AccelDSP Synthesis Tool

User Guide

UG634 (v11.1) April 27, 2009

XILINX®

Chapter 1: Introducing AccelDSP Synthesis

Chapter 2: Installation

Chapter 3: Understanding the Synthesis Flow

Chapter 4: Creating a New Project

Chapter 5: Understanding Generate Fixed Point

Chapter 6: Exploring Hardware Architectures

Chapter 7: Interfacing to System Hardware

Chapter 8: AccelDSP Commands

Chapter 9: AccelDSP Getting Started Tutorial

Chapter 10: Common Messages

Chapter 11: Reserved Words

About This Guide

The AccelDSP™ synthesis tool is a product that allows you to transform a MATLAB® floating-point design into a hardware module that can be implemented in a Xilinx FPGA. The AccelDSP synthesis tool features an easy-to-use Graphical User Interface that controls an integrated environment with other design tools such as MATLAB, Xilinx ISE tools, and other industry- standardHDL simulators and logic synthesizers. This document is your main guide on how to use the AccelDSP synthesis tool. The other companion documents are the MATLAB for Synthesis Styles Guide and AccelDSP Release Notes.

Guide Contents

This manual contains the following chapters:

- [“Chapter, comma” cross-reference to chapter 1] [explain chapter content here]
- [List each additional chapter in a separate bulleted item.]
- [“Appendix, comma” cross-reference to the first appendix], [explain appendix content here]
- [List each additional appendix in a separate bulleted item.]
- [Do not list the glossary in this bulleted list.]

Additional Resources

To find additional documentation, see the Xilinx website at:

http://www.xilinx.com/literature.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

http://www.xilinx.com/support.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File →Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** `[option_name]` `design_name`
Braces { }	A list of items from which you must choose one or more	**`lowpwr =`**{**`on`**|**`off`**}
Vertical bar |	Separates items in a list of choices	**`lowpwr =`**{**`on`**|**`off`**}

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis ...	Repetitive material that has been omitted	**`allow block`** *block_name loc1 loc2 ... locn;*

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the *Virtex-II Platform FPGA User Guide.*
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

XILINX®

Chapter 1

Introducing AccelDSP Synthesis

Welcome to the AccelDSP™ Synthesis Tool, the only DSP (Digital Signal Processing) synthesis tool that allows you to transform a MATLAB® floating-point design into a hardware module that can be implemented in a Xilinx FPGA. The AccelDSP synthesis tool features an easy-to-use Graphical User Interface that controls an integrated environment with other design tools such as MATLAB, Xilinx ISE tools, and other industry- standard HDL simulators and logic synthesizers.

AccelDSP Synthesis provides the following capability:

- Reads and analyzes a MATLAB floating-point design
- Automatically creates an equivalent MATLAB fixed-point design
- Invokes a MATLAB simulation to verify the fixed-point design
- Provides you with the power to quickly explore design trade-offs of algorithms that are optimized for the target FPGA architectures
- Creates a synthesizable RTL HDL model and a Testbench to ensure bit-true, cycle-accurate design verification
- Provides scripts that invoke and control down-stream tools such as HDL simulators, RTL logic synthesizers, and Xilinx ISE implementation tools.

The graphical user interface (GUI) is shown in Figure 1-1 and features a Design Flow Manager to guide you quickly through the design transformation steps. The GUI also features a Project Explorer window that lets you graphically browse the design hierarchy and view the M-files and the generated HDL source files. A Tcl command line interface is provided through the Tcl window.

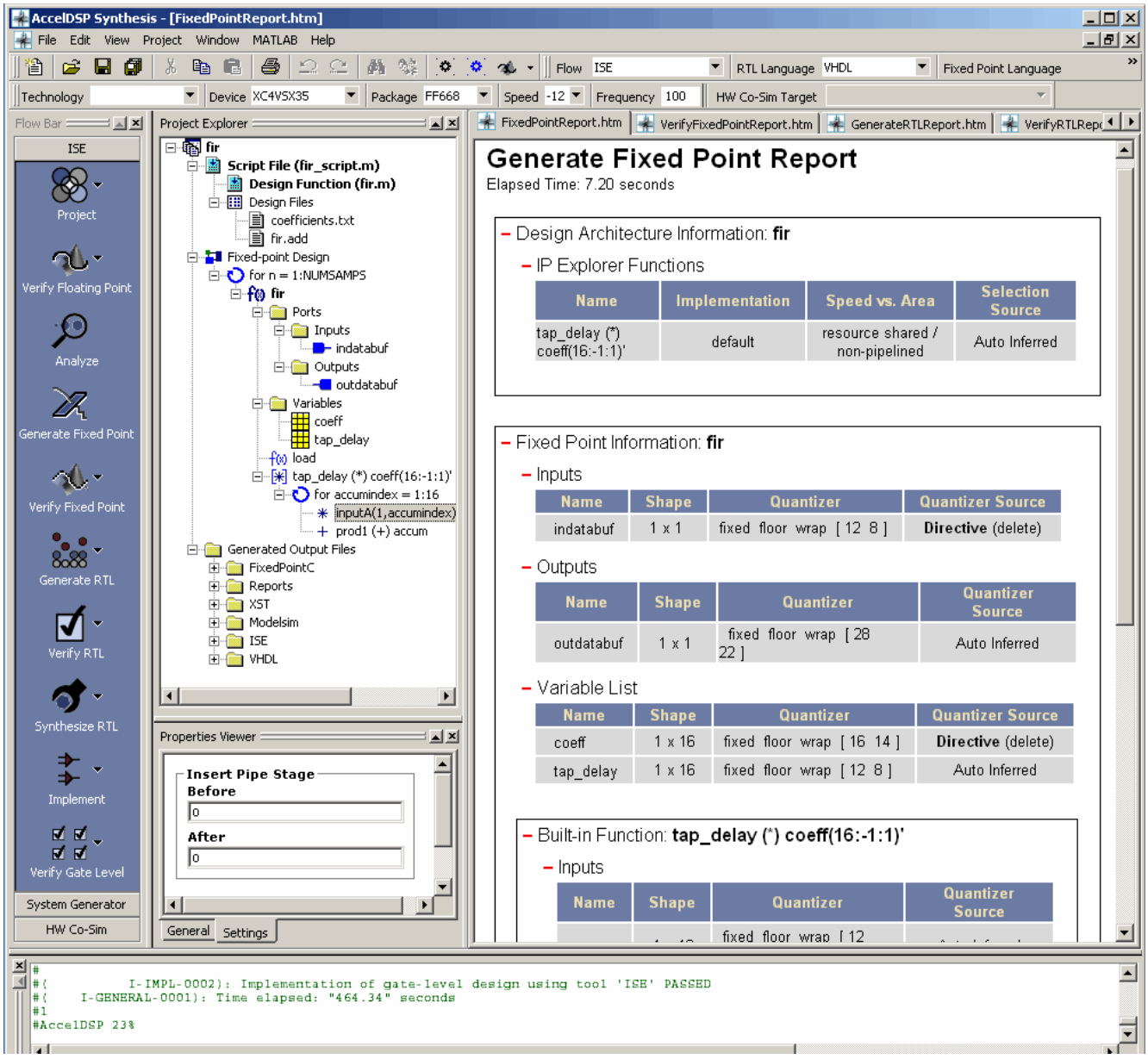


Figure 1-1: **The AccelDSP Synthesis Tool Main Window**

Un-Docking and Re-Docking ToolBars

The ToolBars in the main GUI window are arranged in a particular position by default. As shown below, you may un-dock or re-position a ToolBar by left-clicking on the header and dragging the ToolBar to a new position. If you un-dock a ToolBar, you can re-dock it by double-clicking on the title.

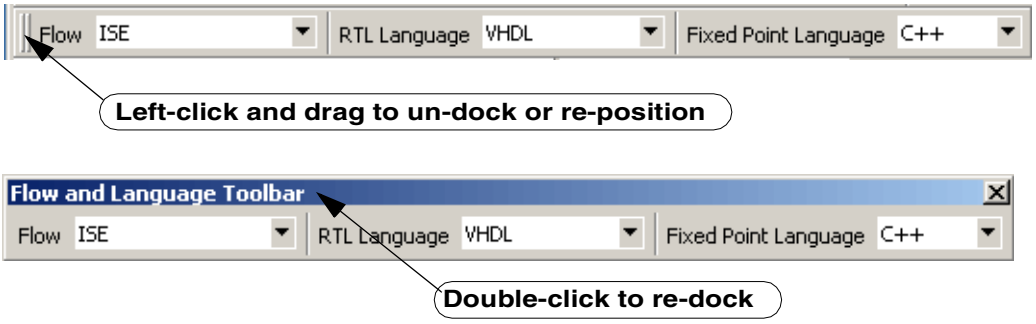


Figure 1-2: **Re-Positioning and Re-Docking ToolBars**

AccelDSP Synthesis Help

You can access the AccelDSP synthesis tool help topics from the **Help** pulldown menu. Help topics are provided in the following two formats:

- Microsoft HTML Help (Win98) for easy access and full text search
- PDF format for easy printing

Getting Started Tutorial

The AccelDSP synthesis tool provides a Getting Started Tutorial to help you get a quick start. As soon as you have AccelDSP Synthesis and other supporting tools installed, you can follow the exercises in the Chapter AccelDSP Getting Started Tutorial.

Transcript Message Help

The AccelDSP transcript provides a Hypertext Links from some messages into the online help. The Hypertext Links are designed to provide more in-depth information on the message and guide you to related background material in the documentation. Just double-click on the blue highlighted message number and the online help will open to the correct page.

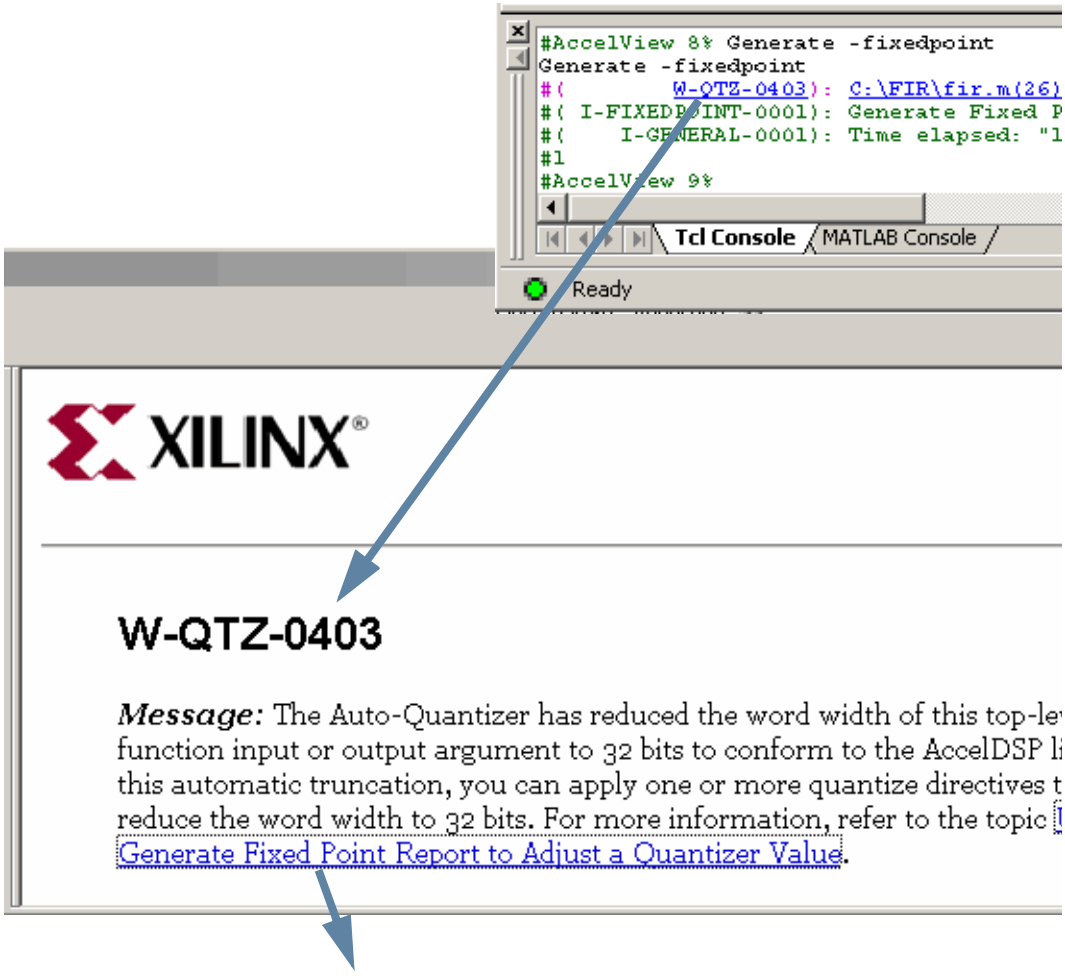
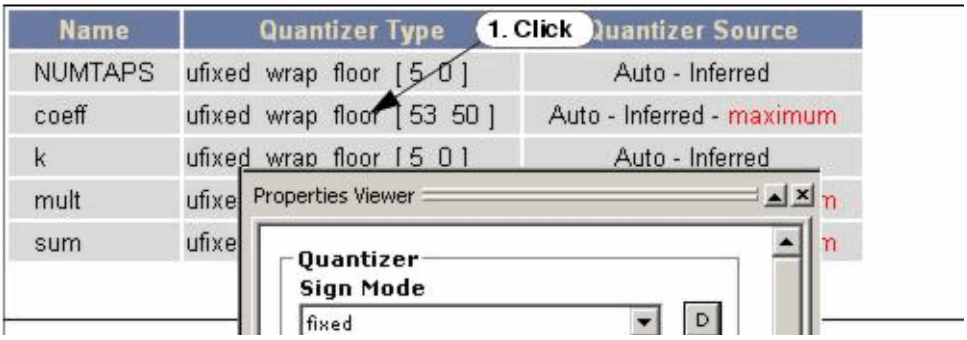


Figure 5-4. Applying a Quantize Directive from



Chapter 2

Installation

Installing AccelDSP Synthesis

Hardware Recommendations

Table 2-1: **Windows-Based Hardware Recommendations**

Recommendation	Notes
2.00 GB of RAM	
600 MB of hard disk space	Minimum Requirement
Xilinx® Hardware Co-Simulation Platform	Required for the Hardware Co-Simulation Flow

Operating System and Software Requirements

Table 2-2: **Windows-Based Operating System and Software Requirements**

Requirement	Notes
Windows XP 32 bit Operating System SP2	The AccelDSP synthesis tool does not support Windows Terminal Services.
Xilinx® ISE Design Suite Release 11.1	
MathWorks MATLAB® Version 2008a or 2008b	
MathWorks Simulink with Fixed-Point Toolbox Version 2008a or 2008b	Required for the System Generator Flow

Compatibility with Other Tools

AccelDSP Synthesis is designed to work with other software tools in an integrated flow. AccelDSP Synthesis is currently compatible with the following tools:

Table 2-3: **Compatibility with Other Tools**

Tool	Version
Mentor Graphics ModelSim® SE and PE	6.4b
Mentor Graphics Precision™ RTL Synthesis	2006a.101
Synopsys Synplify Pro®	8.9 (requires a floating license from Synplicity for AccelDSP integrated optimization)

Installing MATLAB from the MathWorks

MATLAB should be installed prior to installing the AccelDSP synthesis tool. The initial AccelDSP "System Information" screen will show the path and the version of MATLAB that will be used.

If the AccelDSP synthesis tool cannot access MATLAB, try the following:

1. Determine the path to the version of MATLAB you want to run from AccelDSP.
2. In a DOS shell, enter the following:

```
cd $MATLAB\bin\win32
matlab /regserver
```

Using the ISE Design Suite Installer

Before invoking the ISE Design Suite Installer, it is a good idea to make sure that all instances of MATLAB are closed. When all instances of MATLAB are closed, launch the installer and follow the directions on the screen.

Choose MATLAB Version for System Generator

As the last step of the System Generator installation, click the check box of the MATLAB installation you wish to associate with this verison of System Generator, then click **Apply**.

If you don't see a valid version of MATLAB listed, for example a version installed on a network device, click the **Add Version** button, browse to the MATLAB root directory of the unlisted version, then click **Add**. If you wish to associate this version of MATLAB with System Generator, click the check box of the newly listed MATLAB installation, then click **Apply**.

If you have no version of MATLAB available, click **Choose Later** to continue with the installation. At a later time, after you have installed MATLAB, you can associate that version of MATLAB with System Generator by executing the Windows menu item **Start** > **All Programs** > **Xilinx ISE Design Suite 11.1** > **DSP Tools** > **Select MATLAB version for Xilinx System Generator**.

Setting the Default AccelDSP Work Directory

The first time the AccelDSP synthesis tool is invoked after installation, the tool looks for a directory named AccelWork at the same level as the AccelDSP tree. If the directory AccelWork doesn't exist, the tool creates it. The tool then sets the environment variable named ACCELWORK to point to this directory.

If you do not have write permission to the directory containing the AccelDSP tree, the tool prompts you to enter a valid pathname for your default work directory. You may enter the pathname to any directory for which you have write permission.

Once set, the ACCELWORK environment variable will be used with future installations. You can change the ACCELWORK value by bringing up the System Environment page from the Help pulldowm menu, then click on the ACCELWORK Path.

Verifying the Setup of Other Tools

Before you begin working with AccelDSP Synthesis, you should verify the following for each supporting tool in the flow:

1. The tool software is installed according to the installation instructions provided.
2. The tool license file is installed.
3. The tool environment variable is set.

Table 2-4: **Environment variables for Other Tools**

Tool	Environment Variable	Set Value to
ModelSim SE	VSIMBIN	pathname ending in ...\ModelSim\win32
Precision RTL Synthesis	PRECISION	pathname ending in ...\PreSyn2004a75
Synplify Pro	SYNPLIFY	pathname ending in ...\Synplify

When you first invoke the AccelDSP synthesis tool after an installation, the System Environment report is activated to show you the current setting of relevant Environment Variables. As shown in Table 2-4, the pathnames in the report are activated so you can simply click on a pathname to change it.

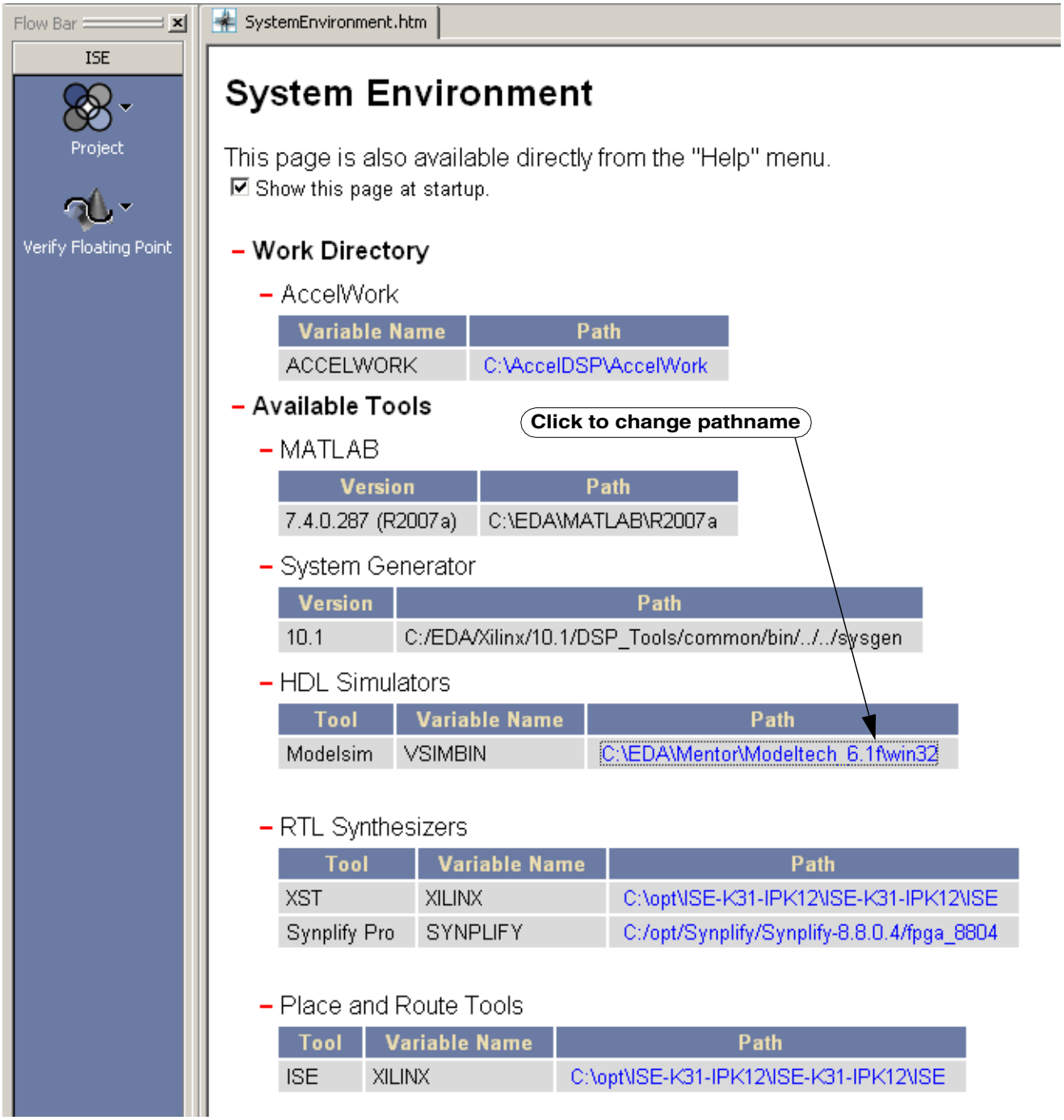
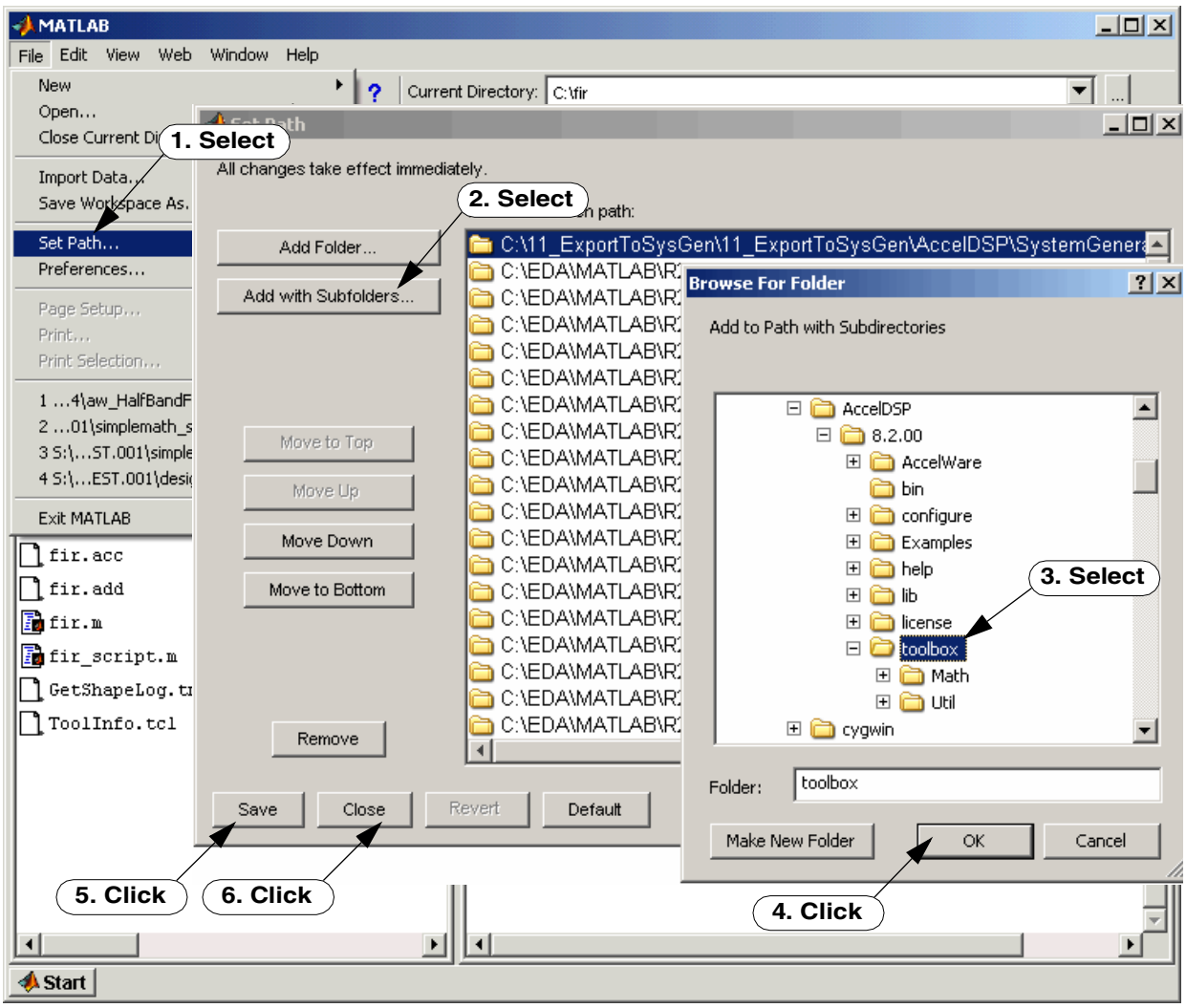


Figure 2-1: **The System Environment Report**

Adding AccelDSP MATLAB Functions to the MATLAB Search Path

AccelDSP provides a number of specialized functions that you may use in writing your MATLAB source code. Before you can use these functions, you must first add the AccelDSP **.../toolbox** folder (and subfolders) to the MATLAB search path. The folders should be added at a priority level that is higher than any of the MATLAB Tool Boxes. The procedure is illustrated below:



1. Invoke MATLAB, then select the pulldown menu **File > Set Path** as shown above
2. Select all references to the AccelDSP folder **.../lib/matlab** that may exist from a previous installation , then click **Remove**.
3. Click on **Add with Subfolders**, select the AccelDSP tree folder **toolbox**, then click **OK**
4. Click on Save, then Close the Set Path dialog box

Initializing AccelDSP Options on Startup

AccelDSP global options are set by executing (sourcing) a file named AccelDSPInit.tcl at startup. On startup, the AccelDSPInit.tcl file that is located in the AccelDSP *configure* directory is sourced first and the global options in this file are set.

If you choose, you can have an additional AccelDSPInit.tcl file in your $HOME directory. This file is loaded next, and if you have an AccelDSPInit.tcl file in you project directory, that file is loaded last. Global options set from the init file in your $HOME directory may overwrite global options set from the init file in the AccelDSP software tree. And, global options set from the AccelDSPInit.tcl file in your project directory may overwrite global options set from the init file in your $HOME directory.

The AccelDSPInit file contains SetGlobalOption commands. For example,

```
SetGlobalOption -targetlanguage VHDL
```

sets the target language to VHDL on startup.

Another interesting command is

```
SetGlobalOption -ignorefunction plot
```

This command tells AccelDSP Synthesis to ignore the MATLAB function **plot** during synthesis.

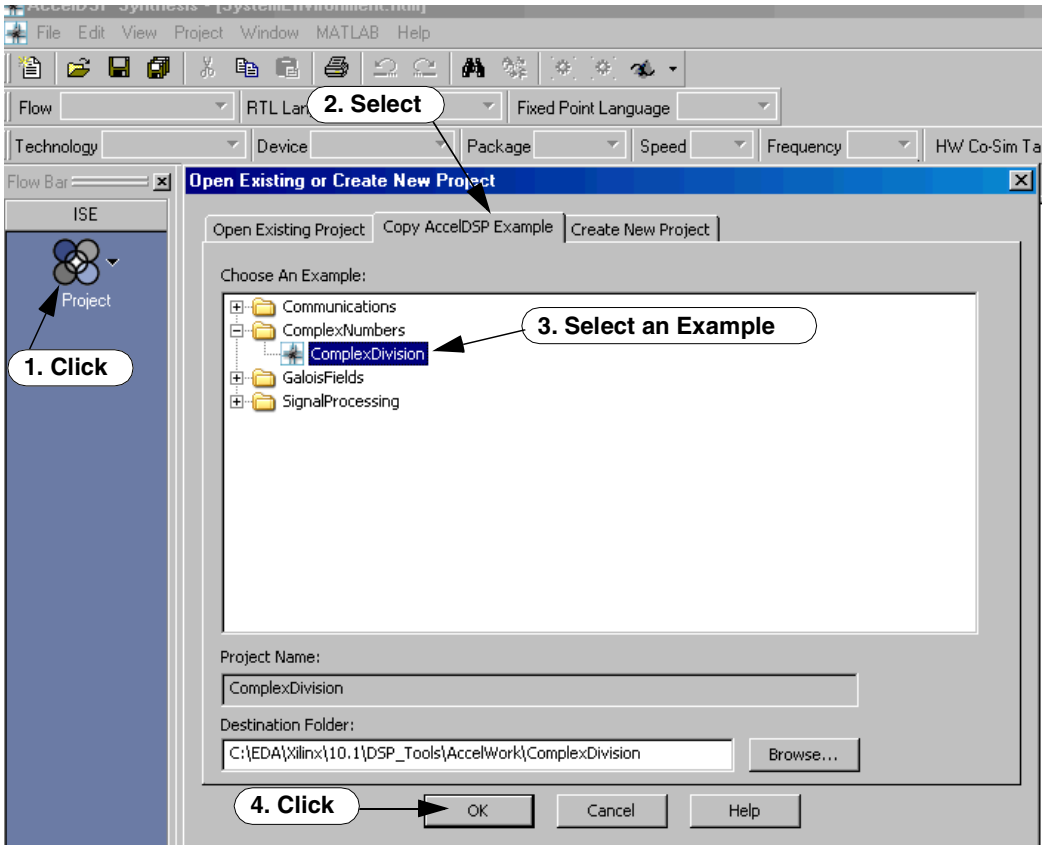
Installing and Verifying Hardware Co-Simulation Platforms

Hardware Co-Simulation platforms that are supported by System Generator are also supported by AccelDSP. For Hardware Co-Simulation platform installation instructions, refer to the System Generator Installation information in the System Generatoronline help.

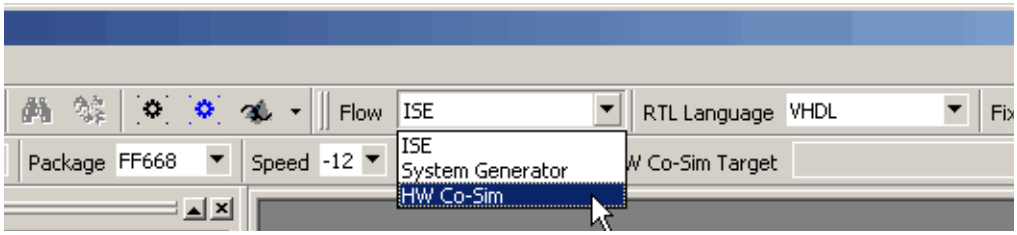
Run a Sample HW Co-Sim Simulation from AccelDSP

You can use the following procedure to verify that the AccelDSP HW Co-Sim Flow for your development board is operating properly.

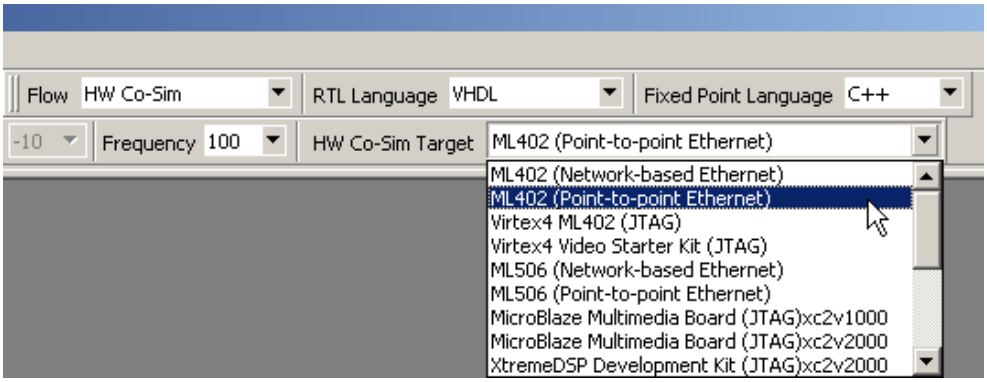
1. Install your Hardware Co-Simulation Platform as described in the System Generator Installation Instructions.
2. Invoke AccelDSP and Open a Project as shown below:



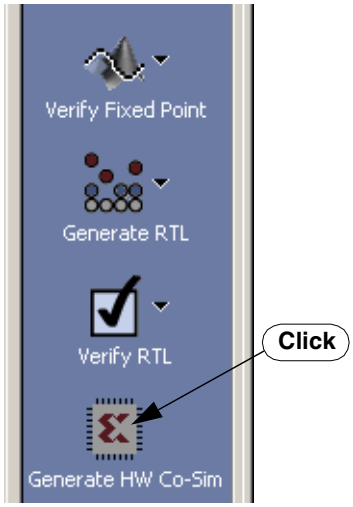
3. Set the HW Co-Sim **Flow** Options
   a. Set the HW Co-Sim Flow option as shown below:



b. Set the **HW Co-Sim Target** to your development platform as shown below:

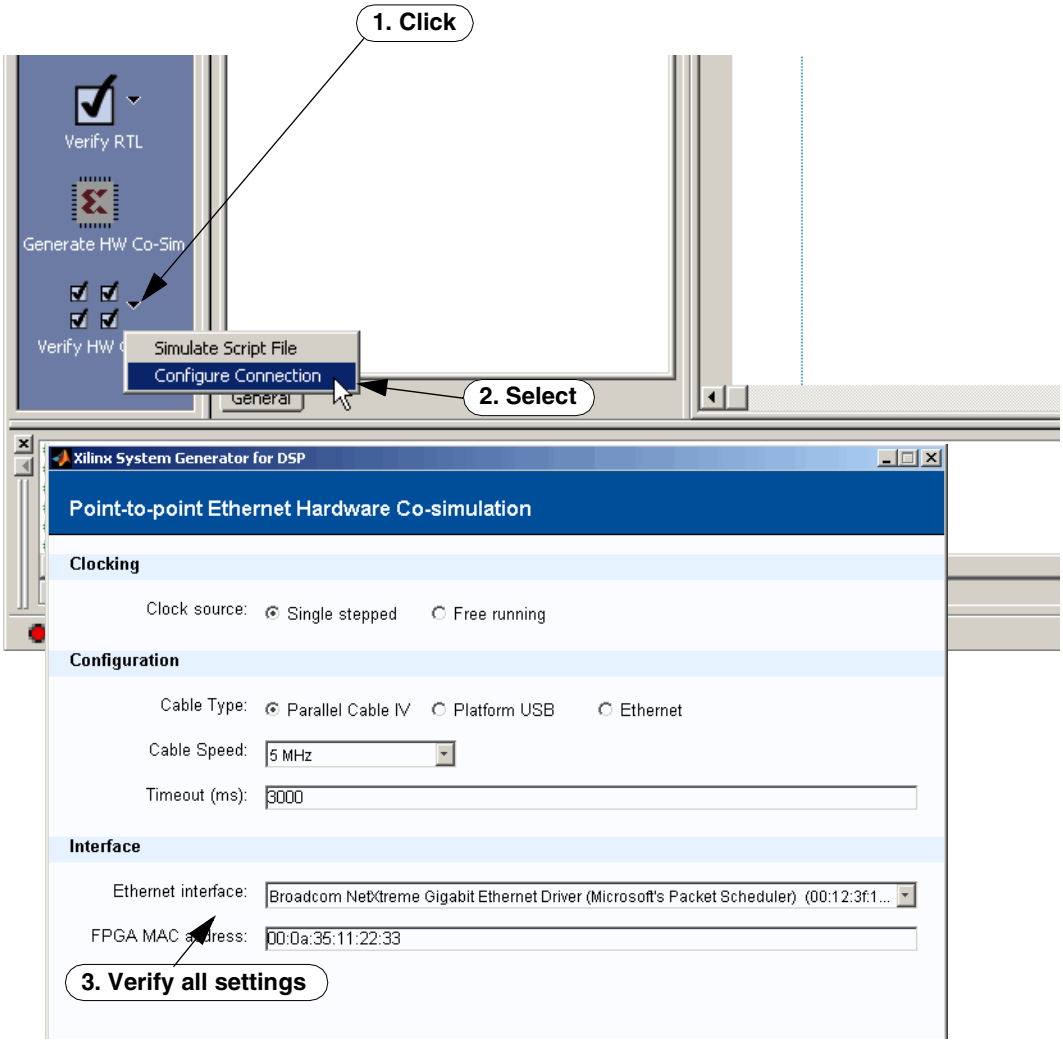


4. Step through the HW Co-Sim Flow through Verify RTL
5. Generate the HW Co-Sim Model by clicking on the icon shown below:

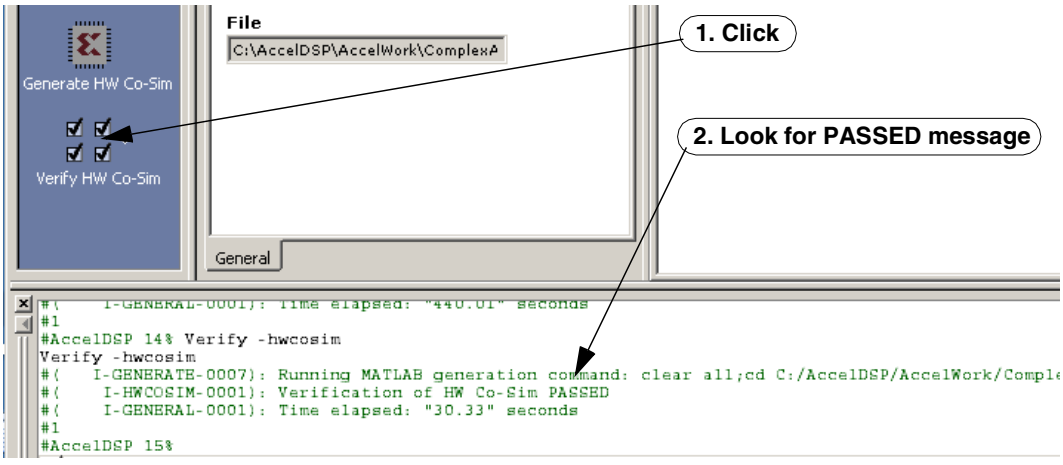


This generation process may take several minutes depending on the size and complexity of the design. A complete implementation flow is being executed including RTL synthesis, place and route, gate-level verification, and bit-stream generation. AccelDSP shows you the status of the flow in a Compilation status box.

6. As shown below, verify the Configuration Connection.



7. Click on the Verify HW Co-Sim flow icon



After the simulation is finished, the message "Verification of HW Co-Sim PASSED" should appear in the Tcl Console.

Chapter 3

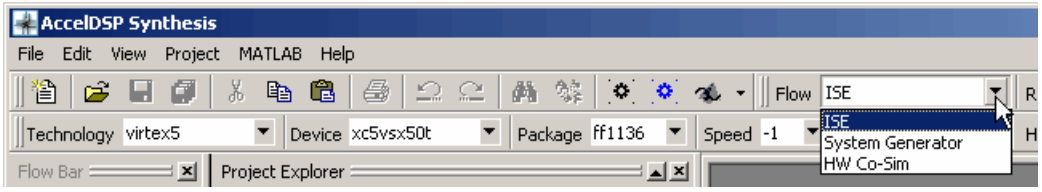
Understanding the Synthesis Flow

Introduction

This chapter explains the AccelDSP Synthesis Flows for transforming a MATLAB® floating-point model into a synthesized hardware module that can be implemented in silicon. The design example used is a general purpose FIR filter.

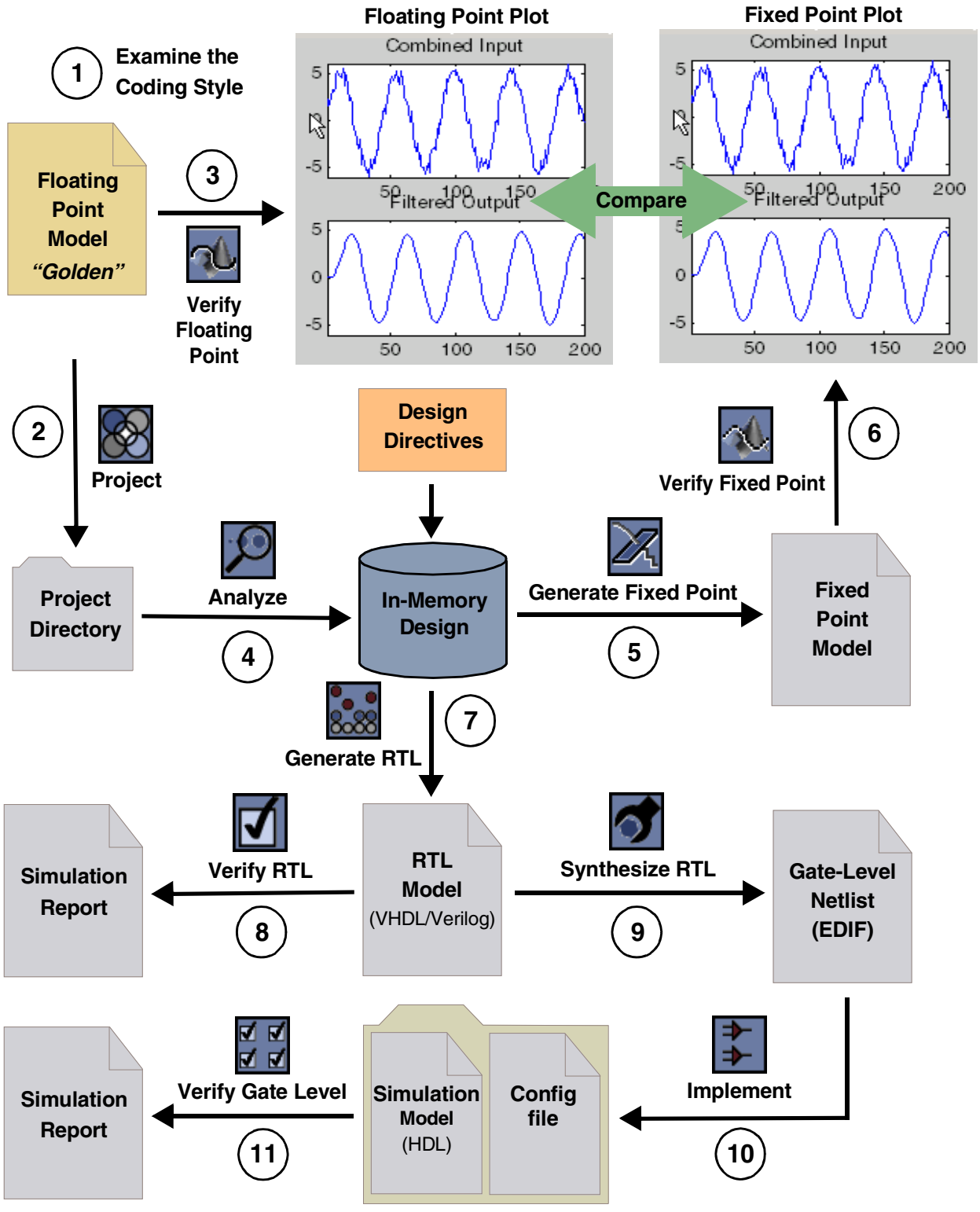
The default synthesis flow is called the **ISE** Synthesis Flow where the main objective is to create an implementation using ISE software and verify the design using HDL gate-level simulation. The second flow is called the **System Generator** flow. In this flow, the design is converted into a System Generator block that can be included in a larger System Generator design. More information on the System Generator flow can be found in the section titled The System Generator Synthesis Flow. The third flow, **HW Co-Sim,** is similar to the **ISE** flow but the objective is to simulate the design in hardware like a Virtex-4 on an ML402 Platform, a Virtex-5 on an ML506 Platform, or a Spartan-3A DSP 1800A Starter Platform. Not only does the simulation run much faster, but this flow proves that the design will run in the target hardware. Installation instructions for development boards that support Hardware Co-Simulation can be found in the System Generator online help under the topic Installation. An AccelDSP Hardware Co-Simulation verification procedure can be found under in the topic Installing and Verifying Hardware Co-Simulation Platforms.

