UNIVERSITATEA BABEȘ-BOLYAI
BABEȘ-BOLYAI TUDOMÁNYEGYETEM
BABEȘ-BOLYAI UNIVERSITÄT
BABEȘ-BOLYAI UNIVERSITY
TRADITIO ET EXCELLENTIA

FACULTATEA DE FIZICĂ
Str. Mihail Kogălniceanu nr.1
Cluj-Napoca, RO-400084
Tel: +4(0)264-405300 | FAX: +4(0)264-591906
secretariat.phys@ubbcluj.ro
www.phys.ubbcluj.ro

UNIVERSITATEA "BABEȘ-BOLYAI" CLUJ-NAPOCA

FACULTATEA DE FIZICĂ

SPECIALIZAREA FIZICĂ INFORMATICĂ

LUCRARE DE LICENȚĂ

Coordonator științific:

Lect. Dr. Ing. Sever Mican

Absolvent:

Vlad-Ionuț Groșan

2022

UNIVERSITATEA BABEȘ-BOLYAI
BABEȘ-BOLYAI TUDOMÁNYEGYETEM
BABEȘ-BOLYAI UNIVERSITÄT
BABEȘ-BOLYAI UNIVERSITY
TRADITIO ET EXCELLENTIA

FACULTATEA DE FIZICĂ
Str. Mihail Kogălniceanu nr.1
Cluj-Napoca, RO-400084
Tel: +4(0)264-405300 | FAX: +4(0)264-591906
secretariat.phys@ubbcluj.ro
www.phys.ubbcluj.ro

UNIVERSITATEA "BABEȘ-BOLYAI" CLUJ-NAPOCA

FACULTATEA DE FIZICĂ

SPECIALIZAREA FIZICĂ INFORMATICĂ

LUCRARE DE LICENȚĂ

PROGRAM PENTRU CONTROLUL UNEI SURE DE ALIMENTARE DE LA DISTANȚĂ

Coordonator științific:

Lect. Dr. Ing. Sever Mican

Absolvent:

Vlad-Ionuț Groșan

2022

Abstract

The purpose of this work is to write a program for a power supply so that it can be controlled remotely using a computer or laptop through a program that can be accessed with a graphical interface. The thesis begins by briefly explaining some of the basics of electronics, beginning with a description of one of the most basic laws of electricity, Ohm's Law. The following laws that are being discussed are Kirchhoff's Laws which are briefly explained. After that, Norton and Thevenin's theorems are explained and proven. Then we discuss the current and voltage sources, ideal and real, and the basics of RS-232 serial communications, as well as COM and USB ports. The next part is based on the basics of programming in Octave, which is the program that creates the GUI interface that controls the voltage source, and the last part explains the code that underlies the operation of this program. The explanation of the code begins with the sequence that connects to the serial port with which the source is connected to the computer, then exemplifies the method by which the GUI panel was created and each button and text box that was used, and how they transmit data to the power supply, so that we can set different values for the output current or voltage, and also how it recieves data from the power supply in order to read the output voltage/current values in real time.

Cuprins

Introducere

Scoupul acestei lucrări este de a programa o sursă de tensiune astfel încât ea să poată fi controlată de la distanță, cu un calculator sau laptop, prin intermediul unui program ce poate fi accesat cu o interfață grafică. Lucrarea este împărțită în trei capitole.

Primul capitol al acestei lucrări începe prin a explica pe scurt anumite noțiuni fundamentale de electronică, începând cu descrierea al uneia dintre cele mai de bază legi din electricitate, Legea lui Ohm. Aici se explică pe scurt definiția și cum se folosește în anumite situații diferite. Urmatoarele legi despre care se discuta sunt Legile lui Kirchhoff. Sunt explicate pe scurt și se dă un exemplu cu un nod de rețea pentru utilizarea primei legi a lui Kirchhoff, și un alt exemplu cu un circuit închis în care se folosește cea dea doua lege a lui Kirchhoff. După aceea sunt explicate și demonstrate teoremele lui Norton si Thévenin, Apoi se discută despre sursele de curent, si de tensiune, ideale sau reale, iar în partea de final al capitolului se vorbește despre fundamentele comunicațiilor seriale RS-232, precum și despre porturile COM și USB, și modul de funcționare al acestora

Capitolul doi se bazează pe noțiunile fundamentale de programare în Octave, care este programul în care se creează intarfața GUI cu care se controloază sursa de alimentare. După o scurtă introducere a programului, se discută despre tipurile de date în Octave, Tipuri de Variabile, instrucțiuni precum *while, for, if*, și de asemenea, despre elementele din Interfața grafică în Octave, cum ar fi butoanele, casetele de text, si așa mai departe.

În ultimul capitol este explicat codul care stă la baza funcționării al acestui program. Se începe prin secvența de cod care face legătura cu portul serial cu care e cuplată sursa la calculator, apoi se exemplifică metoda prin care a fost creat panoul GUI și respectiv fiecare buton și casetă de text care au fost folosite, inclusiv modul de funcționare al acestora.

1. Noțiuni fundamentale de electronică

1.1. Legea lui Ohm

Legea lui Ohm spune că curentul care trece printr-un conductor între două puncte este direct proporțional cu diferența de potențial din cele două puncte. Dacă introducem constanta de proporționalitate, rezistența [1], ajungem la urmatoarea ecuație [2]:

$$I = \frac{V}{R} \tag{1.1}$$

unde *I* este curentul măsurat în amperi, *V* este tensiunea pe conductor măsurată în unități de volți și *R* este rezistența conductorului măsurat în ohmi. Mai exact, legea lui Ohm descrisă de (1.1) afirmă că valoarea lui R în această relație este constantă, independent de curent [3]. Dacă rezistența nu are valoare constantă, ecuația anterioară nu poate fi cunoscută ca legea lui Ohm, dar poate fi totuși utilizată ca definiție a rezistenței statice [4]. Legea lui Ohm este o relație empirică care descrie cu exactitate conductivitatea electrică a majorității materialelor conductoare. Această lege poartă numele fizicianului german Georg Ohm, care a publicat un tratat care descrie măsurătorile tensiunii și curentului aplicate de circuite electrice simple care conține diferite lungimi de fir, în 1827 [5]. Ohm a explicat rezultatele experimentelor sale cu o ecuație ceva mai complexă decât forma de față în relația (1.1).

Trei expresii echivalente ale legii lui Ohm sunt folosite în mod interschimbabil, atunci când vorbim de analiza circuitelor: [5].

$$I = \frac{V}{R} \quad (1.2) \quad \text{sau} \quad V = IR \quad (1.3) \quad \text{sau} \quad R = \frac{V}{I} \quad (1.4)$$

echivalența ecuațiilor (1.2) – (1.4) poate fi reprezentată prin ecuația (1.5):

$$\frac{V}{IR} = 1 \tag{1.5}$$

Rezistoarele din circuite rezistive sunt elemente de circuit care împiedică trecerea sarcinii electrice, în acord cu legea lui Ohm, și sunt proiectate să aibă o anumită valoare a rezistenței *R*. Un rezistor poate fi reprezentat ca un dreptunghi lung sau simbol în zig-zag așa cum putem vedea în figura 1.1. De regulă dreptunghiul e folosit în Europa, iar zig-zag în SUA. Ambele simboluri sunt universal acceptate.

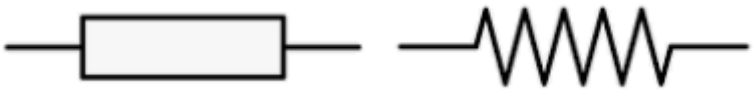
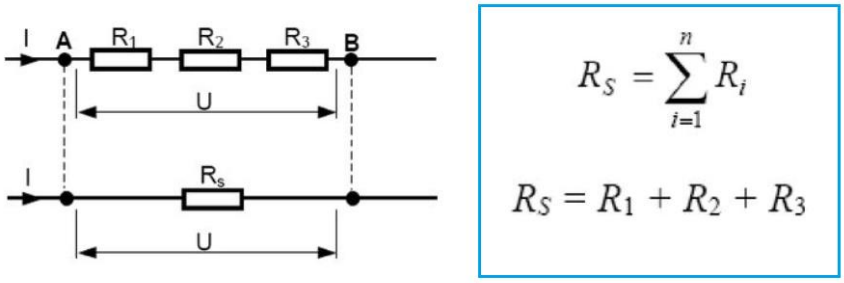


Figura 1.1. Simbolurile folosite pentru rezistor în scheme electrice.

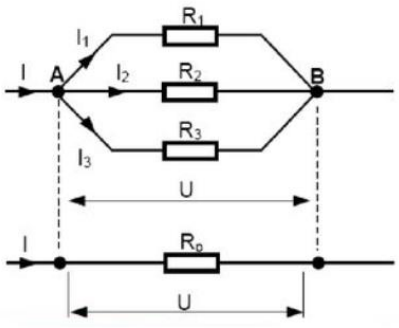
Un element (rezistor sau conductor) care funcționează conform legii lui Ohm într-un anumit interval de funcționare se numește dispozitiv ohmic (sau rezistor ohmic), deoarece legea lui Ohm și o singură valoare a rezistenței sunt suficiente pentru a descrie funcționarea dispozitivului pe elementul respectiv [5]. Legea lui Ohm se aplică circuitelor care conțin doar elemente rezistive (fără condensatoare sau inductori) la orice tensiune sau curent, indiferent dacă tensiunea sau amperajul sunt constante (DC) sau variabile. În oricare din aceste cazuri, legea lui Ohm este se poate aplica pentru aceste circuite. Rezistoarele care sunt în serie sau paralel pot fi grupate într-o singură „rezistență echivalentă" pentru a aplica Legea lui Ohm în analiza circuitelor, așa cum se poate vedea în figurile 1.2 și 1.3:



$$R_S = \sum_{i=1}^{n} R_i$$

$$R_S = R_1 + R_2 + R_3$$

Figura 1.2. Gruparea rezistențelor în serie [6].



$$\frac{1}{R_P} = \sum_{i=1}^{n} \frac{1}{R_i}$$

$$\frac{1}{R_P} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}$$

Figura 1.3. Gruparea rezistențelor în paralel [6].

Atunci când elemente reactive, cum ar fi condensatoare, bobine sau linii de transmisie a energiei, sunt implicate într-un circuit cu tensiuni sau curenți alternativi sau care variază în timp, relația (1.1) dintre tensiune și curent nu se aplică direct, deoarece această formă conține doar rezistențe de valoare R, nu impedanțe complexe care pot conține capacitate (C) sau inductanță

(L) [5]. Ecuațiile circuitului a.c. invariante în timp au aceeași formă ca legea lui Ohm. Cu toate acestea, variabilele sunt generalizate la numere complexe, și formele de undă de tensiune și curent sunt complexe exponențial [7]. Generalizarea complexă a rezistenței este impedanța, de obicei notată cu Z. Se pot demonstra relațiile 1.6 – 1.9 [8]:

$$X_L = 2\pi fL \tag{1.6}$$

$$X_C = \frac{1}{2\pi fC} \tag{1.7}$$

unde *f* este frecvența, X_L este reactanța inductorului și X_C reactanța condensatorului. În figura 1.4 se poate vedea deducerea formulei (1.8) pentru impedanța electrică *Z* [8]:

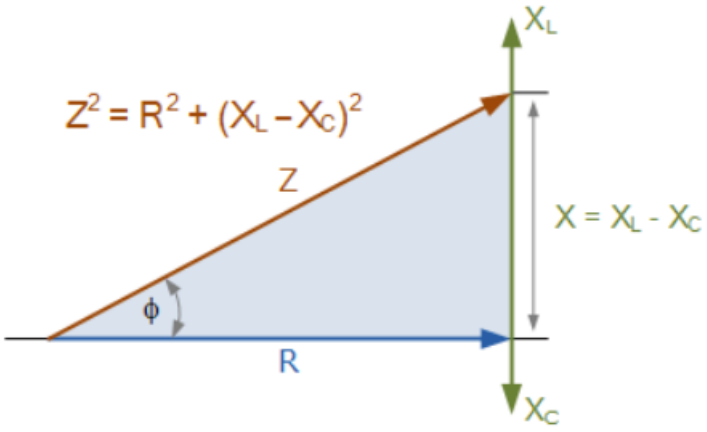


Figura 1.4. Calcularea Impedanței [8].

$$Z^2 = R^2 + (X_L - X_C)^2 \tag{1.8}$$

Ecuatia lui Ohm devine echivalentă cu ecuația (1.9):

$$V = IZ \tag{1.9}$$

Unde *V* și *I* sunt scalari complecși de tensiune și respectiv curent, și *Z* este impedanța complexă. Această formă a legii lui Ohm, unde Z reprezintă R, se generalizează în forma sa cea mai simplă. Când Z este complex, doar partea reală este responsabilă pentru disiparea căldurii. Într-un circuit general AC, *Z* variază puternic cu frecvența *f*, la fel și relația dintre curent și tensiune.

1.2. Legile lui Kirchhoff

Legile de circuit ale lui Kirchhoff sunt două legi care se ocupă de diferența de curent și potențial (denumită adesea tensiune) în modelul concentrat al unui circuit electric. Fizicianul german Gustav Kirchoff a descris aceste legi pentru prima oară în anul 1845. Aceasta este anterioara lucrarii lui James Clerk Maxwell și a generalizat munca lui Georg Ohm [10]. Folosite foarte mult în domeniul ingineriei electrice, sunt cunoscute și ca regulile lui Kirchhoff sau pur și simplu legile lui Kirchhoff [10]. Cele două legi Kirchhoff pot fi înțelese ca o echivalență pentru ecuațiile lui Maxwell în domeniul de frecvență joasă. Sunt corecte pentru circuitele de curent continuu și pentru circuitele de curent alternativ la frecvențe unde lungimea de undă a radiației electromagnetice este foarte mare în comparație cu dimensiunea circuitului.

Prima lege a lui Kirchhoff afirmă că pentru orice nod dintr-un circuit, curentul total care intră în acel nod este egal cu suma curenților care ies din rețeaua de conductori respectivă [10]. Altfel spus, prima lege a lui Kirchhoff spune că suma tuturor curenților care se întâlnesc într-un punct dintr-o rețea de conductori este zero. Reamintind că un curent este o mărime cu semn (pozitivă sau negativă) care reflectă direcția către sau dinspre un nod, acest principiu poate fi enunțat ca în formula (1.10):

$$\sum_{k=1}^{n} I_k = 0 \tag{1.10}$$

unde n este numărul de ramuri de curenți care intră sau ies din nod. Un exemplu concret este prezentat în figura 1.5, unde este prezentat un nod de rețea.

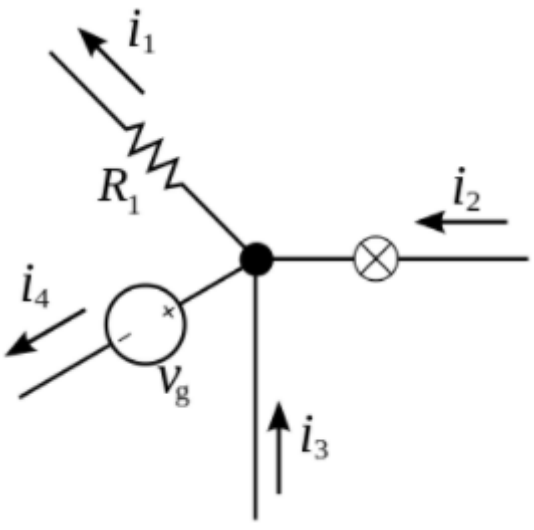


Figura 1.5. un nod de rețea în care avem patru curenți [11].

curentul care intră în orice nod de rețea este egal cu curentul care iese din acel nod, ceea ce inseamnă că: $i_2 + i_3 = i_1 + i_4$. Legea întâi a lui Kirchhoff se bazează pe conservarea sarcinii (în coulombi), care este produsul dintre timp (în secunde) și curent (în amperi) [9]. Dacă sarcina netă într-o regiune este constantă, Prima lege a lui Kirchhoff se va menține la limitele regiunii,înseamnând că se bazează pe faptul că sarcina netă în firele de curent și în componente este constantă [10]. O versiune matriceală a primei legi a lui Kirchhoff stă la baza majorității software-urilor de simulare a circuitelor, cum ar fi SPICE [11] . Prima lege a lui Krichhoff este folosită împreună cu legea lui Ohm pentru a efectua analiza nodurilor și se aplică oricărei rețea de circuit grupată, indiferent de natura acesteia, fie liniară sau neliniară, unilaterală sau bilaterală, pozitivă sau pasivă [9].

A doua lege a lui Kirchhoff spune că suma tuturor a tensiunilor în jurul oricărei bucle închise dintr-un circuit electric este nulă [11]. Similar cu prima lege a lui Kirchhoff, a doua lui lege poate fi formulată în felul următor - formula (1.11):

$$\sum_{k=1}^{n} V_k = 0 \tag{1.11}$$

unde n este numărul total de tensiuni măsurate. Ca un exemplu considerăm bucla de rețea din figura 1.6, unde avem o sursă de tensiune v_4 și trei rezistențe, R_1, R_2 și R_3, cu respectivele tensiuni asupra fiecăreia v_1, v_2 și v_3.

