2 TUTORIAL

This chapter contains the following topics.

Overview

This tutorial demonstrates some of the key features and capabilities of the VisualDSP++ Integrated Development and Debugging Environment (IDDE). The exercises use sample programs written in C, C++, and assembly for ADSP-21xxx DSPs. For these exercises, you will use the ADSP-2106*x* simulator for the ADSP-21065L target.

You can use a different ADSP-21xxx processor with only minor changes to the Linker Description File (`.LDF`) included with each project.

VisualDSP++ includes basic Linker Description Files for each processor type in the `ldf` folder. The default installation path for this folder is:

```
Analog Devices\VisualDSP\21k\ldf
```

The source files for these exercises are installed during the VisualDSP++ software installation.

The tutorial contains five exercises.

- In **Exercise One**, you will start up VisualDSP++, build a project containing C source code, set up a debug session, and run the program.
- In **Exercise Two**, you will create a new project, use Expert Linker to create a Linker Description File for the project, modify sources to call an assembly routine, use Expert Linker to modify the `.LDF` file, and rebuild the project.
- In **Exercise Three**, you will apply a simple convolution algorithm to a buffer of data. You will use the VisualDSP++ plotting engine to view the different data arrays graphically.
- In **Exercise Four**, you will use linear profiling to examine the efficiency of the convolution algorithm used in Exercise Three. Using the collected linear profile data, you will pinpoint the most time-consuming areas of the algorithm, which are likely to require hand tuning in the assembly language.
- In **Exercise Five**, you will install a VCSE component on your system and add the component to the project. Then you will build and run the program with the component.

Tip: Become familiar with the VisualDSP++ toolbar buttons, shown in Figure 2-1. They are shortcuts for menu commands such as **File**, **Open**. Toolbar buttons and menu commands that are not available for the task that you are performing are disabled and displayed in gray.

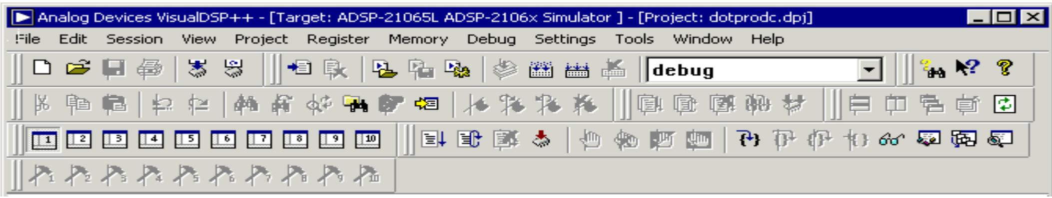


Figure 2-1. VisualDSP++ Toolbar Buttons

Exercise One: Building and Running a C Program

In this exercise, you will:

- Start up the VisualDSP++ environment
- Open and build an existing project
- Set up the debug session and examine windows and dialog boxes
- Run the program

The sources for this exercise are in the `dot_product_c` folder. The default installation path is:

```
Program Files\Analog Devices\VisualDSP\21k\Examples\tutorial\
dot_product_c
```

Step 1: Start VisualDSP++ and Open a Project

To start VisualDSP++ and open a project:

1. Click the Windows **Start** button and select **Programs**, **VisualDSP**, and **VisualDSP++ Environment**.

   If you are running VisualDSP++ for the first time, the **New Session** dialog box (Figure 2-6 on page 2-11) opens to enable you to set up a session.

   a. Select the values shown in Table 2-1.

Table 2-1. Session Specification

Box	Value
Debug Target	ADSP-2106x Family Simulator
Platform	ADSP-2106x Simulator
Session Name	ADSP-21065L ADSP-2106x Simulator
Processor	ADSP-21065L

   b. Click **OK**. The VisualDSP++ main window appears.

   If you have already run VisualDSP++ and the **Reload last project at startup** option is selected on the **Project** page under **Settings** and **Preferences**, VisualDSP++ opens the last project that you worked on. To close this project, choose **Close** from the **Project** menu, and then click **No** when prompted to save the project. Since you have made no changes to the project, you do not have to save it.

2. From the **Project** menu, choose **Open**.

   VisualDSP++ displays the **Open Project** dialog box.

3. In the **Look in** box, open the `Program Files\Analog Devices` folder and double-click the following subfolders in succession.

   `VisualDSP\21k\Examples\tutorial\dot_product_c`

This path is based on the default installation.

4. Double-click the `dotprodc` project (`.dpj`) file.

   VisualDSP++ loads the project in the **Project** window, as shown in Figure 2-2.

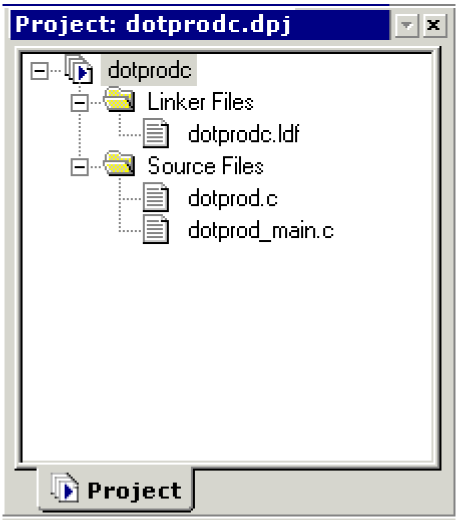


Figure 2-2. Project Loaded in the Project Window

The environment displays messages in the **Output** window as it processes the project settings and file dependencies.

The `dotprodc` project comprises two C language source files, `dotprod.c` and `dotprod_main.c`, which define the arrays and calculate their dot products.

5. From the **Settings** menu, choose **Preferences** to open the **Preferences** dialog box, shown in Figure 2-3.

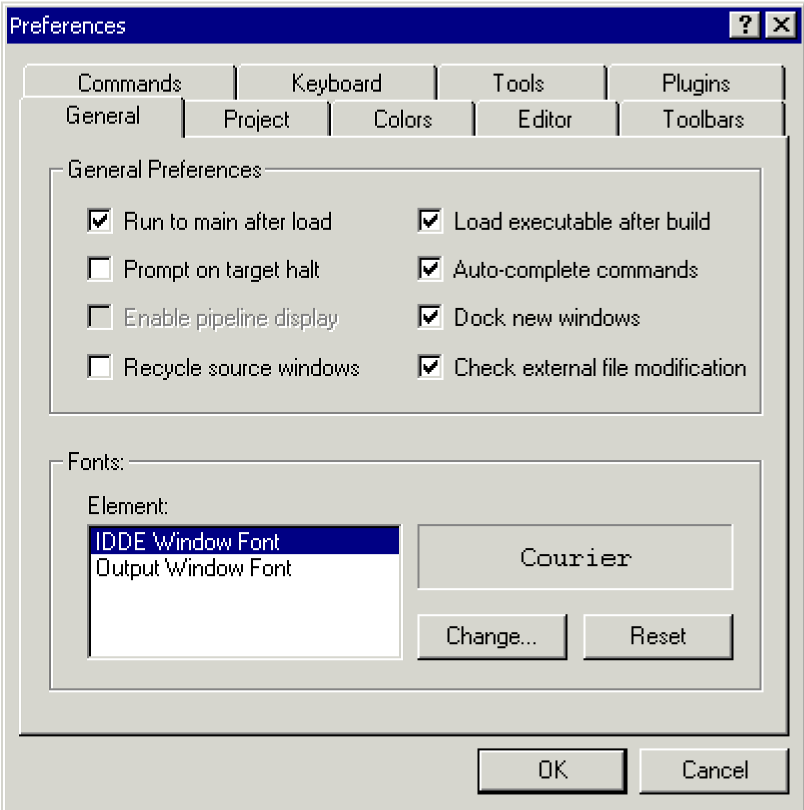


Figure 2-3. Preferences Dialog Box

6. On the **General** page, under **General Preferences**, make sure that the following options are selected.

   - **Run to main after load**
   - **Load executable after build**

7. Click **OK** to close the **Preferences** dialog box.

   You are now ready to build the project.

Step 2: Build the dotprodc Project

To build the `dotprodc` project:

1. From the **Project** menu, choose **Build Project**.

   VisualDSP++ first checks and updates the project dependencies and then builds the project by using the project source files.

   As the build progresses, the **Output** window displays status messages (error and informational) from the tools. For example, when a tool detects invalid syntax or a missing reference, the tool reports the error in the **Output** window.

   If you double-click the file name in the error message, VisualDSP++ opens the source file in an editor window. You can then edit the source to correct the error, rebuild, and launch the debug session. If the project build is up-to-date (the files, dependencies, and options have not changed since the last project build), no build is performed unless you run the **Rebuild All** command. Instead, you see the message “`Project is up to date`.” If the build has no errors, a message reports “`Build completed successfully`.”

In this example (Figure 2-4) notice that the compiler detects an undefined identifier and issues the following error message in the **Output** window.

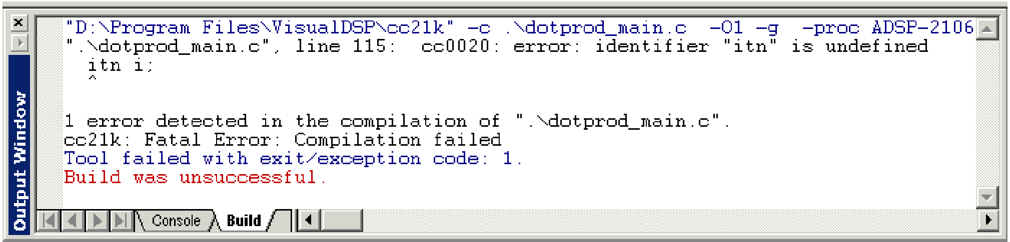


Figure 2-4. Example of Error Message

2. Double-click the error message (black) text in the **Output** window.

   VisualDSP++ opens the C source file `dotprod_main.c` in an editor window and places the cursor on the line that contains the error (see Figure 2-5 on page 2-9).

The editor window in Figure 2-5 shows that the integer variable declaration `int` has been misspelled as `itn`.

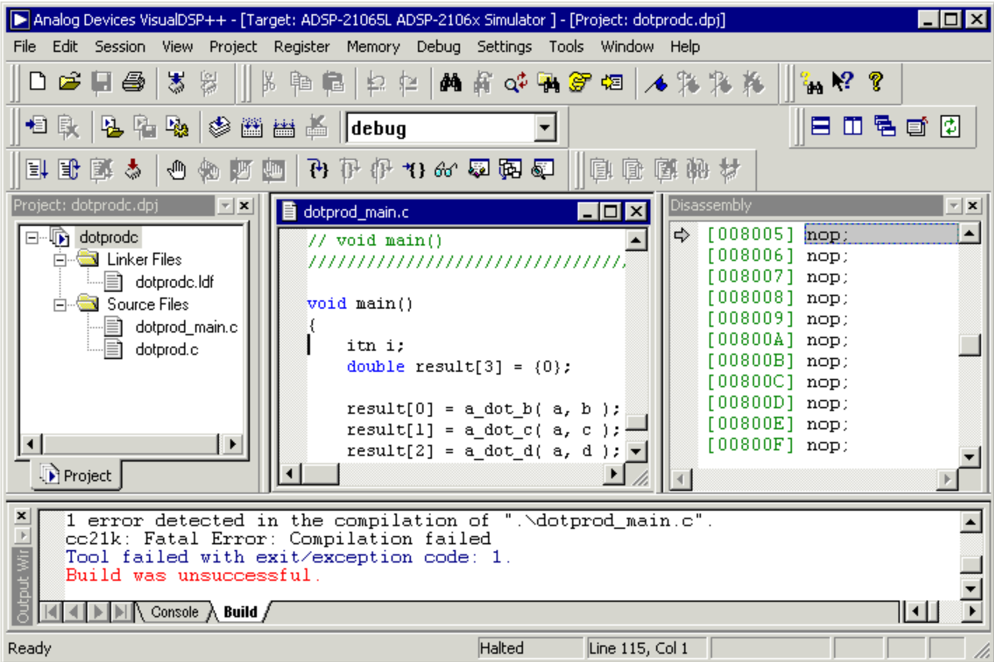


Figure 2-5. Output Window and Editor Window

3. In the editor window, click on `itn` and change it to `int`. Notice that `int` is now color coded to signify that it is a valid C keyword.

4. Save the source file by choosing **Save** from the **File** menu.

5. Build the project again by choosing **Build Project** from the **Project** menu. The project is now built without any errors, as reported in the **Build** view in the **Output** window.

Now that you have built your project successfully, you can run the example program.

Step 3: Set Up the Debug Session

In this procedure, you will:

- Set up the debug session before running the program
- View debugger windows and dialog boxes

Since you enabled **Load executable after build** on the **General** page in the **Preferences** dialog box, the executable file `dotprodc.dxe` is automatically downloaded to the target.

If the selected processor in the debug session does not match the project's build target, VisualDSP++ reports this discrepancy and asks if you want to select another session before downloading the executable to the target. If VisualDSP++ does not open the **Session List** dialog box, skip steps 1–4.