As shown below, after you open a project, you specify the flow from the Flow dialog box.



The topic AccelDSP Getting Started Tutorial is a hands-on tutorial exercise that will guide you step-by-step through the **ISE** flow, provided that you have AccelDSP Synthesis and MATLAB installed on your computer.

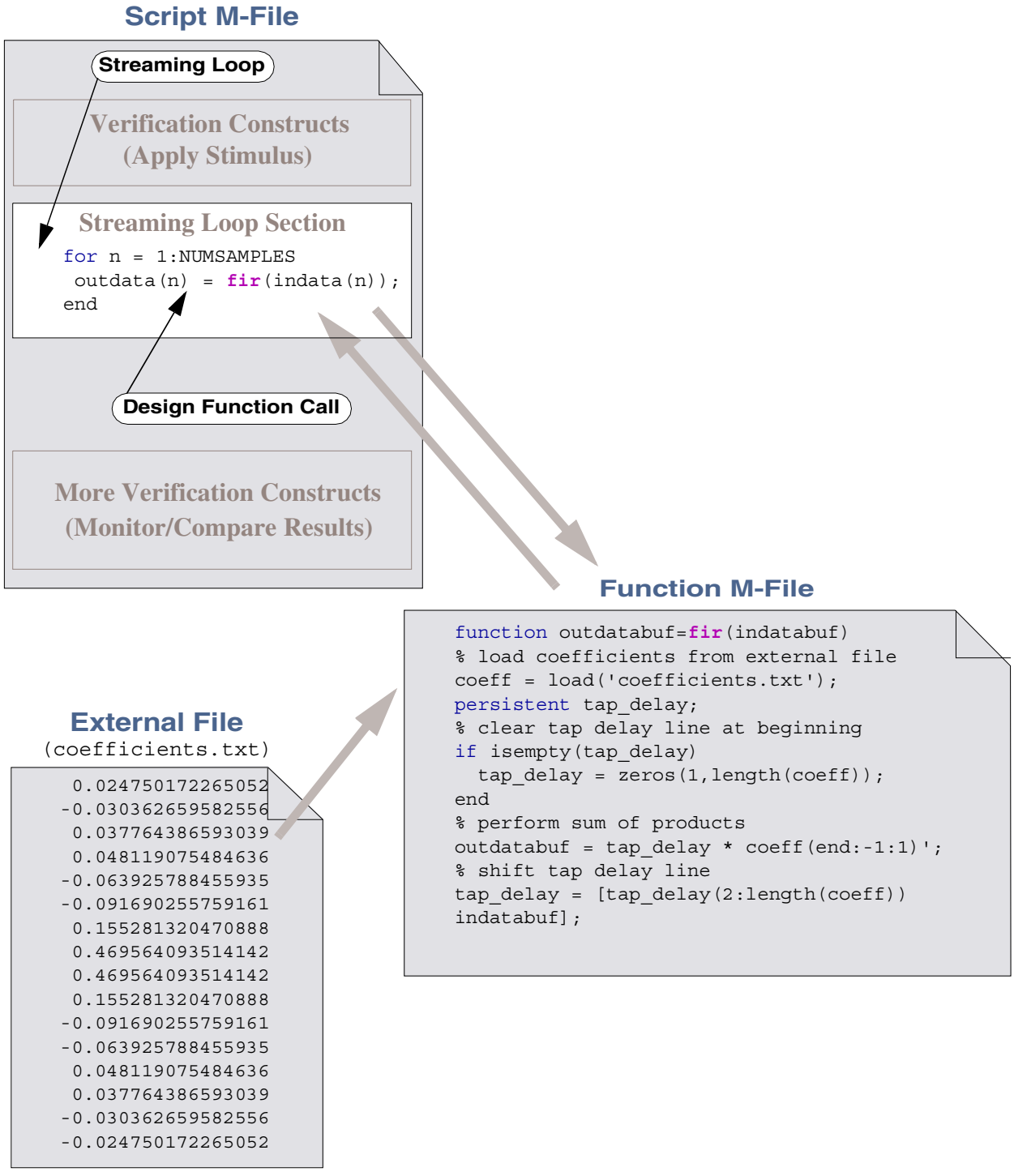
The AccelDSP ISE Synthesis Flow



As shown in the previous illustration, you synthesize a MATLAB floating-point model into a hardware module using the following basic steps in the AccelDSP ISE Synthesis Flow.

1. **Examine the Coding Style of the Floating-Point Model**. You should first verify that the MATLAB design conforms to minimum AccelDSP style guidelines that are explained in the manual titled MATLAB for Synthesis Style Guide.
2. **Create an AccelDSP Project**. You invoke AccelDSP Synthesis, then click **Project** and specify the name of a new Project file. The Project file is placed in the Project Directory where all future AccelDSP-generated files are saved.
3. **Verify the Floating-Point Model**. You should have verification constructs in your MATLAB script file to apply stimulus and plot results. This output plot is the "golden" reference for comparing future results. If you have verified the floating-point model outside of AccelDSP Synthesis, you may skip this step.
4. **Analyze the Floating-Point Model**. This step creates an in-memory model of your design. In a later pass through this flow, you can add design directives that guide AccelDSP toward finding the best hardware architecture for your design.
5. **Generate a Fixed-Point Model**. This step generates a fixed-point model of the design, then places the design files in a newly generated project sub-folder named MATLAB.
6. **Verify the Fixed-Point Model**. When you click **Verify Fixed Point**, AccelDSP automatically runs a MATLAB fixed-point simulation. You then visually compare the Fixed-Point Plot with the Floating-Point Plot to verify a match.
7. **Generate an RTL Model**. This step generates an RTL Model from the in-memory design data base. The RTL model can be generated in VHDL or Verilog format.
8. **Verify the RTL Model with your HDL Simulator**.You click Verify RTL to run your HDL simulator on the generated Testbench. PASSED or FAILED will be indicated in a simulation report.
9. **Synthesize the RTL Design into a Gate-Level Netlist**. This step invokes a pre-specified RTL synthesis tool on your design. The generated gate-level netlist is ready for place and route using the ISE implementation tools.
10. **Implement the Gate-Level Netlist**. You click on **Implement** to invoke the ISE implementation tools to place and route the design. The generated files of interest are a gate-level HDL simulation file and a configuration file containing the bitstream for configuring the FPGA hardware.
11. **Verify the Gate Level Design**.This step uses the AccelDSP Testbench to run a bit-true simulation check on the gate-level HDL simulation model. A PASSED indication means that the implemented design is bit-true with the original fixed-point MATLAB design.

Examining the Floating-Point Model



AccelDSP requires that your floating-point model conform to minimum guidelines for coding style. The previous illustration shows the file structure of a general purpose FIR filter. This floating-point model is represented by a required Script M-file and a Function M-file. Other External Files and functions may be called from the main Script M-file and Function M-file.

Script M-File

The script M-file on the left has three main sections. The top section contains verification constructs that apply stimulus to the streaming loop section (middle). The streaming loop section contains the design function to be synthesized. Additional verification constructs (bottom) monitor and report on the MATLAB® simulation results.

The Streaming Loop

For hardware synthesis, the infinite streams of data entering and leaving the design must be partitioned into manageable groups or "slices" in order to properly process the data. A streaming loop (a for loop or while loop) is the construct that performs this task. The MATLAB script file must have a streaming loop. During analysis, AccelDSP Synthesis attempts to auto-identify the streaming loop. If it can not, the tool prompts you to manually identify the streaming loop in the GUI.

The Top-Level Design Function Call

The *top-level design function* call represents the hardware to be synthesized. This function must reside inside the streaming loop. The AccelDSP synthesis tool attempts to automatically identify the top-level design function. If it can not, the tool prompts you to manually identify the design function call in the AccelDSP Project Explorer window.

Function M-File

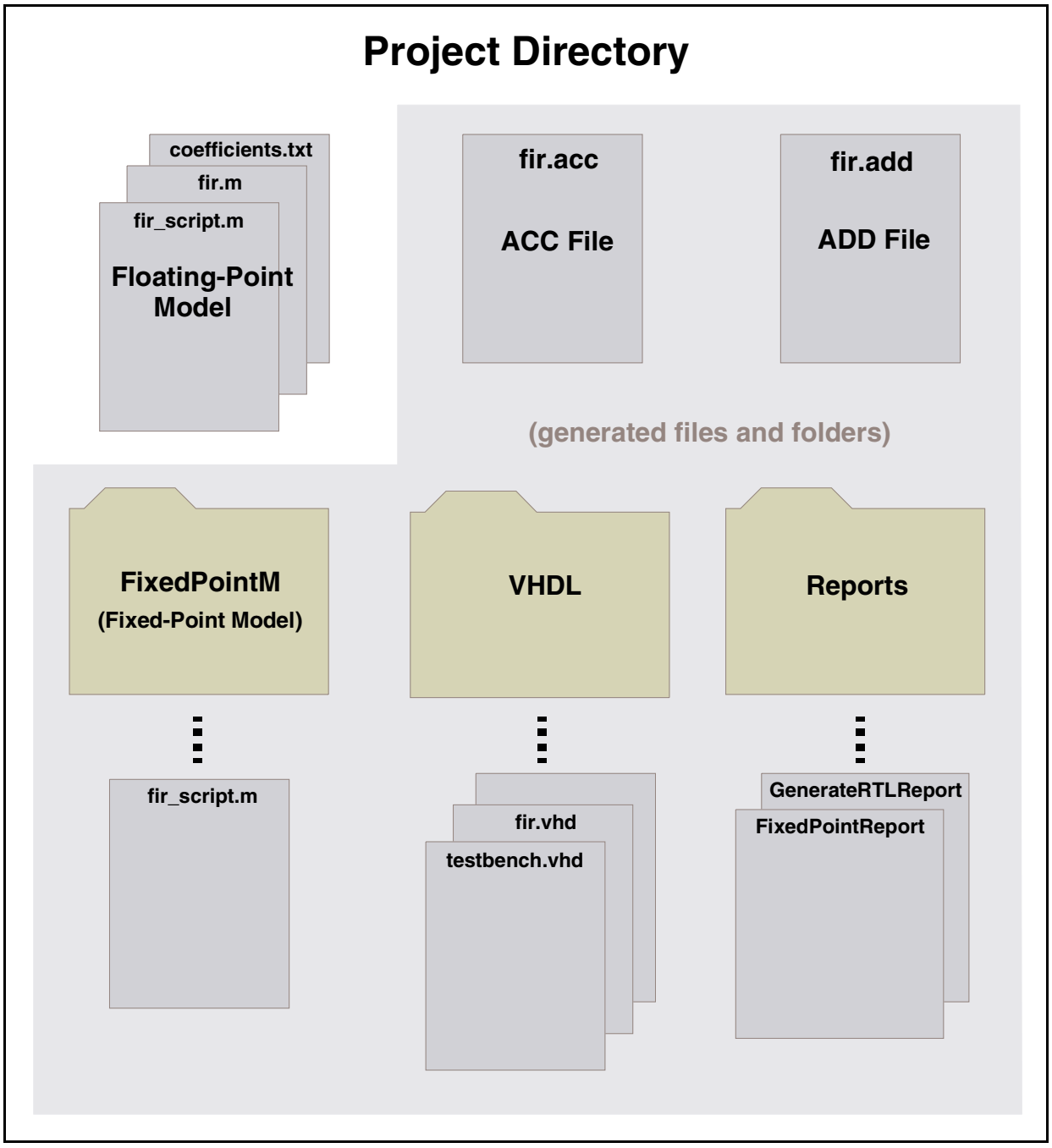
The body of the design function M-file models the main processing algorithm. This code is synthesized into the hardware block. Any function calls from this top-level M-file to other function M-files are synthesized into sub-blocks of hardware.

Input and Output Data Types

Each input and output to the top-level design function must be a variable that represents a **scalar**, a **row vector**, or a **column vector**. *All other argument types are not supported.* For example, array elements a(i), sub-expressions (a+b), and function calls like (abs(x)*2) are not allowed in the top-level design function argument list.

Creating an AccelDSP Project

Project



Before you synthesize a MATLAB design, you must first create an AccelDSP Project. After you invoke AccelDSP Synthesis, you click on the Project icon, navigate to a directory you wish to designate as your project directory and specify the name of your project file. AccelDSP creates a project file (ACC file) by the specified name and places the file in the directory. The project file keeps track of project details like the pathnames to MATLAB design files and project options that you may have set. There can be only one ACC file for each project.

As shown in the previous illustration, the project directory is the place where AccelDSP saves all generated files. The design source files should also reside in this directory. Several folders and files are generated as output during a synthesis run. Some of these files and folders are shown in the previous illustration

ADD (AccelDSP Design Directives) file After the Analyze step, you can manually apply design directives to in-memory design objects to help guide AccelDSP during the synthesis process. These design directives are automatically saved to the ADD file. When you restore the in-memory design model during a possible future synthesis run, the saved design directives are automatically re-applied to the newly-created in-memory data model.

FixedPointM (fixed-point model) This folder contains the generated fixed-point model from the Analyze step. The fixed-point model differs from the original model in that each floating-point number is "quantized" to a fixed-point equivalent. Some mathematical precision is lost, but the design is more suitable for implementing in hardware. After the model is created, you can change quantizer parameters on individual variables, if you choose, in order to increase or decrease the design fidelity.

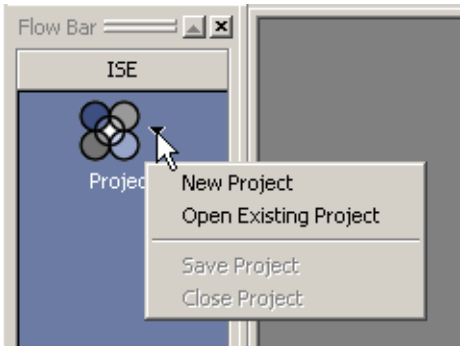
VHDL Contains one or more VHDL files that represent the RTL model of your design.

Verilog Contains one or more Verilog files that represent the RTL model of your design.

Testbench This subfolder is generated for each HDL language that you specify. The folder contains the Testbench that is generated to test and verify the RTL model.

Reports Contains one or more report files that are generated during a synthesis run.

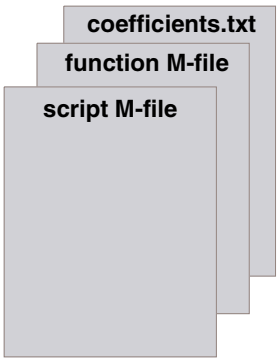
When you invoke AccelDSP Synthesis in the future, you can open an existing project or create a new project by selecting from the icon menu as shown below:

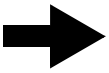


Verifying the Floating-Point Mode

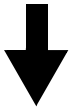
Verify Floating Point

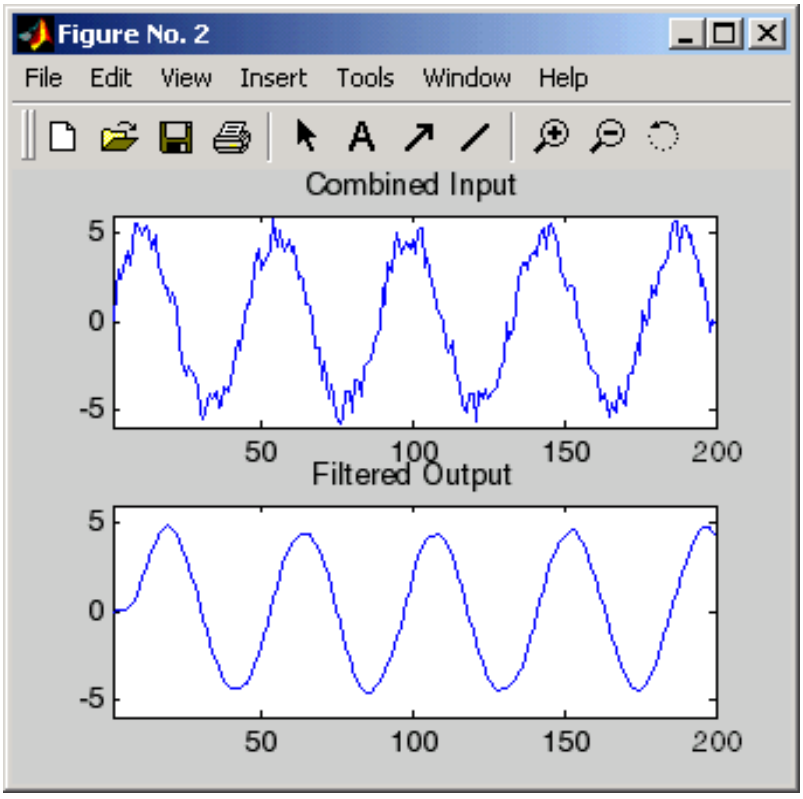
Floating Point Model





(automatically invoked)

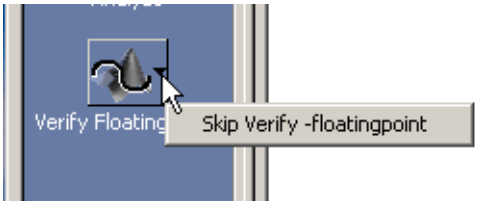




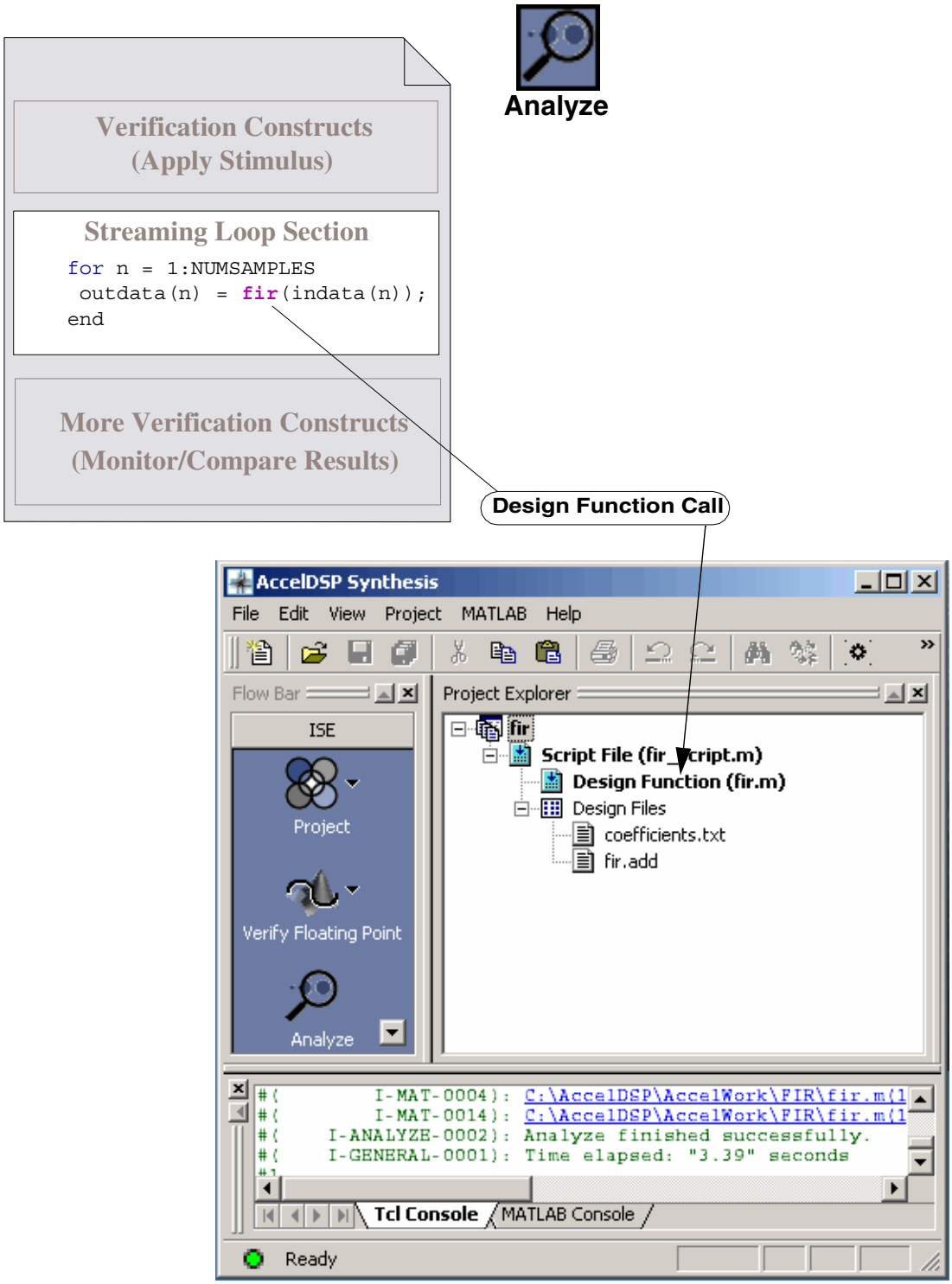
Typically, you will run many MATLAB simulations as you refine the design of your floating-point model. After you invoke AccelDSP Synthesis and open a project, you may run a floating point MATLAB simulation at any time by clicking on the Verify Floating Point icon. In this case, the MATLAB simulation is run and the verification constructs at the end of the script M-file plot the results.

The results of the floating-point simulation are the "golden" results and are used as a basis for comparison in future simulation runs. It is a good practice to save a copy of this plot as a MATLAB FIG file. You can then restore this plot and compare it to the plot that will be generated from an equivalent fixed-point model.

If you have already verified the floating-point model outside of AccelDSP, you may choose to skip this step by selecting from the icon menu, as shown below:



Analyzing the Floating-Point Model



When you click the Analyze icon, the MATLAB® floating-point model is analyzed, and then AccelDSP Synthesis creates an equivalent in-memory data model of your design. Changes that you make to the design from this point are made to the in-memory design, not the "golden" MATLAB source. As shown in the previous illustration, the structure of the in-memory data model is graphically displayed in the Project Explorer window.

Identifying the Streaming Loop and the Top-Level Design Function

If AccelDSP has difficulty identifying the streaming loop during analysis, you will be prompted to select the streaming loop. And if AccelDSP has difficulty identifying the top-level design function, you will be prompted by the GUI to select the top-level design function in the window.

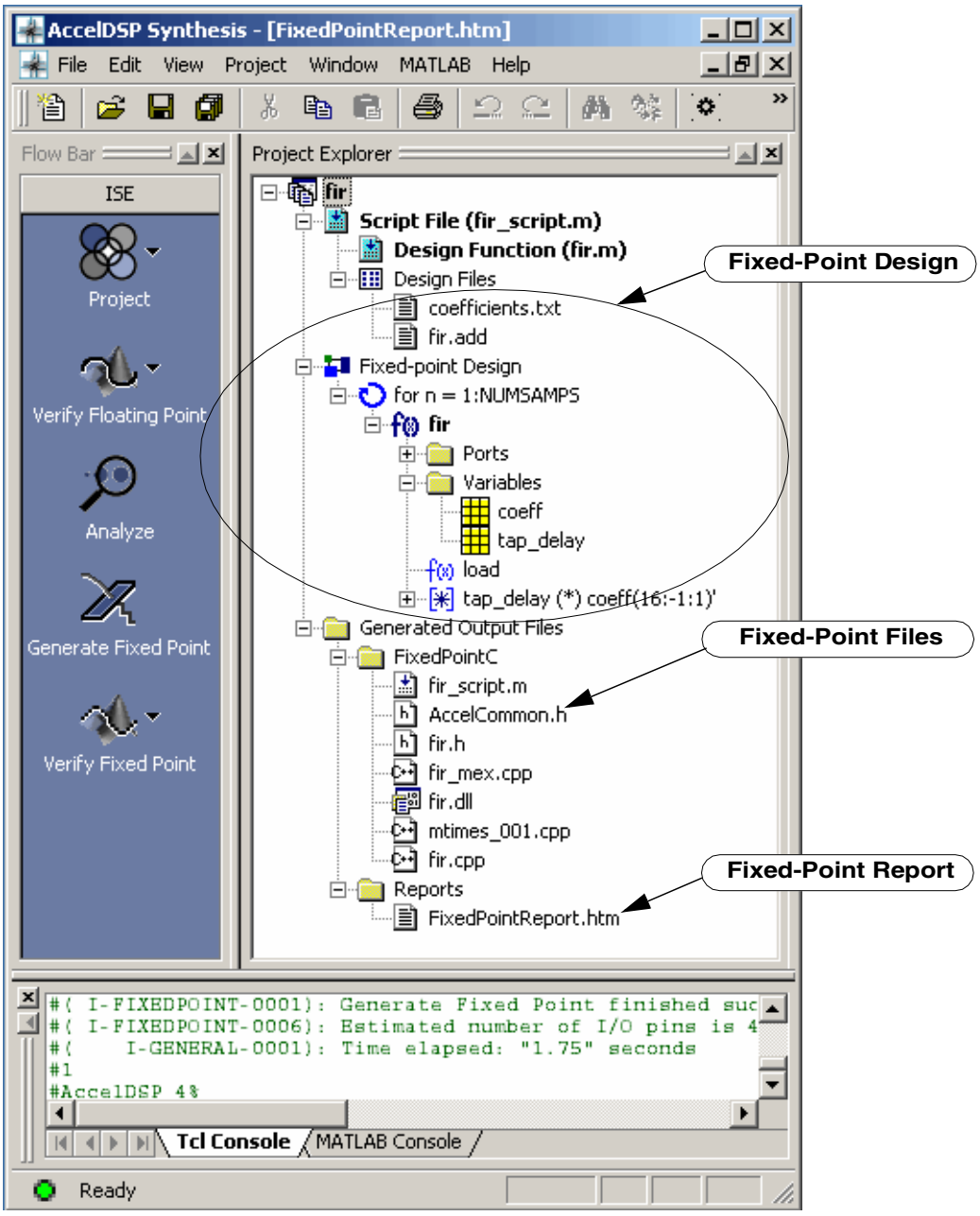
NOTE: If you give the Project File the same name as the top-level design function, AccelDSP Synthesis will always automatically identify the top-level design function.

Analyzing the Shape of Variables

Unlike many other programming languages, the MATLAB programming environment is designed to be an interpretive environment where the type and shape of each variable is determined at run time. AccelDSP Synthesis also attempts to determine the shape of each variable during the **Analyze** step. If it cannot, the AccelDSP GUI will display a form and prompt you to specify the shape. The shape of an array variable is specified as [<rows> <columns>]. More information on specifying the shape is located in the next topic.

Generating the Fixed-Point Mode

Generate Fixed Point



Choosing the Fixed Point Language

By default, AccelDSP Synthesis generates the fixed-point design in C++ because, in most cases, the C++ simulates faster. If you prefer, you may choose MATLAB as the fixed-point language.

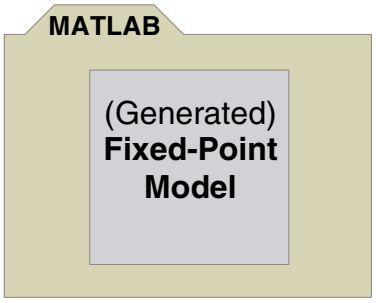
Generating a Fixed Point Model

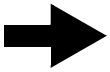
You click on the **Generate Fixed Point** icon to generate an equivalent fixed-point model in either C++ or MATLAB. As shown on the left, AccelDSP Synthesis writes the fixed-point C++ file(s) to a newly-created FixedPoint C folder in the project directory. In the next step, these files will be used in a C++ simulation to verify the fidelity of the fixed-point model.

AccelDSP Synthesis also displays a Generate Fixed Point report (not shown) to provide important information about the generated design.

Verifying the Fixed-Point Model

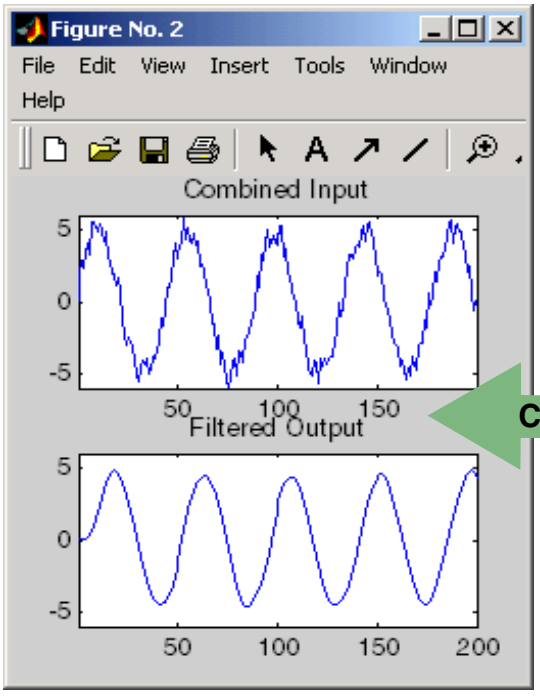
Verify Fixed Point





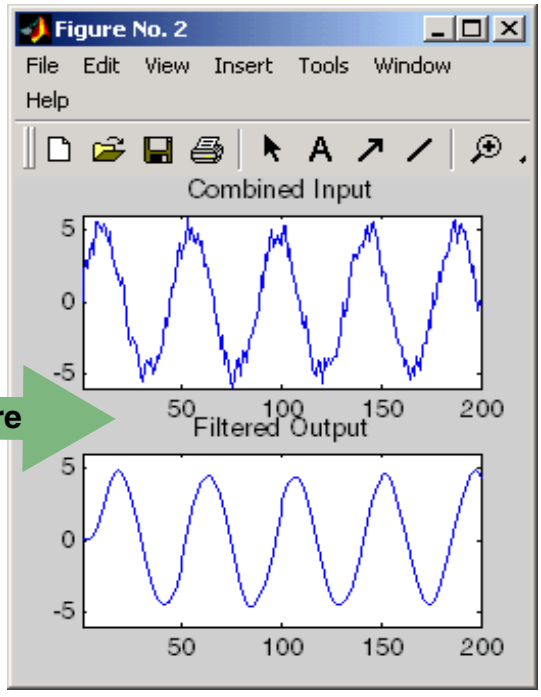
(automatically invoked)

Floating-Point Plot



Compare

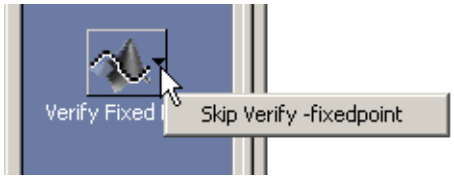
Fixed-Point Plot



Running a MATLAB Simulation

AccelDSP makes it easy to run a MATLAB simulation on the generated fixed-point model. You click on the **Verify Fixed-Point** icon and the MATLAB simulation is automatically run. As you can see in the previous illustration, the fixed-point plot does not exactly match the floating-point plot, but the results are acceptable.

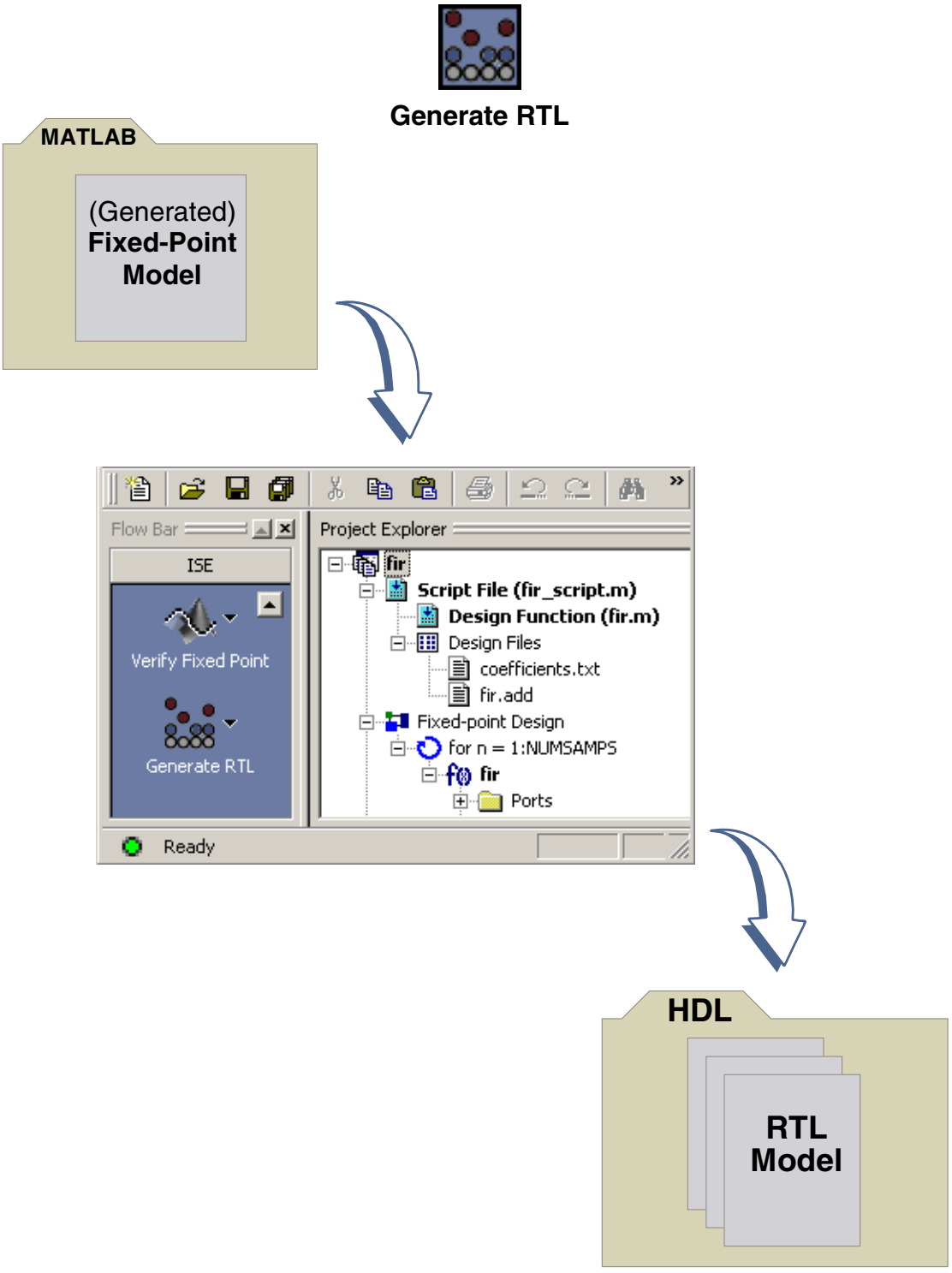
During the fixed-point simulation run, the data streams on the design inputs and outputs are captured and are used as the “golden I/O data” for future 'bit-true' comparisons in later steps. This is why it is important to run this step at least once and every time you change the fixed or floating-point models. If you have already run this step and captured the valid fixed-point input and output data streams, you may elect to skip this step by selecting from the Verify Fixed Point icon menu, as shown below:



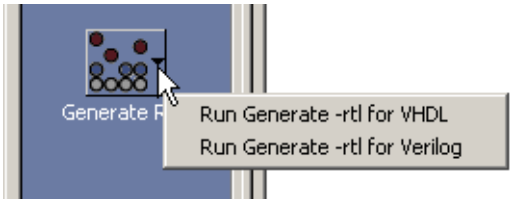
Running a C++ Simulation

If you generated the fixed-point model in C++, you simply click on the **Generate Fixed Point** icon to run the C++ simulation. Using the M2C-Accelerator, simulation run-time performance has been seen to increase by 150X when running mixed MATLAB/Simulink designs and up to 1000X for some designs.

Generating the RTL Model



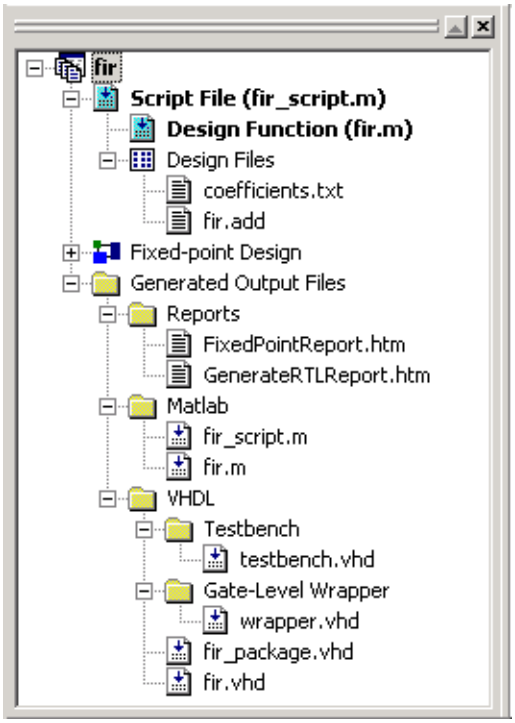
When you click on the **Generate RTL** icon, AccelDSP generates an RTL model of the in-memory design. The RTL (register-transfer-level) model can be generated in VHDL format or Verilog format. If you have not previously selected a language format, you may at this time by choosing from the Generate RTL icon menu, as shown below:



The files for the RTL model are saved to a newly-generated VHDL or Verilog folder in the Project Directory. You can examine the files from the Project Explorer window.

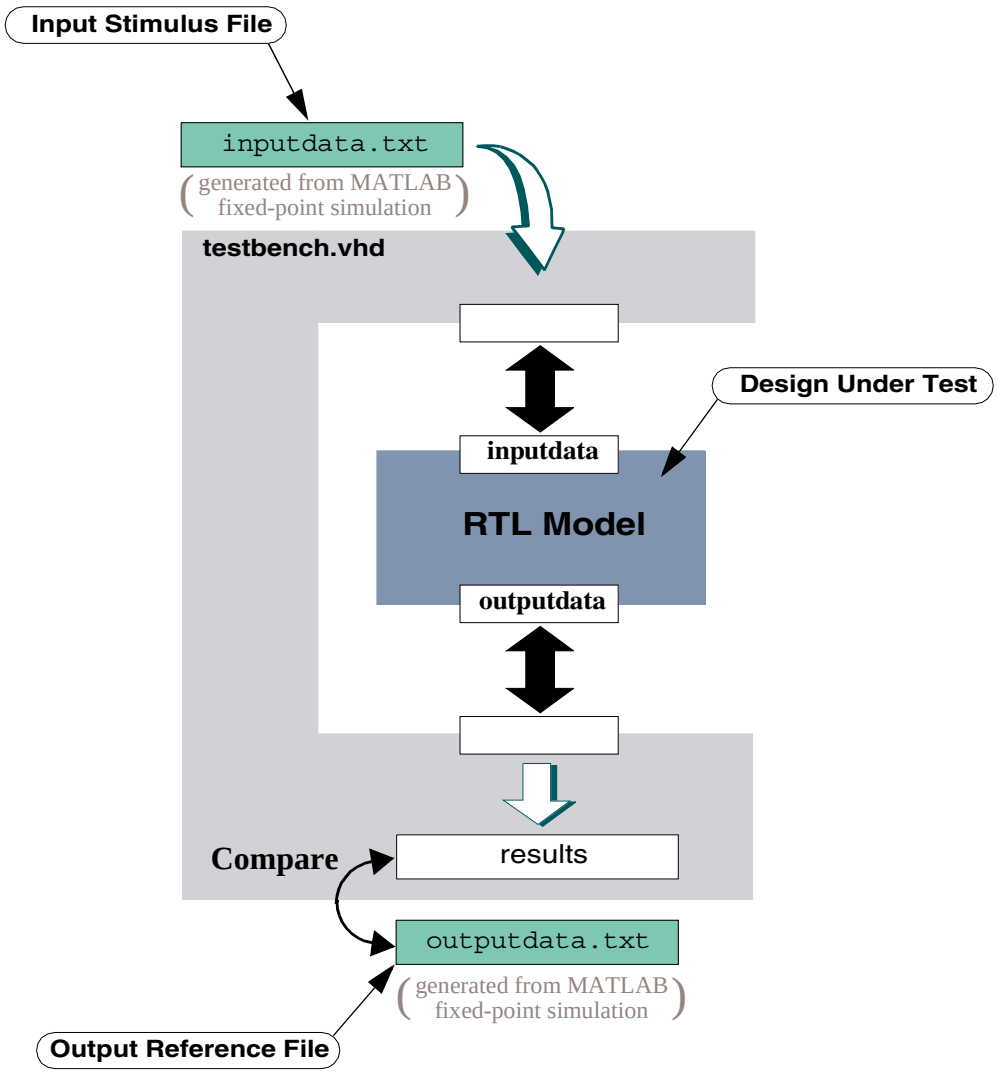
The Testbench

After synthesizing the RTL model, AccelDSP *generates* a Testbench that behaves similar to the verification constructs in the original MATLAB script M-file.The Testbench applies input stimulus to the design, then monitors and compares the output results. The *testbench* file is written in the same language as the RTL model (either VHDL or Verilog) and is saved to a newly-created Testbench folder under the VHDL or Verilog folder. The following illustration shows a typical file structure for the output of the Generate RTL step.