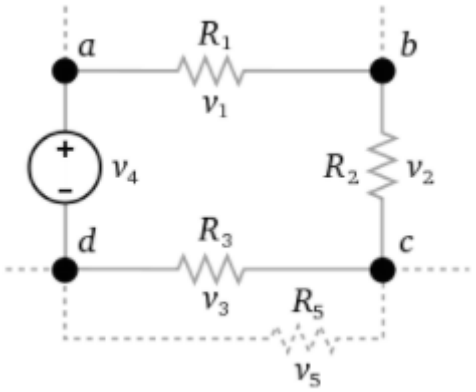


Figura 1.6. Buclă de rețea cu o sursă de tensiune și 3 rezistențe [11].

Suma tuturor diferențelor de potențial în jurul unei bucle este egală nulă, ceea ce înseamnă că $v_1 + v_2 + v_3 + v_4 = 0$.

În domeniul frecvențelor joase, diferența de potențial în jurul oricărei bucle este nulă[11]. Aceasta include buclele care sunt dispuse aleatoriu în spațiu, fără a se limita exact la buclele limitate de conductori și de elementele circuitului. În domeniul frecvențelor joase, acesta

reprezintă o echivalență a legii inducției lui Faraday (care reprezintă una dintre relațiile lui Maxwell) [10]. Aceasta se poate aplica practic în situațiile în care este implicată electricitatea statică. A doua lege a lui Krichhoff depinde de asumarea că sarcina în orice cablu, joncțiune sau componentă aglomerată se menține constantă. De fiecare dată când câmpul electric din părțile circuitului nu se poate neglija, cum ar fi când două fire sunt cuplate capacitiv, legea a doua a lui Kirchhoff poate să nu mai fie corectă. Acest lucru se întâmplă în circuitele de curent alternativ de frecvență înaltă. De exemplu, într-o linie de transmisie, densitatea de sarcină în conductor se poate schimba constant. Sarcina netă în diferite părți ale conductorului se modifică în timp. ceea ce rezulta ca acest lucru încalcă prima lege a lui Kirchhoff [10]. Aproximarea cu model clasic pentru un circuit electric este foarte bună la frecvențele joase. La frecvențe ridicate, fluxurile scurse și densitățile variate de sarcină în fire devin destul de semnificative [11]. Într-o oarecare măsură, este realizabil să modelăm în continuare astfel de circuite dacă anumite componente parazite au fost adăgate. Dacă frecvențele sunt foarte ridicate, cea mai bună metodă este să simulăm câmpurile direct folosind modelare cu elemente finite sau oricare alte metode [11]. Pentru a lua un exemplu putem să presupunem o rețea electrică formată din două surse de tensiune și trei rezistențe, precum este cea prezentată în figura 1.7.

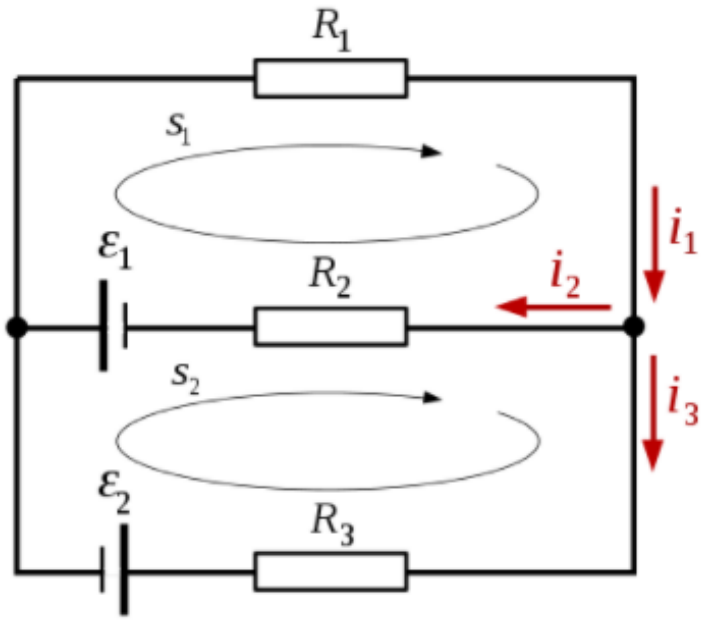


Figura 1.7. Rețea electrică formată din două surse și trei rezistențe [11].

Aplicând prima lege avem: $i_1 - i_2 - i_3 = 0$. Folosind a doua lege buclei inchise *s1*, si folosind legea lui Ohm pentru a substituii tensiunea, obținem următoarea relație: $R_2 i_2 - \mathcal{E}_2 + R_1 i_2 = 0$. Iar dacă folosim a doua lege a lui Kirchoff, iarăși, împreună cu legea lui Ohm, aplicată pe bucla închisă *s2*, obținem că $R_3 i_3 + \mathcal{E}_2 + \mathcal{E}_1 - R_2 i_2 = 0$. Din toate obținând un sistem de ecuații liniare

rezolvabil în i_1, i_2, și i_3. Asumând R_1 = 100Ω, R_2 = 200Ω, R_3 = 300Ω, $\mathcal{E}_1$ = 3V, $\mathcal{E}_2$ = 4V, vom obține soluțiile pentru curenți i_1 = 1/110A, i_2 = 4/275A și i_3 = - 3/220A. Semnul negativ al curentului i3 presupune ca se deplasează invers față de sensul pe care l-am ales în circuit.

1.3. Teorema lui Thévenin

Să considerăm o reţea activă la bornele A şi B la care conectăm un dipol activ sau pasiv, care reprezintă sarcina pentru reţea. Din punctul de vedere al dipolului, reţeaua activă este echivalentă cu o sursă de tensiune cu valoarea u_{ABgol} (tensiunea între bornele A şi B în absenţa sarcinii), conectată în serie cu impedanţa reţelei pasivizate, Z_{AB} [9]. Demonstraţia arată în felul urmator: Presupunem mai întâi situaţia în care dipolul conectat la ieşirea reţelei este un dipol pasiv (figura 1.8) [9]. În ramura conectată la bornele reţelei active se conectează formal două surse de tensiune la fel, cu valoarea aleasa arbitrat, ε, în aşa fel ca ele să-şi anuleze reciproc efectul asupra tensiunii şi curentului prin ramură - figura 1.9 (a). Practic, în circuit nu am schimbat nimic. Descompunem apoi reţeaua în două reţele, aşa cum este arătat în figura 1.9 (b), astfel încât, aplicând teorema superpoziţiei, $i_{sarc} = i_1 + i_2$ [9].

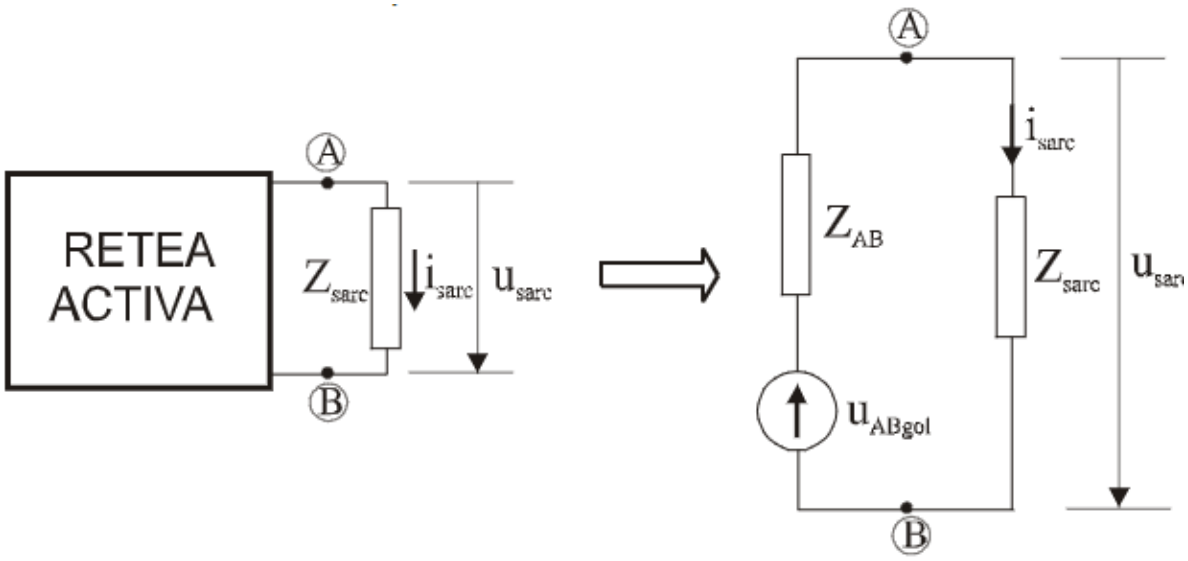


Figura 1.8. dipol pasiv conectat la o rețea activă [9].

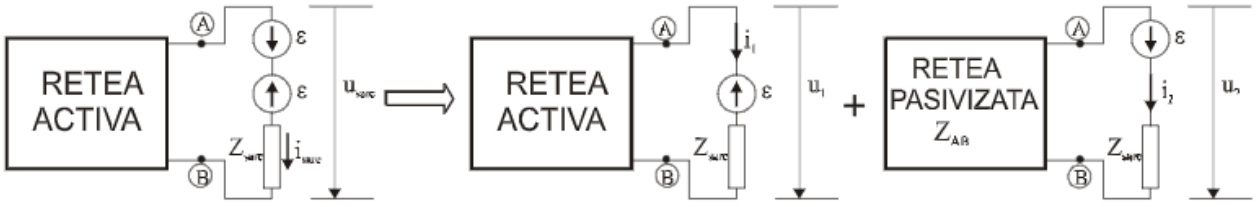


Figura 1.9. (a) Rețea activă cu 2 surse ce se anulează reciproc; (b) Descompunerea rețelei active [9].

Pentru tensiunea ε selectăm o valoare astfel încât $i_1 = 0$. Dacă i_1 este nul, se poate decupla sarcina realizând condiţiile de mers în gol, tensiunea la bornele libere fiind nulă. Acest lucru se poate realiza numai dacă alegem $\varepsilon = u_{ABgol}$ [9]. Să presupunem acum o situaţie în care la bornele reţelei este conectat un dipol activ - figura 1.10 (a). Utilizând teorema superpoziţiei, schema de față poate fi separată în două mai simple - figura 1.10 (b) [9].

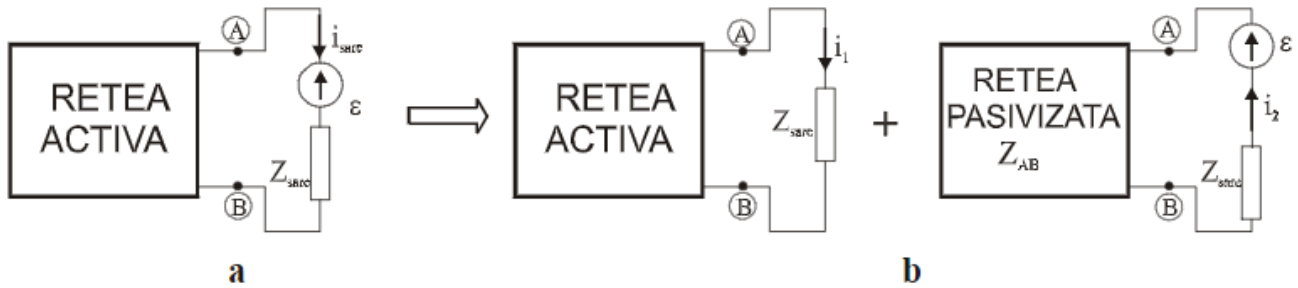


Figura 1.10. (a) Rețeaua echivalentă Thevenin; (b) Construcția rețelei echivalente Thevenin [9].

Având în vedere că știm cum se comportă o reţea activă faţă de un dipol pasiv, cele două scheme se pot transforma ca în figura 1.11, iar dacă aplicăm incă o dată teorema superpoziţiei dar în sensul opus, obţinem o schemă echivalentă care corespunde cu teorema [9].

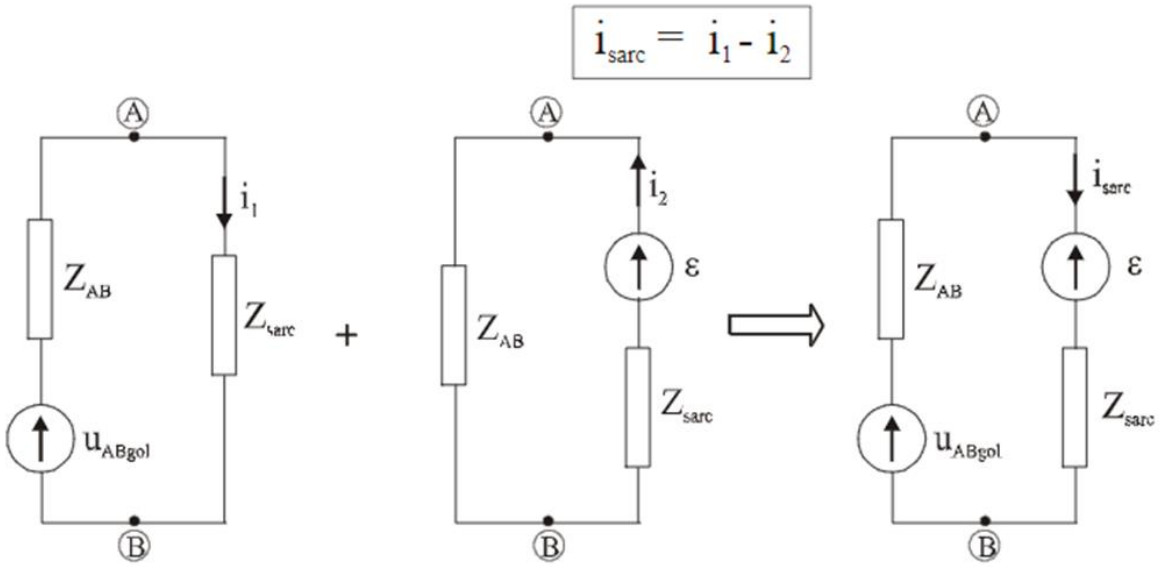


Figura 1.11 transmformarea celor 2 scheme de circuit într-o schemă echivalentă [9].

1.4. Teorema lui Norton

Considerăm reţeaua activă la bornele căreia este conectat un dipol activ. Din punctul de vedere al dipolului, reţeaua activă este echivalentă cu o sursă de curent cu valoarea i_{ABsc}, conectată în paralel cu impedanţa reţelei pasivizate, Z_{AB} (figura 1.12) [9].

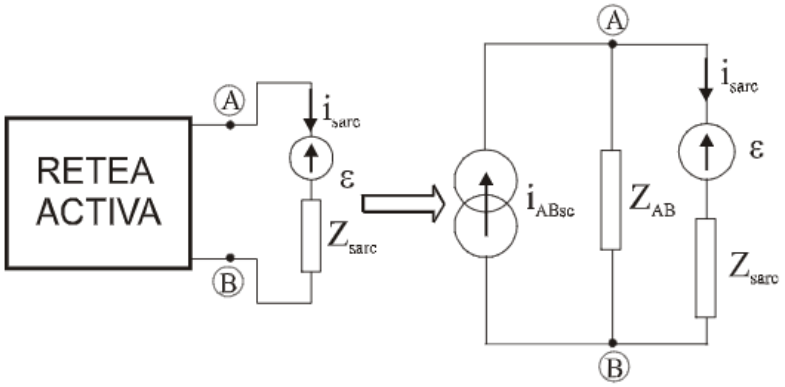


Figura 1.12. rețea activă conectată la un dipol activ și circuitul echivalent [9].

Demonstraţia arată în felul urmator: se aplică teorema lui Thévenin (demonstrată anterior) şi echivalenţa generator de tensiune – generator de curent, schema poate fi transformată succesiv ca în fig.1.13 cu menţiunea că raportul u_{ABgol} / Z_{AB} este curentul de scurt circuit, $i_{sc,}$ al reţelei active [9].

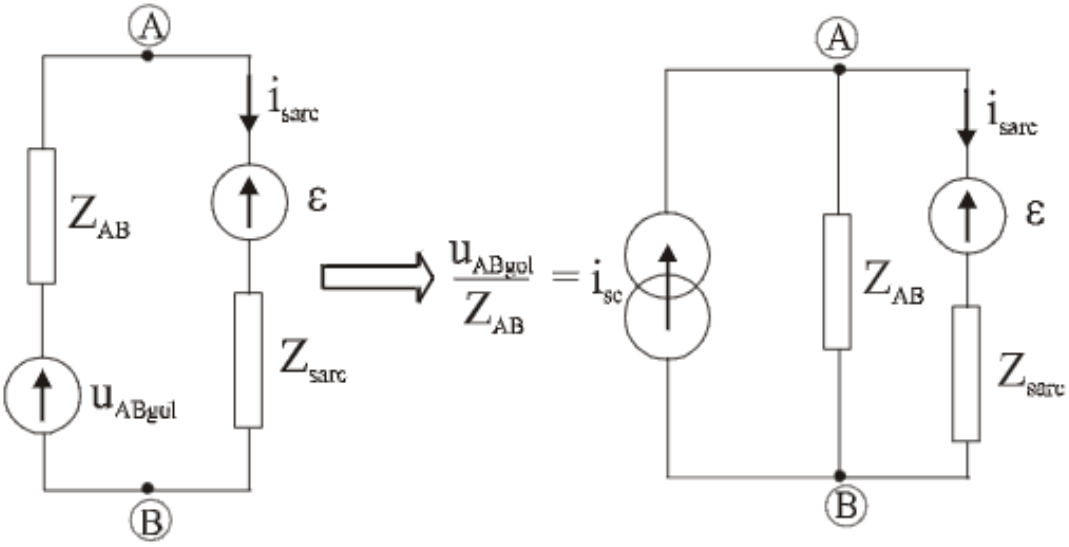


Figura 1.13 Transformarea schemei Norton echivalente [9].

