrii posibili sunt numele vectorilor de întrerupere, prezenți în ultima coloană a tabelului 3.1. Sintaxa funcției este:

Algoritmul 3.1 Rutina de deservire a întreruperii - sintaxă generală

```
1 ISR (isr_vector)
2 {
3     //ISR_code
```

```
4 }
```

3.1 Exemplu de aplicaţie

Să se elaboreze și simuleze, utilizându-se Tinkercad, un program care aprinde şi stinge la fiecare apăsare de buton, un led. Tratarea aprinderii/stingerii ledului trebuie să se realizeze utilizându-se sistemul de întreruperi al microcontrolerului. Schema electrică a schemei propuse este prezentată în figura 3.4. Componentele utilizate în schemă sunt:

- 1x 3 x AA Battery;
- 1x Arduino Uno R3;
- 2x 1 kohm Resistor;
- 1x Yellow LED;
- 1x Pushbutton;
- 1x Breadboard Small.

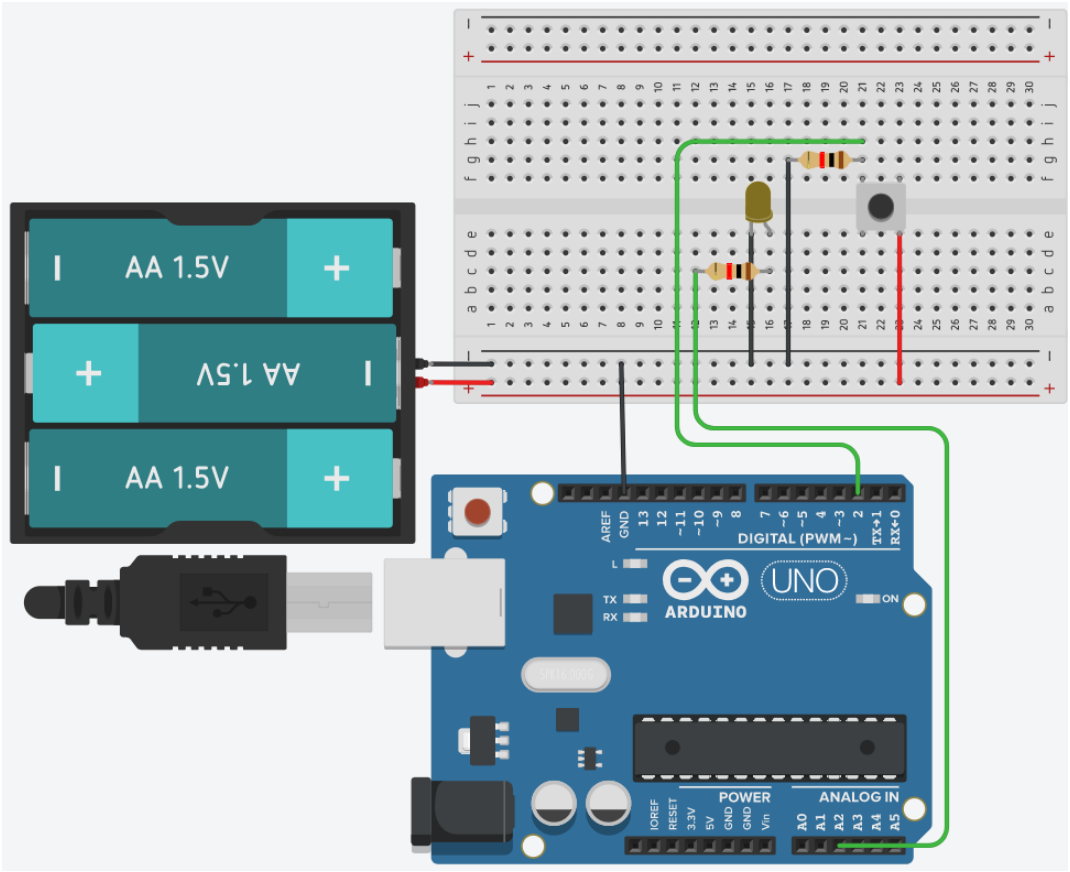


Fig. 3.4 Schema electrică de funcţionare a aplicaţiei cu întreruperi.

Programul care realizează cerinţele impuse este următorul:

Algoritmul 3.2 Utilizarea sistemului de întreruperi.

```
1 #include <avr/interrupt.h>
2
3 // definire rutina de tratare a
4 // intreruperii INT0
```

```
5  ISR(INT0_vect)
6  {
7      // dezactivare intreruperi globale
8      SREG &= ~(1 << SREG_I);
9
10     // verific starea curenta a LED-ului
11     // si o schimb in cea opusa
12     if ((PORTC & (1 << PC2)) == 0) // LOW -> HIGH
13     {
14         PORTC |= (1 << PC2);
15     }
16     else // HIGH -> LOW
17     {
18         PORTC &= ~(1 << PC2);
19     }
20
21     // activare intreruperi globale
22     SREG |= 1 << SREG_I;
23 }
24
25 void setup()
26 {
27     // configurare PORTC ca iesire
28     DDRC = 0x00;
29     // setare PORTC pe LOW
30     PORTC = 0x00;
31
32     // activare intreruperi globale
33     SREG |= (1 << SREG_I);
34
35     // configurare intreruperi externe
36     EICRA = (0 << ISC11) | (0 << ISC10) | (1 << ISC01) | (1 <<
           ↪ ISC00);
37     EIMSK = (0 << INT1) | (1 << INT0);
38     EIFR = (0 << INTF1) | (0 << INTF0);
39     PCICR = (0 << PCIE2) | (0 << PCIE1) | (0 << PCIE0);
40 }
41
42 void loop() {
43 }
```

În algoritmul anterior rutina de intrerupere pentru pinul INT0 este compusa din 3 secțiuni. La început sunt dezactivate toate intreruperile. Acest lucru previne apariția unei alte întreruperi în timp ce se tratează întreruperea curentă. În continuare se verifica starea pinui PC2 (la care este conectat LED-ul). In funcție de aceasta se inversează starea pinului pentru a obține un efect vizual pe led (stingere sau aprindere). La final este reactivat sistemul de întreruperi global al microcontroller-ului.

În interiorul funcției setup() se realizează, la început, inițializarea portului C (liniile 23 și 24). Pe pinii acestui port se realizează aprinderea led-ului. La linia 26 este activat sistemul de întreruperi global, prin trecerea bitului 7 (SREG_I) pe nivelul logic High. Liniile 29:32 permit configurarea întreruperii, astfel:

- linia 36: trecerea Low-High pe pinul INT0 generează întrerupere;
- linia 37: activare mască INT0 (activare întrerupere INT0);
- linia 38: resetarea biților ce marchează apariția unei întreruperi;
- linia 39: dezactivare întrerupere la nivel de port.

3.2 Aplicaţie propusă

Să se implementeze o aplicație software pentru microcontroler care să permită numărarea unor piese. Pentru simularea circuitului precum și testarea codului software se poate utiliza platfoma on-line Tinkercard. Afișarea rezultatului numărării se va realiza prin intermediul unui afișaj cu 7 segmente. Incrementarea valorii din contor se va realiza prin apăsarea unui buton fara reținere. Acest eveniment rebuie tratat într-o întrerupere externă, pe frontul descrescător al semnalului provenit de la buton. Decrementarea valorii din contor se va realiza prin apăsarea unui al doilea buton tot fara reținere. Acest eveniment rebuie tratat într-o întrerupere externă, pe frontul descrescător al semnalului provenit de la buton. Resetarea contorului se va face prin intermediul celui de-al 3 buton. Acesta va fi citit fără a utiliza intreruperi externe. La apasarea acestui buton, valoarea din contor va deveni 0.

4. Modulul de masurare a timpului (Timer)

Regiștrii Timer-ului
Exemplu de aplicație
Aplicație propusă

Modulul de masurare al timpului, eng. *Timer*, este resursa microcontrolerului ce permite măsurarea cuantelor de timp. Pentru a păstra o terminologie similară cu aceea a literaturii de specialitate, în cadrul acestei lucrări, modulul de timp va fi referit prin termenul *Timer*. Principala caracteristică a unui Timer este aceea că acesta este implementat hardware, adică, întrg mecanismul de cronometrare a timpului este gestionat independent de unitatea aritmetică și logică (UAL), aceasta putând fi utilizată pentru a rula programul principal.

Un Timer este capabil să cronometreze timpul scurs utilizând un aşa numit accumulator (eng. *Counter*). Valoarea acumulată este staocată în registru *TimerCouNTer* (TCNT).Fiecare incrementare a acumulatorului este realizată atunci când osciloatorul emite un puls. Fiecare puls reprezintă simbolic scurgerea unei cuante de timp. Drept exemplu, se consideră un oscilator cu o frecvență de 4.000.000 de Hz. Incrementând acumulatorului la fiecare puls al osciloatorului, când am ajunge la valoarea 4.000.000 în acumulator am ști că a trecut 1 secundă. Pentru a stoca o valoare precum 3.999.999 în memoria microcontrolerului sunt necesari 4 bytes de memorie . Constructiv, modulul Timer nu poate avea un acumulator mai mare de 1 sau 2 bytes (8 sau 16 biți) de memorie, respectiv nu poate acumula o valoare mai mare de 255, respectiv 65535. Soluția este ca Timer-ul să genereze o întrerupere atunci când a ajuns la valoarea maximă posibilă (eng. *TOP*). Dacă s-ar mai fi adăugat o singură cuantă în accumulator s-ar fi produs o așa numită depășire (eng. *overflow*). Pentru flexibilitate, modulul Timer permite definirea unui prag variabil, numit eng. *MAX*, la atingerea căruia se va realiza întreruperea. În esență, max-ul defineşte pragul care dacă este atins se generază o întrerupere de tip timer, iar TOP-ul definește valoarea maximă posibilă a acumulatorului. În întreruperea de timer generată, utilizatorul poate gestiona informația pentru a determina timpul scurs. Se consideră un Timer de 8 biți (acumulatorul poate avea valoarea maxima 255) și un oscilator de 4.000.000 Hz. Pentru a considera scurgerea unei secunde ar trebui să

numărăm 15625 (adica 4.000.000 / 256) de intreruperi de timer. Fiecare intrerupere se va genera atunci cand s-a atins valoarea maximă în acumulator.

În funcție de cum este configurat, când un acumulator ajunge la depășire, modulul Timer-ul poate realiza una din următoarele acțiuni:

- se resetează la 0 (ca și cum MAX ar fi fost egal cu TOP). În acest caz, counterul numără de la 0 până la MAX - 1, după care se întoarce din nou la zero. Modificând valoarea lui MAX (valoarea de TOP nu o putem modifica, fiind stabilită la nivel de hardware), putem face numărătorul să declanșeze evenimente mai des sau mai rar. Stabilirea valorii MAX la zero ar trebui sa aibă ca efect dezactivarea lui (și resetarea se face doar când contorul ajunge la TOP);
- începe sa numere în jos - vom vedea în continuare situații în care acest lucru poate fi util;
- generează un eveniment (la fel ca în situația 1), însă nu se resetează, ci își continuă număratul până la TOP, de unde se resetează și numără din nou, și așa mai departe;
- generează un eveniment și se oprește din numărat (se mai numește și *one-shot timer*).

La fiecare incrementare automată a registrului TCNT are loc o comparație între acest registru și o valoare stocată în registrul OCRn. Această valoare poate fi încarcată de către programator prin scrierea registrului OCRn. Dacă are loc egalitatea se generează o întrerupere, în caz contrar incrementarea continuă(vezi figura 4.1).

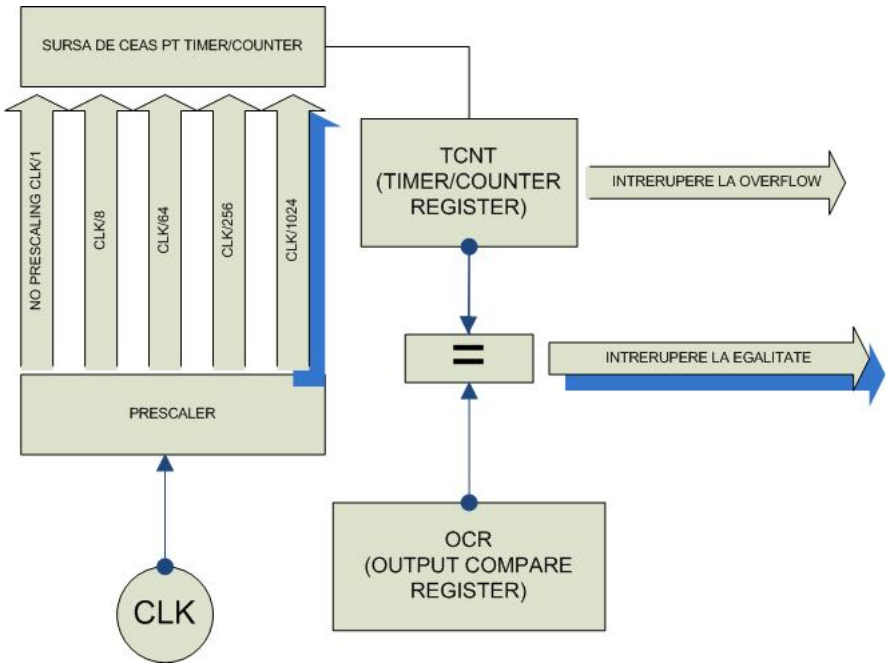


Fig. 4.1 Modul de funcționare al unui timer pentru un microcontroler din familia Atmega.

Modulul Timer este prevăzut cu mai multe registre accumulator. Acest lucru permite numărători paralele. De exemplu, microcontrolerul Amega 328P deține 3 astfel de registre, două de 8-biți și unul de 16-biți.

În funcție de valoarea până la care se face acumularea, când se face resetarea acumulatorului sau felul în care se face accumularea (increment sau decrement) pot exista următoarele

moduri de funcționare:

- *normal*. La resetare, numărătoarea începe de la 0 și depășirea se produce la 255. Acest mod este folosit uzual pentru a măsura timpul;
- *CTC (Clear Timer on Compare)*. Pornește de la valoarea 0. Registrul OCR definește valoarea la care se face depășirea.
- *fast PWM*. Pornește de la valoarea 0. Registrul OCR sau valoarea 255 definesc când se face depășirea. Pinul OC0x generează forme de undă de tip PWM;
- *phase-correct PWM*. Registrul TCNT se incrementează până la depășire după care se decrementează. Valoarea de depășire se poate defini până la OCR sau 255. Pinul OC0x generează forme de undă de tip PWM.

Descrierea detaliată a fiecărui mod de lucru al Timer-ului poate fi urmărită în foaia de catalog a fiecărui microcontroler.

4.1 Regiștrii Timer-ului

Microcontrolerul Atmega 328P deține 2 module de tip timer/counter de 8 biți, cu funcție de comparare și prescalar separat, și un timer de 16 biți cu funcții de comparare, capturare și prescalar separat.

Modul de funcționare al timerului de 8 biți poate fi urmărit în diagrama bloc din figura 4.2, unde n reprezintă numărul unității de timer, respectiv A, B reprezintă unitatea de comparare.

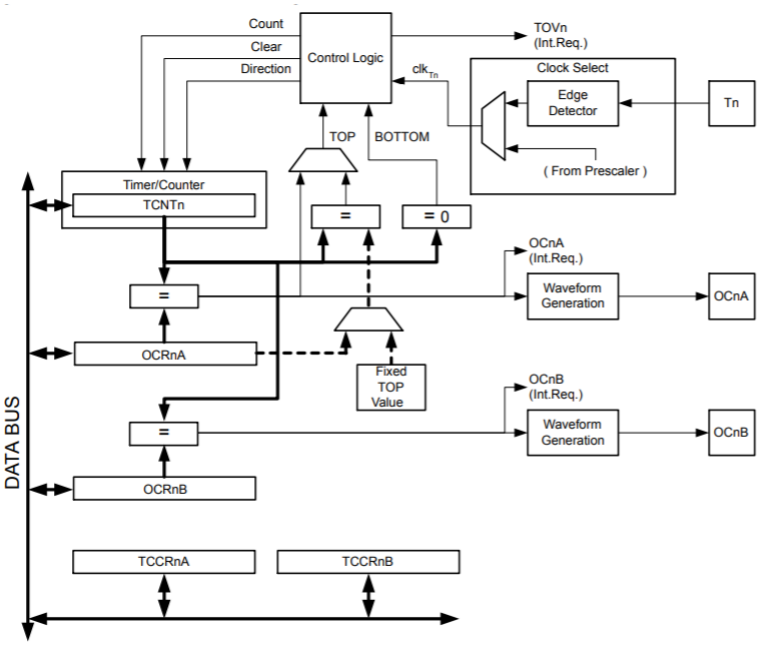


Fig. 4.2 Diagrama bloc a modulului timer de 8 biți.

