

Întrucât sistemele numerice folosesc pentru reprezentarea numerelor un cuvânt de o anumită lungime operațiile aritmetice se vor executa cu o anumită precizie (limitată). Numerele vor fi astfel reprezentate pe un cerc aritmetică.

## 1.2 Reprezentarea întreagă

Pe un cuvânt de  $n$  biți se pot reprezenta  $2^n$  numere plasate în gama numerelor egal distanțate cu pasul de cuantizare  $q=1$ .

- Reprezentarea binară fără semn**, pentru un număr reprezentat pe  $n$  biți este:  $B_2 = b_{n-1} b_{n-2} \dots b_1 b_0$  cu valoarea zecimală:  $B_{10} = b_{n-1} * 2^{n-1} + \dots + b_1 * 2^1 + b_0$  aflată în gama  $[0, 2^n - 1]$ . Se poate vedea în Fig. 2 "roata reprezentare" pe care se pot verifica proprietățile adunării pentru numerele reprezentate pe un număr finit de biți, în acest caz  $n = 4$ .

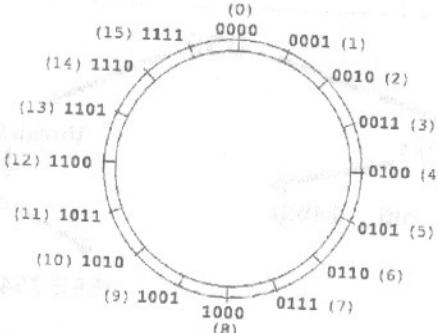


Fig. 2. "Roata numerelor" fără semn reprezentate pe 4 biți

**Exemplu.** Folosind reprezentarea pe 4 biți fără semn se incrementează cel mai mare număr reprezentabil astfel:

$$\begin{array}{r} 1111_2 = 15_{10} \\ 1_2 = 1_{10} \end{array}$$

$1|0000_2 = 0_{10}$ , numărul  $10000_2$  nu mai este pe 4 biți, a apărut depășire!

Bit de transport

**Numerele binare fără semn pe  $n$  biți reprezintă un sistem modulo  $2^n$**

- Pentru a reprezenta **numerele binare negative** (deci cu semn) se utilizează reprezentarea în complement față de 2. Un număr în complement față de 2 pe  $n$  biți este:  $B_2 = b_{n-1} b_{n-2} \dots b_1 b_0$ , cu valoarea zecimală:  $B_{10} = -b_{n-1} * 2^{n-1} + \dots + b_1 * 2^1 + b_0$  aflată în gama  $[-2^{n-1}, 2^{n-1} - 1]$ .

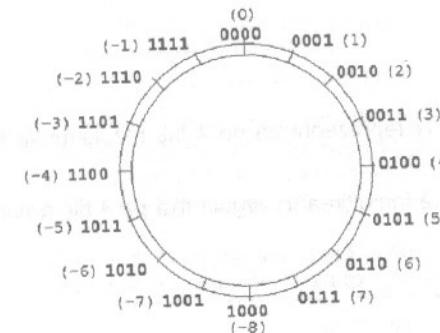


Fig. 3. "Roata numerelor" cu semn reprezentate pe 4 biți

**Observație.** La reprezentarea zecimală numerele  $100_{10}$  și  $0100_{10}$  sunt identice, dar la reprezentarea în complement față de 2 pentru numerele  $1010_2$  și  $01010_2$  nu putem presupune același lucru. De ce? La reprezentarea zecimală se folosește pentru semn un caracter special (-), pe când la reprezentarea binară semnul este dat de bitul de semn. De aceea, la reprezentarea cu semn, valoarea zecimală a numărului  $1010_2$  depinde de lungimea pe care este reprezentat: pe 5 biți are valoarea  $10_{10}$ , iar dacă este pe 4 biți are valoarea  $-6_{10}$ .

### Proprietăți

- Inmulțirea a două numere binare întregi, în virgulă fixă, reprezentate pe  $n$  biți dă un rezultat pe  $2n$  biți
- Adunarea a două numere reprezentate pe  $n$  biți poate da un rezultat pe  $n+1$  biți

Multiplicarea a doi întregi în virgulă fixă duce cu mare probabilitate la depășire.

La  $N$  adunări succesive pentru a nu avea depășire sunt necesari  $\log_2(N)$  biți suplimentari.

**Exemplu.** Folosind reprezentarea în virgulă fixă pe 4 biți să se facă înmulțirea:  $3_{10}$  cu  $2_{10}$ .

**Observație.** Rezultatul este pe 8 biți și se va ține cont de extensia semnului la lungimea cuvântului rezultatului.

$$\begin{array}{r}
 1101 = -3_{10} \\
 0010 = 2_{10} \\
 \hline
 00000000 \\
 1111101 \\
 000000 \\
 00000 \\
 \hline
 11111010 = -6_{10}
 \end{array}$$

Dacă se revine la reprezentarea pe 4 biți rezultatul va fi  $1010_2 = -6_{10}$ , deci este corect.

**Exemplu.** Să se facă înmulțirea în virgulă fixă pe 4 biți a numerelor  $-3_{10}$  și  $6_{10}$ .

$$\begin{array}{r}
 1101 = -3_{10} \\
 0110 = 6_{10} \\
 \hline
 00000000 \\
 1111101 \\
 111101 \\
 00000 \\
 \hline
 11101110 = -18_{10}
 \end{array}$$

Dacă se revine la reprezentarea pe 4 biți, rezultatul va fi  $1110_2 = -2_{10}$ , deci nu este corect.

**Proprietăți.** Orice secvență de operații a cărui rezultat final se poate reprezenta corect în gama dată se poate calcula corect chiar dacă apar depășiri la rezultatele intermediare.

**Exemplu.** Folosind reprezentarea pe 4 biți cu semn (gama  $[-8, 7]$ ) să se calculeze expresia:  $7 + 6 - 8 = 5$

$$\begin{array}{r}
 0111+ 7+ \\
 0110 6 \\
 \hline
 1101 + -3+ (\text{depășire}) \\
 1000 -8 \\
 \hline
 0101 5 (\text{corect})
 \end{array}$$

### 1.3 Reprezentarea fracționară

Pentru a se evita problema depășirilor în cazul înmulțirii, reprezentarea fracționară în virgulă fixă limitează sau normalizează numerele în gama  $[-1.0, 1.0]$ , caz în care problema este rezolvată.

**Excepție.**  $(-1) * (-1) = +1$  ! care nu apartine gamei  $[-1, +1]$ . La adunare și scădere depășirea este totuși posibilă!

#### Observații.

- Rezultatul multiplicării este trunchiat pierzând astfel din precizie prin suprimarea bițiilor mai puțin semnificativi.
- Reprezentarea fracționară pe  $n$  biți se obține prin deplasarea punctului cu  $n-1$  poziții spre stânga, adică reprezentarea conține bit de semn, punct radix și  $n-1$  biți fracționari, deci:  $B_{2^{\text{fracționar}}} = b_0 b_1 \dots b_{n-2} b_{n-1}$ , cu valoarea zecimală  $B_{10} = -b_0 * 2^0 + b_1 * 2^{-1} + \dots + b_{n-1} * 2^{-(n-1)}$ .
- Pasul de cuantizare este  $q = 2^{-(n-1)}$ .

Reprezentarea fracționară este cunoscută și sub numele de reprezentarea  $Q_x$ , unde  $x$  este numărul de biți fracționari. Numărul total de biți ai reprezentării este  $x+1$ . Reprezentarea fracționară ușuală la DSP-urile în virgulă fixă (16 biți) este  $Q15$ .

Localizarea punctului nu este neapărat precizată aceasta fiind un instrument de programare. Astfel folosind reprezentarea pe 16 biți gama numerelor poate fi:

- [0, 65535] - la reprezentarea întreagă fără semn
- [-32768, +32767] - la reprezentarea întreagă cu semn
- [-1, +0.999969] - la reprezentarea fracționară

Toate aceste variante sunt utilizate la aplicațiile cu DSP.

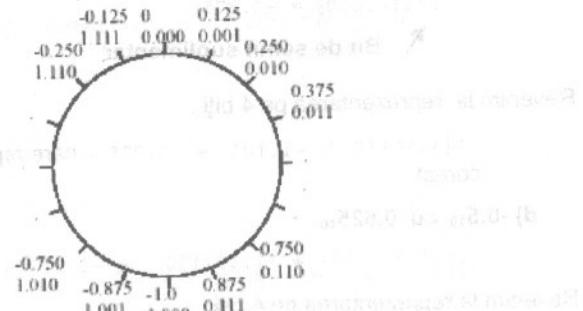


Fig. 4. "Roata numerelor" pentru reprezentarea fracționară pe 4 biți

**Observație:** La adunarea a două numere binare reprezentarea lor trebuie făcută în aceeași convenție.

#### Exemple:

Să se reprezinte în format:

- a) Q3, numerele  $-0.375$  și  $0.75$

La DSP în virgulă fixă punctul radix (zecimal) nu este disponibil, aceasta cade în sarcina programatorului. La operațiile aritmétice cu date fracționare sau cu semn se folosește aceeași unitate aritmetică și logică (UAL).

Reprezentarea fracționară se obține scalând valoarea cu  $2^x$ , unde  $x$  reprezintă numărul de biți fracționari. Deci:

$$\begin{aligned} -0.375_{10} &= 1.101_2 \cdot 2^3 = 1101_2 = -3_{10} \\ 0.75_{10} &= 0.110_2 \cdot 2^3 = 0110_2 = 6_{10} \end{aligned}$$

b) Q15, numărul  $X_{10} = -8.9969653E-003$

Se calculează  $X \cdot 2^{15} = -294.8 \sim -295$ , care în reprezentarea în complement față de 2 este  $X_{Q15} = 0FED9h$ .

Să se înmulțească folosind reprezentarea fracționară pe 4 biți:

c)  $-0.5_{10}$  cu  $0.75_{10}$

**Observație.** Atenție la extensia de semn. Rezultatul va avea 6 biți fracționari din 8.

$$\begin{array}{r} 1.100 \\ \times 0.110 \\ \hline 11.101000_2 \end{array}$$

$$11.101000_2 = -0.375_{10}$$

Bit de semn suplimentar

Revenim la reprezentarea pe 4 biți:

$$\begin{array}{r} 1|1.101|000_2 = 1.101_2 = -0.375_{10} \text{ care reprezintă un rezultat} \\ \text{corect.} \end{array}$$

d)  $-0.5_{10}$  cu  $0.625_{10}$

$$1.100 \cdot 0.101 = 11.101100_2 = -0.3125_{10}$$

Revenim la reprezentarea pe 4 biți:

$$\begin{array}{r} 1|1.101|100_2 = 1.101_2 = -0.375_{10} \text{ care este incorrect datorită} \\ \text{trunchierii.} \end{array}$$

e)  $-0.25_{10}$  cu  $0.5_{10}$  în format Q15.

$$\begin{array}{r} 1.110\ 0000\ 0000\ 0000 * \\ 0.100\ 0000\ 0000\ 0000 \\ \hline 11.11\ 1000\ 0000\ 0000 \end{array}$$

Trecem la format Q15, eliminăm bitul de semn suplimentar și rezultă:

$$1.1110\ 0000\ 0000\ 0000 = -0.125_{10}, \text{ ceea ce este corect.}$$

**Avantaj:** Utilizând fracții binare se obține o viteză mai mare în calculele în buclă închisă.

**Dezavantaj:** Rezultatul poate să nu fie exact. După cum rezultă de mai sus în memorie se obține valoarea  $(-4/16)$ . Biții cuprinși între  $2^4$  și  $2^6$  au fost trunchiați. Rezultatul corect este  $(-3/16)$ !

Este de menționat că operația de înmulțire pe 4 biți nu se face la capacitatea reală a C28x care operează pe 32 de biți. În acest caz trunchierea va afecta biții cuprinși între  $2^{32}$  și  $2^{64}$ . În cele mai multe cazuri se trunchiază numai zgometul. Cu toate acestea, în unele aplicații cu reacție (de exemplu filtrele IIR) eroile mici pot adăuga și conduce la un anumit nivel de instabilitate. Este responsabilitatea proiectantului de a recunoaște această potențială sursă de eșec în folosirea fracțiilor binare.

### 1.3.1. Operații cu numere mai mari ca 1

**Exemplu.** Să se înmulțească eșantionul  $X_i = 0.5$  cu coeficientul 1.375. Se presupune reprezentarea pe 4 biți (Q3).

Se scalează toți coeficienții din algoritm astfel încât să intre în gama [-1,1] iar ca efect rezultă o atenuare a semnalului la ieșire păstrând răspunsul în frecvență.

Se pot utiliza:

- Proprietățile înmulțirii întregi cu 1:  $A \cdot B = (A-1) \cdot B + B$

$$0.5 \cdot 1.375 = 0.5 \cdot (0.375 + 1) = 0.5 \cdot 0.375 + 0.5 = 0.6875$$

$$\begin{array}{r} 0.100 * 0.5_{10} * \\ 0.011 0.375_{10} \end{array}$$

$$\begin{array}{r} 00.001100 + 0.1875_{10} + \\ 0.100 0.5_{10} \end{array}$$

$$\begin{array}{r} 00.101100 0.6875_{10} \\ 0.101_2 = 0.625_{10} \end{array}$$

Revenim la reprezentarea pe 4 biți (Q3):  
Rotunjirea datorată numarului limitat de biți.

- Proprietățile înmulțirii întregi cu 2:  $A \cdot B = A/2 \cdot B + A/2 \cdot B$

$$\begin{array}{r} 1.375/2 = 0.6875_{10} = 0.1011_2 \\ \text{care în format Q3 rotunjit este } 0.101_2 = 0.625_{10} \end{array}$$

$$\begin{array}{r} 0.5_{10} * 0.625_{10} = 0.3125_{10} \\ 0.3125_{10} + 0.3125_{10} = 0.625_{10} \end{array}$$

$$0.100_2 * 0.101_2 = 00.011001_2$$

$$\begin{array}{r} 00.010100_2 + 00.010100_2 = 00.011000_2 \approx 0.101_2 \text{ (în format Q3).} \\ 0.101_2 = 0.625_{10} \end{array}$$

#### 1.4 Teme

Se dă mai jos cei 15 coeficienți ai unui filtru FIR care trece sus. Să se convertească coeficienții în format Q15:

$h_1 = h_{15} = 8.99696E-3;$	$h_2 = h_{14} = -7.86406E-3;$
$h_3 = h_{13} = -3.34992E-2;$	$h_4 = h_{12} = -6.53486E-2;$
$h_5 = h_{11} = -9.8897E-2;$	$h_6 = h_{10} = -0.12856;$
$h_7 = h_9 = -0.14897;$	$h_8 = 0.84375$



## 2. SIMULAREA APlicațiilor PENTRU FAMILIA TMS320C2X

### 2.1 Dezvoltarea programelor în limbaj de asamblare pentru procesoarele TMS320C2X

În prezentă lucrare se vor parcurge toate etapele necesare în dezvoltarea și simularea aplicațiilor pe procesoarele de semnal din familia TMS320C2x. Firmele producătoare de DSP pun la dispoziția utilizatorilor toate instrumentele software necesare dezvoltării aplicațiilor. Pentru a standardiza dezvoltarea aplicațiilor firma Texas Instruments propune utilizarea formatului "Common Object File Format" (COFF) pentru fișierele obiect. Codul se poate împărții în mai multe module care pot fi asamblate separat. Modulele obiect rezultate în urma asamblării se linkedită separat. Împreună obținându-se aplicația executabilă. Linkeditorul realizează alocarea eficientă a resurselor pentru fiecare modul în parte.

Instrumentele software necesare la dezvoltarea unei aplicații sunt: Editor, Asamblor, Linkeditor, Simulator și Debugger (Depanator). Pentru dezvoltarea unei aplicații în limbaj de asamblare pentru procesoarele de semnal din familia TMS320C2x se parcurg etapele prezentate în figura 1.

Fișierul sursă .asm se introduce cu un editor de text clasic și apoi se asamblează cu asamblorul **dspla.exe**. Asamblorul generează un fișier obiect (.obj) și un fișier listing (\*.lst). Fișierul obiect se linkedită cu **dsplink.exe** având opțiunile de linkeditare în fișierul de comenzi link25.cmd, fișier ce precizează harta de memorie a sistemului pe care va rula aplicația. Linkeditorul furnizează un fișier executabil (\*.out) și un fișier care arată harta de memorie ocupată de aplicație (\*.map). Codul conținut în fișierul (\*.out) se poate încărca într-un sistem hardware sau în simulatorul sim2x.exe. Simulatorul sim2x.exe se configureră pentru a putea rula aplicația prin intermediul unui fișier de comandă siminit.cmd. Simulatorul ne permite de asemenea să atașăm fișiere în format ASCII (4 caractere) la porturile de intrare și la porturile de ieșire astfel încât să se poată simula și achiziționa de semnal analogic. Depanarea programelor prin rularea pas cu pas și vizualizarea zonelor de memorie și a reșîrșilor procesor este de asemenea posibilă.

## 2.2 Prezentarea utilitarelor software

### 2.2.1 Asamblorul dspa.exe

**Asamblorul dspa.exe** este un asamblor universal pentru procesoarele de semnal din familia TMS320Cxx în virgulă fixă. La intrare primește fișiere sursă în format ASCII (\*.asm) și generează un fișier de tip obiect (\*.obj).

Utilizarea asamblorului dspa.exe se realizează în modul următor:

```
dspa <fișier intrare> [<fișier obiect> [<fișier listing>]] [- opțiuni]
- <fișierul de intrare> - este fișierul sursă (*.asm)
- <fișierul obiect> - este fișierul ce va conține codul obiect obținut în urma asamblării. Precizarea lui este opțională, dacă nu se specifică va fi creat fișierul cu același nume ca și sursa, dar cu extensia .obj
- <fișierul listing> - este fișierul în care se generează listingul aplicației. Precizarea lui este opțională
- opțiuni care trebuie date asamblorului:
-v10 - versiunea 320C10
-v25 - versiunea 320C25
-v50 - versiunea 320C5X
-l - produce fișier listing ( cu extensia .lst )
-s - toate simbolurile se vor găsi în fișierul .obj
-w - avertizează asupra erorilor de pipeline
```

Exemplu: DSPA sin.asm -v25 -l

Definirea unei secțiuni se realizează după următoare sintaxă:

```
.sect <nume secțiune>
```

#### Secțiuni predefinite

.text – este secțiunea în care se definesc instrucțiunile programului.

Instrucțiunile trebuie să respecte sintaxa următoare:

```
<etichetă> : <memonică> <operanzi> ; <comentariu>
<etichetă> - opțională, trebuie să înceapă pe prima coloană
<memonică> - numele instrucțiuni
<operanzi> - pot lipsi sau să fie mai mulți
<comentariu> - tot ce apare în cadrul liniei după simbolul ";"
```

Dacă există pe prima coloană caracterul "\*" atunci întreaga linie este considerată comentariu.

.data - în această secțiune sunt definite variabilele programului.

Definirea unei variabile se realizează astfel:

```
<etichetă> : .word <listă_valori> ; <comentariu>
```

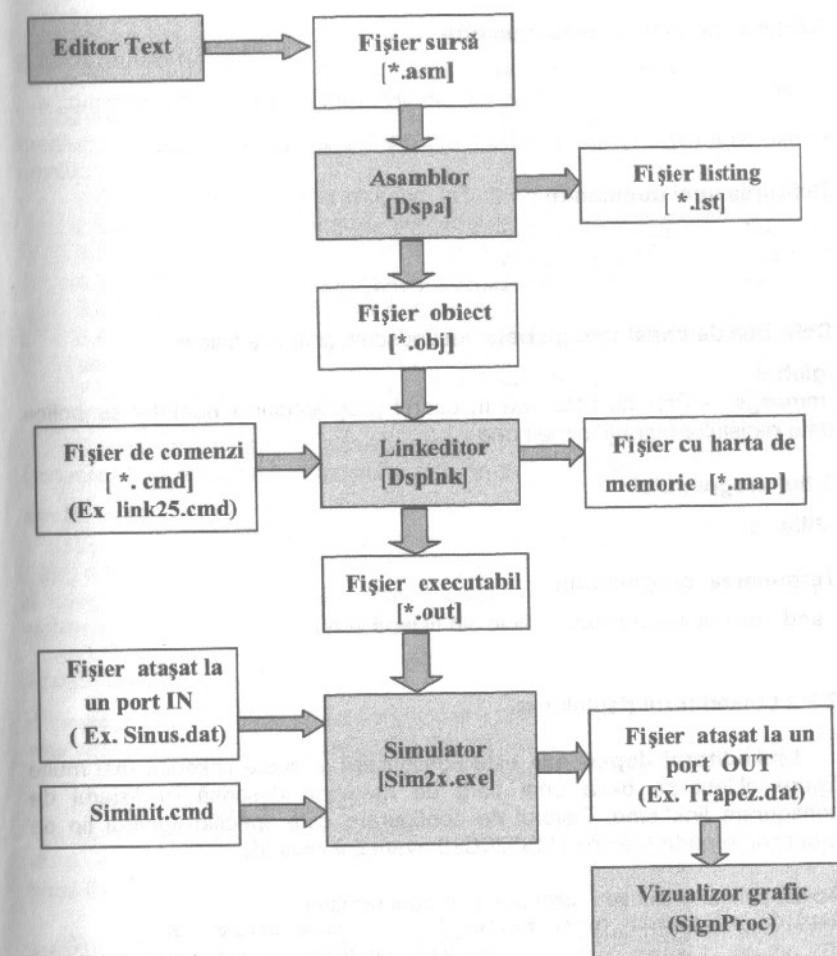


Fig. 1. Etapele dezvoltării unei aplicații pentru simulare

Alocarea spațiului de memorie pentru o variabilă neinițializată se realizează în modul următor:

```
.bss <simbol>,<lungime>,<flag>
<simbol> - este numele variabilei
<lungime> - mărimea zonei de memorie alocată variabilei (word)
<flag> - este un parametru care specifică dacă zona rezervată trebuie să fie continuă sau nu.
```

### Rezervarea unei zone de memorie

```
<simbol> .space <dimensiune>
<simbol> - este simbolul ce definește adresa de început a zonei de memorie rezervată
<dimensiune> - dimensiunea zonei de memorie rezervată
```

### Definirea unei constante

```
<simbol> .set <valoare>
<simbol> - numele constantei
<valoare> - valoarea atribuită constantei
```

### Definirea de constante globale recunoscute și în alte fișiere

```
.global <simbol1>,.....<simbol_n>
.mmregs - Permite utilizarea în cadrul programului a numelor simbolice date reștrictorilor mapării în memorie
```

### Titlul programului

```
.title <titlu>
```

### Terminarea programului

```
.end este obligatoriu ultima linie din fișierul sursă.
```

### 2.2.2 Linkeditorul dsplnk.exe

Linkeditorul **dsplnk.exe** este configurabil și poate linkedita mai multe fișiere obiect pe baza unei hărți de memorie descrisă în fișierul de configurare **link\*.cmd**. Fișierul de configurare este specific fiecărui tip de procesor în parte (pentru TMS320C25 avem **link25.cmd**)

Apelul linkeditorului se realizează în modul următor:

```
dsplnk <opțiuni> <nume fișier_1..... nume fișier _N>
-m <nume fișier>.map - crează un fișier cu maparea memoriei rezultat în urma linkeditării
-o <nume fișier>.out - specifică că se va crea fișierul executabil (.out )
```

#### Exemplu:

```
dsplnk sin.obj link25.cmd -m sin.map -o sin.out
```

**sin.obj** - este programul obiect rezultat în urma asamblării  
**link25.cmd** - este fișierul de comenzi al linkeditorului care conține harta memoriei și adresele la care linkeditorul va pune diferențele secțiuni ale programului executabil. Un exemplu de fișier **LINK25.CMD** se prezintă în anexa 1.

