Developing applications under CODE COMPOSER STUDIO®

1. General Overview

Code Composer Studio (**CCS®**) is a very efficient instrument for the fast development of applications that are written for the DSP families TMS320C54x, C55x, C6x. **CCS** allows: construction, configuration, testing, running and analyzing applications in real time. It facilitates running applications on different development systems or on a simulator. Selection of the syetem for which the applications are developed is done through **Setup CCStudio** component**.**

Development phases of a **CCS** application are shown in fig 1;

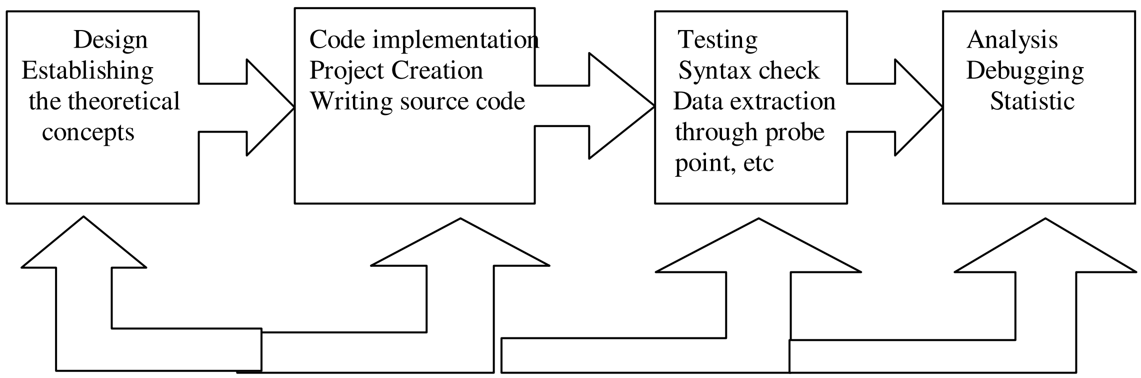


Fig.1 Development phases of a CCS application

Code Composer Studio includes the following components:

- **TMS 320C54X Code Generation Tools-** contains the main software elements needed in application development under **CCS.**
- **CCS Integrated Development Environment (IDE)** – environment that integrates and manages all the CCS components that allow the design, edit and debug of applications.
- **DSP/BIOS –** are libraries of functions that allow communication with the application during running on hardware systems. DSP/BIOS component is structured in two sections:

- **DSP/ BIOS Plug-Ins** offer the possibility of application analysis for performance estimation with minimum impact over the real time performance.

- **DSP/BIOS API** (Application Interface) provides the software components called through source application.

API components offer the possibility of interfacing the applications that run on DSP system with remote application that run on PC.

All this components work together as shown in figure2.

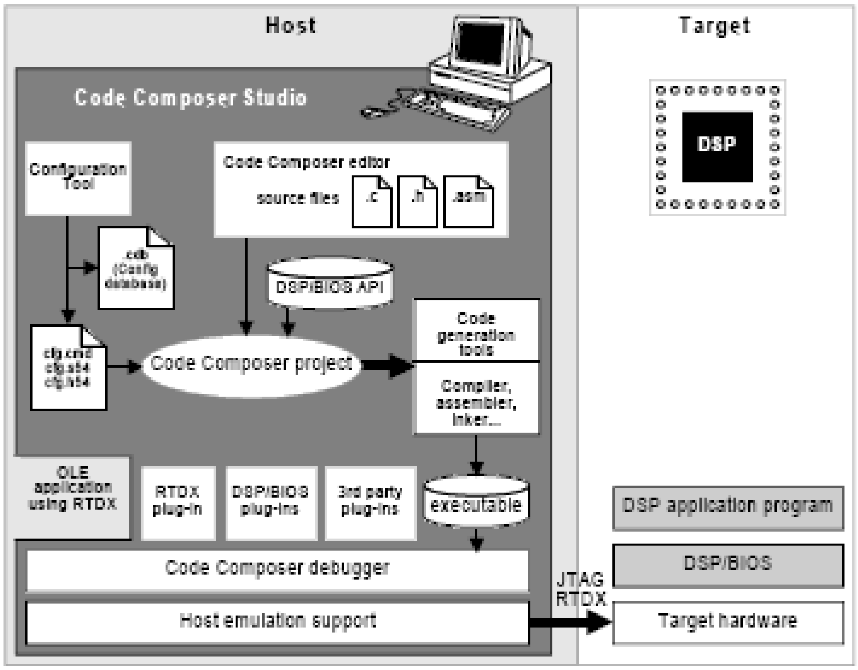


Fig2. Components used when developing an application under CCS.