1.5. Sursa de curent

Figura 1.14 prezintă simbolul schematic al unei surse de curent ideale cuplată la o rezistență. Există două tipuri: Surse de curent independente (sau chiuvete) care furnizează curent constant și surse de curent dependente care furnizează putere de curent proporțională cu alte tensiuni sau curenți din acel circuit [10].

Figura 1.14. Sursă de curent ideală [12].

O sursă de curent ideală este una care asigură un curent constant, indiferent de tensiune. Schematic, sursa de curent apare așa cum se arată în Figura 1.15, unde săgeata indică direcția curentului pozitiv (simbolul „I” poate sau poate să nu fie prezent) [12].

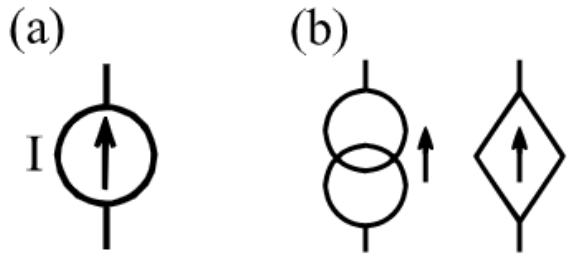


Figura 1.15. (a) Reprezentarea schematică a sursei de curent; (b) Reprezentări alternative [12].

Simbolul schematic utilizat pentru o sursă de curent constant este cel din figura 1.15 (a). Câteva simboluri alternative sunt uneori folosite pentru sursele de curent, cum ar fi în figura 1.15 (b). Simbolul romb este folosit în mod obișnuit pentru a desemna „sursă dependentă”, unde curentul depinde de ceea ce se întâmplă în altă parte a circuitului [10]. În circuitul din Figura 1.16 (a) putem afla tensiunea pe sursa de curent folosind reducerea circuitului. Rezistorul poate fi înlocuit cu un rezistor echivalent de valoare 1k + (2k‖3k) = (1 + 6/5) k = 2,2k. Circuitul echivalent, așa cum se vede de sursa de curent este prezentat în Figura 1.16 (b). Intensitatea curentului este de 30mA și, prin urmare, folosind legea lui Ohm, tensiunea la rezistența echivalentă este 2,2k × 30mA = 66V. În acest caz, a doua lege Kirchhoff cere ca aceasta trebuie să fie și mărimea tensiunii la sursa de curent. Majoritatea surselor de alimentare comerciale pot fi modelate sub forma unei surse de tensiune constante, deși pot apărea excepții [13].

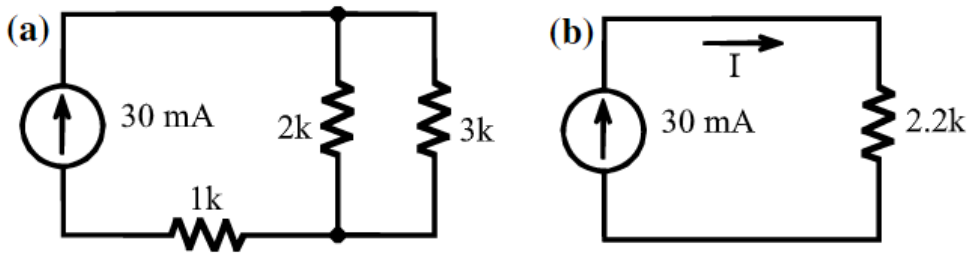


Figura 1.16(a) Circuit electric cu 2 bucle (b) Circuit echivalent simplificat [12].

O metodă de a găsi tensiunea la sursa de curent pentru un circuit, așa cum se arată în Figura 1.16 (a), este reducerea circuitului prin găsirea unei rezistențe echivalente așa cum este prezentată în Figura 1.16 (b).

1.6. Sursa de tensiune

Sursa de tensiune este un dispozitiv cu două terminale la care este posibilă menţinerea unei tensiuni fixe [4]. Sursa ideală de tensiune poate menține o tensiune fixă indiferent de rezistența de sarcină sau curentul de ieșire. Cu toate acestea, o sursă de tensiune reală nu poate furniza curent nelimitat [14]. Sursele reale de energie, cum ar fi bateriile și generatoarele, pot fi modelate pentru analiză ca o combinație între o sursă ideală de tensiune și o combinație complementară de elemente de impedanță [10]. Simboluri folosite pentru surse se pot vedea în figura 1.17.

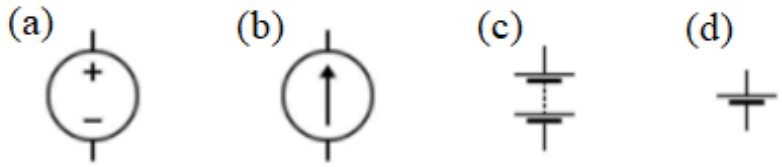


Figura1.17. (a) sursă ideală de tensiune; (b) sursă ideală de curent; (c) baterii în serie; (d) baterie singulară [14].

Sursa ideală de tensiune este un dispozitiv cu două terminale care menține o cădere fixă de tensiune la bornele sale. Este adesea folosită ca o abstractizare matematică pentru a simplifica analiza circuitelor reale [15]. Dacă tensiunea pe sursa ideală de tensiune poate fi determinată independent de orice altă variabilă din circuit, atunci se numește sursă de tensiune independentă [10]. Totuși, dacă tensiunea pe o sursă ideală de tensiune este determinată de o altă tensiune sau de un curent diferit dintr-un circuit, se numește sursă dependentă de tensiune sau sursă controlată de tensiune [13]. Un model sub formă matematică al unui amplificator poate fi sub forma unei surse dependente de tensiune a cărei valoare este dată de o relație fixă cu semnalul de intrare

[16]. Rezistența internă a unei surse ideale de tensiune este nulă, având capacitatea de a furniza sau absorbi orice cantitate de curent. Mărimea curentului care curge prin sursa ideală de tensiune este complet determinată de circuitul extern. Când este conectată la un circuit deschis, avem curent nul și, prin urmare, putere nulă. Când e conectată la un rezistor de sarcină, curentul prin sursă tinde spre infinit, deoarece rezistența de sarcină tinde spre zero (scurtcircuit). Astfel, o sursă de tensiune ideală poate furniza putere nelimitată [13].

Nicio sursă de tensiune reală nu este ideală; toate au rezistență internă efectivă diferită de zero și niciuna nu poate furniza curent nelimitat [10] .Cu toate acestea, rezistența internă a unei surse de tensiune reală este modelată eficient în analiză circuitelor integrând un circuit liniar prin combinarea unui rezistor diferit de zero în serie cu o sursă de tensiune ideală (un circuit echivalent Thévenin) [13]. Sursele de tensiune paralele divizază curentul de sarcină: dacă o copie exactă a tensiunii este conectată în paralel cu originalul, unul dintre ele va furniza jumătate din curentul care a furnizat originalul voltaj [15]. Circuitul rămas rămâne neschimbat: cele două surse de tensiune asigură aceeași tensiune și același amperaj față de varianta originală - figura 1.18.

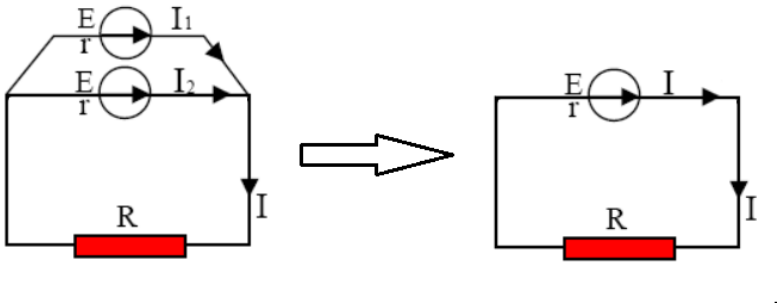


Figura 1.18. Gruparea surselor de tensiune identice. [17].

1.7. Surse stabilizate: Sursa de alimentare în comutație

O sursă de alimentare este un dispozitiv electric care convertește curentul electric ce vine de la o sursă de energie, în valoarea tensiunii necesare pentru alimentarea unei sarcini cum ar fi un motor sau un dispozitiv electronic [18]. Există două modele principale de surse de alimentare: sursă de alimentare liniară și sursă de alimentare în comutație [18]. Modelul de alimentare liniar folosește un transformator pentru a reduce tensiunea de intrare. Tensiunea este apoi rectificată și convertită într-o tensiune DC, care este apoi filtrată pentru a îmbunătăți calitatea formei de undă. Sursele de alimentare liniare folosesc un regulator liniar pentru a menține o tensiune de ieșire constantă. Aceste regulatoare liniare disipă orice energie suplimentară sub formă de căldură. Comutarea sursei de alimentare este o metodă dezvoltată recent pentru a rezolva multe probleme legate de proiectarea sursei de alimentare liniare, inclusiv dimensionarea transformatorului și

reglarea tensiunii. În proiectarea sursei de alimentare cu comutație, tensiunea de intrare nu mai scade; în schimb, este rectificată și filtrată la intrare. Tensiunea este apoi trecută printr-un „cutter", transformându-l într-un tren de impulsuri de înaltă frecvență. Înainte ca tensiunea să ajungă la ieșire, aceasta va fi filtrată și redresată.

Timp de mulți ani, sursele de alimentare liniare AC/DC au convertit puterea AC de la rețea în tensiune DC pentru aparatele de uz casnice sau iluminatoare [15]. Surse cu dimensiuni mai scăzute care pot furniza putere mare au devenit utile pentru aplicații industriale specifice și utilizări medicale, unde sunt încă necesare datorită zgomotului lor redus [18]. Au fost adoptate sursele de alimentare cu comutație deoarece sunt mai mici, mai eficiente și au capacități de manipulare a puterii mari. Figura 1.19 ilustrează conversia de la curent alternativ la curent continuu folosind o astfel de sursă [18].

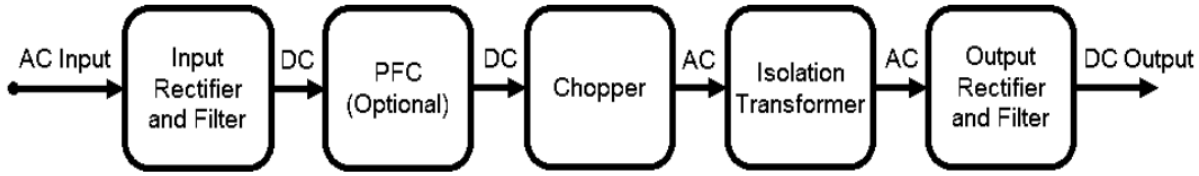


Figura 1.19. Sursă de alimentare izolată AC/DC în mod comutat [18].

Prin redresare se intețege procesul de convertire a unei tensiuni alternative într-un curent continuu [15]. Acesta reprezintă primul pas în sursele de alimentare AC/DC în mod comutat. Se crede în mod obișnuit că tensiunea de curent continuu este un flux constant de tensiune dreaptă, constantă, ca modelul care iese dintr-o baterie. Totuși, ceea ce definește curentul continuu (DC) este fluxul de curent continuu de încărcare. dar nu neapărat constantă [15]. Unda sinusoidală este cea mai tipică formă de undă AC și este pozitivă pentru prima jumătate a oscilației, dar negativă pentru restul oscilației. Dacă porțiunea negativă este inversată sau eliminată, curentul încetează să mai fie alternativ și devine curent continuu. Acest lucru se poate face printr-un proces care se numește redresare [15].

Putem realiza o astfel de redresare folosind un redresor pasiv în jumătate de punte care, folosind o diodă, elimină partea negativă a undei de formă sinusoidă - Figura 1.20. Dioda permite doar în timpul domeniului pozitiv al oscilației, trecerea curentului prin aceasta, blocând curentul în cazul contrar [18].

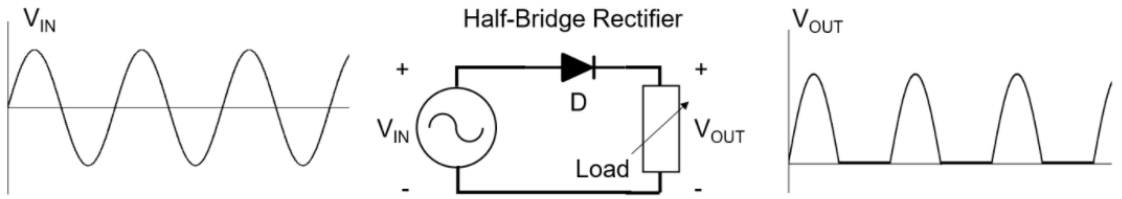


Figura 1.20. Redresare monoalternantă (Half-Bridge) [18].

Unda sinusoidală rezultată după redresare are o putere medie scăzută și nu poate alimenta dispozitivele în mod foarte eficient [15]. O metodă mult mai eficientă este schimbarea polarității semi-undei negative și să o trecem în latura pozitivă. Această metodă este cunoscută ca redresare cu undă completă și pentru realizarea ei e nevoie de doar patru diode puse într-o configurație de punte ca în figura Figura 1.21. Acest aranjament poate să mențină o direcție stabilă de deplasare a curentului, independent de polaritatea tensiunii [15].

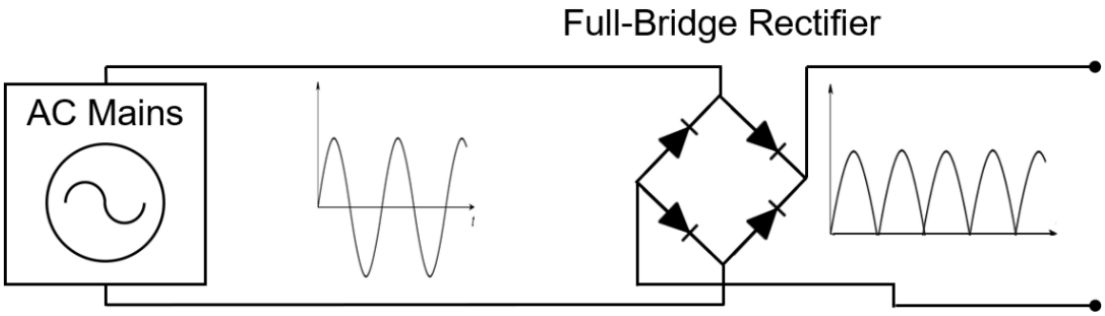


Figura 1.21. Redresare bialternantă (Full-Bridge) [18].

Dacă o undă e complet redresată, va avea o tensiune medie de ieșire mult mai mare decât cea care e produsă cu o redresare monoalternantă, dar este încă foarte departe de a fi echivalentă cu forma de undă constantă de curent continuu necesară pentru alimentarea dispozitivelor. Deși aceasta este o undă de curent continuu, folosirea acesteia pentru alimentarea unui dispozitiv este destul de ineficientă pentru că forma undei de tensiune este foarte variabilă, schimbând-și valoarea foarte rapid și des. Modificarea periodică a tensiunii de curent DC se numește ondulație Reducerea sau eliminarea acestei ondulații este absolut necesară pentru ca o sursă de alimentare să fie eficientă [15].

Cea mai intalnită metodă utilizată pentru reducerea ondulației este folosind un condensator mare la ieșirea redresorului. Acest condensator poartă numele de filstru de netezire (Figura 1.22) [18]. Condensatorul reține tensiune în timpul valorii maxime al undei, apoi când tensiunea tensiunea scade sub nivelul tensiunii semnalulu redresat de la intrarea condensatorului, el va alimenta sarcina cu curent. Forma de undă care reiese din acest proces se apropie mult mai mult de forma pe care o dorim și poate fi considerată o tensiune DC, care furnizează curent

continuu. Această formă finală de tensiune poate fi utilizată fără probleme pentru alimentarea dispozitivelor DC.