### 2.2.3 Simulatorul sim2x

Simulatorul **sim2x** dispune de o interfață prietenoasă cu ferestre și meniuri. Se poate opera cu mouse-ul și tastatura. Simulatorul ne oferă următoarele facilități:

1. Configurarea memoriei program și a memoriei de date
2. Vizualizarea și modificarea memoriei program dezasamblată
3. Salvarea/încărcarea zonelor de memorie în/din fișiere
4. Vizualizarea fișierelor în fereastra simulatorului
5. Vizualizarea și modificarea memoriei de date
6. Conectarea de fișiere la porturile I/O (simulare I/O eșantioane semnal)
7. Vizualizarea și modificarea reștrictorilor procesorului
8. Execuția programelor
9. Rularea programelor pas cu pas

Lansarea în execuție a simulatorului se face cu comanda:

```
sim2x <fișier.out> <opțiuni> unde:
- <fișier.out> este fișierul generat după linkeditarea fișierului obiect (.obj) de către dsplnk.exe
- <opțiuni> este o listă de parametri opționali care pot transmite informații suplimentare simulatorului.
```

ACESTE OPȚIUNI SUNT:

- l <nume cale> specifică calea de căutare a simulatorului
- mv <versiune> specifică versiunea procesorului
  - (Ex. -mv25 pentru sim2x procesorul C25)
- s - încarcă tabela de simboluri
- v - încarcă fără tabela de simboluri
- t <nume fișier> identifică un nou fișier de configurare ce se încarcă în locul fișierului siminit.cmd

Simulatorul sim2x rulează sub sistemul de operare DOS în mod text. Fereastra principală a programului afișează bara cu comenzi acceptate și patru subferestre de lucru.

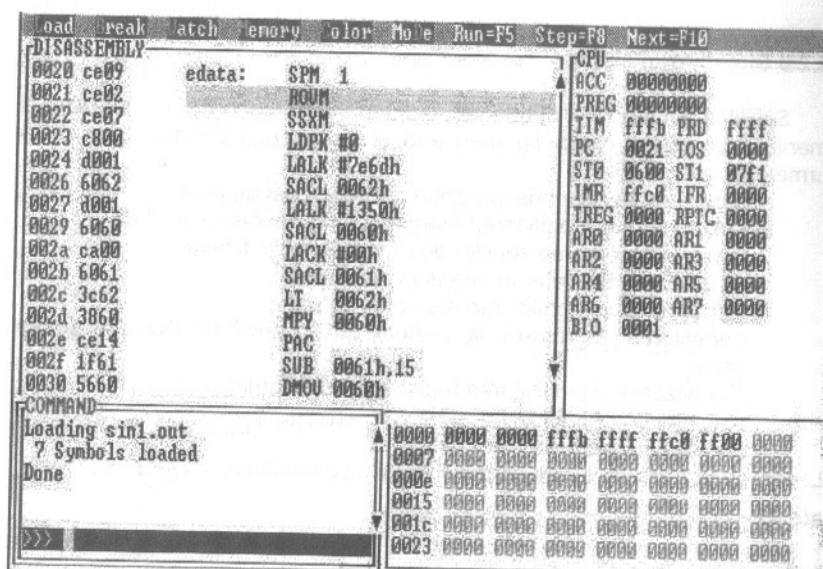


Fig. 2. Fereastra simulatorului sim2x

**Fereastra de dezasamblare (DISASSEMBLY)**

Conține codul dezasamblat din codul obiect încărcat în simulator. Codul dezasamblat se află afișat în următorul format:

|Adresa hexa 4 cifre|cod obiect hexa 4cifre|etichetă|cod limbaj asamblare|  
Exemplu: 0020 | bf00 | edata: | SPM 0 |

**Fereastra memoriei de date(MEMORY)**

Afișează conținutul memoriei de date în formatul următor

|Adresa hexa pe 4 cifre|conținutul memoriei pe 4 cifre hexa |  
Exemplu: | 0023 | 0000 FFFF FFFF 0000 0000 0000 0000 |

Valorile ce se modifică ca urmare a execuției programului sunt afișate intens. Cu ajutorul comenzi **mem** ne putem deplasa rapid în fereastra de memorie precizând adresa de început a zonei afișate. Cuvintele din zona de memorie ce nu sunt cuprinse în configurația simulatorului se afișează cu culoarea roșie.

**Fereastra regeștrilor procesor (CPU)**

Afișează conținutul regeștrilor CPU în format 4 cifre hexa. Valorile ce se modifică ca urmare a execuției programului sunt afișate cu alb intens.

**Fereastra de comandă (COMAND)**

Permite introducere comenzi la prompterul „>>>” și afișarea mesajelor date de simulator. Fereastra memorează ultimele 50 de comenzi.

Deplasarea în lista de comenzi memorate se poate face cu tasta TAB înapoi și cu SHIFT + TAB înainte.

**Comenzi ce se pot lansa de la prompter din fereastra de comenzi:**

-take <nume fisier comenzi> permite lansarea în execuție a unui fișier de comenzi.

-system <comanda DOS> permite lansarea în execuție a comenziilor dos fără a părași simulatorul. Revenirea în fereastra simulatorului se realizează prin apăsarea tastei Enter.

-Quit permite ieșirea din simulator

**- MA Adresa,Tip,Lungime,Tipul Alocării**

MA = memory add

Adresa = adresa de început a zonei de memorie

Tip = tipul zonei de memorie

0=Memorie program

1=Memorie date

2=Spatiu I/O

Lungime = lungimea blocului de memorie

Tipul Alocării = RAM, OPORT/IMPORT

permite adăugarea unei zone de memorie în configurația simulatorului.

**- MC Adresa,Tip,Fisier,Parametru de deschidere**

MC= memory connect

Adresa = adresa de început a zonei de memorie

Tip = tipul zonei de memorie

0=Memorie program

1=Memorie date

2=Spatiu I/O

Fisier = nume fișier

Lungime = lungimea blocului de memorie

Tipul Alocării = RAM,OPORT/IMPORT

Parametru de deschidere = READ sau WRITE

Comanda permite conectarea memorii de tip IMPORT, OPORT, IOPORT la un fișier.

**- MD Adresa,Tip,Lungime,Tipul Alocării**

MD=memory delete

Adresa =adresa de început a zonei de memorie

Tip = tipul zonei de memorie

0 = Memorie program

1 = Memorie date

2 = Spatiu I/O

Lungime = lungimea blocului de memorie

**Tipul Alocării = RAM, OPORT/INPORT**

Comanda permite ștergerea unei zone de memorie din configurația simulatorului.

**- MI Adresa,Tip,Fisier,Parametru de deschidere**

**MI**= memory disconnect

**Adresa** =adresa de început a zonei de memorie

**Tip** = tipul zonei de memorie

0 = Memorie program

1 = Memorie date

2 = Spatiu I/O

**Fisier** =nume fisier

**Lungime** = lungimea blocului de memorie

**Tip Alocării** =RAM,OPORT/INPORT

**Parametru de deschidere** =READ sau WRITE

Comanda permite deconectarea memorilor de tip INPORT,OPORT,IOPORT de la un fisier.

**-MR** (memory reset) permite ștergerea tuturor setărilor referitoare la memorie din configurația simulatorului.

**Bara de comenzi**

Este dispusă în partea de sus a ecranului și are disponibile pe lângă comenzi uzuale și majoritatea comenziilor ce se pot introduce în fereastra de comenzi. Comenzi uzuale sunt:

- Încărcarea unui program. La comanda Load se tastează în fereastra apărută numele fișierului și se dă comanda OK
- Rularea unui program. Se apasă tasta F5 pentru a rula programul.
- Întreruperea rulării unui program. Se apasă tasta ESC pentru a întrerupe rularea

d. Rularea pas cu pas. Se apasă tasta F8 pentru a rula pas cu pas. Se rulează pas cu pas și instrucțiunile din interiorul funcțiilor apelate.

e. Rularea instrucțiune cu instrucțiune. La tastarea lui F10 se rulează programul instrucțiune cu instrucțiune fără a intra în funcțiile apelate.

f. Adăugarea unui Breakpoint (întreruperea rulării unui program într-un punct dorit). Se dă click dreapta pe instrucțiunea unde se dorește a fi plasat breakpoint-ul.

g. Ștergerea unui Breakpoint. Se dă click dreapta pe instrucțiunea unde este plasat breakpoint-ul

**Configurarea simulatorului**

Configurarea simulatorului se realizează prin încărcarea implicită la pornire a fișierului siminit.cfg. Încărcarea unui alt fișier de configurare în locul fișierului implicit se poate realiza cu opțiunea -t(take) la lansarea în execuție a simulatorului.

Fișierul siminit.cfg conține comenzi ce sunt accesibile și la prompterul simulatorului. Dacă se dorește reconfigurarea simulatorului se încarcă un alt fișier de configurare de la prompter cu ajutorul comenzi take.

**Exemplu de configurare:**

```
;MEMORIA PROGRAM
MA 0X0000,0,0x0020,RAM
MA 0X0020,0,0x0F00,RAM
MA 0X0FB0,0,0x0050,RAM
MA 0X1000,0,0x0400,RAM
; VECTORI DE INTRERUPERE
; ROM
; RAM
; OFF CHIP

;MEMORIA DATE
MA 0X0000,1,0x0006,RAM
MA 0X0060,1,0x0020,RAM
MA 0X0200,1,0x0400,RAM
MA 0X0800,1,0x0800,RAM
;MMR
;B2
;B0,B1
;OFF CHIP

; PORTURI
MA 0X0000,2,0X0001,OPORT
MC 0X0000,2,SINUS.DAT,WRITE
```

Este foarte important să corespundă configurația simulatorului cu cea a linkeditorului. Dacă acest lucru nu este realizat aplicația nu poate fi rulată pe simulator. Se obișnuiește să se construiesc câte un fișier de configurație pentru fiecare aplicație în parte.

**2.2.4 Vizualizatorul grafic pentru fișiere „SignProc”**

Vizualizatorul („osciloscopul”) este un program care permite vizualizarea sub formă grafică a fișierelor de date (.dat) în format întreg pe 16 biți “conectate” la simulator ca intrare sau ieșire de semnal .

Programul dispune de două ferestre grafice ce pot afișa două fișiere diferite. Alegerea fișierului dorit pentru vizualizare se face din meniu Fisiere alegem opțiunea Deschide. Salvarea fișierului sub un alt nume se face selectând opțiunea Salvează din meniu Fisiere. Modificarea parametrilor de vizualizare se realizează cu ajutorul grupului de butoane Zoom plasat în partea inferioară a ferestrei principale.

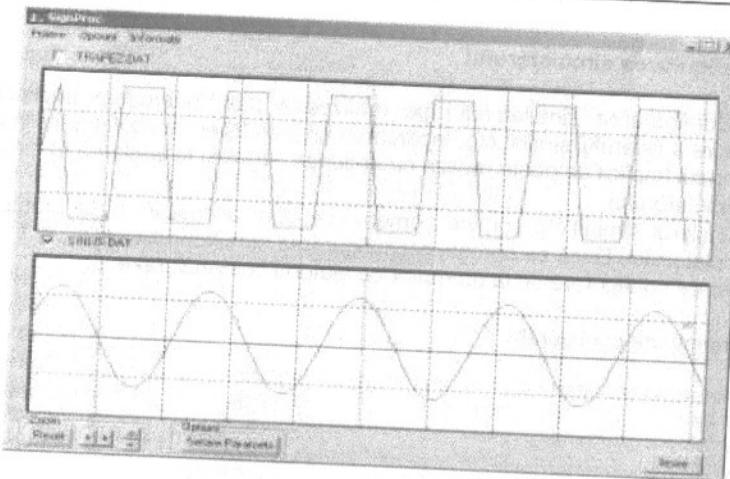


Fig. 3. Vizualizorul grafic de fișiere (.dat) „SignProc”

### 2.3 Aplicație

Aplicația propusă generează un semnal trapezoidal dintr-un semnal triunghiular care se amplifică pentru a intra în limitare ca și în figura 4.

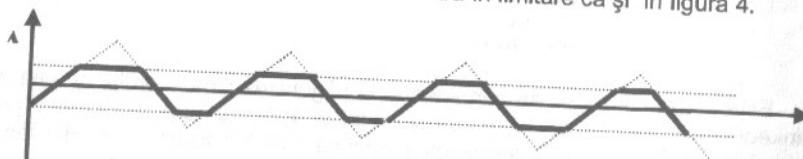


Fig. 4. Semnalul trapezoidal ca semnal triunghiular limitat

Listingul de mai jos prezintă aplicația **trapez.asm**:

```
;Declararea variabilelor utilizate
maximm .set 8000h ; cel mai negativ număr (= -1, în Q15)
pas .set 400h ; pasul dintre 2 eșantioane consecutive
const .set 4000h ; constanta egală cu 0.5 în Q15

;Se rezervă spațiul de memorie pentru variabile
.bss x,1
.bss y,1
.bss z,1
.bss p,1
.bss m,1
.bss ct,1

;Se setează bitul OVM (overflow mode) din ST0
begin sovm
```

;Se initializează registrul DP cu primii 9 biți (cei mai semnificativi) ai numărului x. Variabila x este adresa primului cuvânt rezervat. Prin urmare DP se va poziționa astfel încât să indice pagina în care se găsesc locațiile de memorie rezervate prin directiva .bss

```
ldp #x
```

;Se initializează variabila x și m cu valoarea cea mai negativă
lalk #maximm
sacl x
sacl m

;Se initializează variabila p cu valoarea pasului dintre două eșantioane
lalk #pas
sacl p

;Se initializează variabila ct cu valoarea 0.5 în Q15
lalk #const
sacl ct

;Se încarcă registrul auxiliar AR7 cu valoarea 0 și va fi utilizat pentru contorizarea numărului de eșantioane generate la ieșire acest lucru ne ajută doar în procesul de simulare (avem informația despre numărul de eșantioane generate în orice moment) nu are alt rol în cadrul programului
lark ar7,#0

;Registrul auxiliar AR7 devine registrul auxiliar curent pentru adresarea indirectă
mar \*,ar7

;Bucla de generare a semnalului
;Se calculează  $y = |x| - 0.5$ :

;Conținutul locației de memorie cu adresa x deplasat spre stânga cu 16 poziții în scopul de a plasa valoarea în ACCH și se calculează modulul valorii
loop lacc x,16

;Se aplică funcția modul asupra conținutului ACC rezultatul va fi tot în ACC
abs

;Se scade din ACC conținutul locației de memorie cu adresa ct deplasat spre stânga cu 16 poziții
sub ct,16

;Se salvează ACCH în locația de memorie cu adresa y
sach y

;Se calculează:  $z = 2y = 2(|x| - 0.5) = 2|x| - 1$ 
;Se adună de două ori la ACC conținutul loc. de mem. cu adresa y deplasat spre stânga cu 16 poziții

```

add      y,16
add      y,16

;Se salvează ACCH în loc. de mem. cu adresa. z
sach     z

;Se trimite la portul 0 conținutul loc. de mem cu adresa z.
out      z,0

;Se resetează bitul OV din ST0 (daca a apărut cumva o depășire
;acest bit trece pe 1 - el revine pe zero doar la
;recepționarea unei întreruperi sau dacă se execută o
;instrucțiune de salt condiționat de bitul respectiv
Eti     bv    eti

;Se încarcă ACC cu conținutul locației de memorie cu adresa x
;deplasat spre stânga cu 16 poziții
lac     x,16

;Se calculează x=x+p unde p este pasul de increment a lui x
;atât timp cât nu se semnalizează depășire: Se adună la ACC
;conținutul loc. de mem. cu adresa p deplasat spre stânga cu
;16 poziții
add     p,16

;dacă a apărut o depășire se execută salt la adresa
;specificată dacă nu, se continuă cu executarea următoarei
;instrucțiuni
bv     et

;Se salvează ACCH în loc. de mem. cu adresa x
sach     x

;Salt necondiționat la adresa specificată
b     etl

;Se încarcă ACC cu conținutul loc. de mem. cu adr. m deplasat
;spre stânga cu 16 poziții
et     lac     m,16

;Se salvează ACCH în loc. de mem. cu adresa x
sach     x

;Se incrementează reg. auxiliar curent, se actualizează
;contorul numărului de eșanțioane generate la ieșire
etl     mar     *+

;Se închide bucla de generare a semnalului prin salt
;necondiționat la adresa de început. Semnalul va fi trimis spre
;portul 0 atât timp cât nu se apasă tasta ESC
b     loop

.end

```

## 2.4. Mersul lucrării

- Se studiază utilizarea asamblorului (**dspa**), a linkeditorului (**dsplnk**) și a simulatorului (**sim2x**).
- Se studiază cu atenție aplicația.
- Se lansează în execuție fișierul de comenzi **trapez.bat** prezentat mai jos.

```

@echo off
dspa -v25 -l -x trapez.asm
dsplnk -ar trapez.obj link25.cmd -m trapez.map -o
trapez.out
sim2x trapez.out
signproc.exe

```

- Se rulează programul pas cu pas apăsând tastă F8 .
- După o parcurgere de ~60 de ori a buclei de program ce generează semnalul trapezoidal se tastează **quit** la linia de comandă a simulatorului pentru a se abandona.
- Se lansează vizualizorul grafic și se vizualizează fișierul **trapez.dat**.

## 2.5. Teme

- Să se modifice programul **trapez.asm** astfel încât să genereze semnal triunghiular. Pentru a genera un semnal triunghiular se renunță la amplificarea semnalului, astfel încât acesta să se încadreze între limite.
- Ce modificare trebuie adusă programului de generare a semnalului triunghiular astfel încât acesta să genereze semnal rampă ?



### 3. APlicații PE PLACA "SIDERAL" TMS320C25

#### 3.1 Prezentarea plăcii acceleratoare SIDERAL - TMS320C25

Placa acceleratoare, "SIDERAL", folosită pentru experimentările de prelucrare a semnalului vocal este realizată pe baza procesorului de semnal TMS 320C25 (realizată la ITC Software Cluj-Napoca), având schema bloc prezentată în figura 1.

Din schemă se observă că elementul central al schemei este memoria cu dublu acces: din partea DSP-ului și a calculatorului gazdă, IBM-PC. Memoria este împărțită în două secțiuni: memorie de program și memorie de date, conform arhitecturii Harvard a procesorului de semnal. Oscilatorul este realizat extern pentru a putea modifica ușor caracteristicile sale. Pentru realizarea multiplexorului de date/adrese în cadrul memoriei bipart s-au folosit buffere cu 3 stări (I8286).

Cu ajutorul circuitului generator de stări wait se pot introduce între 0-2 stări de wait la accesarea memoriei. Dintre cele 64 Kcuvinte pentru memoria program și 64 Kcuvinte pentru memoria de date s-au implementat doar câte 32 Kcuvinte, suficient pentru multe dintre aplicațiile posibile.

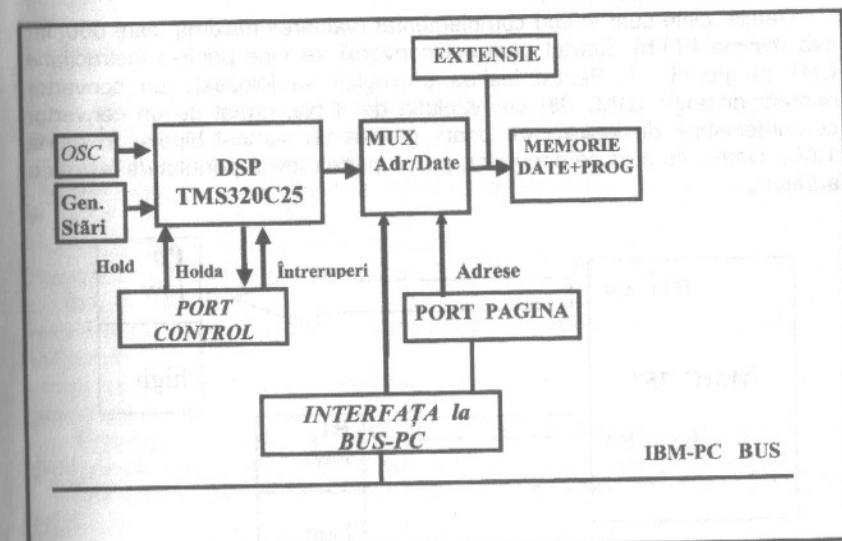


Fig.1. Schema bloc a plăcii acceleratoare cu TMS 320C25

Acest spațiu este paginat folosind adresele superioare, fiind văzut în "feli" de 8 Koctei de date/program de către PC. Portul de paginare este "văzut" la adresa 311H de către PC.

Multiplexorul de adrese/date către PC și DSP este realizat cu buffere "three-state", a căror activare este făcută cu semnalul BHOLD livrat de DSP în urma unei cereri de magistrală prin HOLD. Din punctul de vedere al calculatorului gazdă decodificarea memoriei este făcută între A0000H-EFFFFH, rezervată în orice PC pentru memoria video și extensia BIOS. Se recomandă folosirea zonei D0000H-EFFFFH pentru "fereastra" de 8 Kocteji în acord cu adresa fixată de managementul de memorie.

Porturile folosite pentru paginare/comandă sunt plasate în zona 300H-311H. Întreruperile DSP către PC se aplică la alegere pe unul dintre nivelele IRQ4-IRQ9 și se pot invalida de către PC punând ENIRQ=0 (pe portul de control). Placa mai conține un conector DIN 2X32 pini pe care s-a realizat extensia pentru achiziția de semnal.

Placa de achiziție semnal facilitează achiziția semnalelor de joasă frecvență în spectrul audio (15 Hz-15kHz). Pe canalul analogic de intrare avem un convertor analog-numeric de 12 biți (MMC 757), cu rata de conversie maximă de 25 kHz, pentru semnale în gama de  $\pm 3V$ . Citirea celor 12 biți de către procesor se face astfel: biți 11-4 se citesc pe partea low a portului cu adresa 0, iar biți 3-0 pe partea low a portului 1.(Porturile la TMS320C25 sunt pe 16 biți).

Datele citite sunt în cod complementar (valoarea maximă este 000, iar cea minima FFFh). Startul unei noi conversii se face printr-o instrucție OUT pe portul 1. Pentru ieșirea analogică se foloseste un convertor numeric-analogic (DAC 08) cu rezolutia de 8 biți, urmat de un convertor curent-tensiune de ieșire care poate genera un semnal bipolar în gama 10V<sub>vv</sub>. Datele se scot pe convertor, pe pe partea low a portului de ieșire cu adresa 0.

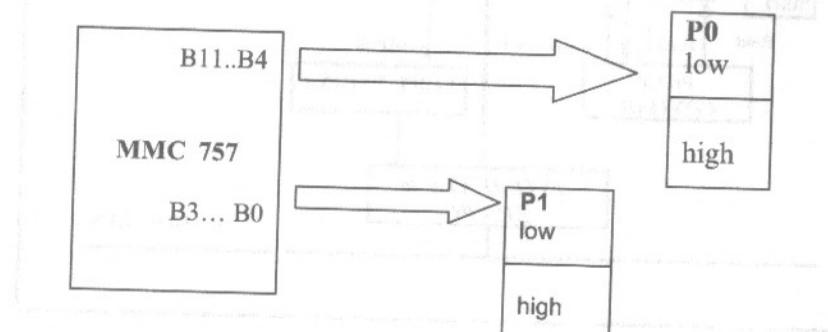


Fig.2 Conectarea convertorului MMC757 la porturi

## 3.2 Prezentarea soft-ului pentru dezvoltarea de aplicații

### 3.2.1. Asamblorul ASC25

Lansarea asamblorului se face cu comanda:

`asc25 <fișier sursă> [<fișier obiect>] [<fișier listing>]`  
unde:

- <fișier sursă> este numele fișierului de tip text care conține programul pentru TMS320C25 scris în limbajul de asamblare al procesorului;
- <fișier obiect> este numele fișierului care va conține codul obiect rezultat în urma asamblării programului din fișierul sursă;
- <fișier listing> este numele fișierului de tip text care va conține listingul programului din fișierul sursă.

