

DEZVOLTAREA APLICAȚIILOR SUB CODE COMPOSER STUDIO®

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. Selecția sistemului pentru care se dezvoltă aplicațiile se face prin intermediul componentei **Setup CCStudio**.

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

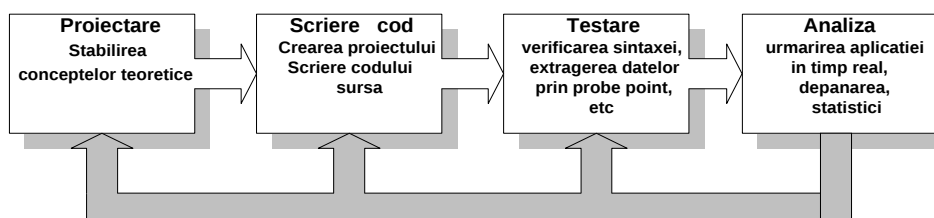


Fig. 1 Fazele dezvoltării unei aplicații sub CCS

Code Composer Studio include următoarele componente:

- ❑ **TMS 320C54X Code Generation Tools** – Conține principalele elemente software necesare în dezvoltarea aplicațiilor sub CCS
- ❑ **CCS Integrated Development Enviroment (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 fig. 2.

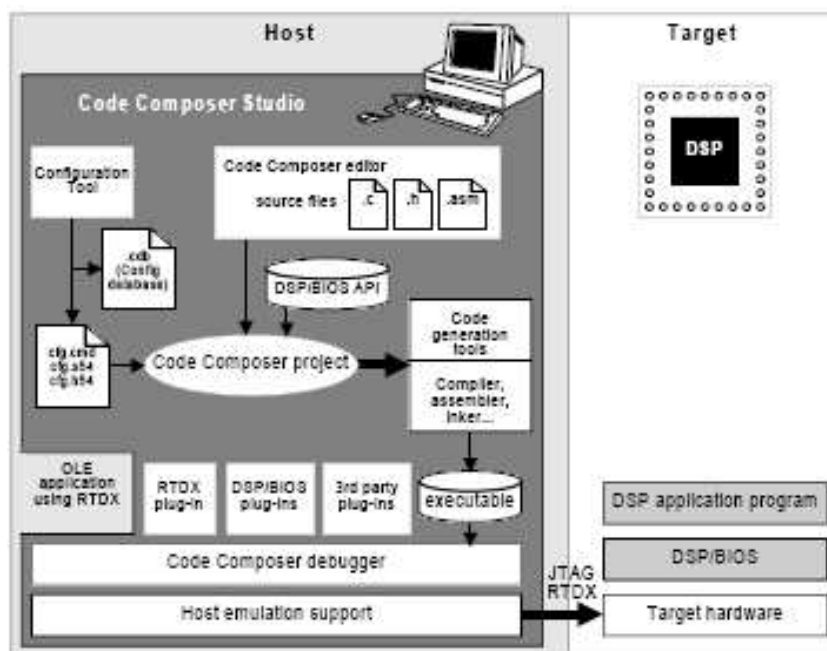


Fig.2 Componentele utilizate la dezvoltarea unei aplicații sub CCS

2. Descrierea utilităților software

Majoritatea utilităților 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 singur modul obiect executabil
- ❑ **Arhivorul** –colectează grupuri de fișiere într-un singur fișier bibliotecă (arhivă). Arhivorul poate modifica biblioteca în sensul: stingerii, reînscrierii, extragerii sau adăugării unui nou membru
- ❑ **Translatorul de cod** –Translatează codul ASM din forma mnemonică în forma algebrică
- ❑ **Managerul de biblioteci** –gestionează bibliotecile run-time
- ❑ **Bibliotecile run-time** -conțin:
 - o funcții pentru lucru în timp real
 - o funcții pentru calcul aritmetic în virgulă flotantă
 - o funcții I/O suportate de standardul C - ANSI
- ❑ **Utilitarul de conversie hex** – converteste un fișier obiect de tip COFF într-un fișier obiect de tip Ti-Tagged, ASCII-hex, Intel, Motorola-S sau Tektronix;
- ❑ Modulul **cross-reference lister** folosește fișiere obiect pentru a produce un fișier de tip .LST ce va arata simbolurile, definițiile și referințele în fișierele sursa linkeditate.
- ❑ Modulul **absolute lister** – primește la intrare fișiere obiect linkeditate și creează fișiere de tip .ABS la ieșire. Fișierul .ABS se assemblează pentru a produce un fișier .LST ce va conține în locul adreselor relative, adrese absolute.

