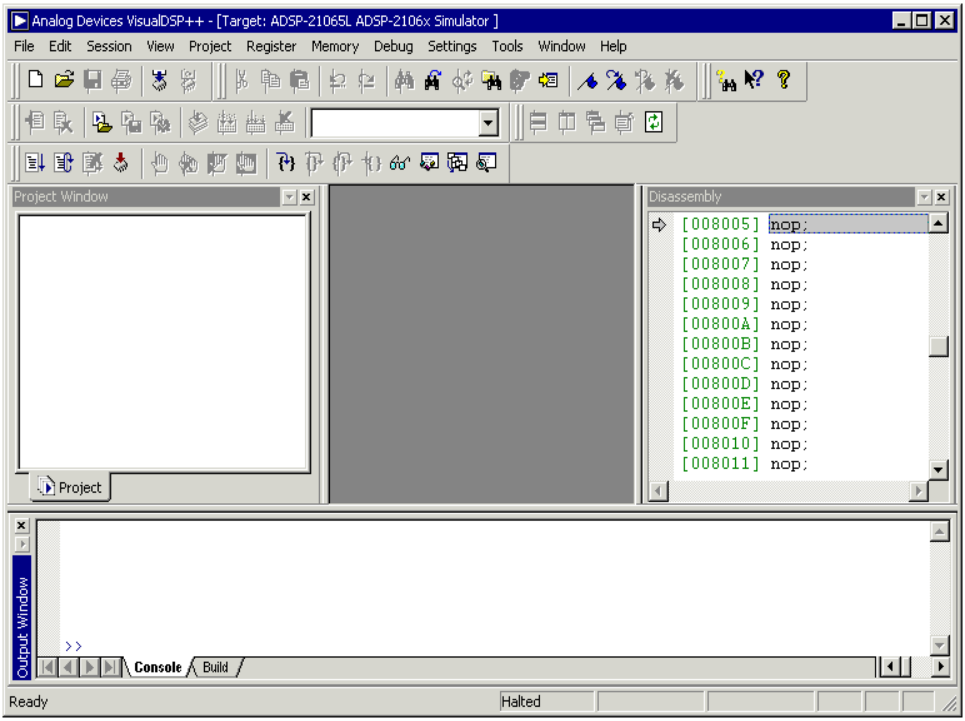
3 ENVIRONMENT

In This Chapter

This chapter contains the following topics:

Parts of the User Interface

VisualDSP++ is an intuitive, easy-to-use user interface for programming Analog Devices' DSPs. When you open VisualDSP++, the application's main window (Figure 3-1) appears.



```
[008005] nop;
[008006] nop;
[008007] nop;
[008008] nop;
[008009] nop;
[00800A] nop;
[00800B] nop;
[00800C] nop;
[00800D] nop;
[00800E] nop;
[00800F] nop;
[008010] nop;
[008011] nop;
```

```
>>
```

Figure 3-1. VisualDSP++ Main Window – Example

This work area contains everything you need to build, manage, and debug your DSP project. You can set up preferences that specify the appearance of application objects (fonts, visibility, and so on).

In addition to the main application window frame, VisualDSP++ includes these parts:

- Title bar
- Menu bar
- Project window
- Control menu
- Toolbars
- Output window
- Status bar

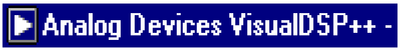
VisualDSP++ also provides many debugging windows to show you what is going on.

You need to learn only one interface to debug all your DSP applications.

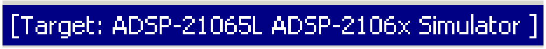
VisualDSP++ supports ELF/DWARF-2 (Executable Linkable Format, Debugging Information Format) executable files. VisualDSP++ supports all executable file formats produced by the linker.

Title Bar

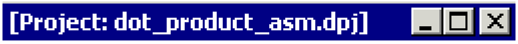
The three figures below show the different parts of the title bar.



Analog Devices VisualDSP++ -



[Target: ADSP-21065L ADSP-2106x Simulator]



[Project: dot_product_asm.dpj]

The title bar includes these components:

- Control menu button ▶
- Application name – Analog Devices VisualDSP++
- Name of the active target
- Project name
- Filename (when an Editor window is maximized in the main window)
- Standard Windows buttons

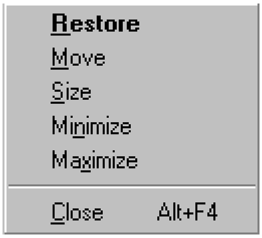
Clicking the control menu button opens the control menu, which contains commands for positioning, resizing, minimizing, maximizing, and closing the window. Double-clicking the control button closes VisualDSP++. The control menu and title bar right-click menu (see below) are identical.

Additional Information in Title Bars

A register window title bar displays its numeric format (such as hexadecimal). An Editor window title bar displays the name of the source file.

Title Bar Right-Click Menus

A menu like the one below appears when you right-click within the VisualDSP++ title bar or the title bar of a child window.

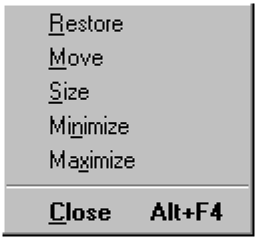


From the VisualDSP++ title bar's right-click menu, you can:

- Resize or move the application window
- Close VisualDSP++

Control Menu

Commands in a control menu (system menu, shown below) move, size, or close a window.



Program Icons

Click a program icon to open a control menu.

Program icon for the application and debugging windows

Program icon for Editor windows

When you place the mouse pointer over a control menu command, a brief description of the command appears in the status bar at the bottom of the application window.

Editor Windows

A floating Editor window's control menu includes **Next**, which moves the focus to another window.

When an Editor window floats in the main application window, its program icon resides at the left side of its title bar. When an Editor window is maximized, the program icon resides at the left end of the menu bar.

Debugging Windows

Each debugging window has a control menu. You can open a debugging window's control menu only when the window is floating in the main window. For more information, see "Debugging Windows" on page 3-50.

Menu Bar

The menu bar, shown in Figure 3-2, appears directly below the application title bar and displays menu headings, such as **File** and **Edit**.

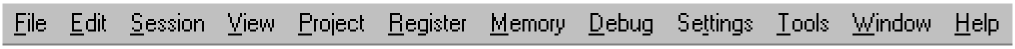


Figure 3-2. VisualDSP++ Menu Bar

To display menu commands and submenus, click a menu heading. You can also access many menu bar commands as follows:

- Click toolbar buttons
- Type keyboard shortcuts
- Right-click the mouse and choose a command from a context menu

Command Information

When the mouse pointer is over a menu bar command (or a toolbar button), a short description (tool tip) of the command appears in the status bar at the bottom of the main window.

Context-sensitive Help is available for each command.

To learn more about an individual menu command:

1. Press **Shift+F1** or click the toolbar's Help button ↖? .

   The pointer becomes a Help pointer ↖? .

2. Move the Help pointer over a menu command.

   If necessary, navigate through submenus.

3. Click the mouse.

   View the description of the command in the ensuing Help window.

Toolbars and User Tools

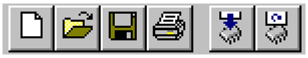
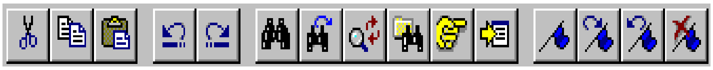
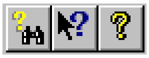
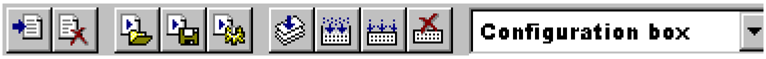
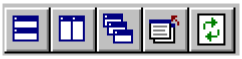
A toolbar is a set of buttons. You can run a command quickly by clicking a toolbar button.

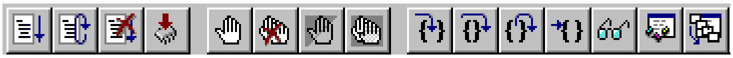
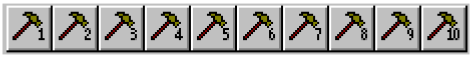
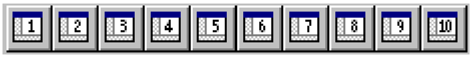
Use toolbars to organize the tasks you use most often. Position the toolbars on the screen for fast access to the tools that you plan to use.

The application includes standard (built-in) toolbars. You can create custom toolbars.

Built-In Toolbars

These are the standard (default) toolbars:

Name	Toolbar
File	
Edit	
Help	
Project	
Window	

Name	Toolbar
Debug	
User Tools	
Workspaces	

To obtain information about a tool, move the mouse pointer over the tool and press the **F1** key.

Toolbar Customization

By default, nine standard toolbars appear near the top of the application window, below the menu bar.

You can change the appearance of toolbars by:

- Moving, docking, or floating the toolbars
- Adding or removing buttons to or from toolbars
- Displaying *cool look* buttons, *large buttons*, or both

You can also:

- Hide toolbars from view
- Add and delete custom-built toolbars

Toolbars: Docked vs. Floating

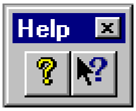
By default, toolbars are located under the application's menu bar. You can move them to the following locations:

- Over a docked window
- On the main window
- Anywhere on the desktop

When a toolbar is attached to a window, it is called a *docked* toolbar.

You can detach a toolbar from a window and move it to another location anywhere on the desktop. A *floating* toolbar is a stand-alone window, as it is not docked. A docked toolbar does not show its name, but a floating toolbar displays its title.

The following example is a floating Help toolbar.



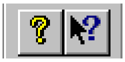
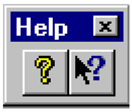
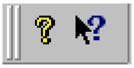
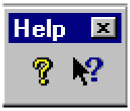
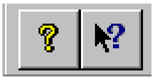
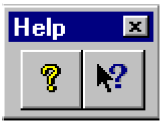
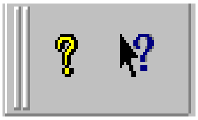
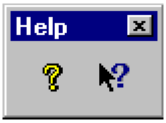
Toolbar Button Appearance

You can choose the appearance of the toolbar buttons. Two options, *cool look* and *large buttons*, provide slightly different button appearances.

The cool look option includes a pair of vertical bars on the toolbar's left side, but removes the square box from each button. The vertical bars visually separate toolbar buttons into groups (toolbars).

The large buttons option makes the area of each button larger.

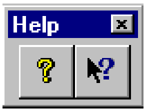
The figures in the following table demonstrate these options.

Option Settings	Docked	Floating
Cool look – Off Large buttons - Off		
Cool look – On Large buttons – Off		
Cool look – Off Large buttons – On		
Cool look – On Large buttons – On		

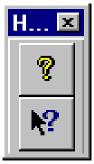
Toolbar Shape

You can change the shape of a floating toolbar. These examples show two toolbar shapes.

Horizontal



Vertical



Note: Depending on the number of tools in the toolbar, you can create other length and width arrangements.

Toolbar Rules

Be aware of these rules when working with toolbars.

- You can customize any built-in toolbar. For example, you can remove a button from the **File** toolbar, but you cannot delete a built-in toolbar.
- For a built-in toolbar, you can reset the buttons to their original default button set.
- You can prevent a toolbar from docking by pressing and holding the **Ctrl** key while dragging the toolbar to a new location.
- You can change the name of a user-defined toolbar, but you cannot change the name of a built-in toolbar. For example, you cannot change the name of the **File** toolbar.

- Moving a toolbar to the bottom of a window creates a new docking site and docks the toolbar. You can tell when a toolbar is going to dock by the size and shape of its moving outline as you drag it. Its outline becomes slightly smaller than its floating outline.

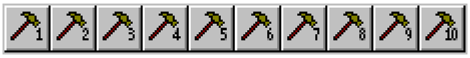
User Tools

Save time running commands by configuring user tools. You can configure up to ten user tools.

A user tool runs a command, which can:

- Contain parameters to launch an application
- Be a Tcl command

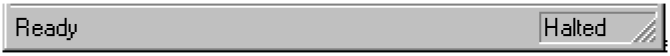
You access configured user tools from the **Tools** menu or from the **User Tools** toolbar, shown below.

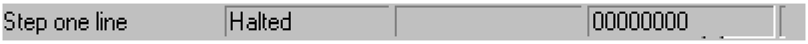


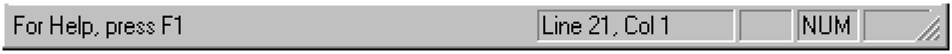
When a user tool is configured, its menu name (label) appears in the **Tools** menu. The label also appears when you move the mouse pointer over a user tool button.

Status Bar

The status bar, located at the bottom of the main application window, provides various informational messages. The following examples show different information displayed on the status bar.







The type of information that appears in the status bar depends on your context (what you are doing).

- When you move the mouse pointer over a toolbar button or a menu bar command, a brief description of the button or command appears.
- When you halt program operation with a **Halt** command, the address where the program halted appears.
- While running a trace, the word “`Running`” appears.
- While using some Tcl commands, the status bar provides information, such as when the menu item has focus.