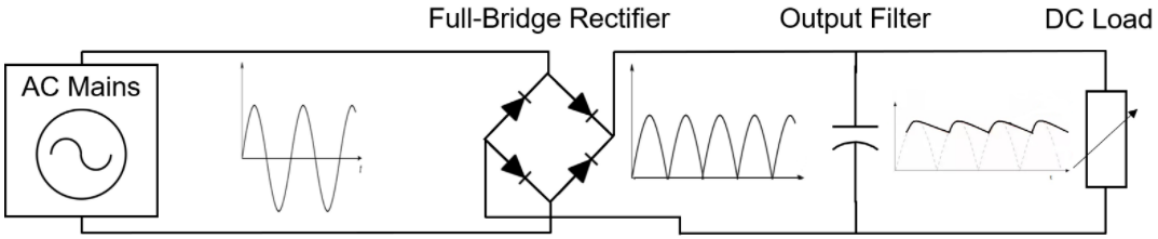


Figura 1.22. Redresare bialternantă cu filtru de netezire [18].

Diodele semiconductoare sunt folosite la redresarea pasivă ca întrerupătoare necontrolate. Redresarea pasivă este cea mai simplă metodă de a redresa o undă de curent alternativ, dar totuși există metode și mai eficiente [18]. Diodele reprezintă comutatoare relativ eficiente, care se pot activa și dezactiva rapid cu pierderi de putere foarte puține. Singura problemă cu acestea este că au căderea de tensiune de polarizare directă de la 0,5 V la 1 V, fact care va reduce semnificativ eficiența. Redresarea activă înlocuiește astfel de diode cu niște comutatoare controlate, de exemplu tranzistoare BJT sau MOSFET-uri (Figura 1.23) [18]. Avem două avantaje în acest caz: Priml avantaj este că redresoarele pe bază de tranzistori elimină această cădere fixă de tensiune de 0,5V până la 1V asociată cu diodele semiconductoare, fiindcă au rezistențe ce sunt făcute arbitrar mult mai mici și, în consecință, au o cădere mică de tensiune. Al doilea avantaj este că tranzistoarele sunt comutatoare controlate, fapt din care reiese că oscilația de comutare poate fi controlată și optimizată intr-un mod eficient. Totuși, avem și un dezavantaj, care este faptul că redresoarele active necesită circuite de control mult mai complicaet, ceea ce insamnă că e nevoie de componente suplimentare care ar putea fi mult mai scumpe și greu de intreținut [18].

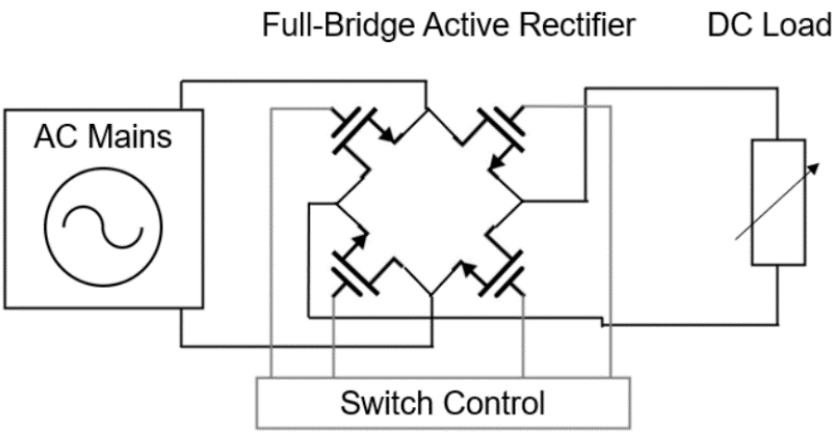


Figura 1.23. Redresare bialternantă activă [18].

1.8. Comunicații seriale

Transmisia de date serială, este o metodă de comunicație între dispozitivele periferice și computer în care biții de date sunt transferați succesiv de-a lungul unui canal de comunicare.

Portul serial al unui computer este portul COM1. Portul serial nu aste altceva decăt o priză care face posibilă conectarea dispozitivelor periferice precum o tastatură sau un alt dispozitiv să se conecteze printr-un cablu la calculator [19]. În general, calculatoarele care sunt mai noi se conectează la dispozitive prin porturi USB, și nu prezintă porturi COM1 [19].

Port serial are acest nume numele datorită faptului că datele trec printr-un un cablu în serie (figura 1.24). În schimb, într-un port paralel datele se circulă într-un computer prin opt fire ce sunt paralele(figura 1.24) [19]. Într-un port paralel, dacă octetul de date părăsește un punct va ajunge simultan celălalt. Octetul este de 8 biți de date, iar fiecare bit călătorește pe unul dintre fire. Acesta este demersul datelor dacă avem un cablu paralel ce este conectat la un port paralel, dar un port serial va trimite toate datele pe același fir și va el trebui să reorganizeze ordinea acestor date încât fiecare bit să-i urmeze pe ceilalți în ordine, spre diferență de portul paralel unde avem 8 biți care se deplasează deodată pe fire paralele [20].

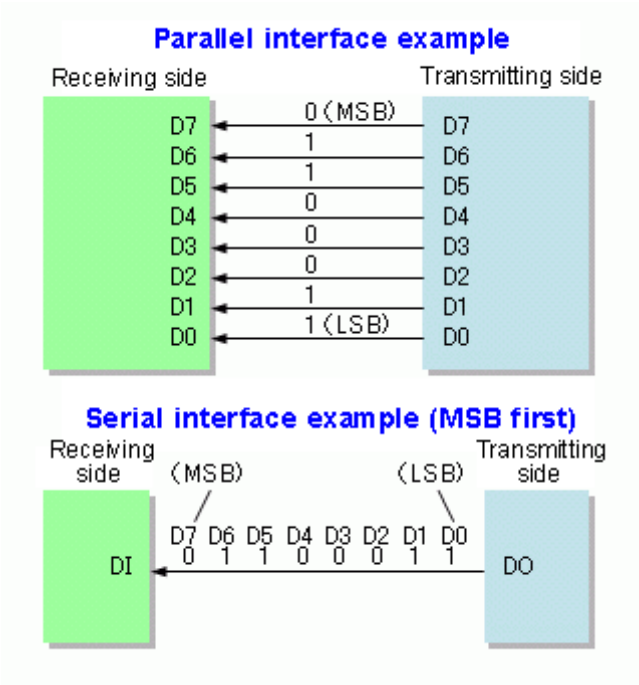


Figura 1.24. Transmisie paralelă și transmisie serială [20].

Un computer vechi, de exemplu, se poate să aibă un port serial, eventual un port paralel și trei porturi „COM”. Un port COM este un port serial. Numele „COM” provine de la „comunicare” și convenția de denumire este creată de Microsoft pentru sistemul lor de operare. Un port COM virtual poate fi creat prin maparea mai multor nume „COM” la același port fizic.

Datorită simplității sale relative și costului hardware scăzut (comparativ cu interfețele paralele), comunicația serială este utilizată pe scară largă în industria de dispozitive electronice. Standardul cu specificația EIA/TIA-232-E este cel mai popular standard de comunicații seriale [21]. Acesta e dezvoltat de Asociația Industriei Telecomunicațiilor (EIA/TIA) și Asociația Industriei Electronice. Numele lui simplu este RS-232, unde RS reprezintă „standard recomandat”.

Denumirea oficială al standardului EIA/TIA-232-E este „Interfață între echipamentele de terminare a circuitelor de date și echipamentele terminale de date folosind schimbul de date binare în serie” [21]. Standardul se ocupă de comunicarea serială a datelor între sistemul gazdă (Data Terminal Equipment sau DTE) și sistemul periferic (Data Circuit Termination Equipment sau DCE) [21]. Standardul EIA/TIA232E a fost introdus în 1962 și a fost actualizat de patru ori de atunci pentru a răspunde nevoii tot mai mari ale aplicațiilor de comunicare serială. Litera „E” din numele standardului indică faptul că aceasta este a cincea revizuire a standardului [21].

Standardurile RS-232 sunt complete. Ceea ce înseamnă că își propun să ofere compatibilitate între sistemele gazdă și periferice prin specificarea [21]:

1. Niveluri comune de semnal și de tensiune
2. Configurație de cablare a pinilor comună
3. O cantitate cât de mică de informație de control între sistemele periferice și gazdă.

Spre deosebire de multe standarde care specifică doar caracteristicile electrice ale unei interfețe date, RS232 specifică proprietăți electrice, funcționale și mecanice pentru a îndeplini cele trei criterii de mai sus [21].

RS232 funcționează pe un sistem de schimb de date în două sensuri [21]. Sunt două dispozitive vor fi conectate,Dispozitivul Data Transmission Equipment și dispozitivul(DTE) Data Communication Equipment (DCE), care au pini precum RTS, CTS, TXD și RXD. din sursa DTE, RTS face cererea de a se trimite date [21]. Apoi, din capătul celălalt DCE, CTS, permit primirea acestor date. Acesta va da un semnal către RTS al sursei DTE pentru trimiterea semnalului [21]. Apoi biții vor fi transmiși de la DTE la DCE. Acum din nou de la sursa DCE, cererea este generată de RTS și CTS ale surselor DTE pentru a a putea fi permisă primirea datelor și va da un semnal pentru trimiterea acestora [21]. Acesta este întregul proces prin care are loc transmiterea datelor. Schema sistemului se poate vedea în figura 1.25.

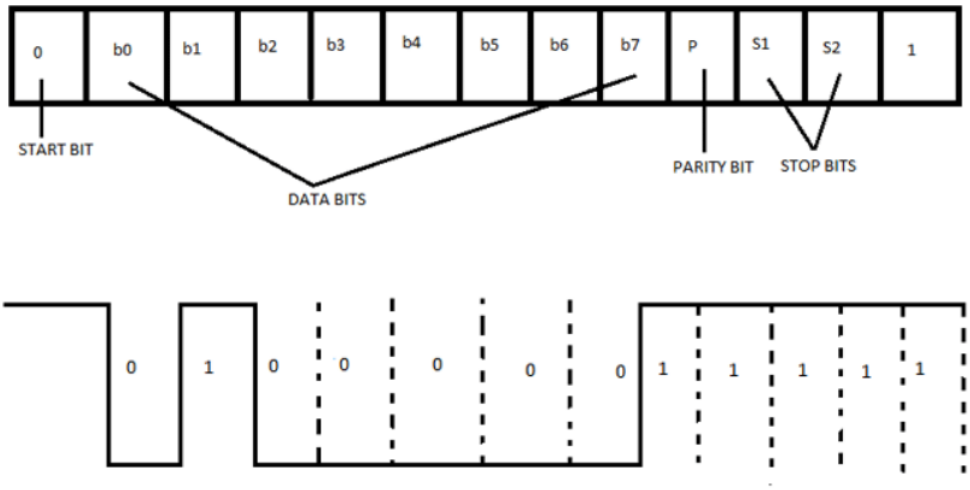


Figura 1.25. Schema sistemului RS232 [21].

Portul USB este interfața standard cu fir pentru computere personale și dispozitive electronice cu consum mare [22]. USB înseamnă Universal Serial Bus, un standard industrial pentru comunicația de date digitale pe distanță mică. Portul USB permite dispozitivelor USB să fie conectate între ele și să transfere date digitale printr-un cablu USB. De asemenea, pot furniza energie electrică prin cablu dispozitivelor care au nevoie de ea [22]. Standardul USB are două versiuni, versiunea cu fir și versiunea fără fir. Numai versiunea cu fir implică porturi și cabluri USB.

Pentru a fi posibilă înțelegerea modurilor în care transferul de date se utilizează în mediul unui USB, este necesară să înțelegerea configurației fizice pentru un port USB și cum apare acest port pe sistem [23]. Pentru majoritatea sistemelor care folosesc un port USB, gazda va fi o formă de computer: laptop, desktop etc. În ceea ce privește definirea entităților prezente în rețeaua USB, există următoarele elemente cheie [23]:

- Gazdă: O gazdă este un computer sau un element care funcționează în principal pe USB. Aceasta conține un hub numit Root Hub.
- Hub: Un hub este un dispozitiv care mărește numărul de porturi disponibile. Un hub poate fi conectat la altul pentru a crește capacitatea.
- Port: Portul oferă accesul la rețeaua USB. El se poate afla pe un hub sau pe o gazdă.
- Funcții: Acestea sunt dispozitivele sau lucrurile la care este conectat USB-ul. Mouse, tastatură, memorie flash etc.
- Dispozitiv: termenul dispozitiv este se folosește în mod colectiv pentru funcții și hub-uri.

Un aranjament fizic obișnuit pentru o rețea USB poate fi obesrvată în figura 1.25.

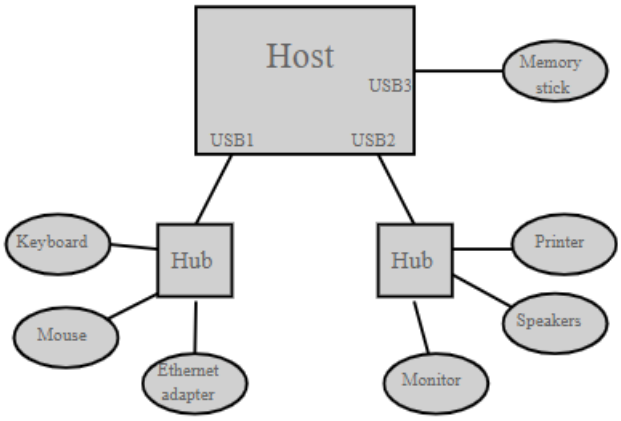


Figura 1.25. Aranjament intr-o rețea USB [23].

Cablurile USB permit mai multe versiuni ale standardului USB de la versiunea 1.1 la 3.1 în prezent. Toate porturile USB arată la fel, indiferent de versiunea USB acceptată [24].

Există mai multe standarde USB majore, USB4 este cel mai recent. Versiunile USB posibile sunt [24]:

- USB4: acceptă 40 Gb/s 40.960 Mbps și e bazat pe specificația Thunderbolt 3
- USB 3.2 Gen 2x2: Cu denumirea anterioară USB 3.2, dispozitivele compatibile cu acesta sunt capabile să transfere date la 20.480 Mb/s, cunoscut sub numele de Superspeed+ USB Duallane.
- USB 3.2 Gen 2: Acesta a fost numit USB 3.1. Dispozitivele compatibile sunt capabile să transfere date la 10.240 Mb/s, sub numele de Superspeed+.
- USB 3.2 Gen 1: Inițial numit USB 3.0, hardware-ul compatibil cu acesta poate atinge o rată de transmisie maximă de 5.120 Mb/s, sub numele de SuperSpeed USB.
- USB 2.0: Dispozitivele compatibile cu acesta pot atinge o rată de transmisie maximă de 480 Mb/s, sub numele de USB de mare viteză.
- USB 1.1: Dispozitivele de acest tip pot atinge o rată de transmisie maximă de 12 Mb/s, cunoscut sub numele de Full Speed USB.

Majoritatea dispozitivelor și cablurilor USB de astăzi sunt compatibile cu USB 2.0 și un număr tot mai mare de USB 3.0. Părțile unui sistem conectat prin USB, inclusiv gazde (cum ar fi computere), cabluri și dispozitive, pot accepta diferite standarde USB, atâta timp cât sunt compatibile. Cu toate acestea, toate aceste părți trebuie să accepte același standard pentru a putea fi posibilă atingerea ratei maxime de date posibile [24].

Multe dispozitive mobile, cum ar fi, cititoarele electronice, telefoanele mobile și tabletele mici, folosesc în principal USB pentru încărcare. Încărcarea prin USB a devenit atât de populară încât prizele de schimb pot fi găsite cu ușurință în magazinele de îmbunătățiri pentru locuințe, cu porturi USB încorporate, eliminând nevoia unui adaptor USB de alimentare.

2. Noțiuni fundamentale de programare în Octave

2.1. Introducere a programului Octave

GNU Octave este un limbaj avansat destinat calculelor numerice. Cel mai des este folosit pentru probleme precum rezolvarea ecuațiilor liniare și neliniare, algebră liniară numerică, analiză statistică și pentru alte experimente numerice. De asemenea, Octave se poate utiliza ca limbaj orientat pe loturi pentru procesarea automată a datelor [25]. Octave în versiunea lui prezentă este executat într-o interfață grafică cu utilizatorul (GUI). GUI folosește un IDE (mediu de dezvoltare integrat) care include un editor de cod cu evidențierea sintaxelor, un browser de documentație, un depanator încorporat, precum și un interpret pentru limbajul în sine. De asemenea, o interfață de tip command-line este valabilă pentru Octave [25]. Pe majoritatea sistemelor de operare, Octave poate fi pornit cu comanda ce se scrie în terminal „octave”, pornind intarfața grafică cu utilizatorul. Interfața de linie de comandă din octave Octave este fereastra centrală din GUI. Folosind această fereastră, Octave afișează inițial un mesaj urmat de un prompt care va indica că va fi pregătit să accepte input-ul [25].

2.2. Tipuri de date încorporate în Octave

În Octave, avem mai multe tipuri de date standard încorporate. Acestea sunt sunt scalari și matrici reali și complexi, un tip de structură de date, intervale, șiruri de caractere, și matrice de celule. Folosind funcțiile descrise mai jos se poate determina respectiv modifica tipul de date al diferitelor variabile.

Funcția **classname = class (obj), class (s, id), class (s, id, p, ...)** va returna clasa obiectului *obj* sau va creea o clasă cu câmpuri din structura *s* si numele *id*. Orice argument suplimentar va numi o listă de clase părinte din care se va derivata noua clasă [25].

