

Algoritmi, Organigrame și Pseudocod

Proiectarea echipamentelor radio definite prin program sau virtuale complexe este preponderent o muncă de echipă. Costurile și succesul dezvoltării produselor depinde în mare măsură de capacitatea membrilor echipei de a formula specificații clare și pertinente pentru modulele analogice și informatice componente. Cei specializați în prelucrarea semnalelor radio analogice și numerice nu sînt de regulă și buni cunoscători ai metodelor de programare eficiente pentru diversele dispozitive de calcul (FPGA, DSP, PC) vizate a fi incluse în echipament, precum și invers, bunii programatori nu au suficiente conștiințe de radiotehnică. Toți trebuie însă să poată comunica, iar de cele mai multe ori sensul apropierei este de la radiotehnică spre informatică prin apel la elementele de macroprogramare, *algoritmi*, *organigrame* și *pseudocod*, ușor accesibile înțelegerii.

Un **algoritm** este o secvență logică de instrucțiuni. Algoritmul este prin urmare o parte fundamentală a calculului. Deși pentru multe nevoi comune de calcul putem dispune de algoritmi standardizați, deseori, pentru soluționarea unor probleme specifice, este nevoie de dezvoltarea unor algoritmi dedicați.

În scopul identificării parametrilor necesari proiectării unui algoritm sînt des utilizate două metode intuitive de descriere a modalității de soluționare a problemei, respectiv două unelte foarte utilizate pentru documentarea algoritmilor și anume, **organigrama** (o reprezentare grafică a soluției propuse) și **pseudocodul** (o reprezentare text a soluției propuse). În general, este recunoscut faptul că organigramele sunt eficiente în cazul problemelor de mică anvergură, în timp ce pseudocodul este indicat pentru problemele mari.

Organigrama reprezintă o metodă descriptivă de tip grafic, cu ajutorul cărei, utilizând descrierea prin simboluri a pașilor care trebuie parcurși pentru soluționarea unei probleme date, se sugerează implicit parametrii specifici ai algoritmului dedicat rezolvării problemei respective. Unele dintre cele mai comune simboluri grafice utilizate pentru realizarea organigramelor, în scopul reliefării pașilor unui algoritm, sunt grupate în figura 1. Transmiterea datelor între zonele (simbolurile) unei organigrame este indicată prin linii cu săgeți.

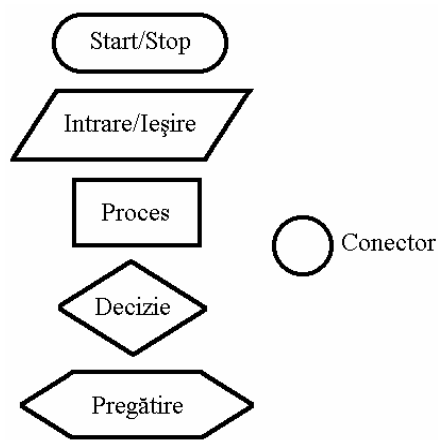


Figura 1

Pseudocodul este o metodă descriptivă, de tip text, cu ajutorul căreia, prin descrierea în cuvinte a pașilor care trebuie parcurși pentru soluționarea unei probleme date, se sugerează implicit parametrii specifici ai algoritmului dedicat rezolvării problemei respective.

Trebuie remarcat faptul că nu este necesar ca descrierea să fie realizată prin utilizarea unui limbaj specific de programare. Textul pseudocodului constă într-o serie de fraze, dispuse în succesiunea logică a raționamentului impus de soluționarea problemei date, nefiind necesară utilizarea unor simboluri proprii organigramelor, deși uneori este utilă trasarea unor săgeți pentru a sugera închiderea unor bucle. Logica unui pseudocod impune uneori chiar și marcarea într-un mod oarecare a instrucțiunilor dintr-o anumită grupă, de regulă prin aceeași aliniere.

Procesul de elaborare a unui program este unul complicat. Mai întâi de toate trebuie înțelese foarte bine specificațiile programului, apoi trebuie făcut un efort de sistematizare a pașilor de urmat și, în final, trebuie scris codul programului. Realizarea mai întâi a pseudocodului programului ușurează foarte mult și scurtează fazele de scriere și testare a modulelor programului. Pseudocodul permite proiectanților să se concentreze numai asupra logicii problemei de rezolvat, nu și asupra detaliilor limbajului de programare.