2. Describing software utilities

The majority of software utilities that are used in the development process of an application under CCS are managed by the IDE environment. Other components are called during application development.

The development flow of an application is shown in figure3.

- **C Complier** – accepts C in ANSI standard and generates source files .ASM
- **Assembler** – turns source files .ASM into object files .COFF
- **Link Editor** – combines object files into a single executable object module.
- **Archiver** – collects groups of files in a single library file (archive).

The archiver can modify the library in the way of: erasing, re-registrating, extracting or adding a new member.

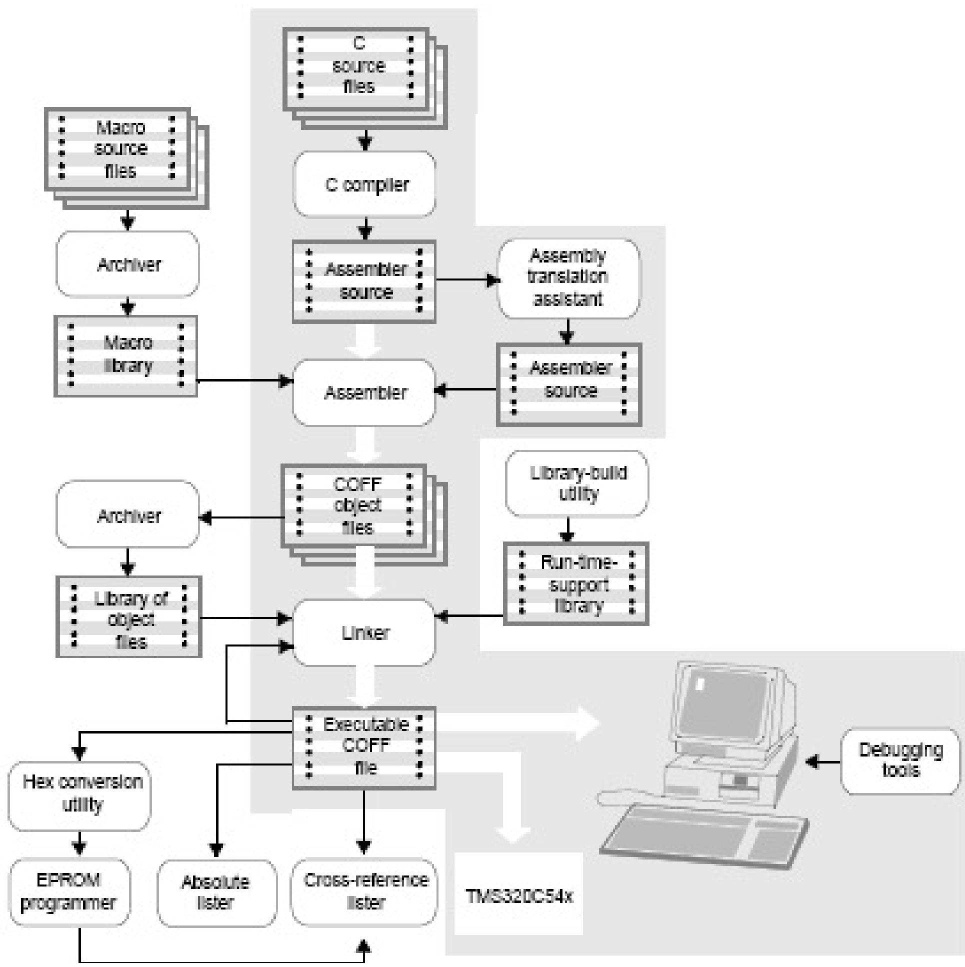
- **Assembly translation assistant**– turns the .ASM form mnemonic into algebraic form.
- **Library Manager** – manages the run-time libraries.
- **Run-Time libraries** – contain:
  - Functions for the real-time operation

- Functions for arithmetic calculus in floating point.
- I/O functions supported by C- ANSI standard.

o **Hex conversion utility** – converts a COFF object file into Ti-Tagged, ASCII-hex, Intel, Motorola –S or Tektronix object file.

o **Cross reference lister module** – uses object files to produce a LST file that will show the symbols, definitions and references in link edited source files.

o **Absolute lister module –** gets as input linkedited object files and create .ABS files as output. The .ABS files is assembled in order to produce an .LST file that will contain absolute addresses instead of relative addresses.