While you are editing a file, the right side of the status bar provides Editor window information:

Item	Indicates
Line ###	Cursor current line number
Col ###	Cursor current column number
CAP	The keyboard's Caps Lock key is latched down
NUM	The keyboard's Num Lock key is latched down
SCRL	The keyboard's Scroll Lock key is latched down

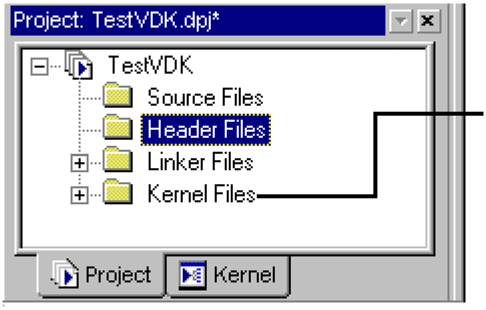
VisualDSP++ Windows

From the application's main window, you can open a Project window, Editor windows, an Output window, and various Debugging windows.

Project Window

The Project window has a **Project** tab .

When a project is VDK-enabled, the **Kernel** tab also appears, as shown in Figure 3-3.



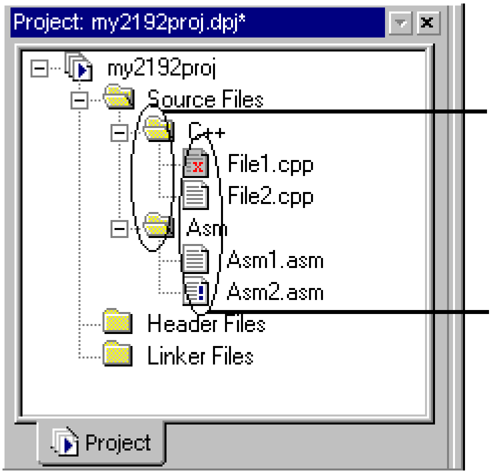
A Kernel folder appears only when the project includes VisualDSP++ kernel support

Figure 3-3. Project Window – with **Kernel** Tab

Project Tab Page

The **Project** tab page displays a tree of your project's folders and files. Nodes are arranged in a hierarchy similar to the file structure in Windows Explorer.

Figure 3-4 shows the node hierarchy on the **Project** tab page.



Add folders to organize your files.

File icons indicate build options.

Figure 3-4. Project Window – Node Hierarchy on the **Project** Tab Page

Project Nodes

The Project window comprises three types of nodes.

Node	Icon	Description
Project	▶	Only one project is permitted
Folder	📁	A closed folder
	📂	An opened folder, revealing its contents
File	📄	A file that uses project settings
	!	A file with options that differ from the project options
	X	A file excluded from the current configuration

Project Tab Page Right-Click Menus

The following right-click menus (also called popup menus or context menus) are available:

From	Description
Node	The menu's content depends on the context (selected node).
Title bar	From this menu, you can dock, hide, or float the window. You can also view project properties.

Project Folders

Project window folders 📁 📂 organize files within a project. You can specify properties for folders.

The following rules apply to folders:

- Folders can be nested to any depth.
- Folders carry no attributes to the build process, as they do not reflect the file system. Folders do not appear in directory listings, as in Windows Explorer.
- When you automatically add files to the project tree, each file is placed in the first folder that has been configured with the same file extension. After automatic placement, you can move a file anywhere manually.

Tip: To move a file out of one folder and into another folder, select the file and drag it onto the other folder.

Project Files

In the Project window, files are represented by the following icons:

🗎 Files that use project options

! Files that use options that differ from project options

✗ Files excluded from the current configuration

The files appear in an expandable and collapsible node tree.

Source files are the C/C++ language or assembly language files in your project. Source files provide the project with code and data. You can add, delete, and modify source files.

Each project must include a `.LDF` *file*, which contains command input for the linker. If you do not include a `.LDF` file in the project, the project is built with a default `.LDF` file.

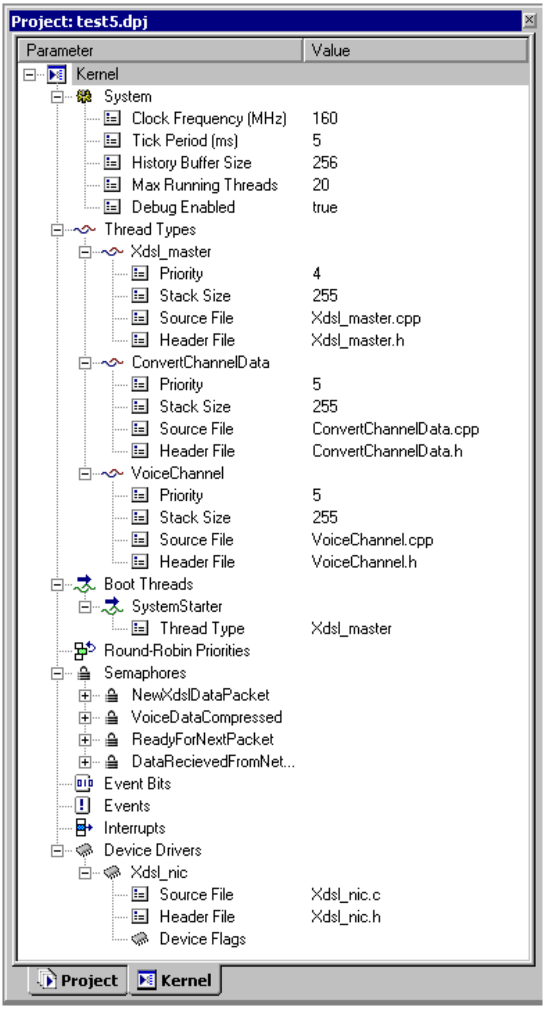
Besides source files, a DSP project may use additional files. Optionally, a project can include data files and header files.

Kernel Tab Page

The **Kernel** tab of the Project window is available only to VDK-enabled projects.

From this tab page, you can add, modify, and delete kernel elements such as thread types, priorities, semaphore, and events. VisualDSP++ automatically updates `vdk_config.cpp` and `vdk_config.h` to reflect the changes you make from the **Kernel** tab page.

The example in Figure 3-5 shows an expanded view of the elements on the **Kernel** tab page for a VDK-enabled project.



Parameter	Value
Kernel	
System	
Clock Frequency (MHz)	160
Tick Period (ms)	5
History Buffer Size	256
Max Running Threads	20
Debug Enabled	true
Thread Types	
Xdsl_master	
Priority	4
Stack Size	255
Source File	Xdsl_master.cpp
Header File	Xdsl_master.h
ConvertChannelData	
Priority	5
Stack Size	255
Source File	ConvertChannelData.cpp
Header File	ConvertChannelData.h
VoiceChannel	
Priority	5
Stack Size	255
Source File	VoiceChannel.cpp
Header File	VoiceChannel.h
Boot Threads	
SystemStarter	
Thread Type	Xdsl_master
Round-Robin Priorities	
Semaphores	
NewXdslDataPacket	
VoiceDataCompressed	
ReadyForNextPacket	
DataRecievedFromNet...	
Event Bits	
Events	
Interrupts	
Device Drivers	
Xdsl_nic	
Source File	Xdsl_nic.c
Header File	Xdsl_nic.h
Device Flags	

Figure 3-5. **Kernel** Tab Page – Expanded View of VDK Elements

Refer to the *VisualDSP++ Kernel (VDK) User's Guide* for complete details about VDK.

Project Window Right-Click Menus

From the Project window you can access four different right-click menus that enable you to operate on Project window objects (the project, folders, or files). These right-click menus are:

- Project window menu
- Project icon menu
- Folder icon menu
- File icon menu

Depending on the context (selected object), right-click menus provide an alternative means of performing an action. You can perform many of the actions from the menu bar commands or toolbar buttons.

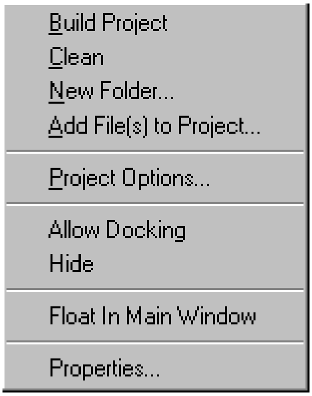
Project Window Menu

The Project window's right-click menu enables you to:

- Hide the Project window from view
- Dock the Project window to the frame
- Float the Project window
- View project properties

Project Icon Menu

The Project icon right-click menu is shown below.

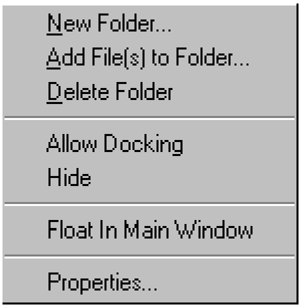


This menu provides a *project* context, which enables you to:

- Build the project
- Clean (delete intermediate and target files)
- Add folders and files
- View and specify project options
- View project properties

Folder Icon Menu

The Folder icon right-click menu is shown below.

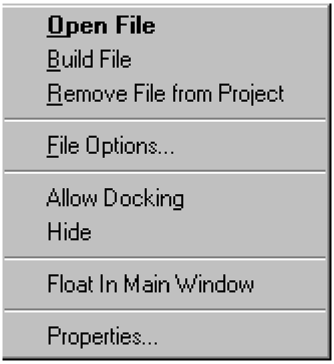


The folder menu provides a *container* context from which to perform these local operations:

- Add or delete a folder
- Add files to the folder
- View folder properties

File Icon Menu

The File icon right-click menu is shown below.



This menu provides a *file* context from which to perform these operations:

- Open the selected file for editing
- Build the file
- Remove the file from a project
- Specify options for the file
- View the file's properties

Automatic File Placement

Automatic file placement helps you save time by automatically placing new project files into folders. Placement is determined by the file extensions associated with each folder.

By default, project folders are associated with these file extensions:

Folder	Default Associations
Source Files	.c, .cpp, .cxx, .asm, .dsp
Header Files	.h, .hpp, .hxx
Linker Files	.ldf, .dlb

If you choose, you can change the file extensions associated with a folder.

Rules

The following rules apply to file placement:

- Any file added to a project is placed in the first folder located with the same extension.
- If no folders exist, the new file goes to the project level.

Example

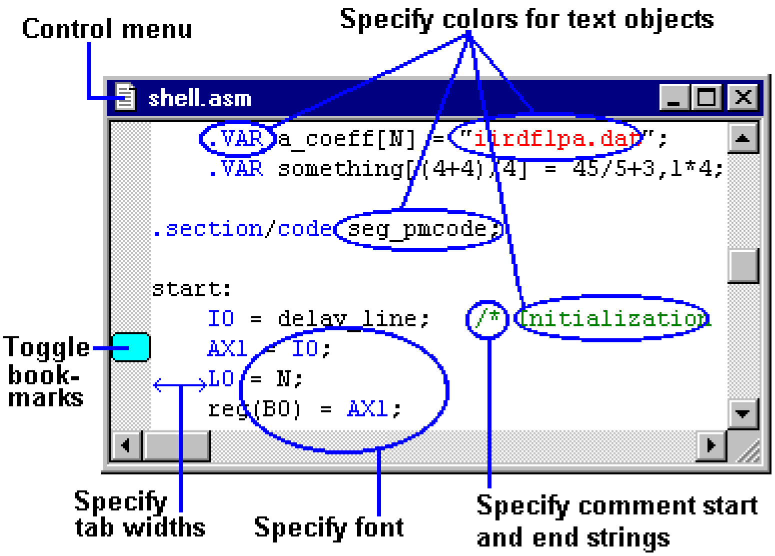
You create a folder labeled “`C Source Files`” and specify it with `.c`, `.cpp`, and `.cxx` file extensions. You create a second folder labeled “`Asm Files`” and associate it with .asm files.

If you add three files (`file1.cpp`, `file1.asm`, and `file2.c`) to the project, `file1.cpp` and `file2.c` are automatically placed in the `C Source Files` folder, and `file1.asm` goes into the `Asm Files` folder.

Note: After automatic placement, you can manually place a file anywhere.

Editor Windows

Use Editor windows to develop source code and edit project files. Figure 3-6 shows items that you can customize in Editor windows.