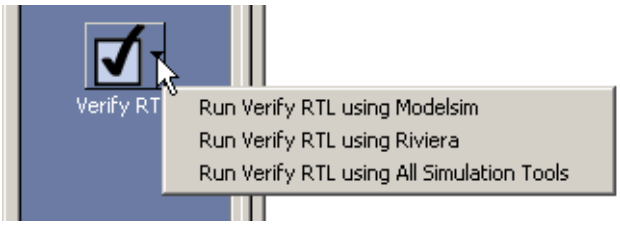


Verifying the RTL Model

Verify RTL

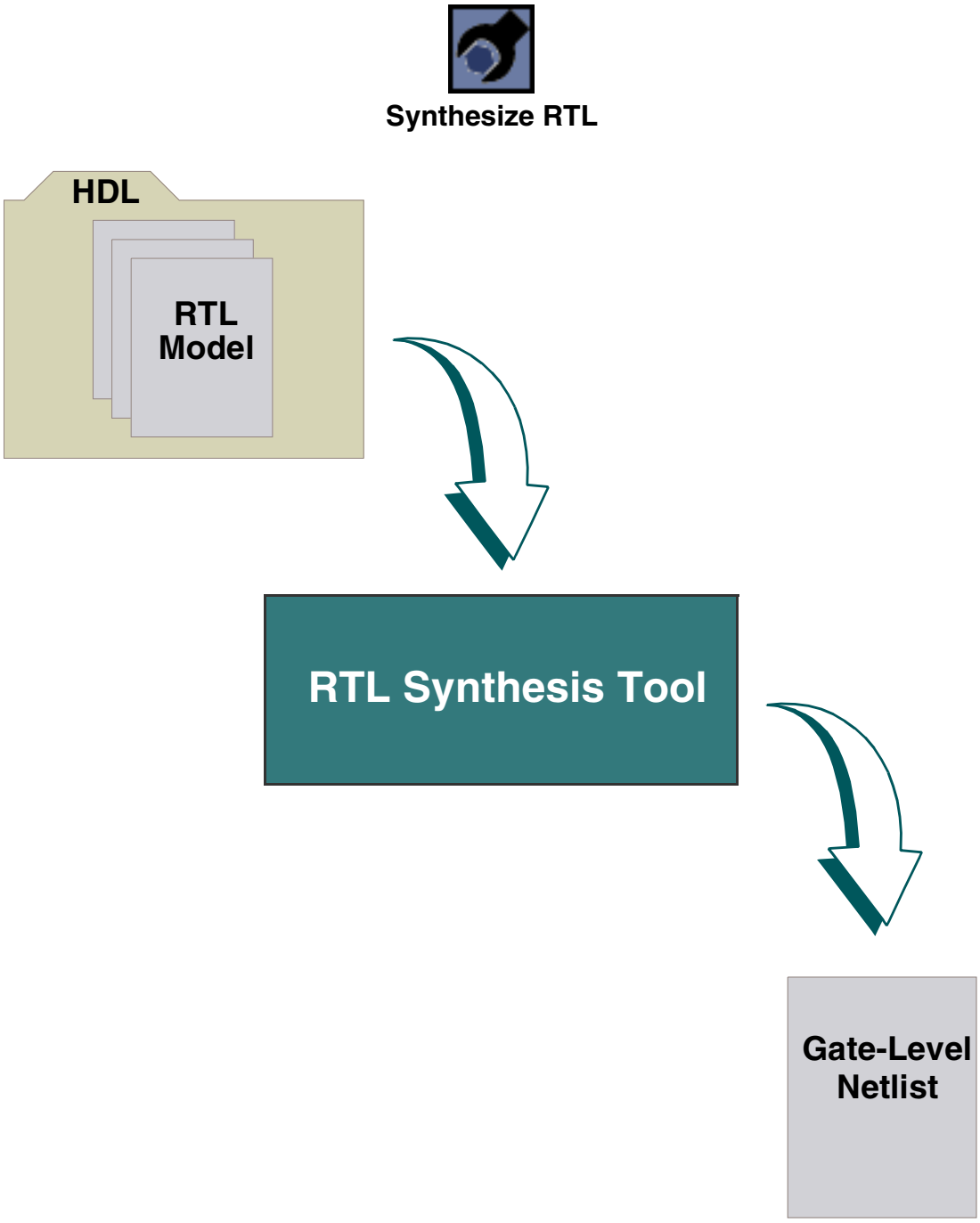


The previous illustration shows a conceptual model of a Testbench applying stimulus to the generated RTL Model (commonly called the Design Under Test). Stimulus is applied from an input data file that was previously generated during the MATLAB fixed-point simulation. The results of RTL simulation are compared to the content of a reference file containing the results of the MATLAB fixed-point simulation. If the RTL results match the fixed-point results, then the simulation passes. If you haven't previously selected the RTL simulation tool for this step, you may do so by choosing from the icon menu, as shown below:

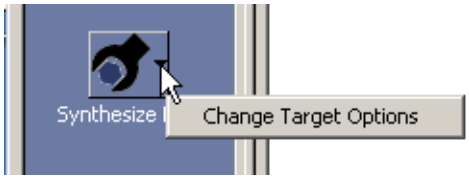


The Testbench is a general purpose device that is built from HDL modules located in an AccelDSP Synthesis library. The Testbench has a control panel that guides the simulation with default parameter settings. You can change the control panel settings from the AccelDSP GUI if you like. Since the Testbench is general purpose in nature, it can also be used down-stream to verify the gate-level HDL model from the RTL Synthesis tool, as well as the output from the ISE place and route implementation step.

Synthesizing the RTL Model



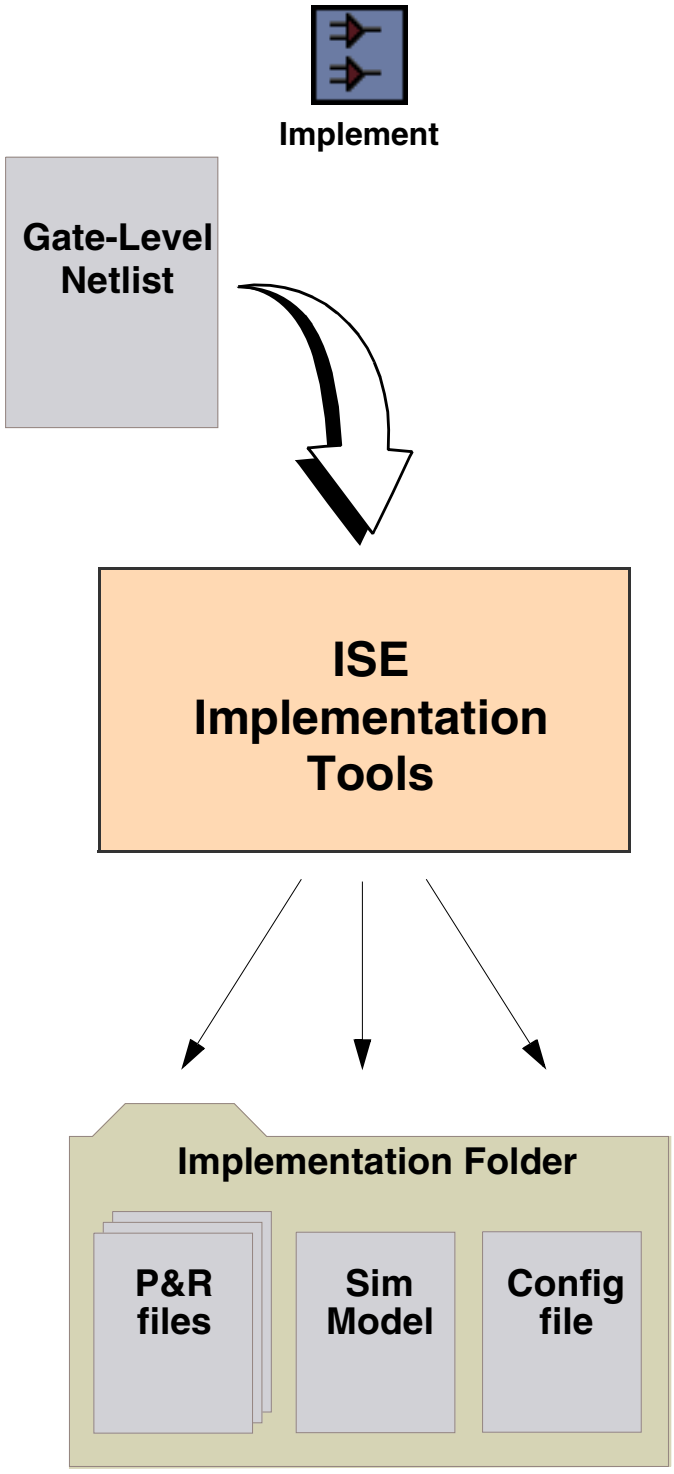
This step in the AccelDSP ISE Synthesis Flow uses a pre-specified external RTL Synthesis tool to transform the RTL Model into a gate-level netlist. The RTL Synthesis tool may be part of the implementation tool set, like XST from Xilinx, or may be a product offering by an independent EDA company like Synplify Pro from Synplicity. If you have not previously specified the synthesis tool or you wish to change the target technology options, you can do so by selecting from the Synthesis RTL icon menu, as shown below:



You can setup the AccelDSP GUI to automatically invoke the RTL Synthesis tool when you click on the **Synthesize RTL** icon in the Flow Bar.

From this point, you are ready to use the ISE implementation tools to place and route the design, use the generated simulation model to verify the implementation, and finally download the configuration bitstream to the FPGA hardware.

Implementing the Design



AccelDSP uses the ISE implementation tools to place and route the gate-level netlist for an FPGA. If you have not previously specified which implementation tools to use, AccelDSP will prompt you to specify the tools before it begins this step.

An implementation folder is created in the project directory and all of the generated output files are placed in this folder. Many files are created and placed in the folder, however, the following files are of particular interest.

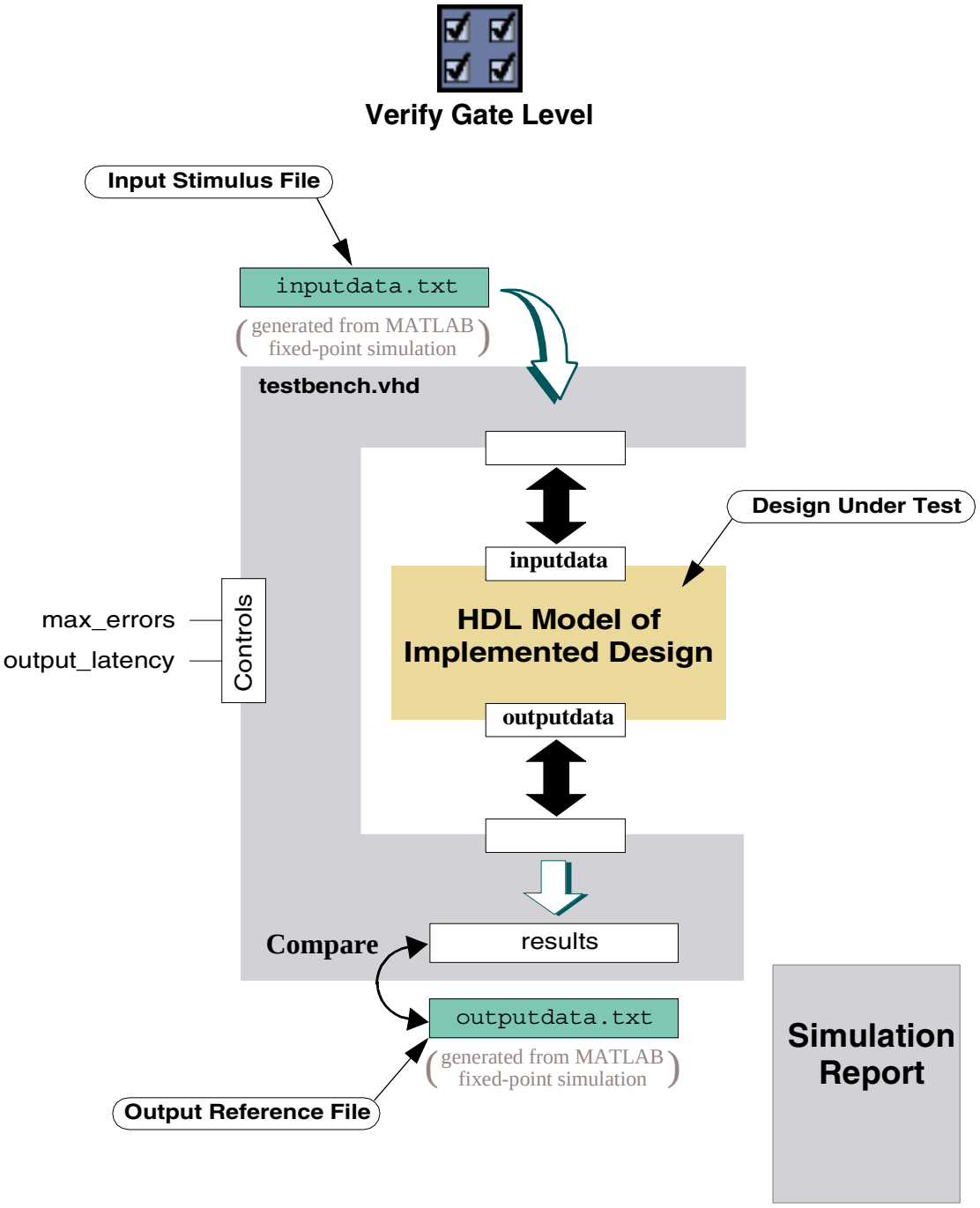
Gate-Level Simulation File

After the design finishes place and route, the implementation tools generate a gate-level simulation file that matches the behavior and timing of the implemented design. This file will be used in the Verify Gate Level step. Although this file may be written in VHDL or Verilog, it is for simulation purposes and cannot be synthesized into hardware.

Configuration File

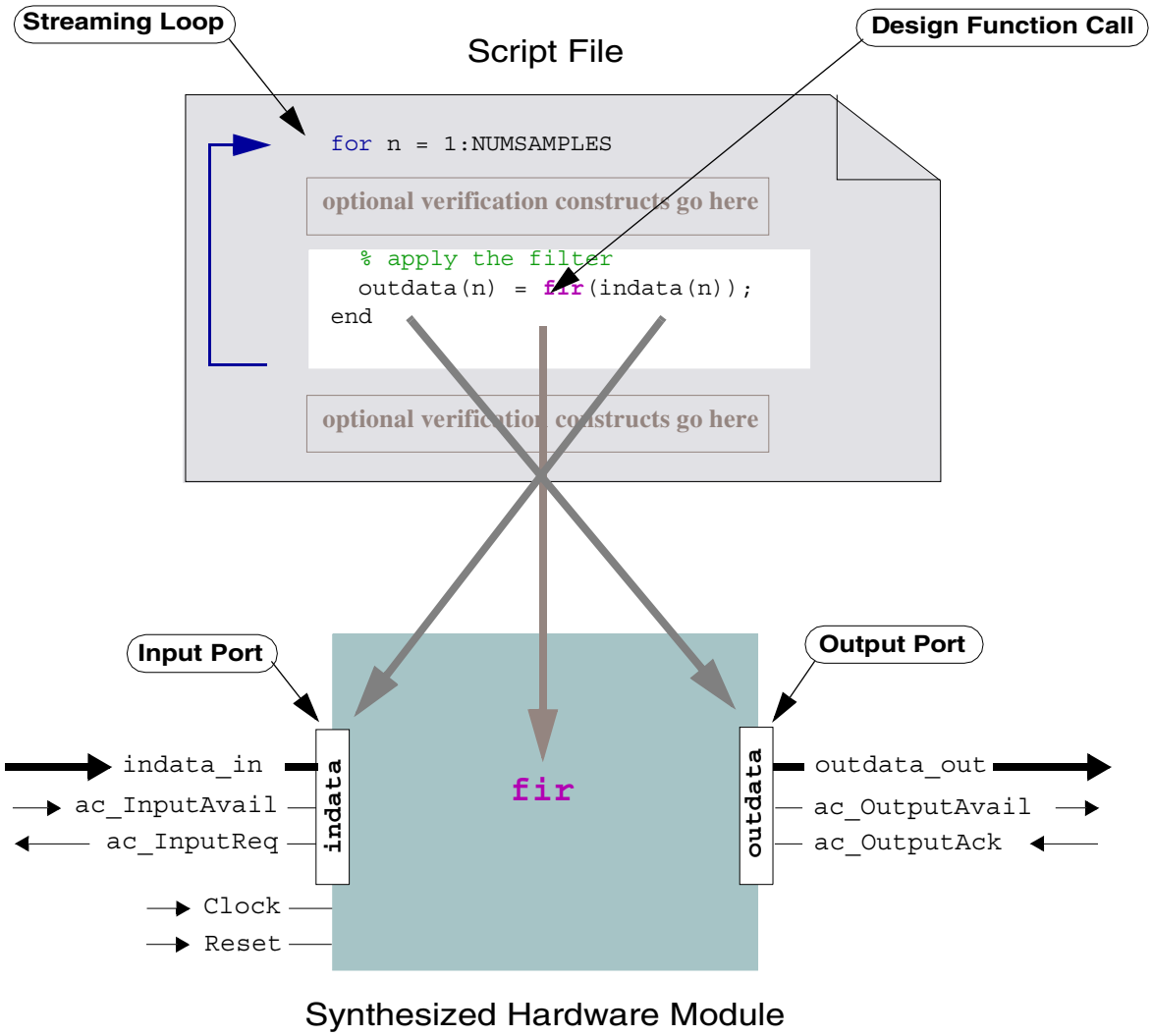
A Configuration file is a file that contains the bitstream that configures the FPGA hardware. You must set a project option to turn on the generation of this file.

Verify the Gate-Level Design



After place and route is finished, the implementation tools generate a gate-level simulation file that matches the behavior and timing of the implemented design. As shown in the previous illustration, this file is used in the Verify Gate Level step. The same input stimulus files that were used in the Verify Fixed Point step are also used here. If the simulation passes, it guarantees that the implemented design is bit-true with the original fixed-point MATLAB design.

How the Design is Mapped to Hardware



NOTE: You can generate handshake signals for each data port by setting the project option **Enable Handshake For Each I/O Port** to 'True'.

The previous illustration shows how the top-level design function call in the script file is synthesized into hardware. The name of the hardware block matches the name of the *design function* call. The names of the input and output ports are derived from the MATLAB variables used to move data into and out of the design function.

The Handshake Interface

A MATLAB design that is synthesized by AccelDSP will typically be a design module that is part of a larger design on the FPGA. The flow of data into and out of the hardware ports is controlled by a handshake interface protocol. The following information about the handshake interface is provided here for reference.

Global Signals

The hardware module has one Clock input and one global Reset. Data transfers on each data port are synchronized to the Clock. The global Reset **must be held active high for at least one clock cycle** and returns all registers to a known state.

Input Synchronizing Signals

ac_InputAvail (input) This signal is controlled by the external design and indicates that all data on the input port(s) are valid. This causes the receiving device (the hardware module) to capture the data on the rising edge of the next clock cycle.

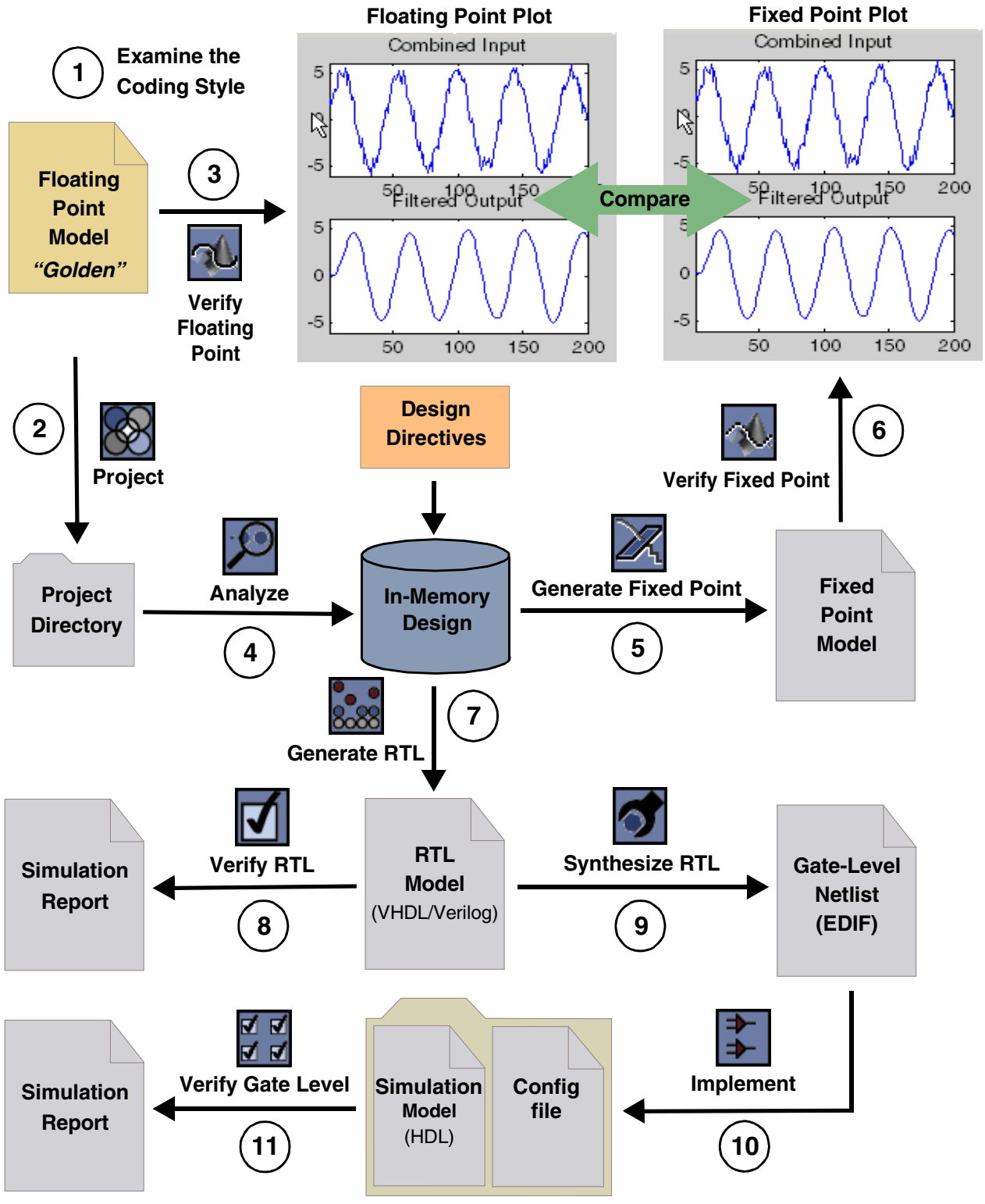
ac_InputReq (output) This signal is controlled by the hardware module and when set high, indicates that the module is ready to capture new data from the input port(s). When the module sets this signal low, the external design should immediately stop sending new data. If the hardware module holds this signal constantly high, then new data will be captured on every clock cycle provided the sending device can send data that fast.

Output Synchronizing Signals

ac_OutputAvail (output) This signal is controlled by the hardware module and indicates that data on the output port(s) is valid. Once this signal is set high, it will remain high until the receiving device acknowledges the data capture by setting ac_OutputAck high. If the external design holds ac_OutputAck constantly high, then the hardware module will send data at the maximum possible rate, as governed by the module clock frequency and the Startup Clock Cycles of the computing algorithm.

ac_OutputAck (input) This signal is controlled by the external design and indicates that the data on the output port(s) has been captured. If the external design holds this signal constantly high, then the hardware module will send data out at the maximum rate possible, as governed by the module clock frequency and the Startup Clock Cycles of the computing algorithm.

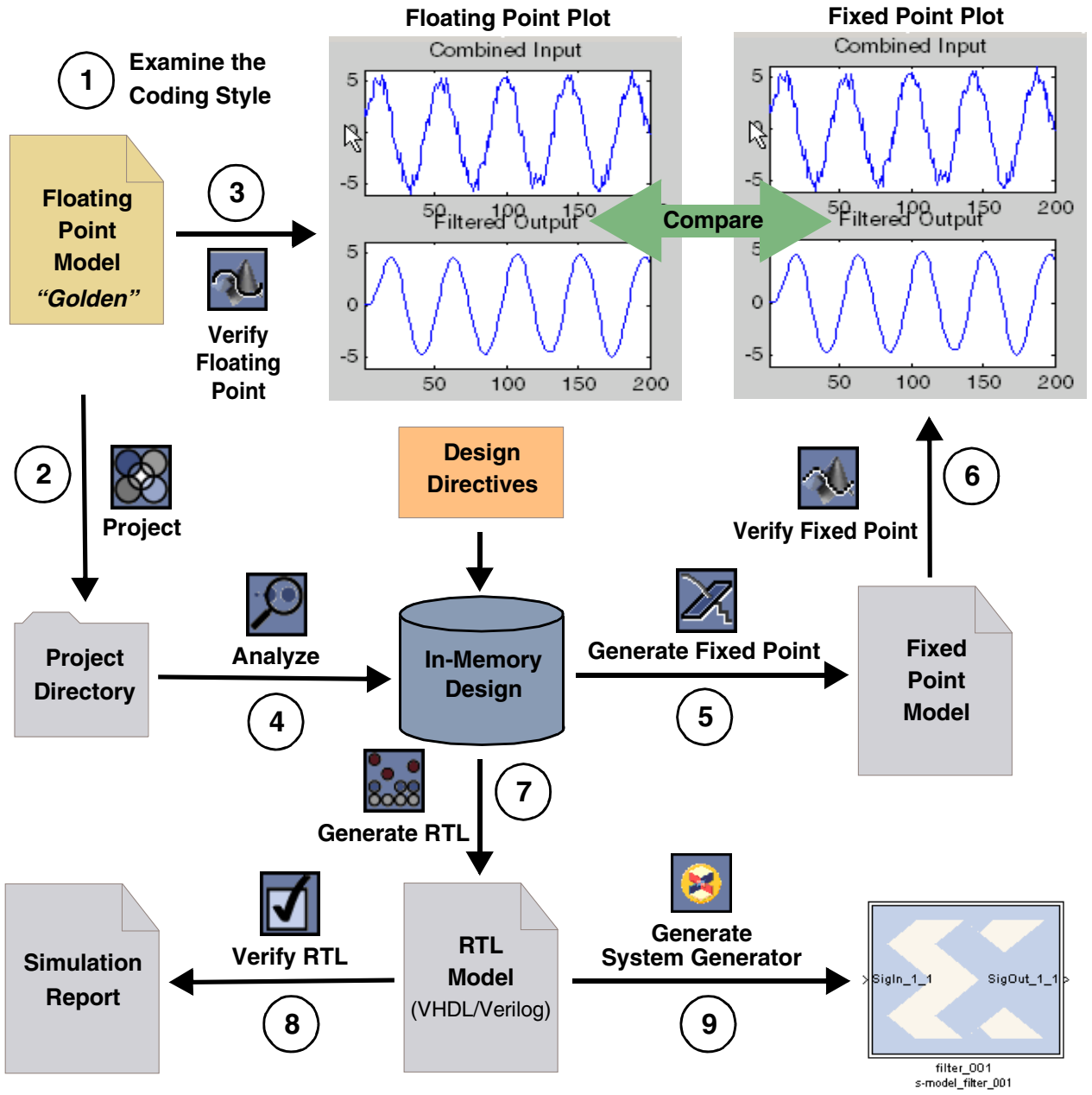
Summary of the ISE Synthesis Flow



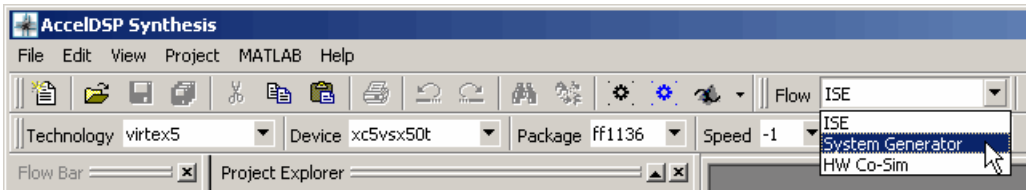
AccelDSP Synthesis is a tool that synthesizes a MATLAB floating-point model into a hardware module that can be implemented in a Xilinx FPGA.

- The floating-point model must conform to minimum coding style guidelines that are described in the manual titled MATLAB for Synthesis Style Guide
- After the “golden” floating-point design is analyzed, an equivalent in-memory design is created.
- AccelDSP Synthesis generates an equivalent fixed-point MATLAB model
- AccelDSP is integrated with MATLAB and allows you to run a MATLAB simulation on both the floating-point and fixed-point model
- You determine if the fixed-point simulation is successful either by visual inspection or other means
- You can guide the synthesis process by applying Design Directives to the in-memory design
- AccelDSP Synthesis generates an RTL Model from the in-memory design in either VHDL or Verilog format
- AccelDSP generates a hardware Testbench to verify the functionality of the RTL Model.
- AccelDSP Synthesis automatically calls an HDL simulator to run the Testbench
- AccelDSP automatically calls an RTL Synthesis tool to transform the RTL Model into a gate-level netlist that is hardware implementation ready
- AccelDSP automatically calls the ISE implementation tools to place and route the design. A simulation model is generated for post-P&R gate-level verification.
- During the last verification step, the input stimulus and output results from the fixed-point MATLAB simulation are applied to the post-P&R simulation model to verify that the implemented design is 'bit-true'
- The implemented design is typically a hardware module that is included as part of a larger design on the FPGA
- The synthesized hardware module communicates with external design elements through a handshake protocol.

The System Generator Synthesis Flow



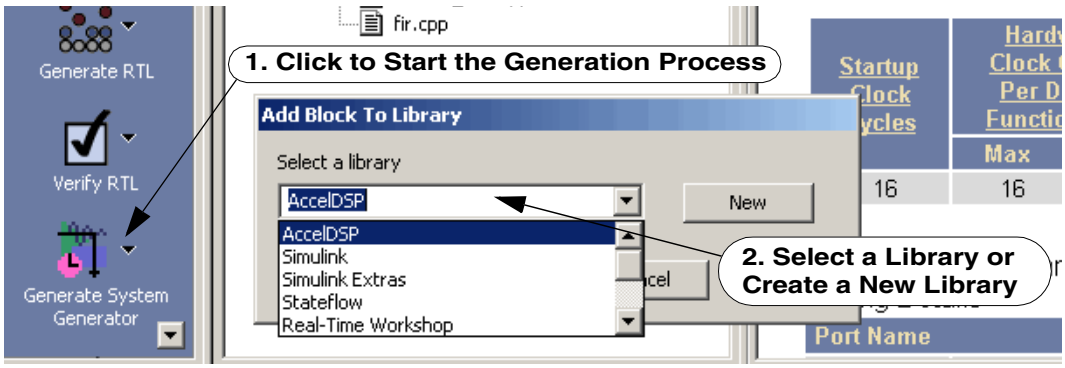
After you open a project, you can specify the **System Generator** flow from the Flow dialog box, as shown below:



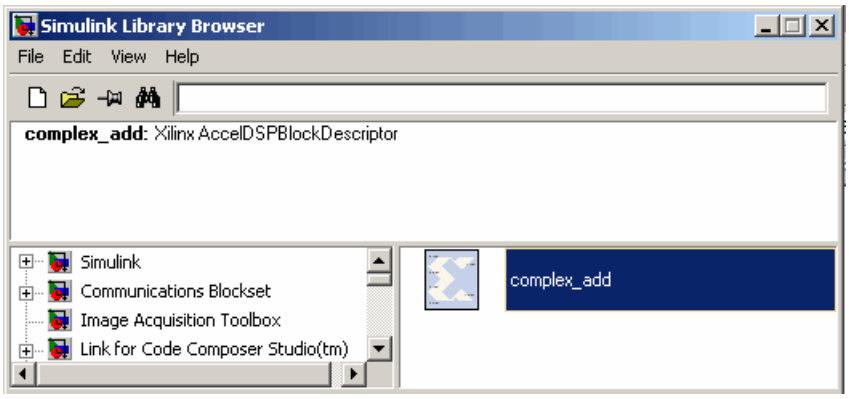
The icons in the GUI Flow Bar will then change to guide you through the **System Generator** Flow.

Generating the System Generator Block

You can start the block generation process any time after the Verify RTL step by clicking on the **Generate System Generator** icon as shown below.



AccelDSP asks you to select an existing library or name a new library in which to place the block. After you click OK, the files for the new block are generated and the block is associated with the target library. All relevant MATLAB Search Path links are automatically created. The next time you open the Simulink Library Browser, you'll see the new block as shown below:



Reserved Names for Block Ports

A generated MATLAB design function cannot have inputs or outputs named **rst**, **ce** or **clk**. If this is the case, you'll need to rename these ports.

Chapter 4

Creating a New Project

Introduction

The AccelDSP synthesis tool makes it easy for you to create a new project. You can start with an existing project, like a project from the AccelDSP Examples directory and copy the project files to a new project folder. Optionally, you can create a new project folder and copy (or move) your MATLAB® source files to the folder. This is treated as a new project. Finally, you can open an existing project by simply specifying the pathname to or navigating to an existing project file.

Creating a New Project from an Example

The AccelDSP synthesis tool provides a rich selection of design examples that you can choose from to start a new project. As shown in Figure 4-1, the idea is to select an example

that closely approximates your target design, copy the design files to a new project folder, then modify or replace the files to match your specifications.

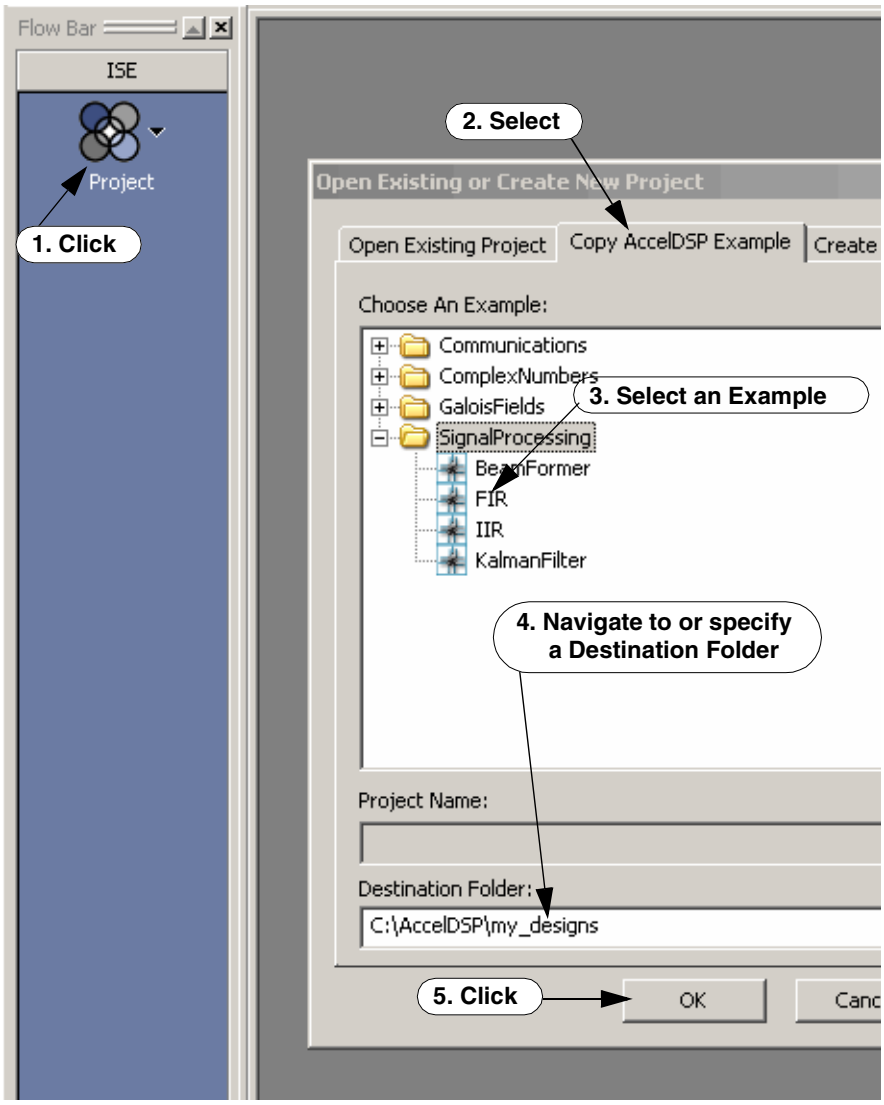


Figure 4-1: **Creating a New Project from an Example**

Creating a New Project from MATLAB Source Files

In Figure 4-2, a floating-point design called **newfilter** was created using MATLAB running on a laptop in an offsite location. The M-files are being transferred from a memory stick (E drive) that is plugged into a desktop computer. The user clicks on the Project icon to open the dialog box, then selects the Create New Project tab.

After entering a Project Name and selecting a Destination Folder, the user navigates to the E drive, selects a file to be copied, then clicks Add. When all files to be added are listed in the right-hand box, the user clicks OK to start the copy process. After the files are copied,

AccelDSP creates a new project file by the specified name and adds it to the Destination Folder.

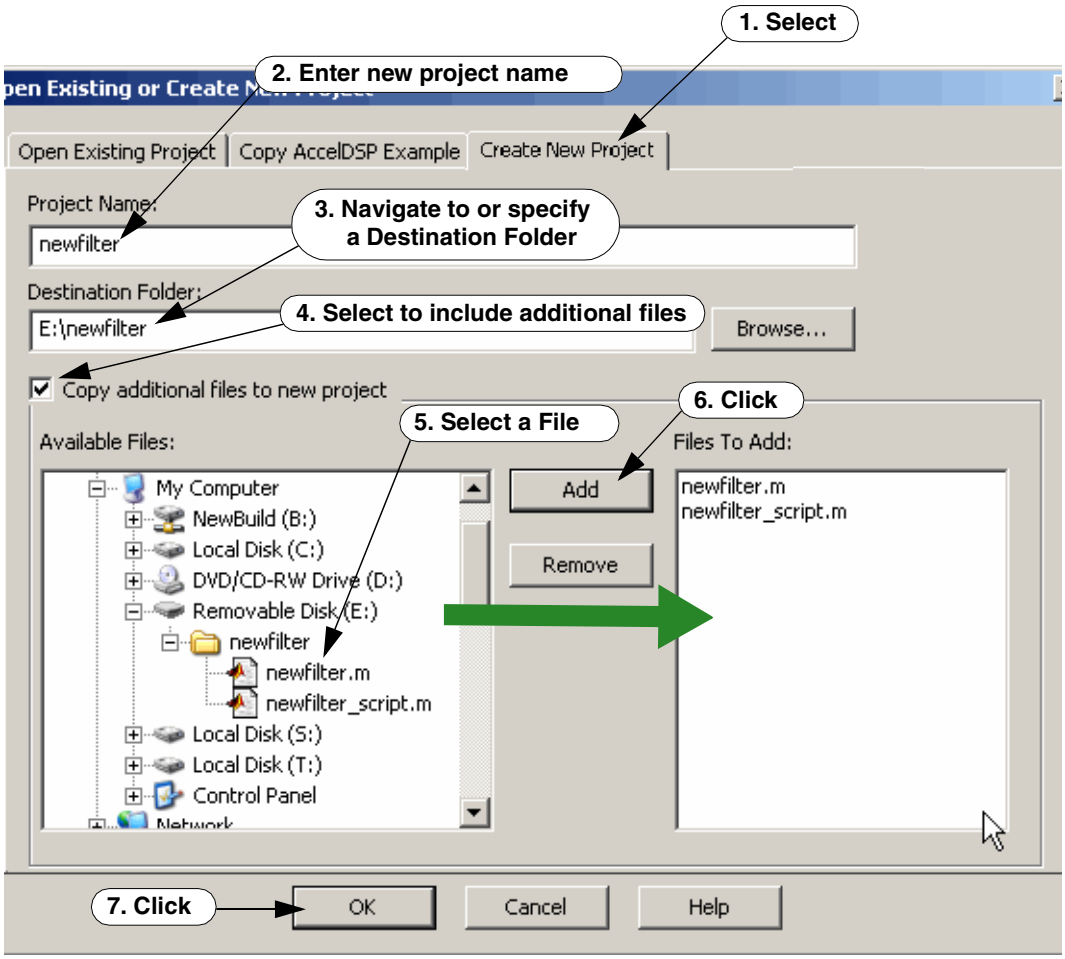


Figure 4-2: **Creating a New Project from MATLAB Source Files**

Opening an Existing Project

After you create and open a project, the AccelDSP synthesis tool remembers where the project resides and remembers the last successfully completed step in the AccelDSP flow. When you click the Project icon at a later time, the most recently opened projects are listed. Simply click on the project name, then click OK. The project is then auto-restored to the state when the project was last closed.

As shown in Figure 4-3, if you have an existing project that has not yet been opened, you can browse to and select the project file, then click OK and the project will open.

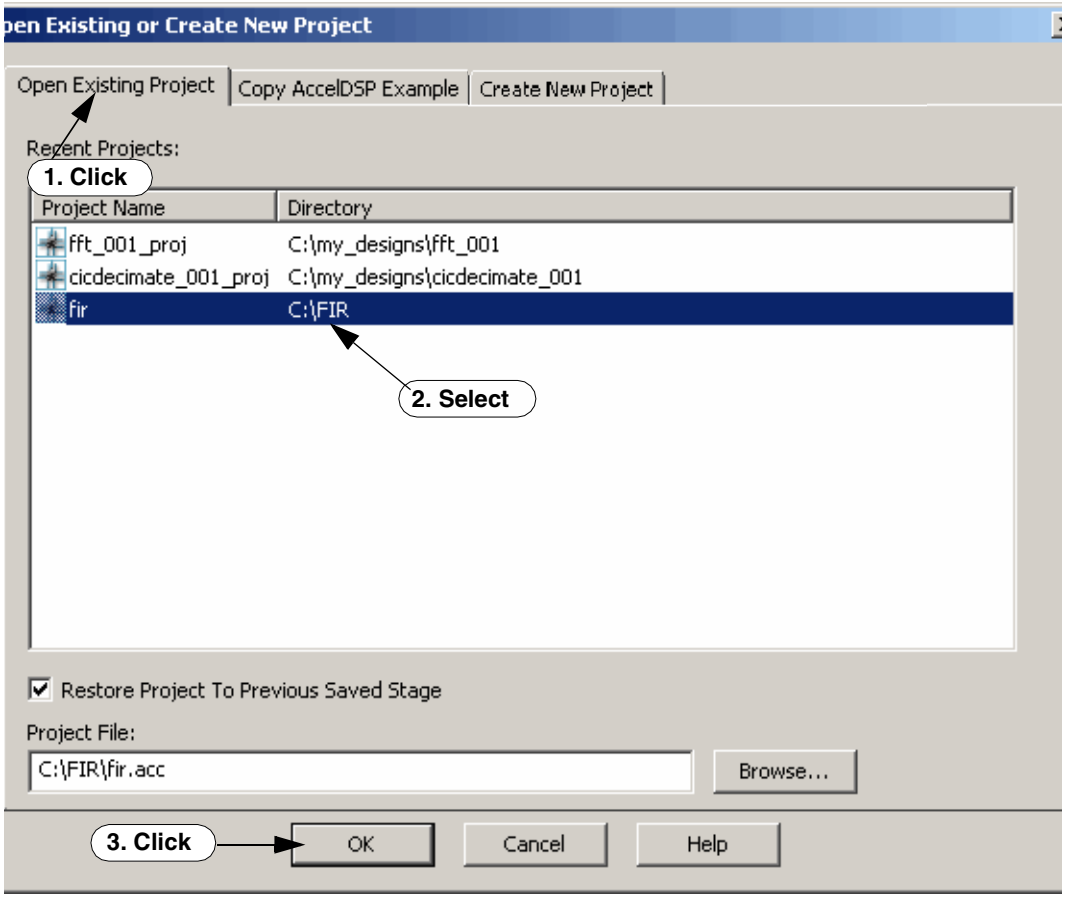


Figure 4-3: **Opening an Existing AccelDSP Project**

Managing the Recent Projects List

As shown in Figure 4-4, the Recent Projects list can contain up to 8 projects. The most recently opened project is listed at the top. In this case, if you open a new project, the fir project at the bottom of the list will drop off.

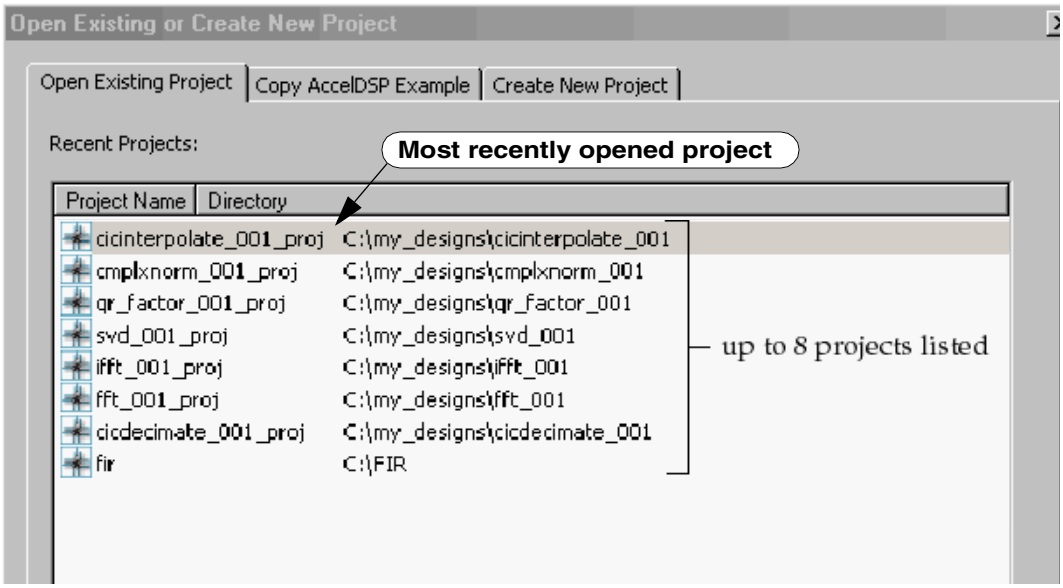


Figure 4-4: **Adding a New Project to the Recent Project List**

As shown in Figure 4-5, you can selectively remove any item in the list. Right-click on the item to be removed, then select Remove From Recent List.