Întocmirea pseudocodului începe prin deschiderea și completarea unei liste cu principalele sarcini pe care le are de îndeplinit programul. Apoi, se detaliază prin analiză fiecare sarcină în parte și este descompusă în sarcini mai mici, subsecvențe, care pot fi explicitate la rândul lor prin câte o frază scurtă. Se poate ajunge astfel până la o corespondență de unu-la-unu între frazele pseudocodului și viitoarele instrucțiuni ale codului sursă a programului propriu-zis.

Într-un pseudocod nu este necesar să se menționeze declarații de variabile. Este bine însă să se descrie inițializarea variabilelor. Deși se recomandă atribuirea de nume diverselor variabile, nu este absolut necesar ca aceste nume să semnifice rolul variabilei (așa cum se practică în limbajele de programare curente). Pe global, nu trebuie scăpat nici un moment din vedere că scopul pseudocodului este de a înlesni scrierea codului și, prin urmare, în pseudocod trebuie adăugate suficient de multe detalii. În lumea programatorilor profesioniști, cei care scriu pseudocodul sunt rareori aceiași cu cei care ulterior scriu codul programului. De fapt, cel mai frecvent, în momentul scrierii pseudocodului nu se știe nici măcar în ce limbaj va fi scris codul.

Iată câteva recomandări:

- Declararea acțiunilor să se facă prin fraze cât mai simple.
- Textul prin care se descrie efectuarea unei operații să înceapă cu un verb.
- Fiecare instrucțiune să fie scrisă pe un rând separat.
- Să se utilizeze cuvinte cheie și tehnica de grupare a instrucțiunilor prin aliniere.
- Fiecare grup de instrucțiuni să fie scris ordonat de sus în jos, și să aibă un singur punct de intrare și unul de ieșire.
- Declarațiile (instrucțiunile) pot fi grupate în module și fiecărui modul să i se dea un nume.
- La intrările de date să se folosească verbele:
 READ CITEȘTE
 GET PREIA
- La predarea rezultatelor să se folosească verbele:

PRINT	TIPĂREȘTE
WRITE	SCRIE
PUT	PUNE
OUTPUT	IEȘIRE
DISPLAY	AFIȘEAZĂ

- Pentru indicarea operațiilor matematice folosiți verbe:

ADD	ADUNĂ
SUBTRACT	SCADE
MULTIPLY	ÎNMULȚEȘTE
DIVIDE	ÎMPARTE
COMPUTE	
CALCULATE	CALCULEAZĂ

- În cazurile de inițializare a variabilelor, asignarea unei valori ca rezultat al unei prelucrări, reținerea unei informații pentru utilizare ulterioară, utilizați verbe:

INITIALIZE	INIȚIALIZEAZĂ
SET	PUNE
SAVE	SALVEAZĂ
STORE	MEMOREAZĂ

Fiecare persoană are un stil propriu de exprimare, lucru valabil și în cazul pseudocodului. Pseudocodul este descrierea soluționării unei probleme, care nu necesită respectarea unui limbaj riguros întrucât este destinată informării factorului uman, nefiind o instrucțiune adresată calculatorului. Deși nu s-a instituit încă un așa numit "standard industrial" în domeniu, este recomandat ca o anumită colectivitate să urmeze aceleași reguli în elaborarea pseudocodului pentru a ușura dezvoltarea lui în cadrul grupului respectiv.

Partea "structurată" a pseudocodului este formată dintr-un grup de șase construcții (elemente) de programare structurată, respectiv:

- **SEQUENCE** (secvență) este o progresie liniară în care o sarcină este efectuată secvențial după o alta. Controlul secvențial este indicat prin scrierea acțiunilor una după alta, fiecare acțiune pe o linie de text proprie ei, toate frazele descriind acțiunile avînd aceeași aliniere. Acțiunile se efectuează înălțuit, în ordinea în care sînt scrise, de sus-în-jos.
- **WHILE** (în timp ce / cît timp) este o buclă (o repetare) cu un test condițional simplu la început. Construcția WHILE este utilizată pentru a specifica o execuție în buclă, după efectuarea prealabilă a unui test. Începutul buclei este indicat prin cuvântul cheie WHILE, iar sfârșitul ei prin cuvântul cheie ENDWHILE. Ele se pot înlocui cu respectivele echivalente românești: ÎN TIMP CE sau CÂT TIMP, și SFÂRȘIT ÎN TIMP CE sau SFÂRȘIT CÂT TIMP.

WHILE <i>condiție</i>	CÎT TIMP <i>condiție</i>
<i>secvență</i>	<i>secvență</i>
ENDWHILE	SFÎRȘIT CÎT TIMP

În buclă se intră numai dacă *condiția* este adevărată. *Secvența* este executată la fiecare iterație. După executarea *secvenței* într-o iterație se evaluează din nou

condiția și bucla este reluată (prin parcurgerea din nou a *secvenței*) atât timp cât aceasta rămâne adevărată.