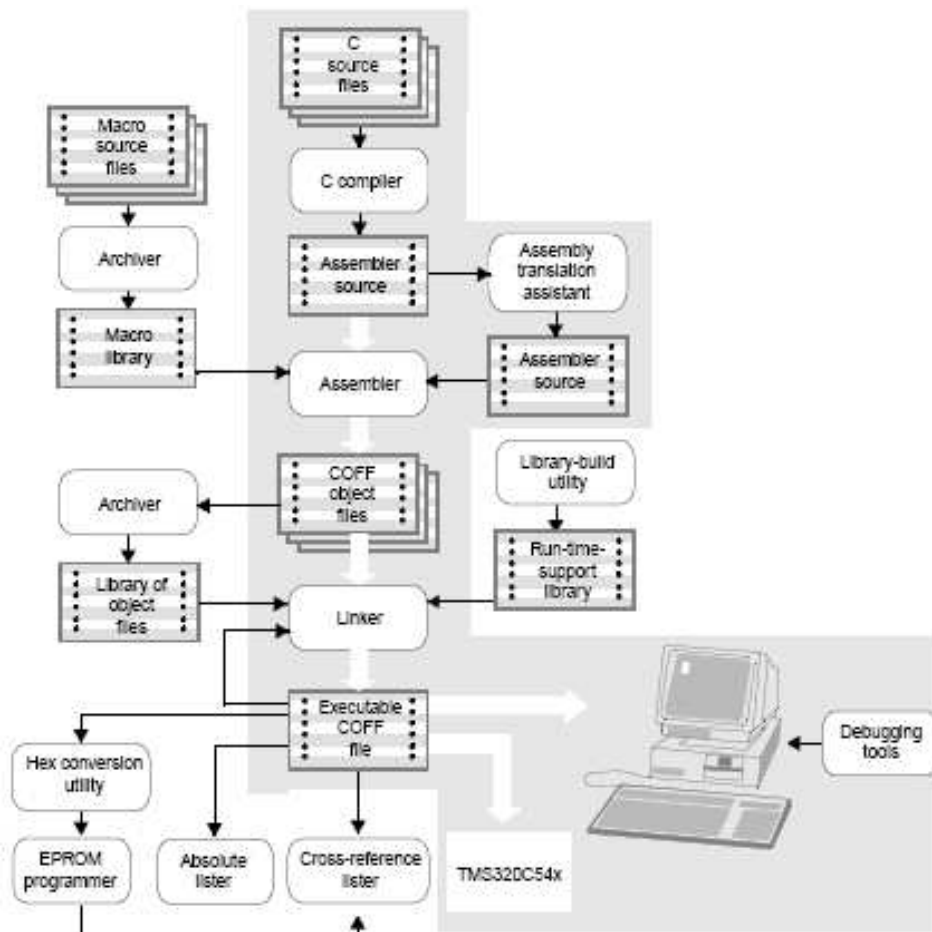


Fig 3 Fluxul de dezvoltare a aplicațiilor sub Cod Composer Studio

3. Elemente necesare in dezvoltarea unei aplicații sub CCS

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.

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 Proiect Wiew în secțiunea GEL files (Click-dreapta ->Load GEL)sau cu **File->Load GEL**.

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)  
{  
    //corpul 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

Controlul GEL de tip dialog

Acest tip de control permite modificarea unei variabilelor prin intermediul căsuțelor de dialog:

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

paramName[1-6] numele parametrilor utilizați în corpul funcției

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

În exemplul de mai jos se definesc două controale GEL unul de tip slider și altul de tip dialog.

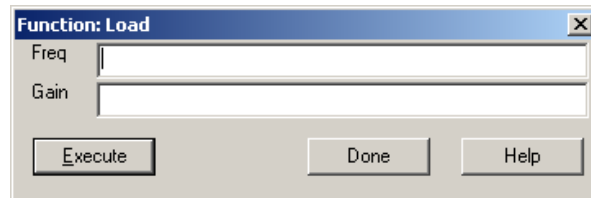
Exemplu de fișier GEL

menuitem "Application Control"

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

```
dialog Load(loadParm1 "Freq",loadParm2 "gain")  
{  
    Freq = loadParm1;  
    gain = loadParm2;  
}
```

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.