To set up the debug session:

1. **In the Session List** dialog box, click **New Session** to open the **New Session** dialog box, shown in Figure 2-6 on page 2-11.

   For subsequent debugging sessions, use the **New Session** command on the **Sessions** menu to open the **New Session dialog box**.

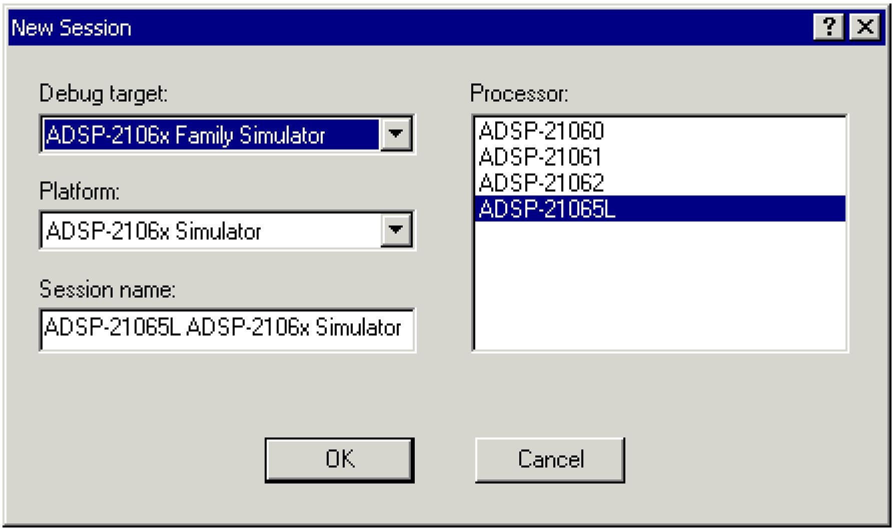


Figure 2-6. New Session Dialog Box

2. Specify the target and processor information listed in Table 2-2.

Table 2-2. Session Specification

Box	**Value**
Debug Target	ADSP-2106x Family Simulator
Platform	ADSP-2106x Simulator
Session Name	ADSP-21065L ADSP-2106x Simulator
Processor	ADSP-21065L

3. Click **OK** to close the **New Session** dialog box and return to the **Session List** dialog box.

4. With the new session name highlighted, click **Activate**.

If you do not click **Activate**, the session mismatch message appears again.

VisualDSP++ closes the **Session List** dialog box, automatically loads your project's executable file (`dotprodc.dxe`), and advances to the main function of your code (see Figure 2-7).

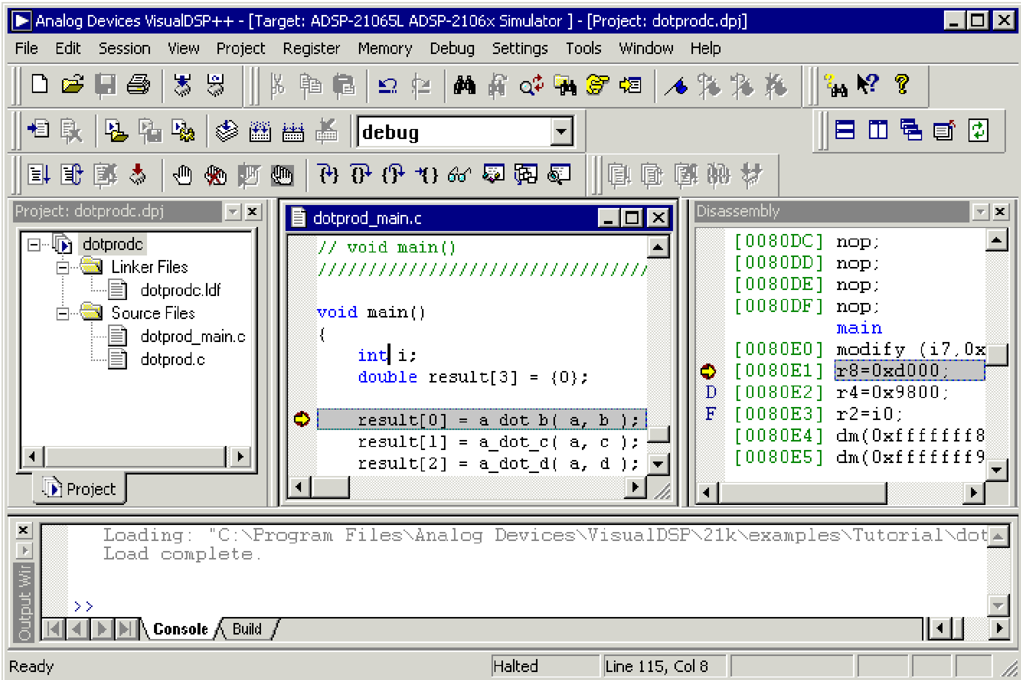


Figure 2-7. Loading dotprodc.dxe

5. Look at the information in the open windows.

   The **Output** window's **Console** view contains messages about the status of the debug session. In this case, VisualDSP++ reports that the `dotprodc.dxe` load is complete.

   The **Disassembly** window displays the machine code for the executable. Use the scroll bars to move around the **Disassembly** window.

   Note that a solid red circle and a yellow arrow appear at the start of the program labeled "`main`".

   The solid red circle indicates that a breakpoint is set on that instruction, and the yellow arrow indicates that the processor is currently halted at that instruction. When VisualDSP++ loads your C program, it automatically sets two breakpoints, one at the beginning and one at the end of code execution.

6. From the **Settings** menu, choose **Breakpoints** to view the breakpoints set in your program. VisualDSP++ displays the **Breakpoints** dialog box, shown in Figure 2-8.

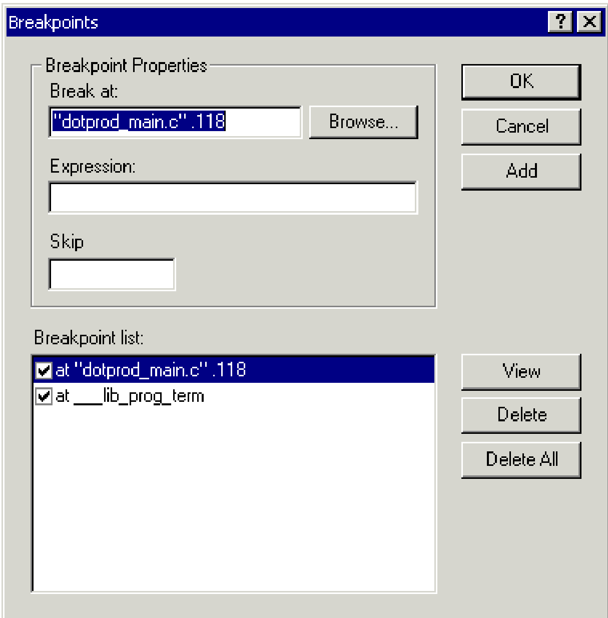


Figure 2-8. Breakpoints Dialog Box

The breakpoints are set at these C program labels:

- `“dotprod_main.c” 118`
- `__lib_prog_term`

The **Breakpoints** dialog box enables you to view, add, and delete breakpoints and to browse for symbols. In the **Disassembly** and editor windows, double-clicking on a line of code toggles (adds or deletes) breakpoints. In the editor window, however, you must place the cursor in the gutter before double-clicking.

Use these tool buttons to set or clear breakpoints:

 Toggles a breakpoint for the current line

 Clears all breakpoints

7. Click **OK** or **Cancel** to exit the **Breakpoints** dialog box.

Step 4: Run dotprodc

To run `dotprodc`, click the **Run** button  or choose **Run** from the **Debug** menu.

VisualDSP++ computes the dot products and displays the following results in the **Console** view (Figure 2-9) in the **Output** window.

```
Dot product [0] = 0.000000
Dot product [1] = 0.707107
Dot product [2] = -0.500000
```

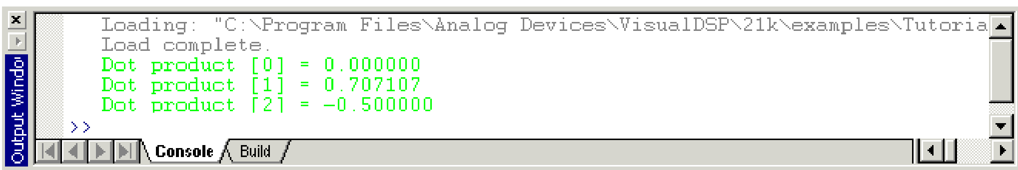


Figure 2-9. Results of the dotprodc Program

You are now ready to begin Exercise Two.

Exercise Two: Calling an Assembly Routine and Creating an LDF

In Exercise One, you built and ran a C program. In this exercise, you will modify this program to call an assembly language routine, create a Linker Description File to link with the assembly routine, and rebuild the project. The project files are largely identical to those of Exercise One. Minor modifications illustrate the changes needed to call an assembly language routine from C source code.

Step 1: Create a New Project

To create a new project:

1. From the **Project** menu, choose **Close** to close the dotprodc project. Click **Yes** when prompted to close all open editor windows. If you have modified your project during this session, you are prompted to save the project. Click **No**.

2. From the **Project** menu, choose **New** to open the **Save New Project As** dialog box, shown in Figure 2-10.

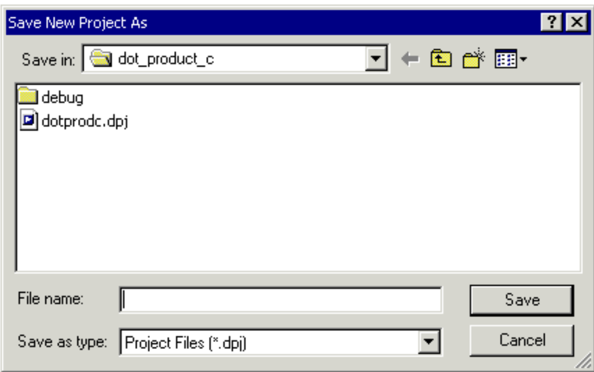


Figure 2-10. Save New Project As Dialog Box

3. Click the up-one-level button  until you locate the `dot_product_asm` folder, and then double-click this folder.

4. In the **File name** box, type `dot_product_asm`, and click **Save**.

   The **Project Options** dialog box (Figure 2-11) appears.

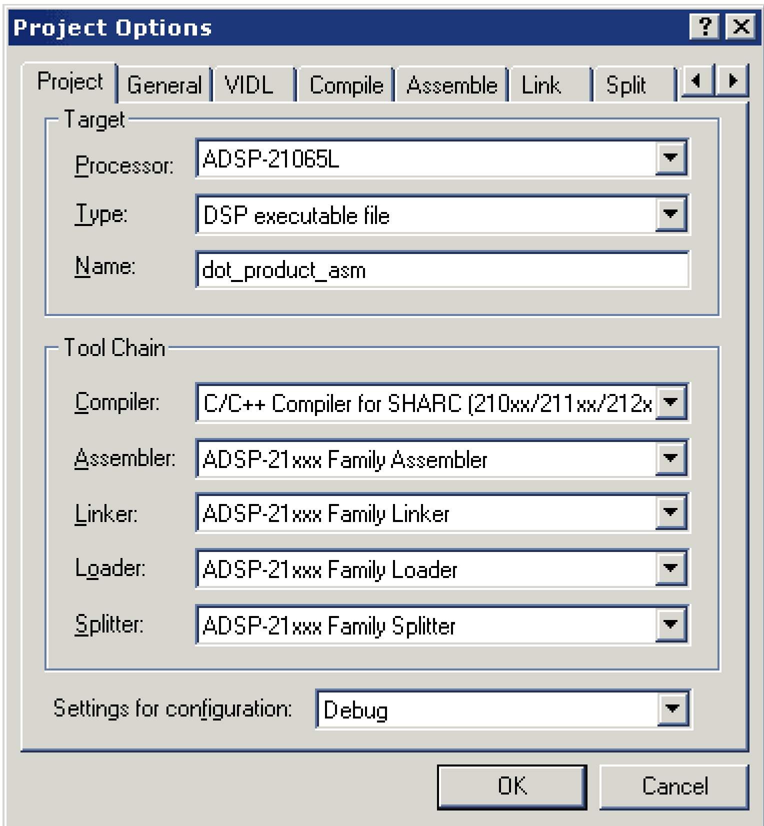


Figure 2-11. Project Options Dialog Box: Project Page

This dialog box enables you to specify project build information.

5. Take a moment to examine the tabbed pages in the **Project Options** window: **Project**, **General**, **VIDL**, **Compile**, **Assemble**, **Link**, **Split**, **Load**, and **Post Build**. On each page, you specify the tool options used to build the project.

6. On the **Project** page (Figure 2-11 on page 2-17), specify the following values.

Table 2-3. Completing the Project Page

Box	Value
Processor	ADSP-21065L
Type	DSP executable file
Name	`dot_product_asm`
Settings for configuration	Debug

These settings specify information for building an executable file for the ADSP-21065L DSP. The executable contains debug information, so you can examine program execution.