Pentru exemplificare, în continuare vor fi descriși regiștrii timer-ul 0. Pentru o descriere completă a regiștrilor celorlalte module timer se recomandă consultarea documentației producătorului [2].

În figurile 4.3, respectiv 4.4 sunt prezentați regiștrii TCCRA (*Timer Counter Control Register A*), respectiv TCCR0B (*Timer Counter Control Register B*). Aceștia permit con-

figurarea modului de generare a formei de undă, respectiv a valorii prescalarului semnalului de ceas.

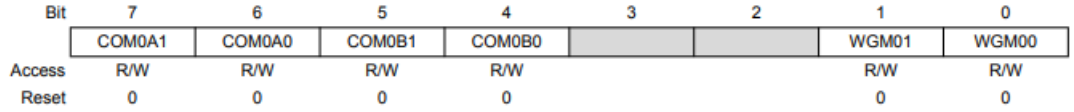


Fig. 4.3 Registrul *timer/counter control register 0A* (TCCR0A).

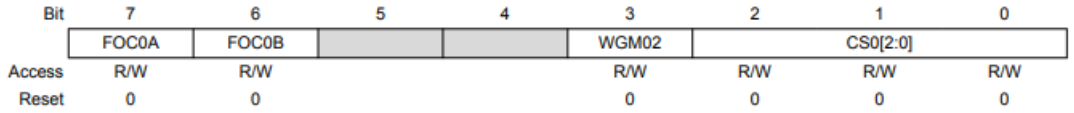


Fig. 4.4 Registrul *timer/counter control register 0B* (TCCR0B).

Principalii biți ai registrului TCCR0A sunt:

– Biții 7:6 - COM0A1 și COM0A0, se adresează doar modului de funcționare cu generare de formă de undă tip PWM. Ei prezintă următoarea funcționalitate:
  - 00: OC0A dezactivat;
  - 01: și WGM02 = 0: PORT cu funcție normală; OC0A deconectat;
  - 10: și WGM02 = 1: basculează OC0A atunci când comparatorul semnalează o potrivire cu cea a valorii de comparare;
  - 11: logică inversată (LOW la bază și HIGH la atingerea valorii de comparare);
– Biții 5:4 - COM0B1 și COM0B0, se adresează doar modului de funcționare cu generare de formă de undă tip PWM. Ei prezintă următoarea funcționalitate:
  - 00: OC0B dezactivat;
  - 01: mod de funcționare tip înapoi (eng. Reversed);
  - 10: mod de funcționare ne-inversat (eng. Non-Inverted) (HIGH la bază și LOW la atingerea valorii de comparare);
  - 11: logică inversată (LOW la bază și HIGH la atingerea valorii de comparare);
– Biții 1:0 - WGM01 și WGM00, în combinație cu WGM02 permite configurarea formei de undă generate de către timer. Funcționalitatea acestor biți este prezentată în tabelul 4.1;

Principalii biți ai registrului TCCR0B:

– Biții 7:6 - FOC0A și FOC0B (eng. F*orce* O*ut* C*ompare*). Sunt activi doar când biții WGM sunt setați pentru a configura o formă de undă de tip PWM. Acest bit nu va genera o întrerupere și nici o resetare a comparatorului atunci când se va ajunge la valoarea de top. Biții FOC0A și FOC0B sunt citiți mereu ca valoare 0;
– Bitul 3 - WGM02, în combinație cu alți regiștrii similari, permite configurarea formei de undă generate de către timer. Funcționalitatea acestui bit este prezentată în tabelul 4.1;
– Biții 2:0 - CS02, CS01 și CS00 permit configurarea prescalarului (vezi tabelul 4.2).

În figura 4.5 sunt prezentați biții registrului TIMSK0 (T*imer* C*ounter* I*nterrupt* M*ask*

Tab. 4.1 Forme de unda ce pot fi generate cu un timer.

Mod	WGM02	WGM01	WGM00	Descriere	TOP
0	0	0	0	Normal	0xFF
1	0	0	1	PWM, Phase corrected	0xFF
2	0	1	0	CTC	OCRA
3	0	1	1	Fast PWM	0xFF
4	1	0	0	Rezervat	-
5	1	0	1	Fast PWM, phase corrected	OCR0A
6	1	1	0	Rezervat	-
7	1	1	1	Fast PWM	OCR0A

Tab. 4.2 Lista prescalarilor disponibili pentru divizarea semnalului de ceas.

CS02	CS01	CS00	Descriere
0	0	0	Timer / counter dezactivat
0	0	1	Fără prescalar
0	1	0	Semnal de ceas / 8
0	1	1	Semnal de ceas / 64
1	0	0	Semnal de ceas / 256
1	0	1	Semnal de ceas / 1024
1	1	0	Sursă externă de ceas conectată la pinul T0 (front crescător)
1	1	1	Sursă externă de ceas conectată la pinul T0 (front descrescător)

R*egister*). Acest registru este utilizat pentru a activa întreruperile generate de către evenimente provenite de la circuitul comparator al timerului, după cum urmează:

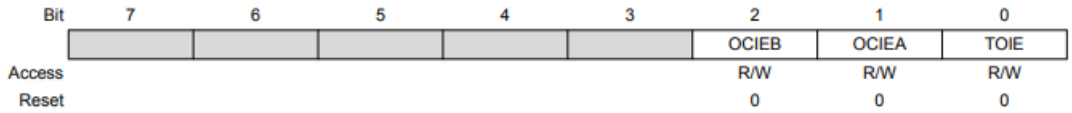


Fig. 4.5 Registrul *timer/counter interrupt mask register* (TIMSK0).

- Bitul 2 - OCIEB (eng. O*utput* C*ompare* I*nterrupt* E*nable* B). Activează (setat pe valoarea 1) sau dezactivează (setat pe valoarea 0) întreruperea generată atunci când valoarea din registrul TCNT0 este egală cu valoarea din OCR0B. Pre-condiție: bitul OCF0B trebuie să fie setat în TIFR0;
- Bitul 1 - OCIEA (eng. O*utput* C*ompare* I*nterrupt* E*nable* A). Activează (setat pe valoarea 1) sau dezactivează (setat pe valoarea 0) întreruperea generată atunci când valoarea din registrul TCNT0 este egală cu valoarea din OCR0A. Pre-condiție: bitul OCF0A trebuie să fie setat în TIFR0;
- Bitul 0 - TOIE (eng. T*imer* O*utput* I*nterupt* E*nable*). Activează (setat pe valoarea 1) sau dezactivează (setat pe valoarea 0) întreruperea generată de depășirea valori maxime acumulate (0xFF) în registrul TCNT0 (eng. overflow interrupt). Pre-condiție: bitul TOV trebuie să fie setat în TIFR0.

În figura 4.6 sunt prezentați biții registrului TIFR0 (T*imer* I*nterrupt* F*lag* R*egister*). În acest registru sunt observabile evenimentele provocate de către unitatea timer. Semnificația biților acestui registru este următoarea:

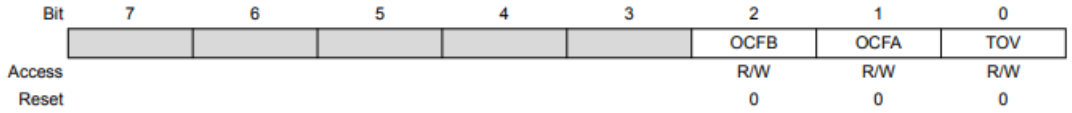


Fig. 4.6 Registrul *timer/counter interrupt flag register* (TIFR0).

- Bitul 2 - OCFB (eng. O*utput* C*ompare* F*lag* B). Acest bit este setat automat de către timer atunci când valoarea din registrul TCNT0 este egală cu valoarea din OCR0B;
- Bitul 1 - OCFA (eng. O*utput* C*ompare* F*lag* A). Acest bit este setat automat de către timer atunci când valoarea din registrul TCNT0 este egală cu valoarea din OCR0A;
- Bitul 0 - TOV (eng.T*imer* OV*erflow*). Acest bit este setat automat de către timer atunci când valoarea din registrul TCNT0 depășește valoarea de TOP (0xFF).

În figura 4.7 este prezentat registrul TCNT0 (T*imer* C*ounter* 0). Acest registru conține valoarea accumulată incremental la excitațiile oscilatorului. Biții lui permit accesul direct, pentru scriere și/sau citire, la contorul (acumulatorul) de 8 biți al timerului 0.

În figura 4.8, respectiv 4.9 sunt prezentate registrele OCR0A (O*utput* C*ompare* R*egister* 0 A), respectiv OCR0B. Acești regștrii conțin valoarile de prag ce sunt folosite de către timer pentru a genera diferite forme de undă.

Tratarea unei întreruperi generate de către un modul timer se face în interiorul funcției ISR(). După cum a fost prezentat în tabelului 3.1 din capitolul anterior, sintaxa funcției

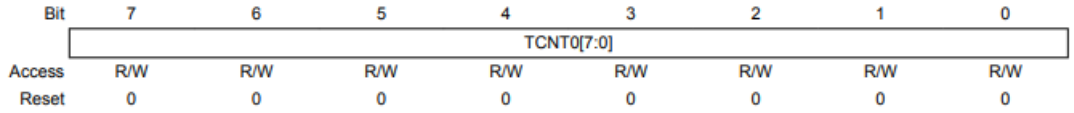


Fig. 4.7 Registrul *timer/counter register* (TCNT0) (folosit pentru a stoca valoarea acumulată).

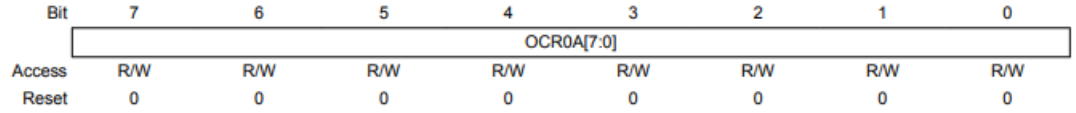


Fig. 4.8 Registrul *output compare register A* (OCR0A).

ISR, pentru diferitele surse de întrerupere provocate de timer (overflow, respectiv compare A și B), este următoarea:

Algoritmul 4.1 Rutina de deservire a întreruperii generate de depășirea valorii de TOP a registrului TCNT0

```
1
2 ISR (TIMER0_OVF_vect)
3 {
4     //ISR_code
5 }
```

Algoritmul 4.2 Rutina de deservire a întreruperii generate atunci când valoarea din registrul TCNT0 este egală cu valoarea din OCR0A

```
1
2 ISR (TIMER0_COMPA_vect)
3 {
4     //ISR_code
5 }
```

Algoritmul 4.3 Rutina de deservire a întreruperii generate atunci când valoarea din registrul TCNT0 este egală cu valoarea din OCR0B

```
1
2 ISR (TIMER0_COMPB_vect)
3 {
4     //ISR_code
5 }
```

4.2 Exemplu de aplicaţie

Să se elaboreze și simuleze, utilizându-se Tinkercad, un ceas digital folosindu-se un timer de 8 biți. Pentru reglarea orei, sistemul va conține două butoane care, tratate utilizând

Bit	7	6	5	4	3	2	1	0
	OCR0B[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Fig. 4.9 Registrul *output compare register B* (OCR0B).

sistemul de întreruperi externe, va oferi posibilitatea reglării orelor, respectiv a minutelor (incremental la fiecare apăsare de buton). Afișarea orei se va realiza pe un afișaj LCD cu 2 lini și 16 coloane. Contorizarea timpului se va face într-o rutină specială pentru întrerupere de tip timer. Se impune ca întreruperea să fie generată odată la fiecare 4 milisecunde. Schema electrică propusă este prezentată în figura 4.10. Componentele utilizate în schemă sunt:

- 1x Arduino Uno R3;
- 1x LCD 16x2;
- 2x 1 kohm Resistor (pull-down pentru butoane);
- 1x 70 ohm Resistor (pentru a stabili intensitatea scrisului);
- 1x 200 ohm Resistor(pentru a stabili iluminarea fundalului LCD-ului);
- 2x Butoane;

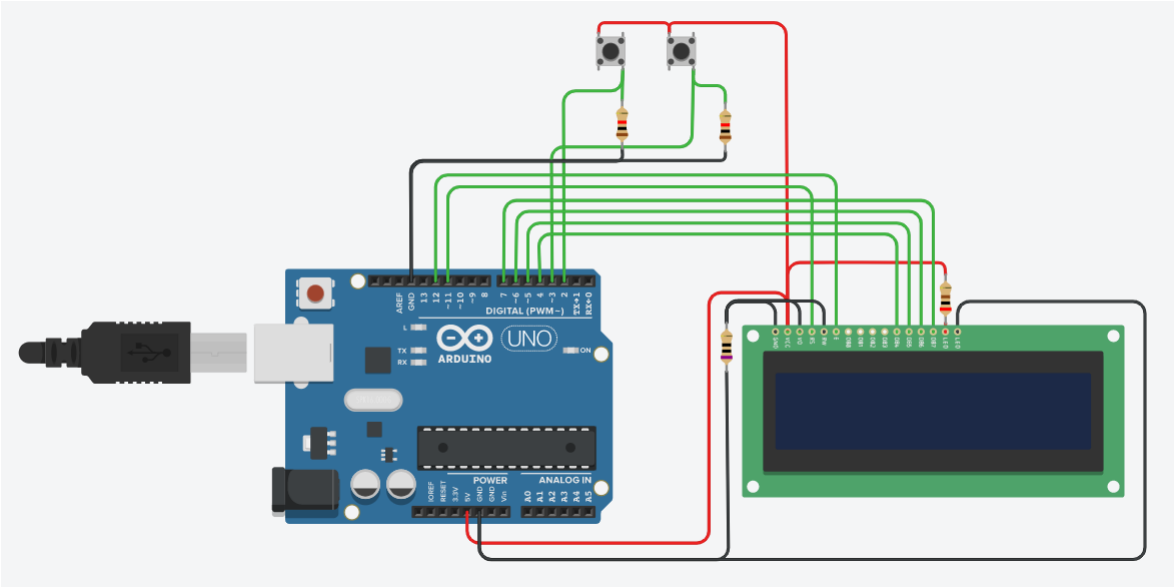


Fig. 4.10 Schema electrică de funcționare a unui ceas digital.