- **IF-THEN-ELSE** (dacă-atunci-altfel) este o decizie (selecție) în care se face o alegere între două cursuri (continuări) alternative ale acțiunii. O alegere binară într-o condiție Bool-eană este indicată prin utilizarea a patru cuvinte cheie: IF, THEN, ELSE și ENDIF. Ele se pot înlocui cu respectivele echivalente românești: DACĂ, ATUNCI, ALTFEL sau ÎN CAZ CONTRAR și SFÎRȘIT DACĂ. Forma generală de utilizare a acestor cuvinte cheie este:

IF <i>condiție</i> THEN	DACĂ <i>condiție</i> ATUNCI
<i>secvența 1</i>	<i>secvența 1</i>
ELSE	ÎN CAZ CONTRAR
<i>secvența 2</i>	<i>secvența 2</i>
ENDIF	SFÎRȘIT DACĂ

Dacă *condiția* este adevărată, se execută *secvența 1*, iar dacă *condiția* nu este adevărată se execută *secvența 2*. Cuvântul cheie ELSE și *secvența 2* sunt opționale.

- **REPEAT-UNTIL** (repetă pînă cînd) este o buclă cu o condiție simplă la sfîrșitul fiecărui ciclu. Această construcție reprezintă o buclă asemănătoare cu bucla WHILE, dar cu diferența că testul este efectuat la sfîrșitul buclei, nu la începutul ei. Se folosesc două cuvinte cheie: REPEAT, UNTIL. Ele pot fi înlocuite de echivalentele românești: REPETĂ, PÂNĂ CÎND. Forma generală a construcției este:

REPEAT	REPETĂ
<i>secvență</i>	<i>secvență</i>
UNTIL <i>condiție</i>	PÂNĂ CÎND <i>condiție</i>

În această buclă *secvența* este executată întotdeauna cel puțin o dată, deoarece testul se face după executarea ei. La sfîrșitul fiecărei iterații se evaluează *condiția*, și se repetă *secvența* din buclă dacă această condiție este falsă (nu este îndeplinită). Buclea este părăsită numai atunci cînd condiția devine adevărată. Între buclele cu pre-testare și cele cu post-testare este o diferență majoră. Versiunea cu pre-testare va lucra chiar dacă nu este nici o valoare numerică de preluat, pe cînd versiunea cu post-testare presupune că instrucțiunile din corpul buclei să fie parcurse cel puțin o dată. Prin urmare, cele două forme sunt adecvate unor circumstanțe diferite.

- **CASE** (în caz că) este o ramificație cu mai multe căi posibil de urmat de acțiunea programului, în funcție de valoarea unei expresii, pe bază de condiții mutual exclusive. CASE este o generalizare a construcției IF-THEN-ELSE. Diversele alternative sunt indicate prin condiții și patru cuvinte cheie: CASE, OF, OTHERS, ENDCASE. Cuvintele cheie pot fi înlocuite de echivalentele românești: ÎN CAZ, CĂ, ÎN CELELALTE (SITUAȚII), SFÎRȘIT DE CAZ.

CASE <i>expresie</i> OF	ÎN CAZ CĂ <i>expresie</i>
-------------------------	---------------------------

<i>condiția 1: secvența 1</i>	<i>condiția 1: secvența 1</i>
<i>condiția 2: secvența 2</i>	<i>condiția 2: secvența 2</i>
.....
<i>condiția n: secvența n</i>	<i>condiția n: secvența n</i>
OTHERS:	ÎN CELELALTE SITUAȚII:
<i>secvență predefinită</i>	<i>secvență predefinită</i>
ENDCASE	SFÎRȘIT DE CAZ

Clauza OTHERS împreună cu *secvența predefinită* asociată este opțională. *Condițiile* sunt în mod uzual numere sau caractere care indică valoarea unei *expresii*, dar pot fi și declarații text ori alte notații care să specifice condiția în care secvența precizată trebuie executată. O anumită *secvență* poate fi asociată la mai multe condiții.