Funcția **isa (obj, classname)** returnează true dacă *obj* face parte din clasa *classname* [25].

Variabila *classname* poate să reprezinte una din categoriile [25]:

- „float” - Valoare cu virgulă ce include clasele „sinle” și „double”.
- "integer" - Valoare întreagă care cuprinde clasele (u)int64, (u)int32, (u)int16, (u)int8.
- "numeric" - Valoare numerică care cuprinde fie o valoare în virgulă mobilă, fie o valoare întreagă.

Dacă *classname* reprezintă o matrice de celule de șir, se va returneaza o matrice logică cu dimensiune egală, care va conţine true pentru fiecare clasă din care aparţine *obj*.

Funcția **Cast (val, "type")** va converti variabila *val* la tipul de date *type* [25]. Atât *val* cât și *type* sunt de obicei una dintre tipurile de date încorporate [25]: **"double", "single", "logical", "char", "int8", "int16", "int32", "int64", "uint8", "uint16", "uint32", "uint64".** Valoarea *val* are putea fi schimbată pentru potrivirea ei în intervalul noului tip. Luând două exemple, **cast (-5, „uint8")** returnează 0, iar **cast (300, „int8"** returnează 127.

Funcția **y = typecast (x, "class")** va returna o nouă matrice y care se va deduce din interpretarea datelor lui x din memorie ca date ale clasei numerice [25]. Nu doar clasa lui *x*, dar și *class* vor trebui să fie una dintre clasele numerice încorporate în Octave: **"logical", "char", "uint8", "uint16", "uint32", "uint64","int8", "int16", "int32", "int64", "double", "single", "double complex", "single complex"** [25]**.** ultimele două vor putea fi folosite doar cu *class*; ele vor indica faptul că rezultatul solicitat este de valoare complexă.

Funcția **swapbytes (x)** Schimba ordinea byte-urilor la valori, convertind din little-endian în big-endian si vice versa [25]. De exemplu, **swapbytes (uint16 (1:4))** returnează: 256 512 768 1024

Funcția **y = bitpack (x, class)** returnează o nouă matrice y rezultată din interpretarea matricei logice x ca modele de biți brute pentru datele clasei numerice [25]. Variabila *class* va trebui să fie una dintre clasele încorporate în Octave: **"double", "single", "double complex", "single complex", "char", "uint8", "uint16", "uint32", "uint64","int8", "int16", "int32", "int64"** [25]**.** Numărul elementelor lui x trebuie să fie divizibil cu lungimea de biți a clasei. Dacă nu este divizibil, biții care rămân în exces vor fi eliminați.

Funcția **y = bitunpack (x)** va returna o matrice logică y care o să fie corespunzătoare modelelor de biți bruti ale lui *x* [25]. Unde *x* aparține uneia dintre clasele incorporate în Octave: **"double", "single", "char", "uint8", "uint16", "uint32", "uint64", "int8", "int16", "int32", "int64"** [25]**.** Dacă x este un vector, rezultatul va fi tot un vector de același tip de care este și x. Dacă x este de tip rând, la fel va fi și rezultatul. Dacă x este de tip coloană, atunci si vectorul y va fi de tip coloană

2.3. Variabile

Variabilele ne permit să numim valorile și apoi să putem face referirea la ele mai târziu. Numele variabilei ar putea să fie o succesiune de litere, cifre și liniuțe de subliniere, dar în nici un caz nu poate ca numele să înceapă cu o cifră. Octave nu impune nici o limită a lungimii maxime posibile pentru numele unei variabile, dar o este rar util cazul atunci variabilele au nume mai lungi de 30 de caractere [25]. Octave are capacitatea de a ține în memorie șiruri de până la

2^{31} - 1 mărime [25]. Cu toate acestea, pentru compatibilitatea cu Matlab, toate aceste nume de câmpuri de variabile, funcții și structuri sunt mai scurte decât lungimea care se returnează de **namelengthmax**. În special, dacă avem variabile stocate într-un format de fișier Matlab (*.mat) ele nu vor avea numele lor complet, în schimb vor fi trunchiate la această lungime [25].

O variabilă globală este o variabila care va fi accesată oriunde în Octave. spre deosebire de o variabilă locală care se nu poate accesa în afara contextului său actual doar dacă această variabilă este transmisă explicit, cum ar fi dacă o includem ca parametru la apelarea unei funcții **(fcn (var_x, var_y))** [25]. Declararea globală pentru variabile se face folosind declarația "global". Toate declarațiile urmatoare sunt declarații de tip global: **global x, global y z, global x = 6, global z = 56 x y = 33**.

Această declarație globală poate fi extinsă numai până la indicatorul ce reprezintă sfârșitul de instrucțiune. Acesta poate să fie virgulă (','), punct și virgulă (';') sau linie nouă (''\n''). De exemplu, codul **global x, y = 4** va declara o variabilă globală, x și o variabilă locală y la care se dau valoarea 4. O variabilă globală va putea fi inițializată o singură dată într-o astfel de isntrucțiune [25]. De exemplu, dacă executăm codul **global x = 11, global x = 12** valoarea variabilei globale x va fi 11, nu 12. Comanda **clear x** nu schimbă comportamentul de mai sus, însă comanda **clear all** va schimba.

Pentru accesarea variabilei globale în cadrul unei funcții, necesară declarația acelei variabile ca globală și în cadrul corpului funcției [25]. De exemplu codul **global y; function f () y = 3; endfuction f ()** nu va seta valoarea variabilei globale y la 3. În schimb, este creată o variabilă locală, denumită y, care va avea valoarea 3. Pentru a modifica valoarea variabilei globale y, trebuie va trebui să existe declarația globală și în corpul funcției, în următorul fel: **function f () global y; y = 3; endfunction**. Trecerea unei variabile globale în lista de parametri ai funcției va face o copie locală, și nu va schimba valoarea globală al acestei variabile [25]. De exemplu, pentru funcția **Functionf (z) z = 10; endfunction** și definiția lui z ca variabilă globală la nivelul superior **global z = 13,** expresia f (z) o să afișeze valoarea lui z din interiorul său ca fiind 10, dar valoarea lui z la nivelul global va fi 13, deoarece funcția funcționează cu o copie a acestei variabile, nu cu variabila globală.

Dacă avem variabilă pe care o vom declara persistent într-o funcție, ea își va putea păstra conținutul în memorie între toate apelurile ulterioare de către aceeași funcție [25]. Diferența dintre variabilele globale și cele persistente este că variabilele persistente au un domeniu de aplicare local pentru o anumită funcție și ele nu se vor putea vedea alte părți. Declararea unei variabile ca fiind una persistentă se poate realiza folosind funcția de declarație **persistent** [25].

Toate urmatoarele declarații sunt de tip persistent: **persistent x, persistent x y, persistent z = 5, persistent x = 7 z y = 9.** Similar cu cazul variabilelor globale, variabilele persistent nu vor putea fi inițializate decăt o singură dată. de exemplu, executând liniile de cod **persistent x = 1, persistent x = 2**, valoarea pe care variabila x pvar o va avea este 1, nu 2.

2.4. Instrucțiuni

Instrucțiunile sunt expresii simple sau pot fi chiar liste complicate de instrucțiuni condiționale și bucle. Instrucțiunile de control precum *if*, *for* și așa mai departe pot controla fluxul de execuție în programele făcute în Octave [25]. Toate instrucțiunile de control trebuie să înceapă cu cuvinte cheie speciale, cum ar fi *for* și *while*, pentru a putea fi distinse de celelalte expresii. Fiecare instrucțiune de acest tip trebuie să aibă o instrucțiune de final corespunzătoare care semnifică sfârșitul instrucțiunii respective. De exemplu, instrucțiunea de final pentru o instrucțiune if este *endif*, și *endwhile* este sintrucțiunea ce semnifică sfârșitul unei instrucțiuni *while*.

Instrucțiunea decizională folosită de octave este *if*-ul[25]. Sunt valabile trei forme care stau la baza instrucțiunii *if*. În forma sa cea mai simplă, arată ca în figura 2.1:

```
if (condition)
  then-body
endif
```

Figura 2.1. Instrucțiunea *if* în formă simplă.

Unde *condition* este expresia care va controlaceea ce se intâmplă cu restul instrucțiunii. *then-body* va fi executat numai în cazul în care condiția e adevărată. Condiția dintr-o instrucțiune *if* va fi considerată adevărată doar dacă valoarea sa nu este nulă, altfel, ea va fi falsă [25]. Dacă valoarea expresiei condiționale dintr-o instrucțiune *if* este reprezentată de o matrice sau de un vector, va fi considerată adevărată numai în cazul în care aceeasta nu este goală și toate elementele ei sunt diferite de zero. A doua formă a unei instrucțiunii *if* arată ca în figura 2.2:

```
if (condition)
  then-body
else
  else-body
endif
```

Figura 2.2. Instrucțiunea *if* în a doua ei formă.

Dacă condiția este adevărată, atunci *then-body* o să fie executat. Dacă condiția e falsă, se va executa corpul *else-body*. Un exemplu de un astfel de cod se poate vedea în figura 2.3:

```
if (rem (x, 2) == 0)
  printf ("x is even\n");
else
  printf ("x is odd\n");
endif
```

Figura 2.3. Exemplu de cod ce conține structura *if* [25].

În acest exemplu, dacă **rem (x, 2) == 0** este considerată ca expresie adevărată, insemnând ca valoarea lui x se împarte la 2, atunci prima instrucțiune **printf** va fi executată, în caz contrar va fi executată a doua instrucțiune **printf.** A treia și cea mai generală formă a instrucțiunii if permite combinarea mai multor decizii într-o singură declarație. Arata ca în figura 2.4:

```
if (condition)
  then-body
elseif (condition)
  elseif-body
else
  else-body
endif
```

Figura 2.4. cea mai generală formă a structurei *if.*

Numărul de clauze else-if nu este limitat. Fiecare condiție va fi testată pe rând și, dacă se constată că este adevărate una dintre ele, corpul al cărei condiții a fost adevărată este executat. Dacă niciuna dintre condiții nu este adevărată dar avem la final o clauză else, corpul acesteia este executat. Aceasta poate apărea doar o singură dată și trebuie să fie ultima parte a declarației. În exemplul din figura 2.5, dacă prima condiție este considerată adevărată, adică x se imparte la 2, atunci prima instrucțiune **printf** va fi executată. În caz contrar, a doua condiție este testată dacă e adevarată, iar dacă este, rezultând că x e divizibil cu 3, atunci a doua instrucțiune **printf** va fi executată. În caz contrar, se execută utlima instrucțiune **printf**.

```
if (rem (x, 2) == 0)
  printf ("x is even\n");
elseif (rem (x, 3) == 0)
  printf ("x is odd and divisible by 3\n");
else
  printf ("x is odd\n");
endif
```

Figura 2.5. Exemplu de cod ce conține structura generală *if* [25].

Instrucțiunea *switch* este similară structurii *if*. Foarte comun este să fie nevoie de acțiuni diferite având în vedere valoarea unei variabile. Acest lucru este posibil folosind instrucțiunea *if* ca în figura 2.6:

```
if (X == 1)
  do_something ();
elseif (X == 2)
  do_something_else ();
else
  do_something_completely_different ();
endif
```

Figura 2.6. Secvență de cod complexă ce conține instrucțiunea *if* [25].

Acest tip de cod este foarte greu de scris și de întreținut. Pentru a rezolva problema de față, instrucțiunea switch este acceptată de Octave [25]. Folosind această afirmație, exemplul de mai sus devine ca în figura 2.7:

```
switch (X)
  case 1
    do_something ();
  case 2
    do_something_else ();
  otherwise
    do_something_completely_different ();
endswitch
```

Figura 2.7. Secvență de cod în care se folosește instrucțiunea *switch* [25].

Acest cod face structura repetată a problemei mai clară, ceea ce, la rândul său, face codul mai ușor de citit și, prin urmare, mai ușor de întreținut. De asemenea, dacă variabila *x* urmează să-și schimbe numele, schimbăm doar o linie de cod per caz atunci când sunt folosite instrucțiunile. Forma generală a unei astfel de instrucțiuni poate fi observată în Figura 2.8:

```
switch (expression)
  case label
    command_list
  case label
    command_list
  ...

  otherwise
    command_list
endswitch
```

Figura 2.8. Instrucțiunea *switch* în formă generală.

Unde *label* poate reprezenta orice expresie. Cu toate acestea, dacă valorile label-urilor sunt duplicate, ele nu vor fi detectate execuția va fi fi doar pentru prima listă cu care se potrivește [25]. Pentru ca instrucțiunea switch să fie semnificativă, trebuie să fie prezentă cel puțin o clauză command_list cu un label de caz, în timp ce clauza otherwise command_list este opțională. Dacă label-ul este reprezentat de o matrice, lista de comenzi este executată doar dacă oricare dintre elementele matricei respective se vor potrivii cu expresia [25]. De exemplu, programul din figura 2.9 va tipări „Variable is either 6 or 7”.

```
A = 7;
switch (A)
  case { 6, 7 }
    printf ("variable is either 6 or 7\n");
  otherwise
    printf ("variable is neither 6 nor 7\n");
endswitch
```

Figura 2.9. exempliu de cod ce utilizează instrucțiunea *switch* [25].

Instrucțiunea *while* reprezintă o buclă. O buclă înseamnă o parte a unui program de cod care este sau poate fi executată de mai multe ori la rând [25]. Instrucțiunea *while* reprezintă cea mai simplă instrucțiune de acest gen din Octave. Ea va executa în mod repetat o instrucțiune atâta timp cât o condiție se dovedește ca fiind adevărată. Ca și în cazul condiției dintr-o instrucțiune de tipul *if*, condiția într-o declarație *while* este adevărată doar dacă valoarea nu este nulă, în caz contrar se consideră ca fiind falsă. Dacă valoarea unei astfel de expresii dintr-o instrucțiune *while* este un vector sau o matrice, va fi considerată adevărată numai dacă aceast vector sau matrice nu sunt goale și toate elementele lor sunt nenule [25]. În Octave, instrucțiunea *while* arată ca în figura 2.10:

```
while (condition)
  body
endwhile
```

Figura 2.10. forma generală a instricțiunii *while*.

Aici *body* repreziontă o instrucțiune pe care le numim ”corpul buclei”, iar *condition* reprezintă o expresie care va controla cât timp bucla continuă să funcționeze. Primul lucru pe care îl va face isntrucțiunea *while* este să verifice dacă condiția de la început este adevărată. În caz afirmativ, se execută instrucțiuniile din corpul acestei bucle. După ce instrucțiunile au fost executate, condiția va fi testată încă o dată, iar dacă este încă adevărată, procesul se repetă din nou. Acest proces se va repeta până când condiția din *while* este falsă. Dacă condiția falsă de la început, nu se va executa nimic. Exemplul din figura 2.11 generează primele zece elemente din șirul lui Fibonacci.

```
fib = ones (1, 10);
i = 3;
while (i <= 10)
  fib (i) = fib (i-1) + fib (i-2);
  i++;
endwhile
```

Figura 2.11. Crearea șirului lui Fibonacci folosind instrucțiunea *while* [25].

Aici, în *body* se găsesc două instrucțiuni. Dacă urmarim demersul programului se întâmplă următoarele: mai întâi, lui *i* îi se atribuie valoarea 3. Apoi, instrucțiunea *while* va testa dacă valoarea lui *i* este mai mică sau egală cu valoarea 10. Având în vedere că *i* este egal cu 3, rezultă că condoția este adevărată, corpul buclei va fi executat, deci valoarea al i-lea element a lui *fib* va fi setată la suma dintre cele două valori anterioare. Apoi crește valoarea lui *i* cu 1, și bucla se va repeta până când *i* ajunge la 11.

Instrucțiunea *do-until* este o instrucțiune care funcționează pe aceleași principii cu instrucțiunea *while*, doar că execută în mod repetat o instrucțiune până când o condiție va devenii adevărată, iar testul condiției se va afla la sfârșitul buclei, rezultând faptul că oricare ar fi condiția de la final, corpul se va executa cel puțin o singură dată [25]. Condiția dintr-o astfel de instrucțiune este considerată adevărată doar dacă valoarea sa este nenulă în caz contrar se va considera ca fiind falsă. Dacă valoarea expresiei condiționale dintr-o instrucțiune *do-until* este reprezentată de un vector sau de o matrice, atunci condiția ei va fi considerată adevărată numai dacă matricea sau vectorul nu sunt goale, și toate elementele lor sunt nenule. Instrucțiunea do-until în Octave arată ca în figura 2.12:

```
do
  body
until (condition)
```

Figura 2.12 instrucțiunea *do-until* în forma generală.

Aici, *body* reprezintă corpul buclei, care poate fi o listă de instrucțiuni sau declarații, iar *condition* reprezintă o condiție care va controla cât timp instrucțiunea va continua să funcționeze. Exemplul din figura 2.13 creează o variabilă fib care conține primele zece elemente ale șirului lui Fibonacci:

```
fib = ones (1, 10);
i = 2;
do
  i++;
  fib (i) = fib (i-1) + fib (i-2);
until (i == 10)
```

Figura 2.13. Crearea șirului lui Fibonacci folosind instrucțiunea *do-until* [25].