Control menu
Specify colors for text objects
shell.asm
Toggle book-marks
Specify tab widths
Specify font
Specify comment start and end strings

```
    .VAR a_coeff[N] = "iirdflpa.dat";
    .VAR something[(4+4)/4] = 45/5+3,1*4;

.section/code seg_pmcode;

start:
    I0 = delay_line;    /* initialization
    AX1 = I0;
    L0 = N;
    reg(B0) = AX1;
```

Figure 3-6. Editor Window – Customization

Open as many Editor windows as you like. Editor windows support the following features and functions:

- User-defined color-coded comments, strings, keywords, and tabs
- Print, print preview, and user-defined headers and footers
- Bookmarks
- Find and replace, wrap-around search, regular expression matching
- Go to a specified line number
- Jump to the next or previous syntax error
- Copy, cut, paste, undo and redo more than 500 levels of edits for each open file

Use the Editor window's right-click menu to perform these functions:

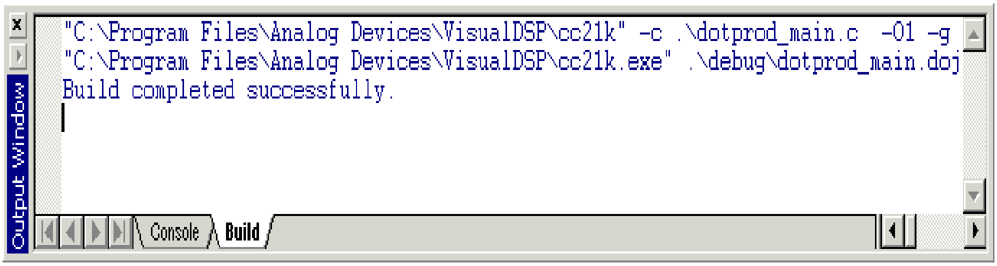
- Undo or redo the last edit
- Cut, copy, or paste text
- Toggle a bookmark

Output Window

The Output window:

- Displays standard I/O text messages such as file load status and error messages
- Displays build status information for the current project build
- Provides access to errors in source files
- Acts as an interface to Tcl

The Output window shown in Figure 3-7 contains build status information.



```
"C:\Program Files\Analog Devices\VisualDSP\cc21k" -c .\dotprod_main.c  -O1 -g
"C:\Program Files\Analog Devices\VisualDSP\cc21k.exe" .\debug\dotprod_main.doj
Build completed successfully.
```

Figure 3-7. Output Window with Build Status Information

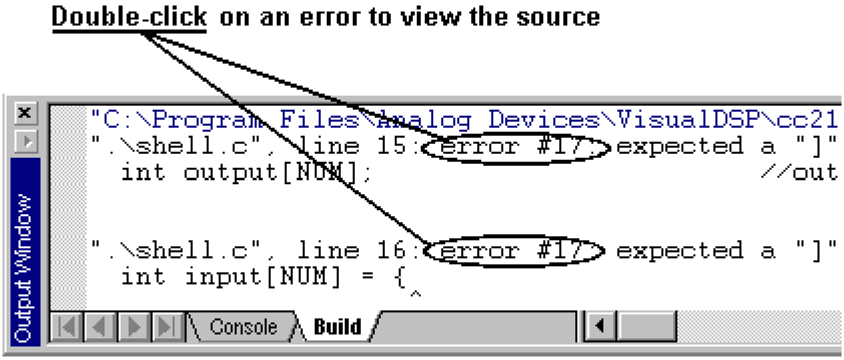
Display the Output window by choosing **Output Window** from the **View** menu.

Output Window Tabs

The Output window's two tabs, **Console** and **Build**, provide different information and capabilities.

Build Tab

The **Build** tab page (Figure 3-8) displays error messages generated during a build. Double-click on an error message to jump to the offending code in an Editor window.



Double-click on an error to view the source

```
"C:\Program Files\Analog Devices\VisualDSP\cc21
".\shell.c", line 15: error #17: expected a "]"
  int output[NUM];                    //out

".\shell.c", line 16: error #17: expected a "]"
  int input[NUM] = {
                    ^
```

Figure 3-8. Output Window – Error Messages

Tip: Scroll through error messages by choosing **Next Error** or **Prev Error** from the **Edit** menu.

By default, VisualDSP++ output is blue, and tool output is black, but you can change these colors in the **Preferences** dialog box.

Console Tab

From the Output window's **Console** tab page (Figure 3-9), you can:

- View VisualDSP++ or target status error messages
- View STDIO output from C/C++ programs
- View I/O (streams) messages
- Scroll through previous commands. Just press the keyboard's up arrow (↑) and down arrow (↓) keys.
- Perform multi-line selection, copy, paste, and clear
- Issue Tcl commands
  - Auto-complete Tcl commands
  - Execute a previously issued Tcl command. Just double-click on the command.
  - Enter multi-line Tcl commands by adding a backslash character (\) to the end of a statement.
- Use bookmarks
  - Toggle a bookmark by pressing **Ctrl+F2**
  - Move to the next bookmark by pressing the keyboard's **F2** key

All text displayed on the **Console** tab page is also written to the VisualDSP++ log file.

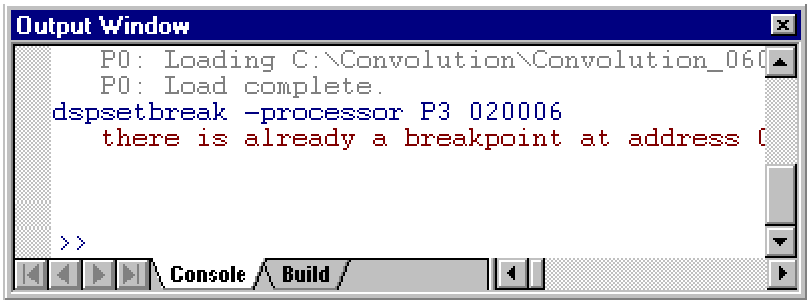
Output Window Customization

You can specify preferences that:

- Configure Output window fonts and colors
- Enable command auto-completion

By default, the Output window resides at the bottom of the main application window. You can resize or move the Output window, which is a Windows docking bar, to a different portion of the screen by dragging it to the selected location.

The Output window's **Console** tab can interact with the VisualDSP++ Tcl engine. All Tcl input and output is sent to the **Console** tab page, shown in Figure 3-9.



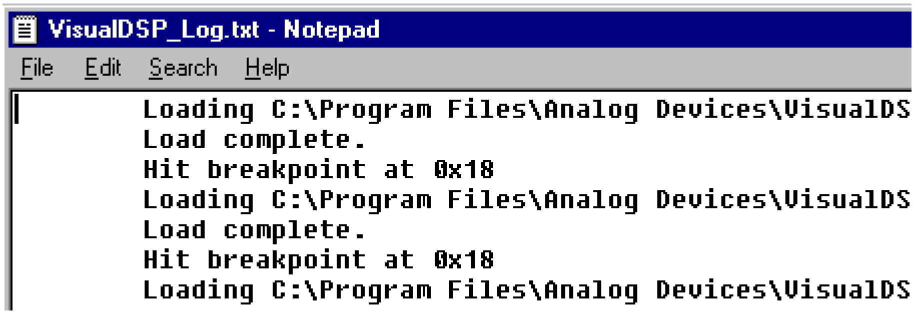
```
   P0: Loading C:\Convolution\Convolution_06
   P0: Load complete.
dspsetbreak -processor P3 020006
   there is already a breakpoint at address
>>
```

Figure 3-9. Output Window – **Console** Tab Page

Log File

The VisualDSP++ log file contains all status and error messages written to the Output window's **Console** tab page.

Figure 3-10 shows a sample log file.



```
Loading C:\Program Files\Analog Devices\VisualDS
Load complete.
Hit breakpoint at 0x18
Loading C:\Program Files\Analog Devices\VisualDS
Load complete.
Hit breakpoint at 0x18
Loading C:\Program Files\Analog Devices\VisualDS
```

Figure 3-10. Sample Log File

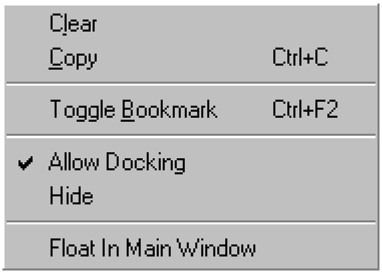
These messages are saved to the log file `VisualDSP_Log.txt`, which is located in the `C:\Program Files\Analog Devices\VisualDSP\Data` directory.

Notes:

- The above file path assumes that you installed VisualDSP++ by accepting default settings.
- All sessions append to the log file. Occasionally, open the file and delete parts of it (or all of it) to conserve disk space.

Right-Click Menu

The Output window's right-click menu is shown below.



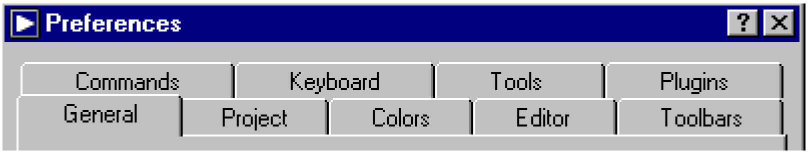
This menu enables you to:

- Dock the window
- Float the window
- Hide the window. Display the hidden window by choosing **Output Window** from the **View** menu.
- Copy selected text
- Clear the text in the windows
- Toggle bookmarks

Environment Customization

User Preferences

You can customize your work environment. Use the tab pages of the **Preferences** dialog box to specify user preferences. These options affect many aspects of the VisualDSP++ user interface.



These are some of the preferences you can specify:

- Toolbars (content and appearance)
- Keyboard shortcuts (modify or add)
- Fonts used in debugging windows and the Output window
- Colors in debugging windows and the Output window
- Project options for files (saving, synchronization, editing), project dependencies, and reload options
- Font, color, and spacing (tabs) for various file types in an Editor window

Other Available Customizations

VisualDSP++ enables you to customize these additional options:

- Editor window appearance (source mode vs. mixed mode)
- Command auto-completion
- Pipeline support
- Colors of focused and unfocused windows
- Workspace (specifying, saving, and loading)
- Visibility of objects (such as the status bar)
- Display of the Target Halted message
- Running to main after load
- Docking new windows
- Recycling Editor windows
- Standard I/O (STDIO) support
- Auto loading an executable after a successful build

Work Environment Customization

The user interface comprises many parts. Arrange portions of the interface to meet your needs.

You can customize the following components to meet your needs:

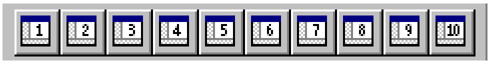
- Toolbars
- Workspaces
- Windows
- Application user preferences

In addition, you can show or hide the Output window, status bar, toolbars, or tool tips.

Workspace Specification

Open multiple windows and place them anywhere you want. After you set up the windows, save the window configuration as a **workspace**, which you can restore (load) at another time.

You can create up to ten workspaces and quickly switch among them via the Workspace toolbar (shown below), menu commands, or keyboard shortcuts.



Keyboard shortcuts (**ALT+1**, **ALT+2**, …, **ALT+0**) provide fast access to Workspace 1, Workspace 2, …, Workspace 10, respectively. Modify the keyboard shortcuts to meet your needs.

Each debug session maintains a default workspace that is automatically saved when you close the debug session. This workspace is automatically restored when you load that session.

Window Color Customization

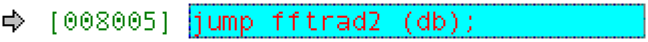
You can specify window colors for items such as symbols and data that appear in VisualDSP++ debugging windows.

One set of options configures the display of objects in all the debugging windows other than the Output window. Another set of options configures the appearance of the Output window.

You specify colors for all debugging windows (other than the Output window) from the **Preferences** dialog box (**General** category on the **Colors** tab page). You can select a color for each of the following items:

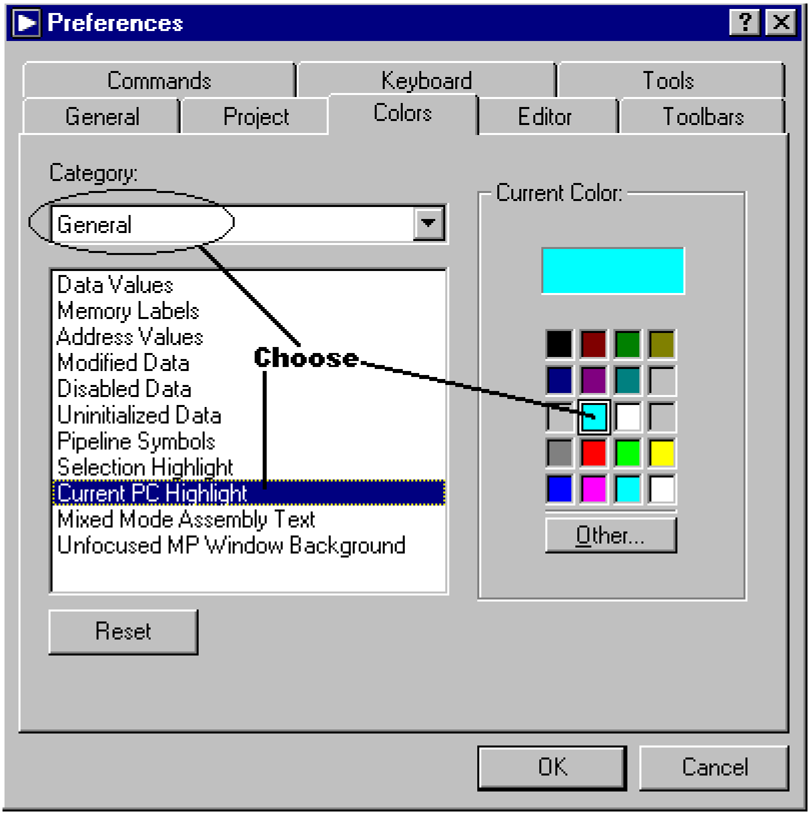
- Data values
- Memory labels
- Address values
- Modified data
- Disabled data
- Uninitialized data
- Pipeline symbols
- Selection highlight
- Current PC highlight
- Mixed mode assembly text

A horizontal bar in the Disassembly window highlights the current source line to be executed, as shown in the following example.



```
[008005] jump fftrad2 (db);
```

By default, the bar is light blue. You can change its color by configuring **Current PC Highlight** on the **Colors** tab of the **Preferences** dialog box, shown in Figure 3-11.