Functiile asamblorului sunt:

- analizarea sintactică a liniilor din fișierul sursă;
- afişarea pe ecran a erorilor detectate în cursul asamblării și numărul liniilor din fișierul sursă în care acestea au apărut;
- generarea codului obiect și depunerea lui în fișierul obiect, dacă este specificat numele acestuia în linia de comandă;
- generarea listingului și depunerea lui în fișierul listing, dacă este specificat numele acestuia în linia de comandă;

#### Definiții:

**Identifier** este un sir de caractere care începe cu o literă și poate avea până la 32 de caractere; el desemnează etichete sau constante simbolice.

**Constante**le numerice pot fi:

- binare când sunt reprezentate printr-un sir de cifre binare terminat cu litera 'b' (de ex.: 011101b);
- zecimale când sunt reprezentate de un sir de cifre zecimale (de ex. 27);
- hexazecimale când sunt reprezentate de un sir de cifre hexazecimale terminat cu litera 'h' (ex. 01bh);
- pozitive, ca în care se scrie valoarea numerică absolută a constantei fără nici un semn;
- negative ca în care se scrie valoarea numerică absolută a constantei precedată de semnul '-'.

**Expresie** este orice combinație de constante numerice și simbolice legate prin operatorii +, -, \*, și /. Parantezele pot modifica precedența operațiilor.

**Comentariu** este orice sir de caractere cuprins între simbolul ';' și sfârșitul liniei.

#### Directive de asamblare

**Title**  
sintaxa: title '<sir de caractere>'

descriere: <sir de caractere> este considerat titlul programului și va apărea în antetul listingului;

#### Equ

sintaxă: <constanta simbolica> equ <expresie>

descriere: se evaluează expresia și se atribuie constantei simbolice valoarea ei;  
Data

sintaxă: [<eticheta>] data <expresie>

descriere: se evaluează expresia și valoarea ei și se utilizează pentru initializarea locației de memorie corespunzătoare valorii curente a contorului de program, după care contorul de program se incrementează cu 1.

#### Aorg

sintaxă: [<eticheta>] aorg <expresie>

descriere: se evaluează expresia și valoarea ei și se atribuie contorului de program, astfel că instrucțiunea imediat următoare va fi asamblată la adresa dată de valoarea expresiei.

#### End

sintaxă: End

descriere: marchează optional sfârșitul programului sursă.

#### Phase

sintaxă: [<eticheta>] phase <expresie>

<instructiun>

dephase

descriere: se evaluează expresia și valoarea ei și se atribuie contorului de program, astfel că instrucțiunile ce urmează până la dephase vor fi asamblate pentru adresa dată de valoarea expresiei; adresa de depunere a codului obținut nu este modificată.

### 3.2.2. Programul Încărător LOADRUN

Programul permite încărcarea aplicațiilor .obj rezultate în urma asamblării în memoria plăcii acceleratoare și lansarea în execuție.

Se lansează LOADRUN, iar apoi se cere introducerea numelui fișierului obiect dorit care se va încărca și apoi se va lansa în execuție. Întreruperea rulării se face prin apăsarea unei taste.

### 3.3. Aplicații

#### 3.3.1. Repetor de semnal

În cele ce urmează se prezintă fișierul listing al aplicației "REPETOR" care preia semnalul de pe intrarea analogică și îl transmite ieșirii analogice.

Linie	Adr.	Cod
Sursă	mem.	mașină;
0001		
0002		;
0003		;
0004	4e20 fc	equ 20000
0005	0019 fe	equ 20
0006	00c7 PRD	equ fc/(4*fe)-1
0007	0000 c800	begin ldpk 0 ;DP=0
0008	0001 d001	lalk PRD ;ACClow =PRD
	0002 00c7	
0009	0003 6003	sacl 3 ;reg. PRD = ACClow (vezi mem.Date)
0010	0004 ca08	lack 8 ;ACClow = masca intr. ptr. timer
0011	0005 4d04	or 4 ;ACClow = ACClow U IMR
0012	0006 6004	sacl 4 ;IMR = ACClow, TINT = 1
0013	0007 ce00	eint ;validez intreruperile
0014	0008 ff80	stai b stai ;astept intrerupere
0015	*****	Rutina de intrerupere TIMER *****
0016	0018	aorg 24 ;.space 16*(24-7);
0017		
0018	807f	in 7fh,PA0 ;citeste portul analogic
		;partea high(b11-b4)
0018	0019 207f	lac 7fh ;ACClow=adr.7Fh)mem.date
0019	001a ce27	cmp1 ;ACClow = not (ACClow)
0020	001b 607f	sacl 7fh ;(adr.7Fh)mem.date = ACClow
0021	001c e07f	out 7fh,PA0 ;trimite esantionul citit la P0
0022	001d e170	out 70h,PA1 ;iniziez noua achiziție
0023	001e ce00	eint ;valideaza intreruperi
0024	001f ce26	ret

#### 3.3.2. Filtru nerecursiv (FTJ)

Aplicația FTJ.asm implementează un filtru FIR trece jos cu 15 coeficienți cu frecvență de tăiere de 0.05\*Fe. Implementarea se bazează pe relația:

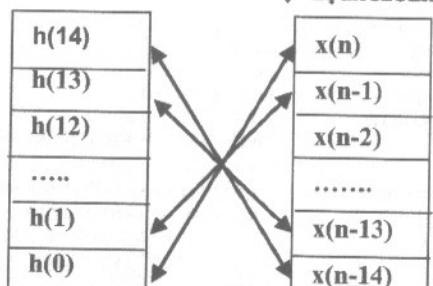
$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$$

unde N=15, h(k) sunt coeficienții filtrului, iar x(n) eșantioanele de intrare. Coeficienții s-au calculat cu un program separat și se păstrează în blocul B0 (plasat în memoria program : FF01h...FF0Fh), iar eșantioanele în blocul B2 (în memoria de date 70h....7Eh) pentru a avea acces în același ciclu la ambele operanzi ai înmulțirii.

B0 (Mem.program)

B2 (Mem.Date)

↓ Eșantioane noi



↓ Eșantioane vechi

```

0001      ;FTJ FIR cu 15 coef. Ftaiere = 0.05*Fe
0002      ;
0003      ;
0004 4e20 fc equ 20000
0005 000f fe equ 10
0006 014c PRD equ fc/(4*fe)-1

0007 0000 c800 begin ldpk 0
0008 0001 d001 lalk PRD
0009 0003 6003 sacl 3
0010 0004 ca08 lack 8
0011 0005 4d04 or 4
0012 0006 6004 sacl 4
0013 0007 ff80 b init
0008 003e
0014           ;space 16*24
0015 0018 aorg 24
0016           ;intrerupere TIMER
0017 0018 8070 in 70h,PA0 ;citesc eșantion nou de la PA0
0018 0019 2870 lac 70h,8 ;Acc =(PA0)* 2^8
0019 001a d006 xork 8000h ;scalare valori de la CAN (B11..B4)
0020 001b 8000
0020 001c 6070 sacl 70h
0021           ;reset ACC si P
0022 001d a000 mpyk 0 ;P=0
0023 001e ca00 zac ;Acc=0
0024           ;filtrarea propriu-zisă
0025 001f 5588 larp ARO ;ARP=0
0026 0020 d000 lrlk ARO,07eh ;AR0=7Eh
0021 007e
0027 0022 cb0e rptk 14 ;repet de 15 ori
0028 0023 5c90 macd Off01h,*- ;vezi ce face MACD!

```

```

0024 ff01 apac ;Acc=Acc+P
0029 0025 ce15 sach 69h ;(69h)= AccH
0030 0026 6869 lac 69h,8 ;AccL=(69h)* 2^8
0031 0027 2869 xork 8000h,8 ;Acc=(Acc) ⊕ 800000h
0032 0028 d806
0029 8000
0033 002a 6869 sach 69h ;(69h)=AccH
0034 002b e069 out 69h,PA0 ;PA0=(69h)
0035 002c e169 out 69h,PA1 ;inițiez o nouă conversie
0036 002d ce00 eint
0037 002e ce26 ret
0038 002f fed9 COEF data 0fed9h ;h14
0039 0030 0102 data 102h ;h13
0040 0031 044a data 44ah ;h12
0041 0032 085d data 85dh ;h11
0042 0033 0ca9 data 0ca9h ;h10
0043 0034 1075 data 1075h ;h9
0044 0035 1312 data 1312h ;h8
0045 0036 1400 data 1400h ;h7
0046 0037 1312 data 1312h ;h6
0047 0038 1075 data 1075h ;h5
0048 0039 0ca9 data 0ca9h ;h4
0049 003a 085d data 85dh ;h3
0050 003b 044a data 44ah ;h2
0051 003c 0102 data 102h ;h1
0052 003d fed9 data 0fed9h ;h0
0053 003e ce09 init spm 1 ;PM=1
0054 003f ce04 cnfd ;B0 în memorie date
0055 0040 5588 larp 0 ;ARP=0
0056 0041 d000 lrlk AR0,0201h ;AR0=201h
0042 0201
0057 0043 cb0e rptk 14 ;mută coeficienții în
0058 0044 fca0 blkp COEF,*+ ;B0 de la 201h...20fh
0045 002f
0059 0046 ce05 cnfp ;flag CNF=1,B0 memorie program
0060 0047 ce00 eint ;validare intreruperi
0061 0048 ff80 stai b stai;astept intreruperi de la timer

```

### 3.4. Terme

- Studiați cele două aplicații și observați modul de utilizare al resurselor procesorului TMS320C25 și a plăcii SIDERAL.
- Asamblați și executați aplicațiile vizualizând semnalele la intrare și ieșire.
- Scrieți un program care să genereze un semnal dreptunghiular cu frecvența de 10KHz și amplitudinea de  $\pm 3V$ .
- Scrieți un program care să genereze un semnal sinusoidal cu frecvența de 2 kHz și amplitudine  $\pm 5V$  folosind metoda tabelară sau recursivă.

e) Pentru un filtru FIR trece sus cu frecvența de tăiere egală cu  $0.1 \cdot f_e$  se dău coeficienții de mai jos:  $h_0=h_{14}=8.99696E-3$ ;  $h_1=h_{13}=-7.86406E-3$ ;  $h_2=h_{12}=-3.34992E-2$ ;  $h_3=h_{11}=-6.53486E-2$ ;  $h_4=h_{10}=-9.8897E-2$ ;  $h_5=h_9=0.12856$ ;  $h_6=h_8=-0.14897$ ;  $h_7=0.84375$ .

Să se treacă în format Q15 și să se modifice în programul FTJ.asm generând un nou program FTS.asm care se va executa vizualizând semnalele de intrare și ieșire.



## 4. DEZVOLTAREA APlicațiilor SUB CODE COMPOSER STUDIO®

### 4.1. Prezentare generală

Code Composer Studio (CCS®) este un instrument foarte eficace pentru dezvoltarea rapidă și performantă a aplicațiilor scrise pentru procesoarele de semnal din familiile TMS320C54x, C55x, C6x. CCS permite: construirea, configurarea, testarea, rularea și analiza în timp real a aplicațiilor. CCS facilitează rularea aplicațiilor pe diferite sisteme de dezvoltare sau pe simulator. Selectia sistemului pentru care se dezvoltă aplicațiile se face prin intermediul componentei Setup CCStudio.

Fazele dezvoltării unei aplicații sub CCS sunt ilustrate în Fig.1.

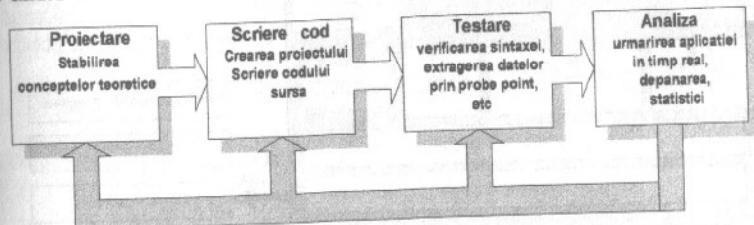


Fig. 1. Fazele dezvoltării unei aplicații sub CCS

Code Composer Studio include următoarele componente:

- **TMS 320C54X Code Generation Tools** – Conține principalele elemente software necesare în dezvoltarea aplicațiilor sub CCS
- **CCS Integrated Development Environment (IDE)** - mediul care integrează și gestionează toate componentele CCS care permit proiectarea, editarea și depanarea aplicațiilor.
- **DSP/BIOS** - sunt biblioteci de funcții care permit comunicarea cu aplicația în timpul rulării pe sistemele hardware. Componenta DSP/BIOS este structurată în două secțiuni:
- **DSP/ BIOS Plug-Ins** oferă posibilitatea de analiză a aplicațiilor pentru estimarea performanțelor cu impact minim asupra performanțelor acestora în timp real.
- **DSP/BIOS API (Application Interfaces)** – furnizează componente software care pot fi apelate prin intermediul sursei aplicației. Componentele API oferă posibilitatea de interfațare a aplicațiilor ce rulează pe sistemul DSP cu aplicații externe care rulează pe PC.

Toate aceste componente lucrează împreună după cum este ilustrat în figura următoare:

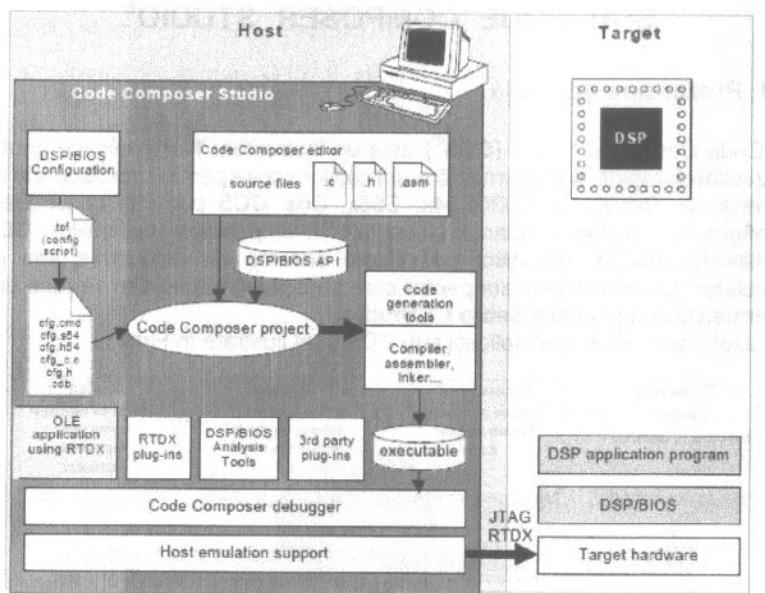


Fig. 2. Componentele utilizate la dezvoltarea unei aplicații sub CCS

#### 4.2. Descrierea utilitarelor software

Majoritatea utilitarelor software utilizate în procesul de dezvoltare a unei aplicații sub CCS sunt gestionate de mediul IDE. Alte componente sunt apelate ulterior pe parcursul dezvoltării aplicației. Fluxul de dezvoltare a unei aplicații este ilustrat în Fig. 3.

- **Compilatorul C** - acceptă sursa C în standardul ANSI și generează fișiere sursă .ASM
- **Asamblorul** - translatează fișierele sursă ASM în fișiere obiect COFF
- **Linkeditorul** - combină fișierele obiect într-un modul obiect executabil
- **Arhivorul** - colectează grupuri de fișiere într-un singur fișier bibliotecă (arhivă). Arhivorul poate modifica biblioteca în sensul: stergerii, reînscrierii, extragerii sau adăugării unui nou membru

- **Translatorul de cod** – translatează codul ASM din forma mnemonică în forma algebraică
- **Managerul de biblioteci** - gestionează bibliotecile run-time
- **Bibliotecile run-time** - conțin:
  - funcții pentru lucru în timp real
  - funcții pentru calcul aritmetic în virgulă flotantă
  - funcții I/O suportate de standardul C - ANSI
- **Utilitarul de conversie hex** - convertește un fișier obiect de tip COFF într-un fișier obiect de tip TI-Tagged, ASCII-hex, Intel, Motorola-S, Tektronix
- **Modulul cross-reference lister** - folosește fișiere obiect pentru a produce un fișier de tip .LST ce va arăta simbolurile, definițiile și referințele în fișierele sursă linkeditate.
- **Modulul absolute lister** - primește la intrare fișiere obiect linkeditate și creează fișiere de tip .ABS la ieșire. Fișierul ABS se asamblează pentru a produce un fișier .LST ce conține în locul adreselor relative, adrese absolute.

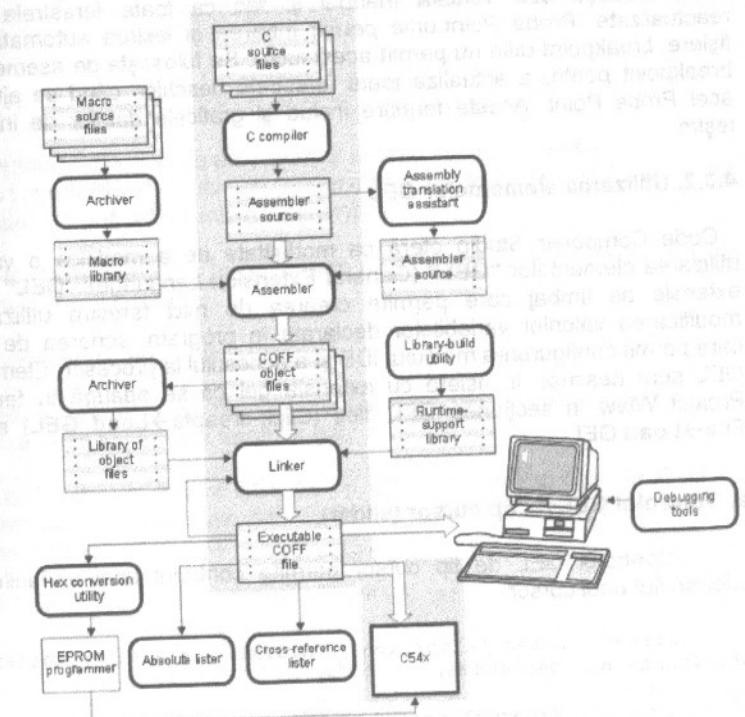


Fig. 3. Fluxul de dezvoltare al aplicațiilor sub Cod Composer Studio

### 4.3. Elemente necesare în dezvoltarea unei aplicații sub CCS

#### 4.3.1. Utilizarea Probe-Point-urilor

Probe Point-urile sunt un instrument util pentru dezvoltarea algoritmilor. Pot fi utilizate pentru :

- a transfera date de intrare dintr-un fișier din PC-ul gazdă într-un buffer din DSP pentru a fi utilizate de algoritm
- a transfera date de ieșire dintr-un buffer din DSP într-un fișier din PC-ul gazdă pentru analiză

-a actualiza o fereastră (cum ar fi un grafic cu date). Probe Point-urile seamănă cu breakpoint-urile prin faptul că ambele întrerup programul ce se execută pe DSP pentru a-și rula acțiunile proprii. Diferențele dintre ele sunt: Probe Point-urile întrerup programul DSP momentan, fac o singură acțiune și continuă execuția programului. Breakpoint-urile întrerup DSP-ul până când execuția este reluată manual și fac ca toate ferestrele să fie reactualizate. Probe Point-urile permit intrarea și ieșirea automată în/din fișiere; breakpoint-urile nu permit acest lucru. Se folosește de asemenea un breakpoint pentru a actualiza toate ferestrele deschise când se ajunge la acel Probe Point. Aceste ferestre includ și graficele datelor de intrare și ieșire.

#### 4.3.2. Utilizarea elementelor GEL

Code Composer Studio oferă ca modalitate de a modifica o variabilă utilizarea elementelor "GEL" (General Extension Language). "GEL" este o extensie de limbaj care permite crearea de mici ferestre utilizate la: modificarea valorilor variabilelor declarate în program, scrierea de funcții care permit configurarea mediului IDE și a accesului la procesor. Elementele GEL sunt descrise în fișiere cu extensia gel ce se adaugă în fereastra Project View în secțiunea GEL files (Click-dreapta→Load GEL) sau cu File→Load GEL.

##### a. Controlul GEL de tip cursor (slider)

Controlul GEL de tip cursor permite controlul unei variabile prin intermediul unui cursor.

```
slider param_definition (minVal, maxVal, increment,
pageIncrement, paramName)
{
    //corpu controlului
}
```

**param\_definition** - este numele cursorului

**minVal** - este o valoare întreagă care specifică valoarea minimă a variabilei controlate de cursor

**maxVal** - este o valoare întreagă care specifică valoarea maximă a variabilei controlate de Slider

**increment** - este o valoare întreagă care specifică valoarea de increment a variabilei controlate de Slider

**pageIncrement** - este o valoare întreagă care specifică valoarea de increment a variabilei controlate de Slider atunci când se apasă tastele PageUp și Page Down

**paramName** - este numele parametrului ce se utilizează în interiorul funcției

##### b. GEL de tip dialog

“nigero clasa 1” – enemem so telurios e neñig condicisiv. S. A  
Acet tip de control permite modificarea unei variabilelor prin intermediul căsuțelor de dialog:

```
dialog funcName( paramName1 "param1 definition", paramName2
"param2 definition", ..., ...
{
    //corpu controlului
}
```

**paramName[1-6]** - numele parametrilor utilizati în corpul funcției

"param1 definition" - numele parametrilor ce apar tipăriți în fereastra controlului. Se pot defini maxim 6 parametri.

**Exemple de fișiere GEL:** În continuare se definesc două elemente GEL de tip slider respectiv dialog.

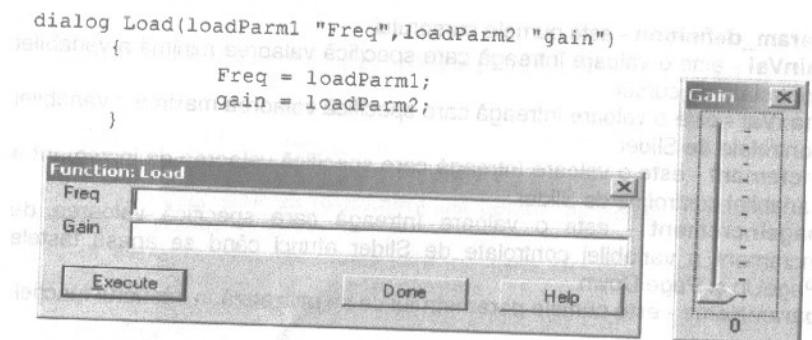
**menuitem "Application Control"**

**Application control** este numele submeniului din meniul GEL care se activează după încărcarea fișierului gel.

a. Se definește un control GEL de tip dialog (slider) cu numele Gain ce poate modifica variabila gain între limitele: 0 și 10 cu increment de 1 și post increment de 1.

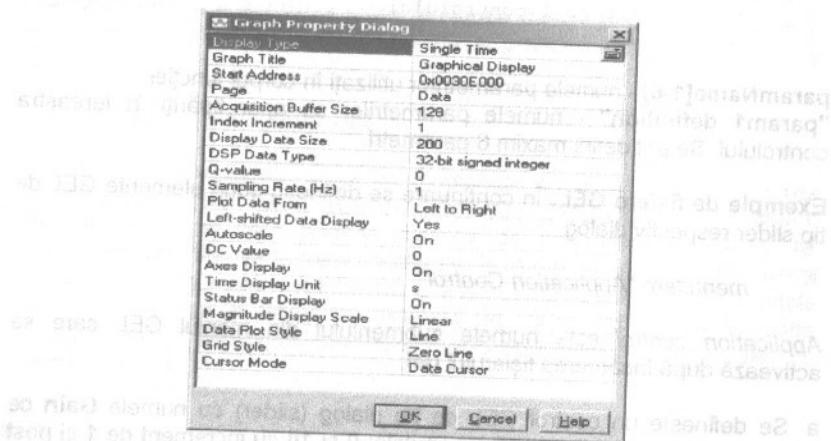
```
slider Gain(0, 10 ,1, 1, gainParm)
{
    gain = gainParm;
}
```

b. Se definește controlul Gel de tip dialog cu numele Load cu două variabile loadParam1 și loadParam2 atașate variabilelor din program Freq respectiv gain. Variabilele se actualizează după apăsarea butonului Execute.