Instrucțiunea *for* este similare cu celelalte instrucțiuni discutate mai sus. Forma generală a declarației for arată ca în figura 2.14 [25]:

```
for var = expression
  body
endfor
```

Figura 2.14. instrucțiunea *for* în forma generală.

Unde *body* reprezintă orice instrucțiune sau listă de instrucțiuni, *expression* este orice expresie validă și *var* poate lua mai multe forme. De obicei, ea poate fi un nume de variabilă simplă sau o variabilă indexată. Dacă valoarea expresiei este reprezentată de o structură, *var* ar putea fi reprezentat de un vector. Expresia de atribuire din instrucțiunea *for* funcționează puțin diferit față de instrucțiunea de atribuire normală a lui Octave [25]. În loc să atribuie rezultatul complet al acelei expresii, aceasta va atribui pe rând fiecare coloană a expresiei la *var*. Exemplul din figura 2.15 arată cum se poate folosi instrucțiunea for intr-o altă modalitate pentru creerea unui vector care conține primele zece elemente ale șirului lui Fibonacci:

```
fib = ones (1, 10);
for i = 3:10
  fib(i) = fib(i-1) + fib(i-2);
endfor
```

Figura 2.15. Crearea șirului lui Fibonacci folosind instrucțiunea *for* [25].

Codul de față funcționează prima oară prin verificarea expresiei *3:10*, pentru producerea unui interval de valori de la 3 la 10 inclusiv. Apoi variabilei *i* i se va atribui primul element din interval și se va executa corpul buclei. Când se va ajunge la sfârșitul acestuia, următoarea valoare din interval este atribuită lui *i*, și se mai execută încă o dată corpul buclei. Acest proces va continua până când nu vor mai fi elemente de atribuit.

2.5. Interfața grafică (GUI) în Octave

Octave este în principal un limbaj de linie de comandă. Cu toate acestea, oferă unele caracteristici pentru crearea de interfețe grafice cu utilizatorul. Elementele GUI disponibile sunt dialoguri I/O, bare de progres și elemente de interfață cu utilizatorul [25]. De exemplu, în loc să codifice numele fișierului pentru ieșire, scriptul ar putea deschide un dialog și să lase utilizatorul să selecteze fișierul. În mod similar, dacă este de așteaptat ca un calcul să trebuiască să dureze

mult timp, un script poate afișa o bară de progres. Diverse elemente ale UI pot fi folosite pentru a personaliza complet fereastra diagramei cu bare de meniu, bare de instrumente, meniuri contextuale, butoane și multe altele. Setul de caracteristici de interfață cu utilizatorul funcționează cel mai bine cu setul de instrumente grafice qt, deși unele funcționalități sunt disponibile cu setul de instrumente fltk [25].

Crearea unui obiect uipanel se poate vedea în figura 2.16

```
uipanel (property, value, ...)
uipanel (parent, property, value, ...)
hui = uipanel (...)
```

Figura 2.16 Crearea unui obiect uipanel.

Acestea sunt folosite pentru a grupa alte controale [25]. Dacă *parent* va fi omis, atunci va fi creat un uipanel pentru figura deja prezentă. Dacă nu va fi disponibilă nicio figură, se va crea mai întâi o figură nouă. Dacă *parent* va fi dat, atunci se va creea un uipanel relativ la el. Orice pereche de valori date la *property* vor înlocui valorile inițiale ale obiectului de tip uipanel creat. Valoarea de returnare opțională *hui* reprezintă un graphiscs handle pentru obiectul de tip uipanel. Un exemplu se poate observa în figura 2.17:

```
## create figure and panel on it
f = figure;
p = uipanel ("title", "Panel Title", "position", [.25 .25 .5 .5]);

## add two buttons to the panel
b1 = uicontrol ("parent", p, "string", "A Button", ...
                "position", [18 10 150 36]);
b2 = uicontrol ("parent", p, "string", "Another Button", ...
                "position",[18 60 150 36]);
```

Figura 2.17. secțiune de cod în care se folosește obiectul uipanel [25].

În figura 2.18 avem crearea unui obiect uibuttongroup și returnarea unui handle la acesta:

```
hui = uibuttongroup (property, value, ...)
hui = uibuttongroup (parent, property, value, ...)
uibuttongroup (h)
```

Figura 2.18 Crearea unui obiect uibuttongroup.

Acestea sunt folosite pentru gruparea altor obiecte uicontrol [25]. Dacă *parent* va fi omis, atunci se va crea un uibuttongroup pentru figura deja prezentă. Dacă nu va fi disponibilă nicio figură, va fi creată mai întâi o figură nouă. Dacă *parent* va fi dat, atunci se va creea un uibuttongroup

relativ la acesta. Orice pereche de valori ce vor fi date la *property* vor înlocui e valorile inițiale ale obiectului de tip uibuttongroup. Un exemplu se poate vedea în figura 2.19

```
## Create figure and panel on it
f = figure;
## Create a button group
gp = uibuttongroup (f, "Position", [ 0 0.5 1 1])
## Create a buttons in the group
b1 = uicontrol (gp, "style", "radiobutton", ...
                "string", "Choice 1", ...
                "Position", [ 10 150 100 50 ]);
b2 = uicontrol (gp, "style", "radiobutton", ...
                "string", "Choice 2", ...
                "Position", [ 10 50 100 30 ]);
## Create a button not in the group
b3 = uicontrol (f, "style", "radiobutton", ...
                "string", "Not in the group", ...
                "Position", [ 10 50 100 50 ]);
```

Figura 2.19. secțiune de cod în care se folosește uibuttongroup-ul [25].

În figura 2.20 avem crearea unui obiect uicontrol și returnarea unui handle la acesta

```
hui = uicontrol (property, value, ...)
hui = uicontrol (parent, property, value, ...)
uicontrol (h)
```

Figura 2.20. crearea unui obiect uicontrol.

Acesta este utilizat pentru a creearea comenzilor interactive simple, cum ar fi casete de selectare, butoane, label-ri etc. [25]. Dacă *parent* va fi omis, atunci se va crea un uicontrol pentru figura deja prezentă. Dacă nu va fi disponibilă nici o figură, se va crea mai întâi o figură nouă. Dacă *parent* va fi dat, atunci se va creea un uicontrol relativ la acesta. Orice pereche de valori ce vor fi date la *property* vor înlocui valorile inițiale ale obiectului de tip uicontrol. Tipurile de uicontrol sunt specificate de parametrul *style*. Dacă nu va fi dată nici o valoare pentru *style*, se va crea un buton simplu. Urmatoarele sunt stiluri valide pentru un obiect uicontrol:

- **"checkbox"** – Creează o casetă de selecție care permite utilizatorului să aleagă dintre pornit și dezactivat.
- **"edit"** - Creează un control de editare care permite utilizatorului să introducă una sau mai multe rânduri de text.
- **"listbox"** - Creează un control listbox care afișează o listă de articole și permite utilizatorului să selecteze unul sau mai multe elemente.

- **"popupmenu"** - Creează un câmp de meniu pop-up care afișează o listă de opțiuni care pot fi selectate atunci când utilizatorul face clic pe câmp.
- **"pushbutton"** - Creează un buton care face posibil pentru utilizator să apese pentru a declanșa o acțiune.
- **"radiobutton"** - Creează un buton radio de control utilizat pentru introducerea reciprocă exclusiv într-un grup de comenzi cu buton radio.
- **"slider"** - Creează un control de tip glisor care permite utilizatorului să selecteze dintr-o gamă de valori trăgând mânerul respectiv al slider-ului.
- **"text"** - Creează un control text static pentru a face posibilă afișarea a una sau mai multe rânduri de text.
- **"togglebutton"** - Creează un buton de comutare care apare ca un buton de apăsare, dar permite utilizatorului să aleagă între două stări.

Un exemplu se poate vedea în figura 2.21

```
## Create figure and panel on it
f = figure;
## Create a button (default style)
b1 = uicontrol (f, "string", "A Button", ...
                   "position", [10 10 150 40]);
## Create an edit control
e1 = uicontrol (f, "style", "edit", "string", "editable text", ...
                   "position", [10 60 300 40]);
## Create a checkbox
c1 = uicontrol (f, "style", "checkbox", "string", "a checkbox", ...
                   "position", [10 120 150 40]);
```

Figura 2.21. secțiune de cod în care se folosește obiectul uicontrol [25].

În figura 2.22 avem crearea unui obiect uitable și returnarea unui handle la acesta:

```
hui = uitable (property, value, ...)
hui = uitable (parent, property, value, ...)
```

Figura 2.22. crearea unui obiect uitable.

Acest obiect este folosit pentru a face posibilă afișarea de tabele de date într-o fereastră cu figuri [25]. Dacă *parent* va fi omis, atunci se va crea un uitable pentru figura deja prezentă. Dacă nu va fi disponibilă nicio figură, se va creată mai întâi o figură nouă. Dacă *parent* va fi dat, atunci se creează un uitable relativ la acesta. Orice pereche de valori care vor fi date la *property* vor înlocui valorile inițiale ale obiectului uitable.

În figura 2.23 avem crearea unui obiect uimenu și returnarea unui handle la acesta:

```
hui = uimenu (property, value, ...)
hui = uimenu (h, property, value, ...)
```

Figura 2.23. crearea unui obiect uimenu.

Dacă *h* va fi omis, atunci se va crea un meniu de tip top-level pentru figura prezentă. Dacă va fi dat *h*, atunci se va crea un submeniu relativ acesta. Aceste obiecte de tip uimenu au proprietățile următoarele:

- **"accelerator"** - Un șir care conține combinația de taste cu CTRL pentru efectuarea intrării de meniu (ca exemplu putem lua „x” pentru CTRL + x).
- **"callback"** – Acesta este apelat când acest element de meniu este executat. Poate fi o secvență de funcții (de exemplu, „myfun”), un descriptor de funcție (de exemplu, @myfun) sau o matrice de celule care conține funcția descriptor și argumentele funcției de apel invers (de exemplu: {@myfun, arg1, arg2}).
- **"checked"** - Poate fi setat la „pornit” sau „dezactivat”. Așează un marcaj pe acest element de meniu..
- **"enable"** - Poate fi setat la „pornit” sau „dezactivat”. Dacă este dezactivat, elementul de meniu nu poate fi selectat și este de culoare gri.
- **"foregroundcolor"** - O valoare de setare a culorii care definește culoarea textului pentru acest element de meniu.
- **"label"** - Un șir care conține eticheta elementului de meniu. Simbolul „&” poate fi folosit pentru a evidenția caracterul accelerator.
- **"position"** - O valoare scalară care va conține poziția relativă a meniului. Intrarea cu cea mai mică valoare este prima din stânga sau de sus.
- **"separator"** - Poate fi setat „pornit” sau „dezactivat”. Dacă va fi activat, acesta va trage o linie de despărțire deasupra poziției curente.

Un exemplu se poate vedea în figura 2.24:

```
f = uimenu ("label", "&File", "accelerator", "f");
e = uimenu ("label", "&Edit", "accelerator", "e");
uimenu (f, "label", "Close", "accelerator", "q", ...
           "callback", "close (gcf)");
uimenu (e, "label", "Toggle &Grid", "accelerator", "g", ...
           "callback", "grid (gca)");
```

Figura 2.24. secțiune de cod în care se folosește obiectul uimenu [25].

În figura 2.25 avem crearea unui obiect uicontextmenu și returnarea unui handle la acesta:

```
hui = uicontextmenu (property, value, ...)
hui = uicontextmenu (h, property, value, ...)
```

Figura 2.25. crearea unui obiect uicontextmenu.

Dacă *h* va fi omis, atunci se va crea un uicontextmenu pentru figura deja prezentă. Dacă nu va fi disponibilă nici o figură, se va crea mai întâi o figură nouă. Dacă *h* va fi dat, atunci se va creea un uicontextmenu relativ la acesta. Orice pereche de valori ce vor fi date la *property* vor înlocui valorile inițiale ale obiectului creat. Un exemplu se poate vedea în figura 2.26:

```
## create figure and uicontextmenu
f = figure ();
c = uicontextmenu (f);

## create menus in the context menu
m1 = uimenu ("parent", c, "label", "Menu item 1", ...
             "callback", "disp('menu item 1')");
m2 = uimenu ("parent", c, "label", "Menu item 2", ...
             "callback", "disp('menu item 2')");

## set the context menu for the figure
set (f, "uicontextmenu", c);
```

Figura 2.26. secțiune de cod în care se folosește obiectul uicontextmenu [25].

În figura 2.27 avem crearea unui obiect uitoolbar:

```
uitoolbar (property, value, ...)
uitoolbar (parent, property, value, ...)
hui = uitoolbar (...)
```

Figura 2.27. crearea unui obiect uitoolbar.

Acesta va afișa butoanele uitoggletool și uipushtool [25]. Dacă *parent* va fi omis, atunci se va crea un uitoolbar pentru figura deja prezentă. Dacă nu va fi disponibilă nici o figură, se va crea mai întâi o figură nouă. Dacă *parent* va fi dat, atunci se va creea un uitoolbar relativ aceasta. Orice pereche de valori ce vor fi date la *property* vor înlocui valorile inițiale ale obiectului creat. Valoarea opțională de returnare hui este reprezentată de un ”graphics handle” pentru acest obiect. Un exemplu se poate vedea în figura 2.28:

```
% create figure without a default toolbar
f = figure ("toolbar", "none");
% create empty toolbar
t = uitoolbar (f);
```

Figura 2.28. secțiune de cod în care se folosește obiectul uitoolbar [25].

În figura 2.29 avem crearea unui obiect uipushtool:

```
uipushtool (property, value, ...)
uipushtool (parent, property, value, ...)
hui = uipushtool (...)
```

Figura 2.29. crearea unui obiect uipushtool.

Uipushtool e un buton care apare pe un toolbar [25]. Butonul va fi creat cu o margine care este afișă atunci când utilizatorul va trece cu mouse-ul peste buton. O imagine ar putea fi setată dacă se folosește o proprietate numită *cdata*. Dacă *parent* va fi omis, atunci se va creat un uipushtool pentru figura deja prezentă. Dacă nu va fi disponibilă nicio figură, atunci se va crea mai întâi o figură nouă. Dacă *parent* va fi dat, atunci se va creea un uipushtool relativ acestuia. Orice pereche de valori ce vor fi date la *property* vor înlocui valorile inițiale ale obiectului uipushtool. Valoarea opțională de returnare *hui* reprezintă un ”graphics handle” pentru acest obiect creat de noi. Un exemplu se poate vedea în figura 2.30:

```
% create figure without a default toolbar
f = figure ("toolbar", "none");
% create empty toolbar
t = uitoolbar (f);
% create a 19x19x3 black square
img=zeros(19,19,3);
% add pushtool button to toolbar
b = uipushtool (t, "cdata", img);
```

Figura 2.30. secțiune de cod în care se folosește obiectul uipushtool [25].

În figura 2.31 avem crearea unui obiect uitoggletool:

```
uitoggletool (property, value, ...)
uitoggletool (parent, property, value, ...)
hui = uitoggletool (...)
```

Figura 2.31. crearea unui obiect uitoggletool.

Acesta este un buton care va apărea pe un toolbar [25]. Butonul va fi creat cu o margine care este afișată atunci când utilizatorul o să treacă cu mouse-ul peste buton. O imagine ar poate fi setată folosind o proprietate numită *cdata*. Dacă *parent* va fi omis, atunci se va crea un uitoggletool pentru figura deja prezentă. Dacă nu va fi disponibilă nici o figură, se va crea mai întâi o figură nouă. Dacă *parent* va fi dat, atunci se va creea un uitoggletool relativ la acesta. Orice pereche de valori ce vor fi date la *property* vor înlocui valorile implicite ale obiectului uitoogletool. Valoarea opțională de returnare hui reprezintă un ”graphics handle” pentru obiectul acest creat de noi. Un exemplu se poate vedea în figura 2.32:

```
% create figure without a default toolbar
f = figure ("toolbar", "none");
% create empty toolbar
t = uitoolbar (f);
% create a 19x19x3 black square
img=zeros(19,19,3);
% add uitoggletool button to toolbar
b = uitoggletool (t, "cdata", img);
```

Figura 2.32. secțiune de cod în care se folosește obiectul uitoogletool [25].