Category: General

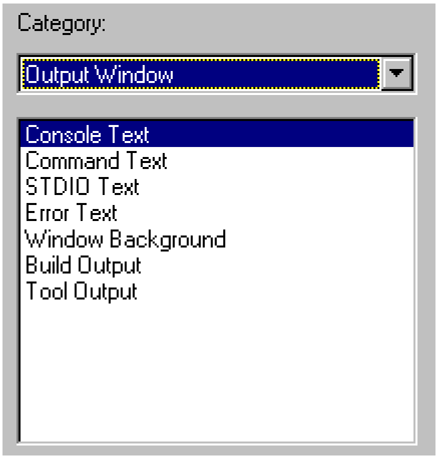
Current Color:

- Data Values
- Memory Labels
- Address Values
- Modified Data
- Disabled Data
- Uninitialized Data
- Pipeline Symbols
- Selection Highlight
- Current PC Highlight
- Mixed Mode Assembly Text
- Unfocused MP Window Background

Choose

Figure 3-11. Preferences Dialog Box – Colors Tab Page, General Category

You can customize colors for various categories, such as the **Output Window**, selected below.



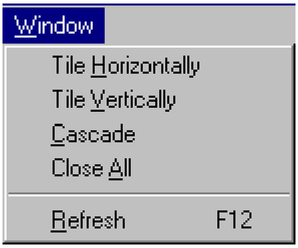
For the **Output Window** category, you can customize the color of these Output window items:

- Console text
- Command text
- STDIO text
- Error text
- Window background
- Build output
- Tool output

Window Operations

Window Manipulation

The **Window** menu commands, shown below, enable you to manipulate your windows display and update windows during program execution. Refer to your Windows documentation for more information.



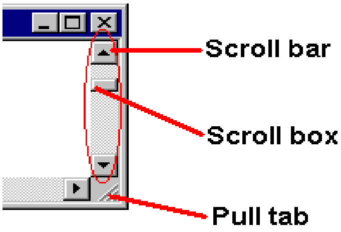
Right-Click Menu Options

Each window presents a menu when you right-click in the window or on its title bar. Select these menu options to affect window's behavior.

Option	Description
Allow Docking	Enables/disables docking
Close	Closes the window
Float in Main Window	Causes the window to become a normal MDI child window (like an Editor window) and disables its docking ability

Scroll Bars and Resize Pull-Tab

Scroll bars appear along the right and bottom edges of the application or document window, as shown below.



The scroll boxes inside the scroll bars indicate your vertical and horizontal location in the document. Use the mouse to scroll to other parts of the document.

When the application window is not maximized, the resize pull-tab appears in the lower-right corner of the window. Click and drag the pull-tab to resize the application window.

Windows: Docked vs. Floating

When a window is attached to the application's frame, it is referred to as a *docked window*.

You can detach a window from the main window and move it to another location anywhere on the desktop. A *floating window* stands alone, because it is not docked.

Depending on your needs, you can:

- Dock a window to the application's main window (frame)
- Float a window

A window's right-click menu provides commands to dock or float the window. The **Allow Docking** option and the **Float In Main Window** option are mutually exclusive.

Example of a Docked Window

The Project window shown in Figure 3-12 is docked (**Allow Docking** is selected):

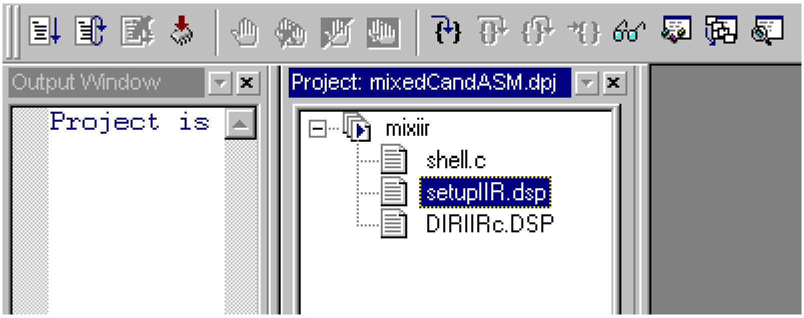


Figure 3-12. Project Window – Docked

Tip: To prevent a window from docking, hold down the keyboard's **Ctrl** key while dragging the window to another position.

Examples of Floating Windows

The following illustrations show floating (not docked) windows.

The Project window in Figure 3-13 is floating in the main window (**Float In Main Window** is selected):

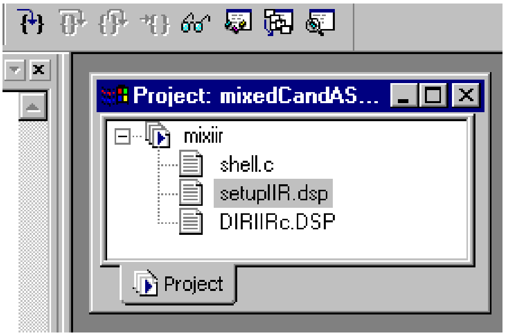


Figure 3-13. Project Window – Floating (Example 1 of 3)

The Project window in Figure 3-14 is also floating in the main window (**Float In Main Window** is selected):

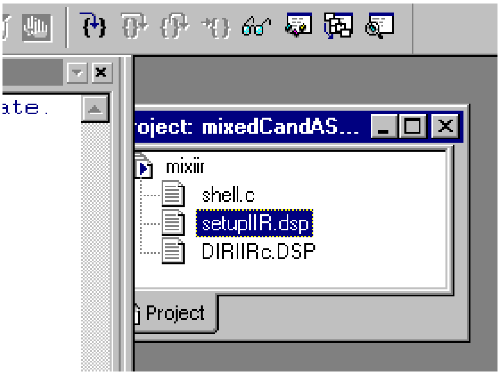


Figure 3-14. Project Window – Floating (Example 2 of 3)

The Project window in Figure 3-15 is floating, but not in the main window (**Float In Main Window** is not selected)

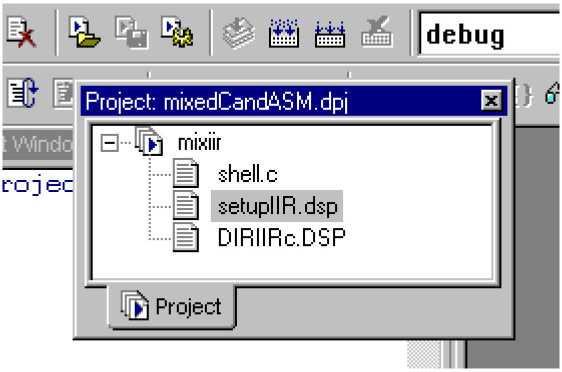


Figure 3-15. Project Window – Floating (Example 3 of 3)

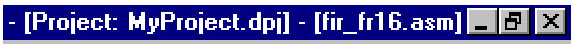
Window Position Rules

These rules apply to window positions:

- Unless **Allow Docking** is disabled, a window must reside within the main window.
- An Editor window cannot be docked to the main window.
- A window specified as an MDI child cannot be positioned over a docked window.
- Unless the Output window is floating in the main window, you cannot position a window specified as an MDI child over the Output window.

Standard Windows Buttons

The standard Windows buttons are located on the right side of the title bar, as shown below.



- [Project: MyProject.dpj] - [fir_fr16.asm]

These buttons resize and close the window as described in the following table:

Button	Purpose
_	**Minimize** reduces the window to its Windows icon.
☐	**Maximize** enlarges the window to fill the screen.
🗗	**Restore**, which appears only after maximizing the window, returns the window to its last non-minimized, non-maximized position.
☒	**Close** closes the application window and exits the program.

Debugging Windows

VisualDSP++ provides these debugging windows to display DSP program operation and results:

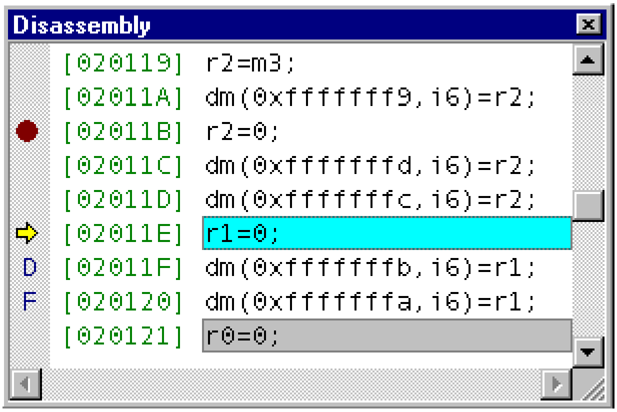
Window	Provides
Output	The **Console** tab displays standard I/O text messages such as file load status, and error messages and streams. You can interactively enter Tcl commands and view Tcl output. The **Build** tab displays build messages.
Editor	Syntax coloring, context-sensitive expression evaluation, and status icons that indicate breakpoints, bookmarks, and the current PC position
Disassembly	Code in disassembled format. This window provides fill and dump capability.
Expressions	The means to enter an expression and see its value as you step through program execution
Trace	A history of processor activity during program execution, including buffer depth (instruction lines), cycle count, and instructions executed such as memory fetches, program memory writes, and data/memory transfers
Locals	All local variables within a function. Use this window with step or halt commands to display variables as you move through your program.
Linear Profiling Results	(Simulation only) The target's PC register is sampled at every instruction cycle to provide an accurate picture of where instructions were executed. Linear profiling is much slower than statistical profiling.

Window	Provides
Statistical Profiling Results	(JTAG emulation only) VisualDSP++ randomly samples the target processor's program counter (PC) and presents a graphical display of the resulting samples, showing where the application spends time.
Call Stack	A means of moving the call stack back to the previous debug context
Register	Current values of registers. You can change register contents and change the number format.
Memory	A view of DSP memory. Similar number format and edit features as register windows, plus fill and dump capability.
Memory Map	The memory map of the selected processor
Plot	A graphical display of values from memory addresses. The window supports linear and FFT (real and complex) visualization modes and allows you to export an image to a file, the clipboard, or to a printer.
Pipeline	Instruction pipeline

Disassembly Windows

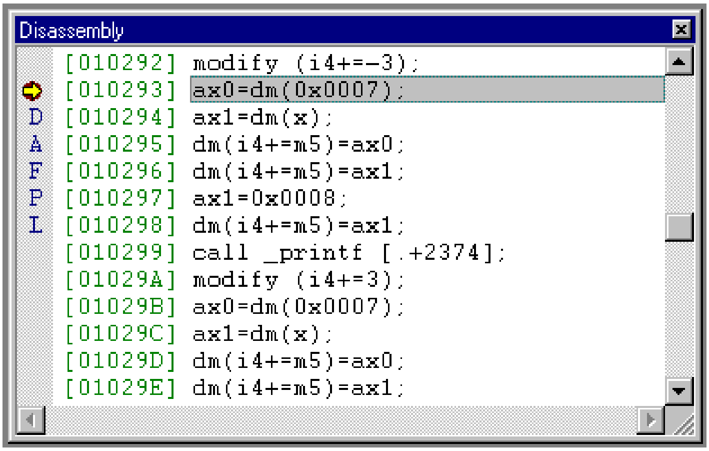
By default, a Disassembly window appears when you open a new session.

Figure 3-16 and Figure 3-17 show examples of Disassembly windows.



```
[020119] r2=m3;
[02011A] dm(0xfffffff9,i6)=r2;
[02011B] r2=0;
[02011C] dm(0xfffffffd,i6)=r2;
[02011D] dm(0xfffffffc,i6)=r2;
[02011E] r1=0;
[02011F] dm(0xfffffffb,i6)=r1;
[020120] dm(0xfffffffa,i6)=r1;
[020121] r0=0;
```

Figure 3-16. Disassembly Window (Example 1 of 2)



```
[010292] modify (i4+=-3);
[010293] ax0=dm(0x0007);
[010294] ax1=dm(x);
[010295] dm(i4+=m5)=ax0;
[010296] dm(i4+=m5)=ax1;
[010297] ax1=0x0008;
[010298] dm(i4+=m5)=ax1;
[010299] call _printf [.+2374];
[01029A] modify (i4+=3);
[01029B] ax0=dm(0x0007);
[01029C] ax1=dm(x);
[01029D] dm(i4+=m5)=ax0;
[01029E] dm(i4+=m5)=ax1;
```

Figure 3-17. Disassembly Window (Example 2 of 2)

Disassembly windows display code in disassembled form, which is useful for temporarily modifying the code to test a change or to view code when no source is available. The Disassembly window allows you to examine the assembly code generated by the C/C++ compiler.

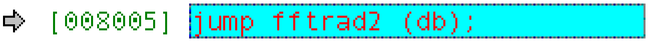
Note: To make changes permanent, modify the code, and rebuild the project.

Disassembly windows provide:

- Number format and edit features, similar to register windows
- Dump and fill capability
- Symbols at the far left of the window, denoting program execution stages and pipeline stages

  You can enable and disable the display of pipeline symbols while in mixed mode (C/C++ and assembly).

By default, the current source line to be executed is highlighted by a light-blue horizontal bar, as shown in the following example:



```
⇨ [008005] jump fftrad2 (db);
```

You can configure the color of the current source line and other window items.