The development flow of an application is shown in figure3

3. Elements needed in developing an application under CCS

Probe-Point usage

The Probe-Points are useful instruments for algorithm development. It can be used for:

- transferring input data from a local file into a DSP buffer to be used by the algorithm.
- transferring output data from a DSP buffer into local file for the analysis.

- updating window (e.g. data graphic). Probe-Points are similar with breakpoints because both interrupt the program executed on DSP for running its own actions. Differences between them are: Probe-Point interrupt DSP program for a moment, perform a single action and continue program execution. The breakpoints interrupt the DSP until the execution is manually run and update all the windows. The Probe-Points permit automatically input and output in/from files. Breakpoints don't allow this. A breakpoint is also used for updating all the opened windows until it traces into that Probe Point. These windows also include the input and output data graphs.

3.2. Gel elements usage

To modify a variable Code Composer Studio offers the possibility to use the GEL (General Extension Language). «GEL » is a language extension that allows creation of small windows used for: modifying variable values declared in the program, writing of functions that allow IDE environment configuration and access to processor. GEL elements are described in files with the .gel extension. These are added in the Project View Window in GEL file section. (Right-click ->**Load GEL**) or with **File->Load GEL.**

Gel Cursor control (slider) allows control of variable through a cursor.

```
slider param_definition( minVal, maxVal, increment, pageIncrement, paramName)
{
  //control body
}
```

param_definion – the cursor's name

minVal – an integer value that specifies the minimum variable values controlled by the cursor.

maxVal – an integer value that specifies the maximum variable values controlled by the slider.

incremet – an integer values that specifies the increment value controlled by the slider.

pageIncrement – an integer value that specifies the increment value controlled by the slider when the keys Page Up and Page Down are pressed.

paramName - the parameter's name used inside the function.

Dialogue Gel Control

This type of control allows the modifications of variable through dialogue boxes.

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

paramName[1-6] parameter's name used in body function

"param1 definition" the name of parameter typed in the control window. We can define maximum 6 parameters.

In the example given below two GEL controls are defined: a slider and a dialogue.

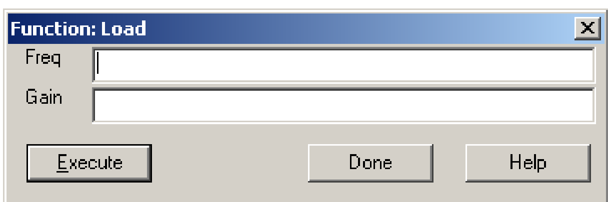
GEL file example.

menuitem "Application Control"

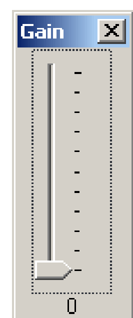
Application control is the name of the submenu's name from GEL menu which is activated after the GEL file loading.

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

GEL Control of type "dialogue" is defined with the name **Load** with two variables **loadParam1** and **LoadParam2** which are attached to **Freq** and **Gain.** After pressing the Execute button variables are updated.




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



Slider GEL control is defined with the name **Gain** that can modify the gain variable between 0 and 10 with **1** increment and **1** post increment.

3.2 Graphical visualization of memory area – "Graph Window"

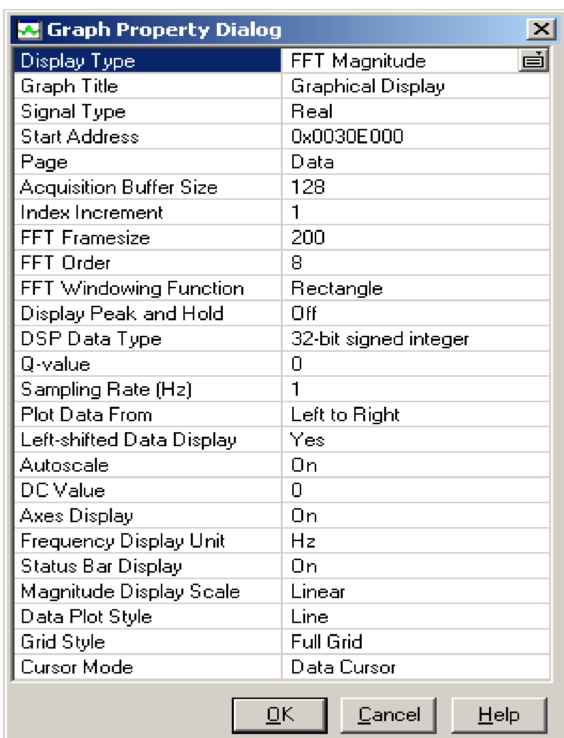
By using the Graph Windows we can see areas of memory graphically in amplitude-time or double-time. FFT- amplitude, FFT complex, FFT phase, FFT phase and amplitude.