7. Click the **Compile** tab to display the **Compile** page, shown in Figure 2-12 on page 2-19.

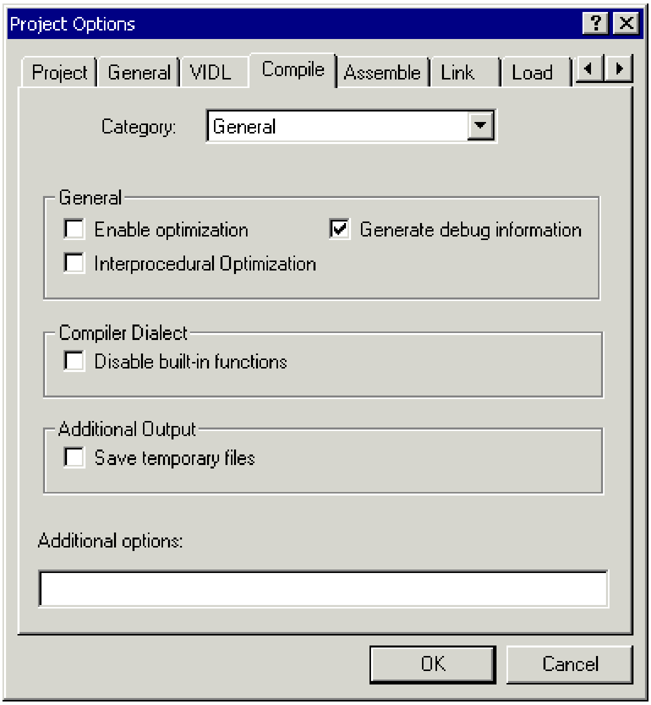


Figure 2-12. Project Options Dialog Box: Compile Page

8. In the **General** group box, select the **Generate debug information** check box, if it is not already selected, to enable debug information for the C source.

9. Click **OK** to apply changes to the project options and to close the **Project Options** dialog box.

When prompted to add support for the VisualDSP++ kernel, click **No**. Once added, kernel support cannot be removed.

You are now ready to add the source files to the project.

Step 2: Add Source Files to dot_product_asm

To add the source files to the new project:

1. Click the **Add File** button , or from the **Project** menu, choose **Add to Project** and then choose **File(s)**.

   The **Add Files** dialog box (Figure 2-13) appears.

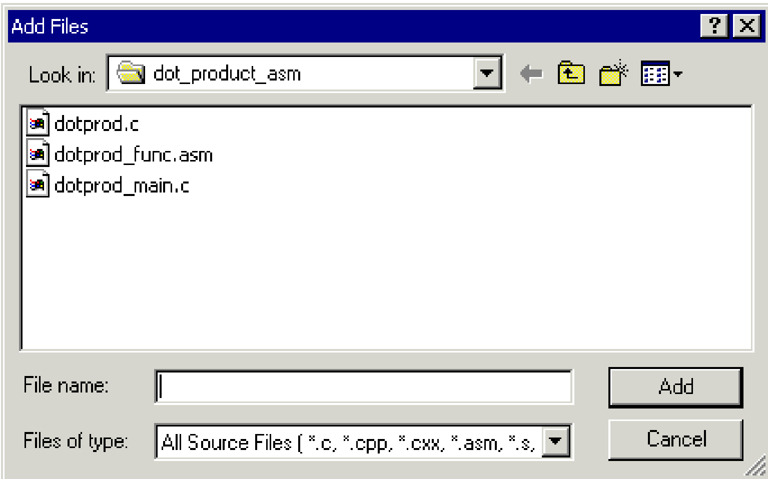


Figure 2-13. Add Files Dialog Box: Adding Source Files to the Project

2. In the **Look in** box, locate the project folder, `dot_product_asm`.

3. In the **Files of type** box, select **All Source Files**.

4. Hold down the **Ctrl** key and click `dotprod.c` and `dotprod_main.c`. Then click **Add**.

   To display the files that you added in step 4, open the `Source Files` folder in the **Project** window.

You are now ready to create a Linker Description File for the project.

Step 3: Create a Linker Description File for the Project

In this procedure, you will use the Expert Linker to create a Linker Description File for the project.

To create a Linker Description File:

1. From the **Tools** menu, choose **Expert Linker** and then choose **Create LDF** to open the **Create LDF Wizard**, shown in Figure 2-14.

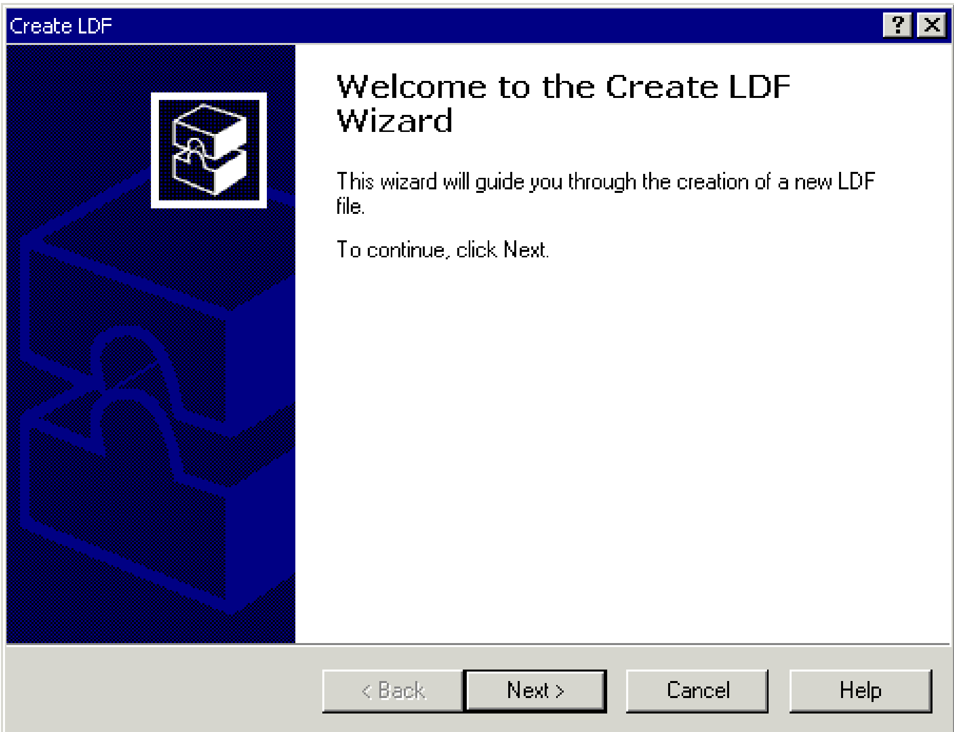


Figure 2-14. Create LDF Wizard

2. Click **Next** to display the **Create LDF – Step 1 of 3** page, shown in Figure 2-15.

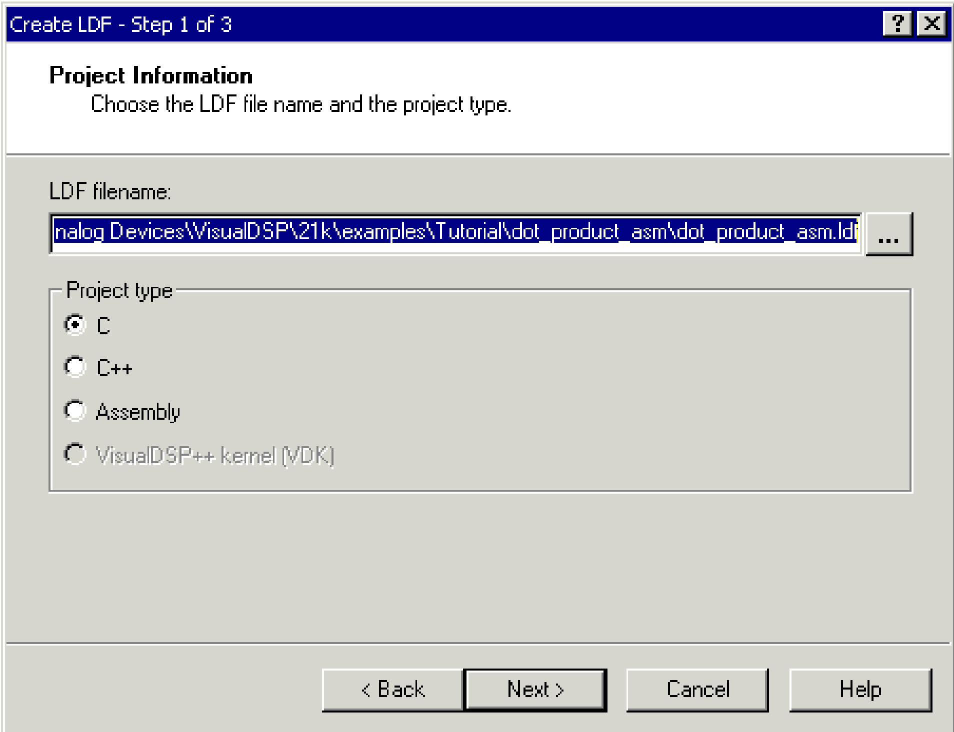


Figure 2-15. Create LDF – Step 1 of 3 Page

This page enables you to assign the **LDF file name** (based on the project name) and to select the **Project type**.

3. Accept the values selected for your project and click **Next** to display the **Create LDF – Step 2 of 3** page, shown in Figure 2-16 on page 2-23.

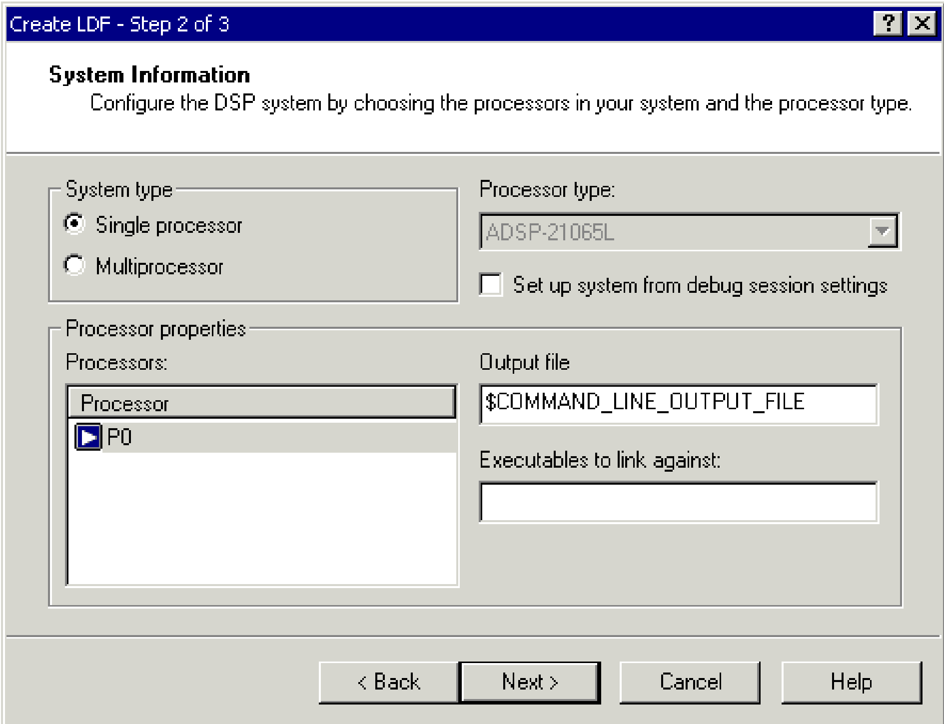


Figure 2-16. **Create LDF – Step 2 of 3** Page

This page enables you to set the **System type** (defaulted to **Single processor**), the **Processor type** (defaulted to **ADSP-21065L** to match the project), and the name of the linker **Output file** (defaulted to the name selected by the project).

4. Accept the default values and click **Next** to display the next page (**Create LDF – Step 3 of 3**), shown in Figure 2-17 on page 2-24.

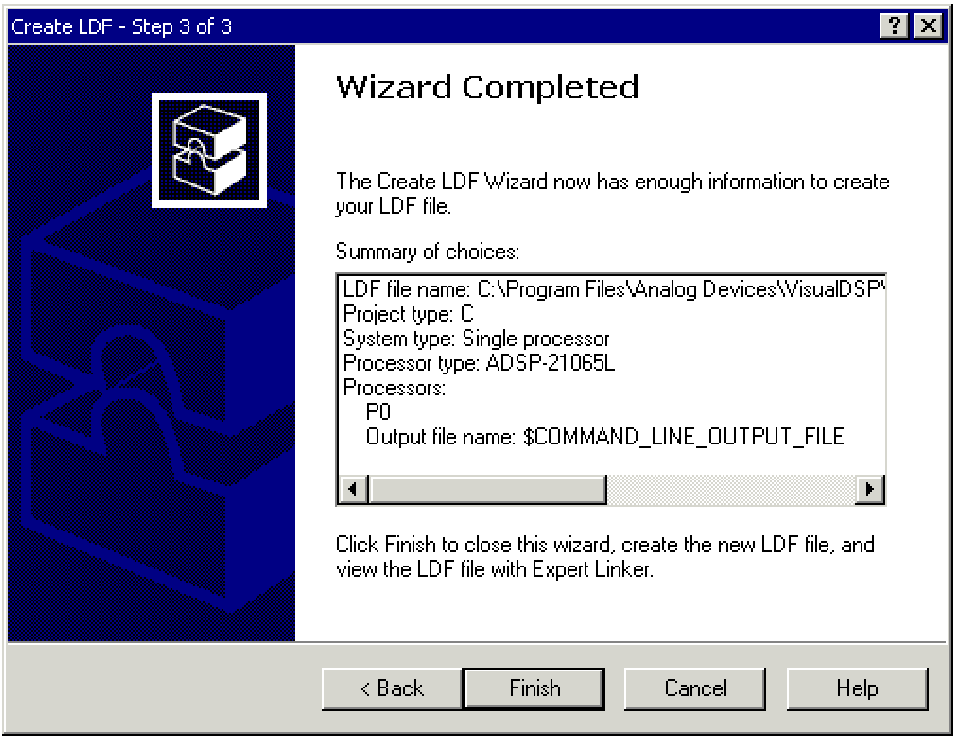
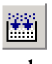


Figure 2-17. Create LDF – Step 3 of 3 Page

5. Review the **Summary of choices** and click **Finish** to create the `.LDF` file.

   You now have a new `.LDF` file in your project. The new file is in the **Linker Files** folder in the **Project** window.