Programul care realizează cerințele impuse este următorul:

Algoritmul 4.4 Utilizarea modulului timer 0 pentru implementarea unui ceas digital.

```
1 /*
2  * Conectarea pinilor la placa Arduino UNO:
3  *  LCD RS -> 11
4  *  LCD E (Enable) -> 12
5  *  LCD DB4 -> 4
6  *  LCD DB5 -> 5
7  *  LCD DB6 -> 6
8  *  LCD DB7 -> 7
```

```
 9   *  LCD RW -> GND
10   *  LCD VSS -> GND
11   *  LCD V0 -> Rezistor 70Ohm -> GND
12   *  LCD VCC -> 5V
13   */
14
15  // include libraria pentru LCD
16  #include <LiquidCrystal.h>
17
18  // asociere pini Arduino <-> LCD
19  const byte RS = 11, EN = 12;
20  const byte DB4 = 4, DB5 = 5, DB6 = 6, DB7 = 7;
21  LiquidCrystal lcd(RS, EN, DB4, DB5, DB6, DB7);
22
23
24  int contor = 0;
25
26  int secunde = 0;
27  int minute = 0;
28  int ore = 0;
29
30  void setup()
31  {
32      // definire numar linii si coloane pentru LCD
33      lcd.begin(16 , 2);
34      // afisare mesaj pe prima linie a LCD-ului
35      lcd.print("Ora exacta: ");
36
37      // setare pini butoane ca pini de intrare
38      DDRD |= (1 << 2) | (1 << 3);
39
40      // configurare intreruperi externe
41      EICRA |=(1 << ISC11) | (1<< ISC10) | (1 << ISC01) | (1 <<
            ↪ ISC00);
42      EIMSK |=(1 << INT1) | (1 << INT0);
43      EIFR |=(0 << INTF1) | (0 << INTF0);
44      PCICR |=(0 << PCIE2) | (0 << PCIE1) | (0 << PCIE0);
45
46      // configurare timer sa execute o intrerupere
47      // la fiecare 4ms
48      // seteaza mod de functionare CTC
```

```
49      TCCR0A = (1 << WGM01) | (0 << WGM00);
50
51      // configurare valoare de TOP
52      OCR0A = 0xF9;
53
54      // activeaza intrerupere aferenta modului de
55      // functionare de tip comparare
56      TIMSK0 |= (1 << OCIE0A);
57
58      // configurare prescalar 256
59      TCCR0B = (1 << CS02) | (0 << CS01) | (0 << CS00);
60
61      // activare intreruperi globale
62      SREG |= (1 << SREG_I);
63  }
64
65  void loop()
66  {
67      // afisare ora
68      lcd.setCursor(0, 1);
69      if (ore < 10)
70      {
71          // daca avem mai putin de 10 ore (0-9),
72          // atunci afisam un 0 inainte (ex: 08)
73          lcd.print(0);
74          lcd.setCursor(1, 1);
75      }
76      lcd.print(ore);
77
78      // afisare minute
79      lcd.setCursor(2, 1);
80      lcd.print(":");
81      lcd.setCursor(3, 1);
82      if (minute < 10)
83      {
84          lcd.print(0);
85          lcd.setCursor(4, 1);
86      }
87      lcd.print(minute);
88
89      // afisare secunde
```

```
90      lcd.setCursor(5, 1);
91      lcd.print(":");
92      lcd.setCursor(6, 1);
93      if (secunde < 10)
94      {
95          lcd.print(0);
96          lcd.setCursor(7, 1);
97      }
98      lcd.print(secunde);
99  }
100
101 // functie pentru incrementarea orelor
102 // numarul maxim de ore pe zi este 24 (0-23)
103 inline void incrementOre()
104 {
105     ++ore;
106     if (ore >= 24)
107     {
108         // daca avem mai mult de 24 de ore,
109         // revenim la 0
110         ore %= 24;
111     }
112 }
113
114 // functie pentru incrementaerea minutelor
115 // numarul maxim de minute intr-o ora este 60 (0-59)
116 inline void incrementMinute()
117 {
118     ++minute;
119     if (minute >= 60)
120     {
121         // daca avem mai mult de 60 de minute,
122         // atunci incrementam orele
123         incrementOre();
124         // revenire la 0 minute
125         minute %= 60;
126     }
127 }
128
129 // rutina de intrerupere pentru contorizarea secundelor
130 ISR(TIMER0_COMPA_vect)
```

```
131 {
132     // dezactivare intreruperi globale
133     SREG &= ~(1 << SREG_I);
134 
135     // au mai trecut 4ms, deci incrementam contor
136     ++contor;
137 
138     // daca contorul a ajuns la 250, adica 250x4ms=1s
139     if (contor >= 250)
140     {
141         // incrementam secundele si resetam contor
142         ++secunde;
143         contor = 0;
144 
145         if (secunde >= 60)
146         {
147             // daca avem 60 de secunde, incrementam
148             // minutele si revenim la 0 secunde
149             incrementMinute();
150             secunde %= 60;
151         }
152     }
153 
154     // activare intreruperi globale
155     SREG |= (1 << SREG_I);
156 }
157 
158 // rutina intrerupere pentru INT0 (pinul 2)
159 ISR(INT0_vect)
160 {
161     SREG &= ~(1 << SREG_I);
162     // incrementam minutele
163     incrementMinute();
164     SREG |= (1 << SREG_I);
165 }
166 
167 // rutina intrerupere pentru INT1 (pinul 3)
168 ISR(INT1_vect)
169 {
170     SREG &= ~(1 << SREG_I);
171     // incrementam orele
```

```
172     incrementOre();
173     SREG |= (1 << SREG_I);
174 }
```

Pentru exemplul de aplicație mai sus prezentat s-a considerat un oscilator de 16 MHz (disponibil de altfel și placa de dezvoltare Arduino UNO) și un timer de 8 biți. Pentru cronometrarea timpului se are în vedere generarea unei întreruperi provenite de la modulul timer la fiecare 4 ms (0.004s). Se vor avea în vedere urmatoarele considerații.

Frecvența de apariție a întreruperii este:

$$f_{dorita} = \frac{1s}{0.004s} = 250Hz \tag{4.1}$$

Deoarece timer-ul este de 8 biți, valoarea de TOP a registrului accumulator TCNT0 va fi 256. Astfel, frecventa de apariție a întrerupei va fi 62500 de întreruperi pe secundă.

$$f_{realizata} = \frac{16000000}{256} = 62500Hz \tag{4.2}$$

În aceestă configurație, f_{dorita} este mult mai mică decât $f_{realizata}$. Pentru a reduce $f_{realizata}$ putem reduce frecventa oscilatorului prin utilizarea unui prescalar. Spre exemplu, se poate folosi un prescalar de 256 (vezi linia de cod 61). Noua $f_{realizata}$ devine:

$$f_{realizata} = \frac{16000000}{256 * 256} = 244.41 \tag{4.3}$$

De această dată frecvența este mai mică decât valoarea dorită. Deoarece prescalarul are valori pre-definite, pentru a obține exact valoarea dorită putem să micșorăm doar valoarea de TOP (256) cu o valoare preferențială. Această valoare va fi definită în registrul OCR0A. Pentru acest lucru trebuie să configurăm timerul să funcționeze în modul CTC (Clear Timer on Compare). Setarea acestui mod se face din registrul TCCR0A (vezi linia de cod 52). Acest mod generează o întrerupere de tip timer (TIMER0-COMPA) atunci când registrul TCNT0 este egal cu valoarea definită de utilizator în registrul OCR0A (vezi foaia de catalog a microcontrolerului).

Modul de calcul al valorii ce trebuie scrisă în registrul OCR0A (vezi linia de cod 54) este următorul:

$$\frac{16000000}{256 * n} = 250. \tag{4.4}$$

unde :

$$n = \frac{16000000}{256 * 250} = 250 = 0xFA(hexazecimal) \tag{4.5}$$

Așadar în OCR0A trebuie configurată valoarea:

$$OCR0A = 0xFA - 1 = 0xF9(hexazecimal) \tag{4.6}$$

4.3 Aplicație propusă

Să se implementeze un cronometru digital utilizând platforma de dezvoltare Arduino Uno. Rezoluția de măsurare trebuie să fie de 100 microsecunde. Afișarea timpului scurs va fi făcută pe un afișaj LCD 16x2. Pentru măsurarea timpului se impune folosirea timerului de 16 biți. Pornirea cronometrului se va face la apăsarea unui buton fără reținere. Evenimentul se impune a fi tratat într-o întrerupere externă. Oprirea / punerea pe pauză a timpului cronometrat se va realiza prin re-apăsarea aceluiași buton. Resetarea timpului cronometrat se va face prin apăsarea unui al doilea buton, tot fără reținere. La fel, evenimentul pentru resetare va fi tratat într-o întrerupere externă.

5. Convertor Analog Numeric

Funcționalități ADC
Exemplu de utilizare ADC
Aplicații propuse

Fiind un sistem digital, un microcontroller este capabil să proceseze doar semnale binare. Atunci când acesta este alimentat de la 5V, de exemplu, pe pinii de intrare ieșire percepe valoarea de 5V ca valoarea 1 în binar iar 0V ca 0 în binar. Totuși, se întâmplă deseori ca semnalul conectat la microcontroler să fie unul continuu, de exemplu cuprins în intervalul 0 - 5V. Uzual, senzorilor analogici furnizează un astfel de semnal. Pentru a o procesa un asftel de semnal,microcontrolerul dispune de un modul capabil să transforme un semnal analogic într-o valoare numerică (digitala). Acest circuit se numește convertor Analog Numeric (eng. ADC - A*nalog* to D*igital* C*onvertor*).

Convertoarele A/D integrate pe microcontrolere sunt convertoare cu aproximaţii succesive sau, mai rar, convertoare cu integrare. Rezoluţia este de 8, 10 sau 12 biţi. Modulul de conversie este prevăzut şi cu un multiplexor analogic, fiind astfel disponibile mai multe canale de intrare. Unele microcontrolere sunt echipate şi cu circuit de eşantionare/memorare. În cazul în care circuitul de eşantionare/memorare lipseşte, semnalul analogic trebuie menţinut constant pe durata unei conversii. Tensiunea de referinţă necesară convertorului poate fi generată în circuit sau, dacă nu, este necesar să fie furnizată din exterior [5].

Microcontrolerul ATMega328 deține un singur circuit ADC cu comparații succesive, pe 10 biți. Acest convertor este conectat la un multiplexor analog cu 6 canale, ceea ce permite conectarea a șase intrări distincte de tensiune analogică, la Portul C al microcontrolerului (pinii A0:A5 de pe placa Arduino Uno). Modulul de conversie conține un circuit de eșantionare și reținere integrat, acesta asigurând menținerea tensiunii de intrare la o anumită valoare, pe durata conversiei.

Modulul ADC convertește o mărime analogică de intrare într-o valoare digitală pe 10 biți, prim metoda aproximațiilor succesive [1]. Valoarea minimă care poate fi convertită este valoarea de GND iar valoare maximă este reprezentată de tensiunea de pe pinul AREF (la care se scade 1 LSB). Drept referință, se mai pot folosi AV_{CC} sau o tensiune internă de

1.1V. Atunci când se folosește AV_{CC} drept referință este necesară aplicarea unei tensiuni pe pinul AREF. Pentru folosirea referinței interne, trebuie configurat biții REFSn din registrul ADMUX.

Configurarea convertorului analog numeric este posibilă prin intermediul regiștrilor ADMUX și ADCSRA. Registrul ADMUX este responsabil pentru configurarea multiplexorului modulului ADC. Următorii doi biți fac parte din acest registru (figura 5.1:

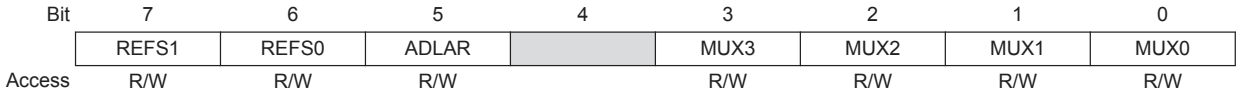


Fig. 5.1 Registrul ADMUX.

– Biții 7:6 – REFSn: Selectarea referinței [n = 1:0]: Acești biți permit selectarea tensiunii de referință pentru ADC. Dacă acești biți sunt schimbați pe parcursul unei conversii, efectul o sa fie vizibil doar după terminarea conversiei curente. Dacă o tensiune externă este aplicată pe pinul AREF, tensiunea internă de referință este posibil să nu mai poată fi utilizată. Următoarele combinații pot fi posibile:
  - 00: AREF, tensiunea internă V_{ref} este oprită;
  - 01: AV_{CC} este utilizată (este necesar un condensator pe pinul AREF);
  - 10: neutilizabil;
  - 11: Tensiune internă de 1.1V.
– Bitul 5 – ADLAR: setarea acestui bit afectează rezultatul conversiei. O valoare de 1 va realiza o ajustare la stânga a rezultatului, în timp ce valoarea 0 va ajusta la dreapta. Rezultatul ajustat al conversiei este în registrul de date ADC: ADCL și ADCH;
– Biții 3:0 – MUXn: Selectare canal analogic [n = 3:0]. Permite selectarea canalului pe care se primește semnalul analogic care urmează a fi transformat în numeric. Pentru combinația 0000 avem ADC0, iar combinația 0101 corespunde ADC5. Combinațiile între cele exemplificate corespund celorlalte canale de intrare.

Cel de al doilea registru important pentru lucrul cu ADC este registrul de control și stare ADCSRA. Următorii biți sunt conținuți de acest registru:

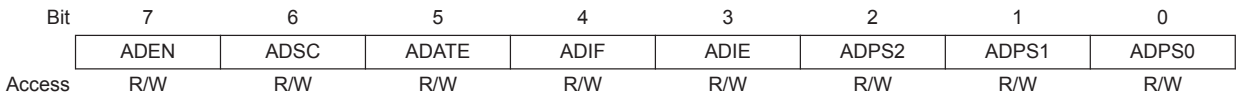


Fig. 5.2 Registrul ADCSRA.

– Bitul 7 – ADEN: Activare ADC: Prin scrierea unui 1 logic în acest bit, este activat modulul ADC. Un zero logic îl dezactivează;
– Bitul 6 – ADSC: Începere conversie ADC: prin scrierea acetui bit este pornită o conversie ADC. Dacă ADC este în modul de conversie "conversie singulară", bitul pornește conversia. În modul "conversie continuă", va determina pornirea primei conversii. În

prima conversie se realizează inițializarea convertorului și procesul de conversie durează mai mult decît în următoarele conversii (aproximativ dublu);
- Bitul 5 – ADATE: Activarea mecanismului de declanșare automată ADC: când acest bit este activat, convertorul analog numeric va porni o conversie pe frontul pozitiv al unui semnal specific de declanșare. Acest semnal este setat din registrul ADCSRB;
- Bitul 4 – ADIF: flag-ul de întrerupere ADC: Acest bit este automat setat când o conversie este finalizată și registrul de date este actualizat. Acest bit este șters în mod automat când este executată întreruperea corespunzătoare. Pentru ștergere manuală, este necesară scrierea unui 1 logic în acest registru;
- Bitul 3 – ADIE: Activare întrerupere ADC. Când acest bit este setat și bitul 7 (I) din registrul SREG este setat, finalizarea procesului de conversie analog numerică va genera o întrerupere;
- Biții 2:0 – ADPSn: selectarea prescalarului pentru ADC [n=2:0]: determină factorul de diviziune între frecvența oscilatorului plăcii de dezvoltare și semnalul de ceas utilizat de ADC. Factorul de divizare variază de la 2 (pentru 000) până la 128 (pentru 111).

Rezultatul conversiei este regăsit în două registre. Aceste registre sunt ADCL și ADCH. Pentru o singură conversie rezultatul este:

$$\mathrm{ADC} = \frac{\mathrm{V_{IN}} \cdot \mathrm{V_{REF}}}{1024} \tag{5.1}$$

5.1 Exemplu de aplicaţie

Să se elaboreze și simuleze, utilizându-se Tinkercad, un program care citește o tensiune variabilă de la un potențiometru și o afișează pe un LCD. Canalul ales pentru intrarea analogică este canalul 0. Tensiunea de referință este de 5V și este aplicată pe pinul AREF. Modificarea tensiunii de alimentare se face prin utilizarea unui potenţiometru. Schema electrică a aplicației propuse este prezentată în figura 5.3. Componentele utilizate în schemă sunt:
- 1x LCD 16x2;
- 1x Arduino Uno R3;
- 1x 220 ohm Resistor;
- 2x Potentiometru;
- 1x TMP36;
- 1x Multimetru;
- 1x Breadboard.