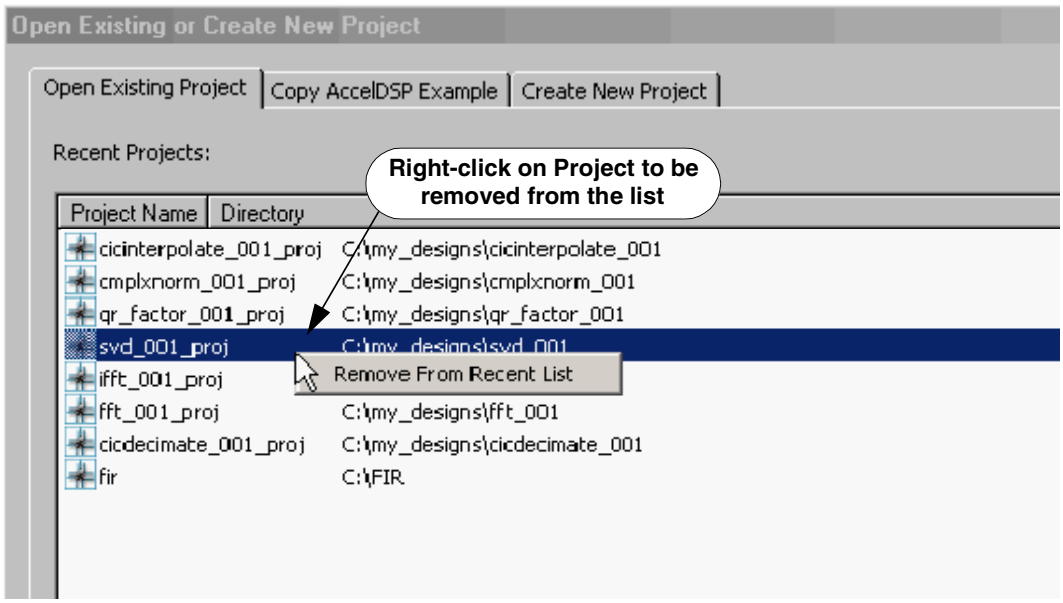


Figure 4-5: **Removing an Item fron the Recent Projects List**

XILINX®

Chapter 5

Understanding Generate Fixed Point

Converting Floating-Point Numbers to Fixed-Point

During algorithm developed, floating-point numbers are often used because they represent infinite precision. When it comes time to realize the algorithm in hardware, floating-point numbers are often not practical. The solution is to convert very precise floating-point numbers to less precise fixed-point numbers. In MATLAB®, this conversion process is called *quantization* and is done using the *quantizer* and *quantize* function from the Filter Design Toolbox or the AccelDSP version of these functions (toolbox not necessary). In the AccelDSP ISE Synthesis Flow, quantizing the floating-point model to fixed-point is done automatically during the Generate Fixed Point step using a built-in AccelDSP *quantize* function. After the automatic conversion is complete, you can manually adjust the quantizing of the floating-point model to reduce the quantization error and increase the fidelity of the output.

What is a MATLAB Quantizer?

A *quantizer* is a MATLAB database object that is created by the *quantizer* function. The quantizer function comes from the Filter Design Toolbox. Once created, the quantizer object has properties that specify the fixed-point or customized floating-point characteristics of the floating-point number to which it is applied. These properties are summarized in the table below:

Table 5-1: **Valid MATLAB Quantizer Properties**

Property Name	Property Value	Description
DataMode	'double'	Double-precision mode. Override all other parameters.
	'float'	Custom-precision floating-point mode.
	'fixed'	Signed fixed-point mode.
	'single'	Single-precision mode. Override all other parameters.
	'ufixed'	Unsigned fixed-point mode.
Roundmode	'ceil'	Round to the closest representable number in the direction of positive infinity.
	'convergent'	Round to the closest representable integer. In the case of a tie, it rounds to the nearest even stored integer. This is the least biased rounding method provided by the Fixed-Point Toolbox.

Table 5-1: **Valid MATLAB Quantizer Properties**

Property Name	Property Value	Description
	'fix'	Round to the closest representable integer in the direction of zero..
	'floor'	Round to the closest representable number in the direction of negative infinity.
	'nearest'	Round to the closest representable integer. In the case of a tie, it rounds to the closest representable integer in the direction of positive infinity. This is the default rounding method for fi object creation and fi arithmetic.
	'round'	Round to the closest representable integer. In the case of a tie, it rounds positive numbers to the closest representable integer in the direction of positive infinity, and it rounds negative numbers to the closest representable integer in the direction of negative infinity.
Overflowmode (fixed-point only)	'saturate'	Saturate a max value on overflow.
	'wrap'	Wrap on overflow.
Format	[wordlength fractionallength]	The format for `fixed` or `ufixed` mode.
	[wordlength exponentlength]	The format for `float` mode.

How is a MATLAB Quantizer Defined and Applied?

In the first line of the following MATLAB code segment, the properties of a quantizer object named qpath are defined. In the second line, qpath is used in the quantize function to convert the floating-point numbers in array x, then assign them to array indata.

```
qpath = quantizer('fixed','floor','wrap',[8,0]);
indata = quantize(qpath,x);
```

An alternative method for quantization is shown in the following MATLAB code segment. Instead of specifying the name of the quantizer object as an argument, the quantizer function call itself is placed directly into the quantize function call.

```
indata = quantize(quantizer('fixed','floor','wrap',[8,0]),x);
```

In MATLAB, if you use the quantizer function without specifying arguments, the following default properties are used:

```
mode = 'fixed';
roundmode = 'floor';
overflowmode = 'saturate';
format = [16 15];
```

What is Auto-Quantization?

When you execute the Generate Fixed Point step in the AccelDSP synthesis tool, an auto-quantization algorithm is applied to the in-memory design to produce a fixed-point model. An equivalent fixed-point Model of one or more M-file(s) is also written as an output.

If the dynamic range of a variable can be statically determined, the Auto-Quantizer uses a bit width and fractional part that can handle the range. For example, assume that your design contains the following two assignments: X = 16; and later on X = .125 The AccelDSP synthesis tool will create a fixed-point model with the quantized statement shown below:

```
'ufixed','floor','wrap',[ 8, 3 ]
```

The quantizer assumes the worst case condition. In this case, the integer part of X must be 5 bits to hold the integer 16 and the fractional part of X must be three bits to hold the fraction .125. If this automated guess is too small or too great, you can modify the range by applying a quantize directive through the AccelDSP GUI, as described later in this chapter.

The AccelDSP quantizer properties that you can specify are a subset of the MATLAB quantizer function properties. These properties only apply to fixed-point data conversion and are listed in the following table:

Table 5-2: **Valid AccelDSP Quantizer Properties**

Property Name	Property Value	Description
Mode	'fixed'	Signed fixed-point mode.
	'ufixed'	Unsigned fixed-point mode.
Roundmode	'floor'	Round to the closest representable number in the direction of negative infinity.
	'nearest'	Round to the closest representable integer. In the case of a tie, it rounds positive numbers to the closest representable integer in the direction of positive infinity, and it rounds negative numbers to the closest representable integer in the direction of negative infinity.
Overflowmode	'saturate'	Saturate a max value on overflow.
	'wrap'	Wrap on overflow.
Format	[wordlength fractionlength]	The format for `fixed` or `ufixed` mode.

The Generate Fixed Point Iterative Flow

Generating the fixed-point model is a very important step and is illustrated in the sub-flow diagram in Figure 5-1. You start by clicking on the Generate Fixed Point icon in Flow Bar. The AccelDSP Auto-Quantizer analyzes the in-memory floating-point design and then generates a fixed-point MATLAB in-memory model, a fixed-point M-file, and a report. You then click on the Verify Fixed Point icon and the AccelDSP synthesis tool automatically invokes MATLAB to simulate the generated fixed-point design. Simulation messages from MATLAB are recorded in the Tcl Console window, and then control is returned to the AccelDSP synthesis tool.

It is up to you to compare the fixed-point results to the floating-point results and determine if the quantization error is acceptable. In this case, the designer has chosen to evaluate the

fixed-point quality by visually comparing the fixed-point plot to the "golden" floating-point plot. Alternatively, the designer may have decided to use a mathematical method to comparing the simulation results.

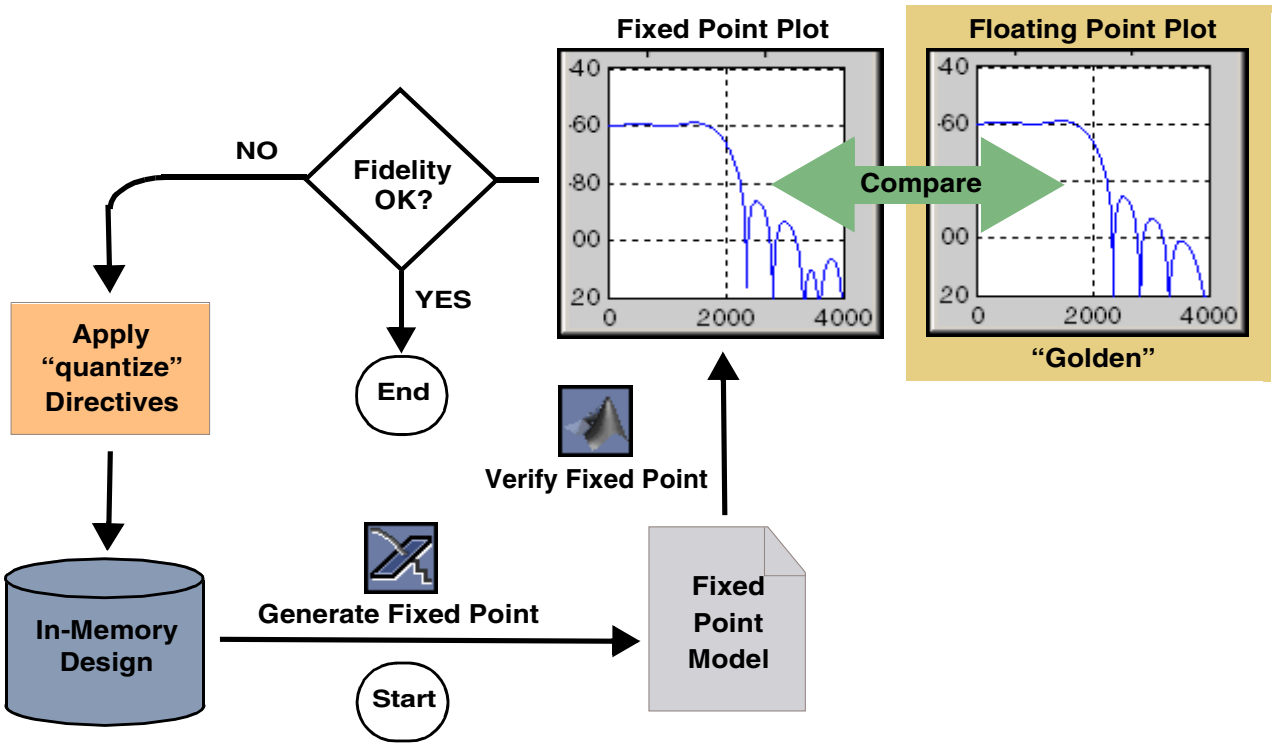
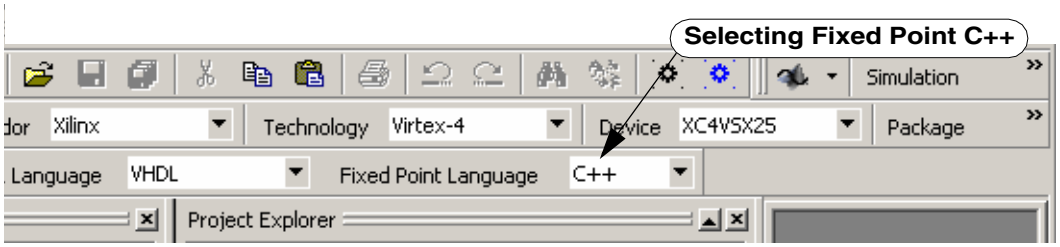


Figure 5-1: **The Generate Fixed Point Iterative Flow**

Choosing the Fixed Point Language

By default, the AccelDSP synthesis tool generates the fixed-point design in MATLAB. As shown below, you may choose C++ as the fixed-point language.



Generating a MATLAB Fixed Point Model

After the floating-point model is analyzed, you click on the **Generate Fixed Point** icon to generate an equivalent MATLAB fixed-point model. The structure of the fixed-point model is displayed in the Project Explorer window, as shown on the left, and the AccelDSP synthesis tool writes the fixed-point M-file(s) to a newly-created FixedPointM folder in the project directory. In the next step, these files will be used in a MATLAB simulation to verify the fidelity of the fixed-point model.

The AccelDSP synthesis tool also displays a Generate Fixed Point report (not shown) to provide important information about the generated design.

Generating a C++ Fixed Point Model

If you have C++ selected as the Fixed Point Language, you click on the **Generate Fixed Point** icon to generate an equivalent C++ fixed-point model. The AccelDSP synthesis tool writes the fixed-point C++ file(s) to a newly-created FixedPoint C folder in the project directory. In the next step, these files will be used in a C++ simulation to verify the fidelity of the fixed-point model.

The AccelDSP synthesis tool also displays a Generate Fixed Point report (not shown) to provide important information about the generated design.

Evaluating the Quality of the Generated Fixed Point

The goal of Generate Fixed Point step is to:

> *achieve acceptable **fidelity** while at the same time maintaining minimum **bit width** and **bit growth***

What is Acceptable Fidelity?

The term "fidelity" in this context is defined as how well the simulation results of the fixed point match the simulation results of the floating point. There will always be a certain amount of precision lost in the conversion. This is called "quantization error." It is up to you to determine if the loss is acceptable for a particular application. If the fidelity loss is unacceptable, you can adjust the quantization on the in-memory design by applying AccelDSP "directives". You can then repeat the Generate Fixed Point sub-flow, until the fidelity is acceptable.

The 53-Bit Limit on All Fixed-Point Data

The AccelDSP Auto-Quantizer recognizes the MATLAB 53-bit limit for simulating bit-true fixed-point arithmetic and makes decisions accordingly. This ensures that the MATLAB simulation results can be used for "bit-true" comparisons later in the AccelDSP ISE Synthesis Flow.

Using fi Objects for Greater Rounding Accuracy

fi objects are fixed-point numeric objects that are used in the the MathWorks Fixed-Point Toolbox. AccelDSP only uses fi objects temporarily for internal operations that produce results greater than 53 bits. The results is then truncated back to 53 bits.

By default, the use of fi objects is turned off to speed the fixed-point simulation run. If your design has internal operations that result in values greater than 53 bits, you may get simulation mis-matches due to rounding error. Turning on the use of fi objects may eliminate the rounding error. The tradeoff is an increase in MATLAB simulation run time.

To turn on the use of fi objects, click on the black snow flake icon to bring up the Project Options dialog box. Click the **Advanced Mode** button (lower-right corner), click on the **Output** category and set the Fi Object option to 'true'.

The 53-Bit Limit on Design Inputs and Outputs

The I/O ports on the generated AccelDSP Testbench are limited to 53 bit unsigned numbers or 54 bit signed numbers. Your design data may exceed this limit, but the ports, never-the-less, will be quantized to 53 bit unsigned numbers or 54 bit signed numbers. If this happens, we recommended that you rescale the data to conform to this Testbench requirement.

Maintaining Realistic Bit Widths

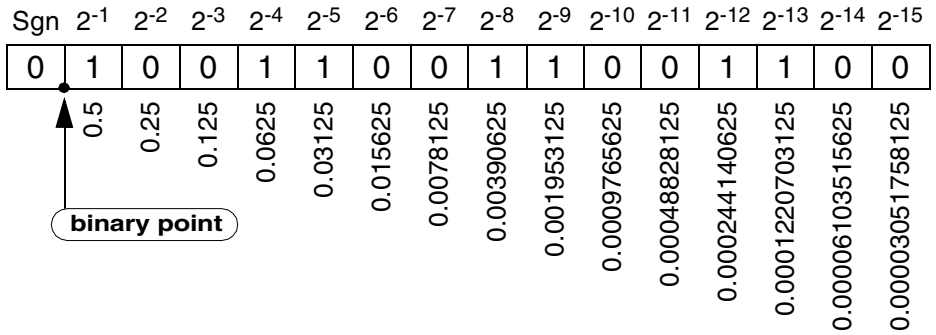
If the bit width of a design input is larger than necessary, you are going to be wasting hardware. In addition, bit growth starts at the design inputs and will increase from that point. Inputs that start too big may cause bit growth to increase unnecessarily as the data passes through multipliers and other math operations.

On the other hand, inputs that are quantized too small may result in un-usable hardware. The Auto-Quantizer *infers* the bit width of each input by analyzing the dynamic range of the data being applied. For example, if you want to test the theoretical response of a FIR filter, you can apply an "impulse" represented by applying a series of zeros and then the integer "1". The MATLAB simulation will produce a nice characteristic plot, but the Auto-Quantizer will quantize the input to one bit, the binary representation of the integer "1".

The solution to the above situation is to apply a more realistic set of random data samples ranging from 0 to 255. This will cause the Auto-Quantizer to evaluate the dynamic range and set the input width to 8 bits. Another possible solution is to keep the "impulse" stimulus, but set an AccelDSP quantize directive on the input specifying 8 bits for the word width.

Managing the Bit Width of Fractional Numbers

In order to understand how bit widths can grow due to fractional numbers, consider how the fraction .6 is represented as sixteen binary bits in a two's complement format.



$$.6 = 2^{-1} + 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} = 0.59997558593$$

Figure 5-2: **How Fractional Numbers are Represented in Binary**

In Figure 5-2, the word width is restricted to 16 bits, so the closest approximation to .6 is 0.59997558593. If the bit width were allowed to expand, the binary number would approach .6 but never quite reach it. In this case, the AccelDSP auto-quantize algorithm would allow this quantization to expand to 53 bits, the MATLAB maximum for bit-true fixed-point arithmetic. It would be up to you to recognize the maximum in the report and

trim back the precision to a reasonable number of bits. For many DSP applications, 16 bits are sufficient.

Understanding the Generate Fixed Point Report

The Generate Fixed Point Report is created as an output of the Generate Fixed Point step and is saved in the project directory under Reports.

The Report is presented in a tabbed format that includes a Variables by Hierarchy list, a Variables List, Operators List, Loop List, and Functions List.

Variables by Hierarchy Report

As shown in Figure 5-3, AccelDSP quantizations are always applied to variables. The quantizer properties can be inferred from the dynamic range of the data being assigned (Auto-Inferred), inferred from an explicitly applied AccelDSP quantize directive, or inferred from an explicit quantize assignment in the MATLAB source. Some variables may be quantized to the maximum which usually indicates that the quantizer properties are unconstrained and need further attention and reduction. You can change the quantize values by clicking on the values in the report.

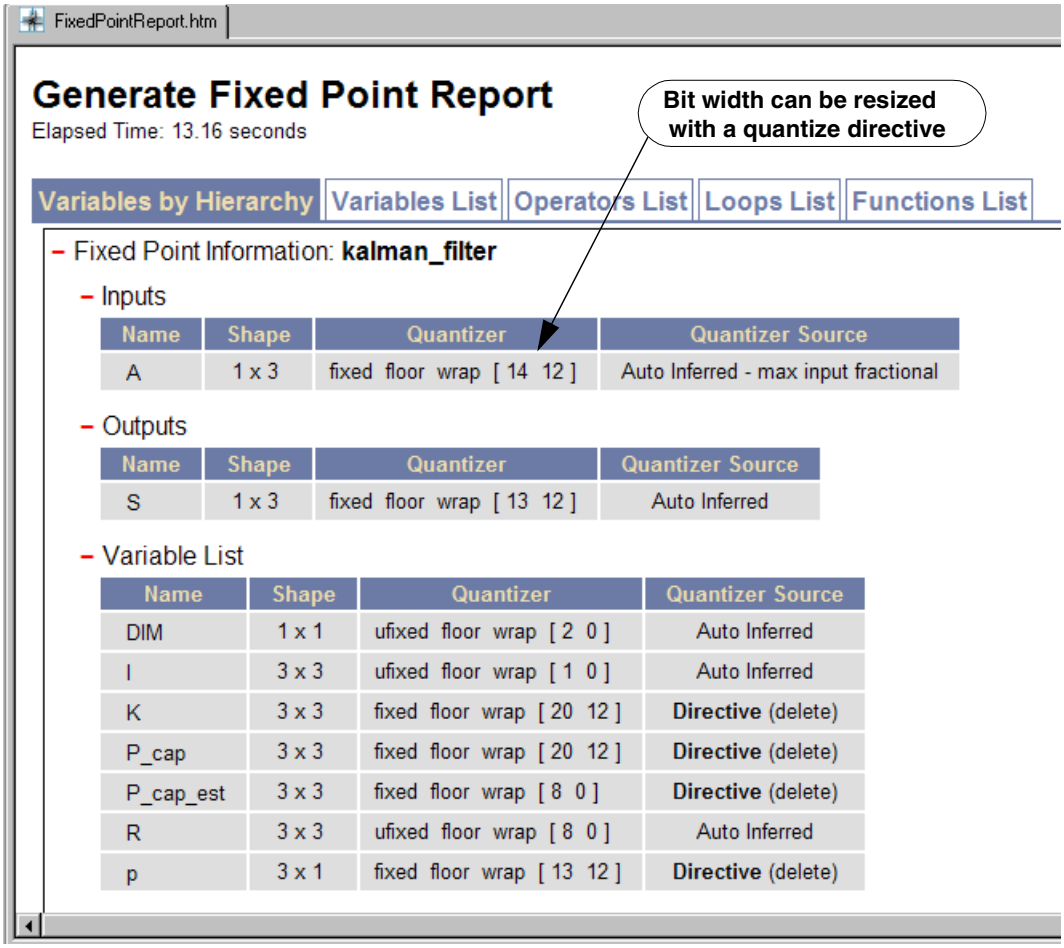


Figure 5-3: **Understanding the Generate Fixed Point Report**

Using Variables by Hierarchy Tab to Adjust a Quantizer Value

Figure 5-4 illustrates how use the report to directly change a quantizer value.

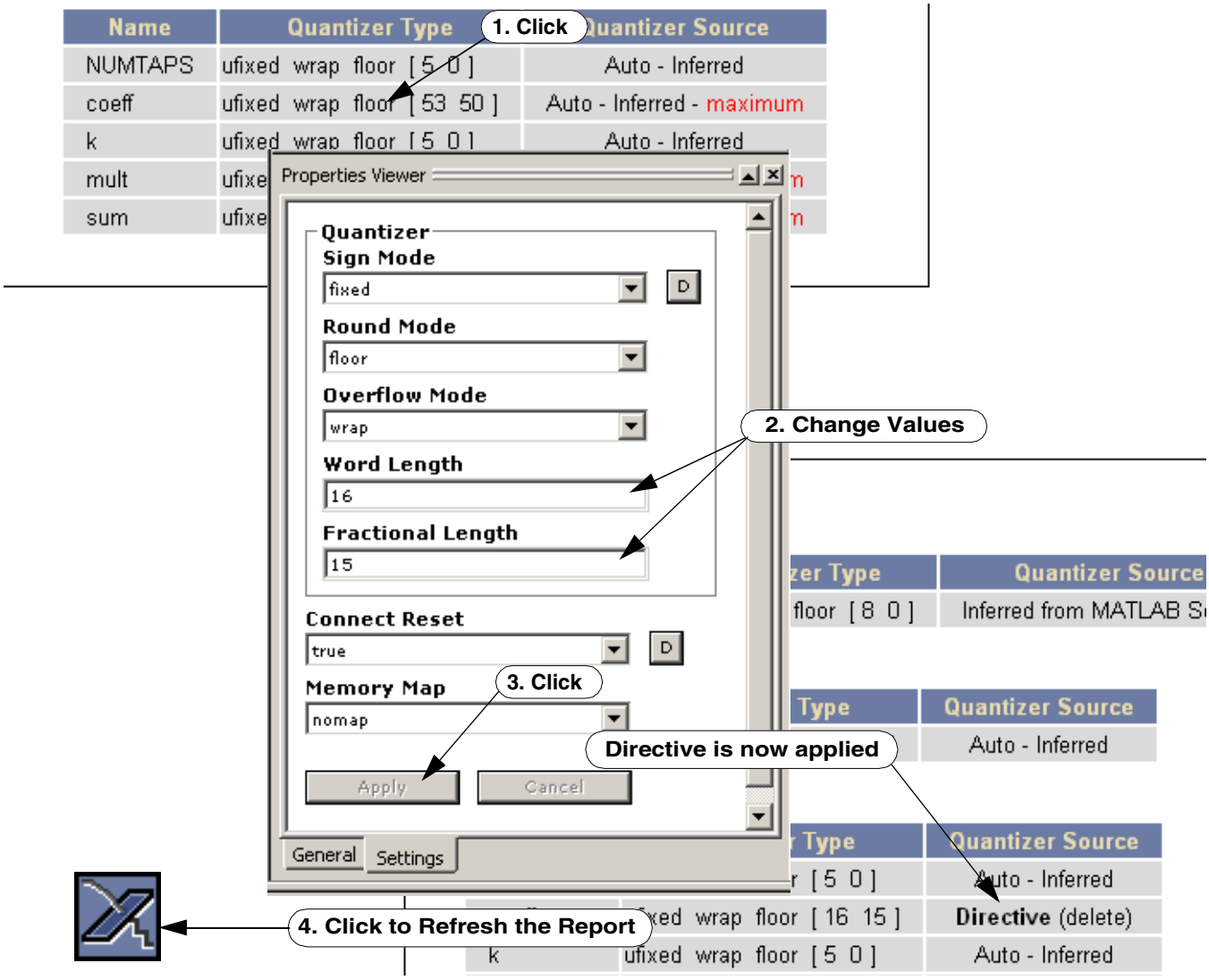


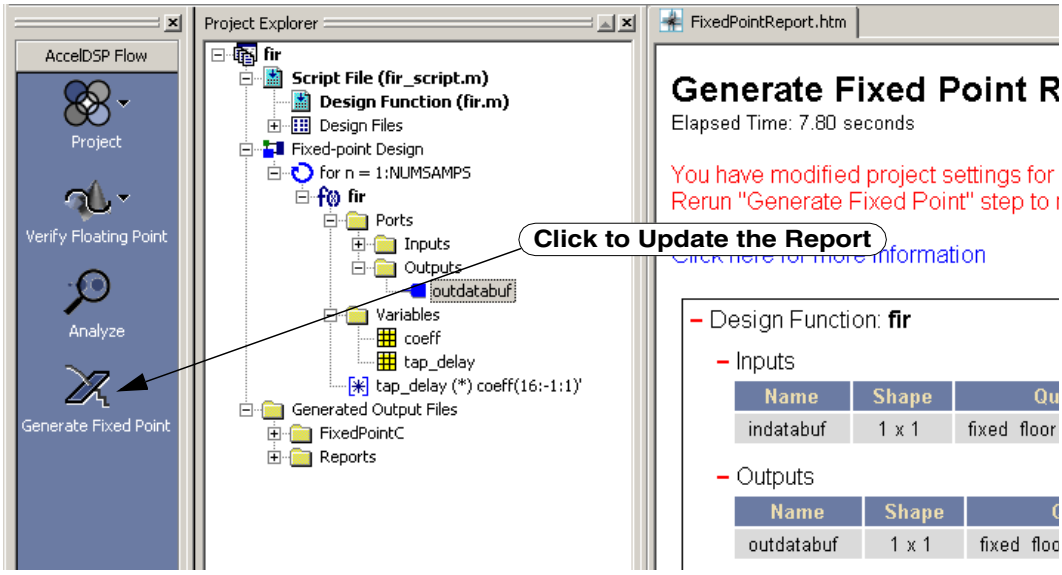
Figure 5-4: **Applying a Quantize Directive from the Fixed-Point Report**

After you change the quantizer properties, the change is made to the in-memory design. Once Applied, if you click on the (delete) label, the directive is deleted from the in-memory design.

If you want to save the quantizer changes, you can execute the pulldown menu **File > Save Project** and your changes will be saved to the ADD file in the project directory. The next time you open the project, the directives defined in the ADD file will be re-applied to the in-memory design.

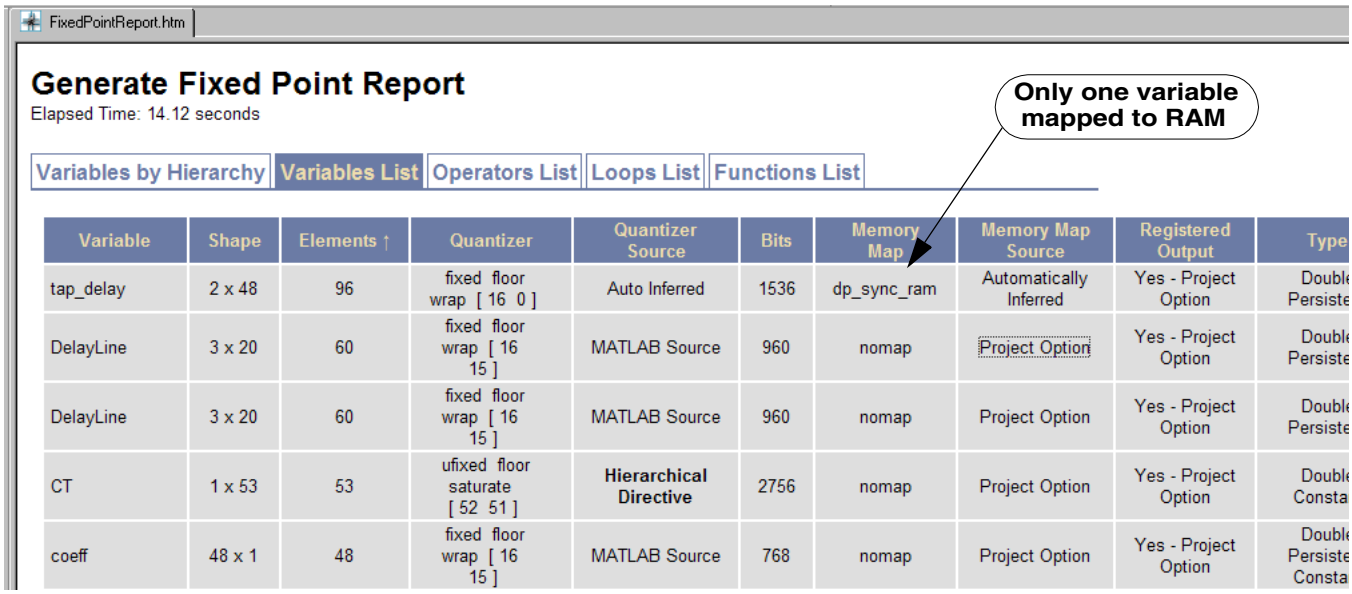
Updating the Generate Fixed Point Report

The Generate Fixed Point Report may not reflect the current state of the design if you have changed an AccelDSP directive or changed a project setting such as the target frequency. To update the report, rerun the Generate Fixed Point step as shown below.



Variables List Report

Figure 5-5 shows the Variables List within the Generate Fixed-Point Report.



Variable	Shape	Elements ↑	Quantizer	Quantizer Source	Bits	Memory Map	Memory Map Source	Registered Output	Type
tap_delay	2 x 48	96	fixed floor wrap [16 0]	Auto Inferred	1536	dp_sync_ram	Automatically Inferred	Yes - Project Option	Double Persiste
DelayLine	3 x 20	60	fixed floor wrap [16 15]	MATLAB Source	960	nomap	Project Option	Yes - Project Option	Double Persiste
DelayLine	3 x 20	60	fixed floor wrap [16 15]	MATLAB Source	960	nomap	Project Option	Yes - Project Option	Double Persiste
CT	1 x 53	53	ufixed floor saturate [52 51]	Hierarchical Directive	2756	nomap	Project Option	Yes - Project Option	Double Consta
coeff	48 x 1	48	fixed floor wrap [16 15]	MATLAB Source	768	nomap	Project Option	Yes - Project Option	Double Persiste Consta

Figure 5-5: **Variables List Fixed Point Report**

This report lists each design variable and provides important information about the variable attributes. You can change an attribute by clicking on the variable row, then change the attribute value in the Properties Viewer. In this example, you can see that only one array variable is mapped to Dual-Port RAM.

You can use this report to tune your design for maximum performance. In this case, only one variable is mapped to RAM because the default value for the RAM Threshold project option is set to 64 (elements). Rather than apply a memmap directive to the other four variables listed below tap_delay, you can simply change the RAM Threshold project option to 48 as shown in the figure below.

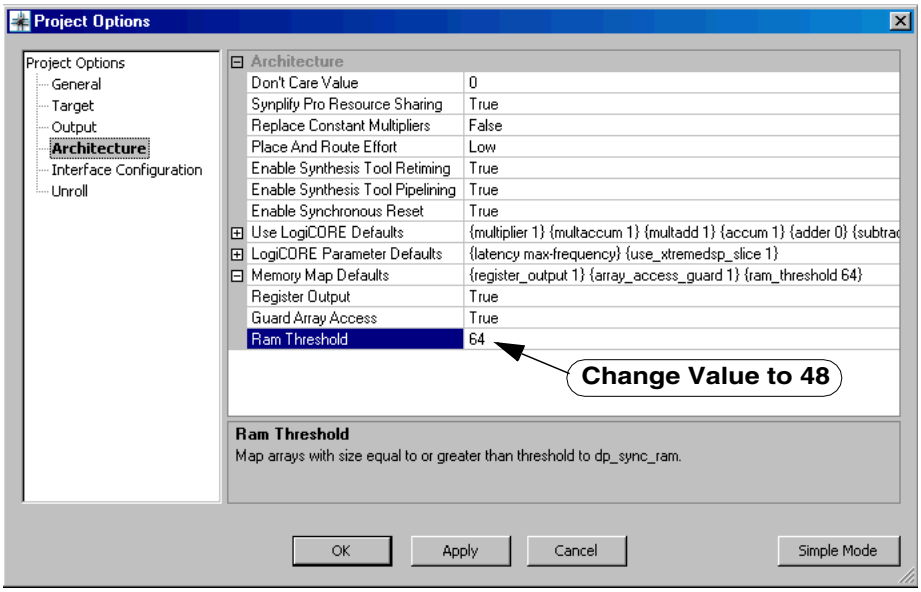


Figure 5-6: **Changing the RAM Threshold Project Option**

The following figure shows how four array variables are mapped to memory with one change in the RAM Threshold project option.

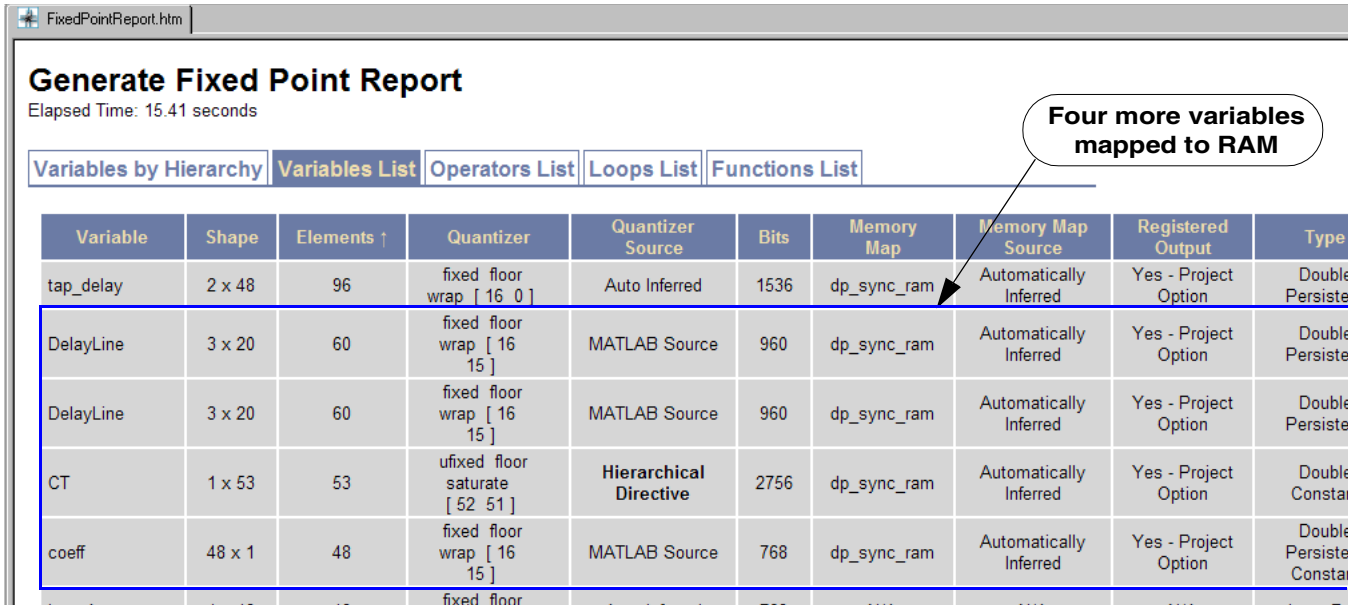


Figure 5-7: **Results of Changing the RAM Threshold**

Operators List Report

Figure 5-8 shows the Operators List within the Generate Fixed-Point Report.

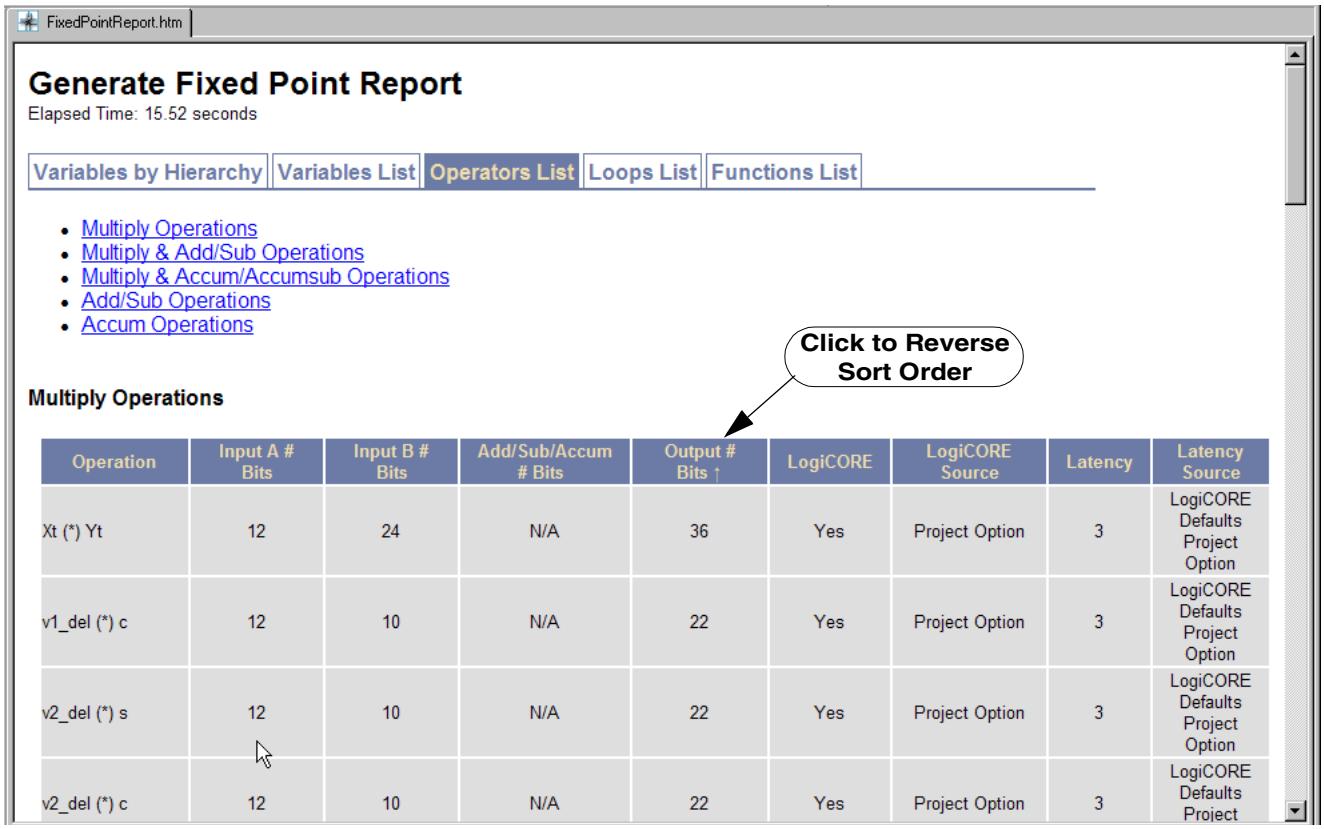


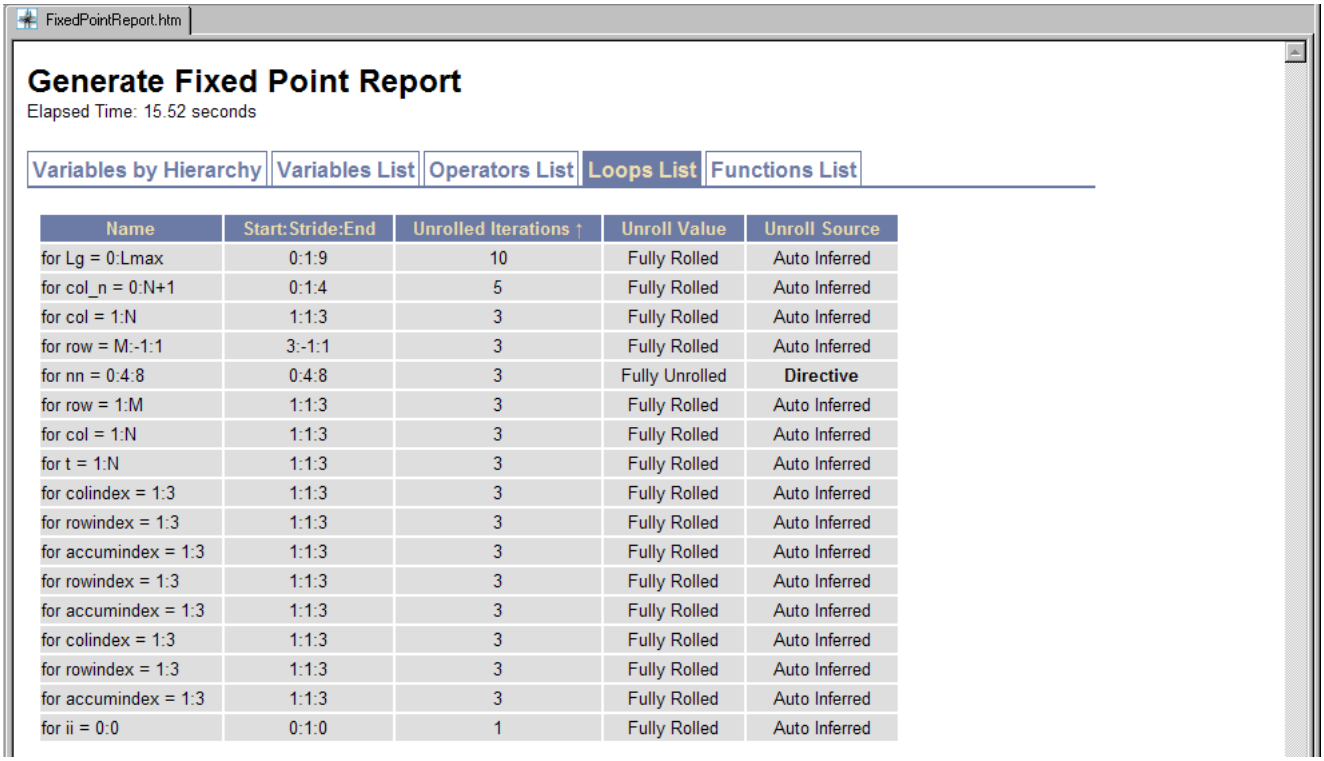
Figure 5-8: **Operators List Fixed Point Report**

This report lists the name of each operator and provides important information about the operator parameters. You can change a parameter by clicking on the parameter in the operator row, then change the parameter value in the Properties Viewer.