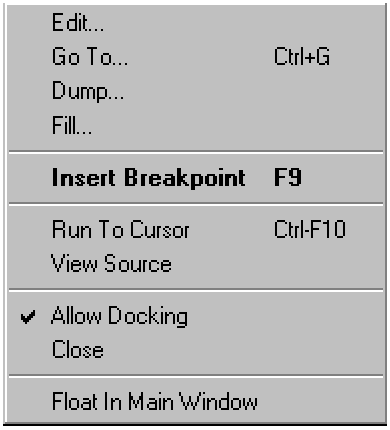
Other Disassembly Window Features

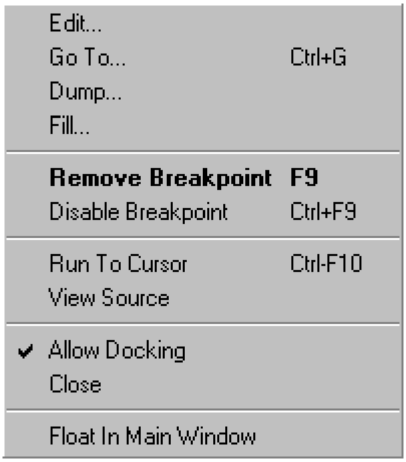
From the Disassembly window, you can do the following:

To	Place the mouse pointer over
Move to a different address	An address field and double-click. Then select the address from the ensuing **Go To** dialog box.
Insert or remove a breakpoint	An instruction and double-click
Toggle (enable or disable) a breakpoint	An instruction and right-click. Then choose the appropriate command from the ensuing menu.

Right-Click Menu

The Disassembly window's right-click menu provides access to these commands:





Disassembly Window Symbols

The Disassembly window denotes program execution stages with symbols at the far left of the Disassembly window.

The symbols, located at the left of the Disassembly window, are:

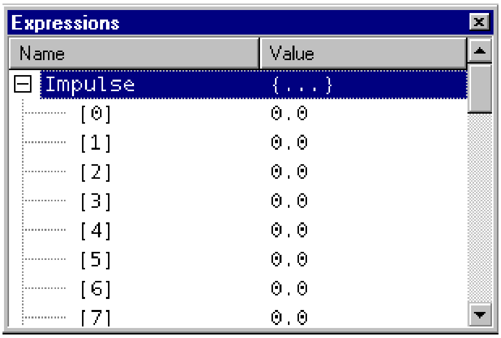
Symbol	Description
⇨	Current source line
⇨	The current instruction is being aborted due to a branch or jump instruction
●	A breakpoint is enabled
○	A breakpoint is disabled
F	This instruction is currently in the Fetch Address stage of the pipeline.
D	This instruction is currently in the Instruction Decode stage of the pipeline.
E	This instruction is currently in the Execute stage of the pipeline.

Note: The display of pipeline stages is available only when:

- The selected session is connected to a simulator target.
- **Enable pipeline display** is selected on the **General** tab page of the **Preferences** dialog box, available from the **Settings** menu.

Expressions Window

The Expressions window, shown in Figure 3-18, enables you enter an expression to evaluate in your program. Expression evaluations are based on the current debug context.

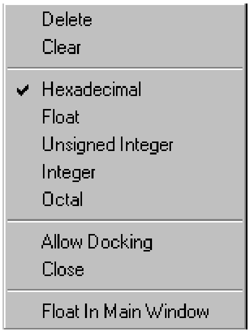


Name	Value
Impulse	{...}
[0]	0.0
[1]	0.0
[2]	0.0
[3]	0.0
[4]	0.0
[5]	0.0
[6]	0.0
[7]	0.0

Figure 3-18. Expressions Window

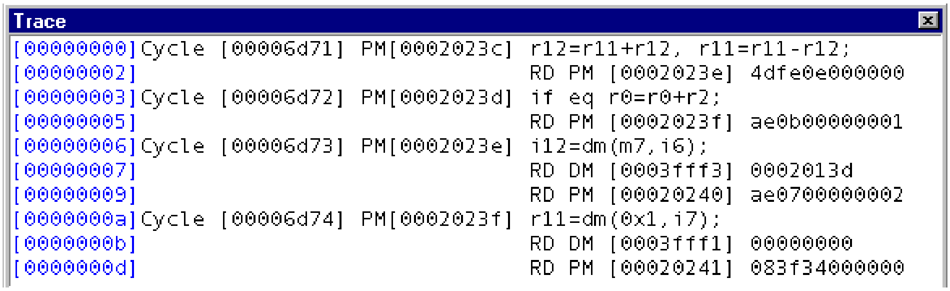
Note: Because of the way registers are saved and restored on the stack, the register value on which the expression relies may be incorrect if you change VisualDSP++'s context with the Call Stack window.

The Expressions window's right-click menu includes commands that let you change the display's number format.



Trace Windows

You perform a trace (also called an execution trace or a program trace) to analyze the run-time behavior of your DSP program, to enable I/O capabilities, and to simulate source-to-target data streaming. Figure 3-19 shows data displayed in a Trace window.



```
[00000000]Cycle [00006d71] PM[0002023c] r12=r11+r12, r11=r11-r12;
[00000002]                              RD PM [0002023e] 4dfe0e000000
[00000003]Cycle [00006d72] PM[0002023d] if eq r0=r0+r2;
[00000005]                              RD PM [0002023f] ae0b00000001
[00000006]Cycle [00006d73] PM[0002023e] i12=dm(m7,i6);
[00000007]                              RD DM [0003fff3] 0002013d
[00000009]                              RD PM [00020240] ae0700000002
[0000000a]Cycle [00006d74] PM[0002023f] r11=dm(0x1,i7);
[0000000b]                              RD DM [0003fff1] 00000000
[0000000d]                              RD PM [00020241] 083f34000000
```

Figure 3-19. Trace Window

The Trace window displays:

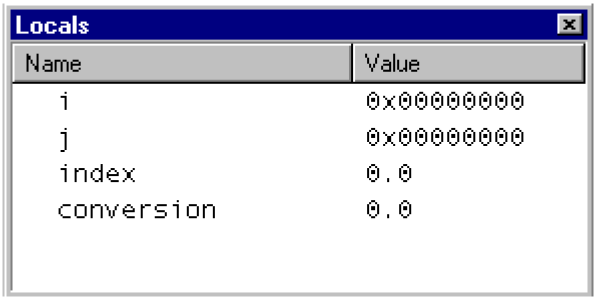
- Buffer depth (as specified by the **Custom** depth box in the **Trace Buffer Depth** dialog box)
- The clock cycle when the instruction occurred
- The address of the instruction that was executed
- The disassembled instruction

Memory results have the following fields:

- Access type (RD or WR)
- Memory type (PM or DM)
- The address, in brackets ([ ])
- The data value that was written or read

Locals Window

The Locals window displays the value of local variables within a function, as shown in Figure 3-20.



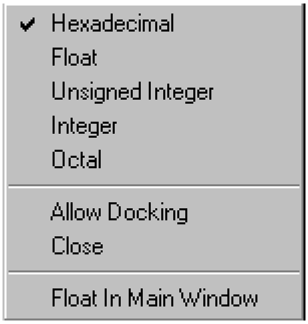
Name	Value
i	0x00000000
j	0x00000000
index	0.0
conversion	0.0

Figure 3-20. Locals Window

Use this window with a **Step** or **Halt** command to display the current value of variables as you move through your program.

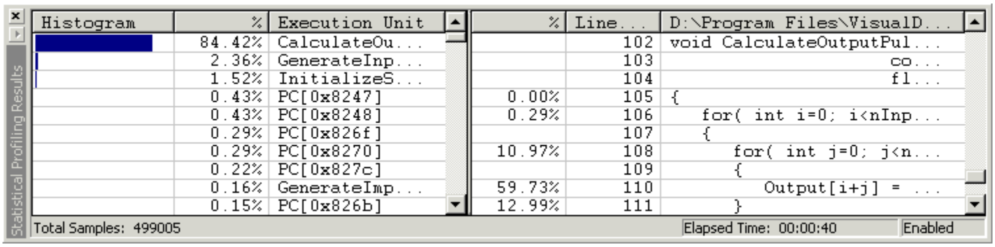
Note: Complex variables, C structures, and C++ classes appear with a plus (+) sign. Click on the plus sign to display all variable information.

The window's right-click menu provides these commonly used commands:



Statistical/Linear Profiling Results Window

Depending on the target, the title of the window can be *Statistical* Profiling Results or *Linear* Profiling Results. The window comprises two panes, as shown in Figure 3-21.



Histogram	%	Execution Unit
█	84.42%	CalculateOu...
▏	2.36%	GenerateInp...
▏	1.52%	InitializeS...
	0.43%	PC[0x8247]
	0.43%	PC[0x8248]
	0.29%	PC[0x826f]
	0.29%	PC[0x8270]
	0.22%	PC[0x827c]
	0.16%	GenerateImp...
	0.15%	PC[0x826b]

%	Line...	D:\Program Files\VisualD...
	102	void CalculateOutputPul...
	103	co...
	104	fl...
0.00%	105	{
0.29%	106	for(int i=0; i<nInp...
	107	{
10.97%	108	for(int j=0; j<n...
	109	{
59.73%	110	Output[i+j] = ...
12.99%	111	}

Total Samples: 499005

Elapsed Time: 00:00:40

Enabled

Figure 3-21. Statistical Profiling Results Window

Left Pane

The left pane shows a list of functions, assembly source lines, PCs (with no debug information) that were executed, and the percent of time each item spent on execution. The order of the items in the display is determined by the percentage of global execution time each item took to execute.

The left pane includes this information:

Column	Displays	Purpose
1	Horizontal bar chart	Graphically represents the execution percentage
2	Execution percent in text format with two decimal places (for example, 15.01%) -or- PC sample count	A right-click menu allows you to choose whether to view the execution as a percent or a sample count
3	List of functions, assembly source lines, and PCs for which there is no debug info	These items are sorted by the percentage of the global execution time each item took to execute, with the highest percentage items at the top of the list and the lowest percentage at the bottom

If you double-click on a line with a function or assembly source line in the left pane, the right pane displays the corresponding source file and jumps to the top of that function or assembly source line, respectively. If you double-click on a PC address that has no debug information, the Disassembly window jumps to that address.

Right Pane

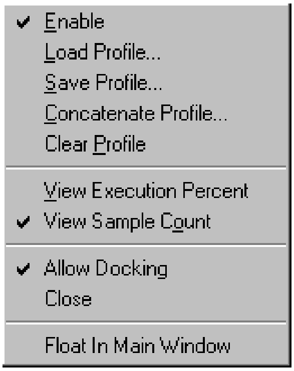
The right pane includes this information.

Column	Displays
1	Execution percent in text format with two decimal places (for example, 1.03%) -or- the PC sample count for each source line
2	Line numbers of the source file
3	Entire source file. Each source line occupies one line in the grid control.

The **status bar** at the bottom of the window shows the total number of PC samples collected and the total elapsed time, and indicates whether statistical profiling is enabled.

Right-Click Menu

The window provides the following right-click menu.



The following table describes the menu commands:

Command	Description
Enable	Enables/disables statistical profiling
Load Profile	Opens the **Select a Statistical** (or **Linear**) **Profile to Load** dialog box from which to restore statistics saved from a previous run
Save Profile	Allows you to save the current run's data to a file
Concatenate Profile	Loads and concatenates statistics saved from a previous run
Clear Profile	Clears statistics saved from a previous run
View Execution Percent	When this option is selected, each execution unit or source line in the Statistical/Linear Profiling Results window shows the execution percent, which is the PC sample count for that line divided by the total number of PC samples collected.
View Sample Count	When this option is selected, each execution unit or source line in the Statistical/Linear Profiling Results window shows the PC sample count for that line.

Call Stack Window

The Call Stack window (Figure 3-22) enables you to double-click on a stack location to move the call stack back to a previous debug context.

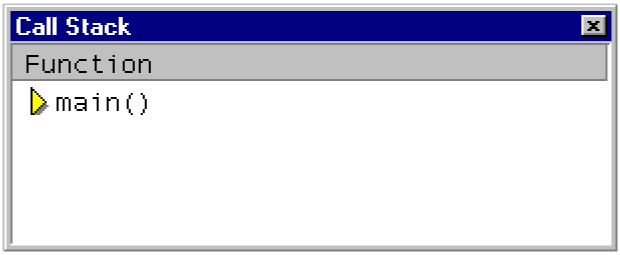


Figure 3-22. Call Stack Window

Note: This window functions with C/C++ code only.

Use this window to analyze the state of parent functions when erroneous data is being passed to the currently executing function and to see the context from which the current function is being called.

Use this debugging feature by walking up the call stack and viewing local variables in different scopes.