În figura 2.33 putem vedea cum ar arăta toate acestea împreună, puse intr-un panou GUI:

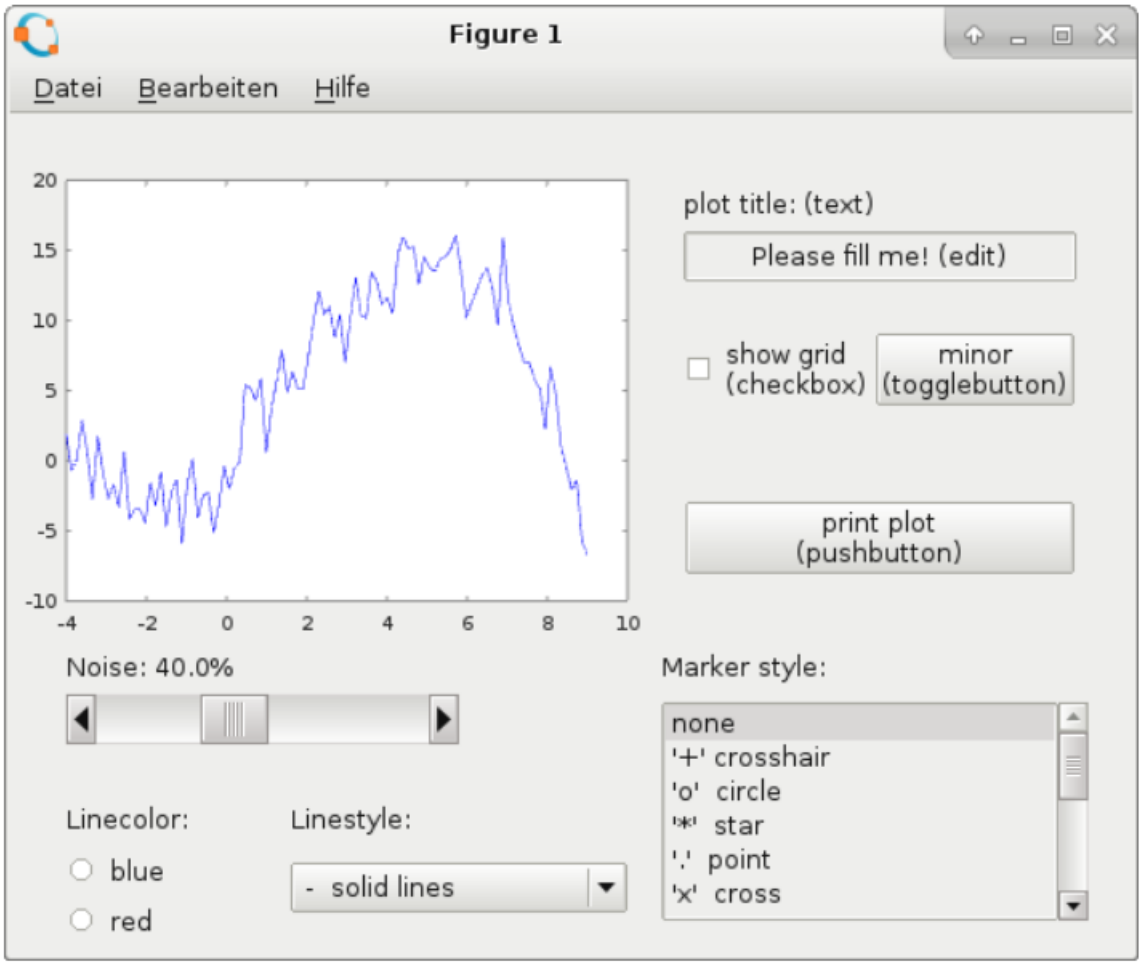


Figura 2.33. panou GUI divers creeat în Octave. [26].

3. Rezultate și discuții

Scopul acestei lucrări este de a realiza un program pentru controlul de la distanță a unei surse de alimentare. Acest program trebuie să facă conexiunea la portul cu care este conectată sursa, și să ofere o interfață grafică ușor de folosit, care să aibă posibilitatea de a transmite si de a citi date către și de la sursă prin intermediul unor butoane.

Sursa de alimentare cu care lucrăm este una de tip DC GPD-X303S. Ea este ușoară, reglabilă și multifuncțională. GPD-2303S are 2 ieșiri de tensiune reglabile independente [27]. GPD-3303S are trei ieșiri independente: două cu niveluri de tensiune reglabile și una cu nivel fix selectabil de la 2,5 V, 3,3 V și 5 V [27]. GPD 4303S are patru ieșiri independente de tensiune care sunt toate complet reglabile [27]. Seria GPD-X303S poate fi utilizată pentru circuite logice în care sunt necesare diverse tensiuni sau curenți de ieșire, și multe altele. Aceasta poate primi comenzi si poate transmite date printr-un port serial. În figura 3.0 putem vedea lista comenziilor ce pot fi trimise către sursă.

ISET<X>:<NR2>	Sets the output current.
ISET<X>?	Returns the output current setting.
VSET<X>:<NR2>	Sets the output voltage.
VSET<X>?	Returns the output voltage setting.
IOUT<X>?	Returns the actual output current.
VOUT<X>?	Returns the actual output voltage.
TRACK<NR1>	Selects the operation mode.
BEEP<BOOLEAN>	Turn on or off the beep.
OUT<BOOLEAN>	Turn on or off the output.
STATUS?	Returns the GPD-X303S status.
*IDN?	Returns the GPD-X303S identification.
RCL<NR1>	Recalls a panel setting.
SAV<NR1>	Saves the panel setting.
HELP?	Shows the command list.
ERR?	Returns the instrument error messages.
BAUD<NR1>	Sets the baud rate.
LOCAL	Returns the instrument to local mode.

Figura 3.0 Lista de comenzi acceptate de sursă [27].

În prima parte din program se face conexiunea cu portul serial cu care este conectată sursa de tennsiune:

```
clear;
clc;
pkg load instrument-control
  comm='COM3';
  bauds=9600;

 global s1 = serial(comm,bauds);
if (exist("serial") == 3)
  disp("connected")
else
  disp("not connected")
endif
fopen(s1);
```

Figura 3.1. secvență de cod în care se face legătura cu portul serial.

Secvența din figura 3.1 începe prin a curăța variabilele vechi si ce a fost scris inainte pe linia de comandă. După aceea, comanda ”pkg load instrument-control” accesează funcțiile necesare pentru a face conexiunea la portul cu care se lucrează. Se inițializează numele si viteza de comunicare a portului, apoi se face conexiunea. în cazul în care conexiunea a reusit, se va afisa ”connected”, iar în caz contrar, se va afișa ”not connected”. Ultima instrucțiune deschide portul.

A doua parte a programului este reprezentată de creerea unui panou, în care ulterior să fie introduse butoanele și comenzile necesare.

```
global fi;
fi = figure();
ax = axes ("xlim", [-5 5], "ylim" , [-5 5]);
li = line("xdata",[-5 5 5 -5 -5],"ydata",[-5 -5 5 5 -5],...
             "linewidth",2,"color" ,"black");
drawnow();
```

Figura 3.2. secvență din cod în care se creează un panou GUI.

În codul din figura 3.2 prima oară se intruduce variabila globală *fi* care este figura în care se atașează toate desenele si butoanele din program. Se creează un sisteam de axe în care este trasată o linie de culoare neagră ce reprezintă panoul în care sunt așezate toate butoanele de control. În figura 3.3 se observă cum arată GUI-ul creat.

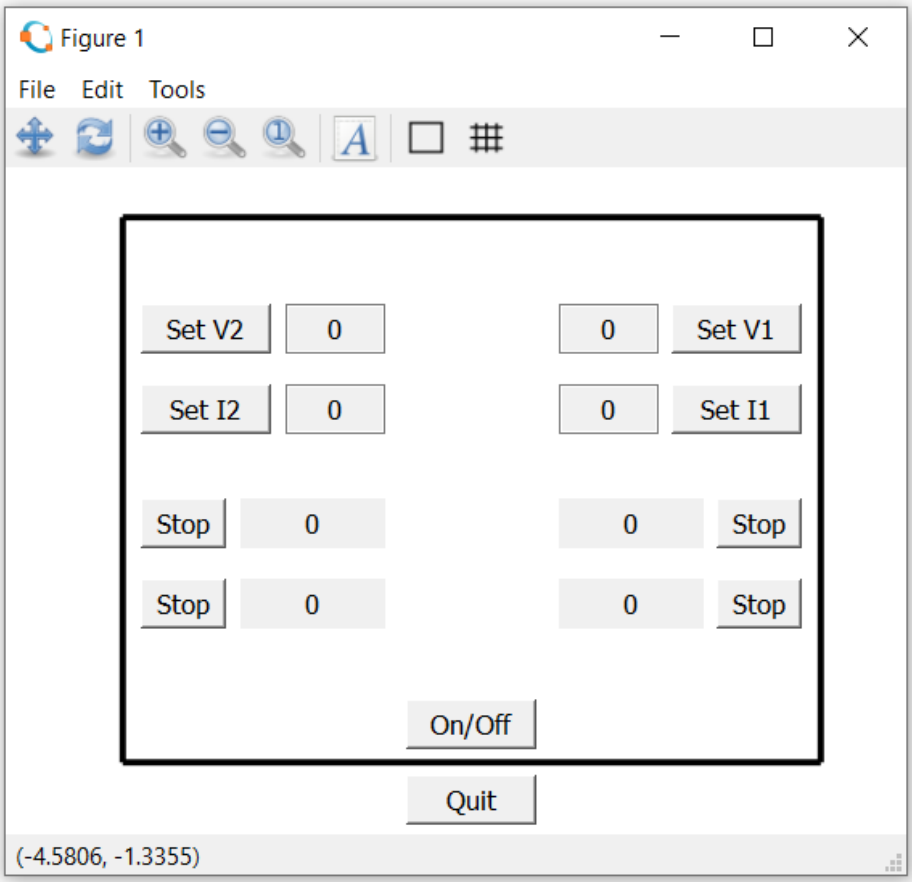


Figura 3.3. Un panou GUI cu 10 butoane, 4 casete de text și 4 label-uri creat în Octave.

Fiecare buton are câte un text inserat pe ele care le reprezintă (*Set V2, Set I2*, etc), iar casetele de text sunt inscripționate inițial cu 0, ca ulterior, textul din interior să poată fi modificat cu valoarea de care este nevoie.

Cel mai simplu buton din această interfață este butonul ”Quit”. Când acesta este apăsat, panoul GUI se va inchide. Codul care stă la baza funcționării acestuia este prezentat în figura 3.4.

```
43 global h;
44 h = 1;

46 bt3 = uicontrol(fi,"style","pushbutton","units","normalized","string","Quit",...
47                  "callback","bt3_pressed","position",[0.445,0.017,0.145,0.075])

78 function bt3_pressed
79  global h;
80  h=0;
81    global fi;
82    close(fi);
83 endfunction
84
```

Figura 3.4. Secvență de cod pentru creearea si funționarea butonului ”Quit”.

Începem prin a declara global o variabilă numită *h*, la care îi dăm valoarea 1. Variabila care creează butonul este numită *bt3*, iar funcția care se apelează când acesta e apăsat este *bt3_pressed*. În interiorul funcției *bt3_pressed* începem prin declararea globala în interiorul funcției, a variabilei *h* (pentru a folosi o variabilă globală în interiorul unei funcții, ea trebuie declarată și în acea funcție), la care îi este dată valoarea 0. Acesta este pentru a opri ciclul infinit din structura *while* care va fi discutată mai jos. Apoi se declară global variabila *fi*, care este figura, iar comanda *close(fi)* o va închide.

Butonul ”On/Off” este probabil cel mai important buton, deoarece el cuplează sau decuplează sursa. În figura 3.5 se poate vedea codul prin care funcționează acesta.

```
21  global on = 1;
25  global n;
49  bton = uicontrol(fi,"style","pushbutton","units","normalized","string","On/Off",.
50                   "callback","bton_pressed","position",[0.445,0.13,0.145,0.075])
141 function bton_pressed
142   global on;
143   global n;
144   global s1;
145   if (on == 0)
146     n = srl_write(s1, "OUT1\n\r");
147     on = 1;
148   else
149     n = srl_write(s1, "OUT0\n\r");
150     on = 0;
151   endif
152 endfunction
153
```

Figura 3.5. Secvență de cod pentru creearea si funționarea butonului ”On/Off”.

Se declară două variabile globale, *on*(a cărei valoare este setată să fie 1) și *n*. variabila *n* este folosită pentru a utiliza comenziile de scriere pe serial, iar variabila *on* e folosita în funcția *bton_pressed* pentru a vedea în ce stare se află sursa. Variabila în care se creează butonul este numită *bton*, care atunci când e apăsat, face apel la funcția *bton_pressed*. Această funcție funcționează în felul următor: sunt declarate cele 3 variabile globale cu care se lucrează, *on, n,* și *s1,* apoi avem o structură if-else care în cazul în care variabila on este 0, se va transmite pe serial ”OUT1/n/r”, care va porni output-el și va schimba valoarea lui on la 1, ca în caul în care apelăm iar funcția să se știe ca output-ul este pornit. Iar daca valoarea lui on era deja 1, se execută blocul de comenzi a lui *else,* în care se va scrie pe serial ”OUT0/n/r”, și va schimba valoarea lui on în 0.

Urmatoarul lucru de care o să discutăm o să fie caseta de text din dreapta butonului ”Set V2”. În secvența următoare din figura 3.6 este prezentat codul pentru creerea acestei casete respectiv ce se întâmplă când modificăm codul din interiorul acesteia.

```
26  global t5;
45  global c5;

64  t5 = uicontrol(fi,"style","edit","units","normalized","string","0",...
65                     "callback","t5_update","position",[0.31,0.72,0.11,0.075]);

85  function t5_update
86    global c5;
87    global t5;
88    c5 = get(t5, 'String');
89  endfunction
90
```

Figura 3.6. Secvența de cod din spatele casuței de text din dreapta butonului ”Set V2”.

Declarăm două variabile globale, variabila *t5* care este caseta în sine, și o variabila c5. Funcția *t5_update* este apelată atunci când se modifica textul din interiorul casetei. Aceasta funcționează în felul următor: sunt declarate cele două variabile globale, *t5* și c5, iar comanda *c5 = get(t5, ‘string’)* îi atribuie variabilei c5 valoarea string-ului care a fost inserat în casetă

Butonul din stânga acestei casetei de text, intitulat ”Set V2”, Este folosit pentru a seta tensiunea din canalul 2 a sursei cu valoarea care a fost inserată în caseta respectivă. Codul prin care acest buton funcționează este prezentat în figura 3.7.

```
61  bt5 = uicontrol(fi,"style","pushbutton","units","normalized","string","Set V2",...
62                     "callback","bt5_pressed","position",[0.15,0.72,0.145,0.075]);

133 function bt5_pressed
134     global s1;
135     global n;
136     global c5;
137     c=strcat("VSET2:",c5,"\n\r");
138     n = srl_write(s1,c);
139 endfunction
140
```

Figura 3.7. Secvență de cod pentru creerea si funcționarea butonului ”Set V2”.

Acest buton l-am denumit *bt5* și cand el este apăsat, va face apel la funcția *bt5_pressed.* În funcție sunt declarate cele 3 variabile globale care se utilizează: *c5, s1, n.* unde c5 reprezintă stringul optinut din textul introdus în caseta din dreapta lui, s1 reprezintă portul serial și n este o variabilă folosită pentru a scrie pe acel port. Apoi intr-o variabilă locală din funcție, *c,* prin comanda *c = strcat(”VSET2:”,c5,”\n\r”)* îi este atribuită valoarea stringului obținut din concatenarea a 3 stringuri diferite, *”VSET1*:”,*c5,* și *”\n\r”.* Această sintaxă se folosește pentru a da comanda citibila de sursa programabilă. de exemplu, pentru seta I1 la 0,5 amperi, comanda ar trebui să arate în felul următor: ISET1:0.5. iar *”\n\r”* e folosit pentru ca dupa scrierea comenzii pe serial să se treaca la o linie noua de la început. La finalul funcției, comanda *n = srl_write(s1, c);* scrie pe serialul s1, stringul obținut în variabila c, discutată mai sus, astfel setând în V2, valoarea scrisă în caseta din dreapta acestuia.