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

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



3.3 Vizualizarea grafică a zonelor de memorie – “Fereastra Graph”

Prin utilizarea ferestrelor 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

StartAddress - adresa de start

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

AcquisitionBufferSize–mărimea bufferului citit de către Graph

IndexIncrement –rezoluția afișării

DisplayData Size – numărul de eșantioane afișate

DSP DataType –formatul de afișare a datelor

Q-value – tipul datelor afișate în format fracționar

Left-shiftedData Display - tipărire cu deplasare spre dreapta

SamplingRate(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

AxesDisplay-afișarea axelor

Time displayUnit - unitatea de măsură pe axa timpului: s, ms, μ s, eșantioane

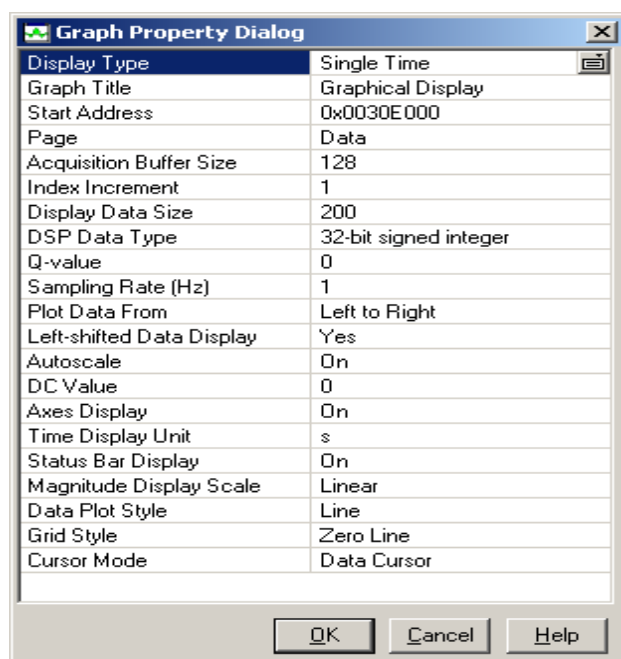
StatusBar Display-afișarea barei de stare pe care se tipăresc cordonatele cursorului

Magnitude Display Scale - tipul scalei pentru amplitudine liniar sau logaritmic

Data Plot Style-Stilul de afișare Line sau Bar

Grid Style - Afișează rastrul (Full Grid), numai linia de zero (ZeroLine) sau nimic (No Grid)

Cursor Mode - Fără cursor (No Cursor), Cursor pentru date (Data Cursor), Cursor pentru Zoom (Zoom Cursor)