Memory Windows

Memory windows, similar to register windows, provide:

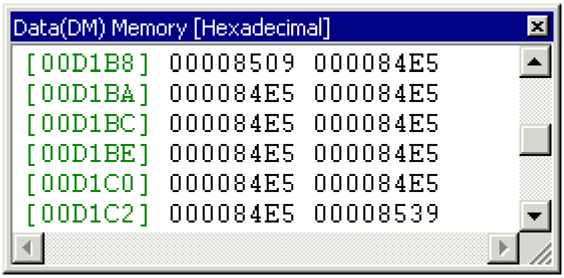
- Number format and edit features
- Fill and dump capability

From a Memory window, you can:

- Display the address of a value. Move the mouse pointer over the value, and press and hold the keyboard's **Ctrl** key.
- Lock the number of columns currently displayed, which resizes the window horizontally without altering the display
- Track expressions that the window tracks whenever it refreshes

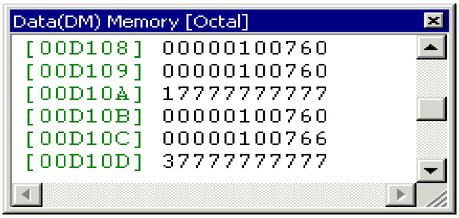
Memory Types and Number Formats

The four Memory windows that follow show examples of different number formats for the Data Memory type.



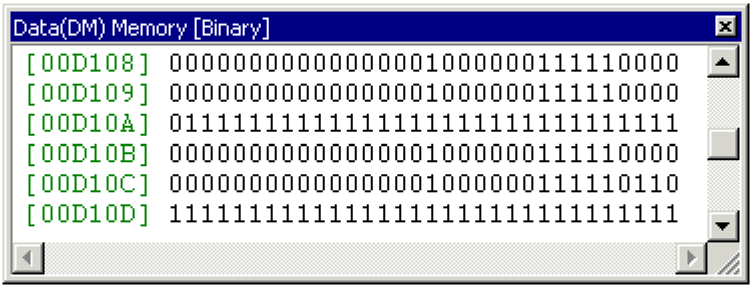
```
[00D1B8] 00008509 000084E5
[00D1BA] 000084E5 000084E5
[00D1BC] 000084E5 000084E5
[00D1BE] 000084E5 000084E5
[00D1C0] 000084E5 000084E5
[00D1C2] 000084E5 00008539
```

Figure 3-23. Data Memory Window (Hex Number Format)



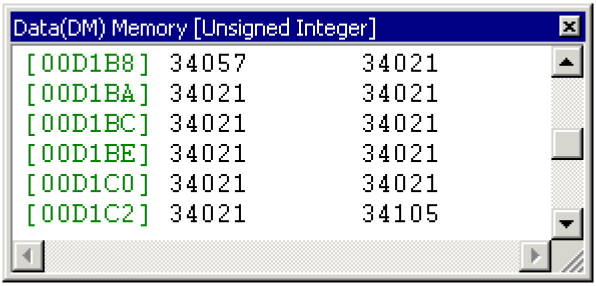
```
[00D108] 00000100760
[00D109] 00000100760
[00D10A] 17777777777
[00D10B] 00000100760
[00D10C] 00000100766
[00D10D] 37777777777
```

Figure 3-24. Data Memory Window (Octal Number Format)



```
[00D108] 00000000000000001000000111110000
[00D109] 00000000000000001000000111110000
[00D10A] 01111111111111111111111111111111
[00D10B] 00000000000000001000000111110000
[00D10C] 00000000000000001000000111110110
[00D10D] 11111111111111111111111111111111
```

Figure 3-25. Data Memory Window (Binary Number Format)

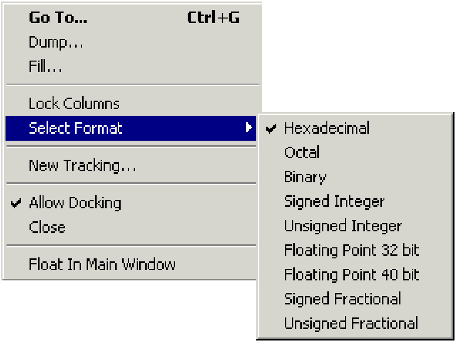


```
[00D1B8] 34057          34021
[00D1BA] 34021          34021
[00D1BC] 34021          34021
[00D1BE] 34021          34021
[00D1C0] 34021          34021
[00D1C2] 34021          34105
```

Figure 3-26. Data Memory Window (Unsigned Integer Format)

Right-Click Menu

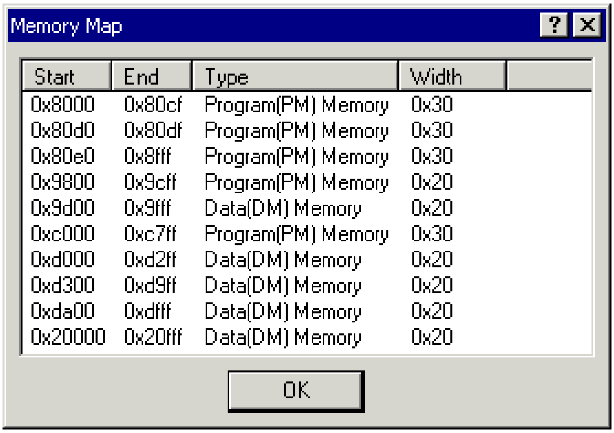
Memory windows provide the following right-click menu:



This menu enables you to run commands, such as those used to change the number format of the display.

Memory Map Windows

The Memory Map window (Figure 3-27) displays the memory map for the selected processor.



Start	End	Type	Width
0x8000	0x80cf	Program(PM) Memory	0x30
0x80d0	0x80df	Program(PM) Memory	0x30
0x80e0	0x8fff	Program(PM) Memory	0x30
0x9800	0x9cff	Program(PM) Memory	0x20
0x9d00	0x9fff	Data(DM) Memory	0x20
0xc000	0xc7ff	Program(PM) Memory	0x30
0xd000	0xd2ff	Data(DM) Memory	0x20
0xd300	0xd9ff	Data(DM) Memory	0x20
0xda00	0xdfff	Data(DM) Memory	0x20
0x20000	0x20fff	Data(DM) Memory	0x20

Figure 3-27. Memory Map Window

If no DSP program is loaded into the processor, the memory map displays all available memory in the processor.

If a program is loaded, the memory map is the map defined in the `memory` section of the program's `.LDF` file.

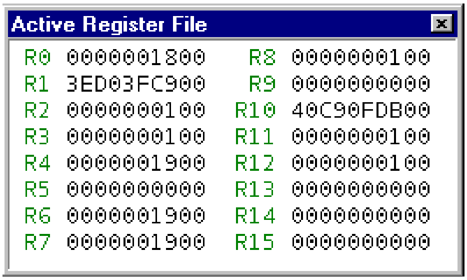
For each portion of memory, the window displays the start address, end address, and width.

Register Windows

Depending on your processor, you have access to various Register windows. A Register window enables you to:

- View and change register contents
- Change the presentation (number format)

Figure 3-28 shows an example of an active register file in a Register window.



```
R0  0000001800   R8  0000000100
R1  3ED03FC900   R9  0000000000
R2  0000000100  R10  40C90FDB00
R3  0000000100  R11  0000000100
R4  0000001900  R12  0000000100
R5  0000000000  R13  0000000000
R6  0000001900  R14  0000000000
R7  0000001900  R15  0000000000
```

Figure 3-28. Active Register File in a Register Window

Register window number formats include standard formats, such as hexadecimal, octal, and binary. Depending on the DSP, other formats are available.

You can change a register's data directly from a Register window. The modified register content is used during program execution. Edits to data do not affect your source files. To make changes permanent, edit the source file and rebuild your project.

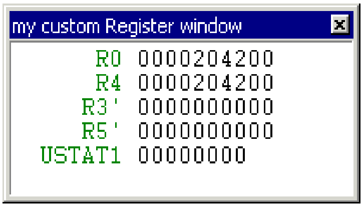
Stack Windows

Depending on your processor, you have access to various Stack windows, such as the Loop Address stack. For more information about your processor's Stack windows, consult online Help.

Custom Register Windows

While debugging, you can configure and display custom Register windows. Each custom register window has a user-specified title and displays only the registers you choose to monitor.

For example, the following custom Register window displays the contents of four registers.



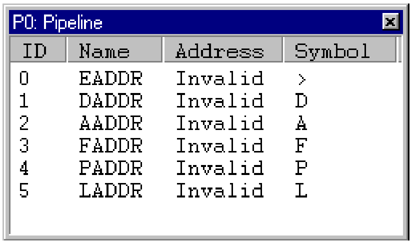
```
    R0  0000204200
    R4  0000204200
   R3'  0000000000
   R5'  0000000000
USTAT1  00000000
```

Figure 3-29. Custom Register Window – Example

A custom register window appears immediately after you create it.

Pipeline Window

The Pipeline window (Figure 3-30) displays the read-only contents of the target's instruction pipeline. Contents are updated on each processor halt.

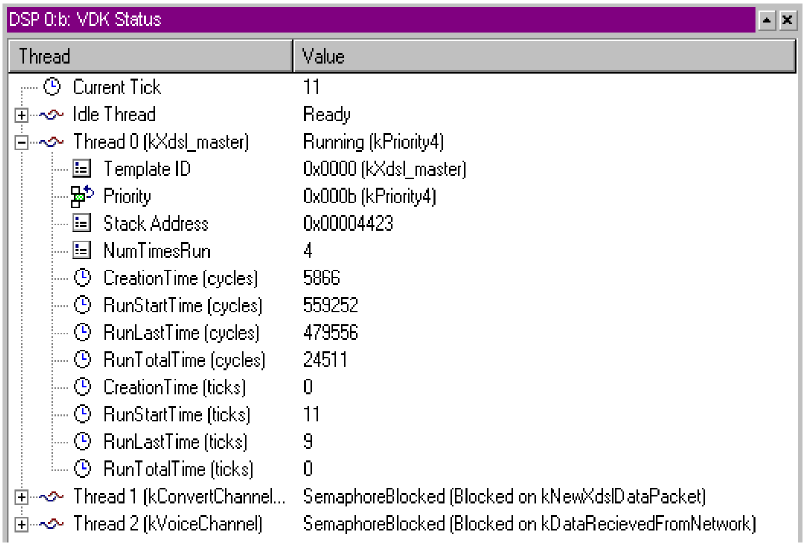


ID	Name	Address	Symbol
0	EADDR	Invalid	>
1	DADDR	Invalid	D
2	AADDR	Invalid	A
3	FADDR	Invalid	F
4	PADDR	Invalid	P
5	LADDR	Invalid	L

Figure 3-30. Pipeline Window

VDK Status Window

The VDK Status window (Figure 3-31) is available when a DSP executable is built with VDK support.



Thread	Value
Current Tick	11
Idle Thread	Ready
Thread 0 (kXdsl_master)	Running (kPriority4)
Template ID	0x0000 (kXdsl_master)
Priority	0x000b (kPriority4)
Stack Address	0x00004423
NumTimesRun	4
CreationTime (cycles)	5866
RunStartTime (cycles)	559252
RunLastTime (cycles)	479556
RunTotalTime (cycles)	24511
CreationTime (ticks)	0
RunStartTime (ticks)	11
RunLastTime (ticks)	9
RunTotalTime (ticks)	0
Thread 1 (kConvertChannel...	SemaphoreBlocked (Blocked on kNewXdslDataPacket)
Thread 2 (kVoiceChannel)	SemaphoreBlocked (Blocked on kDataRecievedFromNetwork)

Figure 3-31. VDK Status Window

When you halt execution of a VDK program, VisualDSP++ reads thread data and displays thread state and status data in this window.

When a thread is created, it is added to the display. A thread is removed from the display when it is destroyed.

Initially, thread information appears in a collapsed state in which only the thread name and its current state are displayed. When a thread is in the Ready state, its priority is displayed.

Clicking the plus sign ⊞ next to the thread name expands the view. When thread status information is not accessible (thread out of context), no information is displayed.

These are the possible thread states:

- Running
- Ready
- SemaphoreBlocked
- EventBlocked
- DeviceFlagBlocked
- SemaphoreBlockedWithTimeout
- EventBlockedWithTimeout
- DeviceFlagBlockedWithTimeout
- Sleeping
- Unknown

Refer to the *VisualDSP++ Kernel (VDK) User's Guide* for details.

State History Window

VDK state history and VDK target load plots are available when a DSP executable is built with VDK support. During execution of a VDK-enabled program, thread and event data is collected in a history buffer. The State History window, described in Figure 3-32, displays the history buffer plots.