Analog, Butoanele ”SETI2”, ”SETV1” și ”SETI1”, respectiv casetele de text de lângă fiecare funcționează pe același principiu. În caseta respectiva se trece valoarea dorită și când se apasă butonul de lângă aceasta, acea valoare va fi setată pentru parametrul pe care dorim să îl modificăm. Secvențele de cod pentru aceste se pot vedea în figurile 3.8 – 3.13.

```
22  global c4;
27  global t4;

67  t4 = uicontrol(fi,"style","edit","units","normalized","string","0",...
68                     "callback","t4_update","position",[0.31,0.6,0.11,0.075]);

91  function t4_update
92    global c4;
93    global t4;
94    c4 = get(t4, 'String');
95  endfunction
96
```

Figura 3.8. secvență de cod pentru creeare și funcționarea casetei de text de lăngă butonul ”SET I2”.

```
52  bt2 = uicontrol(fi,"style","pushbutton","units","normalized","string","Set I2",..
53                     "callback","bt2_pressed","position",[0.15,0.6,0.145,0.075]);

109 function bt2_pressed
110      global s1;
111     global n;
112     global c4;
113     c=strcat("ISET2:",c4,"\n\r");
114     n = srl_write(s1,c);
115 endfunction
116
```

Figura 3.9. secvență de cod pentru creeare și funcționarea butonului ”SET I2”.

```
29  global t1;
24  global c1;

70  t1 = uicontrol(fi,"style","edit","units","normalized","string","0",...
71                     "callback","t1_update","position",[0.615,0.72,0.11,0.075]);

103 function t1_update
104   global c1;
105   global t1;
106   c1 = get(t1, 'String');
107 endfunction
108
```

Figura 3.10. secvență de cod pentru creeare și funcționarea casetei de text de lăngă butonul ”SET V1”.

```
58  bt4 = uicontrol(fi,"style","pushbutton","units","normalized","string","Set V1",...
59                  "callback","bt4_pressed","position",[0.74,0.72,0.145,0.075]);

125 function bt4_pressed
126      global s1;
127     global n;
128     global c1;
129     c=strcat("VSET1:",c1,"\n\r");
130     n = srl_write(s1,c);
131 endfunction
132
```

Figura 3.11. secvență de cod pentru creeare și funcționarea butonului ”SET V1”.

```
23  global c2;
28  global t2;
73  t2 = uicontrol(fi,"style","edit","units","normalized","string","0",...
74                  "callback","t2_update","position",[0.615,0.6,0.11,0.075]);

97  function t2_update
98    global c2;
99    global t2;
100   c2 = get(t2, 'String');
101 endfunction
102
```

Figura 3.12. secvență de cod pentru creeare și funcționarea casetei de text de lângă butonul ”SET I1”.

```
55  bt1 = uicontrol(fi,"style","pushbutton","units","normalized","string","Set I1",...
56                  "callback","bt1_pressed","position",[0.74,0.6,0.145,0.075]);

117 function bt1_pressed
118     global s1;
119     global n;
120     global c2;
121     c = strcat("ISET1:",c2,"\n\r");
122     n = srl_write(s1,c);
123 endfunction
124
```

Figura 3.13. secvență de cod pentru creeare și funcționarea butonului ”SET I1”.

Butonul ”Stop” din partea din dreapta sus respectiv caseta de lângă el, sunt pentru citirea în timp real a valorii lui V2 (tensiunea de pe canalul 2 al sursei). Când butonul ”Stop” este apăsat, citirea continuă se oprește rămânând ultima valoare citită în label. Dacă e apăsat din nou, citirea în timp real începe iarăși. Secvența de cod pentru creearea acestora se află în figura 3.14.

```
34  global tv2;

41  global xv2;
42  xv2 = 1;

202 btv2 = uicontrol(fi,"style","pushbutton","units","normalized","string","Stop",...
203                  "callback","btv2_pressed","position",[0.15,0.43,0.095,0.075]);
204 function btv2_pressed
205   global xv2;
206   if (xv2 == 1);
207     xv2 = 0;
208   else
209     xv2 = 1;
210   endif
211 endfunction
212 tv2 = uicontrol(fi,"style","text","units","normalized","string","0",...
213                  "callback","tv2_update","position",[0.26,0.43,0.16,0.075]);
```

Figura 3.14. Secvență din cod pentru creearea label-ului și butonului ”Stop” folosite la ctitirea lui V2.

Modul concret de funționare al acestui principiu este explicat mai jos, în structura *while*. În secvența de mai sus, tot ce putem observa este creerea celor 2 variabile globale, *tv2*(care reprezintă label-ul) și *xv2*. Creearea butonului ”Stop” în variabila *btv2*, funcția la care face apel acest buton, *btv2_pressed*, care doar schimbă variabila *xv2* din 0 în 1 sau invers prin intermediul unei structuri if-else, și creerea label-ului *tv2*.

Analog, Butoanele ”Stop” pentru I2, V1 și I1 respectiv label-urile de lângă fiecare funcționează pe același principiu. Secvențele de cod se pot vedea în figurile 3.15 – 3.17.

```
33  global ti2;

37  global xi2;
38  xi2 = 1;

187 bti2 = uicontrol(fi,"style","pushbutton","units","normalized","string","Stop",...
188                  "callback","bti2_pressed","position",[0.15,0.31,0.095,0.075]);
189 function bti2_pressed
190   global xi2;
191   if (xi2 == 1);
192     xi2 = 0;
193   else
194     xi2 = 1;
195   endif
196 endfunction
197 ti2 = uicontrol(fi,"style","text","units","normalized","string","0",...
198                  "callback","ti2_update","position",[0.26,0.31,0.16,0.075]);
```

Figura 3.15. Secvență din cod pentru creearea label-ului și butonului ”Stop” folosite la ctitirea lui I2.

```
32  global tv1;

39  global xv1;
40  xv1 = 1;

172 btv1 = uicontrol(fi,"style","pushbutton","units","normalized","string","Stop",...
173                     "callback","btv1_pressed","position",[0.79,0.43,0.095,0.075]);
174 function btv1_pressed
175   global xv1;
176   if (xv1 == 1);
177     xv1 = 0;
178   else
179     xv1 = 1;
180   endif
181 endfunction
182 tv1 = uicontrol(fi,"style","text","units","normalized","string","0",...
183                     "callback","tv1_update","position",[0.615,0.43,0.16,0.075]);
```

Figura 3.16. Secvență din cod pentru creearea label-ului și butonului ”Stop” folosite la ctitirea lui V1.

```
30  global ti1;

35  global xi1;
36  xi1 = 1;

155 bti1 = uicontrol(fi,"style","pushbutton","units","normalized","string","Stop",...
156                     "callback","bti1_pressed","position",[0.79,0.31,0.095,0.075]);
157 function bti1_pressed
158   global xi1;
159   if (xi1 == 1);
160     xi1 = 0;
161   else
162     xi1 = 1;
163   endif
164 endfunction
165 ti1 = uicontrol(fi,"style","text","units","normalized","string","0",...
166                     "callback","ti1_update","position",[0.615,0.31,0.16,0.075]);
```

Figura 3.17. Secvență din cod pentru creearea label-ului și butonului ”Stop” folosite la ctitirea lui I1.

În figura 3.18 se află secvența de cod care conține *while*-ul ce stă la baza funcționării acestor butoane și label-uri.

```
while (h==1)
     if (xi1 == 1)
     n = srl_write(s1, "IOUT1?\n\r");
       pause (0.1);
   [rxdata,count] = srl_read(s1, 8);
r = char(rxdata);
r=substr(r, 1, 6);
 z = strcat("I1=",r);
   set(ti1, 'string', z);
   endif
    if (xi2 == 1)
    n = srl_write(s1, "IOUT2?\n\r");
      pause (0.1);
   [rxdata,count] = srl_read(s1, 8);
r = char(rxdata);
r=substr(r, 1, 6);
z = strcat("I2=",r)
    set(ti2, 'string', z);
   endif
   if (xv1 == 1)
  n = srl_write(s1, "VOUT1?\n\r");
     pause (0.1);
 [rxdata,count] = srl_read(s1, 8);
r = char(rxdata);
if (r(2)=='.')
r=substr(r, 1, 6);
z = strcat("V2=",r)
   set(tv1, 'string', z);
 else
   [rxdata,count] = srl_read(s1, 1);
   r=substr(r, 1, 7);
z = strcat("V2=",r)
   set(tv1, 'string', z);
   endif
   endif
   if (xv2 == 1)
    n = srl_write(s1, "VOUT2?\n\r");
     pause (0.1);
   [rxdata,count] = srl_read(s1, 8);
r = char(rxdata);
if (r(2)=='.')
r=substr(r, 1, 6);
z = strcat("V2=",r)
   set(tv2, 'string', z);
 else
   [rxdata,count] = srl_read(s1, 1);
   r=substr(r, 1, 7);
z = strcat("V2=",r)
   set(tv2, 'string', z);
   endif
     endif
     pause (0.1);
endwhile
```

Figura 3.18. Secvență din cod ce conține while-ul folosit pentru citirea în timp real al valorilor lui V2, I2, V1 și I1.

Aceasta este ultima structură din cod, o structură *while* care se execută în mod continuu, până când este apăsat butonul ”Quit”. Condiția inițială a structuri *while*, este daca valoarea lui *h* este egală cu 1, (ceea ce e adevărat, căci această valoare a fost dată lui h la început), iar singura comanda care modifică valoarea lui *h* se află în interioful funcției apelate din apăsarea butonului ”Quit”, care o va modifica la 0, astfel oprind structura *while*.

În structură se gasesc patru *if*-uri principale, fiecare verificând variabila *xi1, xi2, xv1,* sau *xv2,* pentru respectivele I1, I2, V1 sau V2, iar dacă sunt egale cu 1, se va executa blocul de instrucțiuni, care citește valoarea lui I1, I2, V1 sau al lui V2 și o afisează în caseta de text respectivă. Când butonul ”Stop” e apăsat, această valoare se schimbă în 0, și atunci când se execută *while*-ul, nu se va mai executa *if*-ul respectiv, rămânând ultima valoare citită. Odată ce butonul a fost apăsat din nou, blocul de instructiuni din interiorul *if*-ului va fi executat din nou, și citirea în timp real continuă.

În mod concret, funcționează în felul următor: luând primul *if*, el verifică dacă variabila *xi1* este 1, apoi, în caz afirmativ, execută instrucțiunile din interiorul său: comanda *n = srl_write(s1, "IOUT1?\n\r");* scrie ”IOUT1?”, pe serial, care odată citită de sursă, aceasta ne va transmite valoarea lui I1 la momentul respectiv. *pause (0.1);* oprește *while*-ul pentru o zecime de secundă, pentru a da sursei timp să transmită valoarea lui I. *[rxdata,count] = srl_read(s1, 8);* atribuie rezultatul variabilei *rxdata* ce a fost citită în 8 byte-uri, iar *r = char(rxdata);* dă variabilei *r* valoarea din *rxdata* convertită într-una de tip *char*. *r=substr(r, 1, 6);* pastrează doar primele 6 caractere din *r,* deoarece altfel se păstrează și caracterul de new line care dacă e scris descentrează textul din căsuța respectivă. z=*strcat("I1=",r);* îi dă variabilei z valoarea stringului obținut din contopirea ”I1=” și *r*, iar după, *set(ti1, 'string', z);* va seta stringul obținut din *z* în caseta de text respectivă. Pe același principiu funcționează si *if*-ul pentru I2, insă pentru V1 și V2 avem în interior câte un *if* în plus. Acesta este folosit pentru că în cazul tensiuniilor pot exista valori de zece ori mai mari decât ale intensităților, Ceea ce va face ca în cazul citirii să poată apărea un caracter în plus. Tensiunile pot avea 2 caractere înainte de virgulă, și atunci a fost folosit un *if* care să verifice dacă se întamplă acest lucru și să mai citească un caracter, ca să nu rămână caractere necitite și să interfereze cu citirea celorlalte valori.

Concluzii

Scopul acestei lucrări a fost acela de a realiza un program pentru controlul de la distanță a unei surse de alimentare. Folosindu-ne de cunoștințe de bază de electronică, comunicații seriale, surse de curent, surse de tensiune și noțiuni fundamentale de programare, controlul de la distanță a unei surse de alimentare folosind calculatorul este realizabil.

Sursa de alimentare cu care s-a lucrat este de tip GPD-X303S, care prezintă două canale de ieșire, Channel 1 și Channel 2, fiecare având câte o valoare reglabilă pentru tensiune și pentru curent. Prin intermediul unui port serial, sursa poate primi semnale de la un calculator, și de asemenea, ea poate transmite semnale prin acesta. Acest fapt a făcut posibilă scrierea programului, care e făcut în așa fel încât fiecare buton să transmită o anumită comandă către sursă, sau să citească ce se transmite de la sursă către calculator.

Programul de față are 3 funcții: prima funcție a programului este aceea de a seta valori concrete pentru tensiune sau pentru curent, pe canalul 1 sau pe canalul 2, prin intermediul câte unui buton urmat de o casetă de text. În caseta de text se introduce valoarea dorită pentru tensiune sau pentru curent, pe canalul dorit, și se apasă butonul de lângă casetă pentru a seta valoarea respectivă (De exemplu, butonul pentru setarea valorii tensiuni pe canalul doi este intitulat ”Set V2”). Cea de a doua funcție a programului este de a citi de pe portul serial valorile în timp real ale tensiuniilor si intensității curenților de pe cele două canale. Această valoare este inscripționată intr-un label, Iar lângă acesta este un buton numit ”Stop” care oprește citirea în timp real a valoriilor și ramâne inscripționată ultima valoare care a fost citită. Dacă butonul este apăsat din nou, citirea în timp real al valorilor reîncepe. Ultima funcție pe care programul o poate indeplini este aceea de a cupla și decupla releul de la ieșirea sursei, pornind sau oprind alimentarea cu energie electrică a circuitului alimentat de sursă. Această funcție s-a realizat cu un buton denumit ”On/Off”, care, dacă este apăsat, va opri sau va porni ieșirea sursei de alimentare în funcție de starea acesteia: dacă e oprită, butonul ”On/Off” o va porni, iar dacă e pornită, butonul o va opri.

Bibliografie

[1] Consoliver, Earl L. & Mitchell, Grover I. (1920). Automotive Ignition Systems. McGraw-Hill. p. 4.

[2] Robert A. Millikan and E. S. Bishop (1917). Elements of Electricity. American Technical Society. p. 54.

[3] Oliver Heaviside (1894). Electrical Papers. Vol. 1. Macmillan and Co. p. 283. ISBN 978-0-8218-2840-3.

[4] Horowitz, Paul, Hill, Winfield: The Art of Electronics.

[5] https://en.wikipedia.org/wiki/Ohm%27s_law accesat la data de 13.06.2022.

[6] https://slideplayer.gr/slide/14058498/86/images/27/Gruparea+rezistoarelor.jpg accesat la data de 12.06.2022.

[7] Rajendra Prasad (2006). Fundamentals of Electrical Engineering. Prentice-Hall of India. ISBN 978-81-203-2729-0.

[8] https://www.electronics-tutorials.ws/accircuits/impedance.html accesat la data de 12.06.2022.

[9] Sorin Dan Anghel, Bazele electronicii analogice și digitale, Ed. Presa Universitară Clujeană, Cluj-Napoca 2007.

[10] Bryan H. Suits - Electronics for Physicists. An Introduction-Springer Nature (2020).

[11] https://en.wikipedia.org/wiki/Kirchhoff%27s_circuit_laws accesat la data de 11.06.2022.

[12] https://en.wikipedia.org/wiki/Current_source accesat la data de 12.06.2022.

[13] Schaums-Outlines-Jimmie J.-Cathey-Schaums-Outline-of-Electronic-Devices-and-Circuits-Second-Edition-McGraw-Hill-2002.

[14] https://en.wikipedia.org/wiki/Voltage_source accesat la data de 13.06.2022.

[15] Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss, Digital Systems -Principles and Applications, 10th ed.

[16] K. C. A. Smith, R. E. Alley , Electrical circuits: an introduction, Cambridge University Press, 1992 ISBN 0-521-37769-2, pp. 11-13.

[17] http://msabau.xhost.ro/?Fizic%E3:Electrocinetica:Gruparea_generatoarelor accesat la data de 12.06.2022.

[18] https://www.monolithicpower.com/en/switching-power-supply accesat la data de 12.06.2022.

[19] https://www.techwalla.com/articles/what-is-a-com1-port accesat la data de 11.06.2022.

[20] https://ro.wikipedia.org/wiki/Transmisie_de_date_serial%C4%83#/media/Fi%C8%99ier:Parallel_and_Serial_Transmission.gif accesat la data de 14.06.2022.

[21] https://circuitdigest.com/article/rs232-serial-communication-protocol-basics-specifications#:~:text=The%20term%20RS232%20stands%20for,printers%2C%20factory%20automation%20devices%20etc. accesat la data de 11.06.2022.

[22] https://www.lifewire.com/what-is-a-usb-port-818166 accesat la data de 11.06.2022.

[23] https://www.electronics-notes.com/articles/connectivity/usb-universal-serial-bus/protocol-data-transfer.php accesat la data de 12.06.2022.

[24] https://www.lifewire.com/universal-serial-bus-usb-2626039 accesat la data de 11.06.2022.

[25] John W. Eaton, David Bateman, Søren Hauberg, Rik Wehbring: GNU Octave Free Your Numbers.

[26] https://wiki.octave.org/Uicontrols accesat la data de 12.06.2022.

[27] DC Power Supply-GPD-X303S Series-USER MANUAL-GW INSTEK PART NO. 82PD-433S0M01.

DECLARAȚIE PE PROPRIE RĂSPUNDERE

Subsemnatul, Vlad-Ionuț Groșan, declar că Lucrarea de licență intitulată "Program pentru controlul unei surse de alimentare de la distanță" pe care o voi prezenta în cadrul examenului de finalizare a studiilor la Facultatea de Fizică, din cadrul Universității Babeș-Bolyai, în sesiunea iunie 2022, sub îndrumarea Lect. Dr. Ing. Sever Mican, reprezintă o operă personală. Menționez că nu am plagiat o altă lucrare publicată, prezentată public sau un fișier postat pe Internet. Pentru realizarea lucrării am folosit exclusiv bibliografia prezentată și nu am ascuns nici o altă sursă bibliografică sau fișier electronic pe care să le fi folosit la redactarea lucrării.

Prezenta declarație este parte a lucrării și se anexează la aceasta.

Data,

22.06.2022

Vlad-Ionuț Groșan,

Semnătură