   The **Expert Linker** window opens with a representation of the `.LDF` file that you created. This file is complete for this project. Close the **Expert Linker** window.

6. Click the **Build Project** button  to build the project. The C source file opens in an editor window, and execution halts.

The C version of the project is now complete. You are now ready to modify the sources to call the assembly function.

Step 4: Modify the Project Source Files

In this procedure, you will:

- Modify `dotprod_main.c` to call `a_dot_c_asm` instead of `a_dot_c`
- Save the modified file

To modify `dotprod_main.c` to call the assembly function:

1. Resize or maximize the editor window for better viewing.
2. From the **Edit** menu, choose **Find** to open the **Find** dialog box, shown in Figure 2-18.

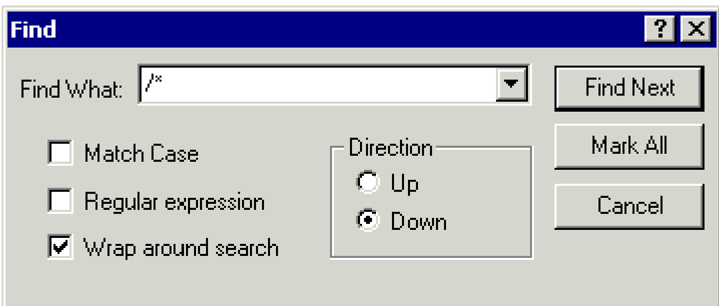


Figure 2-18. Find Dialog Box: Locating Occurrences of `/*`

3. In the **Find What** box, type `/*`, and then click **Mark All**.

   The editor bookmarks all lines containing `/*` and positions the cursor at the first instance of `/*` in the `extern double a_dot_c_asm` declaration.

4. Select the comment characters `/*` and use the **Ctrl+X** key combination to cut the comment characters from the beginning of the `a_dot_c_asm` declaration. Then move the cursor up one line and use the **Ctrl+V** key combination to paste the comment characters at the beginning of the `a_dot_c` declaration. Because syntax coloring is turned on, the code will change color as you cut and paste the comment characters.

   Repeat this step for the end-of-comment characters `*/` at the end of the `a_dot_c_asm` declaration. The `a_dot_c` declaration is now fully commented out, and the `a_dot_c_asm` declaration is no longer commented.

5. Press **F3** to move to the next bookmark.

   The editor positions the cursor on the `/*` in the function call to `a_dot_c_asm`, which is currently commented out. Note that the previous line is the function call to the `a_dot_c` routine.

6. Press **Ctrl+X** to cut the comment characters from the beginning of the function call to `a_dot_c_asm`. Then move the cursor up one line and press **Ctrl+V** to paste the comment characters at the beginning of the call to `a_dot_c`.

   Repeat this step for the end-of-comment characters `*/`. The `main()` function should now be calling the `a_dot_c_asm` routine instead of the `a_dot_c` function, previously called in Exercise One.

   Figure 2-19 on page 2-27 shows the changes made in step 6.

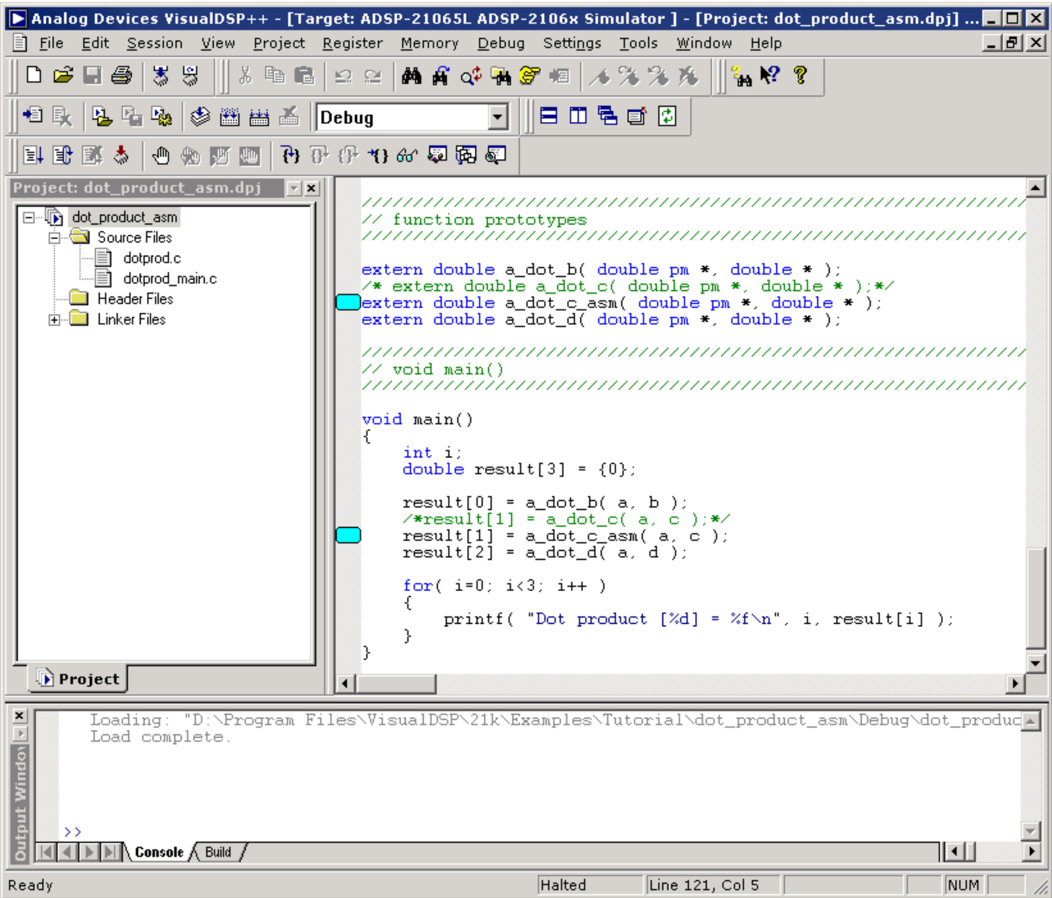


Figure 2-19. Editor Window: Modifying dotprod_main.c to Call a_dot_c_asm

7. From the **File** menu, choose **Save** to save the changes to the file.

8. Place the cursor in the editor window. Then, from the **File** menu, choose **Close** to close the `dotprod_main.c` file.

You are now ready to modify `dotprodasm.ldf`.

Step 5: Use the Expert Linker to modify dot_prod_asm.ldf

In this procedure you will:

- View the Expert Linker representation of the `.LDF` file that you created
- Modify the `.LDF` file to map in the section for the `a_dot_c_asm` assembly routine

To examine and then modify `dot_prod_asm.ldf` to link with the assembly function:

1. Click the **Add File** button .
2. Select `dotprod_func.asm` and click **Add**.
3. Try to build the project by performing one of these actions:
   - Click the **Build Project** button  .
   - From the **Project** menu, choose **Build Project**.

Notice the linker error in the **Output** window, shown in Figure 2-20.

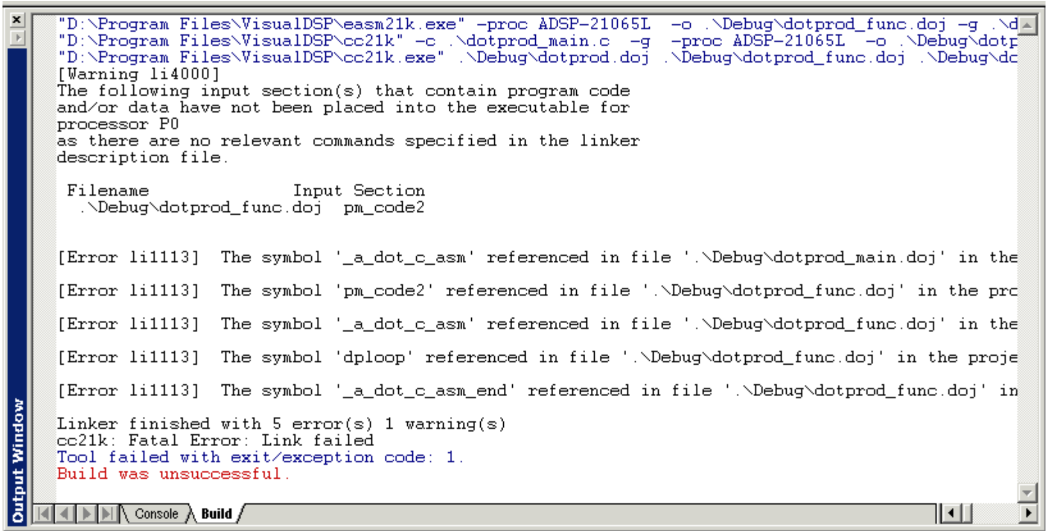


Figure 2-20. Output Window: Linker Error

4. In the **Project** window, open the **Linker Files** folder and double-click the `dot_prod_asm.ldf` file. The **Expert Linker** window (Figure 2-21) opens with a representation of your file.

You might have to resize the **Expert Linker** window and scroll to see both panes (**Input Sections** and **Memory Map**).

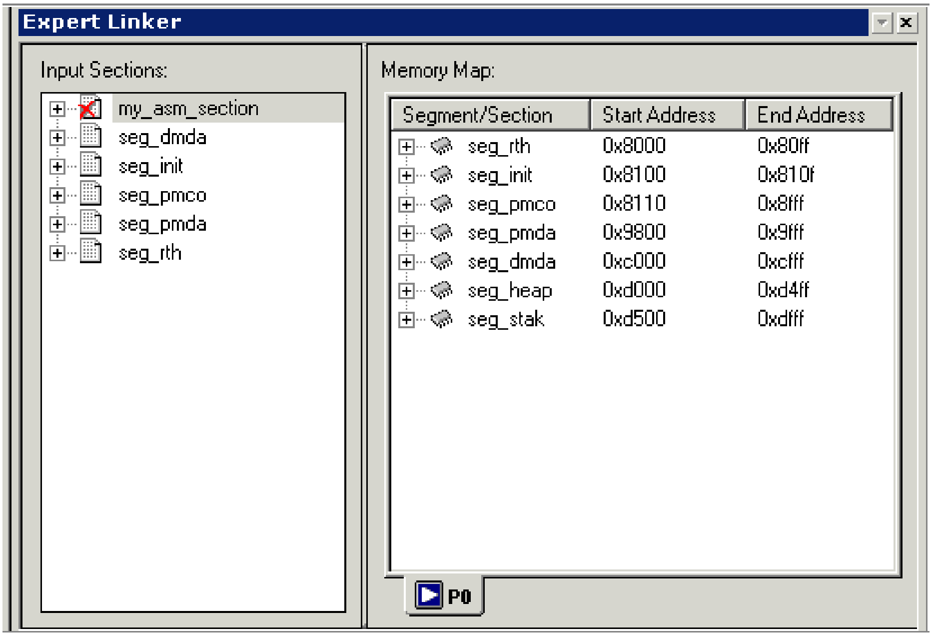


Figure 2-21. Expert Linker Window

The left pane contains a list of the **Input Sections** that are in your project or are mapped in the `.LDF` file. A red `X` is over the icon in front of the section named "`my_asm_section`" because the Expert Linker has determined that the section is not mapped by the `.LDF` file.

The right pane contains a graphical representation of the memory segments that the Expert Linker defined when it created the .LDF file. Change the view mode by right-clicking in the right pane and choosing **View Mode**. Then choose **Memory Map Tree** to display the tree view shown in Figure 2-21 on page 2-30.

5. Map the section `my_asm_section` into the memory segment named `seg_pmco` as follows.

   Open the `my_asm_section` input section by clicking on the plus sign in front of it. The input section expands to show that the linker macro `$OBJECTS` and the object file `dotprod_func.doj` both have a section that has not been mapped. Drag the icon in front of `$OBJECTS` to the memory map pane and onto `seg_pmco`. As shown in Figure 2-22 on page 2-32, the red `X` should no longer appear because the section `my_asm_section` has been mapped.