You can use this report to tune your design for maximum performance. For example, you can make sure that an operator is mapped to a LogiCORE, if possible, then change the LogiCORE latency to the desired value, as-well-as, adjust the input quantization of each input. If you click on the column header of a particular parameter, the sort order for that column will reverse.

Loops List Report

Figure 5-5 shows the Loops List within the Generate Fixed-Point Report.



FixedPointReport.htm

Generate Fixed Point Report

Elapsed Time: 15.52 seconds

Variables by Hierarchy | Variables List | Operators List | Loops List | Functions List

Name	Start:Stride:End	Unrolled Iterations ↑	Unroll Value	Unroll Source
for Lg = 0:Lmax	0:1:9	10	Fully Rolled	Auto Inferred
for col_n = 0:N+1	0:1:4	5	Fully Rolled	Auto Inferred
for col = 1:N	1:1:3	3	Fully Rolled	Auto Inferred
for row = M:-1:1	3:-1:1	3	Fully Rolled	Auto Inferred
for nn = 0:4:8	0:4:8	3	Fully Unrolled	**Directive**
for row = 1:M	1:1:3	3	Fully Rolled	Auto Inferred
for col = 1:N	1:1:3	3	Fully Rolled	Auto Inferred
for t = 1:N	1:1:3	3	Fully Rolled	Auto Inferred
for colindex = 1:3	1:1:3	3	Fully Rolled	Auto Inferred
for rowindex = 1:3	1:1:3	3	Fully Rolled	Auto Inferred
for accumindex = 1:3	1:1:3	3	Fully Rolled	Auto Inferred
for rowindex = 1:3	1:1:3	3	Fully Rolled	Auto Inferred
for accumindex = 1:3	1:1:3	3	Fully Rolled	Auto Inferred
for colindex = 1:3	1:1:3	3	Fully Rolled	Auto Inferred
for rowindex = 1:3	1:1:3	3	Fully Rolled	Auto Inferred
for accumindex = 1:3	1:1:3	3	Fully Rolled	Auto Inferred
for ii = 0:0	0:1:0	1	Fully Rolled	Auto Inferred

Figure 5-9: **Loops List Fixed Point Report**

This report showsthe start, stride and end values for each loop, the total number of unrolled iterations, the unroll values and the source of the unroll value. You can change the unroll value of a particular loop by clicking on the loop row, then changing the unroll value in the Properties Viewer.

Functions List Report

Figure 5-5 shows the Functions List within the Generate Fixed-Point Report.

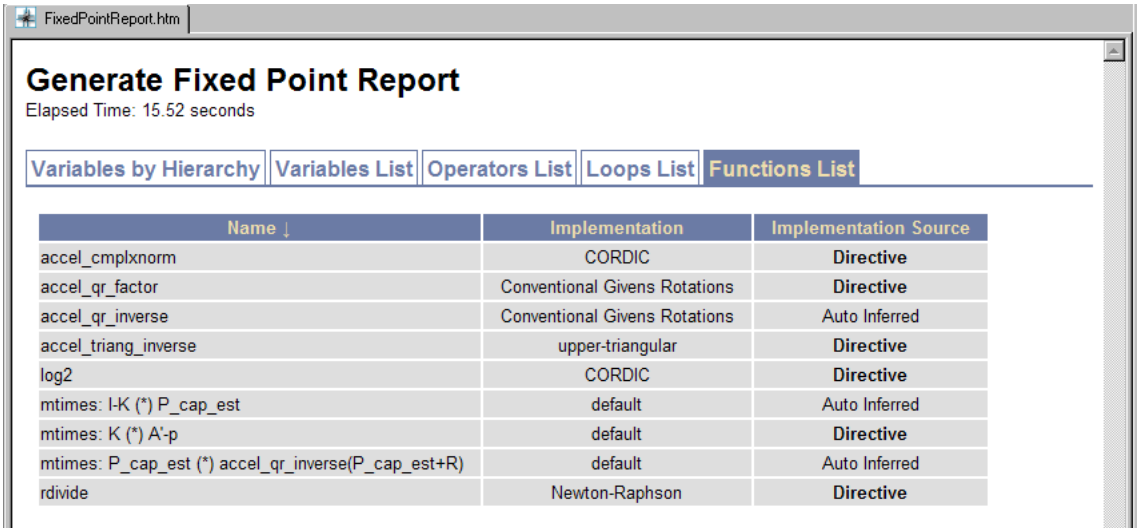
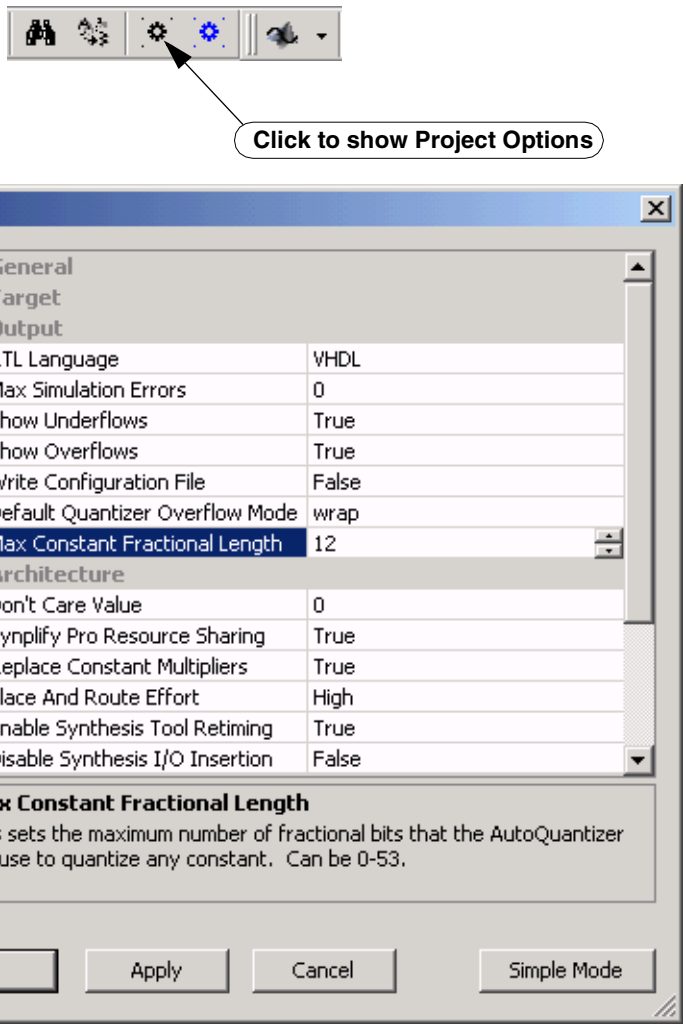


Figure 5-10: **Functions List Fixed Point Report**

This report provides the name of each function in the design, the implementation and the source of the implementation (Auto Inferred or Directive). You can change the implementation of a particular function by clicking on the function row in the report, then change the implementation value in the Properties Viewer.

Controlling the Fractional Length of All Constants

If left uncontrolled, many constants with a fractional part will infer a bit width of 53 bits. The AccelDSP synthesis tool has a project option that limits the fraction part of all constants to 12 bits, by default. You can raise or lower this default through the Project Option dialog box, as shown below:



Applying a Quantize Directive to a Constant

If you use a constant in your MATLAB source, it is important to limit the fixed-point word width to a realistic value. This is especially true for constants that have a fractional part. Assume for example, that you have the following code segment in your MATLAB floating-point design:

```
A = .6 + B;
```

The AccelDSP Auto-Quantizer will infer the quantization of the variable B from previous assignments to B. From the previous topic, you know that the Auto-Quantizer will select a 12 bit quantization (the default maximum) for the constant (.6), therefore a word width of no less than 12 bits will be inferred and assigned to A. The problem here is that the

AccelDSP quantize directive can only be applied to variables. Since the constant .6 is not a variable, it won't show up in the Generate Fixed Point Report. Therefore, you can't apply a quantize directive to the constant. The solution is to change the MATLAB source slightly, as shown below:

```
C = .6;
A = C + B;
```

In the above code, the constant is first assigned to the variable C, then used in the expression that follows. In this case, the variable C will show up in the Generate Fixed Point Report and you can apply a quantize directive to it.

Applying a Quantize Directive to a Sub-Expression

Consider the following complex MATLAB expression:

```
a = b+(m*n)+(p*q);
```

When the Auto-Quantizer infers the quantization of the variable a, the worst case quantization of the sub-expressions (m*n) and (p*q) will be used. If you want better control over the quantization of these sub-expressions, you could slightly modify the code as follows:

```
tmp1=(m*n);
tmp2=(p*q);
a = b+tmp1+tmp2;
```

In this modified code, you can control the overall quantization of the sub-expressions by applying a quantize directive to tmp1 and tmp2. Further, if the quantization of one of the sub-expressions results in an overflow or an underflow, the MATLAB transcript message will specifically list the line number of the offending sub-expression. Otherwise it will be more difficult to find the problem in the original code.

Eliminating Overflows and Underflows

An *overflow* occurs when the magnitude of a number being assigned to a variable exceeds the number of bits allocated to the integer part of the fixed-point word. The excess in magnitude can happen with a negative number as well as a positive number. As shown below, if the Show Overflows project option is set true (the default) and an overflow occurs, the overflow is recorded in the Verify FixedPoint Report.

An *underflow* occurs when a very small fractional number gets rounded to zero. For example, if the number 0.00001365 gets assigned to a variable with only 4 bits allocated to the fractional part of the word, then the number is rounded to zero. Underflows are usually more common and less serious than overflows. However, to a DSP designer, a small number is very different then zero, especially when it is used in a downstream math operation like multiplication. As shown below, if the Show Underflows project option is set to true (the default) and an underflow occurs, the underflow is recorded in the Verify FixedPoint Report.

Verify FixedPoint Report

– Overflows

# of Overflows	Location	Quantizer	Overflow Value	
			Min	Max
8	test.m (28)	ufixed wrap floor [30 29]	548.238	987.32
4	test.m (33)	ufixed wrap floor [30 29]	2.41098	4.69541

– Underflows (rounded to zero)

# of Underflows	Location	Quantizer	Underflow Value	
			Min	Max
8	test.m (32)	ufixed wrap floor [12 12]	1e-006	1e-006

Table 5-3: **Verify Fixed Point Report**

How to Eliminate an Overflow

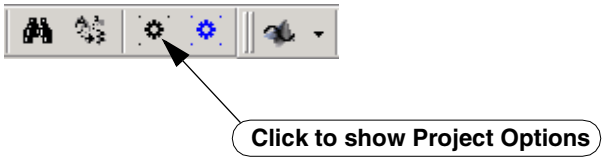
As previously explained, when a variable is auto-quantized by AccelDSP, the fixed-point word width is statically determined by analyzing the dynamic range of numbers being assigned to it. Consider the following code segment:

```
sum = 0;
for k = 1:NUMTAPS
    mult = indatabuf(k) * coeff(k);
    sum  = sum + mult;
end
```

Because the variable sum is used in the for loop and sum feeds back on itself, the quantization necessary to prevent an overflow cannot be statically determined. Depending on the number of loop iterations, the final value of sum may exceed the maximum magnitude of the quantized word. The solution is to run the Verify Fixed Point step and look for overflow messages in the Tcl Console transcript.

To eliminate an overflow, you should do the following:

1. If the transcript report is cluttered with too many underflow messages, you should turn off the underflow messages by clicking the Project Options icon as shown below:



   Click on the Advanced Mode button on the dialog box (lower-right corner), click on Output, then set Show Underflows to False.

2. Re-run the Verify Fixed Point step and look for overflow messages.

3. Open the generated fixed-point M-file to the line indicated in the report and find the offending variable.
4. Try increasing the bit allocation of the integer part of the word. You do this by going to the Generate Fixed Point Report and decreasing the bit allocation of the fraction part of the word. For example, if the bit allocation is [26 15], change the allocation to [26, 14], then re-run Verify Fixed Point.
5. Run the Verify Fixed Point step again. If the overflow message disappears, then you have fixed the problem. If not, reduce the fraction allocation by one bit, then repeat the process until the overflow message disappears.
6. Repeat this process for each variable that appears to be overflowing.

How to Eliminate an Underflow

As previously explained, an underflow happens when a very small number gets rounded to zero. Underflows are caused when there are not enough bits allocated to the fractional part of the fixed-point word. If in the process of fixing overflows you decrease the allocation of the fraction part too much, you may create an underflow situation.

Assuming that you have increased the bit allocation of the integer part of a word just enough to avoid overflows, you can now increase the bit allocation to the fraction part to eliminate the underflows (if necessary).

To eliminate an underflow, do the following:

1. Turn on the underflow messages if you previously turned them off.
2. Run the Verify Fixed Point step and look for underflow messages in the transcript.
3. Open the generated fixed-point M-file to the line indicated in the report and find the offending variable.
4. Go to the Generate Fixed Point Report and increase the number of bits allocated to the fractional part of the word; also increase the word length by the same amount. For example, if the bit allocation is [26 9], change the allocation to [27, 10].
5. Re-run Verify Fixed Point. If the underflow message disappears, then you have fixed the problem. If not, repeat the process of adding one bit to each quantizer property.

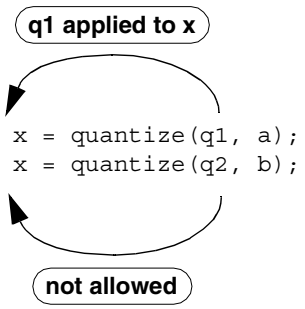
Rules for Quantization

The recommended approach to creating a fixed-point MATLAB design is to let the AccelDSP synthesis tool auto-quantize the floating-point design, then you can adjust the quantization by explicitly applying quantize directives to in-memory design objects. In some advanced designs, however, the designer may choose to explicitly embed MATLAB quantize functions in the floating-point source. The AccelDSP synthesis tool will honor these function calls as long as the following rules are followed:

Quantization Rule: All occurrences of any named variable must have the same quantization

Unlike MATLAB, the AccelDSP synthesis tool can apply only one quantization to all occurrences of a named variable in the design.

The following code is not allowed:



Once the quantization defined by q1 is applied to x, no other quantization (like q2) can be applied to x. In addition, the q1 source quantization of x will override any quantize directive that you may apply to x from within the AccelDSP synthesis tool.

Quantization Rule: A variable that is used as an input argument to a function cannot be explicitly quantized inside the body of the function

The following code is not allowed:

```
function y = my_func(a,b)
%define the q1 quantizer properties
q1 = quantizer('fixed','floor','wrap',[16,8]);
for n = 1:a
b = quantize(q1,b + 1);%quantization is applied to var on left side
end
y = b;
```

The AccelDSP synthesis tool must have the freedom to infer the quantization of input arguments with each call to the function. Therefore, an explicit quantization of the variable b inside the function body is not allowed.

The solution to the above restriction is to assign b + 1 to a temporary variable, then the quantization is assigned to the temporary variable. The following code is allowed:

```
function y = my_func(a,b)
%define the quantizer properties
q1 = quantizer('fixed','floor','wrap',[16,8]);
for n = 1:a
tmp = quantize(q1,b + 1);%quantization is applied to var on left side
end
y = tmp;
```

Using accel_probe to Observe the Fixed Point Effects on Fidelity

During a MATLAB simulation, it is not easy to observe the behavior of variables inside MATLAB functions because of their scope. By inserting probes into the MATLAB source using the accel_probe function, the process of gathering data and plotting that data for analysis is automated. You can now easily compare (both graphically and numerically) the behavior of a variable in fixed-point versus how it behaves in floating-point.

Note: The accel_probe functionality is not supported in the Hardware Co-Sim Flow.

Usage

1. Add the function accel_probe_plot to the end of the MATLAB script file. This function generates the plots at the end of the MATLAB simulation.
2. Add any number of accel_probe functions inside of the design function.

Design Example

Script file example:

```
clear all;
close all;
a = sin([0:pi/32:4*pi]);
for n = 1:length(a)
    c(n) = hw_design(a(n));
end
% Display plots
accel_probe_plot;
```

Design function example:

```
function c = hw_design( a )
b = [a a^2];
% Insert a probe on b
accel_probe(b);
c = a + b(1)*b(2);
% Insert a probe on c named 'c_output' and
% only capture the first 100 samples
accel_probe(c,'c_output',100);
```

Plot Descriptions

During floating-point simulation, accel_probe generates one plot for each probe that is added to the design. During fixed-point simulation the fixed-point results are plotted on top of the floating-point results. The largest absolute different between the floating-point and fixed-point results along with the sample it occurred is shown on the plot. The fixed-point plot for scalar probes also reports the signal to quantization noise ratio on that variable.

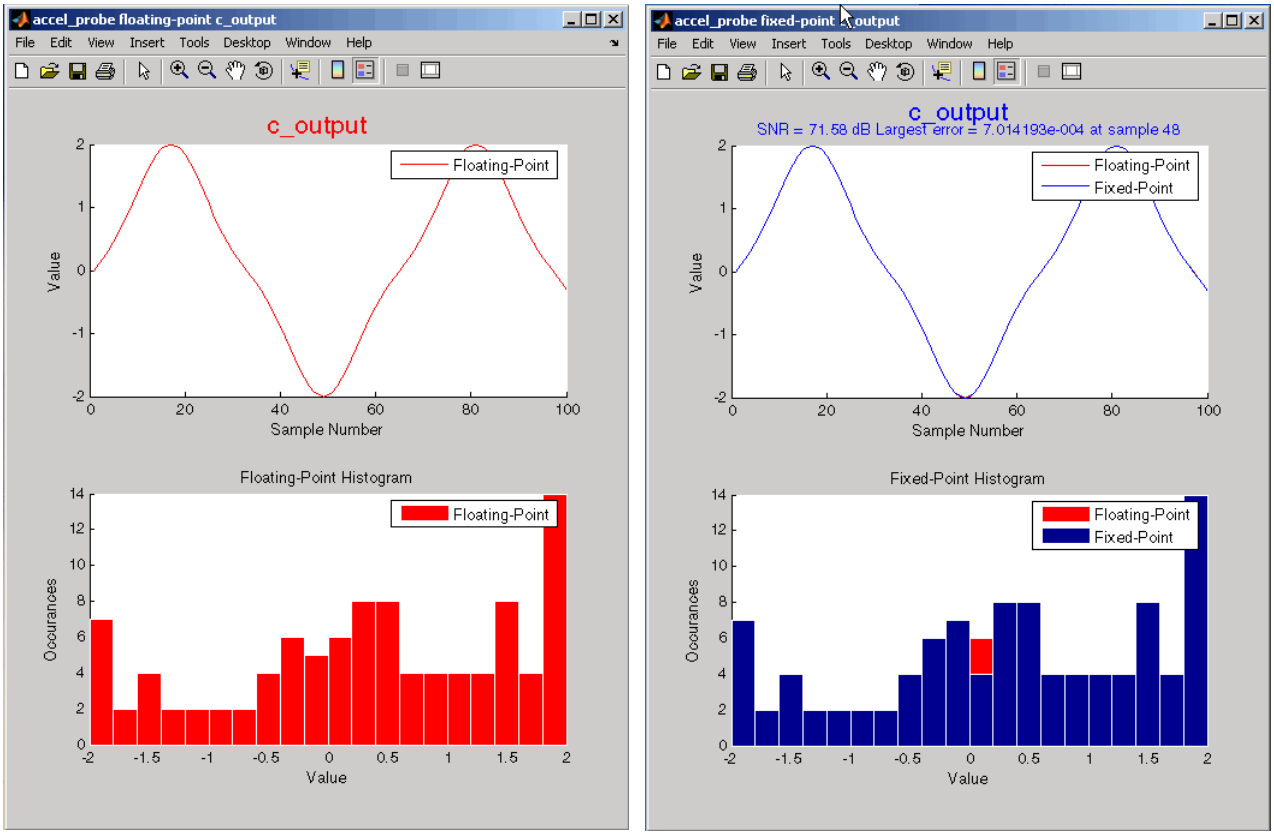
The following is used to calculate the SNR:

```
SNR = 20 * log10(RMS of floating-point signal / RMS of the error
floating vs. fixed-point noise)
```

Plot Descriptions

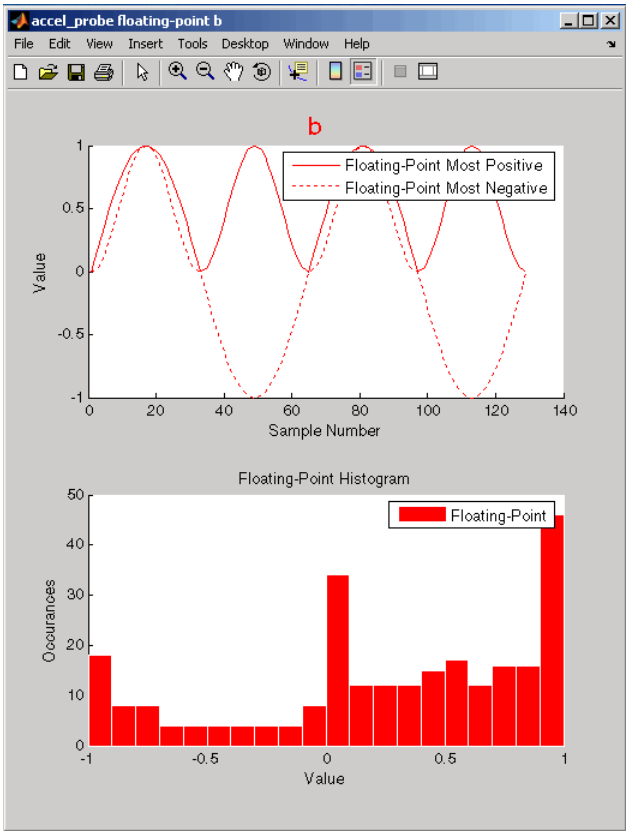
The plot example below is for probes on scalar variables. The left plot is after the floating-point simulation and the right plot is after fixed-point simulation. When the probe is inserted on any size array, there will be an additional floating-point and fixed-point line on the plot as shown in the two plots below. One line plots the most positive number in the array and the other plots the most negative. Therefore, on any sized array the two extreme

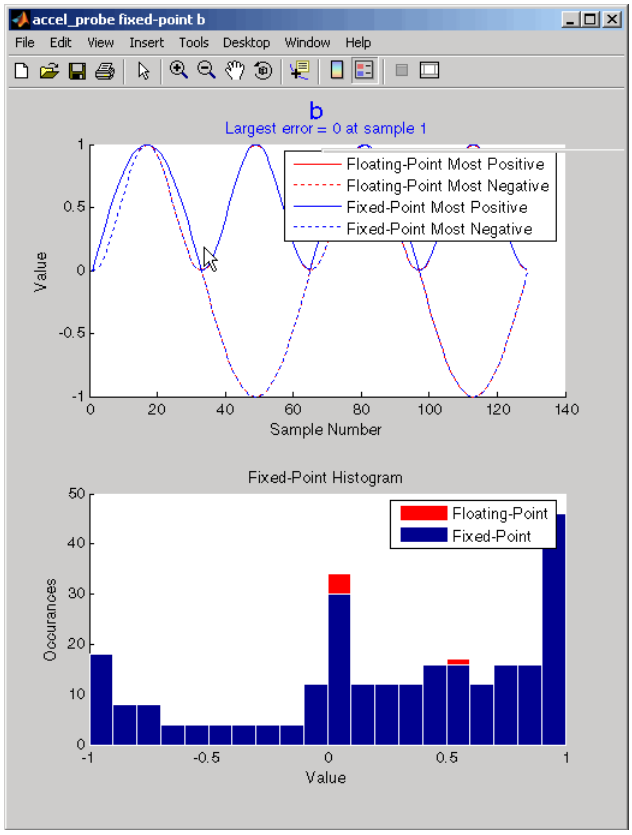
values in that array can be seen at any time. These two extremes provide key information when selecting quantizations for that array.



As before, the red traces are floating-point values. The solid trace is the most positive number in the array and the dashed trace is the most negative. The blue traces are floating-point values. The solid trace is the most positive number in the array and the dashed trace is the most negative. The largest differences between the floating-point and fixed-point

results along with the sample it occurred are shown for each extreme on the plot. The SNR is not reported for arrays.





Use Variations

accel_probe

When adding probes, the accel_probe function takes one required argument: the variable to probe. The second and third arguments are optional and specify a name for the probe point and the maximum number of points to collect for that variable, respectively. If the second argument is not specified, the probe name defaults to the same as the MATLAB variable name. If the third argument is not specified, the default is a maximum of 1000 sample points.

```
% Probe variable "a" and collect the default number of samples (1000)
accel_probe(a)
% Probe variable "a" and name it 'a1'
accel_probe(a,'a1')
% Probe variable "a", name it 'a2', and collect a maximum of 100 samples
accel_probe(a,'a2',100)
```

Note that depending on the algorithm, a sample point may not be the same as a streaming loop iteration. For example, if a probe is placed inside of a loop it will be called more that one time per streaming iteration.

If a variable is assigned a value at two different points in the same design, the data will be collected together unless a different probe name is used.

For example:

```
b = indata;
accel_probe(b);
b = b + 1;
accel_probe(b)
```

In the above design, a single plot of “b” will be displayed with the value of the two “b” probes interleaved.

```
b = indata;
accel_probe(b);
b = b + 1;
accel_probe(b,'b_second_probe')
```

In this case, the second probe of “b” will be plotted separately because it is given a different name “b_second_probe”.

accel_probe_plot

The function accel_probe_plot does not return a value and has no inputs. It generates a plot for every probed variable. If no probes are used, no plots will be generated.

Additional Details

During floating-point simulation, vectors are stored in a file called accel_probe_floating_data.mat inside the project directory. During fixed-point simulation, vectors are stored in a file called accel_probe_fixed_data.mat inside the AccelDSP synthesis tool FixedPointM directory.

Messages

The table below lists the messages that are generated by accel_probe and accel_probe_plot.

Table 5-4: **Messages from accel_probe and accel_probe_plot**

Message	Description
accel_probe b plotted	Generated after a plot is successfully generated for each probe.
Data sample buffer for accel_probe c_output reached maximum of 100 samples. Additional data will not be plotted	The maximum number of samples have been collected. The following shows how to collect 10,000 samples: accel_probe(a,'a',10000)
No accel_probes found. No data to plot	Accel_probe_plot was called but no internal variables were probed.
First argument is not a valid variable to probe. It must be a scalar or array variable.	The first argument is invalid.
Second argument is not a valid variable to probe. It must be a string that is a valid MATLAB variable name.	The second argument is invalid. The probe name is used as a MATLAB variable to store the data so it much be a valid MATLAB variable name.
Third argument is not a valid maximum count. It must be a positive integer.	The third argument is invalid.
Probe names have changed after floating-point vectors were stored. Please rerun the floating-point simulation.	After floating-point simulation was run a probe name was changed or a probe was added or deleted. Please rerun Verify –floatingpoint to update the floating-point vectors for comparison.
Probes have changed after floating-point vectors were stored. Please rerun the floating-point simulation.	The use or parameters to a probe changed after floating-point simulation was run. The floating-point vectors on the disk need to updated. Rerun Verify –floatinpoint.

XILINX®

Chapter 6

Exploring Hardware Architectures

The AccelDSP synthesis tool provides you with the ability to explore different hardware architectures without changing the "golden" floating-point model. You make architectural changes by applying AccelDSP directives to the in-memory design. The synthesized RTL model then reflects the architectural changes. The directives for a particular AccelDSP project are saved in an AccelDSP Design Directives (ADD) file. You can change the ADD file content by making changes from the GUI or you can edit the file directly with a text editor.

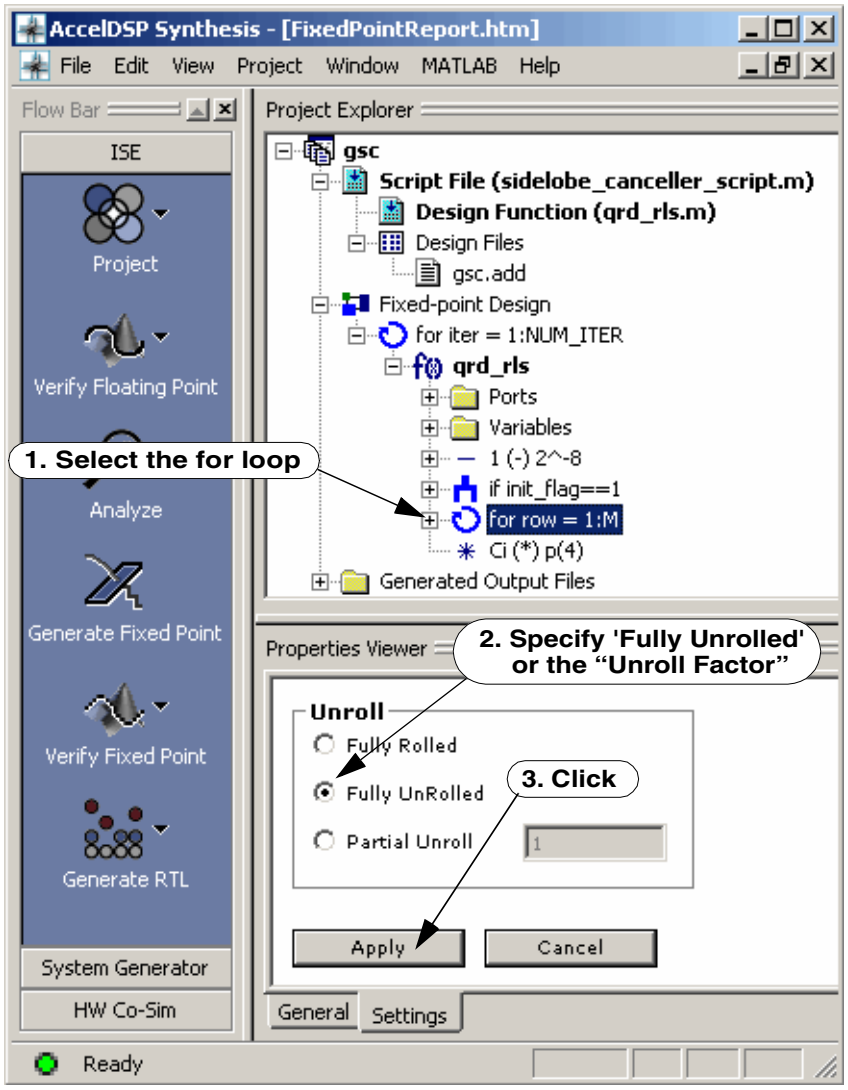
Unrolling a Loop to Increase Hardware Performance

A loop construct is synthesized into a similar loop structure in hardware where data is applied to a single data path a specific number of times (called the iteration interval). A loop with an iteration interval of sixteen, for example, results in hardware where data is applied to a single data path sixteen times before processing ends. If the loop construct is completely unrolled, then the AccelDSP synthesis tool builds a different hardware structure where sixteen data samples are applied simultaneously to sixteen identical parallel data paths. performance is increased sixteen times at a cost of increasing the hardware area by sixteen times.

If a full unroll consumes too much hardware, then a partial unroll may be a suitable trade off between area and performance. For a partial unroll, the unroll factor you specify must be an integer that is an integral divisor of the loop extent and should not exceed the loop extent.

How to Apply the Unroll Directive to a for Loop

To apply the unroll directive to a 'for' loop, select the target 'for' loop as shown below in the Project Explorer window and follow the directions to apply the unroll directive:



Unrolling All for Loops (Except a Few)

Rather than apply a separate unroll directive to every loop in the design (or almost every for loop), you can set the Project Option -unroll_for_loops to 'true' (1), then selectively apply an unroll directive to keep some of the loops rolled, or partially rolled. For example, consider the following command sequence:

```
SetProjectOption -unroll_for_loops 1
SetDirective {for n.design_hw.for i.for j} -unroll 1
```

The first command tells the AccelDSP synthesis tool to unroll all for loops in the design. The second command applies an unroll directive to the loop 'for j' which overrides the effects of the Project option. The parameter of the unroll directive is '1' which specifies to keep the loop fully rolled.

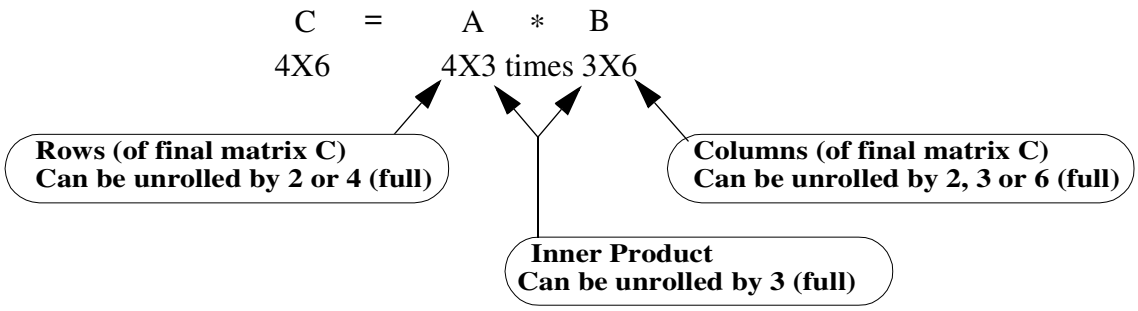
Unrolling a Matrix Multiply Operation

Consider the following line of code:

```
C = A*B
```

where A is a 4X3 matrix and B is a 3X6 matrix.

The resulting matrix C will be a 4X6 matrix, as shown below:

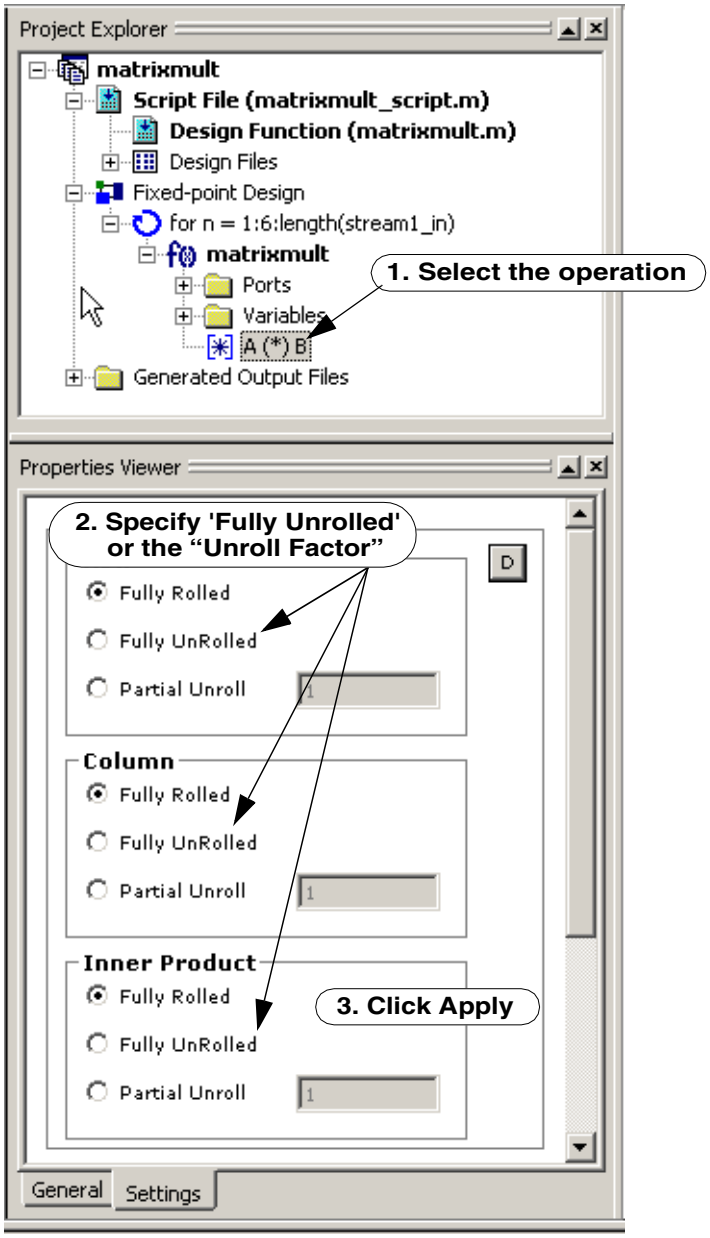


During the matrix multiply operation, the AccelDSP synthesis tool iterates over three portions of the resulting matrix while the multiplication operations are performed, the rows (4), the columns (6), and the inner product (3). These iterations can be considered implied loops and each loop can be partially or fully unrolled according to your specifications in the unroll directive.

Like a for loop, if a full unroll consumes too much hardware, then a partial unroll may be a suitable trade off between area and performance.The unroll factor you specify must be an integer that is an integral divisor of the loop extent. In this case, the loop extent of the final row is 4, so the loop can be partially unrolled by 2. The loop extent of the final column is 6, so the loop can be partially unrolled by 2 or 3. And the loop extent of the inner product is 3, so the inner product cannot be partially unrolled.

How to Apply the Unroll Directive to a Matrix Multiply

To apply the unroll directive to a matrix multiply, do the following:



Unrolling All Matrix Multiplies (Except a Few)

Rather then apply a separate unroll directive to every matrix multiply in the design (or almost every for matrix multiply), you can set the Project Option - unroll_matrix_multiplies to 'true' (1), then selectively apply an unroll directive to keep some of the matrix multiplies rolled or partially rolled. For example, consider the following command sequence:

```
SetProjectOption -unroll_matrix_multiplies 1
SetDirective {for n.design_hw.for i.for j} -unroll 1
```

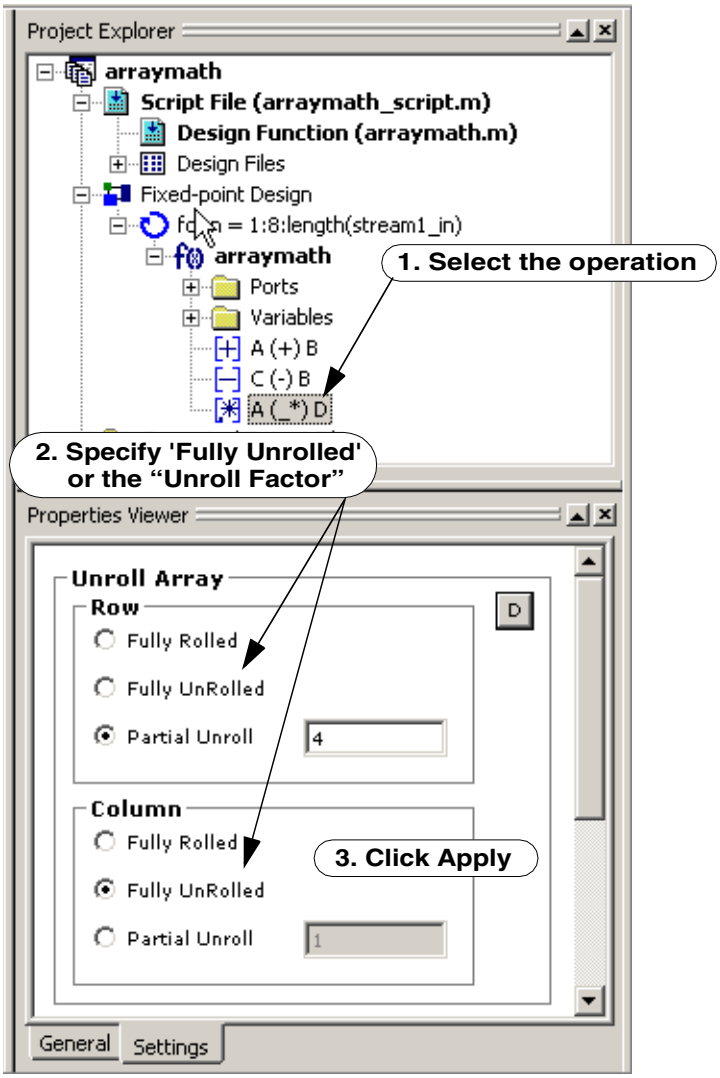
The first command tells the AccelDSP synthesis tool to unroll all for loops in the design. The second command applies an unroll directive to the loop 'for j' which overrides the effects of the Project option. The parameter of the unroll directive is '1' which specifies to keep the loop fully rolled.

Unrolling Array Math Operations

When you add, subtract or multiply two arrays, both arrays must be the same shape. You can unroll the resulting row and/or column to improve hardware performance. Unlike a matrix multiply, there is no inner product.

How to Apply the Unroll Directive to Array Math

To apply the unroll directive to an array multiply, do the following:



Unrolling All Array Math Operations (Except a Few)

Rather then apply a separate unroll directive to every array addition operation in the design (or almost every for array addition operation), you can set an 'unroll all' Project Option, then selectively apply an unroll directive to keep some of the matrix addition operations to keep them rolled or partially rolled. For example, consider the following command sequence:

Array Addition

```
SetProjectOption -unroll_array_adds 1
SetDirective -unroll {1 1} {for n.arraymath.A (+) B}
```

The first command tells the AccelDSP synthesis tool to unroll all array additions in the design. The second command applies an unroll directive to the array addition A + B which

overrides the effects of the Project option on that operation. The parameters of the unroll directive are '1 1' which specifies to keep the operation fully rolled.

Likewise, array subtraction and array multiplication can be handled in a similar manner:

Array Subtraction

```
SetProjectOption -unroll_array_subtracts 1
SetDirective -unroll {1 2} {for n.arraymath.C (-) B}
```

Array Multiplication

```
SetProjectOption -unroll_array_adds 1
SetDirective -unroll {3 5} {for n.arraymath.A (_*) D}
```

Mapping a Variable to Memory

If a MATLAB variable represents a large array, it is often more efficient to map the variable to embedded memory on the FPGA rather than let it be built from distributed logic. The AccelDSP synthesis tool allows you to map a variable to a dual-port synchronous RAM.