### 4.3.3. Vizualizarea grafică a zonelor de memorie – “Fereastra Graph”

Prin utilizarea fereștrelor de tip Graph putem vizualiza zone de memorie sub formă grafică în domeniul amplitudine-timp sau dublu (dual time), FFT-amplitudine, FFT complex, FFT fază, FFT fază și amplitudine. Fereastra de configurare a unui Graph amplitudine-timp cu setările aferente se prezintă mai jos.



**Display Type** - domeniul în care se afișează

**Graph Title** - titlul ce se afișează pe grafic

**Start Address** - adresa de start

**Page** - pagina de memorie ce se afișează Program Data sau I/O

**Acquisition BufferSize** - mărimea bufferului citit de către Graph

**Index Increment** - rezoluția afișării

**Display Data Size** - numărul de eșanțioane afișate

**DSP Data Type** - formatul de afișare a datelor

**Q-value** - tipul datelor afișate în format fracționar

**Left-shifted Data Display** - tipărire cu deplasare spre dreapta

**Sampling Rate(Hz)** -  $F_{es}$  în Hz pentru afișarea valorilor pe grafic

**Plot data From** - tipărește datele de la stânga la dreapta sau invers

**Autoscale** - autoscalare în amplitudine

**DC Value** - offset-ul

**Axes Display** - afișarea axelor

**Time Display Unit** - unitatea masură pe axa timpului: s, ms,  $\mu$ s, eșanțioane

**Status Bar Display** - afișare bare de stare pentru tipărire cordonate cursor

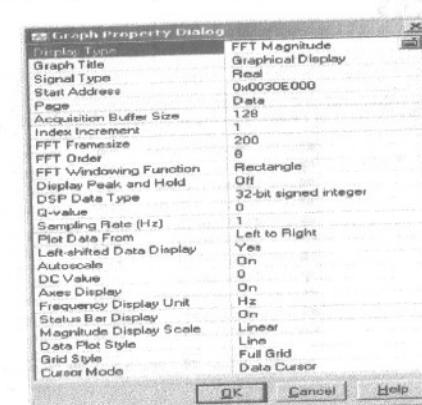
**Magnitude Display Scale** - tipul scalei pt amplitudine: liniar sau logarithmic

**Data Plot Style** - stilul de afișare Line sau Bar

**Grid Style** - afișează rastrul (grid-ul), linia de zero sau nimic

**Cursor Mode** - fără cursor, cursor date, cursor Zoom

În cazul în care graficul este în domeniul frecvență, avem următorii parametri noi:



**SignalType** - tipul semnalului real sau complex

**FFT-Frame size** - mărimea ferestrelui FFT

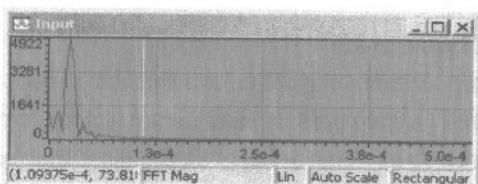
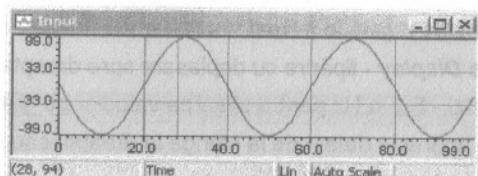
**FFT Order** - ordinul FFT

**FFT Windowing Function** - tipul ferestrei de ponderare

**Display Peak and Hold** - detectează și păstrează maximele

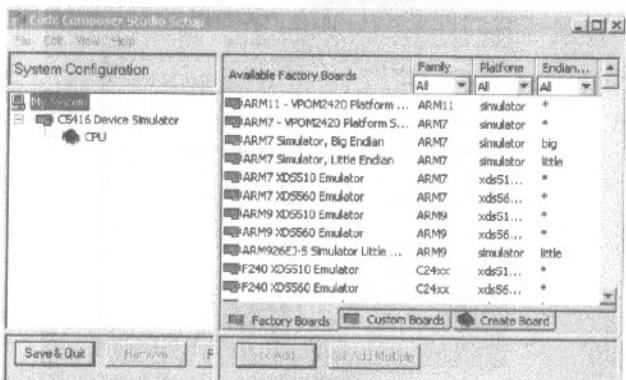
**Frequency Display Unit** - unitatea de afișare pt frecvență: Hz, KHz, MHz

În continuare se prezintă același semnal de intrare în domeniul timp respectiv frecvență:



#### 4.3.4. Exemplu de lucru cu CCS, folosind Simulatorul C5416

După ce ati instalat CCS v3.1, configurați-l astfel încât să lucreze cu simulatorul "C5416 Simulator". Această configurare se face prin intermediul componentei **Setup CCS**.

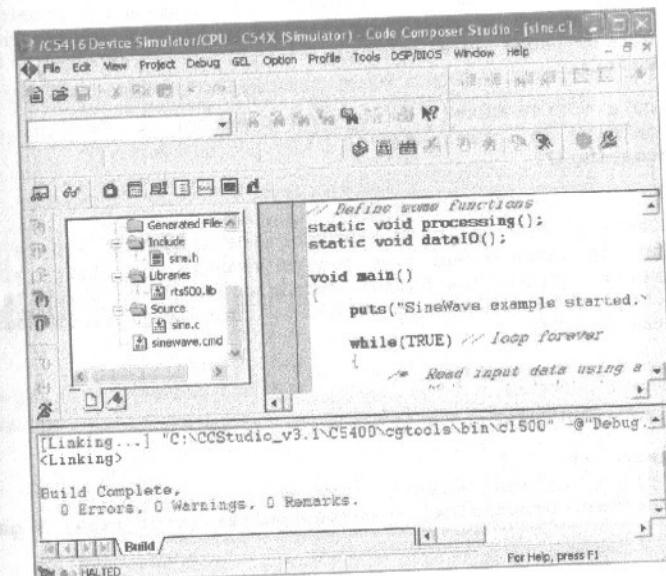


**Simulatorul** este un program care rulează pe calculatorul gazdă. El acceptă la intrare cod obiect și simulează operațiile efectuate de procesorul DSP aşa cum ar rula acesta programul, dar nu în timp real, ci sub controlul utilizatorului. Interfața utilizator prezintă memoria, registrele interne, I/O, etc., și efectele asupra acestor resurse după execuția fiecărei instrucțiuni. Pot fi inserate puncte de întrerupere (break points), sau programul poate fi rulat

pas cu pas, iar conținutul oricărui registru intern, a memoriei interne de program sau de date pot fi examinate în diferite moduri de fișare alese de utilizator.

În continuare sunt descriși pașii ce trebuie urmați pentru a dezvolta o aplicație în CCS și pentru a o rula pe simulator. Aplicația prezentată în **sinewave.pjt** realizează controlul amplificării unui semnal de intrare.

- 1) Se alege meniul **Project → Open** și se deschide proiectul **sinewave.pjt**, din **Tutorial/sim54x/Sinewave**. Proiectul contine fisierele **sine.h**, **rts500.lib**, **sinewave.cmd** precum și sursa principală a aplicației **sine.c**.



În continuare este prezentată sursa aplicației :

```
*****
/*sine.c ; Acest program folosește Probe Point-uri pentru a
obține un semnal sinusoidal de intrare, iar apoi acestui semnal
i se aplică un factor de căstig.
*****
#include <stdio.h>
#include "sine.h"

// variabila de control a căstigului
int gain = INITIALGAIN;

// declarare și inițializare Buffer IO
BufferContents currentBuffer;
```

```

// definire funcții
static void processing();
// prelucrează intrarea și generează semnalul de ieșire
static void dataIO();
// funcția ce se folosește pentru ProbePoint

void main()
{
    puts("SineWave example started.\n");

    while(TRUE) // buclă continuă
    {
        /* citește datele de intrare folosind un probe-point conectat
        la un fișier gazdă. Scrie datele de ieșire într-un graf conectat
        tot printr-un probe-point */
        dataIO();

        /* pentru a obține datele de la ieșire se aplică câstigul
        asupra datelor de intrare */
        processing();
    }
    /* Funcția urmatoare aplică o transformare de semnal asupra
    semnalului de intrare pentru a genera semnalul de ieșire și are
    ca parametrii: structura BufferContents ce conține șirurile de
    intrare/ieșire de dimensiune BUFSIZE; funcția nu returnează
    nici o valoare */

    static void processing()
    {
        int size = BUFSIZE;

        while(size--)
        { // aplică câstigul asupra intrării
        currentBuffer.output[size] = currentBuffer.input[size] * gain;
        }

        /* Funcția urmatoare citește semnalul de intrare și scrie
        semnalul de ieșire procesat folosind ProbePoints; funcția nu
        are parametrii și nu returnează nici o valoare. */
    }

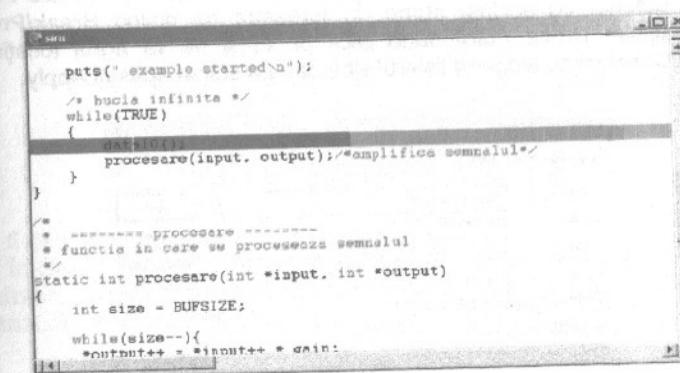
    static void dataIO()
    {
        return;
    }
}

```

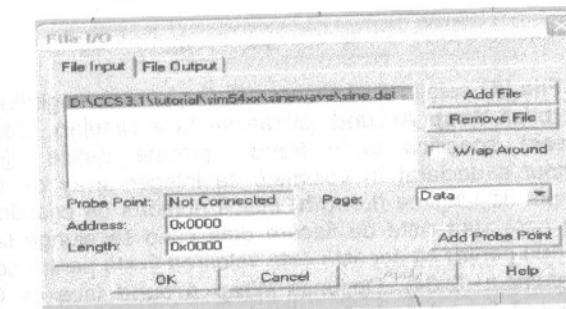
- 2) Comparați fișierele sursă cu Project→Rebuild All sau apăsați pe butonul (Rebuild All) din toolbar.
- 3) Încărcați fișierul obiect în memoria DSP din meniu File→Load Program. Selectați programul pe care tocmai l-ați reconstruit, sine.out și apăsați Open.
- 4) Dați dublu-click pe fișierul sine.c din Project View.

5) Pentru a ataşa fișierul ce conține semnalul ce urmează să fie procesat este necesar să inserăm un ProbePoint (care citește date dintr-un fișier de tip .dat). În acest scop poziționați cursorul pe rândul din funcția main unde se apelează funcția DataIO(); Funcția DataIO rezervă un loc unde se pot face adăugiri mai târziu. Pentru moment vom conecta un "Probe Point" care introduce date dintr-un fișier din PC.

6) Apăsați pe butonul (Toggle Probe Point) din toolbar. Rândul e evidențiat pe display.



7) Alegeti File→File I/O. Dialogul File I/O apare pentru a se putea selecta fișierele de intrare și de ieșire.



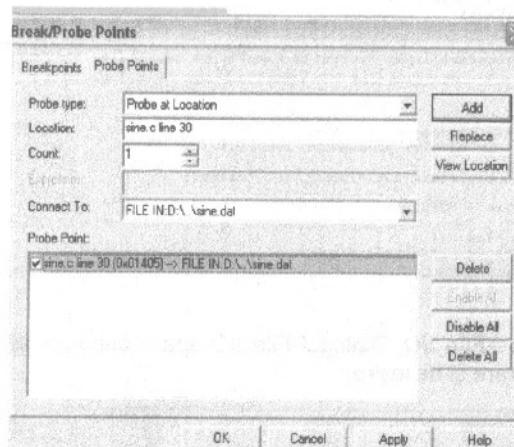
8) În meniul File Input apăsați pe Add File.

9) Alegeti fișierul sine.dat. Observați că puteți selecta formatul datelor în fereastra Files of type. Fișierul sine.dat conține valori hexa pentru o formă de undă sinusoidală. Selectați Open pentru a adăuga acest fișier în lista de dialog File I/O. Va apărea o fereastră de control pentru fișierul sine.dat (poate fi acoperită de fereastra CCS). Mai târziu, când rulați programul,

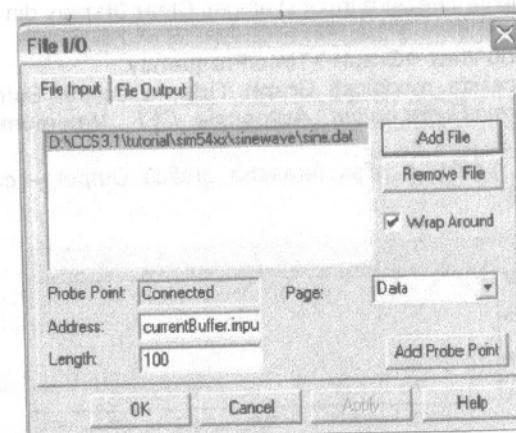
puteți folosi această fereastră pentru comenzi ca: start, stop, rewind sau fast forward în cadrul fișierului de date.



10) În fereastra File I/O urmărol pas este să adăugam un Probe Point utilizând butonul cu același nume. În fereastra de dialog Break/Probe Points validăm Probe Point dând click pe el și ne va arăta locația și selectăm Connect to alegând fișierul sine.dat. La sfârșit apăsați Apply.

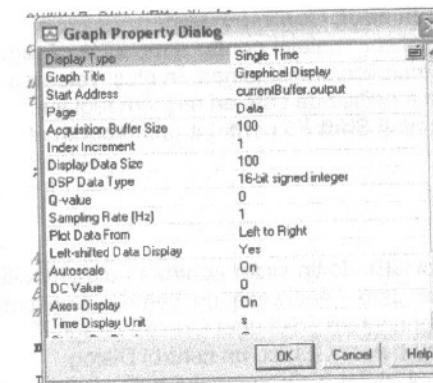


11) În dialogul File I/O setați adresa *currentBuffer.input* și lungimea la 100. De asemenea, bifați Wrap Around (citirea se face circular). Câmpul de adresă (Address) specifică unde trebuie plasate datele din fișier. *CurrentBuffer.input* e declarat în *volume.c* ca integer array de valoarea BUFSIZE. Câmpul de lungime (Length) indică numărul de eșantioane din fișierul de date care sunt citite de fiecare dată când se ajunge la Probe Point. Folosim 100 pentru că aceasta este valoarea fixată pentru constanta BUFSIZE în *volume.h* (0x64). Opțiunea Wrap Around face ca CCS să înceapă să citească fișierul de la început după ce a ajuns la sfârșitul lui. Aceasta permite fișierului de date să fie tratat ca un flux continuu de date deși conține doar 1000 de valori și la fiecare Probe Point sunt citite câte 100 de valori.

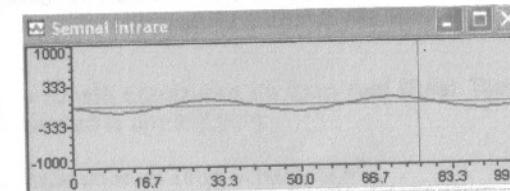


12) Select View→Graph→Time/Frequency.

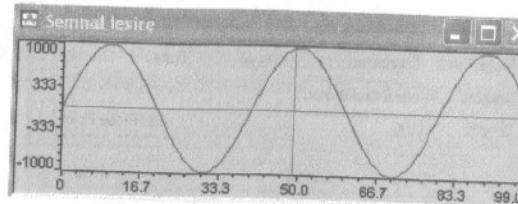
13) În Graph Property Dialog modificați caracteristicile: Graph Title, Start Address, Acquisition Buffer Size, Display Data Size, DSP Data Type, Autoscale, și Maximum Y-value cu valorile din figură.



14) Apăsați OK. Apare o fereastră grafică pentru bufferul de intrare.



- 15) Click-dreapta pe fereastra Input și alegeți Clear Display din meniu care apare.
- 16) Alegeți din nou View→Graph→Time/Frequency.
- 17) De data aceasta modificați Graph Title în Output Buffer și Start Address în currentBuffer.output, Autoscale OFF, MaximumY-Value la 1000.
- 18) Apăsați OK pentru a afișa fereastra grafică Output – care conține semnalul generat.



- 19) Dați click-dreapta pe fereastra grafică și alegeți Clear Display din meniu afișat.
- 20) Alegeți File→Load Gel→volume.gel. Acest fișier conține un control GEL de tip slider și unul de tip dialog. Deschideți fișierul pentru a-i vedea conținutul.
- 21) Selectați Gel→Application Control→Gain.
- 22) În fereastra Gain utilizați reglajul pentru a modifica câștigul. În fereastra bufferului de ieșire se modifică amplitudinea. În plus, valoarea variabilei gain din fereastra Watch se modifică de câte ori mișcăm reglajul.
- 23) Click (Halt) sau apăsați Shift F5 pentru a opri programul.

#### 4.5. Teme

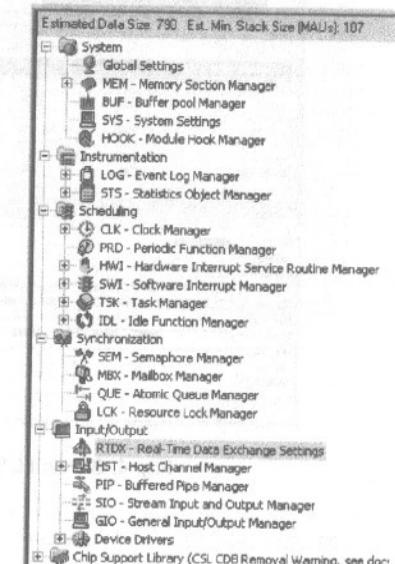
- a) Adăugați un control GEL de tip slider pentru a putea modifica câștigul.
- b) Scrieți un program care generează un semnal sinusoidal. Vizualizați buffer-ul de ieșire prin intermediul unei ferestre Graph.
- c) Înlocuiți controlul cu cursor GEL cu un control Dialog.
- d) Modificați programul astfel încât semnalul preluat din fișierul de intrare sine.dat să fie modulat în amplitudine cu semnal sinusoidal generat în program.
- e) Modificați Graph astfel încât să afișeze semnalul rezultat în domeniul frecvență.

#### 5. Sistemul de operare DSP, numit DSP/BIOS (basic input/output system)

CCS pune la dispoziția utilizatorului un sistem de operare DSP, numit DSP/BIOS (basic input/output system). Acesta reprezintă un nucleu software de timp real care permite utilizatorului să organizeze sarcinile în programul DSP și să monitorizeze performanța de timp real a codului pentru DSP, fără a fi necesară stabilirea unor puncte de oprire. Spre deosebire de depanarea convențională, unde utilizatorul trebuie să opreasă execuția DSP prin puncte de stop, DSP/BIOS furnizează servicii run-time (în timpul rulării) și poate detecta punctele slabe care nu ar fi putut fi detectate prin depanarea convențională.

##### Activitate de studiu individual:

- a. Studiați interfața grafică pentru setarea statistică a sistemului, utilizând instrumentele de configurație DSP/BIOS GUI (File→New→DSP/BIOS Configuration).



Obiectivele DSP/BIOS principale, pe care să le aveți în vedere, sunt :

- Intreruperile hardware (HWI)
- Intreruperile hardware (SWI)
- Task-uri (TSK, IDL)
- Fluxuri de date și I/O: RTDX, SIO, PIP, HST
- Sincronizare și comunicare (SEM)
- Timing (PRD, CLK)
- Log și statistici (LOG, STS)

- b. Studiați în ce constă Etapizarea de timp real (Real Time Scheduling). Obiectivele pe care să le urmăriți sunt:

- Ce este un Real Time Scheduler?

- De ce avem nevoie de un Real Time Scheduler ?
- Fire de execuție DSP/BIOS
- Exemplele din CCS legate de Real Time Scheduler

c. Studiați instrumentele de analiză în timp real. În mod clasic procesorul era oprit pentru examinarea variabilelor din memorie. Metoda este invazivă și nu este indicată pentru sistemele de timp real. Analiza în timp real constă în analiza datelor în timpul funcționării unui sistem fară a-l opri. Obiectivele pe care să le urmăriți sunt:

- Utilizarea modului LOG
- Utilizarea modului STS
- Exemplele din CCS legate de analiza în timp real

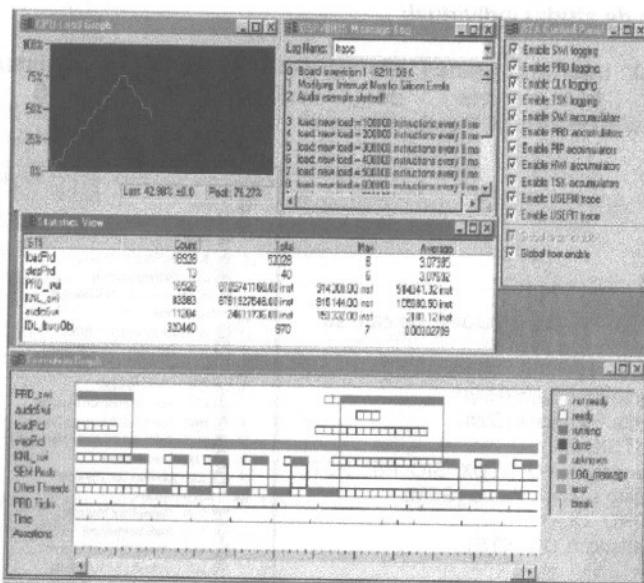


Fig. 4. Servicii DSP/BIOS oferite de CCS



## 5. SISTEMUL DE DEZVOLTARE TMS320VC5416 DSK

### 5.1. Prezentare generală

TMS320VC5416 DSK este un sistem de dezvoltare de sine stător. Permite evaluatorilor să examineze diferite caracteristici ale procesoarelor de semnal C5416, pentru a observa dacă îndeplinesc cerințele aplicației lor. Modulul este o platformă excelentă pentru a dezvolta și a rula programe pentru procesoarele digitale din familia TMS320VC5416.

O schemă simplificată a plăcii VC5416 DSK este prezentată în Fig. 1:

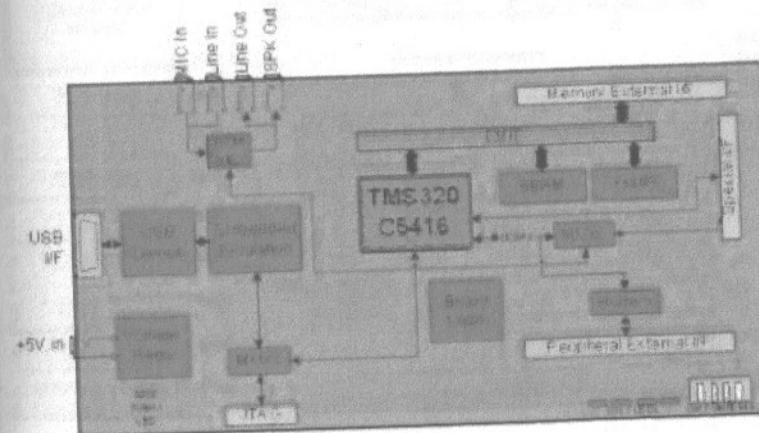


Figura 1. Placa TMS320VC5416 DSK

Sistemul are următoarele caracteristici:

- 64k cuvinte memorie SRAM;
- 256k cuvinte on-board flash ROM;
- 3 conectoroare de extensie (interfață memorie, interfață pentru periferice, interfață pentru portul gazdă - HPI (Host Port Interface));
- controler USB JTAG cu drivere plug and play;
- codec stereo Burr Brown PCM 3002;
- operații la +5V;
- VC5416 lucrează la 16-160 MHz;

DSK 5416 este o placă multi-layer de 210x115mm, alimentată de o sursă de putere externă de 5V. Placa este echipată cu o interfață I/O externă care suportă porturi paralele I/O și porturi serial-sincrone (echipate cu buffere) pe mai multe canale. Interfețele importante ale DSK includ interfețele pentru RAM și ROM, FPGA, interfața pentru Codec și interfața de expansiune. Patru mufe jack stereo asigură intrări și ieșiri de la codec.