- **FOR** (pentru) este o buclă specială în care o variabilă index este inițializată, incrementată și testată în mod automat. Se folosesc trei cuvinte cheie: FOR, TO, ENDFOR. Ele pot fi înlocuite de echivalentele românești: PENTRU, PÂNĂ LA, SFÂRȘIT PENTRU. Forma generală a acestei bucle este:

FOR <i>index=start</i> TO <i>sfârșit</i>	PENTRU <i>index=start</i> PÂNĂ LA <i>sfârșit</i>
<i>secvență</i>	<i>secvență</i>
ENDFOR	SFÎRȘIT PENTRU

La intrarea în buclă, variabila *index* capătă valoarea inițială *start*. La sfârșitul fiecărei iterații, valoare variabilei *index* crește automat cu o unitate. *Secvența* din buclă este repetată până când variabila *index* atinge valoare finală *sfârșit* (bucloa este părăsită în momentul când variabila *index*, după incrementare, depășește cu o unitate valoarea *sfârșit*).

Aceste construcții sunt suficiente pentru a defini logica de control a oricărui algoritm, fiind de preferat să se utilizeze denumirile din limba engleză întrucât acestea sunt similare celor folosite în toate limbajele de programare curente de nivel superior. În practică s-a dovedit că primele trei din cele șase construcții sunt suficiente pentru descrierea celor mai mulți algoritmi. Construcțiile pot fi incluse oricare în oricare, și pentru ușurarea înțelegerii pseudocodului este foarte utilă diferențierea lor printr-o ierarhie de alinieri progresive, de la stânga la dreapta.

Ori de câte ori o anumită porțiune dintr-un algoritm este parcursă repetat în desfășurarea acțiunii este bine ca acea parte a algoritmului să fie evidențiată ca o funcție. Funcțiile (sau subprogramele) sunt unități logice care execută ceva bine definit. O problemă mare, complexă, poate fi desfăcută într-un număr de probleme mai mici care pot fi rezolvate mai ușor, acestea putând devenii funcții. Funcțiile pot fi testate separat și pot fi reutilizate în alte programe, sporind astfel eficiența activității de programare.

Când se scrie pseudocodul unei funcții trebuie avute în vedere următoarele aspecte:

- Care sunt mărimile de intrare necesare funcției? (Care sunt parametrii de intrare?)

- Ce trebuie să ofere la ieșire funcția? (Ce "întoarce" funcția? Trebuie ea să tipărească/afișeze ceva? Trebuie ea să actualizeze un parametru (eventual apelat prin referință)?)
- Dacă nu este evident după mărimile de ieșire, care este scopul funcției?
- Care sînt pre-condițiile și post-condițiile la care trebuie să răspundă funcția?

*

Pentru fixarea a cel puțin unora dintre noțiunile introduse mai sus, în contextul proiectării/dezvoltării echipamentelor radio definite prin program sau virtuale, se exemplifică utilizarea organigramei în cazul unui algoritm de demodulare de amplitudine și utilizarea pseudocodului la descrierea unui algoritm de translație spectrală.

Modulul de program care va trebui să realizeze demodularea de amplitudine după schema bloc de principiu din figura 2-(a) și organigrama din figura 2-(b) are următoarea **specificație**:

„Fiind disponibile în memoria sistemului de calcul secvențele de câte M eșantioane sincrone ale componentelor în cuadratură ale unui semnal radio, $\{I\}_M = \{I[0]; I[1]; I[2]; \dots; I[M-1]\}$ și $\{Q\}_M = \{Q[0]; Q[1]; Q[2]; \dots; Q[M-1]\}$, să se determine și să se memoreze secvența corespunzătoare a eșantioanelor amplitudinii semnalului radio $\{d\}_M = \{d[0]; d[1]; d[2]; \dots; d[M-1]\}$.”