Automatic Mapping to Memory

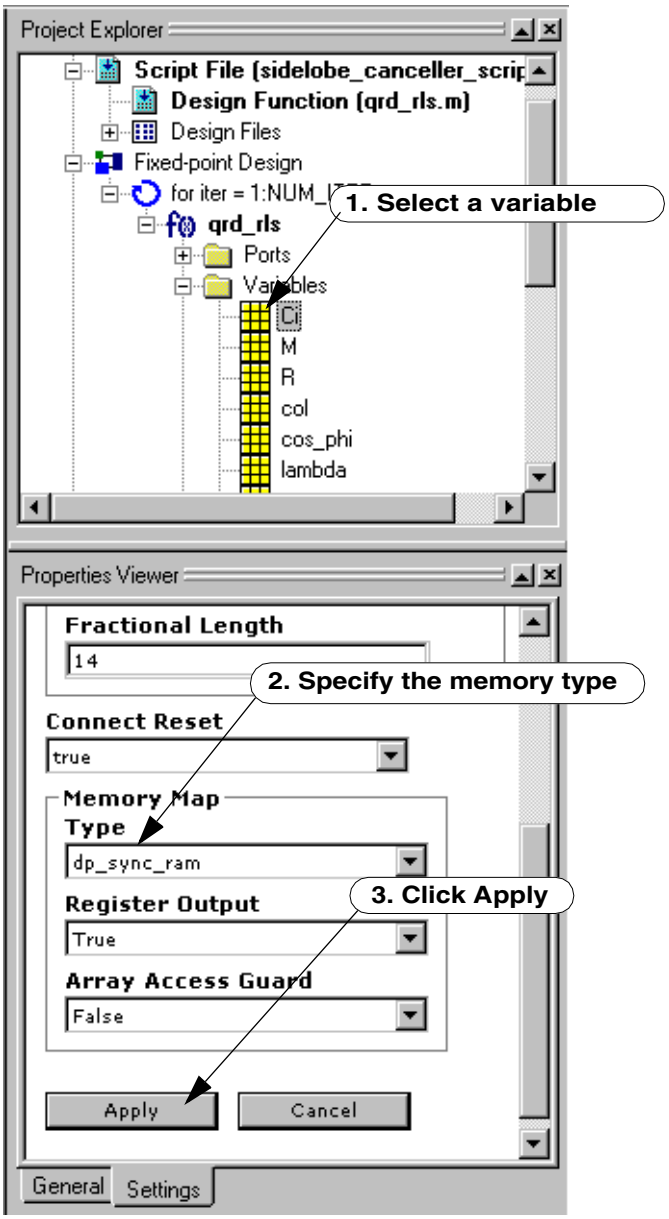
AccelDSP has a memmap_defaults project option called “ram_threshold”.automatically map arrays to dp_sync_ram if the threshold is a positive integer and the array size is greater than or equal to the threshold. The default “ram_threshold” is to 64:

A ram_threshold set to 0 means that there is no automatically mapping to RAM. Instead, the array is mapped to registers.

If an array has a memmap Directive set to -nomap and the array size meets the ram_threshold criteria, the array will not be automatically mapped to RAM. This directive allows you to selectively unmap an array that meets the ram threshold criteria while other qualifying arrays in the design will be mapped.

How to Apply the Memmap Directive to a Variable

To apply the memmap directive to a MATLAB variable from the AccelDSP GUI, do the following:



From the Tcl Console, a typical command line entry might be as follows:

```
SetDirective {for n.fir.Variables.coeff} -memmap dp_sync_rom
```

The Performance Trade-Off When Mapping to Memory

You may or may not wish to map a large array to memory depending on the design constraints. In general, when you map an array to memory, the down-stream RTL synthesis tool will map the specified variable into a structure that is optimized in terms of

area and speed for storing values. The main disadvantage of mapping to memory is that data elements in the memory cannot all be accessed at the same time. Since access to the memory is limited, it will take longer to perform the design algorithm if many reads from, or writes to, the array are required.

While the MATLAB code being synthesized does not have the notion of a system clock, the hardware circuit being generated does. If you map an array to a dual-port RAM, then at most one read and one write can happen in the same clock cycle. .

How Memory is Initialized

FPGA memory locations are initialized to zero by default. So, if a memory-mapped array is initialized to zero in MATLAB, the AccelDSP synthesis tool does not create any memory initialization hardware. If, however, the array is initialized with non-zero values, the AccelDSP synthesis tool generates memory initialization hardware. For example, assume the design function contains the following memory-mapped variable:

```
persistent A;
if isempty(A)
A = [1:2000];
end
```

In order to initialize the 2000 elements of A, the AccelDSP synthesis tool generates initialization hardware similar to the following:

```
simple_process_combinational_1:
process ( simple_process_state, ac_ram_init_counter ) is
  begin
    case simple_process_state is
      when sm_ram_init => case ac_ram_init_counter is
       when "00000000000" => dp_sync_ram_0_d <= accel_resize("1", 11);
       when "00000000001" => dp_sync_ram_0_d <= accel_resize("10", 11);
       when "00000000010" => dp_sync_ram_0_d <= accel_resize("11", 11);
        .....
        .....
        when "11111001110" => dp_sync_ram_0_d <= "11111001111";
        when others => dp_sync_ram_0_d <= "11111010000";
    end case; --ac_ram_init_counter
end process simple_process_combinational_1;
```

This hardware block will initialize the RAM on power-up.

Improving Performance by Removing the Array Access Guard

AccelDSP has a feature that guards against conflicts in the read and write access to array variables. The Array Access Guard is on (True) by default. When the Array Access Guard is on, AccelDSP does not pipeline the design as aggressively.

If your design has array variables and you want to improve the thoughput of the design, you can turn off the Array Access Guard to one or more array variables, then re-run the AccelDSP flow to see if the throughput improves.

If you receive an error message (E-VERIFY-0007) during the Verify RTL step, then the act of removing the Array Access Guard is causing a conflict.

The variable(s) with conflicts are identified in the simulation part of the AccelDSP log (accel.log). To resolve the conflist, you must re-apply the Array Access Guard to the offending variable(s).

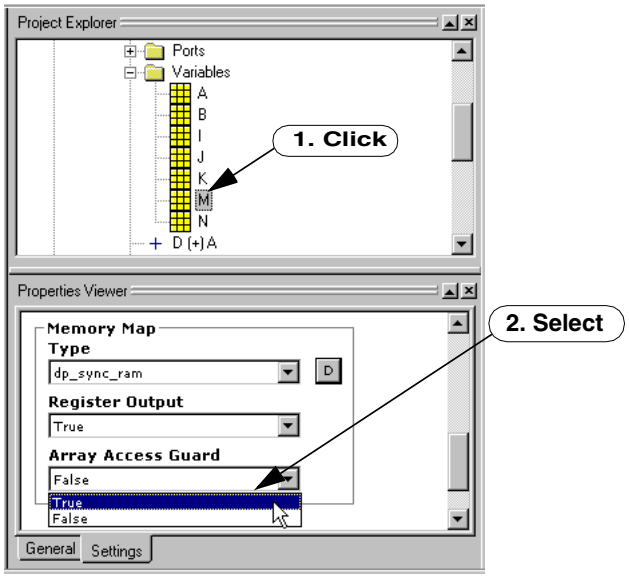
As an example, assume that you have several array variables in your design and you disable the Array Access Guard on each one to allow AccelDSP to more aggressively pipeline the design. This automatic pipelining may improve the design throughput, but may also cause simulation mismatches. You then re-run the AccelDSP flow and see the following message after the Verify RTL step:

```
#( E-VERIFY-0007): Simulating VHDL design: Modelsim Failed.  This is due
to a pipeline register in a feedback loop.
# Remedy: Enable the Array Access Guard for feedback variable(s)
indicated in the accel_verify.log file and rerun Generate -fixedpoint.
```

Following the directions in the message, you scroll up in the Transcript window and see the following message from the ISE Simulator:

```
# ** Warning: Encountered array conflict - Please enable the Array
Access Guard for variable 'M'.
```

To remedy the conflict, you then select the variable M in the Project Explorer window, as shown below, and set the Array Access Guard in the Properties Viewer window back to **True**.



Improving Performance with an Insertpipestage Directive

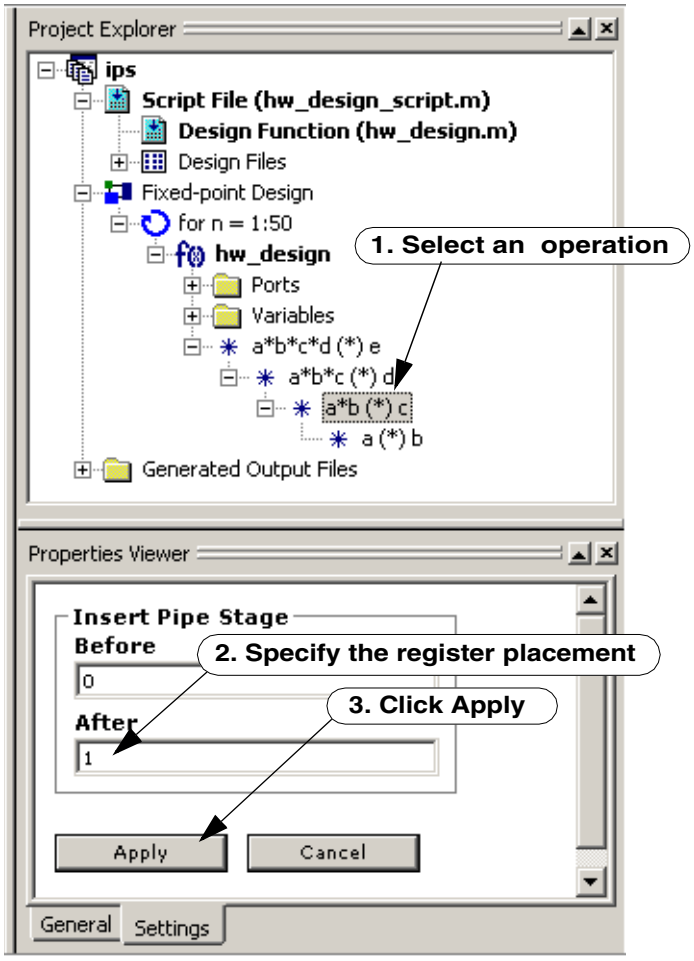
An RTL design is characterized by a coding style that specifies all the registers in the design and the combinational logic between them. The combinational logic between two registers is called a *path*. The path with the longest delay is called the critical path. The critical path determines the maximum frequency of the design.

The AccelDSP synthesis tool allows you to break a path into two parts (called stages) in order to reduce the amount of combinational logic in any one path. If the path is the critical path, this action increases the maximum frequency of the design.

For example, assume that you have a path where the expression y = a * b * c * d * e is executed in one cycle. You can increase the frequency of the design by inserting a pipeline register after the second multiplier. This breaks the original path into two new paths, each having two multipliers. Since the amount of logic is reduced in each new path, the maximum frequency is increased. The tradeoff is that the circuit Startup Clock Cycles is also increased from 1 to 2. If the insertpipestage directive causes the design to be out of balance, the AccelDSP synthesis tool will automatically insert additional registers in other places to re-balance the circuit.

How to Apply the Insertpipestage Directive to a Subexpression

To apply the InsertPipeStage directive to a subexpression, do the following:



InsertPipeStage Multiplier Design

The following example will further illustrate the effects of the insertpipestage directive. Consider the multiplier design shown below. Assume the the use of LogiCORES is turned off:

```
function [ y ] = simplemult(a_in,b_in,c_in,d_in,e_in);
persistent a b c d e;
if isempty (a)
a = 0;
b = 0;
c = 0;
d = 0;
e = 0;
end
y = a * b * c * d * e;
a = a_in;
b = b_in;
c = c_in;
d = d_in;
e = e_in;
```

The first time the function is called, the persistent variables are initialized to zero and the function returns zero. The second time the function is called, the previously received inputs are multiplied together and returned to output y.

Figure 6-1 shows the hardware circuit that is produced by the AccelDSP ISE Synthesis flow

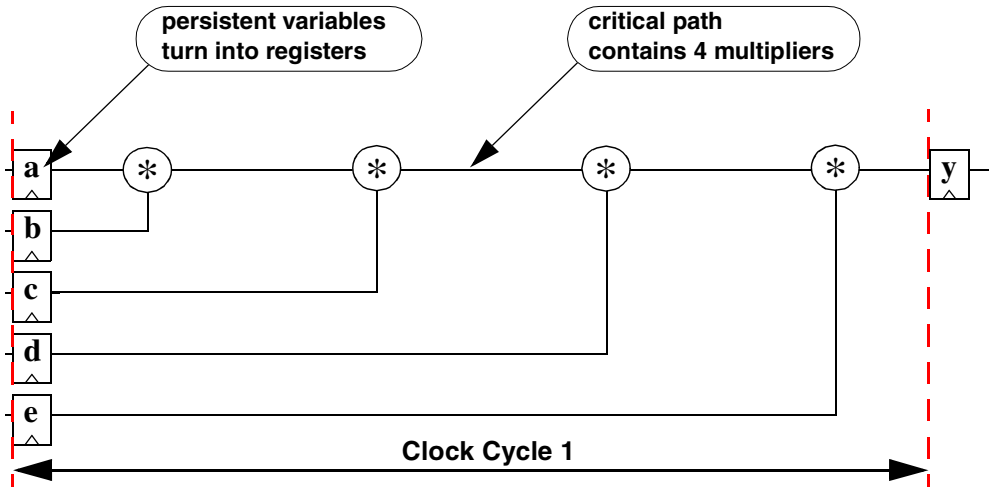


Figure 6-1: **Multiplier Design with No Pipe Stages**

Notice that the persistent variables a, b, c, d, e turn into registers on the inputs. The critical path contains four hardware multipliers. Although a result is returned in 1 clock cycle, the four multipliers keep the path delay long and maximum frequency low.

Inserting One Pipe Stage

Figure 6-2 shows how a pipeline register is inserted into the data path after an InsertPipeStage directive is applied to the second multiplier.

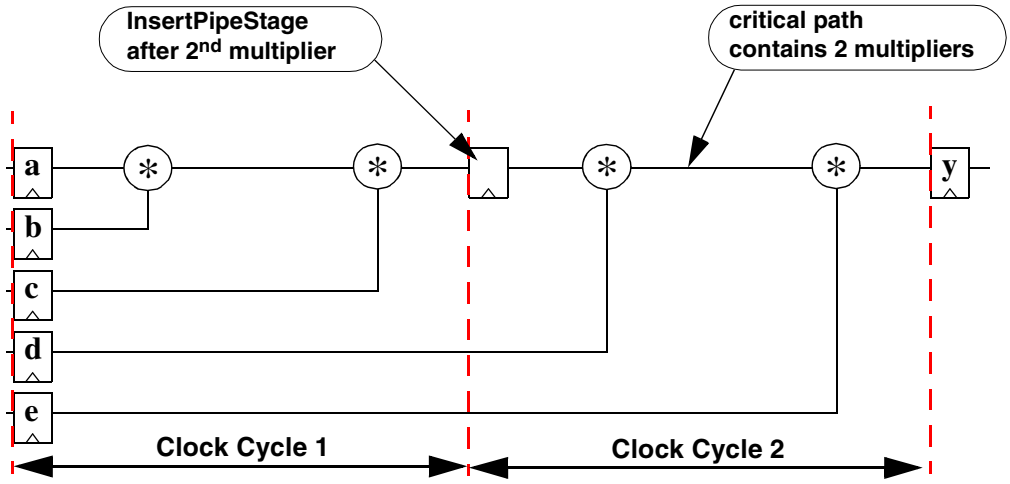


Figure 6-2: **Multiplier Design with One Pipe Stage**

Now, the longest path between registers contains only two multipliers, so the circuit can run at a much higher frequency. The tradeoff is that it takes an additional clock cycle for the result to be assigned to output y.

Inserting Two Pipe Stages

Figure 6-3 shows how a pipeline register is inserted into the data path after an InsertPipeStage directive is applied to the first multiplier.

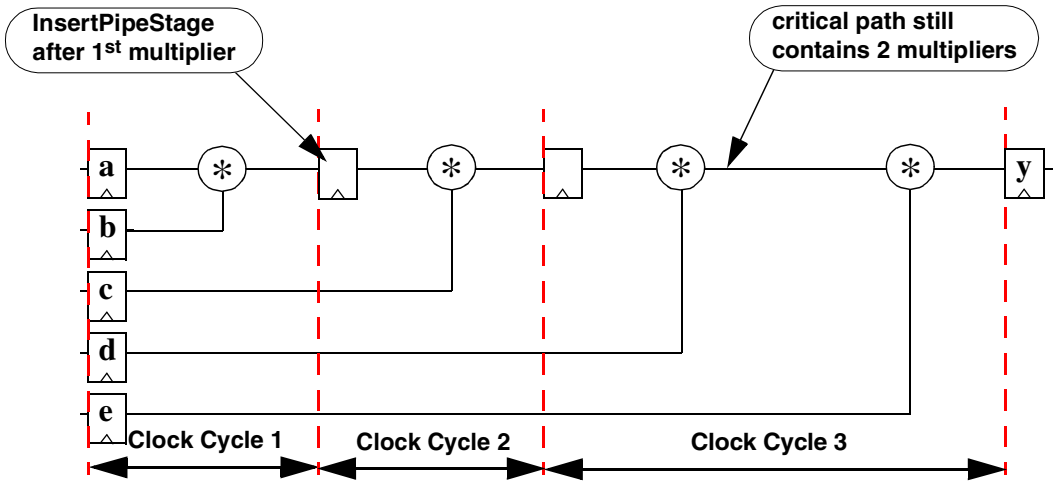


Figure 6-3: **Multiplier Design with Two Pipe Stages**

Although the path is again divided in two with one multiplier is each half, the operating frequency doesn't change. This is because the longest path between registers still contains two multipliers. This InsertPipeState directive has the effect of adding another clock cycle without increasing the frequency of the design.

Inserting a Third Pipe Stage

Figure 6-4 shows how a pipeline register is inserted into the path after an InsertPipeStage directive is applied to the third multiplier.

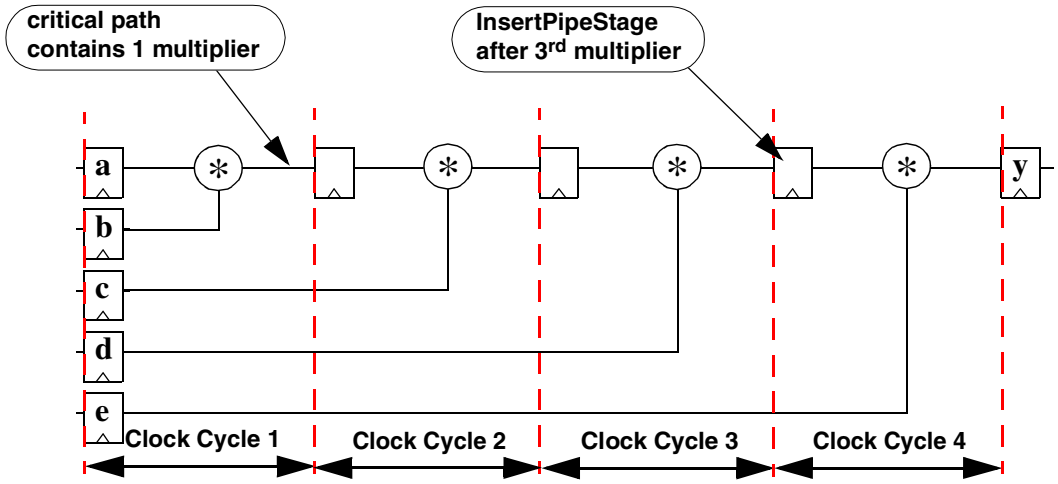


Figure 6-4: **Multiplier Design with three Pipe Stages**

Now the longest path between registers contains only one multiplier. This fully pipelined design will run more than three times faster than the original design.

Improving Performance with a Use_logicore Directive

This directive tells AccelDSP to use a LogiCORE for the specified operator in the design. For example, if you wish to use a LogiCORE for a particular operator in order to improve performance, you can target that operator in the design tree using this directive

You can also use this directive to override the Use LogiCOREs project option. For example, assume that the Use LogiCORE project options are set to true and you discover that the performance of a particular operator is degraded because of design feedback. You can use this directive to disable the use of a LogiCORE for this one operator. AccelDSP will use LogiCOREs on all eligible operators in the design, but will disable the use of a LogiCORE for this particular operator.

Mapping I/O Ports to Hardware Pins

Overview

You may have an FPGA hardware board and wish to use the AccelDSP synthesis tool to generate code that will include specific pin numbers on the IO ports that match the pin out on the hardware board. This section describes how you can do this from the AccelDSP user interface.

Mapping Ports to Pins in the XST to ISE Flow

1. Complete the AccelDSP flow through **Verify RTL**.

2. Create a text file named **pinout.xcf** and place it in the AccelDSP project directory. It should contain the names of the ports in your design along with a pin number for each port.

Figure 6-5: **Mappinig VHDL Ports to Pin Numbers**

As shown in Figure 6-5 and Figure 6-6, the base names of all of the ports can be seen in the generated VHDL or Verilog code. Constraints for vector ports are specified using the greater than and less than sign < > for each pin in the constraint file.

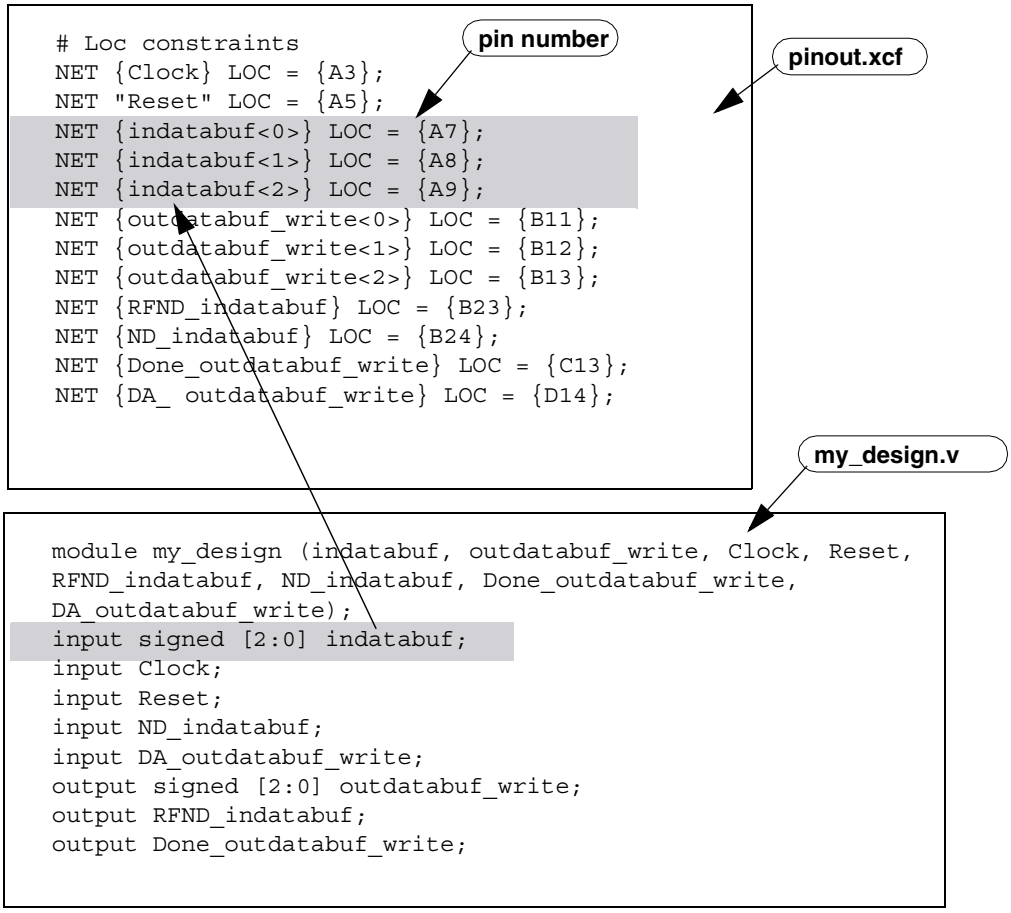


Figure 6-6: **Mapping Verilog Ports to Pin Numbers**

3. Open the Project Options pop-up window. This can be done by clicking on the black snow flake on the tool bar or by selecting **Project -> Project Options** from the pulldown menu. Under the "Target" heading set the "Constraint File" to point to your new **pinout.xcf** file.

When you run the **Synthesize RTL** and **Implement** steps, the Xilinx place and route tools will map the design ports to the pin numbers that you specified in the **pinout.xcf** constraint file.

Mapping Ports to Pins in the Synplify Pro to ISE Flow

1. Complete the AccelDSP flow through **Verify RTL**. If you selected Synplify Pro as the RTL synthesis tool, AccelDSP will automatically create a constraint file named **Synplicity.sdc** in the project directory.

2. Open the text file named **Synplicity.sdc** in the AccelDSP project directory. As shown in Figure 6-7, enter the names of the ports in your design along with a pin location for your specific FPGA. Other Synplicity attribute definitions may also be located in this file.

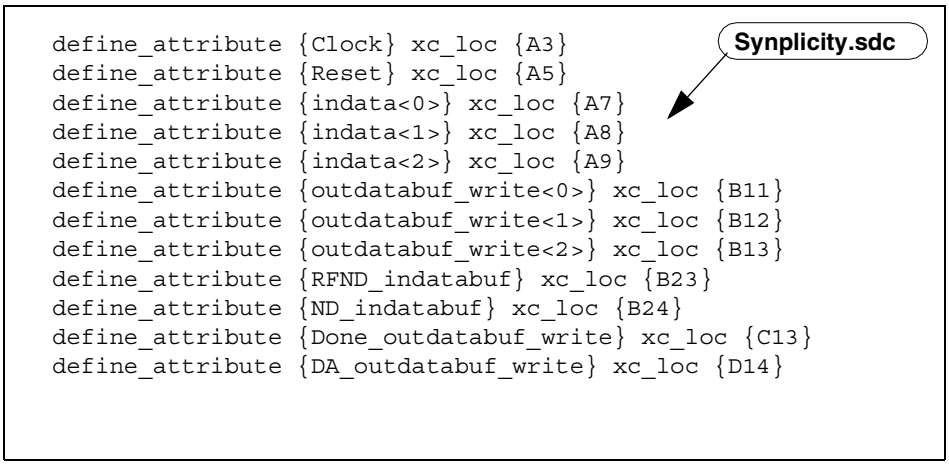


Figure 6-7: **Mapping Ports to hardware Pins in the Synplify Pro to ISE Flow**

The base names of all of the ports can be seen in the generated VHDL or Verilog code. Constraints for vector ports are specified using the greater than and less than sign for each pin in the constraint file. See Figure 6-5 and Figure 6-6 for examples.

Now when you run the **Synthesize RTL** and **Implement** steps, the place and route tools will map the design ports to the pin numbers that you specified in the **pinout.xcf** constraint file.

Additional Pin-Mapping Information

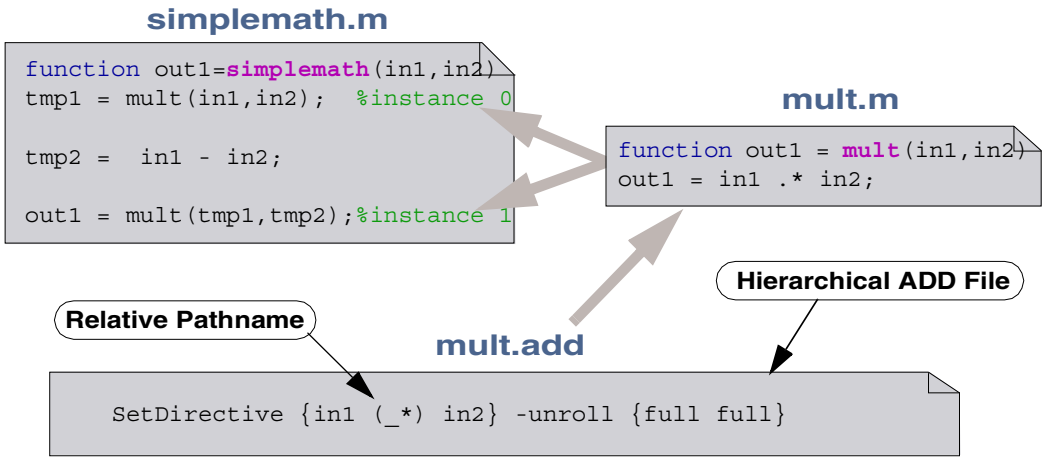
- Xilinx ISE generates a file in the ISE directory with a **.pad** extension that lists the I/O ports and the pin numbers for each port.
- You will not need to reload the MATLAB source or regenerate RTL when making changes to the pin out. Only the Synthesize RTL and Implement steps need to be run again for changes to take effect.
- When the AccelDSP project is saved, the location and name of the pin constraint file will be saved as part of the project for the next time you load this project.

Understanding Hierarchical Directives

Hierarchical directives are AccelDSP directives that are specified in an ADD file that is local to a particular function. These directives serve as default settings for each call to that function. A hierarchical directives file must be named the same as the function M-file it supports and must be placed next to the M-file in the same directory. Hierarchical directives can only be changed by opening the hierarchical ADD file and editing the content with a text editor.

Creating a Hierarchical Directives File

The illustration below shows the relationship between a design function named *simplemath* and a function mult that simplemath calls twice. The mult function has a related hierarchical ADD file that contains an unroll directive.

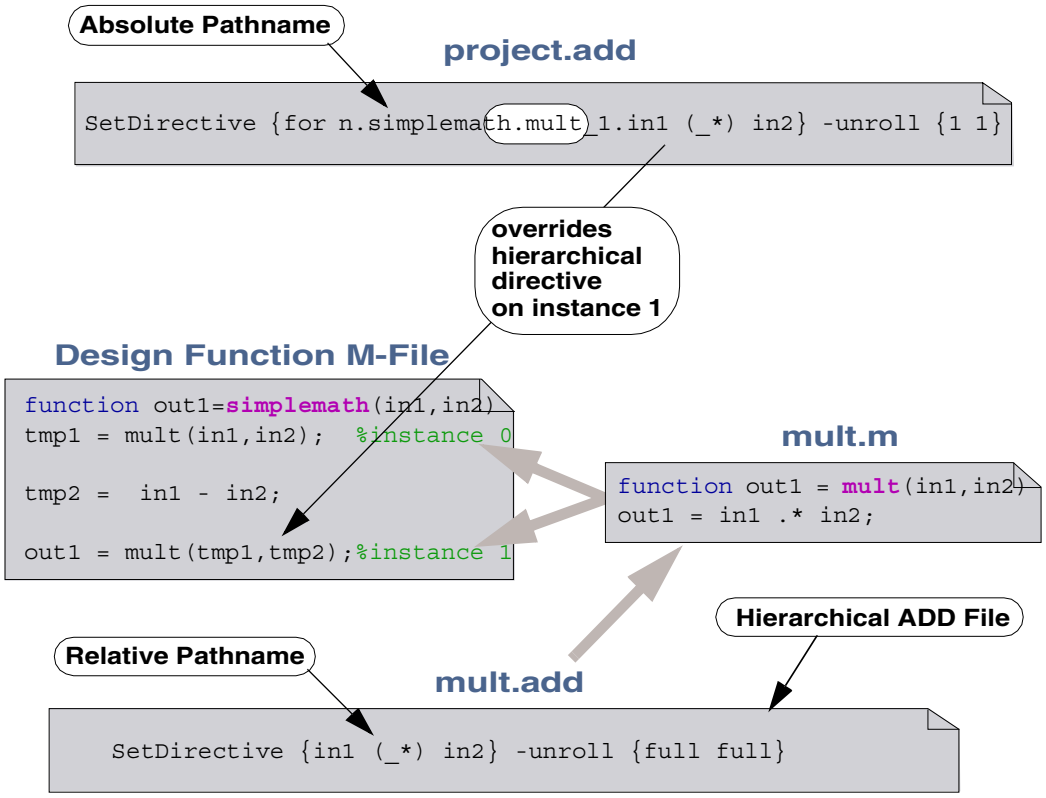


In the simplemath function, the mult function is called twice. The first call is referred to as “instance 0”, while the second call is referred to as “instance 1”.

The M-file for mult is shown along with its related hierarchical ADD file (mult.add). The unroll directive is applied to each call (instance) of the mult. Notice that the instance pathname in the SetDirective command is a relative pathname (relative to the function M-file) and is not a full (absolute) pathname that would be specified in the project-level ADD file.

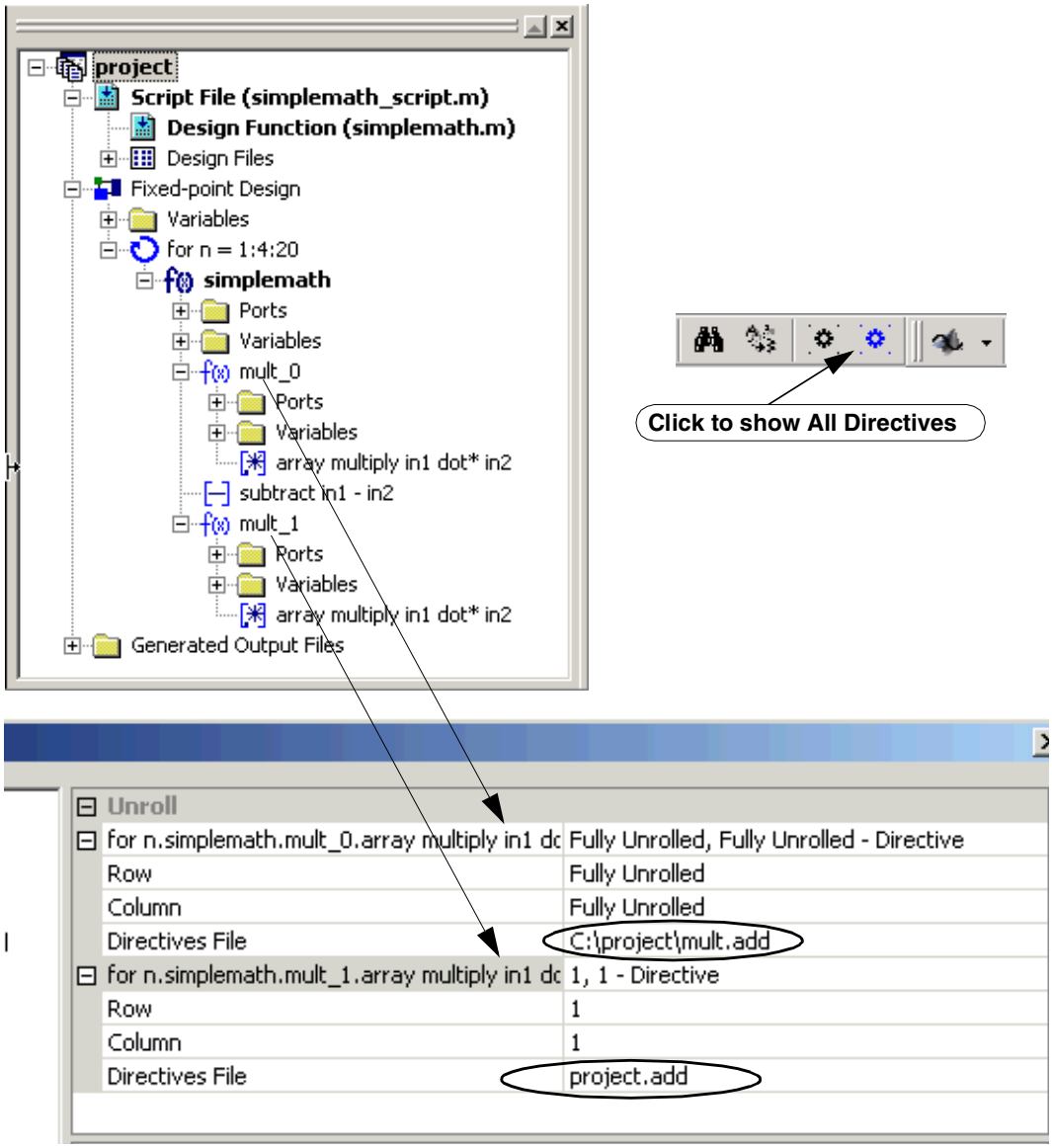
Overriding a Hierarchical Directive with a Project-Level Directive

Although a hierarchical directive cannot be changed from the AccelDSP GUI, a project-level directive can be used to **override** the effects of the hierarchical directive on an instance-by-instance basis.



As shown in figure above, the user has applied a project-level unroll directive to instance mult_1 after the Generate Fixed Point step.Notice that the absolute (full) pathname to mult_1 is specified in the project-level ADD file. This project-level directive overrides the hierarchical directive. If the user removes the project-level directive, the hierarchical directive on mult_1 will be re-applied.

The figure below shows how the simplemath design appears in the AccelDSP Project Explorer window. In the fixed-point model,the two instances of mult are renamed mult_0 and mult_1. Notice that when you click on the All Directives icon (shown below), a dialog box is opened and the source of the unroll directive on each instance is given. The unroll directive on instance mult_0 comes from the hierarchical ADD file located at pathname C:/project/mult.add. The unroll directive on instance mult_1 comes from the file project.add (the project-level ADD file).



Chapter 7

Interfacing to System Hardware

The Full Handshake Protocol

A MATLAB design that is synthesized by AccelDSP will typically be a design module that is part of a larger design on the FPGA. The flow of data into and out of the hardware ports is controlled by an interface protocol.

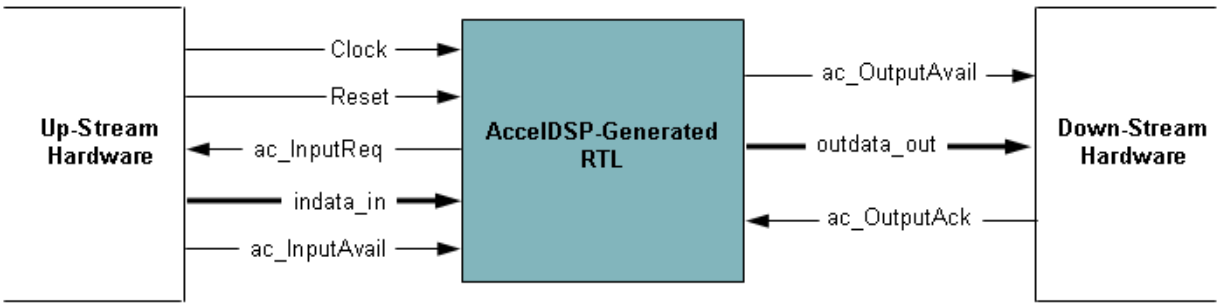


Figure 7-1: **The AccelDSP Handshake Interface**

Global Signals

Clock The AccelDSP hardware module has one clock input. Data transfers on each data port are synchronized to the clock.

Reset The global reset must be held active high for at least one clock cycle and returns all registers to a known state.

Full Input Handshake Protocol

Figure 7-2 illustrates the full input handshake protocol for a design module with a latency of 2.

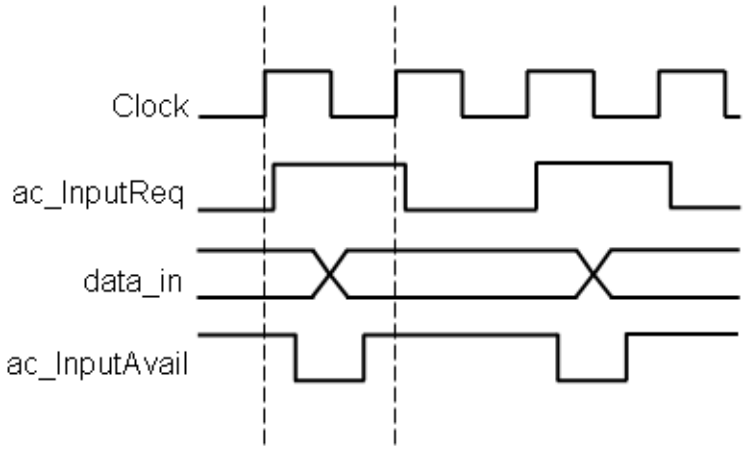


Figure 7-2: **Full Input Handshake Protocol**

The signal sequence is described as follows:

1. **ac_InputReq goes high (true)** This signal tells the up-stream hardware that the AccelDSP module is ready to capture new data on the next rising clock edge. The signal is typically set high soon after a rising clock edge.
2. **New input data is applied** The up-stream hardware applies new data to the input port(s). This new data will be captured on the next rising clock edge. The up-stream hardware should hold this data constant until ac_InputReq goes high again.
3. **ac_InputAvail goes high (true)** The up-stream hardware sets this signal high to tell the AccelDSP module that new data is ready for capture on the next rising clock edge.
4. **Next Rising Clock Edge** New data is captured by the AccelDSP hardware module.
5. **ac_InputReq goes low (false)** Means that the AccelDSP module is not ready to capture new data on the next rising clock edge. The up-stream hardware should hold the current data constant on the input ports until this signal goes high again. If the AccelDSP module holds this signal constantly high, then new data will be captured on every rising clock edge provided the sending device can send data that fast.

Full Output Handshake Protocol

Figure 7-3 illustrates the full output handshake protocol for a design module with a latency of 2. The signal sequence is described as follows.

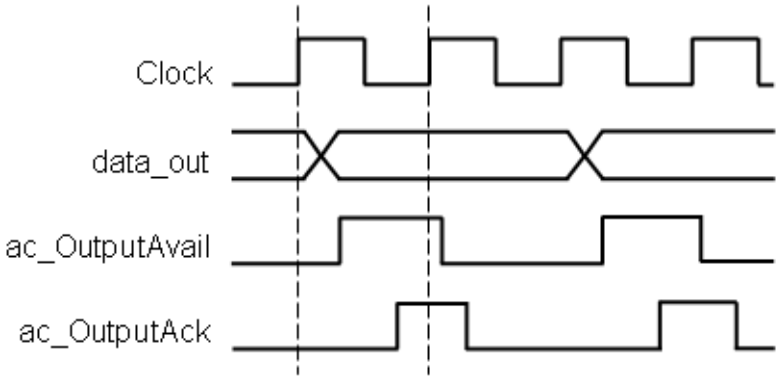


Figure 7-3: **Full Output Handshake Protocol**

1. **New output data is applied** Soon after a rising clock edge, the AccelDSP module applies new data to the output port(s).
2. **ac_OutputAvail goes high (true)** This signal tells the down-stream hardware that the new data is ready to be captured. Once this signal is set high, it will remain high until the down-stream hardware acknowledges the data capture by setting ac_OutputAck high (true).
3. **ac_OutputAck goes high (true)** This signal is controlled by the down-stream hardware and indicates that the data on the output port(s) of the AccelDSP module has been captured. If the down-stream hardware holds this signal constantly high, then the AccelDSP module will send data at the maximum possible rate, as governed by the module clock frequency and the Startup Clock Cycles of the computing algorithm.

Push-Mode Handshake Protocol

Push-Mode is the default interface protocol that is implemented for all designs that have a constant throughput. This interface protocol eliminates all up-stream handshake signal lines. Figure 7-4 illustrates the push-mode signal lines.

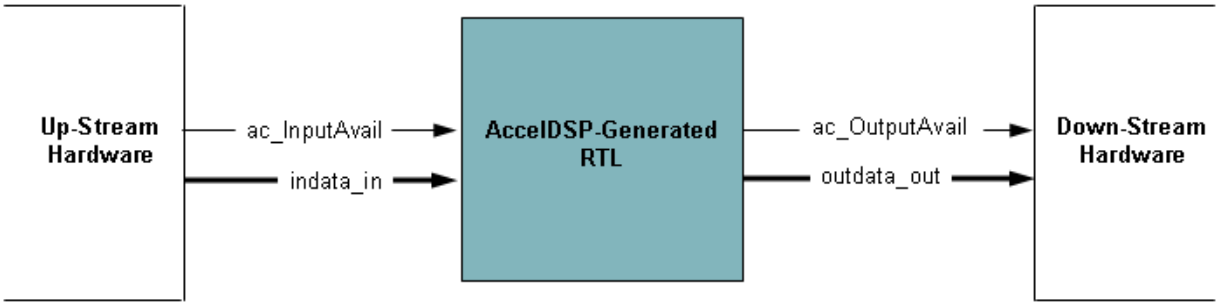


Figure 7-4: **Push-Mode Handshake Interface**

The push-mode interface has the following characteristics:

1. The interface is currently limited to designs with a constant throughput
2. The Generate RTL Report indicates if the push-mode interface is implemented
3. The interface allows for higher clock rates in some designs

Note: If AccelDSP is unable to implement the Push-Mode protocol, the 'Full' protocol is implemented and a warning message is displayed to alert the user. The Generate RTL Report also indicates what protocol is being used.