DSK permite verificarea codului VC5416 la viteza maximă. Cu 64k cuvinte memorie RAM, 256k cuvinte memorie flash ROM, și un codec stereo Burr Brown PCM 3002, placa reprezintă soluția pentru o mulțime de probleme.

Elementul central al plăcii este procesorul de semnal TMS320C5416 a cărui arhitectură multibus avansată este prezentată în continuare:

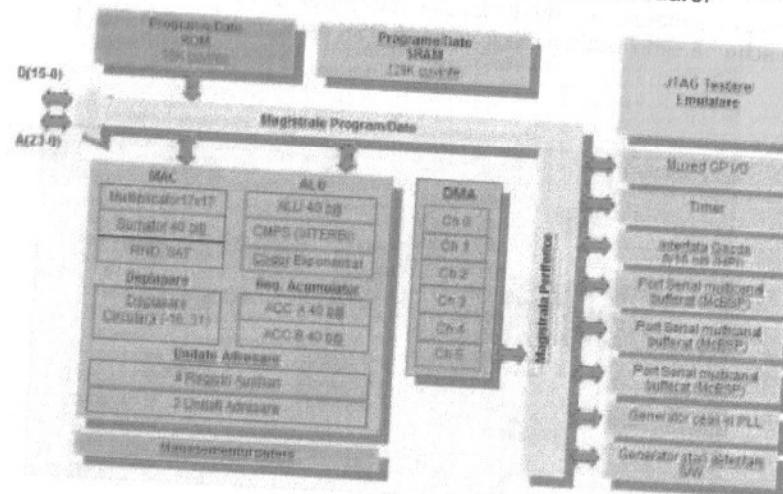


Fig. 2. Schema bloc a procesorului de semnal TMS320C5416

- arhitectură avansată multibus (3 bus-uri separate de memorie date pe 16 biți și unul de memorie program);
- unitate aritmetică și logică pe 40 biți, incluzând 2 registre acumulator pe 40 biți și unul de shiftare;
- multiplicator paralel 17x17 biți cuplat cu un sumator dedicat, pe 40 biți, pentru operații de multiplicare/acumulare într-un singur ciclu "non-pipelined";
- unitate de comparare, selectare și stocare (CSSU) pentru selecție, adunare/comparare a operatorului Viterbi;
- codificator exponential pentru a calcula o valoare exponențială a unei valori din acumulatorul de 40 biți (1 ciclu);
- două generatoare de adrese cu 8 registre auxiliare și 2 unități de registre aritmetice auxiliare;

- mod de adresare extins pentru 8M x 16biti – spațiu maxim de adresare pentru program extern;
- spațiu maxim de adresare memorie 192k x 16 biți (64k cuv. program, 64k cuv. date, 64k cuv. I/O);
- memorie ROM pe cip cu câteva configurații de memorie date/program;
- memorie RAM pe cip cu acces dual;
- repetarea unei instrucțiuni și operații de repetare pe bloc pentru codul de program;
- instrucțiuni de mutare blocuri de memorie - pentru o mai bună programare și manevrare a datelor;
- instrucțiuni cu operanzi pe cuvinte de 32 biți;
- instrucțiuni cu citiri de 2 sau 3 operanzi;
- instrucțiuni aritmetice cu stocare și încărcare paralelă și stocare condiționată;
- întoarcere rapidă din întineruperi;

### 5.1.2. Harta memoriei sistemului

DSK include 64k cuvinte memorie SRAM și 256k cuvinte flash ROM pentru încărcare. Decodarea memoriei externe este realizată printr-un circuit CPLD. Programul asociat acestui dispozitiv selectează memoria RAM, flash ROM sau perifericele de pe placă.

#### Memoria de date

Spațiul pentru memoria de date se întinde pe 64k cuvinte (2 pagini de câte 32k cuvinte). Pentru sistemul C5416 DSK memoria externă de date este mapată între adresele 0x8000h și 0xFFFFh. Memoria DARAM (Dual Acess RAM) este mapată în spațiul de date (între adresele 8000h și FFFFh) dacă bitul DROM (bitul 3 din registrul PMST) este setat pe „1”. În caz contrar în acest spațiu este mapată memoria externă.

Hex	Date
0x8000	Memory-Mapped Registers
0x80FF	Scratch-Pad RAM
0x9000	On-Chip DARAM-3 (32k x 16-bit)
0x7FFF	On-Chip DARAM-7 (DROM=1) or External (DROM=0)
0x8000	DARAM0: 0x0000-0x1FFF DARAM1: 0x2000-0x3FFF DARAM2: 0x4000-0x5FFF DARAM3: 0x6000-0x7FFF DARAM4: 0x8000-0x9FFF DARAM5: 0xA000-0xBFFF DARAM6: 0xC000-0xDFFF DARAM7: 0xE000-0xFFFF

Adresa memory fix din chip DARAM in data memory area

Fig. 3. Memoria de date a DSK TMS320VC5416

### Memoria program

Configurarea memoriei program se poate realiza în două moduri folosind bitul 5, OVLY, al registrului PMST. Dacă OVLY=0, memoria RAM on-chip este adresabilă în memoria de date iar dacă OVLY=1, memoria RAM este mapată atât în memoria de date cât și în memoria program, conform următoarei figuri:

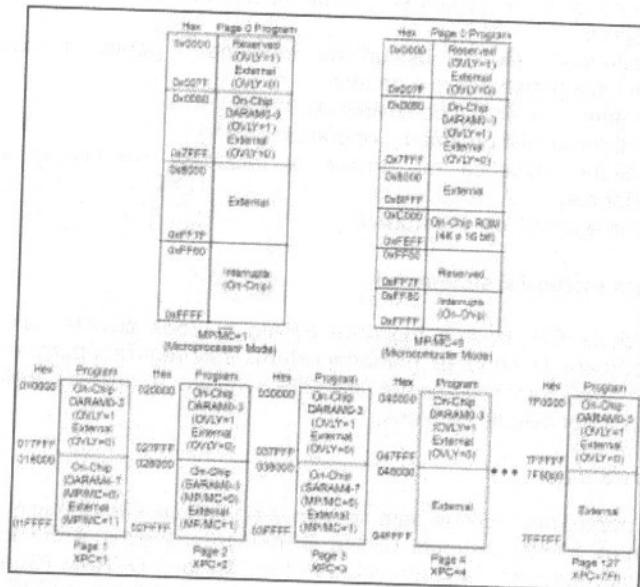


Fig. 4. Memoria program a DSK TMS320VC5416

VC5416 folosește circuitul CPLD (cu 8 registri) pentru a selecta (interfața) memoria flash ROM, memoria SRAM, perifericele de pe placă sau pentru controlul codecului. Cei 8 registri ai CPLD sunt mapăți în spațiul I/O de adrese, începând de la adresa 0x0000h până la adresa 0x0007h.

Avantajele operării cu memoria on-chip ar fi:

- performanță mai ridicată deoarece nu sunt necesare stări de wait
- costuri mai mici decât cele ale memoriei externe
- consum mai redus de putere decât memoria externă
- posibilitatea de a accesa spațiu de adresare mai mare

### 5.1.3. Codec PCM3002

DSK folosește un Codec stereo PCM3002, pentru a asigura intrări și ieșiri analogice. Interfațarea cu codecul se face prin două canale, unul pentru control, iar celălalt pentru date.

Circuitul CPLD al DSK este utilizat pentru interfațarea cu canalul de control prin cei 2 registri pe 8 biți în spațiul I/O, CODEC\_L și CODEC\_H, prin care se transmit comenzi pe 16 biți codecului.

După trimiterea cuvântului de control, CPLD va transmite automat datele, serial, interfeței de control a CODEC-ului, prin semnalele de control master. Semnalele de control serial master ale CODEC-ului sunt: CODEC\_MC (master clock), CODEC\_MD (master data) și CODEC\_ML (master load).

În plus, CPLD generează toate semnalele de tact pentru PCM3002 printr-un oscilator de clock și prin registrul de control CODEC\_CLK. Valoarea implicită a ceasului sistem este de 12.288 MHz. CPLD folosește ceasul sistem al CODEC-ului pentru a genera un semnal de tact de 3.0122 MHz și un semnal de sincronizare cadre de 48 kHz. Registrul CODEC\_CLK permite programatorului să modifice frecvența de eșantionare prin controlul ratei de divizare a frecvenței CODEC-ului PCM 3002. Datele sunt transmise codecului prin interfața VC5416, McBSP2.

Un bit din registrul MISC al CPLD va permite registrului McBSP2 să fie rutat la conectorul de expansiune HPI; calea de bază este asigurată prin Codecul PCM3002.

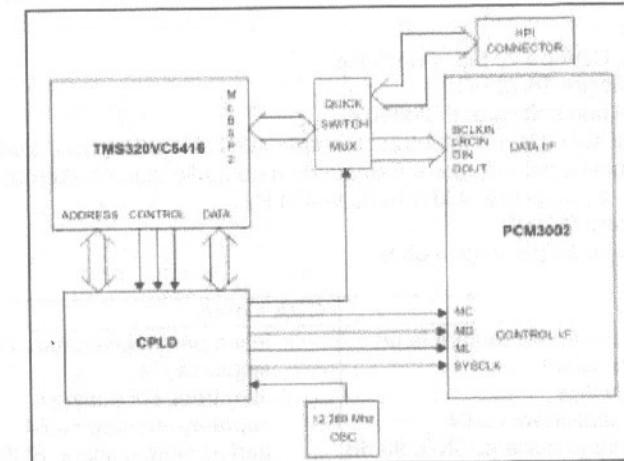


Fig. 5. Interconexiunile dintre PCM3002, CPLD și DSP

PCM3002- Interfața de control are 4 registri interni pe 16 biți care sunt controlați software printr-o interfață serială.

### Programarea interfeței de Date

Interfața de Date a PCM3002 este conectată la pinii McBSP2 ai VC5416.

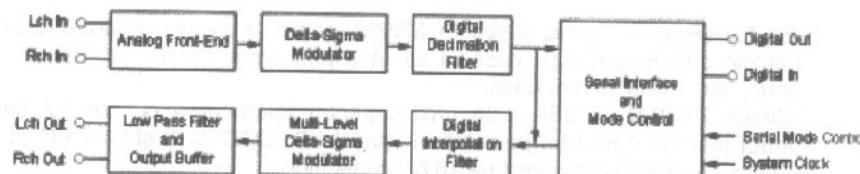


Fig. 6. Codicul PCM3002

**PCM3002** este un codec audio cu un singur cip (convertoare analog-digite și digital-analogice). CAD și CDA implică modulație delta-sigma cu rata de supræșantionare egală cu 64. CAD include un filtru de decimare digital, iar CDA implică un filtru de interpolare digital pentru supræșantionarea cu 8. CDA include de asemenea atenuare digitală, detecție zero infinită, și funcția soft mute, pentru a forma un subsistem complet. PCM3002 oferă un mod de putere care funcționează cu ambele convertoare independențial.

PCM3002 este fabricat folosind un proces avansat CMOS, și sunt disponibile într-un pachet SSOP pe 24 de pini. Sunt potrivite pentru o gamă largă de aplicații, pentru care costurile sunt importante, aplicații pentru care performanța este o prioritate.

Pentru PCM3002, funcțiile programabile sunt controlate prin software.

#### Funcții:

- CAD și CDA pe 20 biți, monolitice
- intrare/ieșire 16/20 biți
- Control prin software: PCM3002
- Funcții speciale (PCM3002): atenuare digitală (256 pași); soft mute; digital loopback; patru formate de date audio digitale alternative;
- Rata de eșantionare: 4 kHz până la 48 kHz
- Alimentare la 3 V
- Aplicații audio portabile/mobile

CAD stereo	CDA stereo
- intrare pentru alimentare la un singur capăt	- ieșire pentru alimentare cu un singur capăt
- filtru antialias	- filtru trece jos analogic
- supræșantionare cu 64	- supræșantionare cu 64
- performanțe ridicate: SNR: 90 dB	- performanțe ridicate: SNR: 94 dB

Funcțiile speciale ale codicului PCM3002 sunt controlate utilizând 4 registri fiecare de câte 16 biți, vezi Anexa 1.

Placa DSK are 16 conectori prin care se realizează accesul utilizatorilor la placă. Conectorii, mărimea acestora, funcția fiecărui, sunt prezentate în tabelul de mai jos, iar descrierea lor în Anexa 2.

### 5.2. Aplicații

Aplicațiile prezентate în continuare se vor realiza folosind mediul Code Composer Studio- C5416 Simulator. După descrierea pașilor necesari pentru a crea un nou proiect de simulare, pentru fiecare aplicație în parte se vor descrie concepțele teoretice de bază folosite. De asemenea vom prezenta formulele utilizate, schemele bloc ale aplicațiilor, sursele programelor și reprezentările grafice ale semnalelor, atât înainte cât și după procesare.

#### 5.2.1. Etapele necesare pentru a crea un nou proiect de simulare

Procedeul următor a fost scris pentru a instala aplicația *template*, care este stocată în fișierul *template.zip*. Pentru alte proiecte, schimbați numele *template* în cel al fișierului zip folosit (de exemplu *delay*).

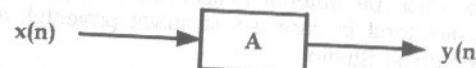
1.	Start Code Composer Studio pentru TMS320C5416 DSK.
2.	Select Project → New. Pentru Project Name, tastează cuvântul <i>template</i> . Click pe butonul Finish. Un nou proiect a fost creat. Rețineți directorul în care a fost salvat proiectul, apoi minimizați Code Composer Studio.
3.	Folosind Windows Explorer, mergeți în directorul unde a fost salvat noul proiect, de exemplu C:\ti\myprojects\template
4.	Copiați fișierul <i>template.zip</i> în directorul C:\ti\myprojects\template
5.	Extrageți fișierele din <i>template.zip</i> folosind WinZip la directorul curent cu opțiunea Extract Here.
6.	Întoarciți-vă la Code Composer Studio. Select Project → Add Files to Project. Selectați calea C:\ti\myprojects\template. Folosiți Control+click mouse stânga pentru a evidenția toate fișierele .c. Click pe butonul Open pentru a le adăuga la proiect.
7.	Selectați Project → Add Files to Project. Folosind săgeata jos în Files la Type box, selectați Linker Command File (*.cmd). Folosiți click mouse stânga pentru a evidenția fișierul <i>template.cmd</i> . Click pe butonul Open pentru a adăuga fișierul <i>template.cmd</i> la proiect.

8.	Selectați Project → Add Files to Project. Folosind săgeata jos în Files la Type box selectați Asm Source Files (*.a*, *.s). Folosiți Control + click mouse stânga pentru a evidenția fișierele .asm (dacă există). Dacă există fișiere .asm în proiect, click pe butonul Open pentru a adăuga fișierele .asm la proiect. Dacă nu, click pe Cancel.
9.	Nu este nevoie sa adăgăm fișierele .h la proiect. Aceasta se face în mod automat.
10.	Selectați Project → Add Files to Project. Mergeți la C5400/cgttools/lib și încărcați fișierul rts.
11.	Selectați Project → Rebuild All. De data aceasta proiectul ar trebui să se construiască cu success și fișierul template.out poate fi încărcat folosind File → Load Program.
12.	În continuare se urmează pașii descriși în lucrarea anterioară.

### 5.2.2. Controlul câștigului

Controlul câștigului este implementat în proiectul numit VOLUME. Avem creată o aplicație care folosește opțiunile oferite de CCS [adaugă fișiere watch sau GEL] pentru a regla un câștig variabil aplicat semnalului de intrare.

Formula folosită în implementare este:  $y(n) = A * x(n)$ ;



Observație: în cazul în care câștigul este egal cu 1, aplicația repetă semnalul de la intrare la ieșire.

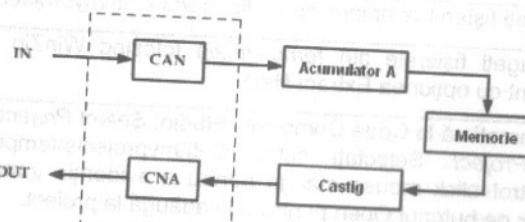


Fig. 7. Schema bloc a aplicației

În continuare se prezintă programul sursă:

```

/* VOLUME.C */

#include <stdio.h>
#include "volume.h"

/* Declarații globale */

int inp_buffer[BUFSIZE]; /* procesarea bufferelor de date */
int out_buffer[BUFSIZE];
int gain = MINGAIN; /* variabila de control a volumului */
unsigned int processingLoad = BASELOAD; /* valoarea de incărcare a rutinei de procesare */

/* Funcții */

extern void load(unsigned int loadValue);
static int processing(int *input, int *output);
static void dataIO(void);

/* programul principal */

void main()
{
    int *input = &inp_buffer[0];
    int *output = &out_buffer[0];
    puts("volume example started\n"); /* afișare mesaj */

    /* buclă infinită */
    while(TRUE)
    {
        dataIO();
        /* Citește datele de intrare folosind un probe-point conectat la un fișier gazdă.*/
        #ifdef FILEIO
        puts("begin processing")
        /* afișare mesaj la inceputul procesării */
        #endif
        processing(input, output);
        /* se aplică câștig prin funcția processing */
    }
}

/* Funcția processing: aplică o transformare (înmulțire cu un factor de câștig) unui semnal primit la intrare;
PARAMETRII: adresele buferelor de la intrare și ieșire ;
VALOAREA RETURNATĂ: TRUE.*/

static int processing(int *input, int *output)
{
    int size = BUFSIZE;
    /* mărimea memoriei tampon */
    while(size--)
        /* pentru toate eșantioanele bufferului... */
        *output++ = *input++ * gain;
}
  
```

```

/* se calculează ieșirea ca fiind intrarea înmulțită cu un
   coeficient, și se incrementează pointerele bufferelor */
}
load(processingLoad);
return(TRUE);
}

/* Funcția dataIO: citește semnalul de la intrare și scrie
semnalul procesat la ieșire.*/

static void dataIO()
{
    return;
}

/********************* */

```

Aplicând un câștig de 10 vom obține următorul rezultat:

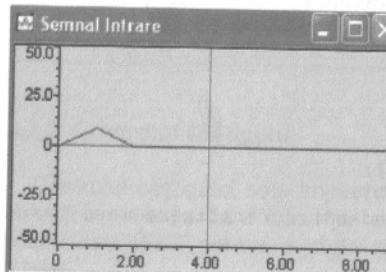


Fig. 8. Semnal intrare

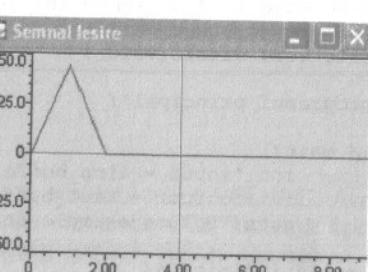


Fig. 9. Semnal ieșire

### 5.2.3. Întârzierea

Un alt fenomen des întâlnit în procesarea semnalelor este cel al întârzierii semnalului. Aceasta poate să apară datorită condițiilor nefavorabile, imperfecțiunilor canalului sau datorită diferitelor tehnici aplicate semnalului.

Formula de calcul, schema bloc și sursa aplicației sunt prezentate în continuare:

$$y(n) = x(n-d);$$

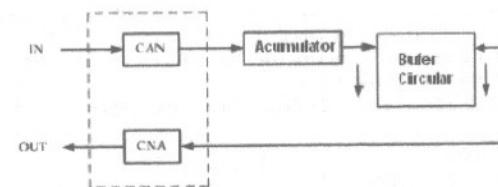
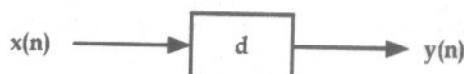


Fig. 10. Schema bloc a aplicației

/\* DELAY . C \*/

```

#include <stdio.h>
#include "delay.h"

/* Declarații globale */
int inp_buffer[BUFSIZE];
/* procesarea buferelor de date */
int out_buffer[BUFSIZE];
int delay = MINGAIN;
/* variabila de control a volumului */

/* Funcții */

extern void load(unsigned int loadValue);
static int processing(int *input, int *output);
static void dataIO(void);

/* programul principal */
void main()
{
    int *input = &inp_buffer[0];
    int *output = &out_buffer[0];

    puts("Delay example started\n");
    /* afișare mesaj */

    /* buclă infinită */
    while(TRUE)
    {
        dataIO();           /*Citește datele de intrare folosind un
                           probe-point conectat la un fișier gazdă. Scrie datele de
                           ieșire într-un graf conectat printr-un probe-point.*/
        #ifdef FILEIO
        puts("begin processing")
        /* afișare mesaj la începutul procesării */
        #endif

        /* se aplică întârzierea folosind funcția processing */
        processing(input, output);
    }
}

```

```

/* Funcția processing: aplică o transformare (atenuare și
întârziere) unui semnal primit la intrare.
PARAMETRII: adresele buferelor de la intrare și ieșire.
VALOAREA RETURNATĂ: TRUE.*/
static int processing(int *input, int *output)
{
int size = BUFSIZE;
/* mărimea memoriei tampon */
while(size--)
{
    /* pentru toate eșantioanele bufferului*/
    /* dacă elementele cerute se află în memorie*/
    *output = *(input+delay);
    /* intrarea întârziată se reproduce la ieșire */
    else
    *output = 0;
    /* altfel se transmite 0 */

    output++;
    /*se incrementează poziția bufferelor */
    input++;
}
return(TRUE);
}

/*Funcția dataIO :cîtește semnalul de la intrare și scrie semnalul procesat la ieșire.
PARAMETRII: nu are; VALOAREA RETURNATĂ: nu are.*/
static void dataIO()
{
return;
}

/*****************/

```

Pentru același semnal de intrare, se obține la ieșire urmatorul semnal, pentru o întârziere egală cu 5:

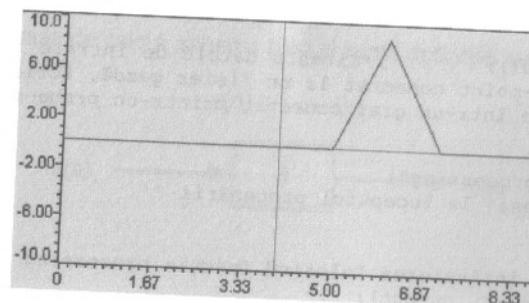


Fig. 11. Semnal ieșire

#### 5.2.4. Reverberația

**Reverberația** este persistența sunetului într-un anumit spațiu după ce sunetul original este înălțat. Când sunetul este produs într-un spațiu, un număr mare de ecouri se formează și apoi scad încet, în timp ce sunetul este absorbit de perete și de aer, creând astfel reverberații. Acest lucru este observat în special când sursa sunetului se oprește iar reflecțiile continuă, micșorându-se în amplitudine, până când nu mai pot fi auzite.