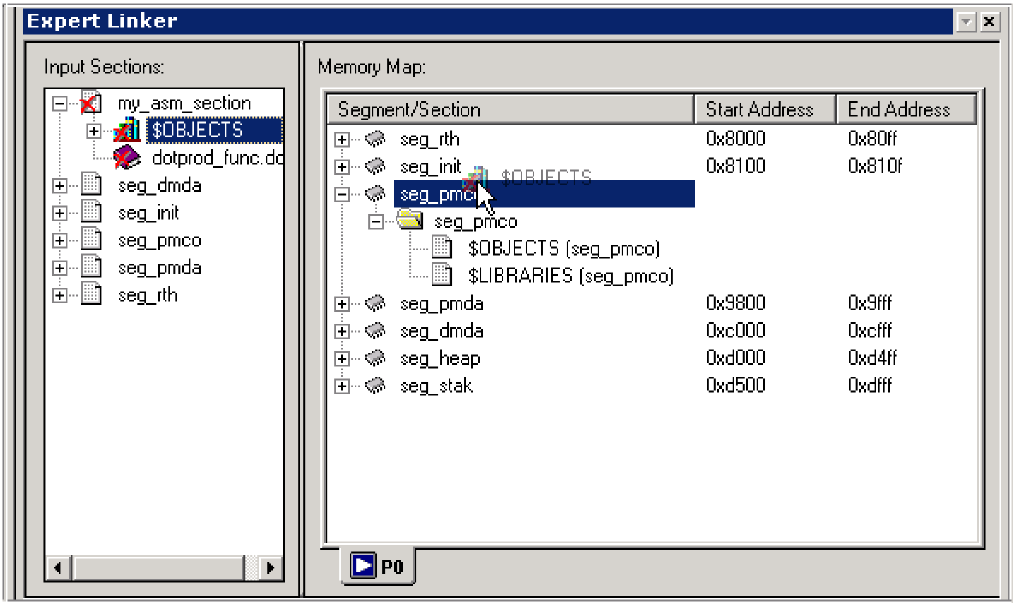


Figure 2-22. Dragging $OBJECTS onto `seg_pmco`

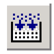
From the **Tools** menu, choose **Expert Linker** and **Save** to save the modified file. Then close the **Expert Linker** window.

If you forget to save the file and then rebuild the project, VisualDSP++ will see that you modified the file and will automatically save it.

You are now ready to rebuild and run the modified project.

Step 6: Rebuild and Run dot_product_asm

To run `dot_product_asm`:

1. Build the project by clicking the **Build Project** button  or by choosing **Build Project** from the **Project** menu.

   At the end of the build, the **Output** window displays "`Build completed successfully`" in the **Build** view. VisualDSP++ loads the program, runs to main, and displays the **Output**, **Disassembly**, and editor windows (shown in Figure 2-23).

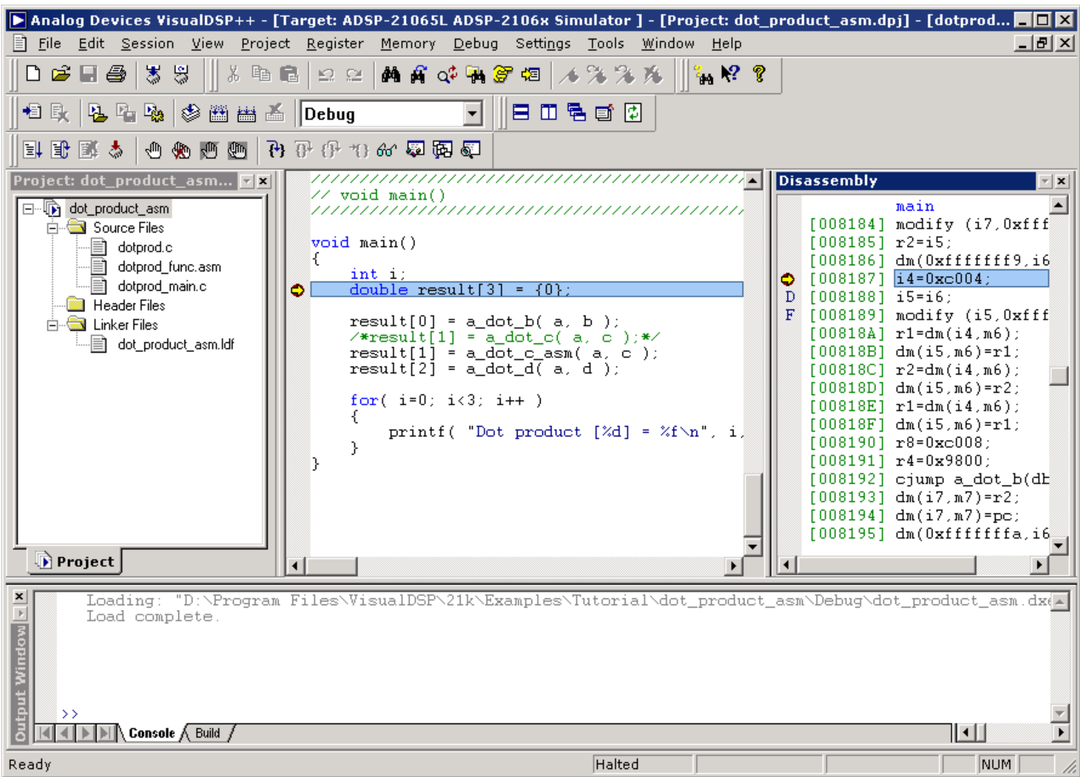


Figure 2-23. dot_product_asm Successfully Built and Loaded

2. Click the **Run** button  to run `dot_product_asm`.

   The program calculates the three dot products and displays the results in the **Console** view in the **Output** window. When the program stops running, the message "`Halted`" appears in the status bar at the bottom of the window. The results, shown below, are identical to the results obtained in Exercise One.

   ```
   Dot product [0] = 0.000000
   Dot product [1] = 0.707107
   Dot product [2] = -0.500000
   ```

You are now ready to begin Exercise Three.

Exercise Three: Plotting Data

In this exercise, you will load and debug a pre-built program that applies a simple convolution algorithm to a buffer of data. You will use the VisualDSP++ plotting engine to view the different data arrays graphically, both before and after running the program.

Step 1: Load the Convolution Program

To load the Convolution program:

1. Close the `dot_product_asm` project, but keep the **Disassembly** window and **Output** window (in the **Console** view) open.
2. From the **File** menu, choose **Load Program** or click  . The **Open a Processor Program** dialog box appears.
3. Select the `convolution.dxe` program to load as follows.
   a. Open your `Analog Devices` folder and double-click the `VisualDSP\21k\Examples\tutorial\convolution\Debug` subfolder.

b. Double-click `convolution.dxe` to load the program. in an editor window.

c. If you are prompted to look for `convolution.cpp`, click **Yes** to open the **Find** dialog box and proceed to step d. If VisualDSP++ opens an editor window, proceed to step 4 on page 2-36.

d. Click the up-one-level button  to access the `convolution` folder.

e. Double-click `convolution.cpp` to display the file in an editor window, as shown in Figure 2-24.

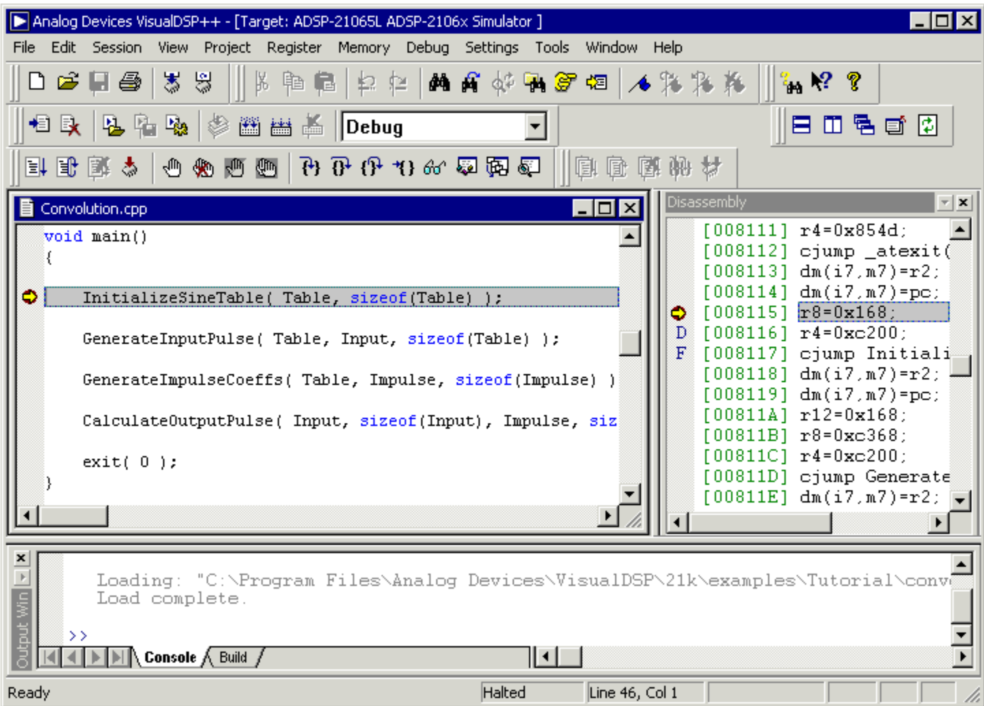


Figure 2-24. Loading the Convolution Program

4. Look at the source code of the Convolution program.

   You can see four global data arrays:

   `Table`

   `Input`

   `Output`

   `Impulse`

   You can also see four functions that operate on these arrays:

   `InitializeSineTable()`

   `GenerateInputPulse()`

   `GenerateImpulseCoeffs()`

   `CalculateOutputPulse()`

You are now ready to open a plot window.

Step 2: Open a Plot Window

To open a plot window:

1. From the **View** menu, choose **Debug Windows** and **Plot**. Then choose **New** to open the **Plot Configuration** dialog box, shown in Figure 2-25 on page 2-37.

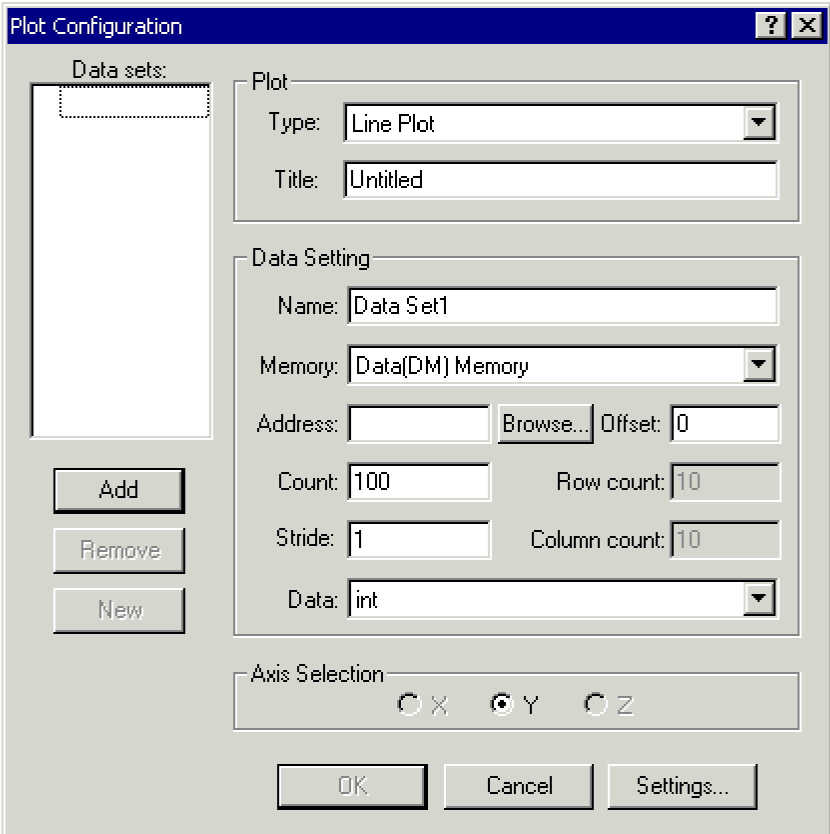


Figure 2-25. Plot Configuration Dialog Box

Here you will add the data sets that you want to view in a plot window.

2. In the **Plot** group box, specify the following values.

   - In the **Type** box, select **Line Plot** from the drop-down menu.
   - In the **Title** box, type **convolution**.

3. Enter three data sets to plot by using the values in Table 2-4.

Table 2-4. Three Data Sets: Table, Input, and Output

Data Setting Field	Table Data Set	Input Data Set	Output Data Set	Description
Name	Table	Input	Output	Data set
Memory	Data(DM) Memory	Data(DM) Memory	Data(DM) Memory	Data memory
Address	Table	Input	Output	The address of this data set is that of the Input or Output array. Click **Browse** to select the value from the list of loaded symbols.
Count	360	360	396	The arrays are 360 and 396 elements long.
Stride	1	1	1	The data is contiguous in memory.
Data	float	float	float	Input and Output are arrays of float values.
Offset	0	0	0	Use zero, the default value.