Amplitude-time window Graph configuration settings are shown below.

Display Type-
Graph title
StartAddress
Page –memory page that displays Program Data or I/O

AcquisitionBufferSize– buffer size read by Graph
IndexIncrement –display resolution
DisplayData Size – number of displayed samples
DSP DataType – data display format
Q-value – data displayed in fractional format
Left-shiftedData Display
SamplingRate(Hz)-F_{es} in Hz for displaying the data on the graph.
Plot data From – plot data from left to right or reversed
Autoscale – amplitude autoscale.
DC Value-offset
AxesDisplay
Time displayUnit: s, ms, µs, samples
StatusBar Display- status bar display on which the cursor coordinates are typed
Magnitude Display Scale – scale type for linear amplitude or logarithmic
Data Plot Style- data plot style line or bar.
Grid Style – displays Full Grid, just ZeroLine or No Grid
Cursor Mode - No Cursor, Data Cursor, Zoom Cursor

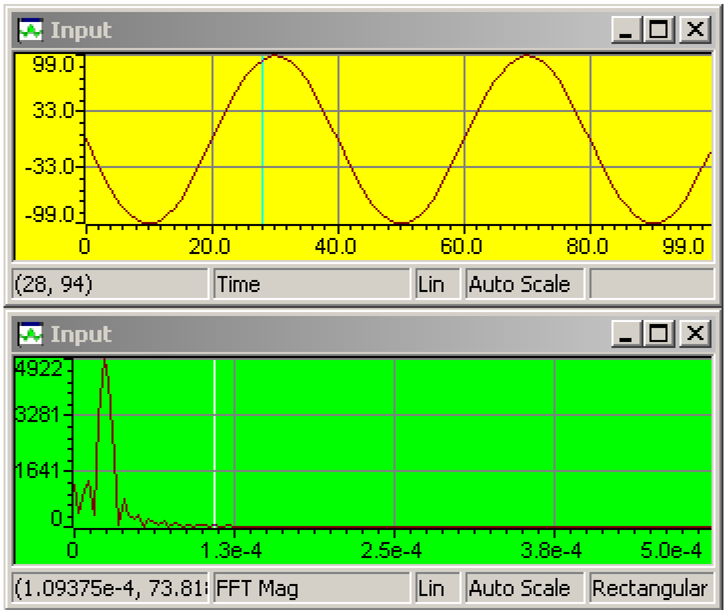
When we have a frequency domain graph we have the following new parameters
SignalType- real or complex
FFT-Frame size
FFT Order
FFT Windowing Function
DisplayPeak and Hold
Frequency Display Unit : Hz, KHz, MHz



Graph Property Dialog

Display Type	FFT Magnitude
Graph Title	Graphical Display
Signal Type	Real
Start Address	0x0030E000
Page	Data
Acquisition Buffer Size	128
Index Increment	1
FFT Framesize	200
FFT Order	8
FFT Windowing Function	Rectangle
Display Peak and Hold	Off
DSP Data Type	32-bit signed integer
Q-value	0
Sampling Rate (Hz)	1
Plot Data From	Left to Right
Left-shifted Data Display	Yes
Autoscale	On
DC Value	0
Axes Display	On
Frequency Display Unit	Hz
Status Bar Display	On
Magnitude Display Scale	Linear
Data Plot Style	Line
Grid Style	Full Grid
Cursor Mode	Data Cursor