Programul care realizează cerinţele impuse este următorul:

Algoritmul 5.1 Utilizarea convertorului analog numeric.

```
1 #include <LiquidCrystal.h>
2
3 #define ADC_VREF_TYPE ((0<<REFS1) | (0<<REFS0) | (0<<ADLAR))
4
```

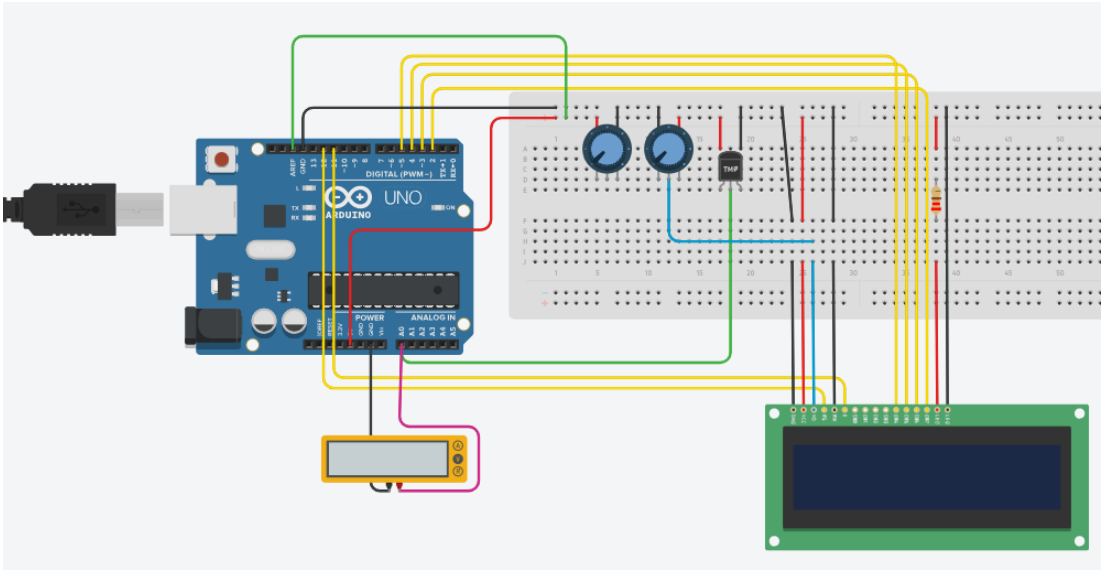


Fig. 5.3 Schema electrică pentru utilizarea modulului ADC.

```
5  /*
6    Conectarea pinilor la placa Arduino UNO:
7   * LCD RS la pinul digital 12
8   * LCD Enable la pinul digital 11
9   * LCD D4 la pinul digital 5
10  * LCD D5 la pinul digital 4
11  * LCD D6 la pinul digital 3
12  * LCD D7 la pinul digital 2
13  * LCD R/W la pinul de masa
14  * LCD VSS la pinul de masa
15  * LCD VCC la pinul 5V
16  * VO la pinul de iesire al potentiometrului
17 */
18
19 // initializare librarie LCD cu pinii corespunzatori
20 LiquidCrystal lcd(12, 11, 5, 4, 3, 2);;//RS,EN,D4,D5,D6,D7
21
22 //stepADC-ul se calculeaza conform formulei (5.1):
23 //fie N rezolutia convertorului (la Arduino N=10, adica 1024) si
     ↪ Vref tensiunea de referinta (la Arduino Vref = Vcc = 5V)
24 //step = Vref / 2^N;
25 //Arduino: step = 5 / 1024 = 0.0048828125
26 float stepADC = 0.0048828125;
27
28 void setup() {
29   // setare numar linii si coloane LCD:
```

```
30    lcd.begin(2, 16);
31    initializareADC();
32  }
33
34  void loop() {
35    // Afisare mesaj pe LCD
36    lcd.print("LCD Display");
37    // setare cursor la coloana 0, lina 1
38    lcd.setCursor(0, 1);
39    // citire intrare analogica
40    unsigned int sensorValue = citesteADC(0);
41    //determinare tenisune de intrare
42    float tensiune = stepADC*sensorValue;
43    //afisare rezultat pe LCD
44    lcd.print(tensiune, 2);
45    //repetare operatiuni la fiecare 100 milisecunde
46    delay(100);
47    //stergere informatie de pe LCD
48    lcd.clear();
49  }
50  unsigned int citesteADC(unsigned char adc_input)
51  {
52    ADMUX= adc_input | ADC_VREF_TYPE;
53    // Delay necesar pentru stabilizarea ADC
54    // pentru tensiunea de intrare analogica
55    delayMicroseconds(10);
56    // Start conversie
57    ADCSRA|=(1<<ADSC);
58    // Asteptare finalizare conversie
59    while ((ADCSRA & (1<<ADIF))==0){}
60    ADCSRA |= (1<<ADIF);
61
62    //Returnare rezultat pe 16 biti
63    return ADCW;
64  }
65
66  void initializareADC()
67  {
68    //dezactivare buffer intrare digitala
69    DIDR0=(0<<ADC5D) | (0<<ADC4D) | (0<<ADC3D) | (0<<ADC2D) | (0<<
        ↪ ADC1D) | (0<<ADC0D);
```

```
70
71    //Configurare multiplexor ADC
72    ADMUX=ADC_VREF_TYPE;
73
74    //configurare registru de control si stare
75    ADCSRA=(1<<ADEN) | (0<<ADSC) | (1<<ADATE) | (0<<ADIF) | (0<<ADIE
          ↪ ) | (1<<ADPS2) | (0<<ADPS1) | (0<<ADPS0);
76 }
```

În programul anterior au fost realizate următoarele setări:

- linia 8: setarea rezoluției ADC conform ecuației 5.1;
- linia 31: configurarea canalului de intrare a semnalului analogic;
- linia 53: configurarea registrului de stare: activare ADC, activarea mecanismului de declanșare automată ADC la funcționare continuă, factor de prescalare de 16.

Drept exemplu concret de utilizare a modulului ADC, exemplul anterior poate fi modificat prin înlocuirea potențiometrului cu un senzor de temperatură analogic. În figura 5.3 este deja adăugat senzorul de temperatură, dar trebuie conectat la canalul 0 al microcontrolerului. În lucrarea de față s-a optat pentru senzorul de temperatura TMP36. Caracteristicile sale sunt următoarele:

- tensiune de operare (2.7V - 5.5V);
- tensiunea de ieșire este proporțională cu valoarea temperaturii, exprimată în °C;
- 10mV/°C;
- ± 2°C acuratețe;
- temperatura de operare: -40°C până la +150°C;
- nu este necesară calibrarea senzorului.

Pentru a înlocuii valoarea citită de la potențiometru cu valoarea efectivă returnată de un senzor de temperatură TMP36, este nevoie de adăugarea următoarei porțiuni de cod:

Algoritmul 5.2 Utilizarea sistemului de întreruperi.

```
1 // calculare temperatura
2 float fTemperature = (stepADC*sensorValue-0.5)*100;
3 //afisare temperatura
4 lcd.print(fTemperature,3);
```

Astfel, pe ecranul LCD se va afișa valoarea temperaturii citite de la senzorul analogic utilizat. Precizia de calcul este de 10 biți, corespunzătoare capabilității maxime a microcontrolerului avut în vedere.

5.2 Aplicaţii propuse

1. Să se implementeze un mecanism de alarmă bazat pe un senzor de gaz (se poate utiliza senzorul analogic MQ2). Aplicația trebuie să fie capabilă să detecteze nivelul de gaz, iar în funcție de concentrație să se afiseze diferite mesaje pe un LCD. Se va utiliza foaia de catalog

a senzorului, pentru determinarea caracteristicilor acestuia.

2. Să se realizeze o stație de monitorizare a temperaturii, umidităţii și a presiuni. Informaţiile colectate sunt afișate pe un ecran LCD. Pentru fiecare mărime trebuie setată o valoare de prag. La depășirea pragului trebuie aprins un led roșu. Atâta timp cât valoarea măsurată este sub prag, un led verde este aprins. Valorile de prag se pot seta din butoane, iar informația este afișată pe un LCD.

6. Comunicaţia serială

Comunicația asincronă
Exemplu de utilizare a comunicației seriale asincrone
Aplicație propusă

Sistemul de comunicație serial permite transferul de date bit cu bit între doua sisteme digitale (de exemplu, două microcontrolere sau un microcontroler și un calculator).Fiind un modul necesar în majoritatea aplicațiilor integrate, majoritatea plăcilor de dezvoltare (de exemplu: Arduino Uno) conțin cel puțin un port serial (cunoscut și sub denumirea de UART sau USART). Transmiterea datelor între dispozitive implică existența a minim doua canale (fire). Canalul TX transmite date, respectiv canalul RX primește date. Canalul TX al dispozitivului care transmite se conectează la canalul RX al dispozivului care primeste și vice-versa pentru celalalt canal. În cadrul unui microcontroler dacă acești pini sunt folosiți pentru comunicația serială, ei nu mai pot fi utilizați pentru alte funcții (de exemplu ca pini de intrare/ieșire).

Mediul de dezvoltare Arduino permite utilizarea unui modul predefinit de monitorizare a sistemului de comunicații serial, prin "*Serial Monitor*" și "*Serial Plotter*", din meniul Tools. Descrierea acestor funcționalități poate fi regasită în secțiunea § 1.2.

Modulul de comunicație serială, conținut de microcontrollerul ATMega328P, permite funcționarea într-o manieră sincronă sau asincronă, cu posibilitatea de a se utiliza rate de transfer foarte mari. Transferul de date se poate face cu pachete de date de dimensiuni ce variază de la 5 biți până la 9 biți. La aceștia se adaugă 1 sau 2 biți de stop. La nivel hardware, este implementat un mecanism de detectare a erorilor de transmisie, verificarea realizându-se prin determinarea parității mesajului.

Configurarea modulului de comunicație serială se realizează prin următoarele registre: UDR0, UCSR0A, UCSR0B, UCSR0C. Primul registru, UDR0, reprezintă registrul buffer din care se face transmiterea, respectiv în care se face recepționarea datelor pe serială. Acesta are dimensiunea de 8 biți. Deoarece un mesaj transmis pe serială are o dimensiune de 5-7 biți, partea superioară neutilizată a registrului este ignorată la transmitere și setată cu zero la recepție.

La transmisie, datele din registrul buffer UDR0 sunt încărcate într-un registru de deplasare și apoi sunt transmise serializat pe pinul Tx. La recepție, datele citite pe pinul Rx sunt încărcate într-un buffer FIFO. Dupa umplerea bufferului, datele sunt copiate în registrul UDR0 unde utilizatorul are access pentru copiere.

Structura generală a Registrului de date UDR0 este prezentată în figura 6.1. Biții acestui regitru sunt aceeași cu cei ai buffer-ului de transmisie sau recepție pe serială.

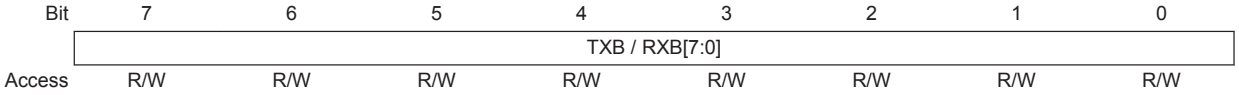


Fig. 6.1 Registrul UDR0.

Registrul de Control și stare UCSR0A conține următorii biți: (figura 6.2):

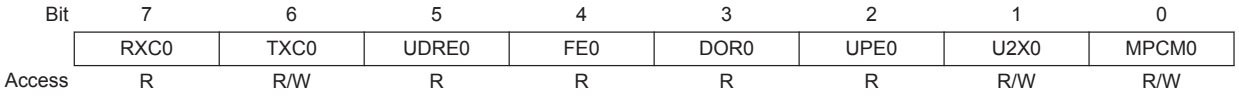
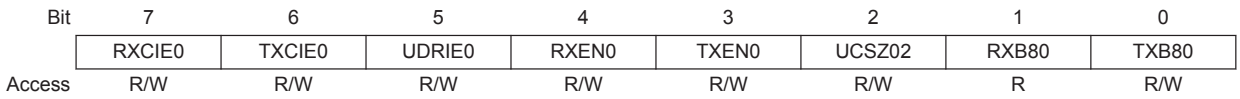


Fig. 6.2 Registrul UCSR0A.

- Bitul 7 – RXC0: Recepție completă de date: acest bit este setat când sunt date necitite în buffer-ul de recepție. Valoarea zero a acestui registru înseamnă că nu sunt date necitite;
- Bitul 6 – TXC0: Transmisie completă: valoarea acestui bit este 1 atunci când întreg mesajul din registrul de deplasare a fost transmis în exterior. Bitul este automat șters când este finalizată întreruperea de transmisie completă, atașată;
- Bitul 5 – UDRE0: Registru de date gol: dacă acest bit este 1, bufferul de date este gol și poate fi scris;
- Bitul 4 – FE0: eroare mesaj: acest bit este setat în cazul unei erori de transmisie;
- Bitul 3 – DOR0: este setat în cazul unei condiții de rulare peste date existente. O astfel de condiție poate fi când buffer-ul de recepție este plin, un nou caracter așteaptă în registrul de deplasare și este detectat un bit nou de start;
- Bitul 2 – UPE0: Eroare de paritate: acest bit este setat dacă următorul caracter în buffer-ul de recepție are o eroare de paritate;
- Bitul 1 – U2X0: dublare viteză transmisie pe serială: în mod asincron, prin setarea acestui bit se reduce factorul de divizare al ratei de transfer de la 16 la 8, astfel realizându-se dublarea vitezei de comunicație;
- Bitul 0 – MPCM0: funcționare în mod multi-procesor: activează funcționarea în mod multi-procesor.

Al treilea registru important pentru realizarea comunicației seriale este registrul de Control și stare UCSR0B figura 6.3. Biții acestui registru sunt următorii:

- bitul 7 – RXCIE0: Activare întrerupere recepție completă: setarea acestui bit va activa mecanismul de întrerupere pentru recepție;

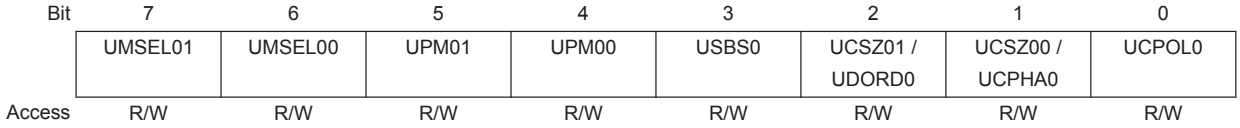


Bit	7	6	5	4	3	2	1	0
	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80
Access	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W

Fig. 6.3 Registrul UCSR0B.

- bitul 6 – TXCIE0: Activare întrerupere transmisie completă: setarea acestui bit va activa mecanismul de întrerupere pentru transmisie;
- bitul 5 – UDRIE0: Activare întrerupere registru de date gol: activarea acestui bit activează întreruperea de registru gol;
- bitul 4 – RXEN0: Activare recepție: setarea acestui bit conduce la activarea mecanismului de recepție;
- bitul 3 – TXEN0: Activare transmisie: setarea acestui bit conduce la activarea mecanismului de transmisie;
- bitul 2 – UCSZ02: dimensiune caracter: În combinație cu biții UCSZ0[1:0] din registrul UCSR0C, permite setarea numărului de biți de date transmiși sau recepţionaţi într-un mesaj;
- bitul 1 – RXB80: Bitul 9 al unui mesaj pe 9 biți la recepție. Trebuie citit înaintea citirii biților din UDR0;
- bitul 0 – TXB80: Bitul 9 al unui mesaj pe 9 biți la transmisie. Trebuie scris înaintea scrierii biților din UDR0.

Registrul UCSR0C permite configurarea formei mesajului ce urmează a fi transmis sau recepționat. Următorii doi biți fac parte din acest registru (figura 6.4):



Bit	7	6	5	4	3	2	1	0
	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01 / UDORD0	UCSZ00 / UCPHA0	UCPOL0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Fig. 6.4 Registrul UCSR0C.

- Biții 7:6 – UMSEL0n: selectare mod [n=1:0]: acești biți permit setarea modului de funcționare USART0, astfel:
  - 00: comunicație serială asincronă;
  - 01: comunicație serială sincronă;
  - 10: rezervat;
  - 11: Master SPI.
- Biții 5:4 – UPM0n: mod paritate [n=1:0]: Acești biți activează și setează tipul de paritate generat și verificat. Dacă este activat, transmițătorul va genera și va trimite automat paritatea mesajului curent. Receptorul va genera la rândul său paritatea mesajului recepționat. Următoarele combinații sunt posibile:
  - 00: dezactivat;
  - 01: rezervat;
  - 10: activat, paritate pară;

- 11: activat, paritate impară.

– Bitul 3 – USBS0: selecție număr de biți de stop: acest bit permite selectarea numărului de biți de stop. Valoarea zero reprezintă un bit de stop, iar valoarea unu corespunde pentru doi biți de stop;
– Biții 2:1 – UCSZ0n: dimensiune mesaj/ordinea datelor [n=1:0]: biții UCSZ0[1:0] combinați cu bitul UCSZ02 din registrul UCSR0B permit setarea numărului de biți de date (dimensiunea mesajului) la recepție și transmisie. Următoarele combinații pot fi posibile:
  - 000: 5 biți de date;
  - 001: 6 biți de date;
  - 010: 7 biți de date;
  - 011: 8 biți de date;
  - 100: rezervat;
  - 101: rezervat;
  - 110: rezervat;
  - 111: 9 biți de date.
– Bitul 0 – UCPOL0: Polaritate semnal ceas.