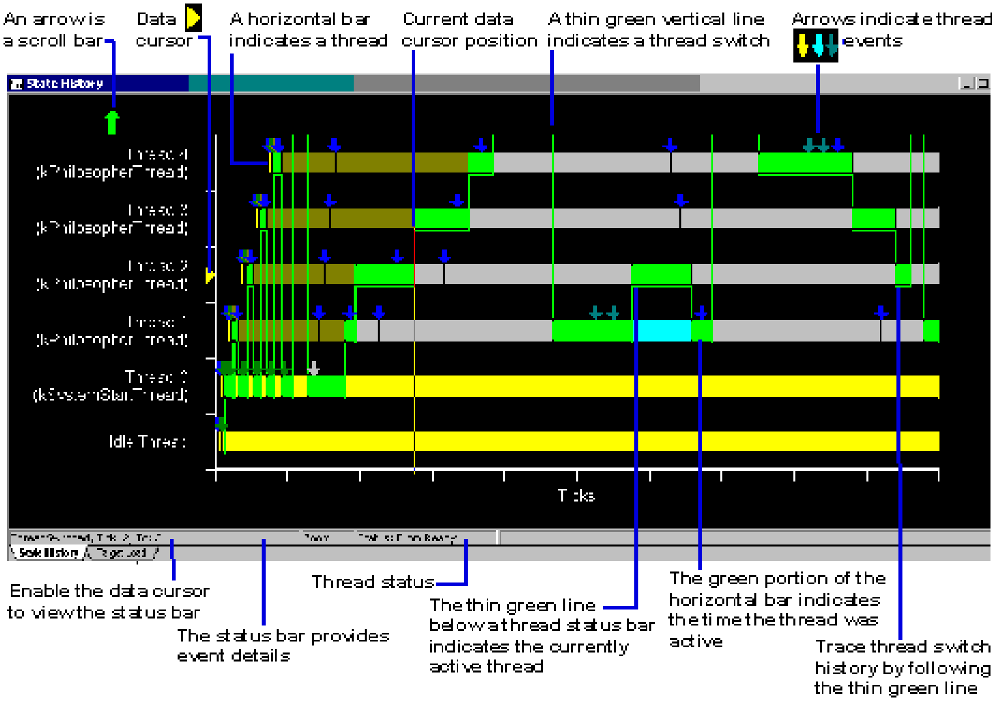

An arrow is a scroll bar
Data cursor
A horizontal bar indicates a thread
Current data cursor position
A thin green vertical line indicates a thread switch
Arrows indicate thread events
State History
Thread 4 (kPhilosopherThread)
Thread 3 (kPhilosopherThread)
Thread 2 (kPhilosopherThread)
Thread 1 (kPhilosopherThread)
Thread 0 (kSystemStartThread)
Idle Thread
Ticks
State History
Target Load
Enable the data cursor to view the status bar
The status bar provides event details
Thread status
The thin green line below a thread status bar indicates the currently active thread
The green portion of the horizontal bar indicates the time the thread was active
Trace thread switch history by following the thin green line

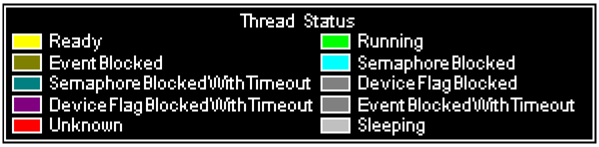
Figure 3-32. State History Window

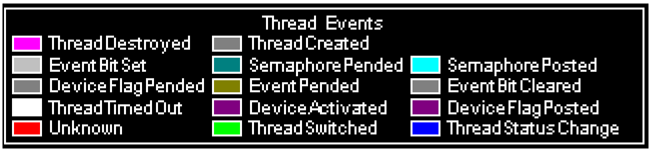
Thread status appears as horizontal bars, and thread events appear as arrows above the horizontal bars. Status bars and event symbols are color-coded, based on thread status and event type.

Thread Status and Event Colors

All events of the same type are drawn in the same color. The thin green line below the thread status bar indicates the currently active thread. When a thread is switched, the green line is drawn vertically to the next active thread. Trace the thread switch history by following the thin green line.

Right-click on the plot and choose **Status Legend** or **Event Legend** to display the legends (shown below) in the State History window.





Window Operations

The state history status bar (bottom of plot) shows the event's details and thread status. Event details include the event type, the tick when the event occurred, and an event value.

The value for a thread-switched event indicates the thread being switched in or out.

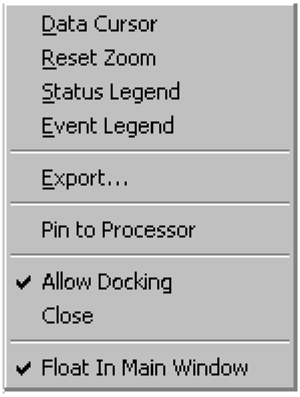
Right-click on the plot and choose **Data Cursor** to activate the data cursor, which is used to display event and thread status details. Based on the event that occurred, the thread status changes. Press the keyboard's right arrow key or left arrow key to move to the next or previous event. When the data cursor hits a thread switch event, it moves to the thread being switch in.

The yellow triangle to the right of the thread name indicates the currently active thread.

You can zoom in on a region to examine that area in more detail. Hold the left mouse button down while dragging the mouse to create a selection box. Then release the mouse button to expand the plot. To restore the plot to its original scale, right-click on the plot and choose **Reset Zoom**.

Actions You can Perform from the Plot

The State History window's right-click menu, shown below, provides easy access to operations that you can perform on the state history plot.



Target Load Window

A target load plot (Target Load window), shown in Figure 3-33, displays the percent of time the target spent in the idle thread.

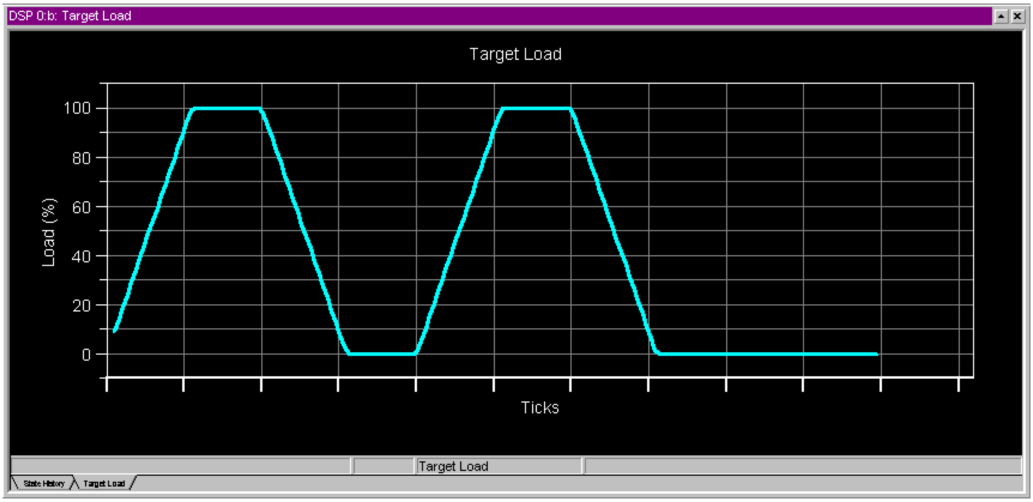


Figure 3-33. Target Load Window

A load of 0% means VDK spent all of its time in the idle thread. A load of 100% means the target did not spend any time in the idle thread.

Load data is processed using a moving window average.

About Debugging Windows

Editor Window Features

An Editor window provides these features:

- Status icons
- Expression evaluation
- Two view formats (source mode or mixed mode)

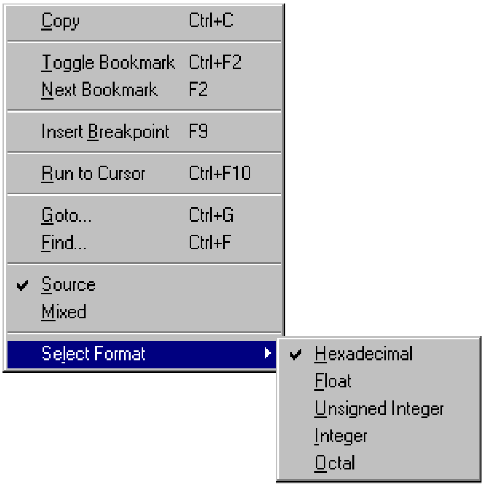
Syntax Coloring

Specify colors to help you locate information in these types of files:

- `.ASM` (assembly)
- `.C` (C files)
- `.CPP` (C++)
- `.H` (header)
- `.LDF` (Linker Description Files)
- `.Tcl` (Tool Command Language script files)

Right-Click Menu

The Editor window's right-click menu provides the following commands:



Notes:

- The available number formats under **Select Format** are DSP-dependent.
- An additional command, **Source Tcl Script**, is available when you are editing a Tcl script.

Editor Window Symbols

The Editor window's gutter (left margin) displays icons that indicate breakpoints, bookmarks, and the current position of the program counter (PC).

Symbol	Indicates
⇨	The current source line to be executed (PC location)
●	An enabled breakpoint
○	A disabled breakpoint
▭	A bookmark

Bookmarks

Bookmarks are pointers in Editor windows. You bookmark a location so you can go it quickly later.

Context-Sensitive Expression Evaluation

You can evaluate an expression in an Editor window only if your `.DXE` program is loaded for debugging.

As you move the mouse pointer over a variable, with the pointer still on top of the variable, VisualDSP++ evaluates the variable. If the variable is in scope, the value appears in a tool tip window.

Viewing an Expression

You can view an expression in different ways.

When the Editor window is in mixed-mode, you can view an expression by moving the pointer over a register in an assembly instruction. The register contents are displayed in a tool tip.

Highlighting an Expression

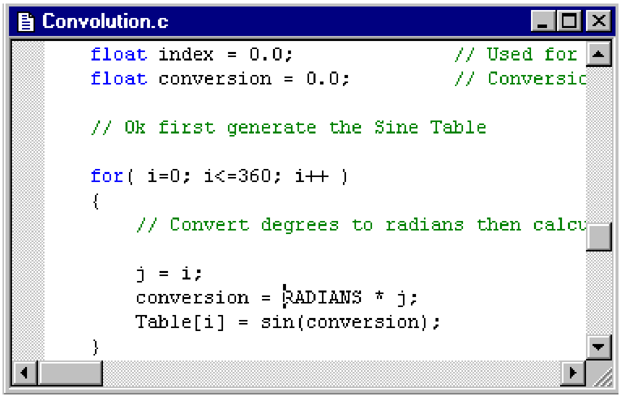
You can highlight an expression in the Editor window and then move the pointer on top of the highlighted expression to display its value in a tool tip.

Source Mode vs. Mixed Mode

You can specify an Editor window's display format. Your two options are: source mode and mixed mode.

Source Mode

Source mode, shown in Figure 3-34, displays C code only.



```
float index = 0.0;              // Used for
float conversion = 0.0;         // Conversic

// Ok first generate the Sine Table

for( i=0; i<=360; i++ )
{
    // Convert degrees to radians then calcu

    j = i;
    conversion = RADIANS * j;
    Table[i] = sin(conversion);
}
```

Figure 3-34. Editor Window – Source Mode Format

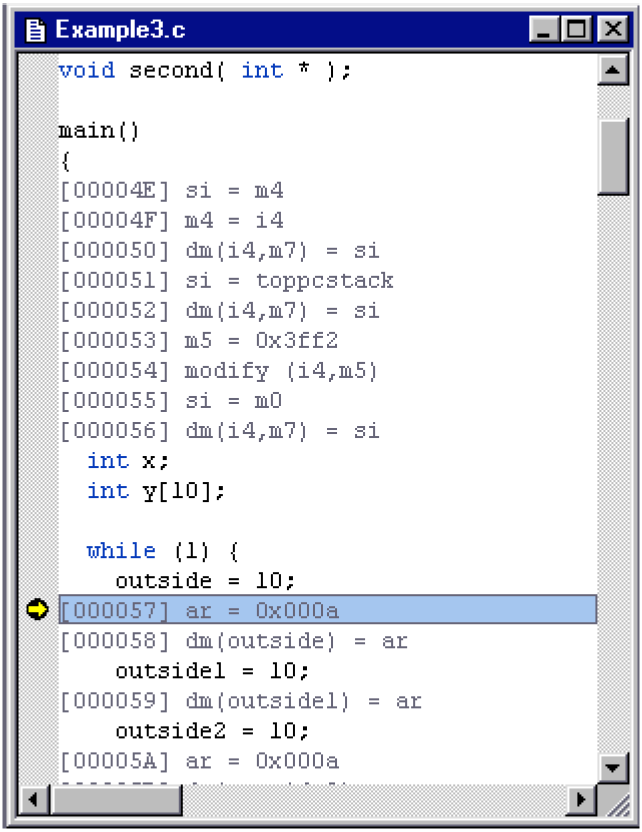
Mixed Mode

Mixed mode displays the assembled code after the line of the corresponding C code. The assembly code takes a specified color.

Note:

- You must compile the source file with debugging information to view the source file in mixed mode.
- You can enable and disable the display of pipeline symbols while in mixed mode.

Figure 3-35 shows an example of the mixed mode format.



```
void second( int * );

main()
{
[00004E] si = m4
[00004F] m4 = i4
[000050] dm(i4,m7) = si
[000051] si = toppcstack
[000052] dm(i4,m7) = si
[000053] m5 = 0x3ff2
[000054] modify (i4,m5)
[000055] si = m0
[000056] dm(i4,m7) = si
  int x;
  int y[10];

  while (1) {
    outside = 10;
[000057] ar = 0x000a
[000058] dm(outside) = ar
    outside1 = 10;
[000059] dm(outside1) = ar
    outside2 = 10;
[00005A] ar = 0x000a
```

Figure 3-35. Editor Window – Mixed Mode Example

Expressions in an Expressions Window

You can enter these types of expressions in an Expressions window:

Expression	Description
Memory address	The memory addresses can be data (DM) or program (PM). The size of an address is 32 bits. Precede memory identifiers with a `$` sign; for example, `$dm16(0x2000)`.
Register expression	Precede register names with a $ sign (for example, `$r0, $ax0, or $imask`).
C/C++ statements	Use standard C/C++ arithmetic and logical operators.