After entering each data set, click **Add** to add the data set to the **Data Sets** list. The **Plot Configuration** dialog box should now look like the one in Figure 2-26 on page 2-39.

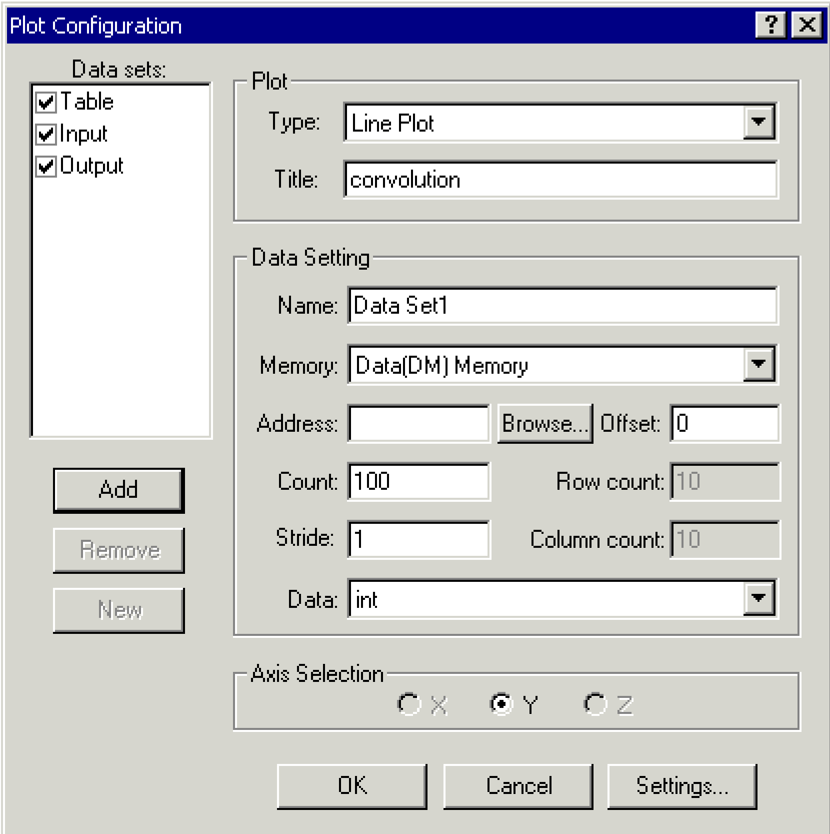


Figure 2-26. Plot Configuration Dialog Box with Table/Input/Output Data Sets

4. Click **OK** to apply the changes and to open a plot window with these data sets.

   The plot window now displays the three arrays. Since, by default, the simulator initializes memory to zero, the data sets appear as one horizontal line, shown in Figure 2-27.

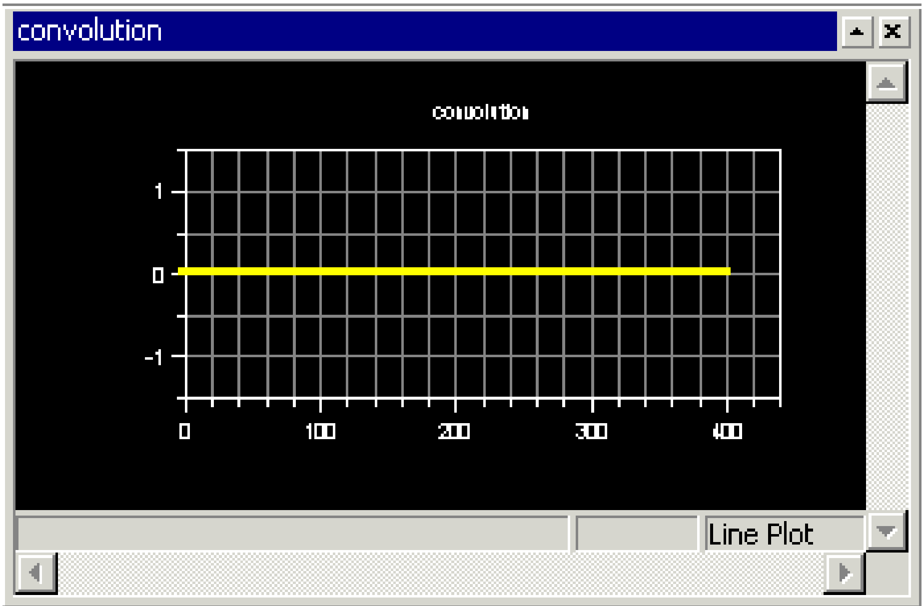


Figure 2-27. Plot Window: Before Running the Convolution Program

To display the legend box in the plot window, right-click in the plot window and choose **Modify Settings**. Then, on the **General** page, select **Legend** in the **Options** group box.

The legend box is not shown in the plot windows shown in this tutorial.

Step 3: Run the Convolution Program and View the Data

To run the Convolution program and view the data:

1. Press **F10** or click the **Step Over** button to step over the first line in main that calls the `InitializeSine Table()` function.

   Stepping over each function enables you to see the data being calculated in a plot window.

2. Step over the call to `GenerateInputPulse()` by using the **Step Over** command as you did in the previous step. The plot window now displays the data for both the Input array and the Table array.

   Once you finish stepping over the function, the word "`Halted`" appears in the status bar at the bottom of the screen. The plot window should now show the sine wave data in the Table array.

3. Press **F5** or click the **Run** button to run to the end of the program.

When the program halts, you see the results of the `convolution` algorithm in the Output array. All three data sets are now visible in the plot window, as shown in Figure 2-28.

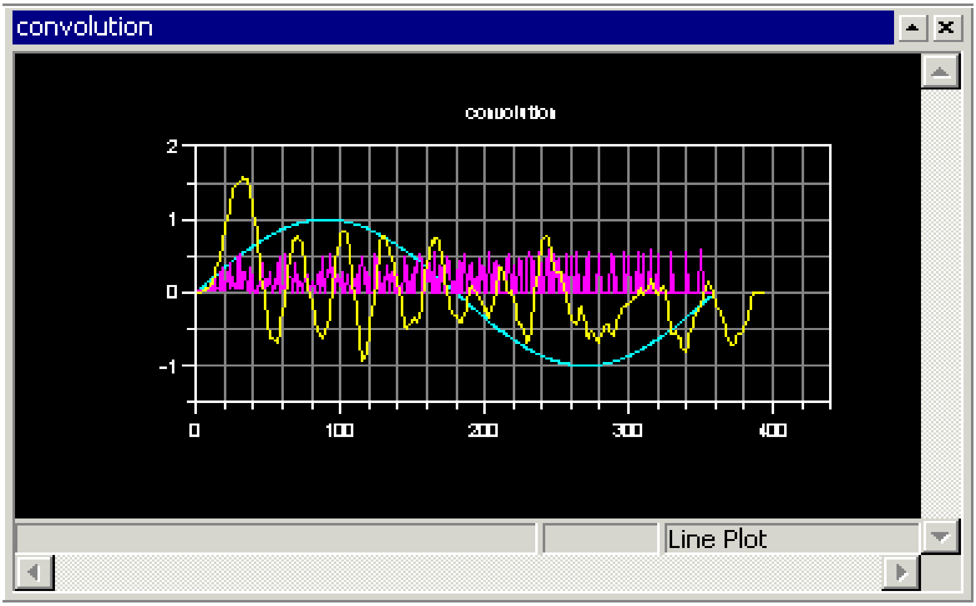


Figure 2-28. Plot Window After Running the Convolution Program to Completion

Next you will zoom in on a particular region of interest in the plot window to focus in on the data.

4. Click the left mouse button inside the plot window and drag the mouse to create a rectangle to zoom into. Then release the mouse button to magnify the selected region.

   Figure 2-29 shows the selected region.

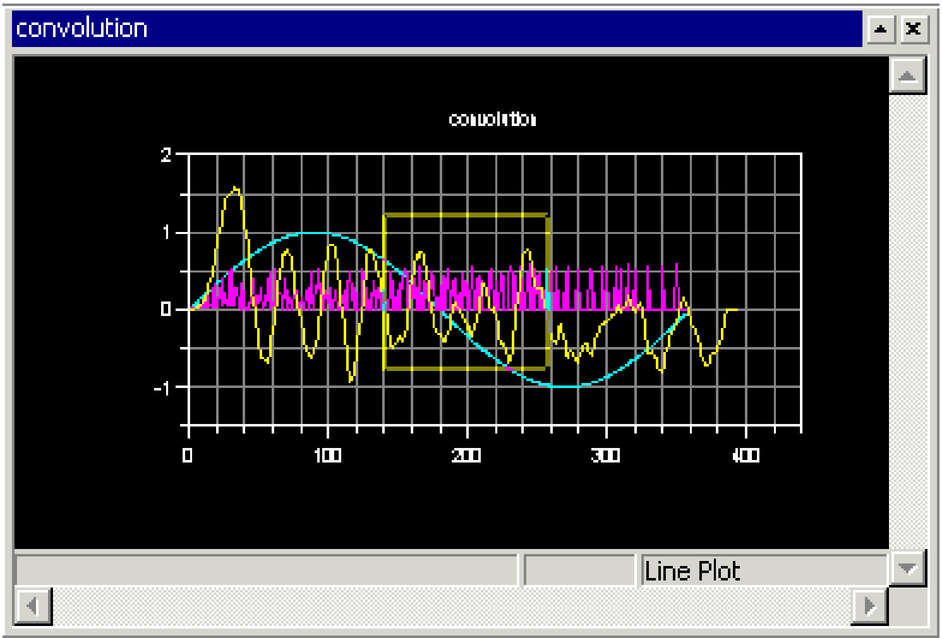


Figure 2-29. Plot Window: Selecting a Region to Magnify

Figure 2-30 on page 2-44 shows the magnified results.

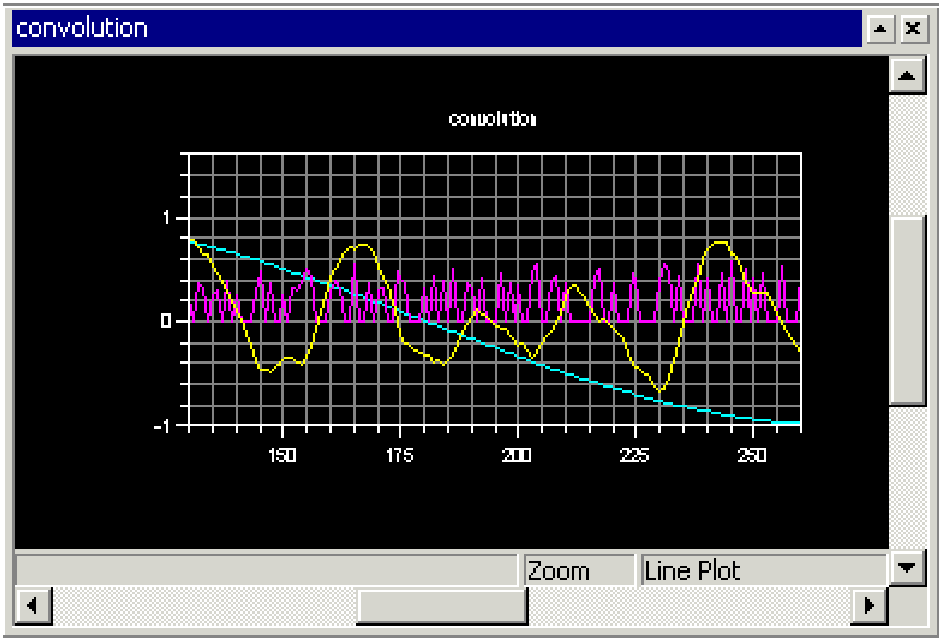


Figure 2-30. Plot Window: Magnified Result

To return to the view before magnification, right-click in the plot window and choose **Reset Zoom** from the menu. You can view individual data points in the plot window by enabling the data cursor, as explained in the next step.

5. Right-click inside the plot window and choose **Data Cursor** from the popup menu. Then move through the individual data points in the current data set by pressing and holding the Left (←) and Right (→) arrow keys on the keyboard. The value of the current data point appears in the lower-left corner of the plot window, as shown in Figure 2-31 on page 2-45.