OK Cancel Help

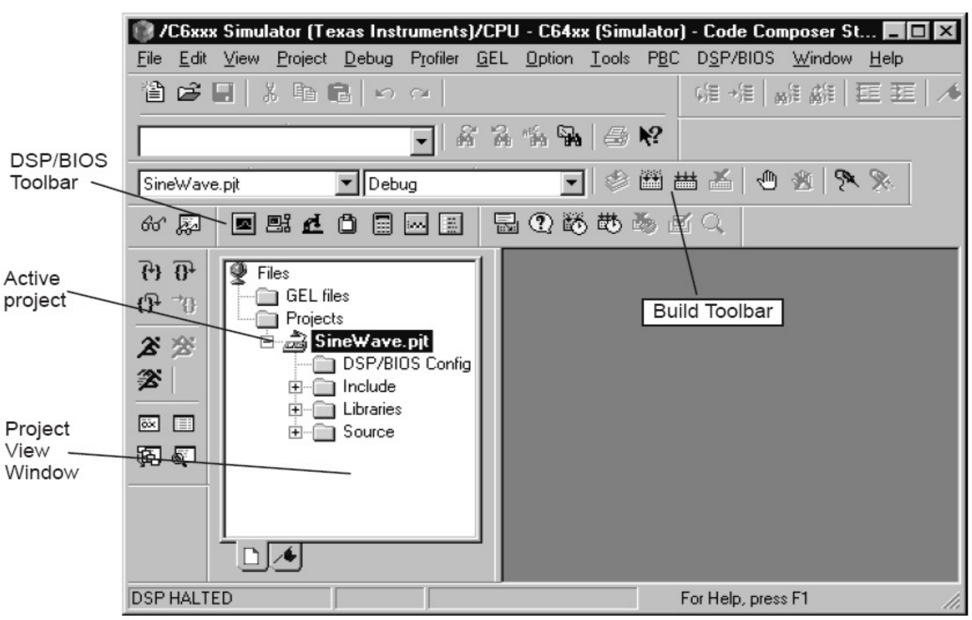


3.4. CSS work example

The following application realizes an input signal amplification control. In order to run the application it is necessary that CCS work with a ***C5416 Simulator.*** This configuration is made through the **Setup CCS** component.

Below are described the steps that need to be followed in order to develop an application in CCS and to be runned on the simulator.

1) Choose the menu **Project** -> **Open** and open the project *sinewave.pjt* from Turorial/sim54x/Sinewave. The project contains the files *sine.h, sinewave.cmd, rts500.lib* also the main source of the application *sine.c*.



Source of the application:

```
/*************************************************************
/*sine.c       ; This program uses Probe Points, a sinusoidal input signal and then
the a gain factor is applied to this signal.
/**************************************************************
**
#include <stdio.h>
#include "sine.h"

// gain control variable
int gain = INITIALGAIN;

// Buffer I/O declaration and initialization
BufferContents currentBuffer;

// function definitions
static void processing();          // processes the input and generate the
output
static void dataIO();              // the function used for ProbePoint

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

   while(TRUE) // loop
   {
      /* read the input data using a probe-point connected to a host file
         Write the output into a graph connected also through a probe-point */
      dataIO();

      /* in order to obtain the output data a gain is applied to the input data   */
      processing ();
   }
}
/* The next function applies a signal transform over the input signal in order to
generate the output signal. The parameters are: BufferContents structure that contains the
I/O streams of BUFFSIZE length; doesn't return any value */

static void processing()
{
   int size = BUFFSIZE;

   while(size--){                // applies the gain over the input
      currentBuffer.output[size] = currentBuffer.input[size] * gain;
   }
}
```

/* The next function reads the input signal and writes the processed output signal using ProbePoints; the function doesn't have any parameters and doesn't return any value. */

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

2) Compile the source files with **Project->Rebuild All** or press the Rebuild All from the toolbar.

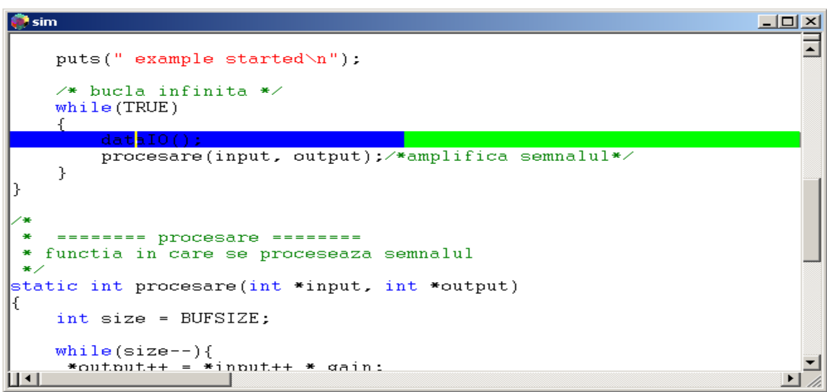
3) Load the object file in the DSP memory from **File->Load Program**. Select the program you just rebuilt, **sine.out** and press **Open**.