Single Registering the Outputs

By default, the outputs of the design are single registered. The figure below illustrates the timing of a typical push-mode design:

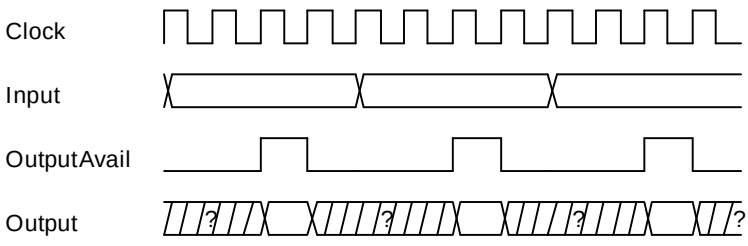


Figure 7-5: **Output Timing when the Outputs are Single Registered**

As shown above, the output is valid when the **OutputAvail** signal goes true and remains valid for one clock cycle.

Double Registering the Outputs

The figure below illustrates when the project option **Register Outputs** is set to 2:

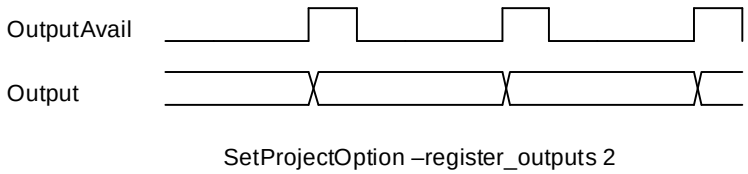


Figure 7-6: **Output Timing when the Outputs are Double Registered**

Notice that the output is valid when the **OutputAvail** signal goes true and remains valid until the **OutputAvail** signal goes true again. Although this adds one more cycle of latency to the design, it gives the down stream hardware more time to capture the output data.

Note: Double registering the output is only valid for constant throughput designs with a push mode interface.

XILINX®

Chapter 8

AccelDSP Commands

Table 8-1: **Alphabetical Command Summary**

Command	Description
Analyze	Analyze the input design.
Exit	Exit the AccelDSP synthesis tool.
ExportSimulink	Create a Simulink library block from the generated RTL model and export the Simulink block to the specified target directory.
DeleteDirective	Delete the specified directive from the specified design object.
DeleteGlobalOption	Delete the specified -ignorefunction global option.
Generate	Generate a design in the specified format.
GetDirective	Return the value of the specified directive.
GetGlobalOption	Return the current value of the specified global option(s).
GetProjectOption	Return the current value of the specified project option.
Help	Return help text on AccelDSP commands.
Implement	Invoke the pre-specified implementation tools on the generated RTL files, then place and route the design.
Project	Open the currently specified project if it exists; otherwise, create a new file in the current directory and give it the specified name.
SetDirective	Set a design directive on the specified in-memory design object.
SetGlobalOption	Set one or more global options.
SetProjectOption	Set one or more project options.
Synthesize	Call the pre-specified RTL Synthesis tool to read the RTL HDL design and generate a gate-level netlist.
Verify	Verify the RTL design using the generated Testbench and the specified HDL simulation tool.

The Tcl Command Interface

The AccelDSP Graphical User Interface is built on the Tcl language with special built-in extensions to provide behavioral synthesis functions. Each time you execute a function in the AccelDSP GUI, like selecting the Add Design menu item, a Tcl command is executed and transcribed in the Tcl Console window. If you choose, you can interactively enter Tcl commands at the Tcl Command Line prompt and execute a Tcl script which effectively runs the tool in “batch” mode.

Standard Tcl Commands

Standard Tcl commands provide the foundation for the AccelDSP synthesis tool command structure. With these commands, you can make variable assignments, handle lists and arrays, manipulate strings, do sorting and execute arithmetic operations. In addition you can execute program control functions like if/case/foreach/while statements. These generic Tcl commands are not documented here, but are fully documented in commercially available books like **Tcl and the Tk Toolkit** by John K. Ousterhout.

AccelDSP-Specific Tcl Commands

The AccelDSP synthesis tool has added a number of specialized Tcl commands to support the behavioral synthesis process. These commands are “built-in” and are executed the same as the standard Tcl commands. These commands are fully documented in the remainder of this chapter.

Analyze

Analyze the input design.

Example

```
An
```

Syntax

```
Analyze
```

Description

The Analyze command is the main mechanism for analyzing the design function.During the analysis, the tool traverses the design function and collects all the design data. A model of the floating-point design is then created as an in-memory data base.

Related Commands

Project
Generate
Verify
Synthesize
Implement
SetProjectOption
SetGlobalOption

DeleteDirective

Delete the specified directive from the specified design object.

Example

```
DeleteD -unroll {for iter.qam_design.adaptive_eq.for k1}
```

Syntax

```
DeleteDirective <instance path>
[-insertpipestage]Delete the insertpipestage directive from the
                    specified object.
[-memmap]Delete the memmap directive from the specified
        object.
[-connectreset]Delete the connectreset directive from the specified
              object.
[-quantize]Delete the quantize directive from the specified
        object.
[-shape] Delete the shape directive from the specified object.
[-unroll]Delete the unroll directive from the specified
        object.
```

Description

The DeleteDirective command removes the specified directive from the specified in-memory design object.

Related Commands

SetDirective
GetDirective

DeleteGlobalOption

Delete the specified -ignorefunction global option.

Example

```
DeleteG -ignorefunction {acos}
```

Syntax

```
DeleteGlobalOption
[-ignorefunction <function_name>]
```

Options

- -ignorefunction

Removes the “ignore” status from the specified function. The AccelDSP synthesis tool will now try to synthesize this function if it is found in the design.

Description

The DeleteGlobalOption command is used to override *-ignorefunction* options that were previously set from an AccelDSPInit.tcl file. For example, the AccelDSPInit.tcl file in the AccelDSP software tree may reside in a protected area on your network. You can place an AccelDSPInit.tcl file in your $HOME directory or the current directory with DeleteGlobalOption commands that will override the settings from the protected file.

Related Commands

SetGlobalOption
GetGlobalOption

Exit

Exit the AccelDSP synthesis tool.

Example

```
Exit
```

Syntax

```
Exit
```

Description

The Exit command causes the AccelDSP synthesis tool to gracefully close down the application. This command has a number of alias commands, so you can type Quit, quit, Bye, bye, and exit.

Related Commands

ExportSimulink

Create a Simulink library block from the generated RTL model and export the Simulink block to the specified target directory.

Example

```
ExportSimulink -directory {C:/my_simulink_block}
```

Syntax

```
ExportSimulink [-directory <directory_pathname>]
```

Description

The ExportSimulink command creates the Simulink library block from the generated RTL model, then exports the block to the specified target directory. Exporting the Simulink model must occur after the Generate RTL step.You include the new block in the Simulink Library Browser by adding the block's directory pathname to the MATLAB search path. The topic Adding AccelDSP MATLAB Functions to the MATLAB Search Path outlines the general procedure.

Related Commands

Verify

Generate

Generate a design in the specified format.

Example

```
Generate -fixedpoint
Generate -rtl
```

Syntax

```
Generate -fixedpoint
Generate -rtl
Generate -fixedpoint_c
Generate -simulink
Generate -system_generator
```

Description

The Generate command is part of the main synthesis flow. When you specify the -fixedpoint switch, the command generates a fixed-point MATLAB or C++ model from the in-memory floating-point design. When you specify the -rtl switch, the command generates an RTL (register-transfer-level) model in the format specified by the SetProjectOption -targetlanguage command (VHDL or Verilog or both).

After the Verify RTL, one or more of the following command can be executed:

Generate -fixedpoint_c

Generates a fixed-point C++ model of the design function.

Generate -fixedpoint_c

Generates a Simulink block from the design function.

Generate -simulink

Generates a System Generator block from the design function.

Generate -sytem_generator

Generates a Xilinx System Generator block from the verified RTL model.

Related Commands

Project
Analyze
Verify
SetProjectOption

GetDirective

Return the value of the specified directive.

Example

```
GetDir -designfunction
#for n.fir
```

Syntax

```
GetDirective
[-all]
[-designfunction]


GetDirective <instance path>
[-connectreset]
[-insertpipestage]
[-memmap]
[-quantize]
[-shape]
[-unroll]
```

Description

The GetDirective command returns the value of the specified directive.

From the AccelDSP GUI, you can quickly display the setting of all AccelDSP Directives by clicking the All Directives icon as show below:

Related Commands

SetDirective
DeleteDirective

GetGlobalOption

Return the current value of the specified global option(s).

Example

```
GetG -u -t
#true 4
```

Syntax

```
GetGlobalOption
[-default_overflow_mode]
[-device]
[-fixedpointlanguage ]
[-ignorefunction]
[-impltool]
[-maxconsolelines]
[-msglvl <messsage_ID>]
[-retiming]
[-register_inputs]
[-register_outputs]
[-signal_name_clock]
[-signal_name_input_available]
[-signal_name_input_request]
[-signal_name_output_acknowledge]
[-signal_name_output_available]
[-signal_name_reset]
[-simtool]
[-sync_reset]
[-synthtool]
[-tab_size]
[-targetlanguage]
[-technology]
[-use_tabs]
[-writeimplconfigfile]
```

Description

The GetGlobalOption command returns the value that is set for the specified global option(s).

Related Commands

SetGlobalOption
DeleteGlobalOption

GetProjectOption

Return the current value of the specified project option.

Example

```
GetProjectO -targetlanguage -technology
#VHDL VIRTEX-4
```

Syntax

```
[-constraint_file]
[-default_overflow_mode]
[-default_round_mode]
[-device]
[-directivesfile]
[-dontcare_value]
[-fi_objects]
[-fixedpointlanguage]
[-frequency]
[-impltool]
[-interface_protocol]
[-memmap_defaults]
[-message_level_info]
[-message_level_warn]
[-package]
[-pnr_effort]
[-quantizer_max_constant_fractional_length]
[-quantizer_max_input_fractional_length]
[-register_inputs]
[-register_outputs]
[-replaceconstantmults]
[-retiming]
[-scalarize_io]
[-scriptfile ]
[-show_overflows]
[-show_underflows]
[-signal_name_clock]
[-signal_name_input_available]
[-signal_name_input_request]
[-signal_name_output_acknowledge]
[-signal_name_output_available]
[-signal_name_reset]
[-simtool]
[-speed]
[-sync_reset]
[-synth_auto_constrain_io]
[-synth_enable_io_insertion]
[-synth_enable_pipelining]
[-synth_fanout_limit]
[-synth_resource_sharing]
[-synthtool]
[-targetlanguage]
[-tb_max_errors]
[-tb_output_latency]
[-technology]
[-unroll_array_adds]
```

```
[-unroll_array_multiplies]
[-unroll_array_subtracts]
[-unroll_for_loops]
[-unroll_matrix_multiplies]
[-use_logicores]
[-writeimplconfigfile]
```

Description

The GetProjectOption command returns the current value of the specified project option.

From the AccelDSP GUI, you can quickly display the setting of all AccelDSP Project Options by clicking the Project Options icon as show below:

Related Commands

SetProjectOption
Project

Help

Return help text on AccelDSP commands.

Example

```
Help ^Set
```

Returns help text for all commands that start with the letters “Set”.

```
Help Option$
```

Returns help text for all commands that end with the letters “Option”.

```
Help .*Project.*
```

Returns help text for all commands that contain the substring “Project”.

Syntax

```
Help
[-commands]
[-all]
[<regular expression>]
```

Options

- -commands

Return a list of all available AccelDSP-specific Tcl commands.

- -all

Return help text on all available AccelDSP-specific Tcl commands.

- <regular expression>

Return help text on the commands specified by the Tcl regular expression. You can think of a regular expression as a pattern matching template using special wildcard characters. See the examples above.

Description

The Help command returns help text on the specified AccelDSP commands.

Implement

Invoke the pre-specified implementation tools on the generated RTL files, then place and route the design.

Example

```
Implement
```

Syntax

```
Implement
```

Description

The Implement command runs a Tcl script that invokes the pre-specified implementation tools on the RTL design, and does place and route. Many output files are generated, including an HDL simulation model of the implemented design and a configuration file containing the bit stream that configures the FPGA hardware.

Related Commands

Project
Analyze
Generate
Verify
Synthesize
SetProjectOption
SetGlobalOption

Project

Open the currently specified project if it exists; otherwise, create a new file in the current directory and give it the specified name.

Example

```
Project -open {C:\fir\project.acc}
Project -new {C:\my_project\project.acc} -sourcefiles
{{E:\design.m}{E:\design_script.m}}
```

Syntax

```
Project
[<project_file_pathname>]
[-open <project_file_pathname>]
[-new <project_file_pathname>] [-sourcefiles <list_of_files>]
[-copyandopenproject {{project_folder_pathname} <space>
{destination_folder_pathname}}]
[-save]
[-close]
```

Options

- <project_pathname>

Open the currently specified project if it exists; otherwise, create a new project file in the current directory and give it the specified name.

- -open <project_file_pathname>

Open the project that is defined by the specified project file.

- -new <project_file_pathname>

Create a new project file using the specified name. The parent directory of the project file is considered the project directory.

- -sourcefiles <list_of_files>

May only be used with the -new option. Specifies a list of one or more files to be copied into the destination directory along with the newly created project file. All files must be specified as absolute pathnames. For example: {{E:\design.m} {E:\design_script.m}}

- -copyandopenproject

This option is primarily used by the AccelDSP GUI to copy a project from the AccelDSP examples directory to another specified location. The first item in the Tcl list is the absolute pathname of the project directory to be copied. The second item in the Tcl list is the absolute pathname of the destination location. A space must separate these to items in the list.

- -save

Update and overwrite the current project (.acc) file and design directives (.add) file on disk.

- -close

Close the currently open project.

Description

The Project command is the main mechanism for creating, opening and closing a project.

Related Commands

Analyze
Generate
Verify
Synthesize
Implement
SetProjectOption
SetGlobalOption

SetDirective

Set a design directive on the specified in-memory design object.

Example

```
SetDirective {for iter.qam_design.adaptive_eq.for k1} -unroll 4
SetDirective {for n.matrixmult.A (*) B} -unroll {2 3 full}
SetDirective {for n.matrixmult.A (*) B} -insertpipestage {before=1
after=2}
```

Syntax

```
SetDirective
[-designfunction <design_function_pathname>]

SetDirective <instance_pathname>
[-connectreset 1 | 0]
[-insertpipestage {before=<integer> | after=<integer> | |
{before=<integer> after=<integer>} | [enable={1|0}]}
{[-memmap dp_sync_ram |nomap]{array_access_guard 1 | 0}
{register_output 1 | 0}}
[-quantize <quantize>]
[-shape {<rows> <columns>}]
[-unroll(loop) <unroll_factor>]
[-unroll(matrix mult) {<row_factor> <col_factor> <inner_prod_factor>} ]
[-use_logicore 1 | 0]
```

Options

- -connectreset

Applied to a variable. This is an advanced directive that should only be used by expert hardware designers. The directive causes the target variable to be included or removed from the RTL HDL reset clause. In many cases, when the reset is removed, the down-stream RTL synthesis tool maps the variable to a shift register which saves on hardware. A typical entry might be as follows:

```
SetDirective {for n.fir.Variables.tap_delay} -connectreset 0
```

The target variable must be a persistent variable that is initialized to zero. This option is useful where a whole word is shifted, such as a tap delay line, where this shift operation can be mapped to dedicated shift register logic on an FPGA.

- -designfunction

Specifies the name of the top-level design function in your MATLAB script file. In a valid MATLAB design, a top-level design function must reside inside the streaming loop. The AccelDSP synthesis tool attempts to automatically identify this function. If it cannot, the AccelDSP GUI prompts the user to identify the top-level design function, then captures the setting by entering a SetDirective -designfunction command in the project file.

NOTE: The following directives are applied to in-memory design objects and require an <instance_pathname> as part of the command specification.

- -insertpipestage

Applied to a subexpression. This is an advanced directive that can be used to break critical paths in the generated hardware in order to increase circuit performance. Typically, you'll run the design through implementation to determine the maximum frequency. If the frequency is unsatisfactory, you can use the GUI to set insertpipestage directives on subexprssions, then re-run the flow to see if the performance increases.

A typical directive entry might be as follows:

```
SetDirective {for n.hw.a*b (*) c.a (*) b} -insertpipestage {after=1}
```

For example, assume that you have a design where the expression z = a*b*c is executed in one cycle and this expression is in the critical path. You can increase the frequency of the design by inserting a register after the first subexpression a*b. This places the two multipliers in two different cycles, thus increasing the frequency. The tradeoff is that the circuit Startup Clock Cycles is also increased from 1 to 2. If the insertpipestage directive causes the design to be out of balance, the AccelDSP synthesis tool will automatically insert additional registers to re-balance the circuit.

The optional enable parameter allows you to disable a hierarchical directive that was generated from another source. Assume that you include as part of your design code that has a hierarchical directive applied to a multiplier. Assume also that you want AccelDSP to use a LogiCORE for the multiplier and that you want the LogiCORE to use "auto" for the latency. Since you cannot delect a hierarchical directive, you can effectively disable the hierarchical directive by applying an insertpipestage directive with the "enable" parameter set to false.(0). The directives might look similar to the following:

```
SetDirective {for n.hw.a*b (*) c.a (*) b} -use_logicore 1
SetDirective {for n.hw.a*b (*) c.a (*) b} -insertpipestage {enable=0}
```

- -quantize

Applied to a variable. This directive is primarily used by the GUI to allow the user to change the quantizer properties on a specific variable in order to minimize quantization error in the generated fixed-point model. A typical command entry is as follows:

```
SetDirective {for n.fir.Variables.tmp} -quantize {fixed floor wrap 16
15}
```

The quantization must be specified in the standard order with the Mode specified first (fixed or ufixed), the Round Mode specified second (floor only), the Overflow Mode specified third (wrap or saturate) and the Word length/Fraction length specified last.

- -memmap

Applied to a variable. Maps the specified variable to a dual-port synchronous RAM. If no_map is specified, then the variable will be mapped to registers in the fabric. The AccelDSP synthesis tool generates a generic RTL model that the down-stream RTL synthesis tool uses to infer the proper technology-specific RAM.

A typical entry might be as follows:

```
SetDirective {for n.fir.Variables.coeff} -memmap dp_sync_ram
```

The 'nomap' option is typically used to override a memmap directive that may have been previously set from a hierarchical directives file.

  - array_access_guard

    When set to 1 (the default), the guard is in place. This provides the maximum protection for read-write operations. When the parameter is set to 0 (false), AccelDSP pipelines the states between the array Read-Write operations more aggressively to minimize processing cycles. This may result in simulation mismatches during the Verify RTL step.

- register_output

  When set to 1 (true), a register is inserted at the output of every memory read operation. In many cases, this will shorten the critical path and increase the maximum frequency. However, if a memory read operation is in a feedback loop, a warning message is issued indicating that bad logic may result. In this case, it is best to turn off this project option or override the setting on this particular variable using a memmap directive with the output register parameter turned off.

- -shape

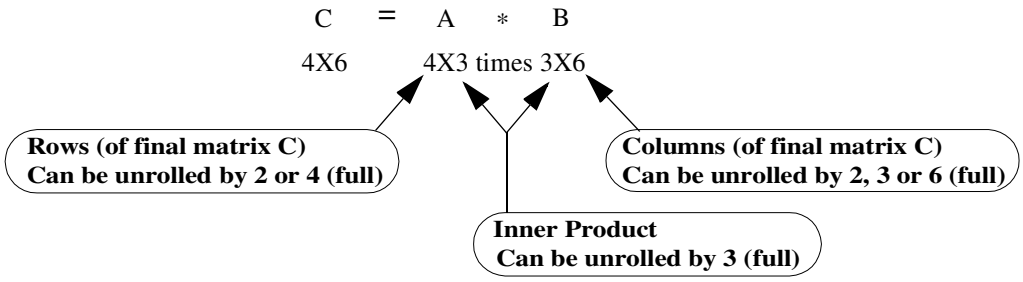
Applied to a variable. The MATLAB programming environment is an interpretive environment where the type and shape of a variable is determined at run time. The shape of a variable is specified as <rows> and <columns>. When a MATLAB program is analyzed by the AccelDSP synthesis tool, the tool attempts to determine the shape of each variable. If it cannot, the AccelDSP GUI prompts the user to enter the shape of a variable. The setting is then recorded as a SetDirective -shape command in the project file. A typical entry might be as follows:

```
SetDirective {for n.fir.Variables.coeff} -shape {1 16}
```

- -unroll

Applied to a loop. A looping construct in HDL is synthesized into a similar structure in hardware where data is applied to a datapath a specific number of times (called the iteration interval). A loop with an iteration interval of sixteen results in hardware where data is applied to a datapath sixteen times before processing ends. Normally, this takes sixteen clock cycles. If the loop is completely unrolled, then the AccelDSP synthesis tool builds a different hardware structure where sixteen data samples are applied simultaneously to sixteen identical parallel datapaths. Hardware Clock Cycles Per Design Function Call is reduced from sixteen to one. performance is increased sixteen times at a cost of increasing the hardware area by sixteen times. If the gain in size is too great, then partially unrolling the loop may be an suitable tradeoff.

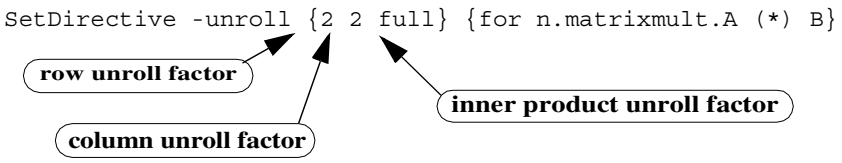
Applied to a matrix multiply. Consider the following line of code C = A*B where A is a 4X3 matrix and B is a 3X6 matrix. The resulting matrix C will be a 4X6 matrix.



During the matrix multiply operation, the AccelDSP synthesis tool iterates over three portions of the resulting matrix while the multiplication operations are performed, the rows (4), the columns (6), and the inner product (3). These iterations can be considered implied loops and each loop can be partially or fully unrolled according to your specifications in the unroll directive.

Like a for loop, if a full unroll consumes too much hardware, then a partial unroll may be a suitable trade off between area and performance. As with the partial unroll for a for loop, the unroll factor you specify must be an integer that is an integral divisor of the loop extent. In this case, the loop extent of the final rows is 4, so the loop can be partially unrolled by 2. The loop extent of the final columns is 6, so the loop can be partially unrolled by 2 or 3. And

the loop extent of the inner product is 3, so the inner product cannot be partially unrolled. A typical command line entry for the above example might be as follows:



Applied to an Array Math Operation. When you add, subtract or multiply two arrays, both arrays must be the same shape. You can unroll the resulting row and/or column to improve hardware performance. Unlike a matrix multiply, there is no inner product. The following are typical commands that apply a unroll directive to an array math operation.

```
SetDirective -unroll {2 4} {for n.arraymath.a (+) b}
SetDirective -unroll {3 5} {for n.arraymath.c (-) b}
SetDirective -unroll {full full} {for n.arraymath.a (_*) d}
```

- -use_logicore

  Applied to an operator. Tells AccelDSP to use a LogiCORE for the specified operator in the design. This directive overrides the use_logicore project option. For example, if the use_logicore project option is set to true (1) and this use_logicore directive is set to false (0) on a particular operator, then AccelDSP will use LogiCOREs on all eligible operators in the design, but will disable the use of a LogiCORE for this particular operator.
  If an insertpipestage directive is also applied to the target operator, then AccelDSP will use the values of the insertpipestage directive to determine the latency of the LogiCORE. For example, consider the following combination of directives:

  ```
  SetDirective -use_logicore {for n.arraymath.a (_*) d} 1
  SetDirective -insertpipestage {for n.arraymath.a (_*) d} {before=1}
  {after=2}
  ```

  The `use_logicore` directive tells AccelDSP to use a LogiCORE for the operator that multiplies array `a` and array `d`. Since the insertpipestage directive specified a total of `3` registers, the latency of the `mult` LogiCORE to set to `3`. If the number of registers specified by the insertpipestage directive is set to zero, then then latency of the LogiCORE is set to zero.

Description

Although you can execute a SetDirective command from the AccelDSP synthesis tool Tcl Console window or a Tcl script, this command is mainly used by the AccelDSP GUI to set design directives on in-memory design objects. When you set a directive using the AccelDSP GUI, the equivalent SetDirective command is executed, then recorded in the Tcl transcript. When you save the project, SetDirective commands are saved to the design directives file (.add). When the project is re-opened at a later time, the ADD file is read and the directives are re-applied to the in-memory design.

From the AccelDSP GUI, you can quickly show, modify, or delete all AccelDSP Directives by clicking the **All Directives** icon as show below:

Related Commands

GetDirective
DeleteDirective

SetGlobalOption

Set one or more global options.

Example

```
SetGlobalOption -use_tabs {true} -tab_size {4}
SetGlobalOption -ignorefunction {acos}
```

Syntax

```
SetGlobalOption
[-default_overflow_mode {wrap|saturate}]
[-device <device_name>]
[-fi_objects 0|1]
[-fixedpointlanguage {MATLAB | C++}]
[-flow ISE|{System Generator}|HW Co-Sim]
[-frequency {<integer> | auto} ]
[-hwcosim_target <name_of_target_device>]
[-ignorefunction <function_name>]
[-impltool {No Implementation}|ISE]
[-logicore_defaults {{latency 0|max-frequency} {use_dsp48 0|1}}]
[-maxconsolelines <number_of_lines>]
[-memmap_defaults {register_output{0|1}} {array_access_guard {0|1}}
     {ram_threshold <numeric_Value>}]
[-msglvl {<message_ID> {1|2|3}]
[-register_inputs {0|N}] (where N can be any integer value)
[-register_outputs {1|N}] (where N can be any integer value)
[-reset]
[-register_inputs {0|1}]
[-register_outputs {1|2}]
[-retiming {0|1}]
[-signal_name_clock <clock_name>]
[-signal_name_input_available <string_name>]
[-signal_name_input_request <string_name>]
[-signal_name_output_acknowledge <string_name>]
[-signal_name_output_available <string_name>]
[-signal_name_reset <reset_name>]
[-simtool {No Simulation}|{ISE Simulator}|Modelsim]
[-sync_reset 0|1]]
[-synthtool {No Synthesis}|{Precision RTL Synthesis}|{Synplify}
     |Synplify Pro|XST]
[-tab_size <integer>]
[-targetlanguage {No Language}|VHDL|Verilog|{All Languages}]
[-technology
     spartan3|spartan3a|spartan3adsp|spartan3e|virtex4|virtex5]
[-techvendor |Xilinx|Other]
[-unsupportedfunction <functionname>]
[-use_logicores {{multiplier 0|1} {multaccum 0|1} {multadd 0|1}
     {accum 0|1} {adder 0|1}{subtractor 0|1}}]
[-use_tabs true|false]
[-writeimplconfigfile {1|0}]
```

Options

- -default_overflow_mode

Specifies the value for the quantizee overflow mode, wrap or saturate. The default is: wrap

- -device

  `Specifies the device name for the target technology.`

- -fi_objects

Allows the use of fi objects in the **Verify Fixed Point** MATLAB simulation. fi objects are fixed-point numeric objects in the MathWorks Fixed-Point Toolbox that allow for greater rounding accuracy. You normally set this options to 1 (true) if the design contains operations that result in values greater than 53 bits. Greater rounding accuracy is achieved at the expense of slower MATLAB run times.

- -fixedpointlanguage

Specifies whether the fixed-point simulation language should be MATLAB or C++. The default is MATLAB.

- -flow

Sets the design flow sequence inthe Flow Bar.

- -frequency

Sets the target design frequency. Specify a numeric value. If your downstream RTL synthesis tool is Synplify Pro and you plan to use the Auto Constraints feature, you may specify “auto” for the frequency.

- -hwcosim_target

Used to set the name of the target device for Hardware Co-Simulation. The current list of supported devices are as follows:

```
{ML402 (Network-based Ethernet)}
{ML402 (Point-to-point Ethernet)}
{Virtex4 ML402 (JTAG)}
{Virtex4 Video Starter Kit (JTAG)}
{ML506 (Network-based Ethernet)}
{ML506 (Point-to-point Ethernet)}
{Spartan-3A DSP 1800A Starter Platform (Point-to-point Ethernet)}
{Spartan-3A DSP 3400A Development Platform (Network-based Ethernet)}
{Spartan-3A DSP 3400A Development Platform (Point-to-point Ethernet)}
{XtremeDSP Development Kit (JTAG)xc2v2000}
{XtremeDSP Development Kit (JTAG)xc2v3000}
{XtremeDSP Development Kit (JTAG)xc2v6000}
{XtremeDSP Development Kit (JTAG)xc2vp30}
{XtremeDSP Development Kit (JTAG)xc4vsx35}
{XtremeDSP Development Kit (PCI)xc2v2000}
{XtremeDSP Development Kit (PCI)xc2v3000}
{XtremeDSP Development Kit (PCI)xc2v6000}
{XtremeDSP Development Kit (PCI)xc2vp30}
```

- `{XtremeDSP Development Kit (PCI)xc4vsx35}`-ignorefunction

Specifies the name of a function to be ignored by AccelDSP for synthesis.

- -ignorefunction

Causes AcceDSP to ignore the specified function for synthesis.

- -impltool

Sets the down-stream implementation tool. The default is: {No Implementation}.

- -maxconsolelines

Sets the maximum number of lines that are displayed in the Tcl Console. The lines displayed can't be less than 50 or greater than 3000. The default is 1024 lines.

- -memmap_defaults
  - register_output

    When set to 1 (true), a register is inserted at the output of every memory read operation. In many cases, this will shorten the critical path and increase the maximum frequency. However, if a memory read operation is in a feedback loop, a warning message is issued indicating that bad logic may result. In this case, it is best to turn off this project option or override the setting on this particular variable using a memmap directive with the output register parameter turned off.

  - array_access_guard

    When set to 1 (the default), the guard is in place. This provides the maximum protection for read-write operations. When the parameter is set to 0 (false), AccelDSP pipelines the states between the array Read-Write operations more aggressively to minimize processing cycles. This may result in simulation mismatches during the Verify RTL step.

  - ram_threshold

    Specifies the number of elements in an array, before the array is automatically mapped to RAM (random access memory). The default is 64.
- -msglvl

Set the message masking level for the specified Information or Warning message. For example, you can set the following line in the AccelDSPInit.tcl file:

```
SetGlobalOption -msglvl {I-GENERAL-0001 2}
```

This causes the information message I-GENERAL-0001 Time elapsed: to be removed from the GUI Tcl Console transcript and the accel.log file transcript. At a later time, you can change the **-message_level_info** project option to reveal the message again. For example:

```
SetProjectOption -message_level_info {1 2}
```

This command cause message I-GENERAL-0001 to be masked in the Tcl Console transcript (level 1), but recorded in the accel.log transcript (level 2).

If you then change the **-message_level_info** project option again, for example:

```
SetProjectOption -message_level_info {2 2}
```

the message I-GENERAL-0001 will be recorded in the Tcl Console transcript as well as the accel.log transcript.

Changing the **-message_level_info** project option back to the default, for example:

```
SetProjectOption -message_level_info {1 1}
```

causes the message I-GENERAL-0001 to be masked (removed) from both transcripts, because the message level for this particular message_ID is set to 2.

- -register_inputs

Sets the number of registers on the inputs of the design. The minimum is 0. The maximum can be any integer value.

- -register_outputs

Sets the number of registers on the outputs of the design. The minimum is 1. The maximum can be any integer value.

- -reset

Used with certain options to return the option value to its default.

- -signal_name_clock

Sets the name of the clock input port. The default is 'Clock'.

- -signal_name_input_available

Sets the name of the “input available” handshake signal. The default is: {ac_InputAvail}

- -signal_name_input_request

Sets the name of the “input request” handshake signal. The default is: {ac_InputReq}

- -signal_name_output_acknowledge

Sets the name of the “output acknowledge” handshake signal. The default is: {ac_OutputAck}

- -signal_name_output_available

Sets the name of the “output available” handshake signal. The default is: {ac_OutputAvail}

- -signal_name_reset

Sets the name of the reset input port. The default is 'Reset'. The global Reset **must be held active high for at least one clock cycle** and returns all registers to a known state.

- -simtool

Sets the down-stream simulation tool. The default is: {No Simulation}

- -sync_reset

Specifies that the registers used in the RTL design will have a synchronous reset.

- -synthtool

Sets the down-stream RTL synthesis tool. The default is {No Synthesis}.

- -tab_size

Set the tab spacing in the text editor to the specified integer value.

- -targetlanguage

Sets the output language format. The default is: {No Language}

- -technology

Specifies the target technology. Must be compatible with the specified technology.

- -techvendor

Specifies the target technology vendor.

- -unsupportedfunction

Causes AcceDSP to error if the specified function is found in the design function.

- -use_logicores

This global option turns on the use of Xilinx optimized LogiCORE IP in the generated HDL for the specified supported operator types. The value 0 (false) tells AccelDSP not to use a LogiCORE for that specified operator type.

Note: If you set this project option to 1 (true) for a specified operator type and the overall performance of your design does not increase, then you should turn this project option back off (0) for

that specified operator type. See the topic Improving Performance with a Use_logicore Directive for details.

- -use_tabs

Specifies whether or not to use tabs.

- -writeimplconfigfile

Sets whether or not to write a bit-stream configuration file during the Implement step. The default is 0 (false) because of the large size of the file.

Description

The SetGlobalOption command is used in an AccelDSP Initialization file (AccelDSPInit.tcl) for setting global options when the AccelDSP synthesis tool is invoked. The AccelDSPInit.tcl file located at pathname ...\configure\AccelDSPInit.tcl in the AccelDSP software tree is loaded first. If you have an AccelDSPInit.tcl file in your $HOME directory, it is loaded next, and if you have an AccelDSPInit.tcl file in you project directory, it is loaded last. Global options set from the init file in your $HOME directory may overwrite global options set from the init file in the AccelDSP software tree. And, global options set from the AccelDSPInit.tcl file in your project directory may overwrite global options set from the init file in your $HOME directory.

Global options are typically set by loading AccelDSPInit.tcl at startup. If you set a global option manually from the Tcl Console, that setting will not be saved when you exit the tool.

A global option may be overridden by setting the equivalent project option on a project-by-project basis.

Related Commands

GetGlobalOption
DeleteGlobalOption
Synthesize

SetProjectOption

Set one or more project options.

Example

```
SetProjectOption -targetlanguage VHDL
```

Syntax

```
SetProjectOption
[-constraint_file <sdcfile_pathname>]
[-clock_enable 0|1]
[-default_overflow_mode {saturate | wrap}]
[-default_round_mode {floor | nearest}]
[-device <device_name>]
[-directivesfile <directivesfile_pathname>]
[-dontcare_value {0|1|DontCare}]
[-fi_objects 0|1]
[-fixedpointlanguage {MATLAB|C++}]
[-flow ISE|{System Generator}|HW Co-Sim]
[-frequency {<integer> | auto} ]
[-hwcosim_target <name_of_target_device>]
[-impltool {No Implementation}|ISE]
[-interface_protocol push|full]
[-logicore_defaults {{latency 0|max-frequency} {use_dsp48 0|1}}]
[-memmap_defaults {register_output{0|1}} {array_access_guard {0|1}}
     {ram_threshold <numeric_Value>}]
[-message_level_info <Tcl Console level> <accel.log level>]
[-message_level_warn <Tcl Console level> <accel.log level>]
[-package <package_name>]
[-pnr_effort {low|medium|high}]
[-quantizer_max_constant_fractional_length] <max_fractional_length>]
[-quantizer_max_input_fractional_length] <max_fractional_length>]
[-register_inputs {0|N}] (where N can be any integer value)
[-register_outputs {1|N}] (where N can be any integer value)
[-replaceconstantmults {0|1}]
[-retiming 0|1]
[-scriptfile <scriptfile_pathname>]
[-show_overflows 0|1]
[-show_underflows 0|1]
[-signal_name_clock <clock_pathname>]
[-signal_name_clock_enable <clock_enable_name>]
[-signal_name_input_available <string_name>]
[-signal_name_input_request <string_name>]
[-signal_name_output_acknowledge <string_name>]
[-signal_name_output_available <string_name>]
[-signal_name_reset <reset_name>]
[-simtool {No Simulation}|{ISE Simulator}|Modelsim]
[-speed <speed_grade>]
[-sync_reset 0|1]]
[-synth_auto_constrain_io 0|1]
[-synth_enable_io_insertion 0|1]
[-synth_enable_pipelining 0|1]
[-synth_fanout_limit <number of fanouts>]
[-synth_resource_sharing 0|1]
```

```
[-synthtool {No Synthesis}|{Precision RTL Synthesis}|{Synplify}
     |Synplify Pro|XST]
[-system_generator_library_name] <system_generator_library_name>
[-targetlanguage {No Language}|VHDL|Verilog|{All Languages}]
[-tb_max_errors <tb_max_errors>]
[-tb_output_latency <tb_output_latency>]
[-technology
     spartan3|spartan3a|spartan3adsp|spartan3e|virtex4|virtex5]
[-techvendor Xilinx|Other]
[-unroll_array_adds 0|1]
[-unroll_array_multiplies 0|1]
[-unroll_array_subtracts 0|1]
[-unroll_for_loops 0|1]
[-unroll_matrix_multiplies 0|1]
[-use_logicores {{multiplier 0|1} {multaccum 0|1} {multadd 0|1}
     {accum 0|1} {adder 0}{subtractor 0}}]
[-writeimplconfigfile 0|1]
```

Options

- -clock_enable

Specifies whether or not to generate a clock enable signal in RTL interface. The default is 0 (false).

- -constraintfile

Specifies the pathname to the RTL synthesis tool constraint file. You can edit this file to add user-specified constraints. For example, if you are using Synplify Pro, adding the command define_global_attribute syn_ramstyle block_ram causes Synplify Pro to map generic RAMs in the RTL to embedded Block RAM on the FPGA.

- -default_overflow_mode

Specifies the value for the quantizer overflow mode, wrap or saturate. The default is: wrap

- -device

Specifies the device name for your target technology.

- -directivesfile

Specifies the pathname to the design directives file (.add file).

- -dontcare_value

When you are creating a technology-specific implementation of a mathematical algorithm, internal signals and variables are created whose values need not be known at specific times. A good example of this is the address line of a RAM. When the design is not accessing the RAM, the address lines can be any arbitrary value. During these times, you can implement the circuit such that these lines (or wires) are either connected to Ground (logic 0), Power (logic 1), or either one (DontCare).

Knowing which value to use depends on your knowledge of your RTL synthesis tool. Many RTL synthesis tools cannot accept a “don’t’ care” value in VHDL or Verilog. Therefore, the DontCare value for this directive cannot be used. If, however, your RTL synthesis tool does support don’t cares, then you can set this directive to DontCare. This gives the tool more flexibility in achieving an optimum circuit. If your RTL synthesis tool does not handle don’t cares and the size of the circuit seems too large, you can try setting this value to one (1) in an attempt to find a better implementation.

- -fi_objects

Allows the use of fi objects in the **Verify Fixed Point** MATLAB simulation. fi objects are fixed-point numeric objects in the MathWorks Fixed-Point Toolbox that allow for greater rounding accuracy. You normally set this options to 1 (true) if the design contains operations that result in values greater than 53 bits. Greater rounding accuracy is achieved at the expense of slower MATLAB run times.

- -fixedpointlanguage

Specifies whether the fixed-point simulation language should be MATLAB or C++. The default is MATLAB.

- -flow

Sets the design flow sequence inthe Flow Bar.

- -frequency

Sets the target design frequency. Specify a numeric value. If your downstream RTL synthesis tool is Synplify Pro and you plan to use the Auto Constraints feature, you may specify “auto” for the frequency.

- -hwcosim_target

Used to set the name of the target device for Hardware Co-Simulation. The current list of supported devices are as follows:

```
{ML402 (Network-based Ethernet)}
{ML402 (Point-to-point Ethernet)}
{Virtex4 ML402 (JTAG)}
{Virtex4 Video Starter Kit (JTAG)}
{ML506 (Network-based Ethernet)}
{ML506 (Point-to-point Ethernet)}
{Spartan-3A DSP 1800A Starter Platform (Point-to-point Ethernet)}
{Spartan-3A DSP 3400A Development Platform (Network-based Ethernet)}
{Spartan-3A DSP 3400A Development Platform (Point-to-point Ethernet)}
{XtremeDSP Development Kit (JTAG)xc2v2000}
{XtremeDSP Development Kit (JTAG)xc2v3000}
{XtremeDSP Development Kit (JTAG)xc2v6000}
{XtremeDSP Development Kit (JTAG)xc2vp30}
{XtremeDSP Development Kit (JTAG)xc4vsx35}
{XtremeDSP Development Kit (PCI)xc2v2000}
{XtremeDSP Development Kit (PCI)xc2v3000}
```

- {XtremeDSP Development Kit (PCI)xc2v6000}

```
{XtremeDSP Development Kit (PCI)xc2vp30}
{XtremeDSP Development Kit (PCI)xc4vsx35}
```

- -impltool

Sets the down-stream implementation tool. The default is: {ISE}.

- -interface_protocol

Sets the interface protocol to either {full} or {push}. The default is: {push}. Refer to the chapter Interfacing to System Hardware for a detailed explanation of each protocal.

- -logicore_defaults

Sets the latency of all LogiCOREs to either zero(0) or maximum frequency. When max-frequency is selected, the tool adds the number of pipeline registers required to achieve the maximum frequency. This increases the latency. Also set whether or not to use DSP48s in the LogiCORE implementation.

- -memmap_defaults
  - register_output

    When set to 1 (true), a register is inserted at the output of every memory read operation. In many cases, this will shorten the critical path and increase the maximum frequency. However, if a memory read operation is in a feedback loop, a warning message is issued indicating that bad logic may result. In this case, it is best to turn off this project option or override the setting on this particular variable using a memmap directive with the output register parameter turned off.
  - array_access_guard

    When set to 1 (the default), the guard is in place. This provides the maximum protection for read-write operations. When the parameter is set to 0 (false), AccelDSP pipelines the states between the array Read-Write operations more aggressively to minimize processing cycles. This may result in simulation mismatches during the Verify RTL step.
  - ram_threshold

    Specifies the number of elements in an array, before the array is automatically mapped to RAM (random access memory). The default is 64.
- -message_level_info

Sets the system message level for Information type messages. For example: assume that you enter the following line in the AccelDSPInit.tcl file:

```
SetGlobalOption -msglvl {I-GENERAL-0001 2}
```

This causes the information message I-GENERAL-0001 Time elapsed: to be removed from the GUI Tcl Console transcript and the accel.log file transcript because the default -message_level_info is {1 1}. At a later time, you can change the -message_level_info project option to reveal the message again. For example:

```
SetProjectOption -message_level_info {1 2}
```

This command cause message I-GENERAL-0001 to be masked in the Tcl Console transcript (level 1), but recorded in the accel.log transcript (level 2).

If you then change the **-message_level_info** project option again, for example:

```
SetProjectOption -message_level_info {2 2}
```

the message I-GENERAL-0001 will be recorded in the Tcl Console transcript as well as the accel.log transcript.

Changing the **-message_level_info** project option back to the default, for example:

```
SetProjectOption -message_level_info {1 1}
```

causes the message I-GENERAL-0001 to be masked (removed) from both transcripts, because the message level for this particular message_ID is set to 2.

- -message_level_warn

Sets the system message level for Warning type messages. For example: assume that you enter the following line in the AccelDSPInit.tcl file:

```
SetGlobalOption -msglvl {W-QTZ-0404 3}
```

This causes the Warning message W-QTZ-0404 to be removed from the GUI Tcl Console transcript and the accel.log file transcript because the default -message_level_info is {1 1}. At a later time, you can change the **-message_level_info** project option to reveal the message again. For example:

```
SetProjectOption -message_level_info {1 3}
```

This command cause message W-QTZ-0404 to be masked in the Tcl Console transcript (level 1), but recorded in the accel.log transcript (level 3).