Formula folosită în implementarea software a reverberației este:

$$y(n) = x(n-d) + 1/2 \cdot x(n-2d) + 1/4 \cdot x(n-3d) + 1/8 \cdot x(n-4d);$$

**Exercițiu:** Plecând de la programele anterioare să se realizeze un proiect care să implementeze efectul de reverberație respectând următoarea schema bloc:

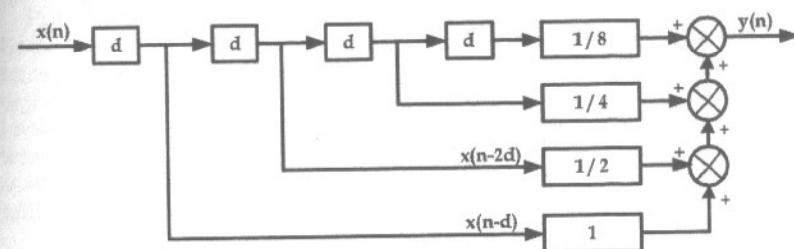


Fig. 12. Schema bloc a aplicației

Diagrama semnalului:

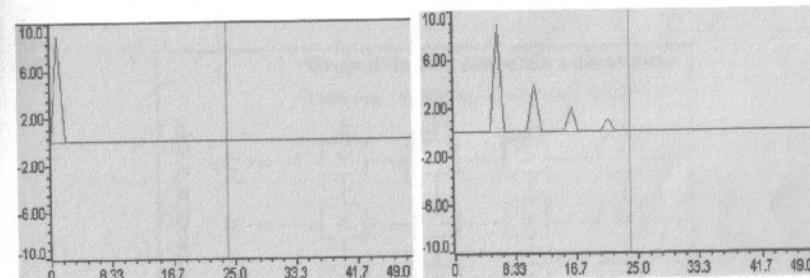


Fig. 13. Semnal intrare

Fig. 14. Semnal ieșire



## 6. DEZVOLTAREA APICAȚIILOR PENTRU SISTEMUL DSK TMS320C5416

Lucrarea prezintă două aplicații realizate folosind mediul Code Composer Studio: detecția tonurilor DTMF utilizând algoritm Goertzel și crearea unor efecte sonore speciale. În prima parte sunt prezentate metodele și algoritmii care stau la baza implementării lor iar apoi sunt descrise programele sursă. Ambele proiecte sunt rulate pe placă DSK TMS320C5416.

### 6.1 Algoritm Goertzel

#### 6.1.1 Detecția tonurilor DTMF

Primul telefon cu tastatură bazată pe tonuri a fost instalat în 1963. Semnalizarea DTMF (Dual Tone Multiple Frequency) folosește tonuri în banda de voce pentru a trimite semnalele de adresa precum și alte informații digitale. Tehnica DTMF este predominant folosită pentru setarea telefoanelor digitale cu butoane reliefate care sunt o alternativă a setării telefoanelor rotative. A fost extinsă mai nou și la poșta electronică sau la aplicațiile bancare prin intermediul telefonului, în care utilizatorii selectează opțiuni din meniu trimițând semnale DTMF de pe telefon.

Într-un sistem de semnalizare DTMF o combinație dintre două tonuri de frecvență reprezintă o cifră specifică, un caracter (A, B, C sau D) sau un simbol (\*, #), după cum se poate observa în urmatoarea figură. De exemplu, tonul tastei 5 este generat simultan de frecvențele 770Hz și 1336Hz, conform figurii 2.

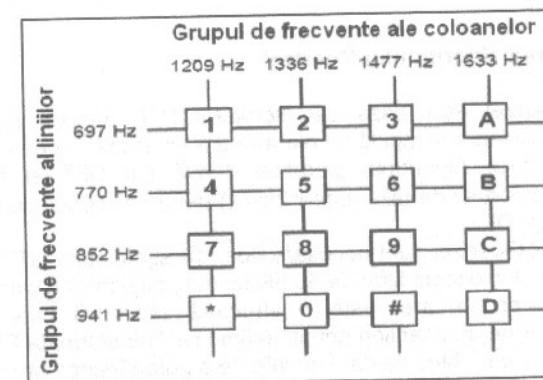


Fig. 1. Frecvențele DTMF și asignarea caracterelor

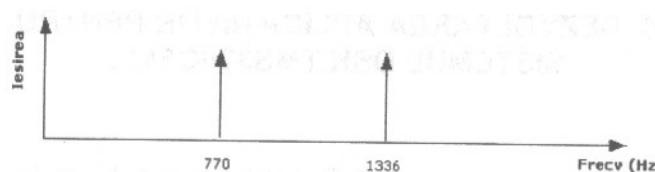


Fig. 2. Tonul Tastei 5

Sistemul DTMF încorporează:

- un codor care traduce apăsarea unei taste sau informația unei cifre în semnale de ton dual;
- un decodor care detectează prezența și conținutul informației semnalelor de ton DTMF primite.

Generarea tonurilor duale se realizează conectând două surse sinusoidale în paralel. Una din metodele folosite în implementare este aproximarea polinomială. Semnalele DTMF trebuie să aibă următoarele caracteristici: rata de eșantionare 8kHz, se transmit 10 digiti pe secundă, durata unui ton mai mare de 40ms.

Sarcina de a detecta tonuri DTMF într-un semnal primit și de a le converti în cifre este mult mai complexă decât procesul de codare. Procesul de decodare este prin natura sa un proces continuu, acest fapt însemnând că trebuie să se verifice în permanență dacă tonurile DTMF sunt prezente în semnalul de date în curs de primire.

Detectarea tonurilor se poate face utilizând mai multe filtre sau folosind Transformata Fourier Discreta (DFT sau FFT). Algoritm Goertzel s-a dovedit a fi mult mai eficient pentru acest tip de aplicație decât celelalte metode amintite.

### 6.1. 2 Descrierea algoritmului Goertzel

Algoritm Goertzel este baza detectorului DTMF. Acesta reprezintă o metodă foarte eficientă și rapidă de extragere a informației spectrale dintr-un semnal de intrare. Algoritm Goertzel derivă din DFT și evidențiază periodicitatea factorului de fază  $\exp(-j*2\pi k/N)$  pentru a reduce complexitatea calculului asociat DFT.

Cu algoritm Goertzel sunt necesare doar 16 eșanțioane DFT pentru 16 tonuri. Algoritm folosește filtre de tip IIR cu doi poli pentru a calcula efectiv valorile DFT. Prin urmare, este o structură recursivă care operează întotdeauna doar pe un eșantion primit, în timp ce Transformata Fourier (sau FFT) are nevoie de un bloc de date înainte de a putea începe procesarea.

Pentru detecția efectivă de ton, informația de magnitudine (aici pătratul magnitudinii) a DFT-ului este suficientă. După un anumit număr de eșanțioane N (echivalent cu dimensiunea unui bloc DFT), ieșirea filtrului

Goertzel converge spre o pseudo valoare DFT  $v_k(n)$ , care poate fi apoi folosită pentru a determina pătratul magnitudinii. În continuare se prezintă o scurtă descriere matematică precum și schema bloc a algoritmului:

#### 1. Calculul recursiv a pseudo-valorilor DFT pentru $n = 0, N$

$$v_k(n) = \underbrace{2\cos\left(\frac{2\pi}{N}k\right)}_{q} \cdot v_k(n-1) - v_k(n-2) + x(n)$$

unde:  $v_k(-1) = 0; v_k(-2) = 0; x(n)$  - eșanțioane intrare

#### 2. Calculul magnitudinii:

$$\begin{aligned} |X(k)|^2 &= y_k(N) \cdot y_k^*(N) \\ &= v_k^2(n) + v_k^2(N-1) - 2\cos(2\pi f_k / f_s) v_k^2(N) v_k^2(N-1) \end{aligned}$$

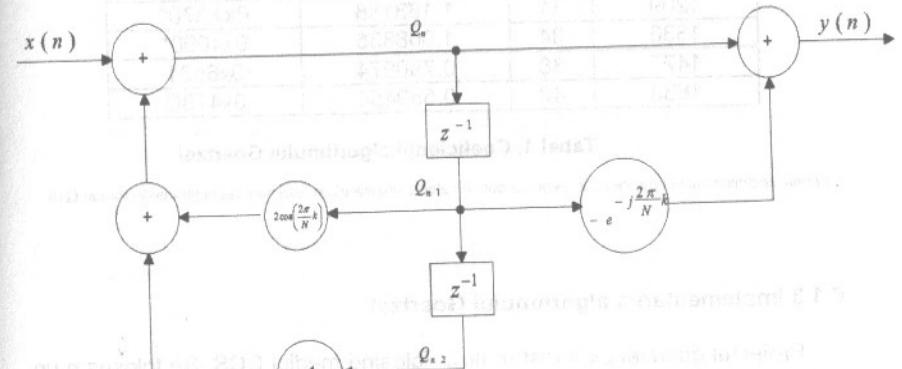


Fig. 3. Schema bloc a algoritmului Goertzel

În esență algoritm Goertzel este o metodă foarte rapidă de a calcula valorile DFT în anumite condiții. Are două avantaje:

- periodicitatea fazelor;
- doar un anumit set din valorile spectrale ale unui DFT sunt necesare (în această aplicație avem tonuri pentru 8 rânduri/coloane)

Parametrul N definește numărul de iterații recursive și de asemenea furnizează mijloacele pentru a acorda rezoluția frecvenței. Valoarea lui k determină tonul pe care încercăm să-l detectăm și este dată de expresia :

$$k = Nx \frac{f_k}{f_s}$$

unde: -  $f_k$  = frecvența tonului

-  $f_s$  = frecvența de eșantionare

- N este fixat la 205

Apoi putem calcula coeficienții ( $\cos(2\pi f_k/f_s)$ ) corespunzători frecvențelor utilizate:

Frecvență	k	Coeficienții (zecimal)	Coeficienții (Q15)
697	18	1.703275	0x6D02*
770	20	1.635585	0x68AD*
852	22	1.562297	0x63FC*
941	24	1.482867	0x5EE7*
1209	31	1.163138	0x4A70*
1336	34	1.008835	0x4090*
1477	38	0.790074	0x6521
1633	42	0.559454	0x479C

Tabel 1. Coeficienții algoritmului Goertzel

\* valorile zecimale sunt împărțite cu 2 (pentru a obține valori subunitare) iar apoi sunt reprezentate în format Q15

### 6.1.3 Implementarea algoritmului Goertzel

Proiectul goertzel.pjt a fost realizat folosind mediul CCS. Se folosește un microfon pentru a recepta sunetele generate de tastele telefonului. Butoanele apăseate sunt identificare folosind algoritmul Goertzel și valorile lor sunt afișate în fereastra de ieșire a CCS. În figura următoare este prezentată organigramă aplicației iar apoi este descrisă metoda implementată. Sursa aplicației goertzel.c poate fi urmărită în Anexa 1.

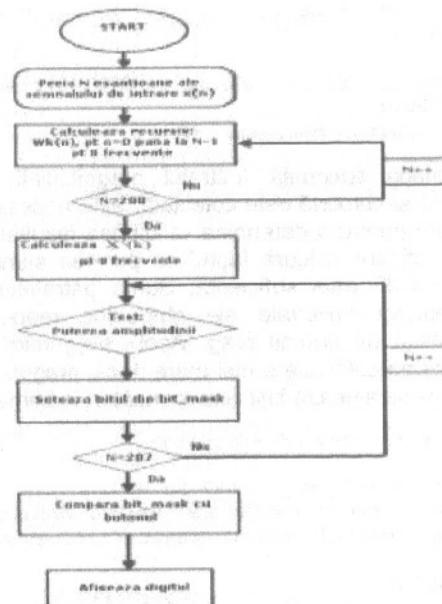


Fig. 4. Organigramă algoritmului Goertzel

Partea principală a acestui detector DTMF este implementată în metoda UserTask(). Mai întâi trebuie inițializat codecul în linia:

```
hCodec = DSK5416_PCM3002_openCodec(0, &setup);
```

apoi programul intră într-o buclă și în fiecare iterație citește semnalele de la canalele de intrare și apoi le trimită la ieșirea codec-ului.

```
while (!DSK5416_PCM3002_read16(hCodec, &left_input));
while (!DSK5416_PCM3002_write16(hCodec, left_output));
while (!DSK5416_PCM3002_read16(hCodec, &right_input));
while (!DSK5416_PCM3002_write16(hCodec, right_output));
```

Programul citește apoi comutatoarele de pe DSK și le afișează valoarea în fereastra de ieșire a CCS; generează de asemenea o intrare mono de la intrarea stânga și dreapta.

```
switch_value = switch_status_display();
mono_input = stereo_to_mono(left_input, right_input);
```

Se verifică apoi starea comutatoarelor. De exemplu, dacă comutatorul este fixat pe valoarea 9, calculează partea recursivă  $V_k(n)$  pentru  $n=1..N$ .

```
if ( 9 == switch_value) {
```

```

goertzel_filter( &delay_697_Hz[0], mono_input,
COEFFICIENT_697_Hz);
.....
goertzel_filter( &delay_1633_Hz[0], mono_input,
COEFFICIENT_1633_Hz);
N++; //incrementeaza N

```

Odată ce informația spectrală (pătratul magnitudinii) pentru fiecare frecvență de pe rând și coloană este colectată, este necesar ca o serie de teste să fie executate pentru a determina validitatea rezultatelor tonului și a cifrelor. O primă verificare asigură faptul că puterea semnalului posibilei perechi de tonuri DTMF este suficientă. Suma pătratelor magnitudinilor vârfurilor componentelor spectrale ale rândurilor, respectiv coloanelor trebuie să depăsească un anumit prag. Apoi se verifică pentru fiecare frecvență și dacă puterea calculată e mai mare decât pragul. În cazul în care rezultatele sunt valide se setează biți corespunzători în bitmask:

```

if ( 199 == N) //First row of keyboard?
{
    goertzel_output_697_Hz =
calculate_goertzel_output( &delay_697_Hz[0], //calculeaza
puterea de iesire Goertzel pentru aceasta frecventa.
COEFFICIENT_697_Hz);
    if ( goertzel_output_697_Hz >
GOERTZEL_THRESHOLD) /* daca rezultatul este mai mare decat
pragul seteaza bitul 0 in mask */
    {
        /*daca este 1 seteaza goertzel_bit_mask |= 0x0001; /* seteaza
bitul 0 in bitmask */
    }
    else
    {
        /*daca este 0 reseteaza goertzel_bit_mask &= 0xFFFF; /* reseteaza
bitul 0 in bitmask */
    }
}

```

Când N =207 programul compară bitmask-ul cu valoarea "butoanelor" declarate în goertzel.h și afișează un mesaj cu numărul butonului apăsat.

```

if ( 207 == N)//afiseaza butonul apasat in Stdout si apoi se
reseteara bitmask
{
    if ( BUTTON_0 == goertzel_bit_mask)
/*compara bitmask-ul cu "butoanele" declarate in goertzel.h*/
    {
        puts("Button 0 pressed.\n");
    }
    N = 0;
    goertzel_bit_mask = 0;
}

```

Apoi valoarea lui N și bitmask-ul sunt resetate pentru a face o nouă citire.

## 6.2 Crearea unor efecte sonore speciale

### 6.2.1 Considerații teoretice

Proiectul alienvoices.pjt prezintă efectul modulației asupra frecvențelor radio. Aplicația arată cum purtătoarea generează sume și diferențe de frecvențe. Astfel se generează vocile ciudate folosite pentru extratereștri în filmele science fiction și în televiziune.

#### Modulații digitale

Semnalizarea bandă de bază și trece bandă este folosită pentru transmiterea datelor pe canalele fizice cum ar fi cablurile telefonice sau canalele de radio frecvență. Sursa de date (biți sau simboluri) poate fi un fișier sau o formă de undă digitalizată (voce, video...). Semnalul transmite informații despre date, iar caracteristicile lui se schimbă odată cu schimbarea ratei de transmisie a datelor.

Când semnalul care transportă informația:

- se extinde de la 0 Hz în sus, este folosit termenul de **semnalizare bandă de bază**
- are puterea centrată în jurul unei frecvențe centrale  $f_c$ , este folosit termenul de **semnalizare trece bandă** sau **Modulație**

Modulația este folosită deoarece:

- Canalul nu include frecvența de 0 Hz și semnalizarea bandă de bază este imposibilă
- Banda canalului este împărțită în mai multe canale pentru multiplexare în frecvență
- Pentru radio-comunicații, dimensiunea antenei descrește când frecvența transmisiă crește

În schemele simple de modulație purtătoarea este modificată cu aceeași rată cu care se transmit datele. Purtaoarea este de obicei scrisă în modul următor:

$$A \cos(2\pi f_c t + \Phi),$$

unde:  $f_c$  = frecvența purtătoarei

$A$  = amplitudinea purtătoarei

$\Phi$  = fază purtătoarei

Cei trei parametrii principali ai purtătoarei: amplitudine, fază și frecvență pot fi modificați astfel încât să transmită informații conducând la modulație în amplitudine, modulație de fază și modulație de frecvență.

În exemplul următor componentele bandei de bază  $Z_1$  și  $Z_0$  sunt modulate în amplitudine de două purtătoare în quadratură.

$$x(t) = \cos(2\pi f_c t + \Phi(t)) = \cos(\Phi(t))\cos(2\pi f_c t) - \sin(\Phi(t))\sin(2\pi f_c t)$$

$$x(t) = Z_I(t)\cos(2\pi f_c t) - Z_Q(t)\sin(2\pi f_c t)$$

unde:

- $f_c$  = frecvența purtătoarei
- $Z_I(t) = \cos(\Phi(t))$  și  $Z_Q(t) = \sin(\Phi(t))$  - componentele bandei de bază
- $purtatoare_I = \cos(2\pi f_c t)$  și  $purtatoare_Q = \sin(2\pi f_c t)$  - purtătoarele (sunt în general semnale analogice RF, generate de oscilatoare analogice)

## 6.2.2 Prezentarea aplicației

În cadrul aplicației *alienvoices.c* semnalul de intrare, preluat de la un microfon sau de la un CD player, este multiplicat cu un semnal sinusoidal pentru generarea sumei și a diferenței de frecvențe și apoi este trimis la ieșire spre difuzoare.

Modularea frecvenței purtătoare este simulață cu ajutorul funcției static `short int ring_modulation (short int input1, short int input2)`, definită în fișierul principal *alienvoices.c* (vezi Anexa 2). Această funcție realizează modulație circulară prin înmulțirea intrării cu semnalul sinusoidal generat.

Generarea semnalului sinusoidal este realizată în fișierul *sinewave.c* studiat în lucrarea anterioară. Fișierul generează două semnale sinusoidale cu frecvență reglabilă folosind funcția *sine()* din *dsplib.lib*.

Cele două funcții:

- `signed int generate_sinewave_1 (signed short int frequency, signed short int amplitude)`

- `signed int generate_sinewave_2 (signed short int frequency, signed short int amplitude)`

primesc ca prim parametru frecvența semnalului sinusoidal (între 10 Hz și 16000Hz), în timp ce al doilea parametru este amplitudinea maximă a sinusului (între 1 și 32767). În această aplicație toate semnalele sinusoidale au o amplitudine de 32767.

Ambele canale (drept și stâng) sunt modulate și apoi rezultatul este trimis spre ieșire la canalul corespunzător.

Pentru unele valori de comutare, rezultatul modulației circulare este apoi filtrat cu un filtru trece sus și/sau trece jos de ordinul patru. Procesul de filtrare este implementat în fișierul *IIR\_filters\_fourth\_order.c* și este descris în aplicația *iir.c*.

Organograma programului este prezentată în figura următoare:

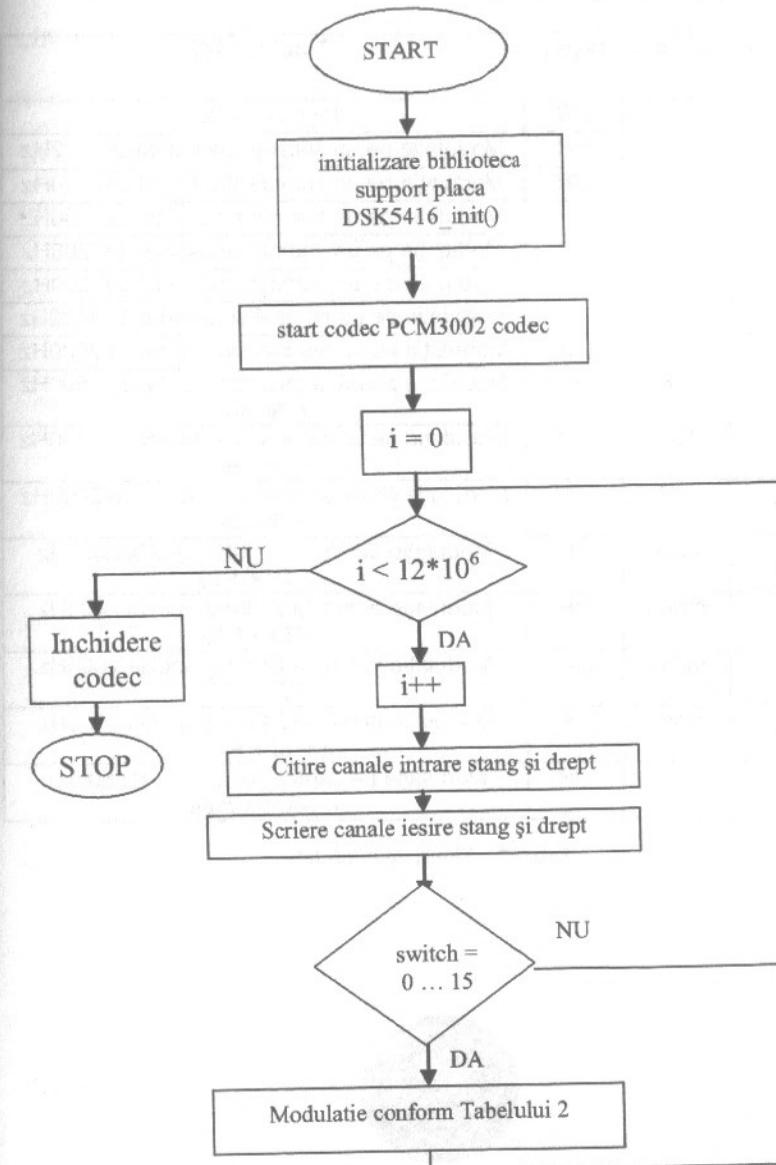


Fig. 5. Organograma aplicației „alienvoices”

Pentru diferite valori de comutare, se pot obține următoarele tipuri de modulații (L/R – stânga/dreapta):

Valoare switch	Intrare	Ieșire	Tipul filtrării
0	L/R	L/R	Ieșire = intrare
1	L/R	L/R	Modulație pe un semnal sinusoidal de 2Hz
2	L/R	L/R	Modulație pe un semnal sinusoidal de 20Hz
3	L/R	L/R	Modulație pe un semnal sinusoidal de 100Hz
4	L/R	L/R	Modulație pe un semnal sinusoidal de 200Hz
5	L/R	L/R	Modulație pe un semnal sinusoidal de 500Hz
6	L/R	L/R	Modulație pe un semnal sinusoidal de 1000Hz
7	L/R	L/R	Modulație pe un semnal sinusoidal de 2000Hz
8	L/R	L/R	Modulație pe un semnal sinusoidal de 600Hz + filtrare
9	L/R	L/R	Modulație pe un semnal sinusoidal de 1000Hz + filtrare
10	L/R	L/R	Modulație pe un semnal sinusoidal de 2000Hz + filtrare
11	mono	L/R	Modulație dublă, la 2000Hz apoi la 2400Hz + FTS + FTJ
12	mono	L/R	Modulație dublă, la 2000Hz apoi la 2400Hz + FTJ + FTS
13	mono	L/R	Modulație dublă, la 2400Hz apoi la 2000Hz + FTS + FTJ
14	mono	L/R	Modulație dublă, la 2400Hz apoi la 2000Hz + FTJ + FTS
15	mono	L/R	Modulație pe semnal sinusoidal de 500Hz + reverberație

Tabel 2 – Tipuri de modulație



## ANEXA 1- Structura fișierului LINK25.CMD

### Linker Command File

#### Microcomputer Mode MC/MP = 1

- setează modul de funcționare Microcomputer/Microprocesor  
1K RAM block mapped into program space

- avem mapat 1K de memorie RAM în spațiul de memorie program  
RAM, OVLY bits = 1,0

#### Block B0 configured as program memory

- blocul 0 este configurat ca și memorie program

#### ST1 - CNF BIT = 0 MEMORY

- specifică setarea hărții de memorie a sistemului

{

#### PAGE 0 : /\* Program Memory

- Memoria program (Pagina0)

#### VECS : origin =0h ,length =20h interrupts

- setează zona de memorie unde sunt localizăți vectorii de întrerupere  
PROG\_ROM : origin =20h ,length = 0F90h ROM

- setează zona de program din memoria ROM

#### PROG\_RAM : origin = 0FB0h,length = 0050h RAM /\*

- setează zona de program din memoria RAM

/\* setează zona de program din memoria externă

#### EXT\_PROG : origin =1000h ,length = 0C400h off-chip

#### PAGE 1 : /\* Data Memory Memoria de date (Pagina1)

/\* setează zona de memorie unde sunt localizăți registri mapăți în mem  
REGS : origin = 0h ,length = 6h

/\* localizare blocul 2 de memorie

#### BLK\_B2:origin = 60h,length = 20h Block B2

/\* setarea mărimea și localizarea zonei de RAM interne

#### INTL\_RAM : origin = 200h,length =400h Block B0 & B1

/\* setarea mărimea și localizarea zonei de ram extern

#### EXT\_DATA : origin = 800h,length = 0F800h off-chip

}

#### SECTIONS - secțiune de setare specifică formatului COFF

{

.vectors: { } > VECS PAGE 0 /\* secțiunea .vectors în Pagina 0

.text : { } > PROG\_ROM PAGE 0 /\*secțiunea .text în Pagina 0 în zona  
memoriei ROM interne

.data : { } > PROG\_ROM PAGE 0 /\* secțiunea .data în Pagina 0 în  
zona memoriei ROM interne

.bss : { } > EXT\_DATA PAGE 1 /\* secțiunea .bss în Pagina 1 în  
zona memoriei externe

}

0-1	1-0	LEP	0-0	1-1	0-0	1-1	0-0	1-1	0-0	1-1	0-0	1-1	0-0	1-1	0-0	1-1
0-0 (MAGNET)	0-0 (MAGNET)															

**ANEXA 2****PCM3002 – descrierea regiștrilor**

Există 4 regiștri, iar biți 9 și 10 determină care registru este utilizat:

	A1	A0	Registru
TRA-0RA	0	0	Registru 0
	0	1	Registru 1
	1	0	Registru 2
	1	1	Registru 3

**Registru 0 - DAC Attenuation Data Left Channel**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res						A1	A0	LDL							AL0-AL7

Biți	Denumire	Valoare implicită	Descriere
B15-B11	Res	00000	Reservati
B10	A1	0	Adresa Registrului AD-TRA-0RA
B9	A0	0	Adresa Registrului AD-TRA-0RA
B8	LDL		DAC Attenuation Data Load Control for Left Channel
B7-B0 (în AL0-AL7)	1111 1111		DAC Attenuation Data for Left Channel

**LDL (DAC Attenuation Data Load Control for Left Channel)** - bit utilizat pentru a seta simultan ieșirile analogice a canalului din stânga, respectiv dreapta.