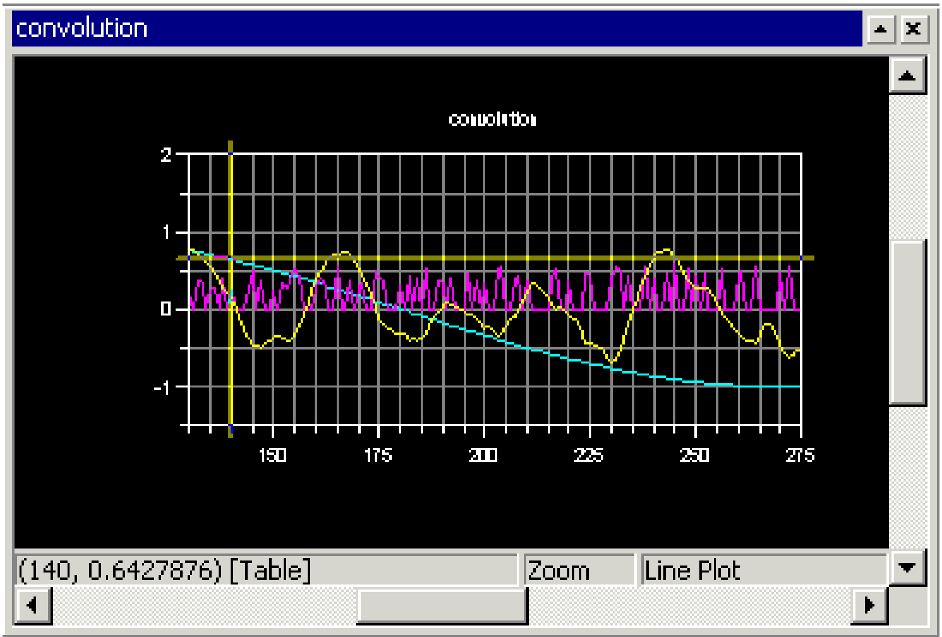


Figure 2-31. Plot Window: Using the Data Cursor Feature

To switch data sets, press the Up (↑) and Down (↓) arrow key.

To disable the data cursor, right-click in the plot window and choose (de-select) **Data Cursor**.

To return to the previous view (before magnification), right-click in the plot window and choose **Reset Zoom** from the popup menu.

You are now ready to begin Exercise Four.

Exercise Four: Linear Profiling

In this exercise, you will load and debug the Convolution program from the previous exercise. You will use linear profiling, however, to evaluate the program's efficiency and to determine where the application is spending the majority of its execution time in the code.

VisualDSP++ supports two types of profiling: linear and statistical.

- You use linear profiling with a simulator. The count in the **Linear Profiling Results** window is incremented every time a line of code is executed.
- You use statistical profiling with a JTAG emulator connected to a DSP target. The count in the **Statistical Profiling Results** window is based on random sampling.

Step 1: Load the Convolution Program

To load the Convolution program:

1. Close all open windows except for the **Disassembly** window and the **Output** window.
2. From the **File** menu, choose **Load Program**, or click . The **Open a Processor Program** dialog box appears.
3. Select the program to load as follows.
    a. Open the `Analog Devices` folder and double-click the `VisualDSP\21k\Examples\tutorial\convolution\Debug` subfolder.
    b. Double-click `convolution.dxe` to load and run the Convolution program.

c. If you are prompted to look for `convolution.cpp`, click **Yes** to open the **Find** dialog box and proceed to step d. If VisualDSP++ opens an editor window, proceed to "Step 2: Open the Profiling Window."

d. Click the up-one-level button  to access the `convolution` folder.

e. Double-click `convolution.cpp` to display the file in an editor window.

You are now ready to set up linear profiling.

Step 2: Open the Profiling Window

To open the **Linear Profiling Results** window:

1. From the **Tools** menu, choose **Linear Profiling** and then choose **New Profile**.

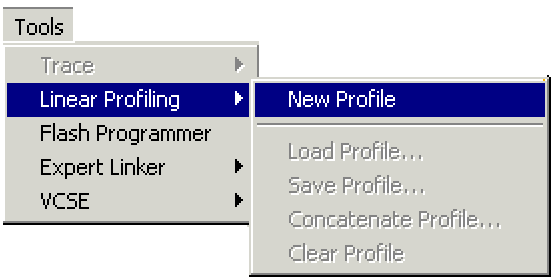


Figure 2-32. Setting Up Linear Profiling for the Convolution Program

The **Linear Profiling Results** window opens.

2. For a better view of the data, use the window's title bar to drag and dock the window to the top of the VisualDSP++ main window, as shown in Figure 2-33.

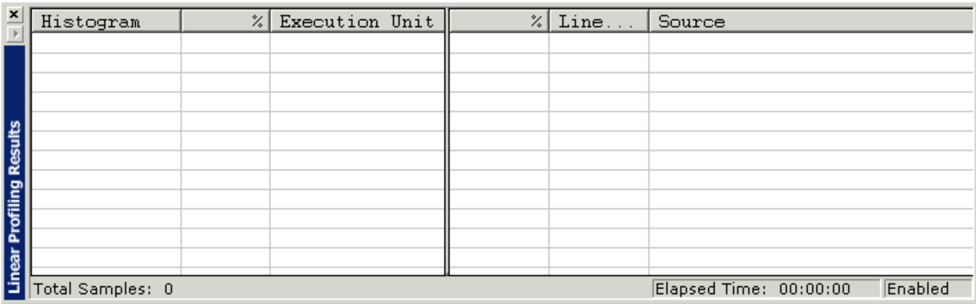


Figure 2-33. Linear Profiling Results Window (Empty)

The **Linear Profiling Results** window is initially empty. Linear profiling will be performed when you run the `convolution` program. After you run the program and collect data, this window displays the results of the profiling session.

Since we are interested only in high level source code at this point, we will filter out any samples that do not map directly to our source code.

3. Right-click in the **Linear Profiling Results** window and choose **Properties** to display the **Profile Window Properties** dialog box, shown in Figure 2-34.

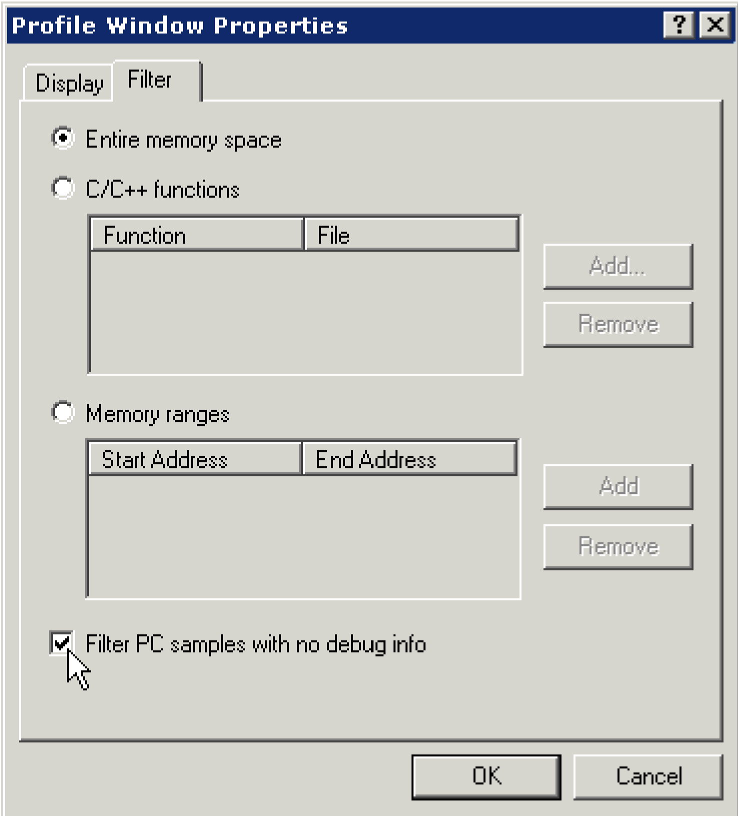


Figure 2-34. Filtering Samples with No Debug Information

4. Select the **Filter** tab (shown in Figure 2-34) and then click in the **Filter PC samples with no debug info** check box to enable the filter. Click **OK** to close the dialog box.

You are now ready to collect and examine linear profile data.

Step 3: Collect and Examine the Linear Profile Data

To collect and examine the linear profile data:

1. Press **F5** or click to run to the end of the program.

   When the program halts, the results of the linear profile appear in the **Linear Profiling Results** window.

2. Examine the results of your linear profiling session.

   The **Linear Profiling Results** window is divided into two, three-column panes. The left pane displays the results of the profile data, as shown in Figure 2-35.

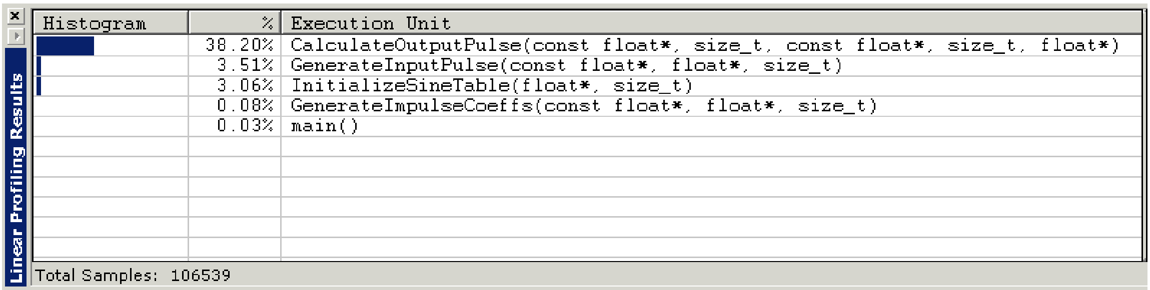


Figure 2-35. Linear Profiling Results of Analyzing the Performance of the Convolution Program – Left Pane

Double-clicking on a line in the left pane displays the corresponding source code for the profile data in the right pane, as shown in Figure 2-36.

If you are prompted to look for `convolution.cpp`, complete these steps:

a. Click **Yes** to open the **Find** dialog box.

b. Click the up-one-level button  to access the `convolution` folder.

c. Double-click `convolution.cpp` to display the file in an editor window.

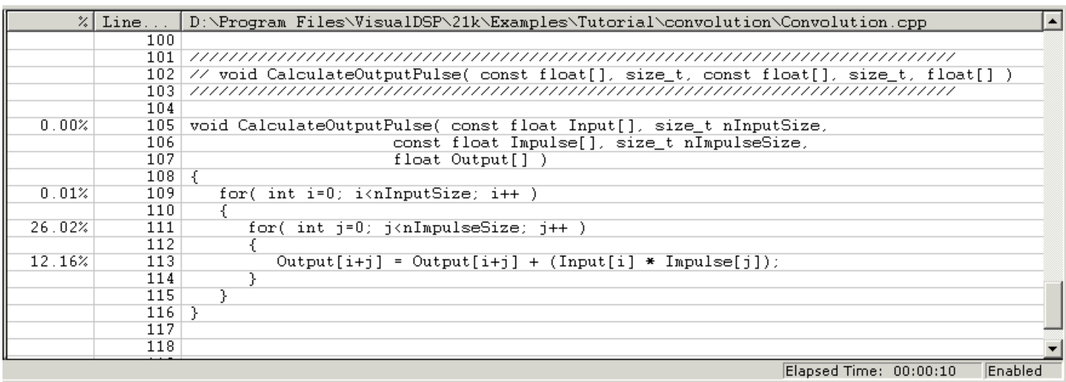


Figure 2-36. Linear Profiling Results of Analyzing the Performance of the Convolution Program – Right Pane

The field values in the left pane are defined on the next page.

Histogram	A graphical representation of the percentage of time spent in a particular execution unit. This percentage is based on the total time that the program spent running, so longer bars denote more time spent in a particular execution unit. The **Linear Profiling Results** window always sorts the data with the most time-consuming (expensive) execution units at the top.
%	The numerical percent of the same data found in the Histogram column. You can view this value as an absolute number of samples by right-clicking in the **Linear Profiling Results** window and by selecting **View Sample Count** from the popup menu.
Execution Unit	The program location to which the samples belong. If the instructions are inside a C function or a C++ method, the execution unit is the name of the function or method. For instructions that have no corresponding symbolic names, such as hand-coded assembly or source files compiled without debugging information, this value is an address in the form of `PC[xxx]`, where `xxx` is the address of the instruction.

If the instructions are part of an assembly file, the execution unit is the assembly file followed by the line number in parentheses.

In Figure 2-35 on page 2-50 the left pane shows that the function `CalculateOutputPulse()` has consumed over 38% of the total execution time. Double-clicking one of these lines displays the source file, `convolution.cpp`, in the right (source) pane. The source pane displays data for each line of executable code in the file for which linear profile data has been collected.

Double-clicking the line with the `CalculateOutputPulse()` function in the left pane displays the linear profile data shown in Figure 2-37 in the right pane.