4) Double click on the file **sine.c** from **Project View.**

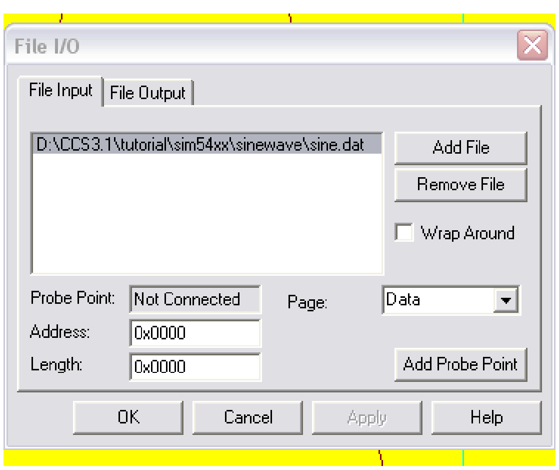
5) In order to attach the file containing the signal that needs to be processed it is necessary to insert a ProbePoint that reads data from a .dat file.

Place the cursor on the line **DataIO()** , from the *main* function. DataIO function reserves a place where adding's could be made later. For now we will connect a « **ProbePoint** » that introduces data from a PC file.

6) The **Toggle ProbePoint** from the toolbar. The line is highlighted on display.



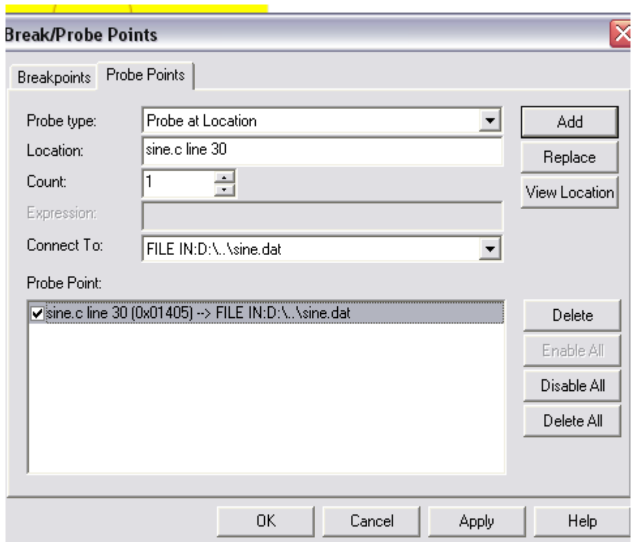
7) Choose **File -> File I/O.** File I/O dialogue appears in order to select the input and output files.



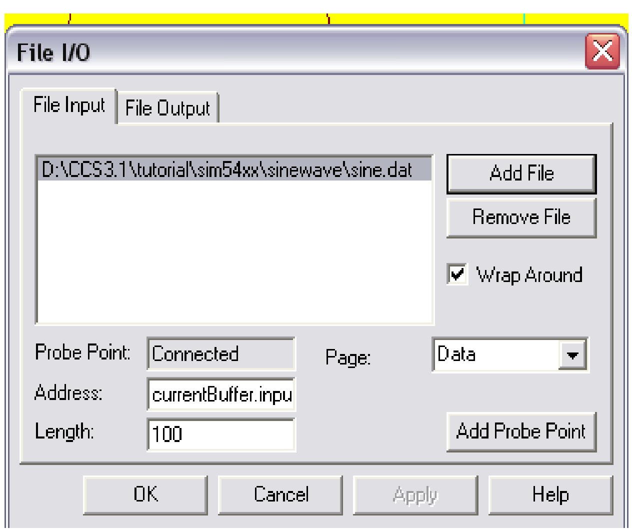
8) In **File Input** menu press **Add File.**

9) Choose sine.dat file. Notice that you can select the data format in the **Files of type**. The sine.dat file contains hex values for a sine. Select **Open** in order to add this file in **File I/O** dialogue list. Control window for the sine.dat file will appear. Later on when you run he program you can use this window for commands as: start, stop, rewind or fast forward in the data file.