- 1 nivelul ieșirii este controlat de datele de atenuare AL (7:0)
- 0 noile date de atenuare vor fi ignorate și nivelul ieșirii va rămâne la nivelul anterior de atenuare.

**Observație:** Bitul LDR din Registrul 1 are funcții echivalente cu ale LDL. Când LDL sau LDR este setat pe "1", nivelele de ieșire a canalelor dreapta și stânga sunt controlate simultan. **ATT** (DAC Attenuation Data for Left Channel) - nivelul de atenuare (ATT) este dat de:

$$ATT = 20 \cdot \log_{10} \left( \frac{ATT \text{ Data}}{255} \right) \text{ (dB)}$$

<b>AT (7:0)</b>	00h	01h	FEh	FFh
<b>Nivel atenuare</b>	- ∞ dB (Mute)	-48.16 dB	-0.03 dB	0 dB (default)

**Registrul 1 - DAC Attenuation Data Right Channel**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	A1	A0	LDR	AR0-AR7											

Biți	Denumire	Valoare implicită	Descriere
B15-B11	Res	00000	Reservat
B10	A1	0	Adresa Registrului
B9	A0	1	Adresa Registrului
B8	LDR	1	DAC Attenuation Data Load Control for Right Channel
B7-B0	AR0-AR7	1111 1111	DAC Attenuation Data for Right Channel

**LDR (DAC Attenuation Data Load Control for Right Channel):-** are aceeași funcție ca și LDL

**AR0-AR7 - DAC Attenuation Data for Right Channel**

Nivelul de atenuare (ATT) este dat de aceeași formulă ca și pentru Registrul 0.

15	11	10	9	8	7	6	5	4	3	2	1	0
res	A1	A0	PDAD	BYP5	PDDA	ATC	IZD	OUT	DEMI	DEM0	MUT	

**PDAD (ADC Power-Down Control):** 0 - modul Power Down dezactivat  
1 - modul Power Down activat

Când este setat pe "0", atât ieșirea de pe canalul de stânga, cât și ieșirea de pe canalul de dreapta ale CDA sunt puse pe mute în același timp. Acest procedeu are loc prin atenuarea datelor din filtrul digital.

**BYP5 (ADC High-Pass Filter Bypass Control):** 0 - activat  
1 - dezactivat (Bypassed)

**PDDA (ADC POWER DOWN):** 0 - modul Power-Down dezactivat  
1 - modul Power-Down activat

Setează CDA în modul de consum redus de putere. Operațiile CDA sunt întrerupte prin oprirea alimentării cu curent a secțiunii CDA, VOUT fiind legat la GND în timpul activării acestui mod.

**ATC (DAC Attenuation Channel Control):**

0 - Individual Channel Attenuation Data Control

1 - Common Channel Attenuation Data Control

Când este setat pe "1", datele de atenuare ale Registrului 0 pot fi folosite pentru ambele canale ale CDA. În acest caz, datele de atenuare ale Registrului 1 sunt ignorate.

**IZD (DAC Infinite Zero Detection Circuit Control):**

0 - detectie zero infinit dezactivată

1 - detectie zero infinit activată

Acest bit activează circuitul de detecție zero infinit al PCM3002: când este activ, acest circuit va deconecta ieșirea analogică a amplificatorului de la CAD delta-sigma dacă intrarea este zero continuu pentru 65.536 de cicluri consecutive ale BCKIN.

**OUT (DAC Output Enable Control):**

0 - ieșirile CDA active (operărie normală)

1 - ieșirile CDA dezactivate

Când sunt setate pe "1", ieșirile sunt forțate la V CC /2 (zero bipolar). În acest caz, toți registrii din PCM3002 rețin datele curente. Când sunt setate pe "0", ieșirile se întorc la stările programate anterior.

**DEMI/DEM0 (DAC De-emphasis Control)**

DEMI	DEM0	De-emphasis
0	0	44.1kHz ON
0	1	OFF (default)
1	0	48kHz ON
1	1	32kHz ON 3

**MUT (DAC Soft Mute Control):** 0 - mute dezactivat (implicit)

1 - mute activ

Când este setat pe "1", atât ieșirea de pe canalul de stânga, cât și ieșirea de pe canalul de dreapta ale CDA sunt puse pe mute în același timp. Acest procedeu are loc prin atenuarea datelor din filtrul digital.

**Registrul 3 - Audio Data Control (valoare implicită 0600h)**

15	11	10	9	8	6	5	4	3	2	1	0
res	A1	A0	res	LOP	res	FMT1	FMT0	LRP	res		

**LOP (ADC to DAC Loop-Back Control):** 0 - loop-back dezactivat (default)  
1 - loop-back activ

Când acest bit este setat pe "1", datele audio ale CAD sunt trimise direct către CDA. Formatul datelor va fi în standardul I<sup>2</sup>S.

#### FMT1/FMT0 (Audio Data Format Select)

FMT1	FMT0	Format
0	0	16-bit, MSB first (default), DAC R-justified, ADC L-justified
0	1	20-bit, MSB first, DAC R-justified, ADC L-justified
1	0	20-bit, MSB first, Left-justified
1	1	20-bit, MSB first, I <sup>2</sup> S

**LRP (Polaritatea LRCIN):** - se aplică doar formatelor 0, 1 și 2.  
0 - canalul stânga este "H", canalul-dreapta este "L".  
1 - canalul stânga este "L", Canalul-dreapta este "H".

## ANEXA 3

### Conectori de expansiune ai sistemului DSK TMS320C5416

Connector	#Pins	Function
P1	80	Memory
P2	80	Peripheral
P3	80	HPI
J1	2	Microphone
J2	2	Line In
J3	2	Line Out
J4	2	Speaker
J5	4	Optional Power Connector
J6	2	+5 Volt
J7	14	External JTAG
J201	5	USB JTAG
JP1	10	CPLD Programming
JP4	8	DSP Configuration Jumper

Tabelul 1. Conectorii placii TMS320VC5416 DSK

DSK TMS320VC5416 conține trei conectori de expansiune care urmează standardele Texas Instruments:

**P1 - Conector de expansiune a memoriei**

**P2 - Conector de expansiune pentru periferice**

**P3 - Conector de expansiune HPI**

**J1 - Conector pentru microfon:** microfonul interacționează cu placa PCM3002E printr-un circuit simplu op-amp. Intrarea o reprezintă o mufă jack stereo de 3.5 mm.

**J2 - Conector Audio de intrare:** este o intrare stereo. Intrarea comunică cu circuitul PCM3002E printr-un circuit bias op-amp. Conectorul este reprezentat de mufa jack stereo de 3.5 mm.

**J3 - Conector Audio de ieșire:** este o ieșire stereo, iar comunicarea cu PCM3002E se realizează ca și la conectorul J2. Este reprezentat de mufa jack de 3.5 mm.

**J4 - Conector pentru căști/boxe:** este condus de un mic amplificator TPA302 conectat la codecul placii, PCM3002E.

**J5 - Conector optional de putere:** va opera cu sursa de putere standard a calculatorului.

**J6 - Conector la +5V:** puterea (5V) este adusă pe TMS320VC5416 DSK prin acest conector. Conectorul are un diametru exterior de 5.5 mm și unul interior de 2.5 mm.

**J7 - Conector extern JTAG:** o interfață cu 14 pini a TMS320VC5416 DSK și reprezintă interfață standard folosită de emulatoarele JTAG pentru a interfața cu DSP-urile Texas Instruments.

**JP1 - Conector PLD de programare:** acest conector interacționează cu ALTERA CPLD, U18. Este folosit pentru programarea CPLD.

**J201 - Conector Universal Serial Bus (USB) încastrat JTAG:** Conectorul J201 asigură o interfață USB emulatorului JTAG încastrat (situat) pe DSK. Aceasta permite dezvoltarea softului și debugger-ului fără a utiliza un emulator extern.

**LED-uri:** TMS 320VC5416 are patru LED-uri care pot fi definite de utilizator. Aceste LED-uri sunt utilizate de către POST (Power On Self Test), dar sunt disponibile pentru programele realizate de utilizator. Pot fi accesate prin adresa I/O 0X0000.

**LED-uri de sistem:** placa are 4 LED-uri de sistem ce indică diferențele stării în care se află placa.

**Switch-uri:** TMS320VC5416 are 2 switch-uri, unul Reset, și un switch DIP pe 4 poziții pentru utilizator.

**Reset Switch/Reset Logic:** există 3 Reset-uri pe placă. Primul reset este „power on reset”. Acest circuit așteaptă până când alimentarea este între anumite limite specificate, înainte de a alimenta pinul reset al plăcii TMS320VC5416. Surse externe care controlează reset-ul sunt butonul S1, și emulatorul USB JTAG.

**Switch-ul DIP pe 4 poziții:** DSK TMS320VC5416 are un switch DIP pe 4 poziții, pentru utilizator, S2. Este accesibil prin Registrul 0 CPLD la locația I/O 0X0000.

Setările principale ale switch-ului DIP sunt următoarele:  
 1. Poziția 0: S2=0, S1=0, S0=0. În această configurație, se activează un canal de audio stâng și un canal de audio drept.  
 2. Poziția 1: S2=0, S1=0, S0=1. În această configurație, se activează un canal de audio stâng și un canal de audio drept.  
 3. Poziția 2: S2=0, S1=1, S0=0. În această configurație, se activează un canal de audio stâng și un canal de audio drept.  
 4. Poziția 3: S2=1, S1=0, S0=0. În această configurație, se activează un canal de audio stâng și un canal de audio drept.

## ANEXA 4

### Algoritmul Goertzel – implementare

```
*****
/* goertzel.c */

/* Aplicația folosește algoritmul Goertzel pentru a identifica tonurile de apelare ale unui telefon; rata de eșantionare este 8000 Hz (8 kHz) */

#include <stdio.h>
/* folosit pentru printf() și puts()
#include "goertzelcfg.h"

#include "dsk5416.h"
#include "dsk5416_pcm3002.h"

#include "bargraph.h"
/*folosit pentru afișajul cu LED-uri */
#include "switches.h"
/*întrerupătoarele de pe placă permit selectarea modului de operare (1-9) */
#include "goertzel.h"
/* definire butoane */

#define GOERTZEL_THRESHOLD 0x100
/*Suma puterilor componentelor spectrale ale frecvențelor de pe linii și de pe coloană(generate la apăsarea unui buton) trebuie să fie mai mare decât această limită */

*****
/* Configurarea regiștrilor PCM3002 */
*****
DSK5416_PCM3002_Config setup = {
/* configurarea implicită */

    0x1FF,
/* programare Registrul 0 - atenuarea CDA pentru canalul stâng
   LDL=1 -> nivelul ieșirii este controlat de campul AL
   (biți 7:0)
   AL=FFh -> atenuare canal stâng 0dB */

    0x3FF,
/* programare Registrul 1 - atenuarea CDA pt canalul drept
   LDL=1 -> nivelul ieșirii este controlat de cîmpul AL
   (biți 7:0)
   AL=FFh -> atenuare canal stâng 0dB

    0x0,
```

```

/* programare Registrul 2
Filtrul trece sus al CAD este activat
Modul Power-Down e dezactivat
Modul detecție zero infinit dezactivat
Ieșirile CDA active (operătie normală)
Modul soft Mute dezactivat (implicit) */

    0x0,

/* programare Registrul 3
DAC pe 16 biți, MSB primul
ADC pe 16 biți, MSB primul
Modul Loop-Back dezactivat
canalul stânga este "H", canalul dreapta este "L" */

};

/*********************************************
/* Pentru compatibilitate cu funcțiile read/write ale pcm3002*/
/* aceste variabile trebuie declarate Int16 sau short int și nu
int */
/********************************************/

Int16 left_input;
Int16 left_output;
Int16 right_input;
Int16 right_output;
Int16 mono_input;

/* variabile pentru puterea de ieșire calculată la aceste
frecvențe */

short int goertzel_output_697_Hz;
short int goertzel_output_770_Hz;
short int goertzel_output_852_Hz;
short int goertzel_output_941_Hz;

short int goertzel_output_1209_Hz;
short int goertzel_output_1336_Hz;
short int goertzel_output_1477_Hz;
short int goertzel_output_1633_Hz;

/* variabile folosite pentru calcularea recursivă a
coeficientilor "v" */

short int delay_697_Hz[3] = { 0, 0, 0 };
short int delay_770_Hz[3] = { 0, 0, 0 };
short int delay_852_Hz[3] = { 0, 0, 0 };
short int delay_941_Hz[3] = { 0, 0, 0 };

short int delay_1209_Hz[3] = { 0, 0, 0 };
short int delay_1336_Hz[3] = { 0, 0, 0 };
short int delay_1477_Hz[3] = { 0, 0, 0 };

```

```

short int delay_1633_Hz[3] = { 0, 0, 0 };

short int goertzel_bit_mask = 0; //biții acestei variabile sunt
indicatori ai frecvențelor valide recepționate
***** Această procedură nu este apelată de main(). Ea este apelată de
DSP/BIOS ****

void UserTask()
{
DSK5416_PCM3002_CodecHandle hCodec;
unsigned long i;
unsigned int switch_value;
unsigned int counter = 0;
short int goertzel_power;
unsigned int N = 0;

/* inițializează codecul */
hCodec = DSK5416_PCM3002_openCodec(0, &setup);

/* Afisează detalii la Stdout */
puts("TMS320C5416 DSK: Goertzel algorithm to identify touch
tones of telephone.\n");

for ( i = 0 ; i < 12000000 ; i++ )
{
    // Citește canalul de intrare stanga
    while (!DSK5416_PCM3002_read16(hCodec, &left_input));
    // Redă la canalul de ieșire stanga
    while (!DSK5416_PCM3002_write16(hCodec, left_output));
    // Citește canalul de intrare dreapta
    while (!DSK5416_PCM3002_read16(hCodec, &right_input));
    // Redă la canalul de ieșire dreapta
    while (!DSK5416_PCM3002_write16(hCodec, right_output));
    /* Citește valorile intrerupătoarelor de
       pe placă și afisează valoarea la Stdout */
    switch_value = switch_status_display();
    /* generează intrare mono de la intrările
       stanga și dreapta */
    mono_input = stereo_to_mono(left_input, right_input);

    /* procesează unul din 6 eșantioane e.g. luate la frecvența de
       8 kHz valoarea lui counter asigură că numai unul din cele 6
       eșantioane este prelucrat */

    if (0 == counter)
        if (0 == switch_value) //Valoare intrerupătoare = 0.
            {left_output = mono_input;
             //redare semnale mono în difuzoare
             right_output = mono_input;
            }
}

```

```

else if (1 == switch_value) /* Valoare intrerupătoare = 1.
Filtru Goertzel la 693 Hz. Primul rând tastatura telefon */
{
left_output = goertzel_value(mono_input, COEFFICIENT_697_Hz);
/* Calculează valoarea Goertzel la această frecvență */

right_output = left_output;
/* folosit pentru testarea acurateței frecvenței Goertzel */
}
else if (2 == switch_value)
/* Valoare intrerupătoare = 2. Filtru Goertzel la 770 Hz.
Al doilea rând al tastaturii telefonului */
{
left_output = goertzel_value( mono_input, COEFFICIENT_770_Hz);
/*Calculează valoarea Goertzel la această frecvență

right_output = left_output;
}
else if (3 == switch_value)

/* Valoare intrerupătoare = 3. Filtru Goertzel la 852 Hz.
Al treilea rând al tastaturii telefonului */
{
left_output = goertzel_value( mono_input, COEFFICIENT_852_Hz);
/*Calculează valoarea Goertzel la această frecvență

right_output = left_output;
}
else if (4 == switch_value)
/* Valoare intrerupătoare = 4. Filtru Goertzel la 941 Hz.
Al patrulea rând al tastaturii telefonului */
{
left_output = goertzel_value( mono_input, COEFFICIENT_941_Hz);
/*Calculează valoarea Goertzel la aceasta frecvență
right_output = left_output;
}
else if (5 == switch_value)
/* Valoare intrerupătoare = 5. Filtru Goertzel la 1209 Hz.
Prima coloană a tastaturii telefonului */
{
left_output = goertzel_value( mono_input, COEFFICIENT_1209_Hz);
/*Calculează valoarea Goertzel la această frecvență
right_output = left_output;
}
else if (6 == switch_value)
/* Valoare intrerupătoare = 6. Filtru Goertzel la 1336 Hz.
A doua coloană a tastaturii telefonului */
{
left_output = goertzel_value( mono_input, COEFFICIENT_1336_Hz);
/*Calculează valoarea Goertzel la această frecvență
right_output = left_output;
}
else if (7 == switch_value)

```

```

/* Valoare intrerupătoare = 7. Filtru Goertzel la 1477 Hz.
A treia coloană a tastaturii telefonului*/
{
left_output = goertzel_value( mono_input, COEFFICIENT_1477_Hz);
/*Calculează valoarea Goertzel la această frecvență
right_output = left_output;
}
else if (8 == switch_value)
/* Valoare intrerupătoare = 8. Filtru Goertzel la 1633 Hz.
A patra coloană a tastaturii telefonului (nefolosit)*/
{
left_output = goertzel_value( mono_input, COEFFICIENT_1633_Hz);
/*Calculează valoarea Goertzel la această frecvență
right_output = left_output;
}
else if (9 == switch_value)
/* Valoare intrerupătoare = 9. Filtre Goertzel de la 697 Hz la
1633 Hz. Butoanele 1 la #
{
/* Filtru Goertzel pentru fiecare din frecvențe */

goertzel_filter(&delay_697_Hz[0],mono_input,
COEFFICIENT_697_Hz);

goertzel_filter(&delay_770_Hz[0],mono_input,
COEFFICIENT_770_Hz);

goertzel_filter(&delay_852_Hz[0],mono_input,
COEFFICIENT_852_Hz);

goertzel_filter(&delay_941_Hz[0],mono_input,
COEFFICIENT_941_Hz);

goertzel_filter(&delay_1209_Hz[0],mono_input,
COEFFICIENT_1209_Hz);

goertzel_filter(&delay_1336_Hz[0],mono_input,
COEFFICIENT_1336_Hz);

goertzel_filter(&delay_1477_Hz[0],mono_input,
COEFFICIENT_1477_Hz);

goertzel_filter(&delay_1633_Hz[0],mono_input,
COEFFICIENT_1633_Hz);

N++; //incrementează N

if (199 == N)
//primul rând al tastaturii telefonului?
{
goertzel_output_697_Hz=
calculate_goertzel_output(&delay_697_Hz[0],COEFFICIENT_697_Hz);
/*Calculează puterea de ieșire Goertzel pt această frecvență.