6.1 Exemplu de aplicaţie

Să se elaboreze și simuleze, utilizându-se Tinkercad, un program care transmite pe serială mesajul "Laborator SMC". Pentru aceasta se va utiliza un modul Arduino Uno și funcționalitatea de Serial Monitor.

Programul care realizează cerinţele impuse este următorul:

Algoritmul 6.1 Utilizarea sistemului de comunicații seriale.

```
1  /* UART SERIAL DEFINES */
2  #define BAUD 9600
3  #define MYUBRR F_CPU/16/BAUD-1
4
5  /* configurare UART */
6  void USART_Init( unsigned int ubrr)
7  {
8     /*Setare baud rate */
9     UBRR0H = (unsigned char)(ubrr>>8);
10    UBRR0L = (unsigned char)ubrr;
11
12   /*activare emitor si receptor */
13    UCSR0B = (1<<RXEN0)|(1<<TXEN0);
14
15    /* setare format: 8data, 2stop bit */
16    UCSR0C = (1<<USBS0)|(3<<UCSZ00);
```

```
17 }
18
19 /* Functia de transmitere pe seriala*/
20 void USART_Transmit( unsigned char data )
21 {
22     while(!(UCSR0A & (1<<UDRE0)));
23     UDR0 = data;
24 }
25
26 //Functia de receive date poe seriala
27 unsigned char USART_Receive( void )
28 {
29    /* Wait for data to be received */
30     while ( !(UCSR0A & (1<<RXC0)) )
31
32     /* Preliare si returnare a datelor receptionate din buffer */
33     return UDR0;
34 }
35
36 //Functie pentru transmiterea unui sir de caractere pe seriala
37 void SendString(char *StringPtr)
38 {
39     while(*StringPtr != 0x00)
40     {
41     USART_Transmit(*StringPtr);
42     StringPtr++;
43     }
44 }
45
46 void setup ()
47 {
48     //Setare comunicatie Seriala
49     USART_Init(MYUBRR);
50 }
51
52 unsigned char c;
53 void loop(){
54         SendString("Laborator SMC\n");
55         delay(1000);
56 }
```

În programul anterior este prezentat un exemplu de implementare a funcţiilor de scriere și citire folosindu-se modulul de comunicație serială. Acest program conține, pe lângă funcțiile de scriere și citire, și o metodă prin care se poate transfera un șir de caractere pe serială.

Programul conține la început două directive de preprocesare. Acestea permit definirea frecvenței la care se dorește realizarea transferului de date pe modulul serial.

Funcția "USART_Int" permite configurarea modulului de comunicație serială. Aceasta presupune setarea ratei de transfer între emitor și receptor, activarea modulului de emitor și receptor, configurarea pachetului transmis. Funcțiile de transmitere și recepționare date permit transferul sau recepția unui byte, folosindu-se registrele descrise anterior. Funcția "SendString" permite trimiterea unui șir de caractere pe serială. Principiul constă în transferul byte cu byte al mesajului.

În funcția de inițializare este facută setarea modulului. În bucla infinită, la fiecare secundă, este transferat mesajul dorit pe serială.

6.2 Aplicaţie propusă

Să se implementeze un mecanism care citește pe serială o comandă pentru un modul PWM. Comanda PWM este atașată unui motor de cc. Comanda este reprezentată de un șir de caractere, format din 2 bytes. După ce comanda a fost trimisă către modulul de PWM, este trimis pe serială un mesaj de succes.

7. Aplicatii

Stație meteo

Această secțiune își propune să prezinte mai multe aplicații practice care să exemplifice utilizarea cunoștințelor acumulate pe parcursul capitolelor anteriaore.

7.1 Stație meteo

Această aplicaţie presupune utilizarea mai multor senzori, pentru a se realiza o stație meteo. Pentru aceasta se va folosi un senzor digital de temperatură și umiditate (DHT22) cât și un senzor analogic de ploaie. Acești senzori se vor conecta la o placă de dezvoltare Arduino Uno. Rezultatul se poate vizualiza pe orice dispozitiv de afișare prezentat în laboratoarele anterioare.

7.1.1 Utilizarea senzorului DHT22

Senzorul digital DHT22 este un senzor care permite măsurarea relativă a temperaturii și a umidității. Pentru a se realiza măsurarea umidității se utilizează un senzor capacitiv, iar pentru a măsura temperatura un termistor. Ambele componente sunt împachetate într-un chip cu 4 pini. Pinii corespunzători sunt cei din figura 7.1. Primul pin reprezintă sursa de alimentare. Aceasta trebuie să fie una continuă, cu valori între 3-5V. Pinul 2 este pinul de date. Senzorul comunică pe un singur fir de date ("One Wire Interface"). Pinul numărul 3 este neconectat. Ultimul pin este cel care se conectează la masă.

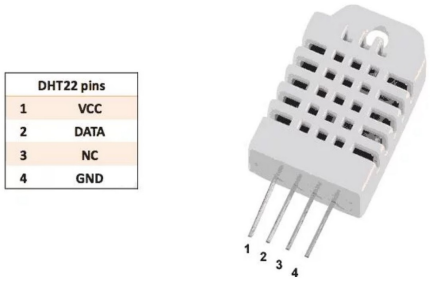


Fig. 7.1 Pinii senzorului de temperatură și umiditate DHT22.

Caracteristicile senzorului sunt:

– Tensiune de alimentare între 3-5V;
– Curent maxim: 2.5mA;
– Umiditatea măsurată: 0-100%, cu o acuratețe între 2-5%;
– Temperatura măsurată: de la $-40°C$ până la $80°C$, cu o acuratețe de $\pm 0.5°C$.

Senzorul de temperatură utilizat în această aplicație poate fi înlocuit și cu DHT11 sau DHT21. Toți fac parte din aceiași clasă de senzori și utilizează același protocol de comunicație.

Citirea datelor de la senzorul DHT22 se va face utilizând-se librăria *DHT_sensor_library*. Pentru a putea fi utilizată aceasta trebuie inclusă în configurația proiectului. Accesul la funcțiile deja definite, de lucru cu librăria, se face prin includerea fișierului corespunzător librăriei dar și prin declararea clasei pentru lucrul cu senzorul. Următoarea secvență de cod exemplifică și celelalte setări de care mai este nevoie pentru a se utiliza librăria. În figura 7.2 este prezentată schema electrică de conectare a senzorului DHT la placa de dezvoltare Arduino Uno.

Algoritmul 7.1 Configurații necesare pentru utilizarea librăriei DHT.

```
1
2  #include "DHT.h"
3
4  //setare pin citire date de la senzor
5  #define DHT_PIN 2
6  //setare tip senzor
7  #define DHTTYPE DHT22
8
9  DHT dht(DHT_PIN, DHTTYPE);
10
11 void setup()
12 {
13     //Initializare comunicatie seriala
14     Serial.begin(9600);
15     //activarea senzorului DHT
16     dht.begin();
17     // Este necesara o intarziere pentru a se realiza
       ↪ initializarea
18     delay(1000);
19 }
20
21 void loop()
22 {
23 }
```

Citirea temperaturii se realizează ușor, folosind clasa DHT din cadrul librăriei. Această

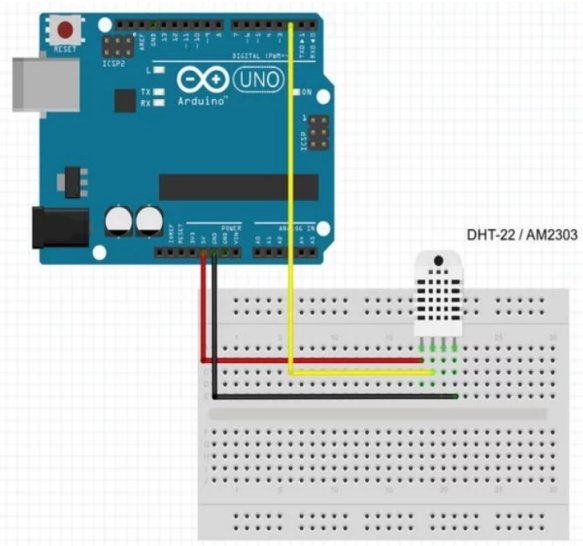


Fig. 7.2 Conectarea senzorului DHT22 la placa Arduino Uno.

clasă oferă funcții de acces direct la senzor și returnează valoarea temperaturii sau a umidității. Valoarea temperaturii poate fi returnată în grade Celsius sau în grade Fahrenheit. Un aspect important care trebuie luat în considerare este faptul că citirea de la senzor se poate face doar la un interval de maxim 0.5 secunde.

Următoarea secvență de cod demonstrează citirea de la senzor a valorii temperaturii/umidității și afișarea valorilor în consolă (pe linia de comunicații serială). Pe lângă citirea informației de la senzor mai este introdusă o linie de cod care verifică validitatea datelor citite de la senzor. În caz de eroare, aceasta este semnalată în consolă sub formă de mesaj.

Algoritmul 7.2 Configurații necesare pentru utilizarea librăriei DHT.

```
1  void loop() {
2      // Citirea temperaturii si a umiditatii mediului dureaza
         ↪ aproximativ 250ms
3      //citire umiditate
4      float h = dht.readHumidity();
5      //Citire temperatura in grade Celsius
6      float t = dht.readTemperature();
7      // Citire temperatura in grade Fahrenheit (isFahrenheit = true
         ↪ )
8      float f = dht.readTemperature(true);
9
10     // Verificam daca citirile au fost corecte
11     if (isnan(h) || isnan(t) || isnan(f))
12     {
13         Serial.println("Eroare la citirea de la senzorul DHT!");
14         return;
15     }
16
```

```
17      Serial.print("Humidity: ");
18      Serial.print(h);
19      Serial.print(" %\t");
20      Serial.print("Temperature: ");
21      Serial.print(t);
22      Serial.print(" *C ");
23      Serial.print(f);
24      Serial.print(" *F\t");
25
26      // Este necesar un delay intre citiri
27      delay(1000);
28  }
```

7.1.2 Utilizarea senzorului de ploaie

Senzorul de detectare a ploii este un mecanism foarte ușor de folosit și care permite detectarea ploii. El funcționează ca un switch, când picăturile de ploaie cad pe placa de citire a datelor. De asemenea, senzorul poate măsura și intensitatea căderii ploii. Sistemul senzorial este format din două componente: placa sensibilă la căderea ploii și o placă de control (separată de placa de citire).

Modulul de control este bazat pe circuitul LM393. Pe măsură ce picăturile de apă cad pe placa de colectare, ele crează trasee paralele care cresc rezistența circuitului. Aceste variații sunt măsurate de unitatea de control.

Senzorul, din punct de vedere funcțional, este un dipol rezistiv, adică va arăta o rezistentă mai mică atunci când este umiditate pe el și returnează o rezistență mai mare când este uscat. Când picături de apă sunt prezente, ele reduc rezistența (apa este un bun conducător electric) prin conectarea liniilor de nichel de pe placa senzorială.

Unitatea de control este prezentată în figura 7.3. Semnificația elementelor componente este următoarea:

- Pinii A: pinii de conectare la placa senzorială;
- Potențiometrul B: permite ajustarea sensibilității citirii datelor de la placa senzorială;
- Pinul C: Pinul de ieșire analogică (A0). Valoarea pe pin variază între 0 și 5V. O tensiune mai mică reprezintă o umiditate mai mare;
- Pinul D: Pinul de ieșire analogică (D0) devine zero atunci când umiditatea depășește un prag presetat;
- Pinul E: Pin care este legat la masă (GND);
- Pinul F: Pin care se leagă la o sursă de alimentare (+5V).

Conectarea senzorului la placa de dezvoltare Arduino Uno se realizează conform cu schema din figura 7.4. Citirea de la senzor se face pe intrarea analogică A0. Ceilalți doi pini utilizați sunt alimentarea și semnalul de masă.

Următoarea secvență de cod prezită modul de citire și interpretare a datelor de la senzorul

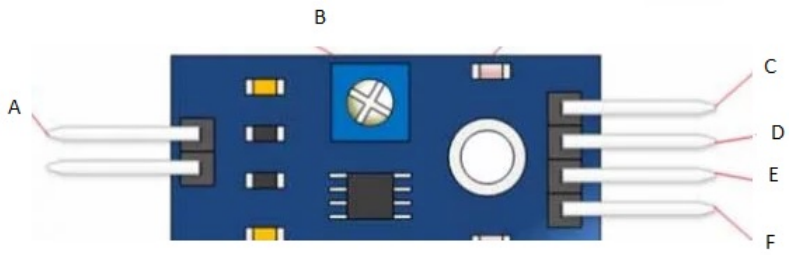


Fig. 7.3 Unitatea de control a senzorului de ploaie.

de ploaie. Informația primită este afișată în consolă, utilizându-se comunicația serială.

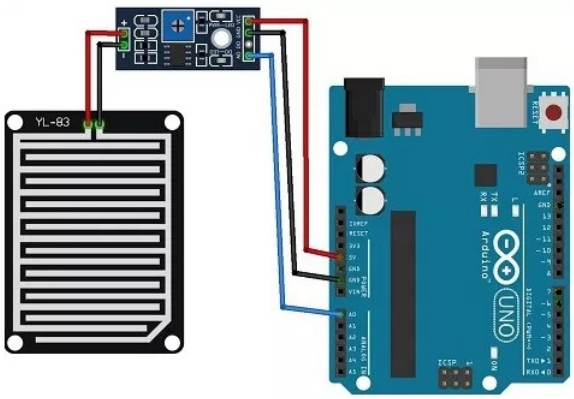


Fig. 7.4 Schema electrică pentru conectarea senzorului de ploaie la Arduino Uno.

Algoritmul 7.3 Citirea și afișarea datelor de la senzorul de ploaie.

```
1  const int sensorMinVal = 0;
2  const int sensorMaxVal = 1024;
3
4  void setup()
5  {
6      Serial.begin(9600);
7  }
8
9  void loop()
10 {
11     //Citire de la senzor, pe pinul A0
12     int citireSenzor = analogRead(A0);
13     int range = map(citireSenzor, sensorMinVal, sensorMaxVal, 0,
           ↪ 3);
14
15     switch (range)
16     {
17         case 0:
18             Serial.println("Ploaie puternica");
```

```
19                 break;
20             case 1:
21                 Serial.println("Ploaie usoara");
22                 break;
23             case 2:
24                 Serial.println("Fara ploaie");
25                 break;
26             default:
27                 Serial.println("Stare necunoscuta");
28                 break;
29       }
30     delay(1000);
31
32 }
```

În exemplul anterior a fost făcută o mapare (categorisire) a datelor citite de la senzorul de ploaie, astfel încât să putem interpreta valorile citite. Maparea este una statică, fără a se ține cont de experimente anterioare.

7.1.3 Aplicaţie propusă

Pornind de la cele două module prezentate anterior, să se creeze o aplicație care să citească datele de la cei doi senzori (temperatura/umiditate si ploaie) și să afișeze pe un LCD 16x2 datele colectate de la senzori.

7.2 Controlul unui proces utilizându-se un regulator PID

Într-un sistem de reglare, rolul unui regulator este unul de decizie. El primește la intrare un semnal de referință și un semnal măsurat. După procesarea celor două mărimi, regulatorul elaborează un semnal de comandă pentru elementul de execuție (prin intermediul căruia se intervine direct asupra procesului reglat).

În practică, cele mai utilizate regulatoare sunt cele de tip Proporțional (P), Proporțional-Integrativ (PI), Proporțional-Derivativ (PD) și Proporțional-Integrativ-Derivativ (PID). Pentru exemplul propus, o să utilizăm un regulator de tip PID. Rolul fiecărei componente, din cadrul regulatorului, în control este următorul:

- componenta proporțională: permite o ajustare a mărimii de ieșire, proporțională cu valoarea erorii. Doar această componentă folosită ca și regulator asigură foarte rar eroare staționară nulă;
- componenta integrativă: pentru a elimina eroarea staționară, este introdusă componenta integrativă. Un astfel de regulator produce ajustări care au la bază eroarea acumulată pe parcursul timpului de evoluție a procesului;
- componenta derivativă: această componentă este responsabilă cu rata de variație (schimbare) a erorii. Această componentă este responsabilă cu predicția erorii. În

principiu, cu cât răspunsul sistemului se apropie mai rapid de referință, cu atât componenta derivativă va încetini evoluția răspunsului sistemului, astfel încât să nu se depășească mărimea de referință. Se foloseste de obicei la procese lente.