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

SignalType-tipul semnalului real sau complex

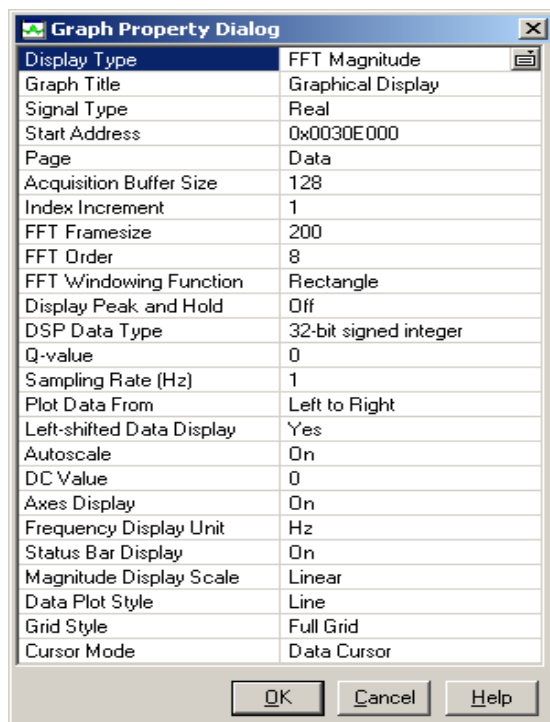
FFT-Frame size-mărimea ferestrei FFT

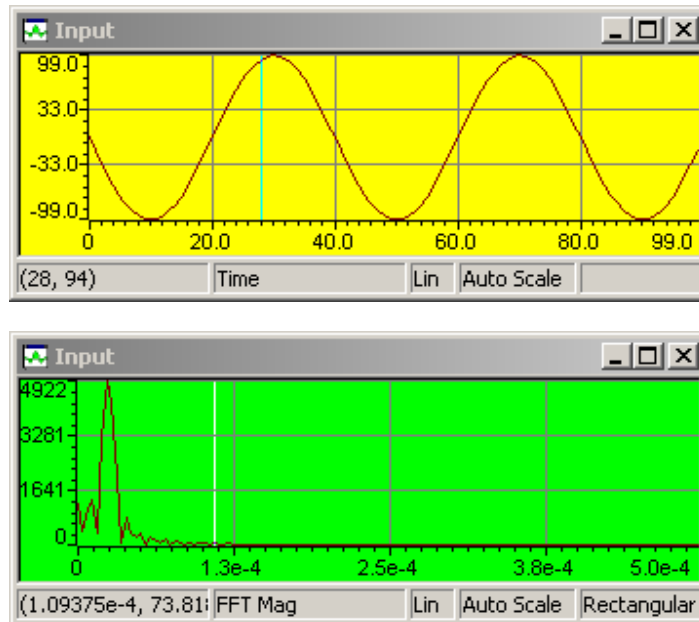
FFT Order-ordinul FFT

FFT Windowing Function –tipul ferestrei de ponderare

DisplayPeak and Hold – detectează și păstrează maximele

Frequency Display Unit –unitatea de afișare pentru frecvență: Hz, KHz, MHz





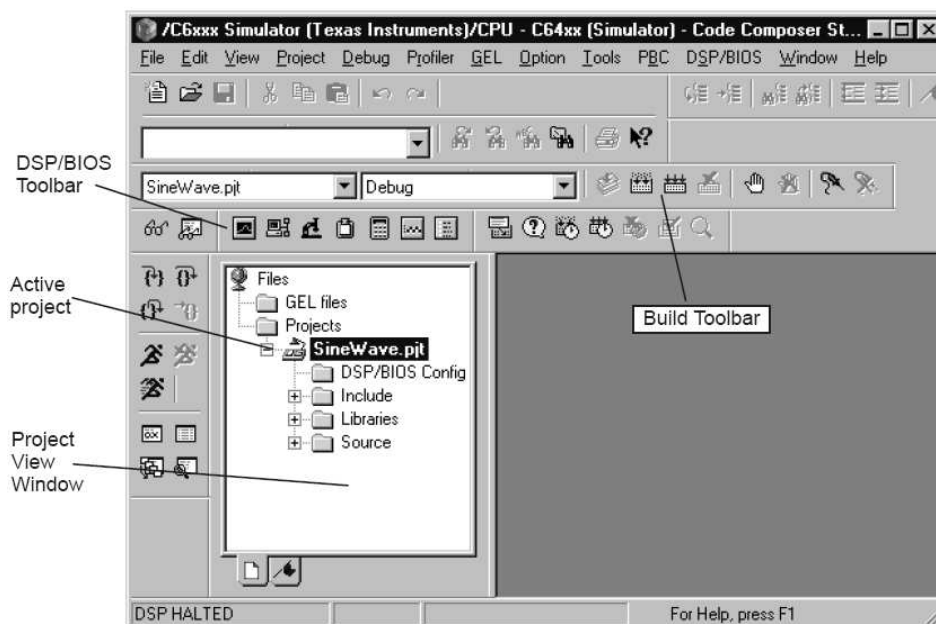
3.4. Exemplu de lucru cu CCS

Aplicația prezentată în continuare realizează controlul amplificării unui semnal de intrare.

Pentru a rula aplicația este necesar ca **CCS** să fie configurat astfel încât să lucreze cu simulatorul **C5416 Simulator**. Această configurare se face prin intermediul componentei **Setup CCS**.

În continuare sunt descriși pașii ce trebuie urmați pentru a dezvolta o aplicație în **CCS** și pentru a o rula pe simulator.

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



In continuare este prezentata sursa aplicatiei :

```

/*****
*/sine.c ; Acest program foloseste Probe Point-uri pentru a obtine un semnal sinusoidal de
intrare, iar apoi acestui semnal i se aplica un factor de castig.
*****/

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

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

// declarare si initializare Buffer IO
BufferContents currentBuffer;

// definire functii
static void processing(); // prelucreaza intrarea si genereaza semnalul de iesire
static void dataIO(); // functia ce se foloseste pentru ProbePoint

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

    while(TRUE) // bucla continua
    {
        /* citeste datele de intrare folosind un probe-point conectat la un fisier gazda
        Scrie datele de iesire intr-un graf conectat tot printr-un probe-point */
        dataIO();

        /* pentru a obtine datele de la iesire se aplica castigul asupra datelor de intrare */
        processing();
    }
}

/*Functia urmatoare aplica o transformare de semnal asupra semnalului de intrare pentru a genera
semnalul de iesire si are ca parametrii: structura BufferContents ce contine sirurile de
intrare/iesire de dimensiune BUFFSIZE; functia nu returneaza nici o valoare */

static void processing()
{
    int size = BUFFSIZE;

    while(size--){ // aplica castigul asupra intrarii
        currentBuffer.output[size] = currentBuffer.input[size] * gain;
    }
}

/* Functia urmatoare citeste semnalul de intrare si scrie semnalul de iesire procesat folosind
ProbePoints; functia nu are parametrii si nu returneaza nici o valoare. */

static void dataIO()
{
    return;
}
```