10) In the **File I/O window** the next step is to add a ProbePoint using the button with the same name. In the dialogue window **BreakProbe Points** validate probe point clicking on it, will show us the location and we select **Connect to** choosing sine.dat file. At the end press Apply.

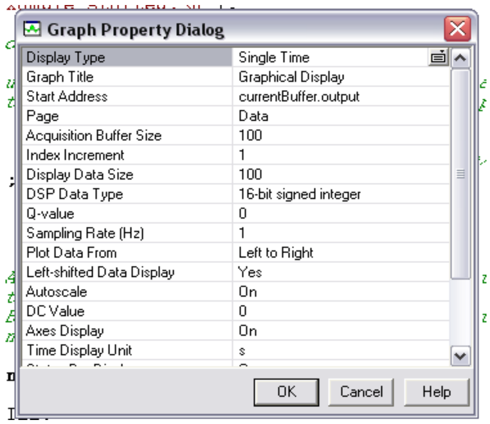


11) In the **File I/O** dialogue set the address *currentBuffer.input* and the length at 100. Also mark **Wrap Around.** Address field specifies where in the file is to be placed the data. *currentBuffer.input* is declared in volume.c as array of integer of BUFSIZE length. **Length** field indicates the number of samples from the data file that read each time it gets to **Probe Point.** 100 is the fixed value for the constant BUFSIZE in *volume.h* (0x64). **Wrap Around** option make CCS begin to read the file from the beginning after it arrived to its end. This allows data file to be taken as a continue data flow although it has just 1000 values and at each **Probe Point** 100 values are read.

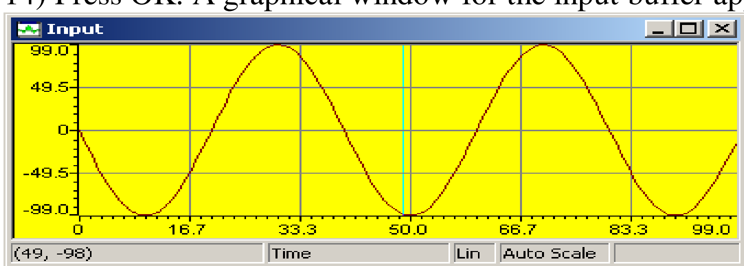


12) Select **View->Graph->Time/Frequency.**

13) In **Graph Property Dialog** modify: **Graph Title, Start Address, Acquisition Buffer Size, Display Data Size, DSP Data Type, Autoscale,** si **Maximum Y-value with** values from figure.



14) Press OK. A graphical window for the input buffer appears.

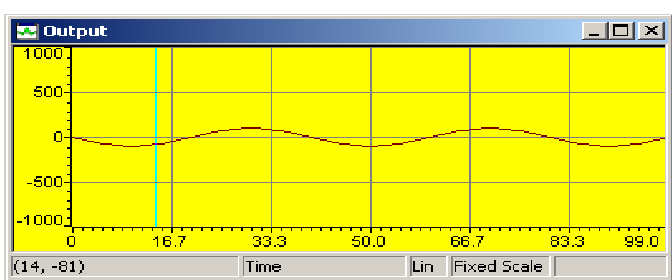


15) Right click on the Input window and choose **Clear Display** from the menu.

16) Choose **View->Graph->Time/Frequency.**

17) This time modify **Graph Title** in Output Buffer and **Start Address** in *currentBuffer.output*, **Autoscale** OFF .**Maximum-Value** at 1000.

18) Press OK in order to display Output graphical window- it contains the generated signal.



19) Right click on the graphical window and choose **Clear Display** from the menu.

20) Choose **File->Load Gel ->volume.gel**. This file contains a slider and dialogue control gel. Open it to see its content.

21) Select **Gel-> Application Control -> Gain.**

22) In the **Gain** window adjust the gain. In the output buffer window amplitude is modified. In addition the gain variable value from **Watch** it is modified every time we make an adjustment.

23) Click (**Halt**) or press **Shift F5** for stop the program.

4. Homework

a) Add a slider GEL control in order to modify the gain.

b) Write a program that generates a sine. Visualize the output buffer by using a Graph Window.

c) Replace the cursor control GEL with Dialogue control.

d) Modify the program so that the signal taken from the input file sine.dat to be amplitude modulated with a sine generated by the program.

e) Modify **Graph** in order to display the output signal in frequency domain.