Schema unui sistem în buclă închisă, care utilizează un regulator PID pentru controlul unui proces este prezentată în figura 7.5.

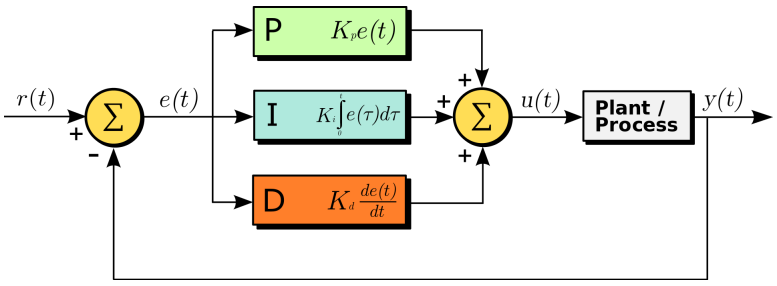


Fig. 7.5 Controlul unui proces utilizându-se un regulator PID.

În schema de control cu regulator PID sunt implicate următoarele mărimi:

– semnalul $r(t)$ reprezintă mărimea de intrare de referință;
– mărimea $y(t)$ reprezintă semnalul de ieșire;
– mărimea controlată este dată de $u(t)$;
– mărimea de eroare este $e(t)$.

Pentru a se obține performanțele dorite, parametrii unui regulator trebuie acordați. Acest proces presupune determinarea constantelor **K** din figura 7.5. De asemenea, calculele constantelor PID trebuie să se desfășoare la intervale cunoscute și exacte de timp.

Pentru implementarea unui regulator PID în cod Arduino este nevoie să se cunoască cinci parametrii: constantele de tip proporțional, integrativ și derivativ, mărimea de referință și mărimea de intrare. Calculele pentru PID trebuie să se desfășoare la intervale cunoscute de timp. Pentru aceasta se recomandă utilizarea de întreruperi de timer. Pentru aplicația propusă o să utilizăm valoarea de 100ms pentru apariția unei întreruperi.

Pentru a se determina eroarea se folosește expresia:

```
1    err = marimeReferinta - marimeIntrare;
```

Componenta integrativă face referire la eroarea acumulată, astfel avem:

```
1    errAcumulata += err;
```

Componenta derivativă a erorii reprezintă rata de variație a erorii:

```
1    errRata = err - errAnterioara;
```

Avându-se în vedere mărimile anterior amintite se poate determina valoarea mărimii de ieșire, astfel:

```
1    iesire = Kp * err + Ki * errAcumulata + Kd * errRata;
```

Valorile constantelor Kp, Ki, Kd sunt predefinite. După determinarea mărimii de ieșire se calculează o valoare temporară a erorii, care o sa fie utilizată în iterația următoare:

```
1       errAnterioara = err;
```

Aplicația propusă presupune controlul unui motor electric de curent continuu, astfel încât acesta să ajungă la o poziție predefinită. Această poziție se obține de la un encoder rezistiv. Valoarea acestuia poate fi determinată cu funcția de citire a unei mărimi analogice. Controlul motorului se face cu ajutorul unui semnal PWM, generat cu ajutorul unui timer. Constantele necesare în program au fost alese empiric.

Algoritmul 7.4 Implementare regulator PID pentru controlul unui motor de curent continuu.

```
1
2  long encoder_ticks = 0;
3  const float wheelPerimeter = 0.31416f;
4  const long ticksPerRevolution = 2240;
5  float speed = 0;
6  float distance = 0;
7  const float invTe = 10.f; // 1/Te
8
9  struct PIDstruct
10 {
11     double Kp;
12     double Ki;
13     double Kd;
14
15     double referinta;
16
17     double errAcumulata;
18     double errAnterioara;
19 } pidMotor;
20
21 // se poate si cu referinte, in locul pointerilor
22 void initializarePID(PIDstruct* pid, double Kp, double Ki, double
       ↪ Kd)
23 {
24     // setare constante regulator
25     pid->Kp = Kp;
26     pid->Ki = Ki;
27     pid->Kd = Kd;
28
29     pid->referinta = 0;
30
31     // setare valori initiale pentru erori
```

```
32      pid->errAcumulata = 0;
33      pid->errAnterioara = 0;
34  }
35
36  void setareReferintaPID(PIDstruct* pid, double ref)
37  {
38      pid->referinta = ref;
39  }
40
41  double calculareIesirePID(PIDstruct* pid, double marimeIesire)
42  {
43      // calculare eroare curenta
44      const double err = pid->referinta - marimeIesire;
45      // aproximare derivata eroare
46      const double errRata = (err - pid->errAnterioara);
47      // aproximare integrare eroare
48      pid->errAcumulata += err; //
49
50      // aplicarea formulei de calcul pentru PID
51      return (pid->Kp * err) + (pid->Ki * pid->errAcumulata) + (pid
            ↪ ->Kd * errRata);
52  }
53
54  void configurare_timer()
55  {
56    // Configurare Timer 1 pentru intrerupere la 10 Hz:
57    // oprire intreruperi
58    cli();
59
60    // initializare registru de control TCCR1A
61    TCCR1A = 0;
62    // initializare registru de control TCCR1B
63    TCCR1B = 0;
64    // initializare contor la 0
65    TCNT1  = 0;
66
67    // setare OCR pentru intrerupere la 10 Hz
68    // frecventa = 16000000 / (64 * 10) - 1 (trebuie sa fie <65536)
69    OCR1A = 24999;
70
71    // porneste CTC mode
```

```
72    TCCR1B |= (1 << WGM12);
73    // Setare CS12, CS11 si CS10 bits pentru presacalar pe 64 biti
74    TCCR1B |= (0 << CS12) | (1 << CS11) | (1 << CS10);
75    // activare intrerupere timer compare
76    TIMSK1 |= (1 << OCIE1A);
77
78    // activare intreruperi globale
79    sei();
80  }
81
82  void configurare_intrerupere()
83  {
84    // dezactivare intreruperi globale
85    SREG &= ~(1 << SREG_I);
86
87    // configurare intreruperi externe
88    //Aparitie intrerupere pe orice front al semnalului de intrare
89    EICRA = (0 << ISC11) | (0 << ISC10) | (0 << ISC01) | (1 << ISC00
        ↪ );
90    //Activare masca intrerupere externa
91    EIMSK = (0 << INT1) | (1 << INT0);
92    //Intializare biti marcare aparitie intrerupere
93    EIFR = (0 << INTF1) | (0 << INTF0);
94    //Permite activarea intreruperilor externe
95    PCICR = (0 << PCIE2) | (0 << PCIE1) | (0 << PCIE0);
96
97    //activare intreruperi globale
98    SREG |= 1 << SREG_I;
99  }
100
101 void setup()
102 {
103   //Configurare timer 1 pentru generare intrerupere la 100ms
104   configurare_timer();
105
106   //Configurare intrerupere externa pe int0 si int1
107   configurare_intrerupere();
108
109   initializarePID(&pidMotor, 2, 5, 1);
110   setareReferintaPID(&pidMotor, 1.5); // ex: 1.5m/s
111 }
```

```
112
113 void loop()
114 {
115 }
116
117 ISR(TIMER1_OVF_vect)
118 {
119   //dezactivare intreruperi globale
120     SREG &= ~(1 << SREG_I ) ;
121
122     //calculam distanta avand in vedere constantele legate de
        ↪ encoderul de pe motor
123     const float dist = encoder_ticks * wheelPerimeter /
        ↪ ticksPerRevolution;
124     //determinam viteza motorului
125     speed = dist * invTe; // V = d / T
126     //Determinam marimea de comanda, folosind regulatorul PID
127     const float pwm = calculareIesirePID(&pidMotor, speed);
128     //Setam semnalul de iesire pentru controlul motorului
129     analogWrite(3, pwm);
130     //reinitializam numarul de fante citite de la encoder
131     encoder_ticks = 0;
132
133   //activare intreruperi globale
134   SREG |= (1 << SREG_I );
135 }
136
137 ISR(INT0_vect)
138 {
139     // dezactivare intreruperi globale
140   SREG &= ~(1 << SREG_I);
141
142    encoder_ticks++;
143
144   // activare intreruperi globale
145     SREG |= 1 << SREG_I;
146 }
```

Aplicația propusă introduce o modalitate prin care se poate controla un motor electric de curent continuu, cu ajutorul PWM. Controlul se poate realiza avându-se în vedere encoderul rezistiv cu care este motorul echipat. Prin configurarea sistemului de întreruperi

este asigurată o frecvență constantă de apariție a întreruperii, astfel încât calculele pentru determinarea poziției curente să se realizeze cât mai bine, iar semnalul de comandă pentru mărimea de ieșire să fie cât mai corect.

7.2.1 Aplicaţie propusă

Pornind de la exemplul anterior, să se realizeze controlul unei platforme robotice mobile, înzestrată cu două roți de tracțiune la care sunt conectate două motoare electrice de curent continuu. Motoarele sunt prevăzute cu encodere de timp optic. Astfel, pentru determinarea erorii trebuie să se țină cont de numărul de fante citite de la motoare.

7.3 Controlul poziției unui motor DC utilizandu-se control la stare

Controlul în spațiul stărilor permite ajustarea mărimii de intrare într-un sistem dinamic, astfel încât să se mențină mărimea de ieșire la o valoare impusă. La fel ca și un regulator PID, un regulator în spațiul stărilor măsoară mărimile de ieșire ale sistemului și calculează în concordanță mărimile de intrare. Diferența majoră față de un sistem de reglare cu regulator de tip PID este dată de faptul că în cazul controlului la stare este luat în considerare și modelul sistemului care este controlat. Definirea unui model pentru un sistem dinamic poate fi în unele situații o problemă mai complicată, fapt care face ca utilizarea acestui tip de regulator să fie una mai greoaie decat un regulator clasic de tip PID.

În general un regulator cu control la stare este indicat să se utilizeze la controlul sistemelor cu o dinamică mai complexă, sau la sisteme care au mai multe intrări sau ieșiri.

Controlul la stare presupune definirea unui sistem general de ecuații diferențiale, exprimat într-o formă matriceală standard. O astfel de reprezentare va purta numele de reprezentare în spațiul stărilor și este caracterizată de următorele ecuații:

$$\dot{\mathbf{x}} = A \cdot \mathbf{x}(t) + B \cdot \mathbf{u}(t) \tag{7.1}$$

$$\mathbf{y}(t) = C \cdot \mathbf{x}(t) + D \cdot \mathbf{u}(t) \tag{7.2}$$

unde:

- $\mathbf{x} \in \Re^X$ reprezintă vectorul mărimilor de stare ale sistemului;
- $\mathbf{u} \in \Re^U$ reprezintă vectorul mărimilor de intrare;
- $\mathbf{y} \in \Re^Y$ reprezintă vectorul mărimilor de ieșire;
- $\mathbf{A} \in \Re^{X \times X}$ reprezintă matricea tranzițiilor stărilor, care descrie modul cum o să evolueze stările în situația în care nu sunt aplicate semnale de intrare de control;
- $\mathbf{B} \in \Re^{X \times U}$ reprezintă matricea de scalare a mărimilor de intrare, care descrie efectul mărimilor de control asupra stărilor;
- $\mathbf{C} \in \Re^{Y \times X}$ reprezintă matricea de scalare a mărimilor de ieșire sau matricea observațiilor, și descrie cum stările sunt mapate pe măsurătorile de pe senzori;
- $\mathbf{D} \in \Re^{Y \times U}$ reprezintă matricea de scalare a mărimilor de ieșire (matricea de tranzitie), și conține informații dacă oricare dintre intrări influențează imediat ieșirile (este de

obicei de valoare zero).

Pornind de la cele 4 matrici anterioare este posibilă descrierea oricărui sistem liniar, care nu este dependent de timp.

Schema bloc generală pentru un sistem de control în spațiul stărilor este prezentată în figura 7.6. Schema propusă permite sistemului să urmărească o mărime de referință impusă.

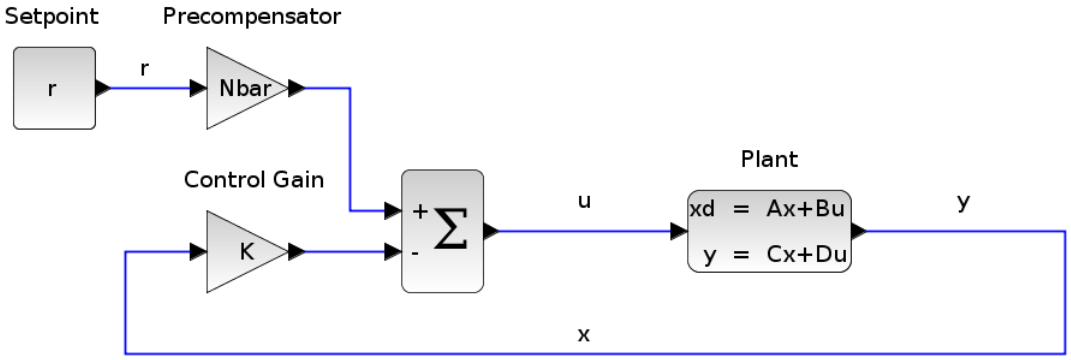


Fig. 7.6 Schema bloc pentru controlul la stare, cu urmărirea referinței.

Matricea **K** contine parametrii care trebuie acordați în funcție de sistemul considerat. În funcție de complexitatea sistemului, matricea **K** poate deveni destul de mare, iar acordarea ei va constitui un proces destul de dificil. Pentru a rezolva această problemă se poate utiliza un algoritm de tip LQR (L*inear–*Q*uadratic* R*egulator*).

În schema bloc pentru controlul la stare matricea $\bar{N} \in \Re^{U \times Y}$ influențează mărimea de control **u** prin introducerea unei compensații relative la semnalul de referință **r**. Cu ajutorul acestei compensații este posibilă atingerea valorii de referință pentru sistem. Matricea se determină în totalitate din matricele sistemului, fără a fi nevoie de acordarea vreunui parametru.

Pentru aplicația propusă ca și exemplu se va utiliza un model cu urmatorii parametrii:

- Momentul de inerție al rotorului: $J = 3,2284e^{-6}$ $[kg \cdot m^2]$;
- Constanta de frecare vâscoasă a motorului: $b = 3,50776e^{-6}$ $[N \cdot m \cdot s]$;
- Constanta electrică a motorului: $K_b = 0,0274$ $[\frac{V \cdot sec}{rad}]$;
- Constanta de cuplu a motorului: $K_t = 0,0274$ $[\frac{N \cdot m}{A}]$;
- Constanta de rezitență electrică: $R = 4$ $[\Omega]$;
- Constanta de inductanță electrică: $L = 2,75e^{-6}$ $[H]$;

Pornind de la legea a doua a lui Newton și legea lui Kirchhof se pot obține ecuațiile care descriu funcționarea motorului. Aceste ecuații sunt utilizate pentru a se obține modelul motorului exprimat în spațiul stărilor. Acest lucru se face prin alegerea ca mărimi de stare a poziției motorului, vitezei motorului și a curentului prin armătura. Tensiunea la capetele armăturii este considerată ca și mărime de intrare iar poziția rotorului este considerată ca

și mărime de ieșire. Ecuațiile în spațiul stărilor sunt:

$$\frac{d}{dt}\begin{bmatrix}\theta\\ \dot{\theta}\\ i\end{bmatrix} = \begin{bmatrix}0 & 1 & 0\\ 0 & -\frac{b}{J} & \frac{K}{J}\\ 0 & -\frac{K}{L} & -\frac{R}{L}\end{bmatrix}\cdot\begin{bmatrix}\theta\\ \dot{\theta}\\ i\end{bmatrix} + \begin{bmatrix}0\\ 0\\ \frac{1}{L}\end{bmatrix} V \tag{7.3}$$

$$y = \begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}\theta\\ \dot{\theta}\\ i\end{bmatrix} \tag{7.4}$$

unde, θ reprezintă poziția unghiulară, $\dot{\theta}$ este viteza unghiulară iar i este curentul prin motor. Pentru controlul în spațiul stărilor o sa utilizăm librăria StateSpaceControl. Aceasta pune la dispoziție o clasă template care are implementat un regulator în spațiul stărilor, utilizat la sisteme cu mai multe intrări și ieșiri. Aceasta poate fi descarcată gratuit de pe repo-ul: https://github.com/tomstewart89/StateSpaceControl.