- 2) Compilați fișierele sursă cu **Project->Rebuild All** sau apăsați pe butonul (Rebuild All) din toolbar.
 - 3) Incărcați fișierul obiect în memoria DSP din meniul **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 atașa fișierul ce conține semnalul ce urmează să fie procesat este necesar să înserăm 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.

```

puts(" example started\n");

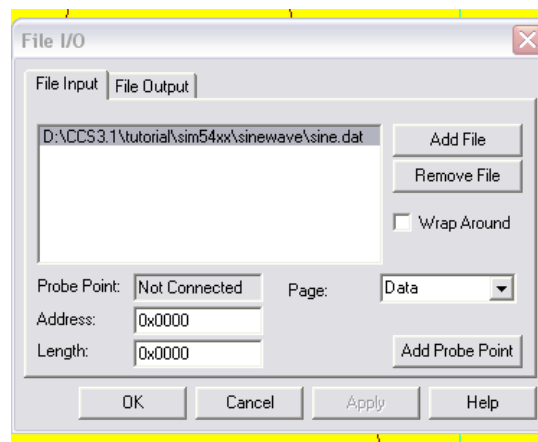
/* bucla infinita */
while(TRUE)
{
    dataIO();
    procesare(input, output); /*amplifica semnalul*/
}

/* ----- procesare -----
 * functia in care se proceseaza semnalul
 */
static int procesare(int *input, int *output)
{
    int size = BUFSIZE;

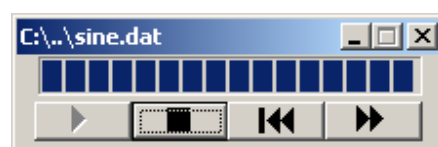
    while(size--){
        *output++ = *input++ * gain;
    }
}

```

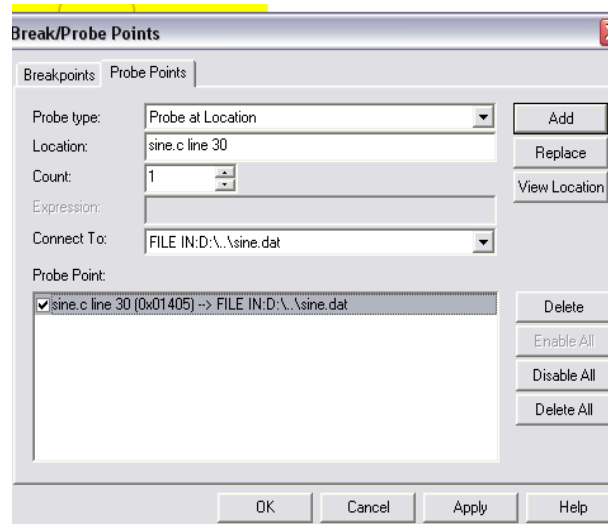
- 7) Alegeți **File->File I/O**. Dialogul File I/O apare pentru a se putea selecta fișierele de intrare și de ieșire.