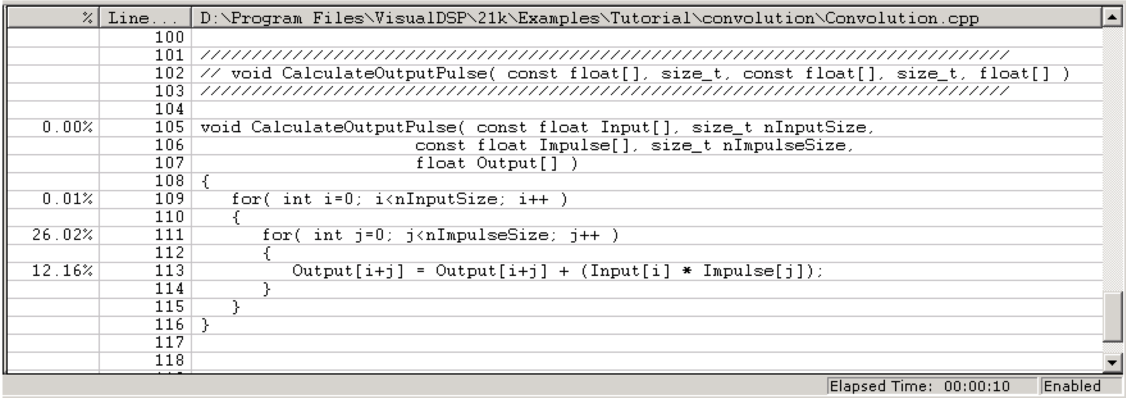


Figure 2-37. Linear Profile Data for Convolution.cpp

The details of the `CalculateOutputPulse()` function show that 26.02% of the time spent running the entire Convolution program is spent inside the nested `for` loop, calculating the convolution.

The data suggests that you should rewrite this function in hand-tuned assembly language to decrease the total running time of the algorithm and improve performance.

You are now ready to begin Exercise Five.

Exercise Five: Installing and Using a VCSE Component

In this exercise, you will complete the following tasks.

- Start up the VisualDSP++ environment and select a new session
- Open an existing project
- Install a VCSE component on your system
- Add the component to the project
- Build and run the program with the component

The sources for the exercise are in the `vcse_component` folder. The default installation path is:

```
Program files\Analog Devices\VisualDSP\21k\Examples\tutorial\
vcse_component
```

Step 1: Start VisualDSP++ and Open the Project

To start VisualDSP++ and open the project:

1. Click the Windows **Start** button and select **Programs**, **VisualDSP**, and **VisualDSP++ Environment**.

   The VisualDSP++ main window appears.

   If you have already run VisualDSP++ and the **Reload last project at startup** option is selected on the **Project** page under **Settings** and **Preferences**, VisualDSP++ opens the last project that you worked on.

   To close this project, choose **Close** from the **Project** menu and then click **No** when prompted to save the project. Since you have made no changes to the project, you do not have to save it.

2. From the **Sessions** menu, choose **New Session**. The **New Session** dialog box appears.

3. From the **Processor** list, choose the **ADSP-21060** processor and click **OK**.

4. From the **Project** menu, choose **Open**.

   VisualDSP++ displays the **Open Project** dialog box.

5. In the **Look in** box, open the `Program Files\Analog Devices` folder and double click the following sub-folders in succession.

   `VisualDSP\21k\Examples\tutorial\vcse_component`

   Note: This path is based on the default installation.

6. Double-click the `useg711.dpj` project file.

   VisualDSP++ loads the project and displays messages in the **Output** window as it processes the project settings.

Note: The first time that you open projects installed from the software kit, VisualDSP++ may detect that files, folders, or both have moved. If you receive a "`Project has been moved`" message, click **OK** to continue.

The useg711 project contains a single C language source file `useg711.c`, which contains the code needed to create an instance of the CULawc component and to invoke the methods of the IG711 interface.

Step 2: Install the EXAMPLES::CULawc Component

The EXAMPLES::CULawc component is distributed as part of VisualDSP++ and is ready to be installed on your system.

1. From the **Tools** menu, select the **VCSE** submenu and then choose **Manage Components**.
2. In the **Display** field, select **Downloaded component package…** from the drop-down list.

   The **Open** dialog box is displayed.
3. In the **Look in** box, open the `Program Files\Analog Devices` folder and double-click the following subfolders in succession.

   `VisualDSP\21k\Examples\tutorial\vcse_component`

   Note: This path is based on the default installation.
4. Double-click the `examples_culawc_21K.vcp` file.

   VisualDSP++ opens the file, extracts the information about the component, and shows it as a downloaded component in the **Component Manager** dialog box (Figure 2-38 on page 2-57).

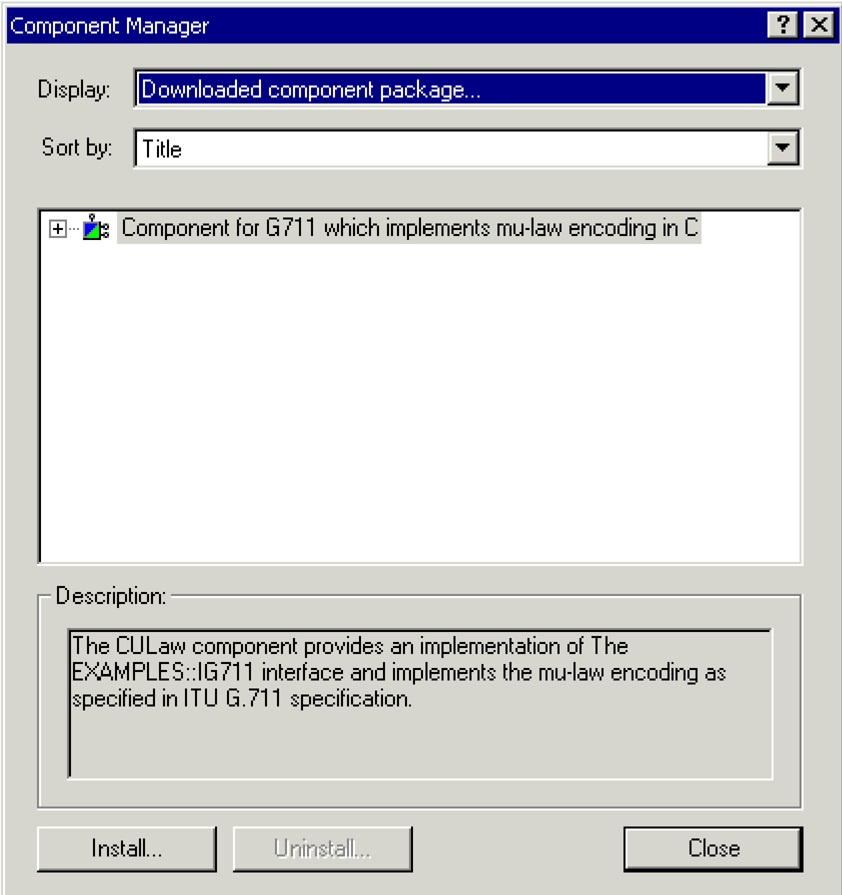


Figure 2-38. Component Manager Dialog Box – Downloaded Component

5. Click the **Install…** button to install the component on your system. Once the component is installed, click **OK**.

6. In the **Display** field, select **Locally installed components** from the drop-down list, and in the **Sort by** field, select **Title**.

   Select **Component for G711 which implements the mu-law encoding in C**. Component Manager displays the dialog box shown in Figure 2-39.

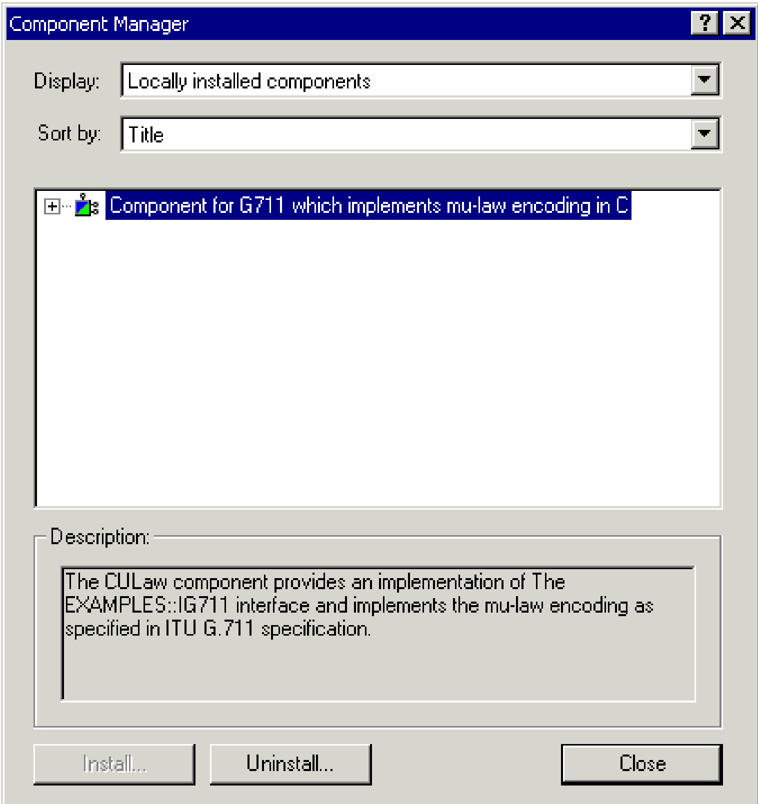


Figure 2-39. Component Manager Dialog Box – Selected Component

7. Click **Close** to close Component Manager.

Step 3: Add the Component to Your Project

To add the newly installed component to the project:

1. From the **Tools** menu, select the **VCSE** submenu and then choose **Add Component**.

2. Click **Component for G711 which implements the mu-law encoding in C** to select it.

   If you have multiple components on your system and you are not sure which one to add, click the expand button ⊞ to display the component information, as shown in Figure 2-40.

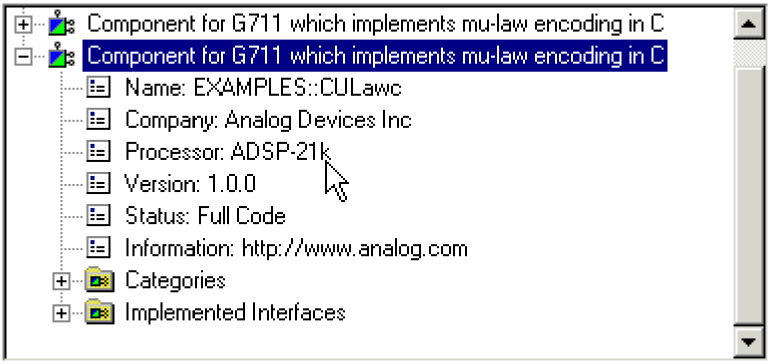


Figure 2-40. Expanded View of Component Information

   Make sure that **Processor: ADSP-21k** is listed for the component that you are adding to your project.

3. Click **Add** to indicate that you want to add the component to the project. Component Manager displays the dialog box shown in Figure 2-41.

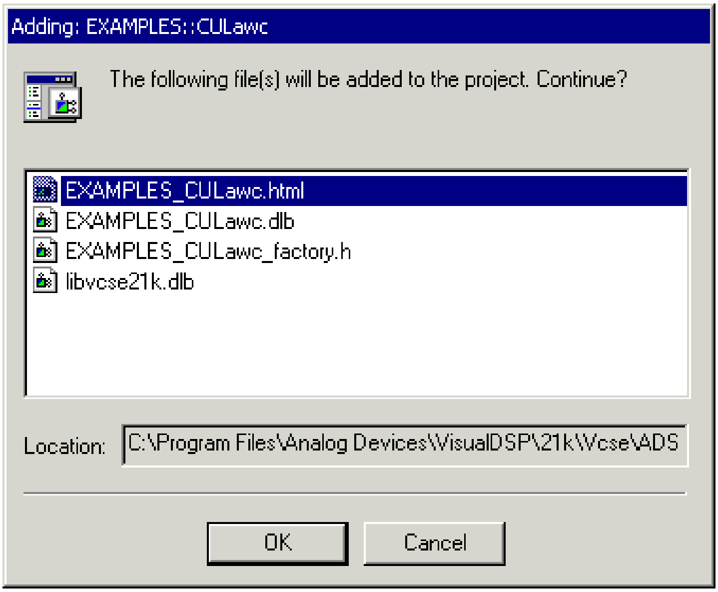


Figure 2-41. Adding Files to the Project

4. Click **OK** to add the component files to the project.

Step 4: Build and Run the Program

To build and run the program:

1. From the **Project** menu, choose **Build Project**.

   VisualDSP++ displays the message shown in Figure 2-42.

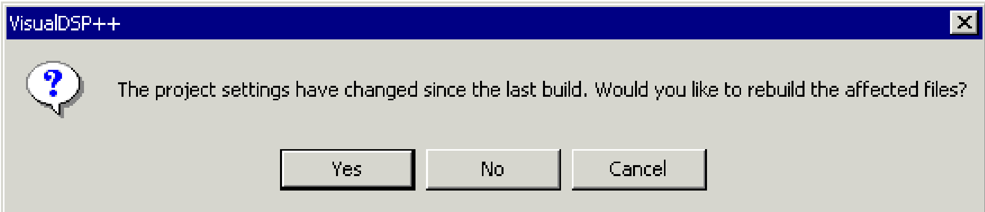


Figure 2-42. Rebuilding Files Affected by Changes to Project Settings

2. Click **Yes**. VisualDSP++ compiles the source files and creates the program.

3. From the **Debug** menu, choose **Run** to execute the program. The program generates the following output.

```
Harness to test component code generated by
EXAMPLES_CULawc.idl

Testing EXAMPLES::IG711
Test Completed result = MR_OK
```

You have now completed this exercise and the tutorial.