Pentru a se putea crea regulatorul în spațiul stărilor, este nevoie la început de definirea unui model care să descrie dinamica sistemului ce urmează a fi reglat. Pentru aceasta este nevoie să se definească numărul de stări, de intrări și de ieșiri pe care sistemul le are. Toate acestea se realizează astfel:

```
1  Model<X,U,Y> model; //unde X este numarul de stari, U numarul
     ↪ de intrari iar Y este numarul de iesiri
```

Librăria de control propusă conține și modelul concret al unui motor electric de curent continuu. Modelul are la bază ecuațiile la stare amintite anterior.

Definirea celor patru matrici utilizate pentru descrierea sistemului ce urmează a fi controlat, se face folosind relații de forma:

```
1  model.A << 1, 2, 3,
2             4, 5, 6,
3             7, 8, 9;
```

Pornind de la definițiile anterioare se poate construi și modelul pentru regulator:

```
1  //se vor inlocui valorile pentru X, U si Y cu valorile
     ↪ corespunzatoare pentru sistemul considerat
2  StateSpaceController<X,U> controller(model);
```

Ajustarea parametrilor matricei de control K se face utilizându-se:

```
1  controller.k << 1, 2, 3, 4;
```

Avand declarațiile anterioare, pentru definirea sistemului și a parametrilor, mai este nevoie de inițializarea efectivă a regulatorului și de setarea valorii de referință. Acest lucru se poate realiza utilizandu-se următoarele linii de cod:

```
1  //Initializare regulator
```

```
2     controller.initialise();
3
4     //Setare marime de referinta
5     controller.r << 2.2;
```

Funcționarea propriu-zisă a regulatorului presupune ca la fiecare etapă să se facă o împrospătare a valorilor regulatorului, avându-se în vedere observația curentă y (datele curente citite de la senzori). Regulatorul cu parametrii actualizați va putea fi de asemenea accesat. Codul corespunzător este următorul:

```
1     controller.update(y, dt); //dt este un interval fix de timp
2
3     //accesarea marimii de intrare
4     controller.u;
```

Pentru controlul poziției unui motor de curent continuu se poate utiliza urmatorul cod. Acest cod are în vedere declarațiile anteriaoare și folosește librăria amintită. Codul rulează pe o placă de dezvoltare Arduino Uno.

Algoritmul 7.5 Controlul poziției unui motor de cc utilizându-se controlul la stare

```
1  #include <StateSpaceControl.h>
2
3  //Definim modelul la stare pentru un motor de cc
4  //parametrii sunt: J, b, k, R, L - conform modelului anterior
5  MotorPositionModel model(0.01, 0.1, 0.01, 1, 0.5);
6
7  //Definim modelul la stare al regulatorului. Modelul pentru
     ↪ pozitionarea motorului utilizeaza 3 stari si o intrare
8  StateSpaceController<3, 1> controller(model);
9
10 //Matricea de iesire
11 Matrix<3> y;
12 //constanta de timp
13 const float dt = 0.01;
14
15 void setup()
16 {
17   Serial.begin(115200);
18
19   //Acordarea matricei K. A fost realizata utilizand un tool
       ↪ disponibil in librarie: TuneThoseGains.ipynb
20   controller.K << 9.75, 0.97, 0.40;
21
```

```
22    //Initializarea regulatorului, astfel se poate determina N_bar
23    controller.initialise();
24
25    //Setarea referintei pentru pozitia la 2.2
26    controller.r << 2.2;
27  }
28
29  void loop()
30  {
31    //Y se actualizeaza folosind informatia de la senzor
32    //mecanismul utilizat pentru determinarea Y curent depinde de la
        ↪  aplicatie la aplicatie
33
34    //actualizarea regulatorului, avand in vedere vectorul
        ↪ actualizat al marimilor de iesire
35    controller.update(y, dt);
36
37    // Afisare rezultate
38    Serial << "unghi = " << y(0) << " viteza = " << y(1) << " curent
        ↪  = " << y(2) << '\n';
39
40    delay(dt * 1000);
41  }
```

În cazul aplicației este nevoie de citirea poziției, vitezei și a curentului prin circuit. Pentru primele două mărimi se poate utiliza un encoder optic. Cu ajutorul acestuia se pot contoriza într-o întrerupere numărul de fante care au trecut prin fața dispozitivului de citire și putem afla poziția. Pentru viteză trebuie să ținem cont de timpul necesar realizării unei rotații complete și de distanța parcursă la o rotație completă. Astfel putem determina viteza cu care se rotește axul motorului. Pentru măsurarea curentului este nevoie de utilizarea unui circuit adițional, de tip ACS712.

Circuitul ACS712 este utilizat pentru măsurarea cu exactitate a curentului alternativ sau continuu. Principiul de funcționare are la bază efectul Hall. Mărimea de ieșire a senzorului este una analogică, proporțională cu curentul din circuitul considerat.

Pinii senzorului sunt prezentați în figura 7.7 și au următoarea semnificație:

- Vcc - sursa de alimentare;
- Gnd - semnal de masă;
- Out - semnalul analogic de ieșire care este proporțional cu curentul din circuit;
- Current IN/OUT - pinii pentru conectarea în serie cu circuitul asupra căruia se dorește măsurarea curentului;

Interfațarea motorului cu placa de dezvoltare Arduino este prezentată în figura 7.8.

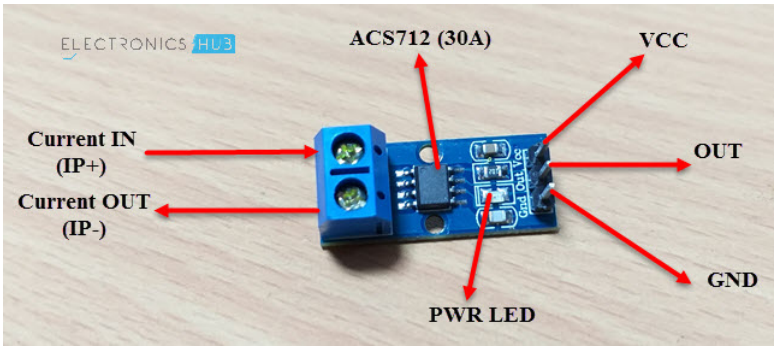


Fig. 7.7 Schema electrică de utilizare a senzorului ACS712.

Motorul este simbolizat de un modul "Load" în schema propusă. Rezultatul măsurătorii se poate afișa în terminal sau pe un LCD. Senzorul poate fi echipat cu 3 tipuri de senzitivitate, iar mărimea de ieșire este:

- pentru un curent de 5A cu o ieșire de 66mV/A;
- pentru un curent de 20A cu o ieșire de 100mV/A;
- pentru un curent de 30A cu o ieșire de 185mV/A.

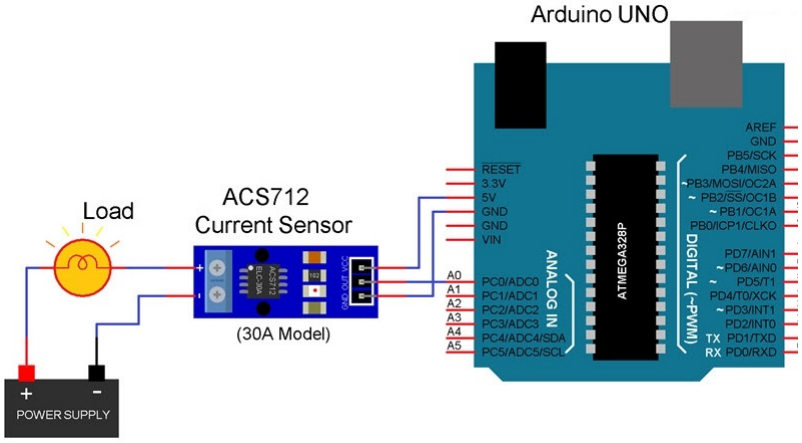


Fig. 7.8 Pinii senzorului ACS712.

Algoritmul 7.6 Determinarea curentului într-un circuit electric folosind senzorul ACS712.

```
1  const int currentPin = A0;
2  //senzitivitatea depinde de tipul senzorului. Pentru cazul de fata
     ↪ este un senzor de 5A
3  int sensitivity = 66;
4  int adcValue= 0;
5  int offsetVoltage = 2500;
6  double adcVoltage = 0;
7  double currentValue = 0;
8
9  void setup()
10 {
11   Serial.begin(9600);
```

```
12 }
13
14 void loop()
15 {
16   adcValue = analogRead(currentPin);
17   //determinare tensiune, pentru un convertor pe 10 biti, cu o
       ↪ tensiune de alimentare de 5000mV
18   adcVoltage = (adcValue / 1024.0) * 5000;
19   currentValue = ((adcVoltage - offsetVoltage) / sensitivity);
20
21   Serial.print("Raw Sensor Value = " );
22   Serial.print(adcValue);
23
24   Serial.print("\t Voltage(mV) = ");
25   Serial.print(adcVoltage,3);
26
27   Serial.print("\t Current = ");
28   Serial.println(currentValue,3);
29
30   delay(2500);
31 }
```

7.3.1 Aplicaţie propusă

Să se implementeze o aplicație care să permită controlul a două motoare de curent continuu montate pe o platforma mobilă. Ambele motoare trebuie controlate cu ajutorul unui regulator. Se va folosi controlul la stare pentru regulator. Pe platforma mobilă se va monta un senzor de culoare care este capabil sa recunoască o dungă albă. În funcție de această bandă, platforma trebuie să se centreze.

7.4 Modul de comunicație wireless

Această aplicație își propune să prezinte o modalitate prin care se pot transmite date între diferite module de tip Arduino. Pentru transferul datelor se vor folosi module de tip emitor-receptor din categoria nRF24L01.

Modulul nRF24L01 este un modul care poatre transmite sau recepționa date. Este un modul ieftin și ușor de integrat în aplicații cu placi de dezvoltare de tip Arduino Uno. Dintre principalele caracteristici ale acestor module putem aminti:

- puterea consumată este de aproximativ 12mA pe parcursul transmisiei (este mai mica și decât puterea consumată de un LED);
- poate funcționa cu rate de transfer de la 250Kbps pana la 2Mbps;
- poate emite pana la 100m depărtare în spațiu liber și este folosită o antenă;

– poate emite sau recepționa în același timp;
– fiecare modul poate comunica cu până la 6 alte dispozitive;
– folosește banda de 2.4GHz pentru transferul de date;
– are 125 de canale diferite de comunicație.

În figura 7.9 este prezentată configurația pinilor modulului. Modulul se conectează la

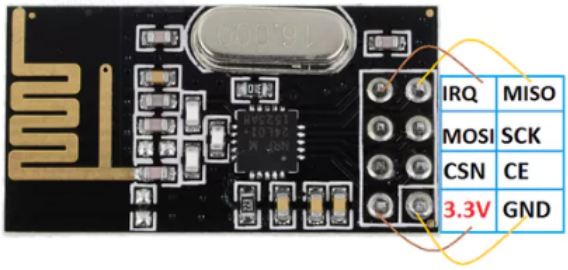


Fig. 7.9 Pinii senzorului nRF24L01.

placa Arduino cu ajutorul comunicației SPI. Configurația pinilor este următoarea:
– 3.3V este tensiunea de alimentare. Modulul poate fi conectat la tensiuni care pornesc de la 1.9V și ajung la 3.6V. Ceilalți pini pot fi conectați la tensiuni de 5V (conectați direct la placa Arduino);
– GND este semnalul de masă;
– MISO, MOSI și SCK sunt cei trei pini necesari realizării comunicației SPI. Ei se conecează la pinii corespunzători comunicației SPI de la placa de dezvoltare Arduino;
– CSN și CE sunt utilizați pentru setarea modulului pe starea activ/inactiv. De asemenea pentru setarea funcţionării ca emitor sau ca receptor;
– pinul IRQ nu trebuie conectat.

Principul de funcționare al aplicației presupune interfațarea a două module Arduino Uno, folosind modulul de comunicație descris anterior. Astfel, se vor trimite date de la un Arduino către al doilea modul Arduino. Atunci când se va apăsa un buton conectat la prima plăcuță, ledul de pe cea de a doua plăcuță se va aprinde. Schema electrică este prezentată în figura 7.10.

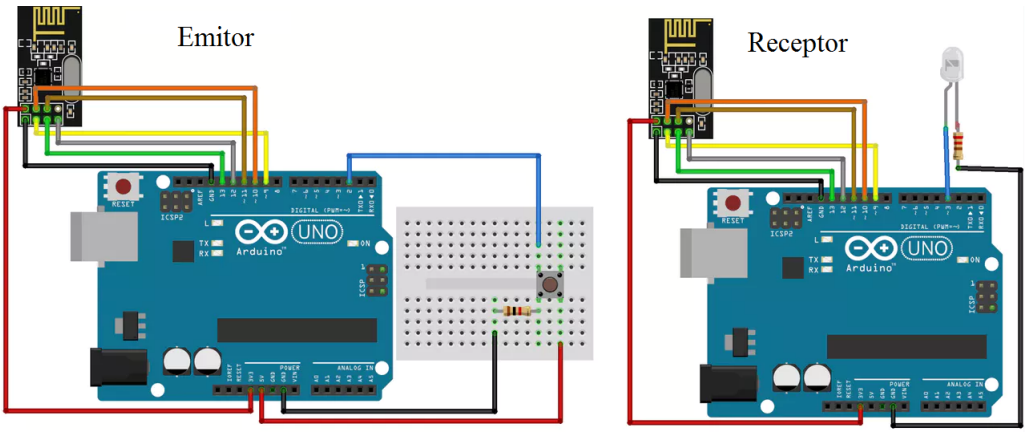


Fig. 7.10 Schemă electrică pentru comunicația wifi între două module Arduino Uno.

Pentru realizarea comunicației propuse se vor utiliza două librării pentru comunicația

wifi și pentru comunicația SPI. Codul corespunzător modulului emitor este următorul:

Algoritmul 7.7 Codul pentru modulul emitor.

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

//initializare obiect de tip nRF, pe pinii 9 si 10 pentru semnalele CE si CSN
RF24 radio(9, 10); // CE, CSN

//Asignarea unei adrese unice pentru modulul catre care se trimit datele. Aceasta adresa trebuie sa fie identica si pe modulul de receptie
const byte address[6] = "00001";

bool stare_buton = 0;

void setup()
{
  //Configurare modul comunicatie
  //Pornirea comunicatiei wifi
  radio.begin();
  //Setarea adresei unde o sa fie trimise datele
  radio.openWritingPipe(address);
  //Valoarea poate fi setata la minim, aceasta depinzand de distanta dintre emitor si receptor
  radio.setPALevel(RF24_PA_MIN);
  //Configurare modul ca si emitor
  radio.stopListening();

  // configurare intreruperi externe pe int0
    EICRA = (0 << ISC11) | (0 << ISC10) | (1 << ISC01) | (1 << ISC00);
    EIMSK = (0 << INT1) | (1 << INT0);
    EIFR = (0 << INTF1) | (0 << INTF0);
    PCICR = (0 << PCIE2) | (0 << PCIE1) | (0 << PCIE0);
}

void loop()
{
```

```
34    if(stare_buton)
35    {
36      const char text[] = "Starea butonului este HIGH";
37      //Transmiterea mesajului la receptor
38      radio.write(&text, sizeof(text));
39    }
40    else
41    {
42      const char text[] = "Starea butonului este LOW";
43      radio.write(&text, sizeof(text));
44    }
45
46    //Transmiterea mesajului la receptor
47    radio.write(&stare_buton, sizeof(stare_buton));
48
49    delay(1000);
50  }
51
52  ISR(INT0_vect)
53  {
54      // dezactivare intreruperi globale
55    SREG &= ~(1 << SREG_I);
56
57    //validare buton apasat
58    if (!stare_buton)
59      stare_buton = true;
60    else
61      stare_buton = false;
62
63    // activare intreruperi globale
64      SREG |= 1 << SREG_I;
65  }
```

Algoritmul 7.8 Codul pentru modulul receptor.