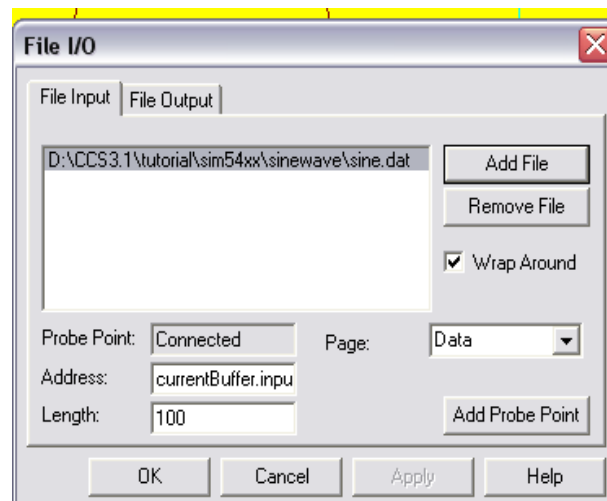
- 8) În meniul **File Input** apăsați pe **Add File**.
- 9) Alegeți fișierul **sine.dat**. Observați că puteți selecta formatul datelor în fereastra **Files of type**. Fișierul **sine.dat** conține valori hexa pentru o formă de undă sinusoidală. Selectați **Open** pentru a adăuga acest fișier în lista 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) In fereastra **File I/O** urmatorul pas este sa adaugam un **Probe Point** utilizand butonul cu acelasi nume. In fereastra de dialog **BreakProbe Points** validam probe point dand click pe el si ne va arata locatia si selectam **Connect to** alegand fisierul sine.dat. La sfarsit apasati Apply.



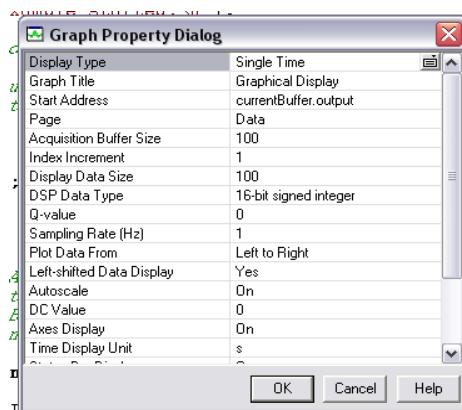
11) In dialogul **File I/O** setati adresa *currentBuffer.input* și lungimea la 100. De asemenea, bifați **Wrap Around** (citirea se face circular). Câmpul de adresă (**Address**) specifică unde trebuie plasate datele din fisier. *currentBuffer.input* e declarat în *volume.c* ca integer array de valoarea 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.



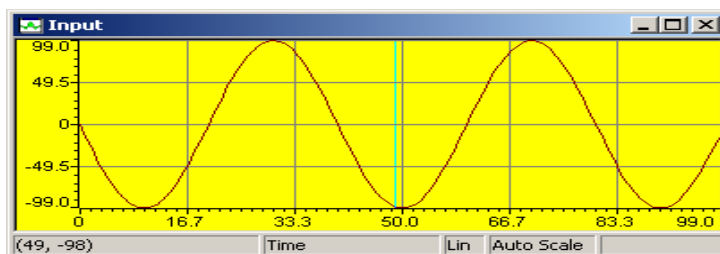
12) **Debug->Run**

13) Selectati **View->Graph->Time/Frequency**.

14) In **Graph Property Dialog** modificati caracteristicile: **Graph Title**, **Start Address**, **Acquisition Buffer Size**, **Display Data Size**, **DSP Data Type**, **Autoscale**, si **Maximum Y-value** cu valorile din figura.



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

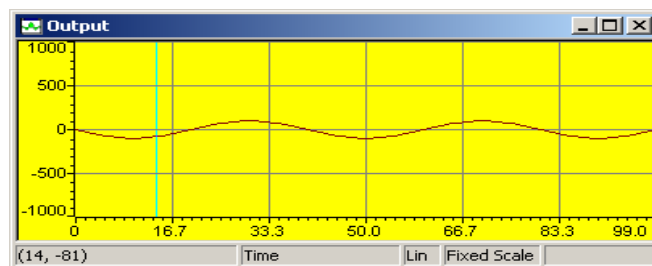


16) Click- dreapta pe fereastra Input și alegeți **Clear Display** din meniul care apare.

17) Alegeți din nou **View->Graph->Time/Frequency**.

18) De data aceasta modificați **Graph Title** în Output Buffer și **Start Address** în *currentBuffer.output*, **Autoscale** OFF .**MaximumY-Value** la 1000.

19) Apăsați OK pentru a afișa fereastra grafică Output – care conține semnalul generat.



20) Dați click-dreapta pe fereastra grafică și alegeți **Clear Display** din meniul afișat.

21) Alegeți **File->Load Gel ->volume.gel**. Acest fisier conține un control GEL de tip slider si unul de tip dialog. Deschideți fisierul pentru a-i vedea conținutul.

22) Selectati **Gel-> Application Control -> Gain**.

23) 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

24) Click (**Halt**) sau apăsați **Shift F5** pentru a opri programul.

5. Teme

- a) Adăugați un control GEL de tip slider pentru a putea modifica castigul.
- 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.