About Expressions

The Expressions window displays the current value of each expression as you step through your program. Expressions are evaluated based on the current debug context.

For example, if you enter expression "`a`" and a global variable "`a`" exists, you see its value. If you then step into a function that has local variable "`a`", you see the local value until the debug context leaves the function. When a variable goes out of context, a string displays next to the variable to inform you that the variable is out of context.

The expressions described above are C expressions. The current syntax also allows you to use registers in expressions. For example, the following is a valid expression:

```
$R0 + $I0
```

Register expressions and C expressions can be mixed in an expression.

Register expressions follow these rules:

- Precede register names with a `$` character.
- Register names can be uppercase or lowercase characters.
- Registers have no context. A register expression always evaluates to the current value of the register.

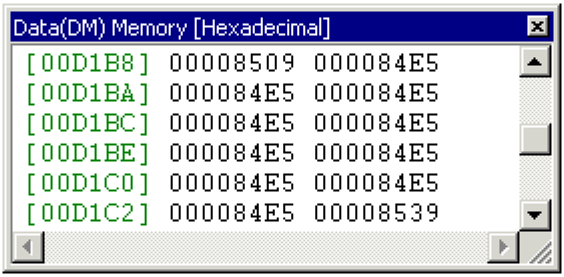
Number Formats

You can select the number format used to display a particular register window or memory window. The available number formats, which depend on your DSP family, can include the following:

- Hexadecimal
- Octal
- Binary
- Signed integer
- Unsigned integer
- Floating Point 32 bit
- Floating Point 40 bit
- Signed fractional
- Unsigned fractional

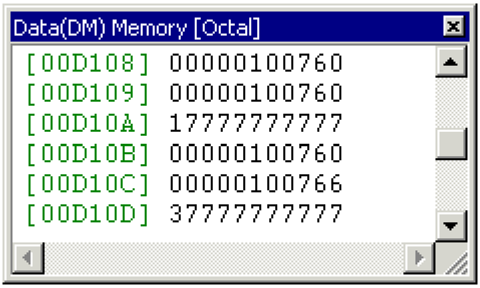
The following windows are examples of three different number formats. The actual appearance depends on your DSP family.

This window appears in hexadecimal format:



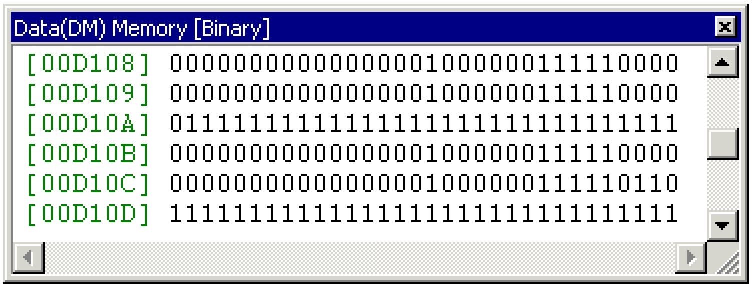
```
[00D1B8] 00008509 000084E5
[00D1BA] 000084E5 000084E5
[00D1BC] 000084E5 000084E5
[00D1BE] 000084E5 000084E5
[00D1C0] 000084E5 000084E5
[00D1C2] 000084E5 00008539
```

This window appears in octal format:



```
[00D108] 00000100760
[00D109] 00000100760
[00D10A] 17777777777
[00D10B] 00000100760
[00D10C] 00000100766
[00D10D] 37777777777
```

This window appears in binary format:



```
[00D108] 00000000000000001000000111110000
[00D109] 00000000000000001000000111110000
[00D10A] 01111111111111111111111111111111
[00D10B] 00000000000000001000000111110000
[00D10C] 00000000000000001000000111110110
[00D10D] 11111111111111111111111111111111
```

Plot Windows

Use a Plot window to display a **plot**, which is a visualization of values obtained from DSP memory. You can display one or multiple Plot windows.

Figure 3-36 shows an example of a plot in a Plot window.

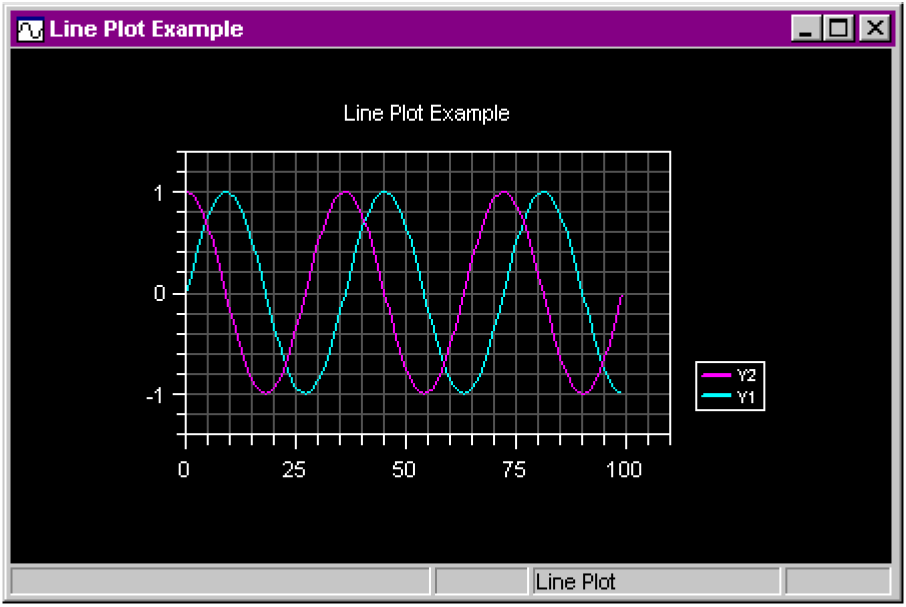


Figure 3-36. Plot Window – Plot Example

You specify the contents and presentation of the plot. You can modify a plot's configuration and immediately view the revised plot.

From a Plot window, you can zoom in on a potion of a plot or view the values of a data point.

You can print a plot, save the plot image to a file, or save the plot's data to a file. For details, refer to the online Help in VisualDSP++.

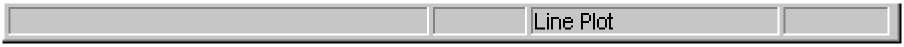
Plot Window Features

Plot windows include a status bar and a right-click menu.

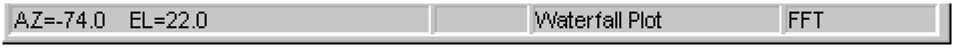
Status Bar

The status bar, located at the bottom of the Plot window, displays the plot type and other information, depending on the plot type and other settings.

The following examples show different plot information displayed on the status bar.



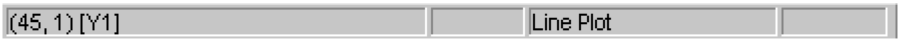
Line Plot



AZ=-74.0 EL=22.0

Waterfall Plot

FFT



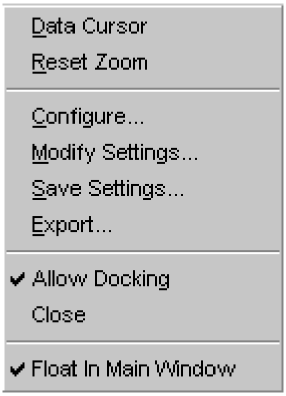
(45, 1) [Y1]

Line Plot

In a waterfall plot, the status bar indicates the azimuth and elevation viewing angles. If you zoom in on a region, the status bar indicates that zoom is enabled. When you use the data cursor, the status bar shows the selected point's data value.

Right-Click Menu

The Plot window's right-click menu is shown below:



This menu provides access to the standard window options (docking, closing, and floating in the main window) and to the Plot window features described in the following table.

Feature	Description
Data cursor position	You can move the Plot window's data cursor over a data point and view the point's memory data value in the left side of the Plot window's status bar. Use the keyboard's arrow keys to move around on the graph.
Zooming	You can zoom in to view a specified region of the plot. You can also reset the Plot window to its initial full-scale display.
Plot configuration	From the Plot window, you can access the **Plot Configuration** dialog box, from which to add, remove, or modify data sets. You can also change the plot type and rename the plot.
Settings modification	You can customize the plot's appearance. You can specify settings for the plot (grids, colors, margins, fonts, axes, and so on), and you can specify settings for each data set (data processing).
Settings storage and retrieval	You can save plot configuration settings for future use. Plot settings are stored, but the data is not stored. You can retrieve settings (.VPS file) and load new plot data.
Export	You can export the plot image to various destinations, including the Windows clipboard. Save the plot image as a file (jpg, gif, tif, eps, txt, or dat format) or print a hard copy.

Plot Configuration

A plot configuration comprises two parts:

- Data values
- Presentation (configuration) settings

You create data sets and configure the data for each data set. A *data set* is a series of data values in DSP memory. You specify the memory location, the number of values, and other options that identify the data. 3-D plots require additional specifications for row and column counts.

VisualDSP++ offers many plot presentation options. You choose the type of plot (for example, waterfall) and the axis associated with each data set. You configure options for titles, grids, fonts, colors, and axis presentation.

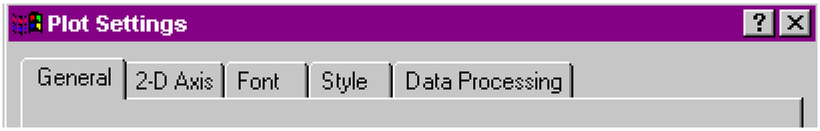
You can recall a plot from saved settings. You must identify the settings (`.VPS` file) to be used. VisualDSP++ uses these settings and reads DSP memory to generate and display a plot in a Plot window.

Plot Window Presentation

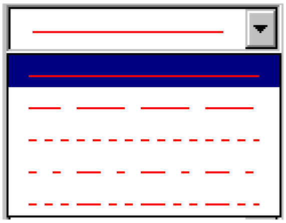
You can customize the presentation of a Plot window to fit your needs. You configure presentation settings from the **Plot Setting** dialog box, which you invoke as follows:

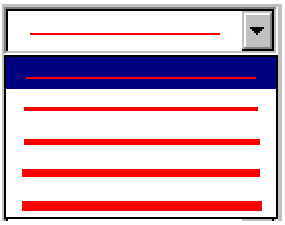
- Right-click from within a Plot window
- Click the **Settings** button from in **Plot Configuration** dialog box

The **Plot Settings** dialog box provides the following tabs:



Options on the tab pages enable you to configure the Plot window's presentation. On the **Style** tab page, for example, you can easily specify symbols for a data set as well as line type and width, as shown below.





In addition to the many presentation options, you can select a rectangular area, as shown in Figure 3-37, and zoom in on it.

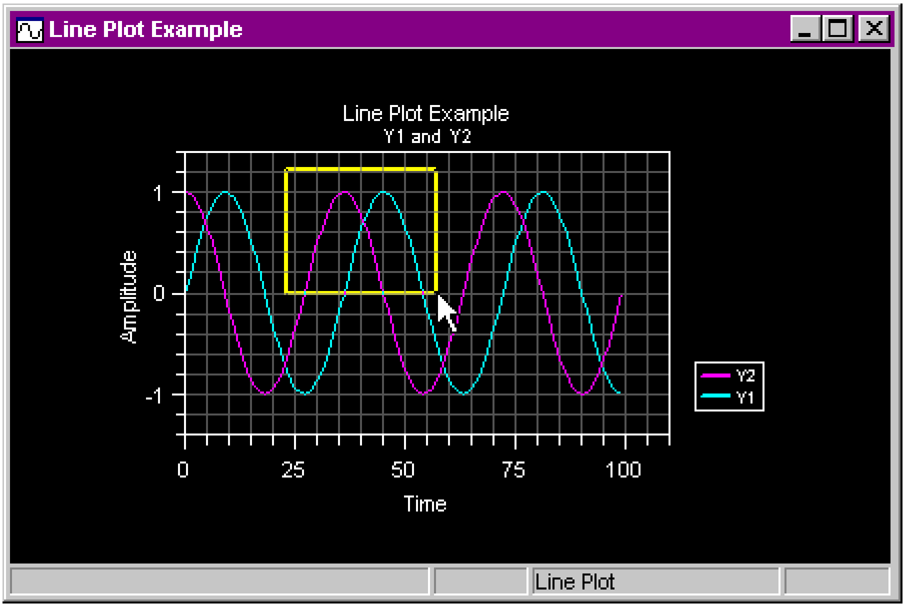


Figure 3-37. Plot Window – Selecting an Area to Zoom in on

Plot Presentation Options

You can configure a plot's presentation. Depending on the type of plot, many options are available.

In the **Plot Settings** dialog box, these options are grouped by function on tab pages, as follows.

Tab Page	Options you can specify
General	Title and subtitle, grid lines, margins, background colors, and legend
2-D Axis	For X-axis and Y-axis: axis titles, start and increment values, scales
3-D Axis	For X-axis, Y-axis, and Z-axis: axis titles, Z-axis settings, step sizes, scale multipliers, color and mesh
Font	Font name, color, and size
Style	For a data set: line type, width, color; symbol and type
Data Processing	For a data set: data processing algorithm, sample rate, number of stored traces, and triggering

You can specify a plot's presentation options before you generate the plot (while configuring the plot), or you can specify plot options after generating the plot.