```
1  #include <SPI.h>
2  #include <nRF24L01.h>
3  #include <RF24.h>
4
5  //initializare obiect de tip nRF, pe pinii 9 si 10 pentru
     ↪ semnalele CE si CSN
6  RF24 radio(9, 10); // CE, CSN
```

```
7
8  //Asignarea unei adrese unice pentru modulul catre care se trimit
       ↪ datele. Aceasta adresa trebuie sa fie identica si pe modulul
       ↪  de receptie
9  const byte address[6] = "00001";
10 bool stare_buton = 0;
11
12 void setup()
13 {
14     Serial.begin(9600);
15     radio.begin();
16     //Setarea adresei unde o sa fie trimise datele
17     radio.openReadingPipe(0, address);
18     //Valoarea poate fi setata la minim, aceasta depinzand de
           ↪ distanta dintre emitor si receptor
19     radio.setPALevel(RF24_PA_MIN);
20     //Configurare modul ca si receptor
21     radio.startListening();
22
23     //Configurare port de iesire
24     DDRD |= (1<<PD3);
25 }
26 void loop()
27 {
28      //verifica daca sunt date disponibile
29     if (radio.available())
30     {
31         //salvam datele primite
32         char text[32] = "";
33         //citim datele
34         radio.read(&text, sizeof(text));
35         radio.read(&button_state, sizeof(button_state));
36
37         if(stare_buton == HIGH)
38         {
39             PORTD = 1 << PORTD3;
40              Serial.println(text);
41         }
42         else
43         {
44             PORTD = 0 << PORTD3;
```

```
45                 Serial.println(text);}
46             }
47         delay(5);
48 }
```

Aplicația anterioară poate fi extinsă la transferul bidirecțional între ambele module. Astfel, fiecare dintre celele două module pot avea rolul de emitor sau receptor, în funcție de necesitate. Schema electrică a unui astfel de sistem este în figura 7.11.

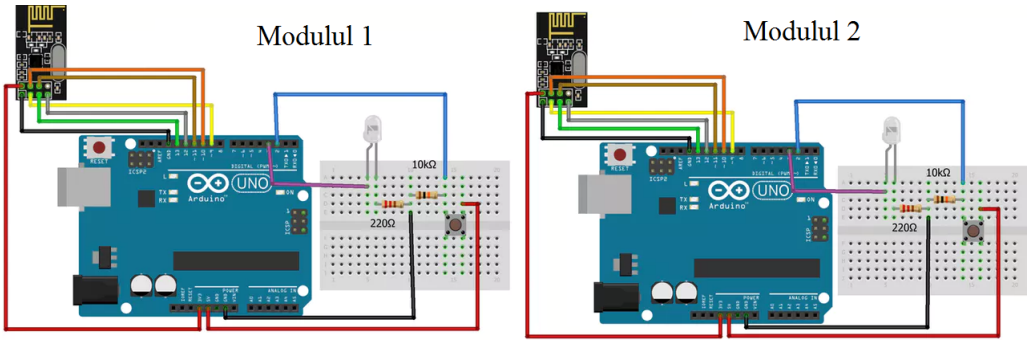


Fig. 7.11 Schemă electrică pentru comunicația wifi între două module Arduino Uno, cu transfer reciproc de date.

Funcțional, sistemul o sa transmită o comandă de aprindere led, de la modulul 1 către modulul 2, apoi al doilea modul o sa trimită o comandă similară către primul modul. Codul corespunzător modulului 1 este următorul:

Algoritmul 7.9 Codul pentru modulul 1.

```
1  #include <SPI.h>
2  #include <nRF24L01.h>
3  #include <RF24.h>
4
5  //initializare obiect de tip nRF, pe pinii 9 si 10 pentru
       ↪ semnalele CE si CSN
6  RF24 radio(9, 10);
7
8  //Setarea adreselor pentru functia de emitor si receptor
9  const byte addresses [][6] = {"00001", "00002"};
10
11 boolean stare_buton_1 = 0;
12 boolean stare_buton_2 = 0;
13
14 void setup() {
15     //Configurare port pentru LED si buton
16     DDRD |= (1<<PD3);
17
18     //configurare intrerupere pe int1
```

```
19        // configurare intreruperi externe pe int0
20      EICRA = (0 << ISC11) | (0 << ISC10) | (1 << ISC01) | (1 <<
            ↪ ISC00);
21      EIMSK = (0 << INT1) | (1 << INT0);
22      EIFR = (0 << INTF1) | (0 << INTF0);
23      PCICR = (0 << PCIE2) | (0 << PCIE1) | (0 << PCIE0);
24
25    radio.begin();
26    //Setarea adresei la care se vor trimite date
27    radio.openWritingPipe(addresses[1]);
28    //setarea adresei de la care se vor trimite date
29    radio.openReadingPipe(1, addresses[0]);
30
31    radio.setPALevel(RF24_PA_MIN);
32  }
33  void loop()
34  {
35    delay(5);
36    //setare modul ca emitor
37    radio.stopListening();
38    //trimitem date
39    radio.write(&stare_buton_1, sizeof(stare_buton_1));
40    delay(5);
41
42    //setare modul ca receptie
43    radio.startListening();
44    //verifica daca sunt date disponibile
45    while(!radio.available());
46    //daca sunt date disponibile, le citeste
47    radio.read(&button_state1, sizeof(button_state1));
48
49    if (button_state2 == HIGH)
50    {
51      PORTD = 1 << PORTD3;
52    }
53    else
54    {
55      PORTD = 0 << PORTD3;
56    }
57  }
58
```

```
59 ISR(INT0_vect)
60 {
61     // dezactivare intreruperi globale
62   SREG &= ~(1 << SREG_I);
63
64   //validare buton apasat
65   if (!stare_buton_1)
66     stare_buton_1 = true;
67   else
68     stare_buton_1 = false;
69
70   // activare intreruperi globale
71     SREG |= 1 << SREG_I;
72 }
```

Codul corespunzător modulului 2 este următorul:

Algoritmul 7.10 Codul pentru modulul 2.

```
1  #include <SPI.h>
2  #include <nRF24L01.h>
3  #include <RF24.h>
4
5  RF24 radio(9, 10);
6
7  //Setarea adreselor pentru functia de emitor si receptor
8  const byte addresses [][6] = {"00001", "00002"};
9
10 boolean stare_buton_1 = 0;
11 boolean stare_buton_2 = 0;
12
13 void setup() {
14  //Configurare port pentru LED si buton
15     DDRD |= (1<<PD3);
16
17     //configurare intrerupere pe int1
18       // configurare intreruperi externe pe int0
19     EICRA = (0 << ISC11) | (0 << ISC10) | (1 << ISC01) | (1 <<
         ↪ ISC00);
20     EIMSK = (0 << INT1) | (1 << INT0);
21     EIFR = (0 << INTF1) | (0 << INTF0);
22     PCICR = (0 << PCIE2) | (0 << PCIE1) | (0 << PCIE0);
23
```

```
24    radio.begin();
25     //Setarea adresei la care se vor trimite date
26    radio.openWritingPipe(addresses[1]);
27    //setarea adresei de la care se vor trimite date
28    radio.openReadingPipe(1, addresses[0]);
29    radio.setPALevel(RF24_PA_MIN);
30  }
31  void loop()
32  {
33    delay(5);
34    //setare modul ca emitor
35    radio.stopListening();
36    //trimitere de date
37    radio.write(&button_state_1, sizeof(button_state_1));
38    delay(5);
39
40    //setare modul ca receptor
41    radio.startListening();
42    //verifica daca sunt date de intrare disponibile
43    while(!radio.available());
44    //citire date
45    radio.read(&button_state1, sizeof(button_state1));
46
47    if (button_state2 == HIGH)
48    {
49      PORTD = 1 << PORTD3;
50    }
51    else
52    {
53      PORTD = 0 << PORTD3;
54    }
55  }
```

7.4.1 Aplicaţie propusă

Să se implementeze o aplicație care controlează la distanță un motor electric de curent continuu. Transferul de informații între cele două module se realizează folosind modulul nRF24L01. Controlul se implementează avandu-se în vedere aplicația cu recgulator PID, prezentată anterior. Afișarea informațiilor primite de la modulele de comunicație se pot afișa în terminal sau se poate atașa un afișaj cu LCD.

7.5 Modul de măsurare a distanței

Pentru a se putea realiza măsurarea de distante, se poate utiliza un senzor cu ultrasunete de tipul HC-SR04. Acesta poate returna distanțe de la 2cm până la 3-4m. Această familie de senzori este una care este cu prețuri reduse și se potrivește foarte bine în aplicații care necesită consum redus de energie.

Senzorul cu ultrasunete HC-SR04 conține două module în interior. Primul modul este folosit pentru transmiterea de date, prin convertirea unui semnal electric într-o sucesiune de pulsuri la o frecvență de 40KHz. Al doilea modul este destinat recepției pulsurilor trimise de primul modul. Dacă acesta le recepționează, o să emită un puls la ieșirea sa, a cărui lățime este dependentă de distanța pe care pulsul emitent a avut-o. Dintre cele mai importante specificații ale sale amintim:

- tensiune de alimentare de 5V;
- curentul de operare de 15mA;
- frecvența de operare de 40KHz;
- distanța minimă și maximă: 2cm - 4m;
- unghi de măsurare de 15 grade.

În figura 7.12 este prezentată configurația pinilor modulului. Configurația pinilor senzorului este următoarea:

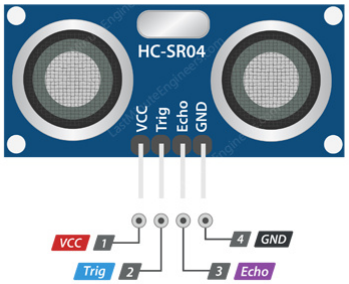


Fig. 7.12 Pinii senzorului HC-SR04.

- Vcc este tensiunea de alimentare (5V);
- GND este semnalul de masă;
- Trig este utilizat pentru declanșarea pulsurilor ultrasonice;
- Echo va produce pe pin un puls cu lățime variabilă, atunci când semnalul reflectat este recepționat.

Pornirea senzorului este dată de aplicarea unui puls de $10\mu s$ pe pinul de Trigger așa cum se poate vedea în figura 7.13. Ca și răspuns la acest impuls aplicat, senzorul transmite o succesiune de pulsuri la o frecvență de 40KHz. Această combinație de 8 pulsuri fac dispozitivul emitent unic, astfel încât să se poată diferenția de alte dispozitive cu ultrasunete. Dacă cele 8 pulsuri sunt reflextate din scenă, receptorul le va înregistra. Pinul de Echo devine activ și începe să formeze semnalul de răspuns, care poate fi interpretat de modulul Arduino. Dacă semnalul emis nu se reflectă, pinul de echo o sa genereze un puls de 38ms.

Aplicația exemplu presupune construirea unui modul care să afișeze distanța față de un obiect aflat în fața senzorului. Afișarea se realizează pe un afișaj LCD 16x2. Schema electrică

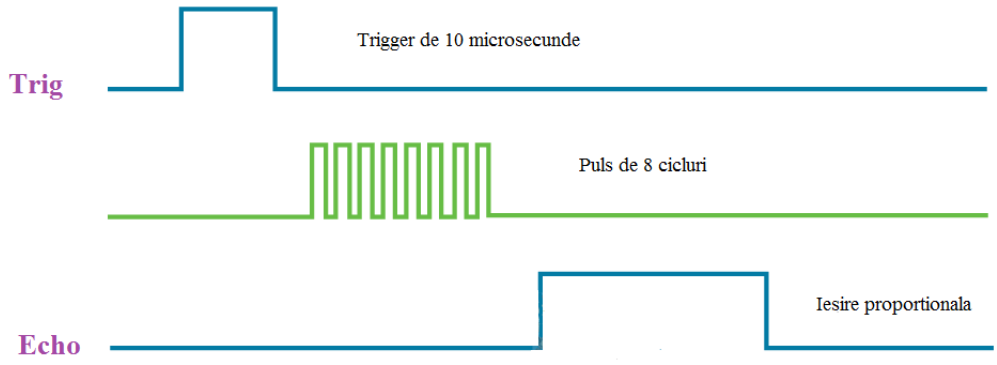


Fig. 7.13 Principiu de funcționare al senzorului cu ultrasunete.

a aplicației propuse este în figura 7.14.

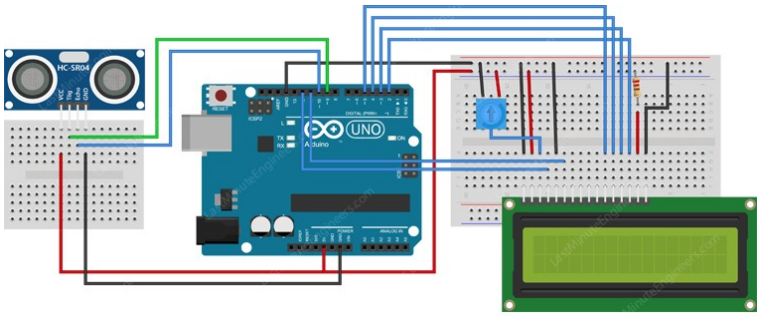


Fig. 7.14 Schema electrică pentru modulul de măsurare a distanței.

Pentru a putea utiliza modulul de afișaj o sa utilizam pachetele LCDBarGraph și LiquidCrystal. Codul corespunzător aplicației propuse este următorul.

Algoritmul 7.11 Codul pentru modulul de măsurare a distanței.

```
1  //include libraria LiquidCrystal
2  #include <LiquidCrystal.h>
3
4  //include libraria LcdBarGraph
5  #include <LcdBarGraph.h>
6
7  //Definim distanta maxima pe care dorim sa o atingem
8  #define distanta_maxima 200
9
10 // Creem un obiect de tip LCD conectat la pinii/semnalele: (rs,
     ↪ enable, d4, d5, d6, d7)
11 LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
12
13 //creem un obiect LCD Bargraph
14 LcdBarGraph lbg(&lcd, 16, 0, 1);
15
```

```
16 const int trigPin = 9;
17 const int echoPin = 10;
18 long durata;
19 int distanta;
20
21 void setup()
22 {
23     //initializam interfata afisajului LCD 16x2
24     lcd.begin(16,2);
25     //Configuram pinii de intrare sau iesire
26     pinMode(trigPin, OUTPUT);
27     pinMode(echoPin, INPUT);
28 }
29
30 void loop()
31 {
32     // Scriem un puls pe pinul Trigger al modulului HC-SR04
33     digitalWrite(trigPin, LOW);
34     delayMicroseconds(2);
35     digitalWrite(trigPin, HIGH);
36     delayMicroseconds(10);
37     digitalWrite(trigPin, LOW);
38
39     //Masuram raspunsul de pe pinul Echo
40     duration = pulseIn(echoPin, HIGH);
41
42     // Determinam distanta din durata semnalului
43     // Utilizam 343 m/s pentru viteza sunetului
44     distanta = distanta*0.034/2;
45
46     // Afisam rezultatelel pe LCD
47     lcd.setCursor(0,0);
48     lcd.print("Distanta: ");
49     lcd.print(distanta);
50     lcd.print(" cm");
51
52     // Desenam un bargraph pe a doua linie a LCD-ului
53     lcd.setCursor(0,1);
54     lbg.drawValue(distanta, distanta_maxima);
55     delay(500);
56 }
```

7.5.1 Aplicaţie propusă

Să se implementeze un modul de măsurare bidirecțional la distanță. Pentru aceasta se vor utiliza două module de senzori cu ultrasunete HC-SR04 care vor transmite informația către o placă de dezvoltare Arduino Uno. Informația va fi transferată pe baza unui modul wifi nRF24L01 către un al doilea modul Arduino. Acesta va afisa informația pe un dispozitiv LCD dar și în consola de terminal.

Bibliografie

[1] *The Successive Approximation Analog to Digital Converter*. London: Springer London, 2008, pp. 89–91. [Online]. Available: https://doi.org/10.1007/978-1-84800-119-0_13

[2] Atmel. [Online]. Available: https://en.wikipedia.org/wiki/Atmel

[3] Autodesk-Tinkercad. [Online]. Available: https://www.tinkercad.com/

[4] B. Craft, *Arduino Projects For Dummies*, 1st Ed. England: For Dummies, 2013.

[5] D. Floroian and G. Macesanu, *Programarea si utilizarea sistemelor cu microprocesoare*. Brasov: Ed. Universitatii Transilvania, 2014.

[6] FTDI. [Online]. Available: https://en.wikipedia.org/wiki/FTDI

[7] Microchip. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf

[8] PointTutorials. [Online]. Available: https://www.tutorialspoint.com/cprogramming/c_constants.html

[9] D. Ungureanu, *Programare obiectuală folosind C++*. Brașov: Ed. Universității Transilvania, 2009.