```

```

        if (goertzel_output_697_Hz > GOERTZEL_THRESHOLD)
/* dacă rezultatul este mai mare decât limita specificată
setează bitul 0 în bit_mask */
        {goertzel_bit_mask |= 0x0001; }
// Setează bitul 0 în bit mask
        else {goertzel_bit_mask &= 0xFFFFE; }
// Resetează bitul 0 în bit mask
    }

    else if ( 200 == N)
//Al doilea rând al tastaturii telefonului?
        {goertzel_output_770_Hz =
calculate_goertzel_output(&delay_770_Hz[0], COEFFICIENT_770_Hz);
//Calculează puterea de ieșire Goertzel pt această frecvență.
        if ( goertzel_output_770_Hz > GOERTZEL_THRESHOLD)
            {goertzel_bit_mask |= 0x0002; }
// Setează bitul 1 în bit mask
        else {goertzel_bit_mask &= 0xFFFFD; }
// Resetează bitul 1 în bit mask
    }

    else if ( 201 == N)
//Al treilea rând al tastaturii telefonului?
        {goertzel_output_852_Hz =
calculate_goertzel_output(&delay_852_Hz[0], COEFFICIENT_852_Hz);
//Calculează puterea de ieșire Goertzel pt această frecvență.
        if ( goertzel_output_852_Hz > GOERTZEL_THRESHOLD)
            {goertzel_bit_mask |= 0x0004; }
// Setează bitul 2 în bit mask
        else {goertzel_bit_mask &= 0xFFFFB; }
// Resetează bitul 2 în bit mask
    }

    else if ( 202 == N)
//Al patrulea rând al tastaturii telefonului?
        {goertzel_output_941_Hz =
calculate_goertzel_output(&delay_941_Hz[0], COEFFICIENT_941_Hz);
//Calculează puterea de ieșire Goertzel pt această frecvență.
        if ( goertzel_output_941_Hz > GOERTZEL_THRESHOLD)
            {goertzel_bit_mask |= 0x0008; }
// Setează bitul 3 în bit mask
        else {goertzel_bit_mask &= 0xFFFF7; }
// Resetează bitul 3 în bit mask
    }

    else if ( 203 == N)
//Prima coloană a tastaturii telefonului
    {goertzel_output_1209_Hz=calculate_goertzel_output(delay_1209_
Hz[0], COEFFICIENT_1209_Hz);
//Calculează puterea de ieșire Goertzel pt această frecvență.
        if ( goertzel_output_1209_Hz > GOERTZEL_THRESHOLD)
            {goertzel_bit_mask |= 0x0010; }
// Setează bitul 4 în bit mask
        else {goertzel_bit_mask &= 0xFFEF; }
}

```

```

// Resetează bitul 4 în bit mask
}

else if ( 204 == N)
//A doua coloană a tastaturii telefonului
    {goertzel_output_1336_Hz = calculate_goertzel_output(
&delay_1336_Hz[0], COEFFICIENT_1336_Hz);
//Calculează puterea de ieșire Goertzel pt această frecvență.
    if ( goertzel_output_1336_Hz > GOERTZEL_THRESHOLD)
        {goertzel_bit_mask |= 0x0020; }
// Setează bitul 5 în bit mask
    else {goertzel_bit_mask &= 0xFFDF; }
// Resetează bitul 5 în bit mask
}

else if ( 205 == N)
//A treia coloană a tastaturii telefonului
    {goertzel_output_1477_Hz = calculate_goertzel_output(
&delay_1477_Hz[0], COEFFICIENT_1477_Hz);
//Calculează puterea de ieșire Goertzel pt această frecvență.
    if ( goertzel_output_1477_Hz > GOERTZEL_THRESHOLD)
        {goertzel_bit_mask |= 0x0040; }
// Setează bitul 6 în bit mask
    else {goertzel_bit_mask &= 0xFFBF; }
// Resetează bitul 6 în bit mask
}

else if ( 206 == N)
//A patra coloană a tastaturii telefonului
    {goertzel_output_1633_Hz = calculate_goertzel_output(
&delay_1633_Hz[0], COEFFICIENT_1633_Hz);
//Calculează puterea de ieșire Goertzel pt această frecvență.
    if ( goertzel_output_1633_Hz > GOERTZEL_THRESHOLD)
        {goertzel_bit_mask |= 0x0080; }
// Setează bitul 7 în bit mask
    else {goertzel_bit_mask &= 0xFF7F; }
// Resetează bitul 7 în bit mask
}

else if ( 207 == N)
//Afisează butonul apăsat și resetează bit_mask
    {if ( BUTTON_0 == goertzel_bit_mask)
//compară bit_mask cu "butoanele declarate în goertzel.h"
        {puts("Button 0 pressed.\n");}
    else if ( BUTTON_1 == goertzel_bit_mask)
        {puts("Button 1 pressed.\n");}
    else if ( BUTTON_2 == goertzel_bit_mask)
        {puts("Button 2 pressed.\n");}
    else if ( BUTTON_3 == goertzel_bit_mask)
        {puts("Button 3 pressed.\n");}
    else if ( BUTTON_4 == goertzel_bit_mask)
        {puts("Button 4 pressed.\n");}
    else if ( BUTTON_5 == goertzel_bit_mask)
        {puts("Button 5 pressed.\n");}
    else if ( BUTTON_6 == goertzel_bit_mask)
        {puts("Button 6 pressed.\n");}
}

```

```

else if ( BUTTON_7 == goertzel_bit_mask)
    {puts ("Button 7 pressed.\n"); }
else if ( BUTTON_8 == goertzel_bit_mask)
    {puts ("Button 8 pressed.\n"); }
else if ( BUTTON_9 == goertzel_bit_mask)
    {puts("Button 9 pressed.\n"); }
else if ( BUTTON_STAR == goertzel_bit_mask)
    {puts ("Button * pressed.\n"); }
else if ( BUTTON_HASH == goertzel_bit_mask)
    {puts ("Button # pressed.\n"); }
    /* Incepe din nou */
N = 0;
goertzel_bit_mask = 0;
}
/* Adună puterile de ieșire a fiecărui filtru */
goertzel_power = goertzel_output_697_Hz +
goertzel_output_770_Hz +
goertzel_output_852_Hz + goertzel_output_941_Hz +
goertzel_output_1209_Hz + goertzel_output_1336_Hz +
goertzel_output_1477_Hz + goertzel_output_1633_Hz ;
/* Calculează ieșirea */

left_output = (short int)((long)(mono_input * goertzel_power)
>>(15 - 2));
right_output = left_output;
}
}
if ( counter < 5 )
//numai unul din 6 eșantioane este procesat, celelalte ignorate
{ counter++; }
else
{ counter = 0; }

/* Afisează ieșirea pe display-ul cu LED-uri */
bargraph_6dB ( left_output, right_output);
/* Termină procesarea. Închide codecul*/
DSK5416_PCM3002_closeCodec(hCodec);

puts("TMS320C5416 DSK has terminated.\n");
}

main()
{
void main()
{ /* Inițializează librăriile de suport ale placii */
    DSK5416_init();
    /* Toate celelalte funcții sunt apelate de DSP/BIOS */
}
}

```

### Crearea unor efecte sonore speciale – implementare

```

*****
/* alien Voices.c */
/*TMS320C5416 DSK folosește o rată de eșantionare 48000Hz
(48 kHz). */
/* Preia semnalul de intrare de la un microfon sau de la un
CD-player, îl multiplică cu un semnal sinusoidal și generează
suma sau diferența de frecvențe, iar în final trimite ieșirea
spre difuzor. */

*****
#include <stdio.h>      /* Required for functions printf() and
puts() */
#include "alien_voicescfg.h"

#include "dsk5416.h"
#include "dsk5416_pcm3002.h"

#include "bargraph.h"
#include "IIR_high_pass_filters.h"
#include "IIR_low_pass_filters.h"
#include "sinewaves.h"
#include "switches.h"

*****
/* Configurarea regiștrilor codecului PCM3002 */
*****
DSK5416_PCM3002_Config setup =
{
    0x1FF,
    // programare Registrul 0 - atenuarea pe canalul stang CDA
    0x1FF,
    // programare Registrul 1 - atenuarea pe canalul drept CDA
    0x0,
    // programare Registrul 2 - diferite moduri de control
    0x0
    // programare Registrul 3 - pt controlul codecului
};

*****
/* funcția ring_modulation() - produce modulație în inel prin
multiplicarea intrării cu un semnal sinusoidal. Parametrii :
intrarea audio și frecvența de modulație */
*****
static short int ring_modulation ( short int input1, short int
input2)
{ signed long result;
    result = (long) ( input1 * input2) >> 15;
    return ( (short int) result);
}

```

```

}

***** Semnal sinusoidal de 2 Hz si amplitudine 32767 ****
/* Pentru compatibilitate cu functiile read/write ale pcm3002*/
/* aceste variabile trebuie declarate Int16 sau short int si nu
int */
***** Int16 left_input;
Int16 left_output;
Int16 right_input;
Int16 right_output;
Int16 mono_input;

***** Această procedură nu este apelată de main, ci este apelată de
DSP/BIOS
***** void UserTask()
{
    DSK5416_PCM3002_CodecHandle hCodec;
    unsigned long i;
    unsigned int switch_value;
    short int sinewave;
    short int temp;

    /* initializeaza codecul */
    hCodec = DSK5416_PCM3002_openCodec(0, &setup);

    /* Afisează detaliile la Stdout */
    puts("TMS320C5416 DSK: Generating alien voices using ring
modulation. Bargraph in 6dB intervals.\n");

    for ( i = 0 ; i < 12000000 ; i++ )
    {
        // Citește canalul de intrare stanga
        while (!DSK5416_PCM3002_read16(hCodec, &left_input));
        // Redă la canalul de ieșire stanga
        while (!DSK5416_PCM3002_write16(hCodec, left_output));

        /* Procesarea canalului stang */
        if ( 0 == switch_value)
        /* intrarea este redată la ieșire plus un procent */
        { left_output = left_input + left_input/8; }

        else if ( 1 == switch_value)
        {
            /* Generează semnal sinusoidal modulator, cu frecvența de 2 Hz
și amplitudine 32767 */
            sinewave = generate_sinewave_1 ( 2, 32767 );
            left_output = ring_modulation ( left_input, sinewave);
            left_output += (left_input/8);
        }
        else if ( 2 == switch_value)
    }
}

```

```

    {
        /* generează semnal sinusoidal de 20 Hz și amplitudine 32767 */
        sinewave = generate_sinewave_1 ( 20, 32767 );
        left_output = ring_modulation ( left_input, sinewave);
        left_output += (left_input/8);
    }
    else if ( 3 == switch_value)
    {
        /* generează semnal sinusoidal de 100 Hz și amplitudine 32767*/
        sinewave = generate_sinewave_1 ( 100, 32767 );
        left_output = ring_modulation ( left_input, sinewave);
        left_output += (left_input/8);
    }
    else if ( 4 == switch_value)
    {
        /* generează semnal sinusoidal de 200 Hz și amplitudine 32767*/
        sinewave = generate_sinewave_1 ( 200, 32767 );
        left_output = ring_modulation ( left_input, sinewave);
        left_output += (left_input/8);
    }
    else if ( 5 == switch_value)
    {
        /*generează semnal sinusoidal de 500 Hz și amplitudine 32767 */
        sinewave = generate_sinewave_1 ( 500, 32767 );
        left_output = ring_modulation ( left_input, sinewave);
        left_output += (left_input/8);
    }
    else if ( 6 == switch_value)
    {
        /* generează semnal sinusoidal de 10000 Hz și amplitudine
32767*/
        sinewave = generate_sinewave_1 ( 1000, 32767 );
        left_output = ring_modulation ( left_input, sinewave);
        left_output += (left_input/8);
    }
    else if ( 7 == switch_value)
    {
        /*generează semnal sinusoidal de 2000 Hz și amplitudine 32767*/
        sinewave = generate_sinewave_1 ( 2000, 32767 );
        left_output = ring_modulation ( left_input, sinewave);
        left_output += (left_input/8);
    }
    else if ( 8 == switch_value)
    {
        /* generează semnal sinusoidal de 600 Hz și amplitudine 32767
și apoi se filtrează */
        sinewave = generate_sinewave_1 ( 600, 32767 );
        temp = ring_modulation ( left_input, sinewave);
        left_output = fourth_order_IIR_direct_form_I (
&IIR_low_pass_600Hz[0], temp);
    }
    else if ( 9 == switch_value)
    {

```

```

/* generează semnal sinusoidal de 1000 Hz și amplitudine 32767
și apoi se filtreză */
    sinewave = generate_sinewave_1 ( 1000, 32767 );
    temp = ring_modulation ( left_input, sinewave);
    left_output = fourth_order_IIR_direct_form_I (
&IIR_low_pass_1000Hz[0], temp);
}
else if ( 10 == switch_value)
{
/* generează semnal sinusoidal de 2000 Hz și amplitudine 32767
și apoi se filtreză */
    sinewave = generate_sinewave_1 ( 2000, 32767 );
    temp = ring_modulation ( left_input, sinewave);
    left_output = fourth_order_IIR_direct_form_I (
&IIR_low_pass_2000Hz[0], temp);
}
else if ( 11 == switch_value)
{
/* se modulează de 2 ori: prima dată trece jos la 2400 Hz, și a
doua oară trece sus la 2000 Hz. */
    sinewave = generate_sinewave_1 ( 2000, 32767 );
    temp = ring_modulation ( mono_input, sinewave);
    left_output = fourth_order_IIR_direct_form_I (
&IIR_high_pass_2000Hz[0], temp);
    sinewave = generate_sinewave_2 ( 2400, 32767 );
    temp = ring_modulation ( temp, sinewave);
    left_output = fourth_order_IIR_direct_form_II (
&IIR_low_pass_2400Hz[0], temp);
    left_output += (mono_input/8);
}
else if ( 12 == switch_value)
{
/* se modulează de 2 ori: prima dată trece sus la 2000 Hz, și a
doua oară trece jos la 2400 Hz. */
    sinewave = generate_sinewave_1 ( 2000, 32767 );
    temp = ring_modulation ( mono_input, sinewave);
    left_output = fourth_order_IIR_direct_form_I (
&IIR_low_pass_2000Hz[0], temp);
    sinewave = generate_sinewave_2 ( 2400, 32767 );
    temp = ring_modulation ( temp, sinewave);
    left_output = fourth_order_IIR_direct_form_II (
&IIR_high_pass_2400Hz[0], temp);
    left_output += (mono_input/8);
}
else if ( 13 == switch_value)
{
/* se modulează de 2 ori: prima dată trece sus la 2400 Hz, și a
doua oară trece jos la 2000 Hz. */
    sinewave = generate_sinewave_1 ( 2400, 32767 );
    temp = ring_modulation ( mono_input, sinewave);
    left_output = fourth_order_IIR_direct_
form_I( &IIR_high_pass_2400Hz[0], temp);
    sinewave = generate_sinewave_2 ( 2000, 32767 );

```

```

    temp = ring_modulation ( temp, sinewave);
    left_output = fourth_order_IIR_direct_form_II (
&IIR_low_pass_2000Hz[0], temp);
    left_output += (mono_input/8);
}
else if ( 14 == switch_value)
{
/* se modulează de 2 ori: prima dată trece jos la 2400 Hz, și a
doua oară trece sus la 2000 Hz. */
    sinewave = generate_sinewave_1 ( 2400, 32767 );
    temp = ring_modulation ( mono_input, sinewave);
    left_output = fourth_order_IIR_direct_form_I (
&IIR_low_pass_2400Hz[0], temp);
    sinewave = generate_sinewave_2 ( 2000, 32767 );
    temp = ring_modulation ( temp, sinewave);
    left_output = fourth_order_IIR_direct_form_II (
&IIR_high_pass_2000Hz[0], temp);
    left_output += (mono_input/8);
}
else if ( 15 == switch_value)
{
/* generează semnal sinusoidal de 500 Hz și amplitudine 32767
și adună reverberație */
    sinewave = generate_sinewave_1 ( 500, 32767 );
    left_output = ring_modulation ( left_input, sinewave);
    left_output += (left_input/8);
}

/* citește canalul de intrare drept */
while (!DSK5416_PCM3002_read16(hCodec, &right_input));

/* redă canalul drept */
while (!DSK5416_PCM3002_write16(hCodec, right_output));

/* citește starea plăcii DSK și afișează la Stdout */
switch_value = switch_status_display();

/* unde este necesar, generează intrare mono din intrările
stânga și dreapta*/
mono_input = stereo_to_mono ( left_input, right_input);

/* procesarea de semnal */

if ( 0 == switch_value)
{
    right_output = right_input + (right_input/8); }

else if ( 1 == switch_value)
{
/* undă sinusoidală de 2 Hz și amplitudine 32767 */
    right_output = ring_modulation ( right_input,
sinewave);
    right_output += (right_input/8);
}

```

```

        else if ( 2 == switch_value)
        {
/* undă sinusoidală de 20 Hz și amplitudine 32767 */
        right_output = ring_modulation ( right_input,
sinewave);
        right_output += (right_input/8);
    }
    else if ( 3 == switch_value)
    {
/* undă sinusoidală de 100 Hz și amplitudine 32767 */
        right_output = ring_modulation ( right_input,
sinewave);
        right_output += (right_input/8);
    }
    else if ( 4 == switch_value)
    {
/* undă sinusoidală de 200 Hz și amplitudine 32767 */
        right_output = ring_modulation ( right_input,
sinewave);
        right_output += (right_input/8);
    }
    else if ( 5 == switch_value)
    {
/* undă sinusoidală de 500 Hz și amplitudine 32767 */
        right_output = ring_modulation ( right_input,
sinewave);
        right_output += (right_input/8);
    }
    else if ( 6 == switch_value)
    {
/* undă sinusoidală de 1000 Hz și amplitudine 32767 */
        right_output = ring_modulation ( right_input,
sinewave);
        right_output += (right_input/8);
    }
    else if ( 7 == switch_value)
    {
/* undă sinusoidală de 2000 Hz și amplitudine 32767 */
        right_output = ring_modulation ( right_input,
sinewave);
        right_output += (right_input/8);
    }
    else if ( 8 == switch_value)
    {
/* undă sinusoidală de 600 Hz și amplitudine 32767 */
        temp = ring_modulation ( right_input, sinewave);
        right_output = fourth_order_IIR_direct_form_II (
&IIR_high_pass_600Hz[0], temp);
    }
    else if ( 9 == switch_value)
    {
/* undă sinusoidală de 1000 Hz și amplitudine 32767 */
        temp = ring_modulation ( right_input, sinewave);

```

```

        right_output = fourth_order_IIR_direct_form_II (
&IIR_high_pass_1000Hz[0], temp);
    }
    else if ( 10 == switch_value)
    {
/* undă sinusoidală de 2000 Hz și amplitudine 32767 */
        temp = ring_modulation ( right_input, sinewave);
        right_output = fourth_order_IIR_direct_form_II (
&IIR_high_pass_2000Hz[0], temp);
    }
    else if ( 11 == switch_value)
    {
/* modulație trece sus la 2000 Hz și apoi modulație trece jos
la 2400 Hz*/
        right_output = left_output;
        right_output += (mono_input/8);
    }
    else if ( 12 == switch_value)
    {
/* modulație trece jos la 2000 Hz și apoi modulație trece sus
la 2400 Hz */
        right_output = left_output;
        right_output += (mono_input/8);
    }
    else if ( 13 == switch_value)
    {
/* modulație trece sus la 2400 Hz și apoi modulație trece jos
la 2000 Hz */
        right_output = left_output;
        right_output += (mono_input/8);
    }
    else if ( 14 == switch_value)
    {
/* modulație trece jos la 2400 Hz și apoi modulație trece
sus la 2000 Hz */
        right_output = left_output;
        right_output += (mono_input/8);
    }
    else if ( 15 == switch_value)
    {
/* undă sinusoidală de 200 Hz și amplitudine 32767, și apoi se
adună reverberații */
        temp = ring_modulation ( right_input, sinewave);
        right_output = reverberation ( temp );
        right_output += (mono_input/8);
    }
    /* se afișează ieșirea */
    bargraph_6dB ( left_output, right_output);
}
/* Termină procesarea. Închide codecul */
DSK5416_PCM3002_closeCodec(hCodec);

```

```

    puts("TMS320C5416 DSK has terminated\n");
}
//*********************************************************************
/* main()*/
//********************************************************************

void main()
{
/* Initialize the board support library */
/* There is no need to initialize the DIP switches
and the LEDs */

    DSK5416_init();

    /* All other functions are scheduled by DSP/BIOS */

}
//*********************************************************************

```

## Bibliografie

1. E. Lupu, R. Arsinte, T. Miclea, *Procesoare digitale de semnal generația TMS320C2X. Prezentare și aplicații* Ed. Promedia Plus ISBN 973-97377-0-6
2. E. Lupu, A. Suciuc, *Procesoare digitale de semnal, Lucrări practice*, U.T.PRES, Cluj-Napoca, 2002
3. Texas Instruments, *TMS320C54x Assembly Language Tools User's Guide*, Literature Number: SPRU102F, 2002
4. Texas Instruments, *TMS320VC5416 DSK Technical Reference*, 506005-0001 Rev. A, 2002
5. Texas Instruments, *A DSP/BIOS PCM3002 Codec Device Driver for the TMS320C5416 DSK*, Literature Number: SPRA855, 2002
6. Geneviève Baudoin, Olivier Venard, Férial Virolleau, *C5000 Teaching Rom*, Texas Instruments, 2003
7. Texas Instruments, *TMS320 DSP platform, Code Composer Studio IDE Development Tools*, Platinum Edition, Version 3.1
8. Texas Instruments, *Code Composer Studio Getting Started Guide*, Literature Number: SPRU509C, 2001
9. Texas Instruments, *Code Composer Studio User's Guide*, Literature Number: SPRU328B, 2000
10. [10. <http://c5000.spectrumdigital.com/dsk5416/>](http://c5000.spectrumdigital.com/dsk5416/)

Simina EMERICH

Eugen LUPU

**PROCESOARE DIGITALE  
DE SEMNAL**  
*Lucrări practice*

[www.galaxiagutenberg.ro](http://www.galaxiagutenberg.ro)

Editura Galaxia Gutenberg  
435600 Târgu-Lăpuș, str. Florilor nr. 11  
tel./fax: 0262-385280; mobil: 0723-377599  
E-mail: contact@galaxiagutenberg.ro

ISBN 978-973-141-542-0

Galaxia Gutenberg

2014

## Cuprins

### PREFATA

1. REPREZENTAREA NUMEREOR ȘI ARITMETICA LA PROCESOARELE DE SEMNAL ÎN VIRGULĂ FIXĂ .....	pag 1
2. SIMULAREA APLICAȚIILOR PENTRU FAMILIA TMS320C2X .....	pag 9
3. APLICATII PE PLACA "SIDERAL" TMS320C25 .....	pag 23
4. DEZVOLTAREA APLICAȚIILOR SUB CODE COMPOSER STUDIO® (CCS) .....	pag 31
5. SISTEMUL DE DEZVOLTARE TMS320VC5416 DSK .....	pag 47
6. DEZVOLTAREA APLICAȚIILOR PENTRU SISTEMUL DSK TMS320C5416 .....	pag 61
ANEXA 1: Structura fișierului LINK25.CMD .....	pag 71
ANEXA 2: PCM3002, descrierea registrilor .....	pag 76
ANEXA 3: Conexiuni de expansiune ai sistemului DSK TMS320C5416 .....	pag 77
ANEXA 4: Algoritmul Goertzel, Crearea unor efecte sonore speciale – implementare .....	pag 79

### BIBLIOGRAFIE

## 1. REPREZENTAREA NUMEREOR ȘI ARITMETICA LA PROCESOARELE DE SEMNAL ÎN VIRGULĂ FIXĂ

### 1.1 Generalități

Fără să aibă informații legate de convenția folosită la reprezentarea numerelor binare în calculator, nu se poate atribui o valoare exactă unui număr binar, acesta reprezentând o înșiruire de biți. De exemplu, ce valoare zecimală are  $10111011B = X$ ? Este pozitiv sau negativ? Răspunsul este: depinde de reprezentarea folosită.

La alegerea unui Procesor Digital de Semnal (DSP) pentru o anumită aplicație, una dintr-o caracteristică importantă la luarea deciziei, o constituie modul de reprezentare (binar) al datelor de către procesor. Din acest punct de vedere DSP-urile de pe piață se pot clasifica după cum arată Fig. 1:

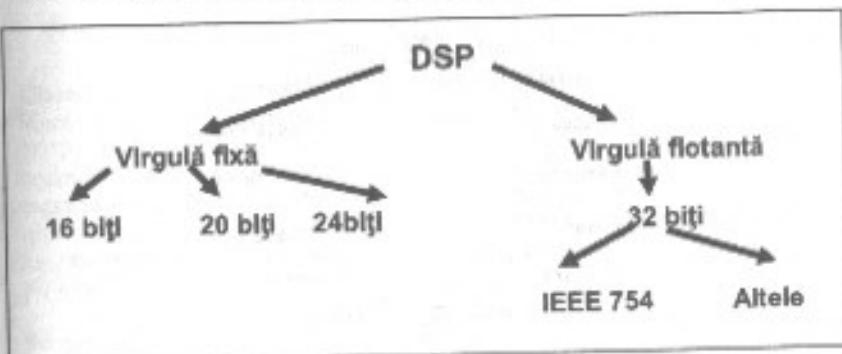


Fig. 1. Reprezentarea numerelor la DSP-urile uzuale

Aritmetică în virgulă fixă a fost folosită de către primele DSP-uri și la ora actuală mai este încă predominantă la multe dintre DSP-urile uzuale.

DSP-urile în virgulă fixă folosesc reprezentarea numerelor fie ca:

- întregi (aritmetică întreagă) – fiind folosită de DSP la operații de control, calcul de adrese sau alte operații care nu implică semnale;
- fracționar (aritmetică fracțională) cu valori între -1 și +1 utilizată în calculele legale de semnale.

Algoritmii și hard-ul folosit la implementarea aritmeticii fracționale sunt virtual identici cu cei folosiți la aritmetică întreagă. Diferența principală între cele două aritmetici apare la modul de folosire a rezultatelor operațiilor de înmulțire. Majoritatea DSP-urilor în virgulă fixă acceptă cele două aritmetici.