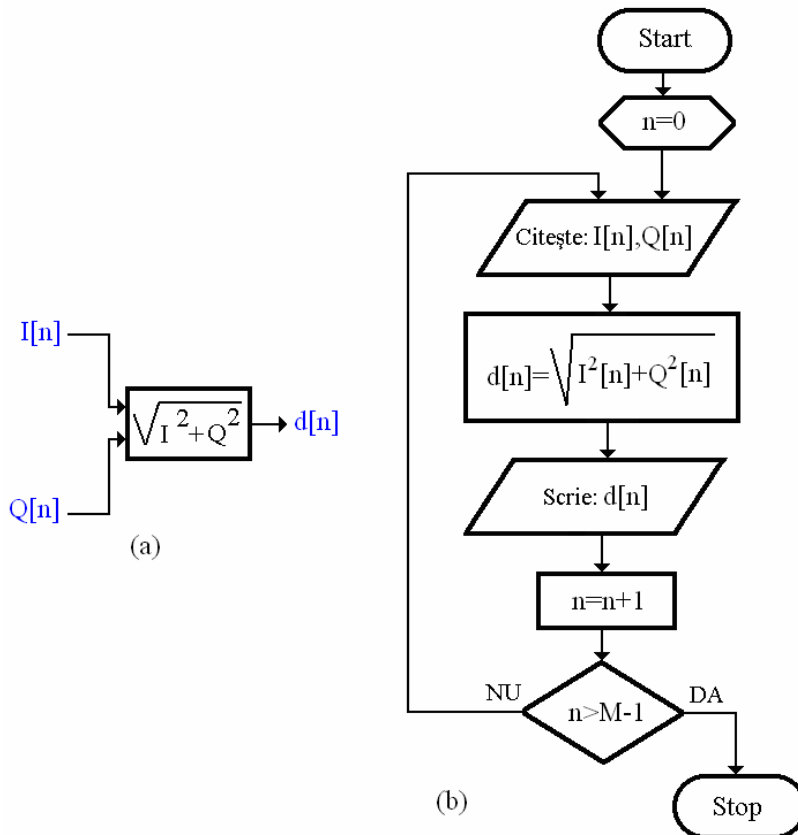


Figura 2

În cel de al doilea exemplu, pe baza schemei bloc de prelucrare din figura 3 în care H este un filtru defazor Hilbert avînd un număr N impar de prize, trebuie să se răspundă la următoarea **specificație**:

„Fiind disponibilă în memoria sistemului de calcul o secvență de M eșantioane ale unui semnal radio $\{s\}_M = \{s[0]; s[1]; s[2]; \dots; s[M-1]\}$, să se determine și să se memoreze secvența corespunzătoare $\{s_s\}_M = \{s_s[0]; s_s[1]; s_s[2]; \dots; s_s[M-1]\}$ a eșantioanelor semnalului radio avînd toate componentele spectrale decalate cu frecvența dF normată la frecvența Nyquist.”

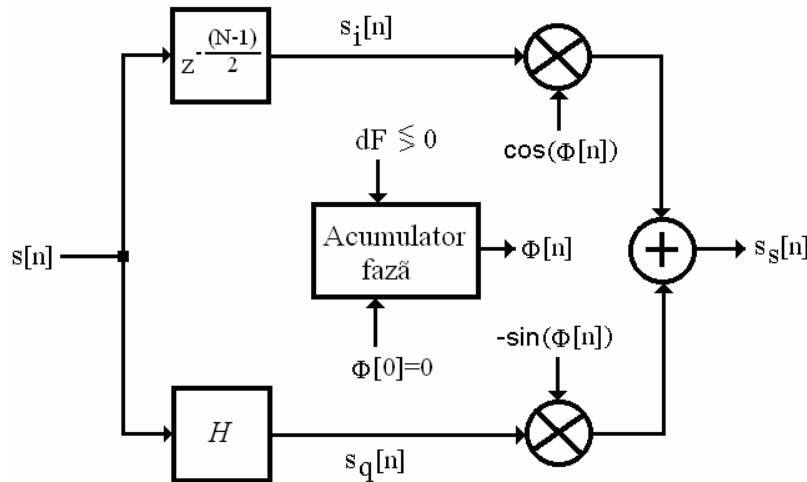


Figura 3

$$\begin{aligned} \Phi[0] &= 0 \\ \Phi[n+1] &= \Phi[n] + \pi dF \quad n \geq 0 \end{aligned} \tag{1}$$

Pseudocodul pentru acest exemplu poate fi formulat astfel:

CITEȘTE: M, N, dF

INIȚIALIZEAZĂ: un filtru defazor Hilbert cu M prize, o linie de întârziere cu $(N-1)/2$ prize, un acumulator de fază

PENTRU $n=0$ PÎNĂ LA $n=M-1$

Se introduce în filtrul Hilbert eșantionul $s[n]$ din secvența $\{s\}_M$.

Se preia de la ieșirea filtrului Hilbert eșantionul $s_q[n]$.

Se introduce în linia de întârziere eșantionul $s[n]$ din secvența $\{s\}_M$.

Se preia de la ieșirea liniei de întârziere eșantionul $s_i[n]$.

Se memorează ca eșantion $s_s[n]$ al secvenței $\{s_s\}_M$ diferența dintre produsul cosinusului eșantionului de fază curent cu eșantionul $s_i[n]$ și produsul sinusului eșantionului de fază curent cu eșantionul $s_q[n]$.

Se incrementează faza cu valoarea πdF .

SFÎRȘIT (PENTRU)

Desigur, modulele de pseudocod referitoare la operațiunile legate de filtrul defazor Hilbert și de linia de întârziere se consideră că sînt tratate separat.