If you then change the **-message_level_info** project option again, for example:

```
SetProjectOption -message_level_info {3 2}
```

the message W-QTZ-0404 will be recorded in the Tcl Console transcript, but not in the accel.log transcript.

Changing the **-message_level_info** project option back to the default, for example:

```
SetProjectOption -message_level_info {1 1}
```

causes the message W-QTZ-0404 to be masked (removed) from both transcripts, because the message level for this particular message_ID is set to 3.

- -package

Specifies the package name for your target technology.

- -pnr_effort

Sets the effort level for the place and route tool to 'low', 'medium', or 'high'. The default is 'high'. Currently, Xilinx ISE is the only implementation tool set that supports this option. This option will be ignored for all other tools.

- -quantizer_max_constant_fractional_length

Specifies the maximum fractional length for all constants in the design. You can override this value on any particular constant by applying a quantize directive. The default is: 12

- -quantizer_max_input_fractional_length

Specifies the maximum fractional length for all input ports in the design. You can override this value on any particular input port by applying a quantize directive. The default is: 12

- -register_inputs

Sets the number of registers on the inputs of the design. The minimum is 0. The maximum can be any integer value.

- -register_outputs

Sets the number of registers on the outputs of the design. The minimum is 1. The maximum can be any integer value.

- -replaceconstantmults

Specifies that multiply operations with at least one constant operand are to be replaced by shift and add operations. This reduces the area of the design. In designs where increased performance is the goal, not replacing the constant multipliers improves the mapping to DSP48 cells and thus increases performance. The default value is Boolean 0 which means to not replace the constant multipliers.

- -retiming

Sets whether or not to turn on the retiming option of the down-stream RTL synthesis tool.

- -scriptfile

Specifies the pathname to the MATLAB script file.

- -show_overflows

Sets whether or not to record MATLAB overflow messages in the Tcl Console transcript. The default is 1 (true).

- -show_underflows

Sets whether or not to record MATLAB underflow messages in the Tcl Console transcript. The default is 1 (true). Typically turned off to reduce clutter in the transcript when looking for overflow messages.

- -signal_name_clock

Sets the name of the clock input port. The default is 'Clock'.

- -signal_name_clock_enable

Sets the name of the clock enable input port. The default is 'ClockEnable'.

- -signal_name_input_available

Sets the name of the “input available” handshake signal. The default is: {ac_InputAvail}

- -signal_name_input_request

Sets the name of the “input request” handshake signal. The default is: {ac_InputReq}

- -signal_name_output_acknowledge

Sets the name of the “output acknowledge” handshake signal. The default is: {ac_OutputAck}

- -signal_name_output_available

Sets the name of the “output available” handshake signal. The default is: {ac_OutputAvail}

- -signal_name_reset

Sets the name of the reset input port. The default is 'Reset'.

- -simtool

Sets the down-stream simulation tool. The default is: {No Simulation}

- -speed

Specifies the speed grade for your target technology.

- -sync_reset

Specifies that the registers used in the RTL design will have a synchronous reset.

- -synth_auto_constrain_io

Sets the auto_constrain_io option in Synplify Pro. The default for this option is 0 (false). When this option is disabled (the default), all I/O port constraints are forward annotated. When this option is enabled, only explicit I/O port constraints are forward annotated.

- -synth_enable_io_insertion

When set to the default 'True' (1), this option tells the down-stream RTL synthesis tool to enable the insertion of IO cells on the design IO ports. This is normally done when the design is not part of a larger design on the FPGA.

- -synth_enable_pipelining

When set to the default 'True' (1), this option tells the Synplify Pro RTL synthesis tool to enable the insertion of pipeline registers. This results in maximum performance.

- -synth_fanout_limit

Sets the fanout_limit option in Synplify Pro. The default for this option is 500. Setting this option to a lower value which may increase design performance at the expense of increased design area due to an increase in replicated paths.

- -synth_resource_sharing

Sets the resource_sharing option in Synplify Pro. The default for this option is 1 (true). Setting this option to 0 (false) turns off resource sharing which may increase design performance at the expense of increased design area.

- -synthtool

Sets the down-stream RTL synthesis tool. The default is {No Synthesis}.

- -systen_generator_library_name

Sets the target System Generator library name into which the newly created System Generator block will be placed.

- -targetlanguage

Sets the output language format. The default is: {No Language}

- -tb_max_errors

Sets the test bench maximum error count. The default is zero. This option tells the Testbench how many output mismatches can occur before the simulation is stopped. The default is 0 (zero), so any number of mismatches can occur and the simulation is never stopped. If you have very large input files and many mismatches start occurring, you'll want the simulation to stop rather that proceed to simulate a design that has obviously gone awry. If you set max_errors to 10, for example, the Testbench will stop after 10 mismatches with the output reference file.

- -tb_output_latency

Sets the Testbench output Startup Clock Cycles. The default is zero. Refers to the number of clock cycles that it takes for a hardware datapath (the algorithm) to compute a result. In a MATLAB simulation, there is no concept of time; the results appear at the function output(s) the instant the input(s) are applied. When the design function is synthesized to hardware, the output(s) can appear any number of cycles after the input(s) are applied. This output_latency parameter tells the Testbench how many clock cycles to wait before comparing the output data to the reference file. In most cases, adjusting this parameter is not necessary, but if excessive mismatches occur, an adjustment here may fix the problem.

- -technology

Specifies the target technology. Must be compatible with the specified technology vendor.

- -techvendor

Specifies the target technology vendor.

- -unroll_array_adds

Sets whether or not to unroll all array add operations in the design. The default is 1 (true).

- -unroll_array_multiplies

Sets whether or not to unroll all array multiply operations in the design. The default is 1 (true).

- -unroll_array_subtracts

Sets whether or not to unroll all array subtract operations in the design. The default is 1 (true).

- -unroll_for_loops

Sets whether or not to unroll all explicit and implicit for loop operations in the design. The default is 0 (false). You may unroll individual for loops by using the unroll directive as explained in the topic How to Apply the Unroll Directive to a for Loop.

- -unroll_matrix_multiplies

Sets whether or not to unroll all matrix multiply operations in the design. The default is 0 (false). You may unroll individual matrix multiply operations by using the unroll directive as explained in the topic How to Apply the Unroll Directive to a Matrix Multiply.

- -use_logicores

This project option turns on the use of Xilinx optimized LogiCORE IP in the generated HDL for the specified supported operator types. The value 0 (false) tells AccelDSP not to use a LogiCORE for that specified operator type.

If you set this project option to 1 (true) for a specified operator type and the overall performance of your design does not increase, then you should turn this project option back off (0) for that specified operator type. See the topic Improving Performance with a Use_logicore Directive for details.

Note: If you set this project option to 1 (true) and the overall performance of your design does not increase, then you should turn this project option back off (0).

- -writeimplconfigfile

Sets whether or not to write a bit-stream configuration file during the Implement step. The default is 0 (false) because of the large size of the file.

Description

The SetProjectOption command is the main mechanism for setting project options. This command is also used in the project file (.acc) to initialize a project on startup.

From the AccelDSP GUI, you can quickly display the setting of all AccelDSP Project Options by clicking the Project Options icon as show below:

Related Commands

GetProjectOption
Project

Synthesize

Call the pre-specified RTL Synthesis tool to read the RTL HDL design and generate a gate-level netlist.

Example

```
Synthesize
```

Syntax

```
Synthesize
```

Description

The Synthesize command is part of the main synthesis flow. It is a Tcl script that calls the target synthesis tool. The synthesis tool reads the RTL HDL design and generates a gate-level netlist. This netlist is typically used as the input to the ISE implementation tools.

Related Commands

Project
Analyze
Generate
Verify
Implement
SetProjectOption
SetGlobalOption

Verify

Verify the RTL design using the generated Testbench and the specified HDL simulation tool.

Example

```
Verify
```

Syntax

```
Verify -floatingpoint |-fixedpoint | -rtl | -gatelevel
```

Options

- -fixedpoint

Invoke MATLAB and run the generated fixed-point model.

- -floatingpoint

Invoke MATLAB and simulate the floating-point model.

- -gatelevel

Invoke the pre-specified HDL simulation tool on the Testbench and the HDL simulation model of the implemented design. A PASSED report means that the implemented design is bit-true with the fixed-point MATLAB model.

- -rtl

Invoke the pre-specified HDL simulation tool on the Testbench/RTL Design.

Description

The Verify command runs the appropriate verification tool on the specified model.

Related Commands

Project
Analyze
Generate
Synthesize
Implement
SetProjectOption
SetGlobalOption

XILINX®

Chapter 9

AccelDSP Getting Started Tutorial

Introduction

This tutorial exercise will guide you through the process of transforming a MATLAB floating-point model into a hardware module that can be implemented in a Xilinx FPGA. The design is a general purpose FIR filter.

Installation

The AccelDSP synthesis tool is designed to work in an integrated flow with MATLAB and down-stream RTL synthesis, simulation, and implementation tools. This tutorial takes you step-by-step through this flow.

Before you begin, verify that you have the AccelDSP synthesis tool installed as specified in Chapter 2 of this manual and that your AccelDSP license is in place. MATLAB is required to complete steps 3 and 6. An HDL simulator (ISE Simulator or ModelSim SE) is required for step 9 and an RTL synthesis tool like XST or Synplify Pro is required for step 10.

If you don't have one or more of the supporting tools installed, you won't be able to run the complete tutorial, however, you can progress through the AccelDSP-specific steps to verify the RTL model.

Tutorial Exercise

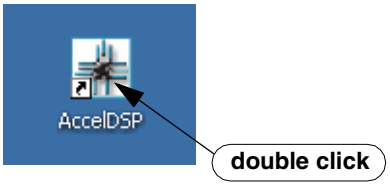
1. Examine the MATLAB Floating-Point Model

Your MATLAB design must conform to a minimum set of coding style guidelines. The basic guidelines for the FIR filter are discussed starting with the topic Examining the Floating-Point Model. For this tutorial, you can assume that the FIR filter conforms to the guidelines. For a more complete discussion about style requirements, refer the online help topic "Style Guide" or the PDF document titled "MATLAB for Synthesis Style Guide".

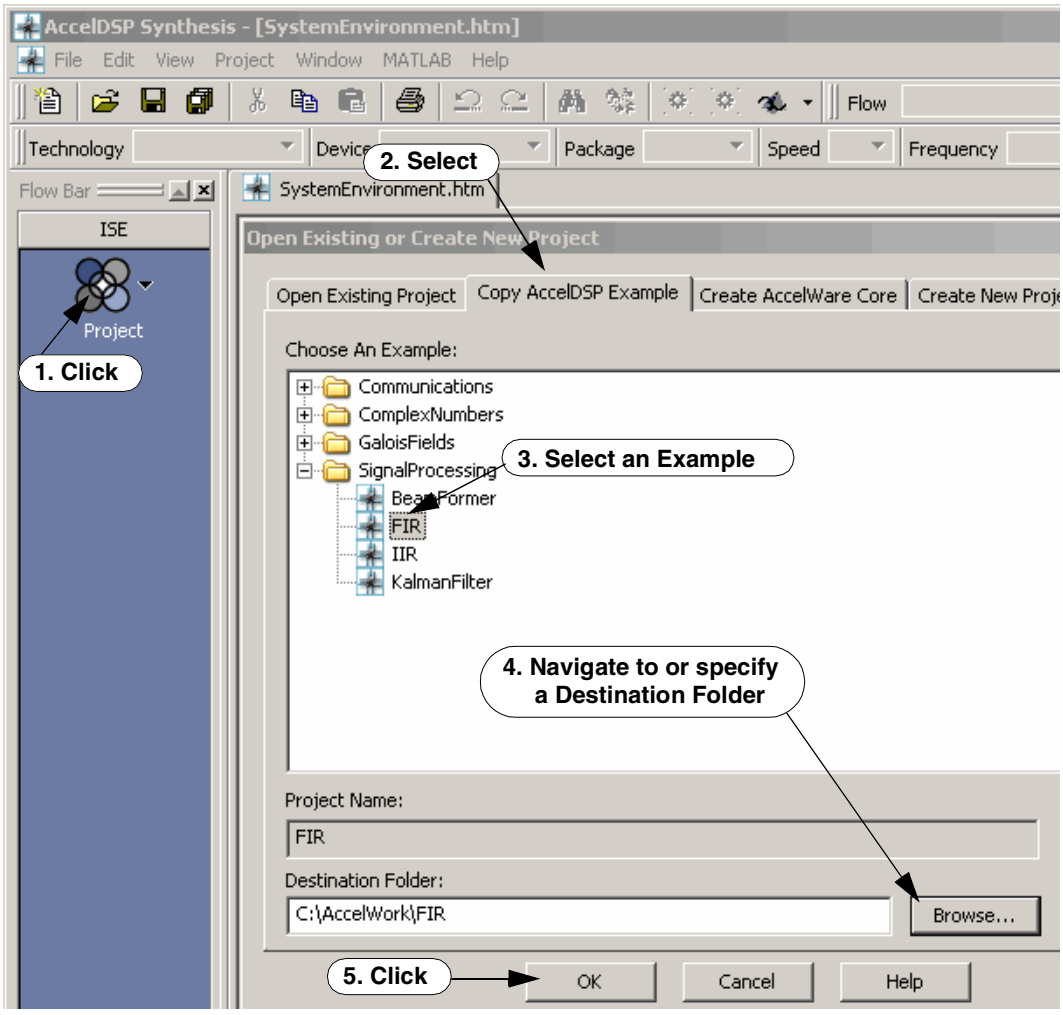
2. Create an AccelDSP Project

In this step, you will create/open an AccelDSP project and analyze the Floating Point Model.

a. Invoke the AccelDSP synthesis tool by double-clicking on the desktop icon.



b. Close the Tip of the Day window, then follow the directions below.



c. For this tutorial, you are creating a new project by copying an existing FIR design that is provide by AccelDSP Synthesis in the Examples directory. By default, the new project is placed in a working area at the pathname specified by the value of the ACCELWORK environment variable.

3. Set the Target Technology and Tool Options

Verify that the target technology and the AccelDSP tool options are set correctly by selecting the pulldown menu **Project > Project Options...**

AccelDSP Synthesis supports a wide range of Xilinx devices.

Set and/or verify the Project Options shown below:

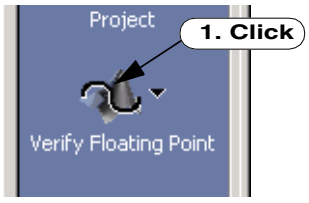
Frequency (MHz): 100
Technology: virtex5
Device: xc5vsx50t
Package: ff1136
Speed Grade: -1
RTL Simulation:<*select HDL simulator*>
RTL Synthesis:XST

To save these settings, select the pulldown menu **File > Save> Save Project**.

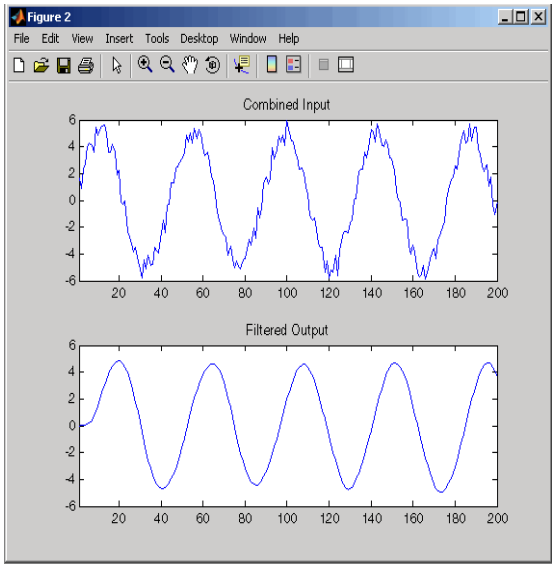
4. Verify the Floating-Point Model

The AccelDSP synthesis tool provides a direct link to MATLAB so you don't have to leave the tool to run a MATLAB simulation. If you have already verified the floating point model, you may skip this step by clicking on the black down-arrowhead next to the icon and selecting the Skip Verify -floatingpoint option.

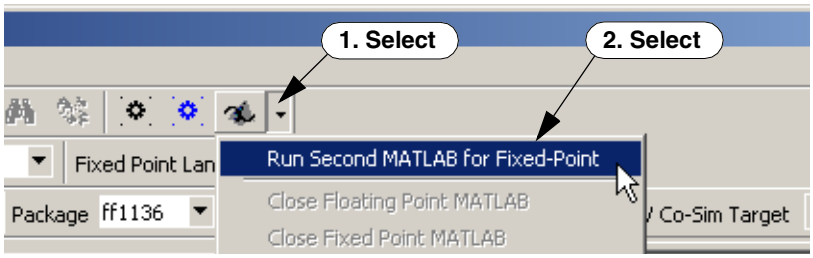
a. To verify that your floating point design passes a MATLAB simulation, click on the Verify Floating Point icon, as shown below:



b. Examine plot Figure 2 of the three plots. You will use this as the reference plot for comparing future results.



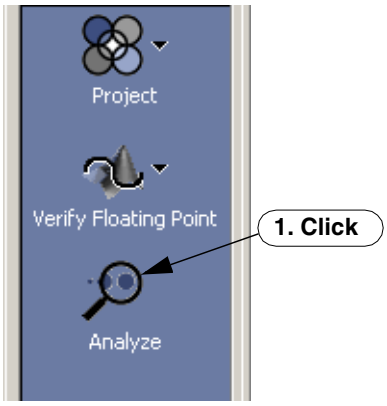
Since you will be comparing this "golden" plot to a generated fixed-point plot in Step 7, you should set the option called **Run Second MATLAB for Fixed-Point** as shown below:



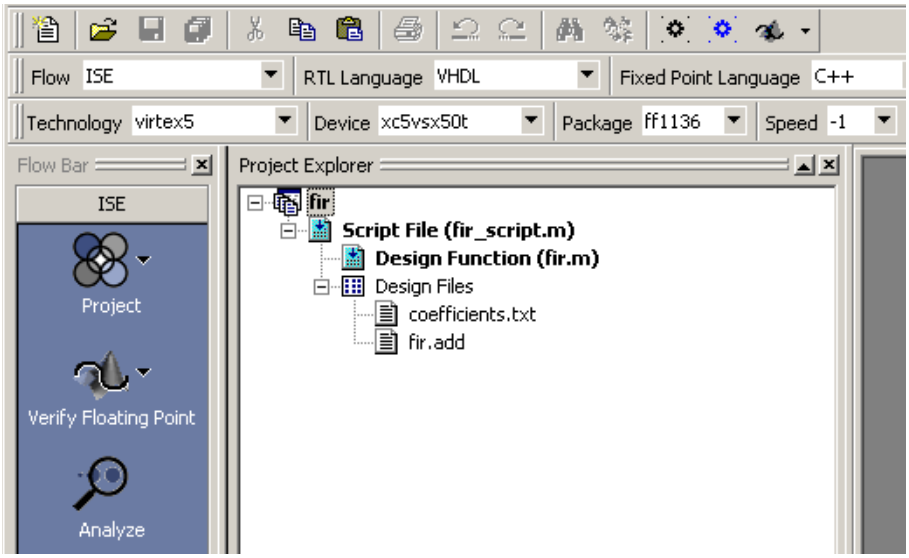
5. Analyze the Floating-Point Model

The **Analyze** icon appears in the AccelDSP Flow bar after the Verify Floating Point step is complete (or skipped).

a. To analyze the floating-point model, click on the Analyze icon as shown below:

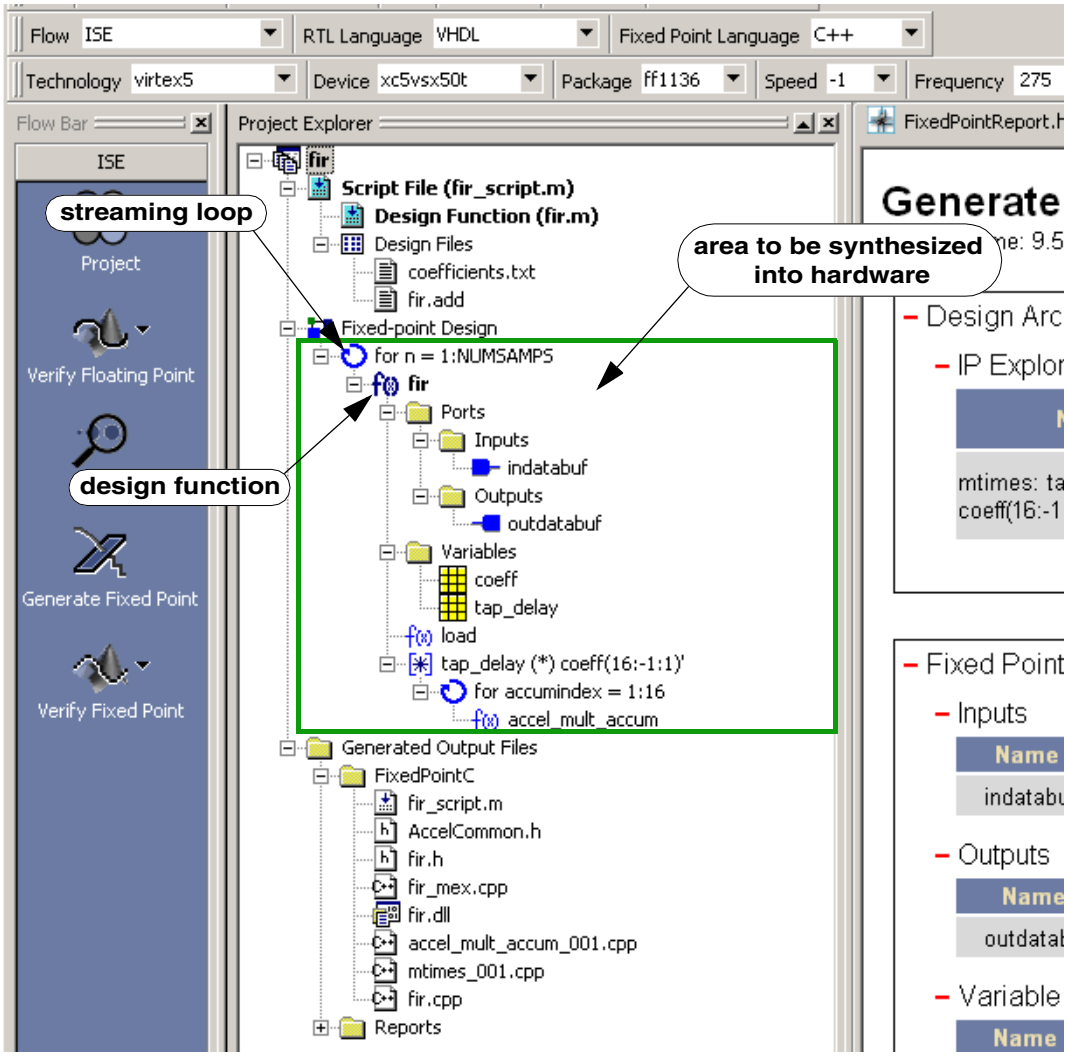


During the Analyze step, the MATLAB files are analyzed by AccelDSP and all the files associated with the design are listed in the Project Explorer window as shown below. The **streaming loop** and the top-level **design function** are identified.



6. Generate a Fixed-Point Model

a. Click on the **Generate Fixed Point** icon to generate an equivalent fixed-point design. The fixed-point design tree is displayed in the Project Explorer window as shown below.



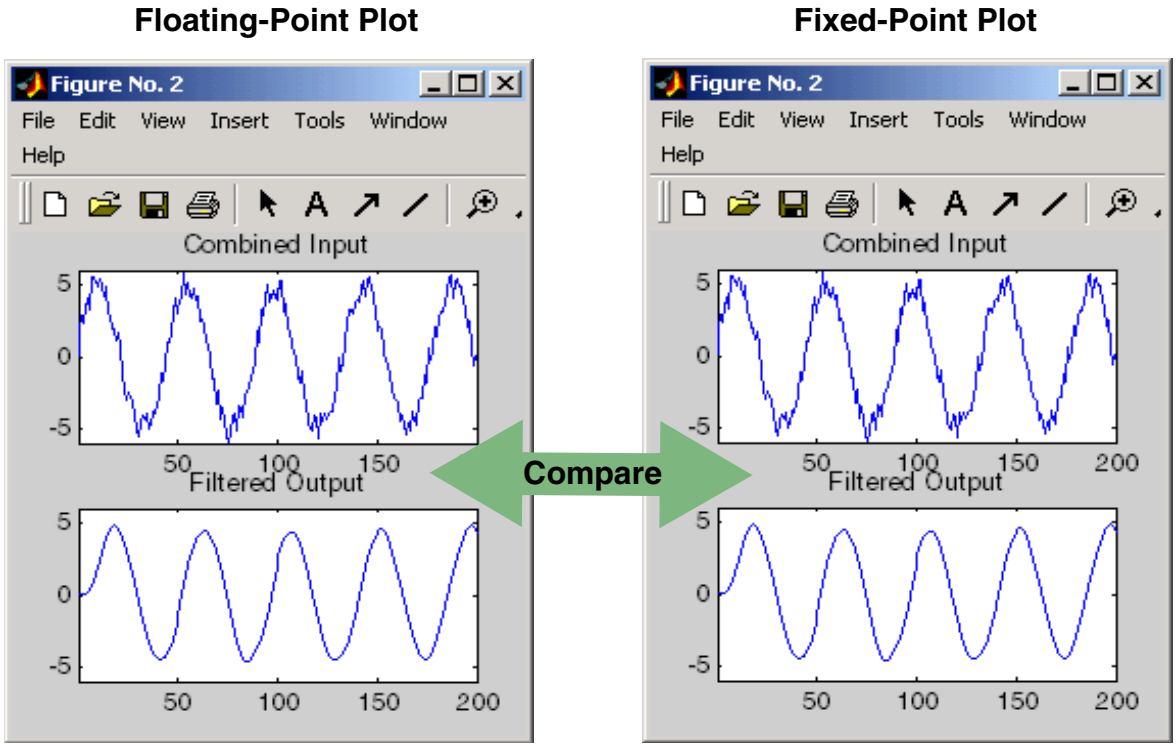
During the Generate Fixed Point process, default assumptions are made about design objects in your design like variables and loops. For future reference, you can right-click on these objects, select **Properties**, then examine the properties that are automatically assigned. If need be, you can change the properties before generating the RTL model. Also note that you can view and change the quantization parameters on variables from the Generate Fixed Point Report.

7. Verify the Fixed-Point Model

Click on the **Verify Fixed Point** icon to start the MATLAB verification process. MATLAB is automatically invoked on the fixed-point model.

NOTE: It may take a few moments to invoke MATLAB and run the simulation. The fixed-point plot is displayed when the simulation is finished.

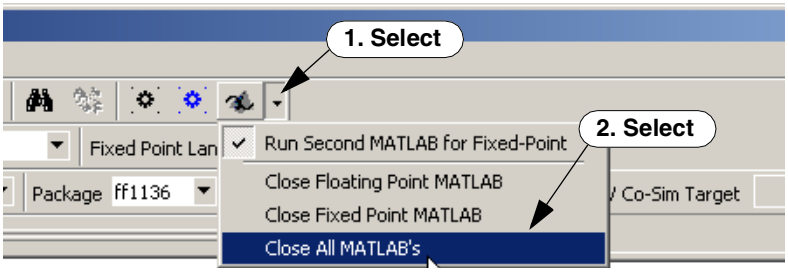
The fixed-point plot is shown below on the right.



The floating-point plot should still be displayed from step 4.

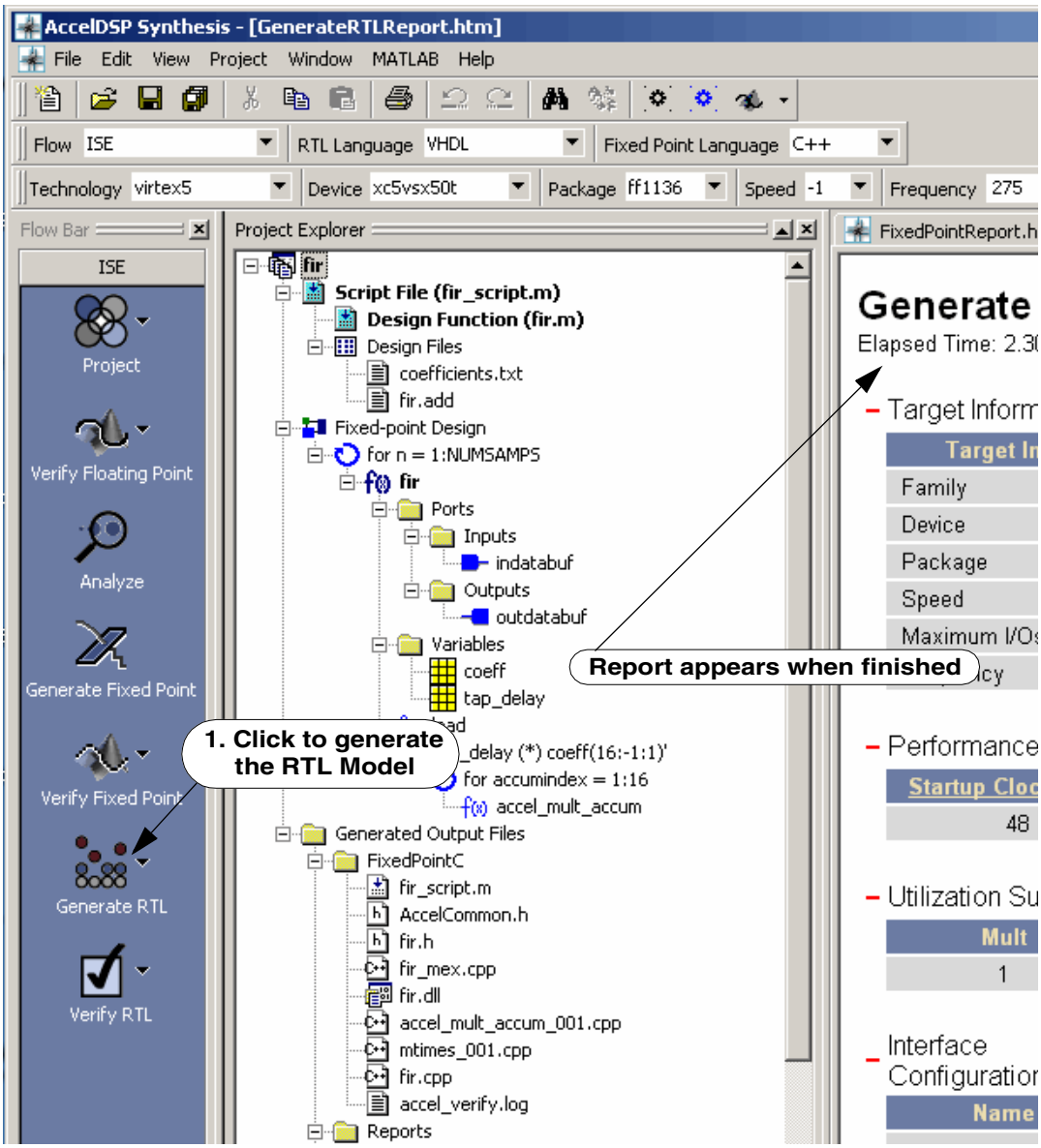
Visually compare the fixed-point plot with the floating-point plot. If you are satisfied with the results, you are ready to proceed to synthesis. If the fixed-point plot deviates too much from the floating-point plot, you may need to adjust the quantizer properties on some design variables, then repeat the verification process. When you are satisfied with the fixed-point verification results, you are ready to generate the RTL design.

As shown below, close the MATLAB windows that were opened by the AccelDSP synthesis tool.



8. Generate the RTL Design

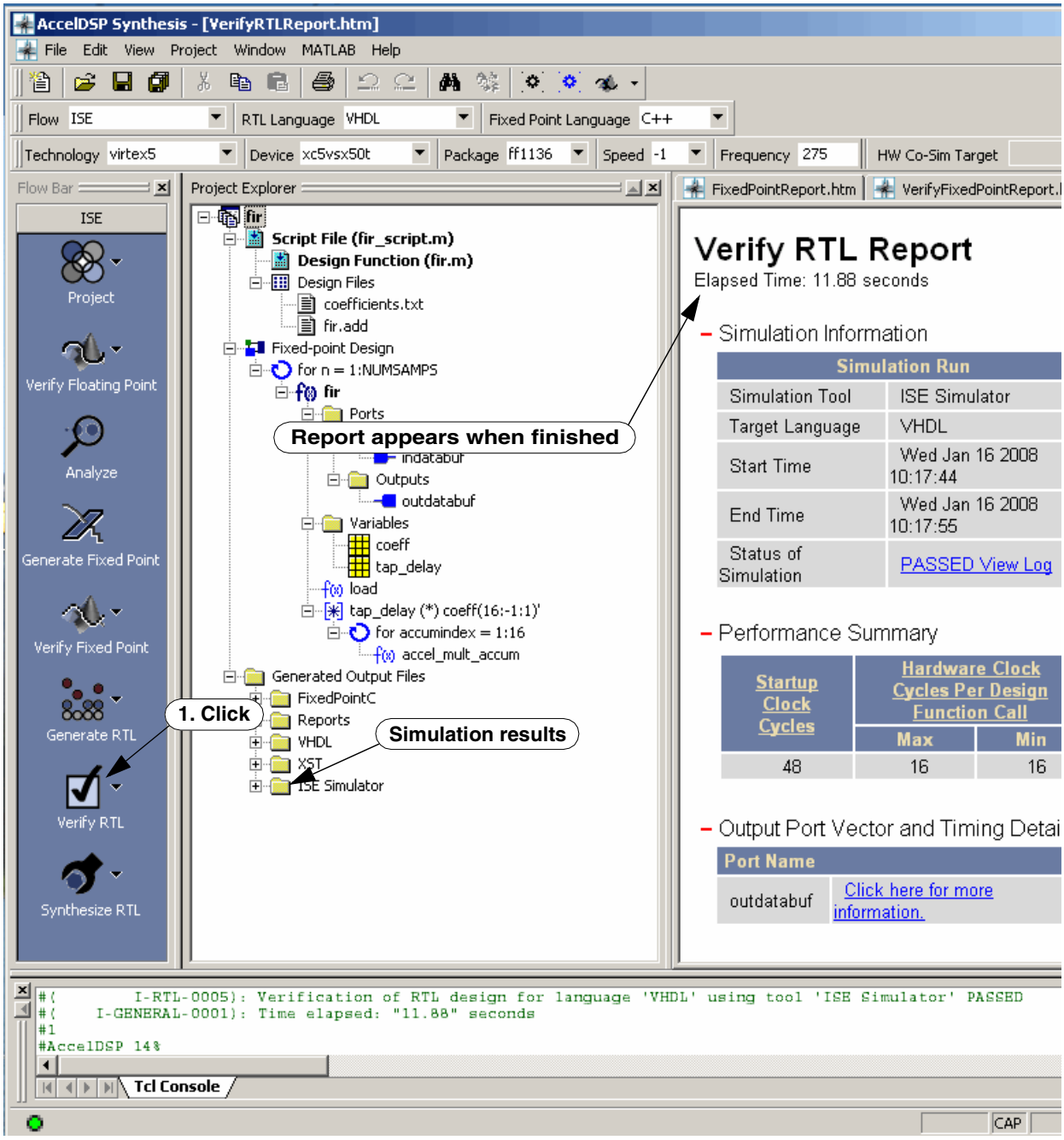
Click the **Generate RTL** icon in the AccelDSP Flow bar as shown below:



The results are summarized in the generated report. You can also expand the **VHDL** or **Verilog** folder to view the generated HDL files and the associated **Testbench** files.

9. Verify the RTL Model

Click on **Verify RTL** as shown below. If you have not yet selected an HDL simulator tool, you will be prompted to do so.



The AccelDSP synthesis tool causes the simulation tool to compile the Testbench and simulate the RTL model. When the simulation is finished, a 'PASSED' message is shown in the transcript window.

10. Synthesize the RTL Model

Once you verify the RTL Model, you are ready to transform the model into a gate-level netlist.

Assuming that you have an RTL synthesis tool installed, do the following:

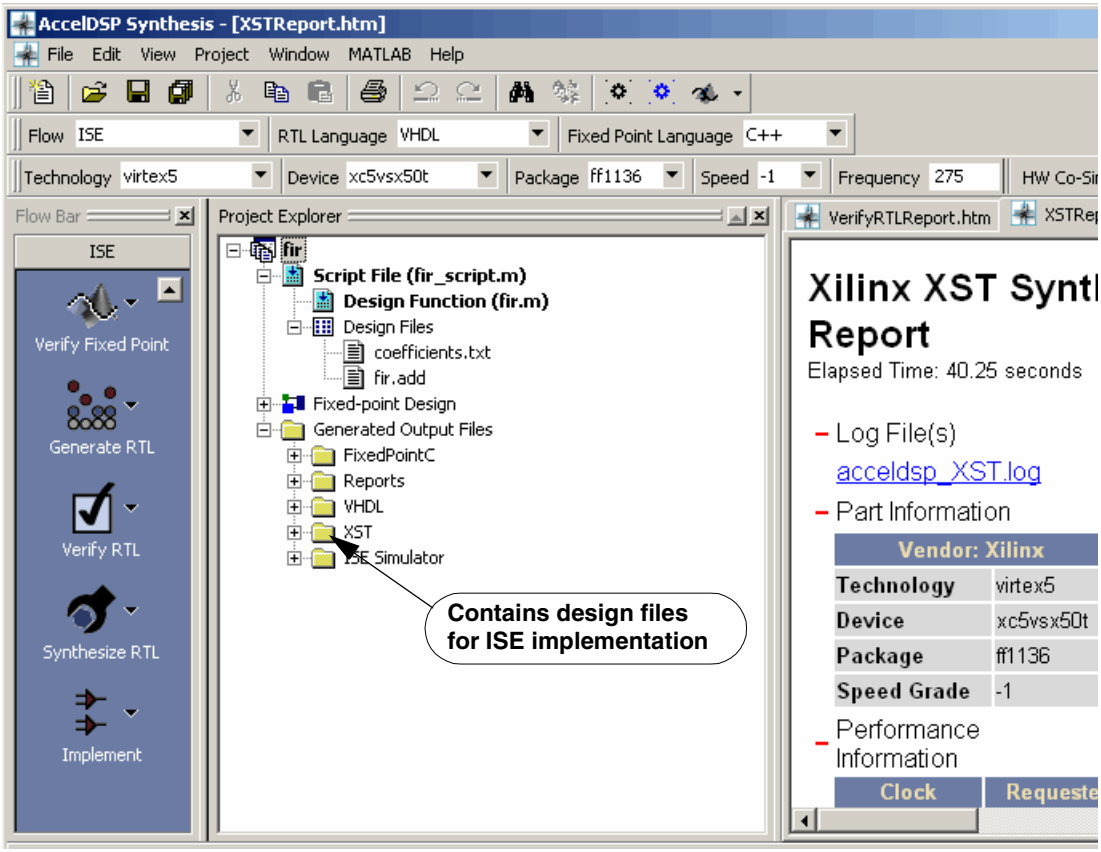
a. Select the **Synthesize RTL** icon in the AccelDSP Flow Bar

b. If the tool prompts you to select an RTL Synthesis Tool, select your tool from the Project Options dialog box, then click **Apply**.

The RTL synthesis step starts and when finished, a message similar to the following appears in the Tcl Console window:

```
#(I-SYNTH-0002): Synthesis of RTL design for language 'VHDL' using tool
'XST' PASSED
```

As shown below, examine the contents of the newly-created folder for the RTL synthesis tool.



You are now ready to finish the design implementation by clicking on **Implement** which invokes ISE on these files.

Summary

In this exercise, you did the following:

1. You invoked the AccelDSP synthesis tool and created a new project by copying an AccelDSP example design. You then used the Project Options dialog box to specify the Target Technology, the RTL Simulation tool and the RTL Synthesis tool. You also selected the HDL output language
2. Next, you ran a MATLAB simulation on the floating-point design, then saved the resulting "golden" plot for a later comparison with future plots
3. You directed the AccelDSP synthesis tool to Analyze the **fir** design. You then examined the tree structure of the analyzed design files in the Project Explorer window
4. Next, you clicked the **Generate Fixed Point** icon to generate a fixed-point model of the in-memory design. You then clicked the **Verify Fixed Point** icon in the AccelDSP Flow bar to run a MATLAB simulation on the fixed-point model
5. Once the fixed-point simulation finished, you opened the FIG file that you previously saved in step 3 and visually compared the floating-point plot to the fixed-point plot. You were satisfied that the fidelity of the fixed-point plot was acceptable
6. You generated an RTL model by clicking on the **Generate RTL** icon, then examined the resulting HDL files. You also examined the Generate RTL report
7. You verified the bit-true accuracy of the RTL model by running your HDL simulator on the AccelDSP-generated Testbench
8. Finally, you invoked your RTL Synthesis tool on the RTL model and synthesized the design to a gate-level EDIF netlist

Chapter 10

Common Messages

Error Messages

The following are common error messages that appear in the AccelDSP transcript.

E-ANALYZE-0001

Message: The Script M-file does not exist.

Refer to the topic Examining the Floating-Point Model for more information on the relationship between the script file and the design function file.

E-ANALYZE-0011

Message: Finished analysis of the Script M-file. Some problems where encountered - please correct.

Refer to the topic Examining the Floating-Point Model for more information on the relationship between the script file and the design function file.

E-ANALYZE-0015

Message: Could not find .m file for the function call.

Refer to the topic Examining the Floating-Point Model for more information on the relationship between the script file and the design function file.

E-BUF-0120

Message: Multiple streams writing to the indicated stream buffer. This version of the AccelDSP synthesizer restricts all stream buffers to be written by only one stream.

Remedy: Please remove multiple assignments to the stream buffer by different streams.

A common reason for this message is that a design function input may be incorrectly assigned a value inside the design function body. Consider the following code segments:

Incorrect:

```
function [outdata] = simple (indata1, indata2)
indata1 = indata1 + 2;
outdata = indata1 + 3*indata2;
```

Correct:

```
function [outdata] = simple (indata1, indata2)
tmp = indata1 + 2;
```

```
outdata = tmp + 3*indata2;
```

E-CODESTYLE-0004

Message: An identifier collides with a VHDL keyword. Identifiers that match VHDL keywords are not supported. Please change the name of this identifier.

Refer to the topic VHDL Reserved Keywords for a complete list of reserved keywords.

E-CODESTYLE-0005

Message: An identifier collides with a Verilog keyword. Identifiers that match Verilog keywords are not supported. Please change the name of this identifier.

Refer to the topic Verilog Reserved Keywords for a complete list of reserved keywords.

E-CODESTYLE-0006

Message: A Function Name collides with a VHDL keyword. Function Names that match VHDL keywords are not supported. Please change the name of this function.

Refer to the topic VHDL Reserved Keywords for a complete list of reserved keywords.

E-CODESTYLE-0007

Message: A Function Name collides with a Verilog keyword. Function Names that match Verilog keywords are not supported. Please change the name of this function.

Refer to the topic Verilog Reserved Keywords for a complete list of reserved keywords.

E-CODESTYLE-0025

Message: A directive was set on an input port that is 2-dimensional or greater. Currently there is no support for 2-dimensional matrices on input ports. Please delete your directive.

E-CODESTYLE-0026

Message: The identifier indicated collides with an AccelDSP reserved keyword. Identifiers that match RTL keywords are not supported. Please change the name of this identifier.

Refer to the topic AccelDSP Reserved Keywords for a complete list of reserved keywords.

E-CODESTYLE-0037

Message: Could not find a Design Function to Synthesize. The Design Function has not been specified. Please specify the Design Function via the Properties menu, or the SetDirective -designfunction command.

Refer to the topic Examining the Floating-Point Model for more information on the relationship between the script file and the design function file.

E-CODESTYLE-0046

Message: The use of a MAT-file inside of the design function is not currently supported. The remedy is to load the content of the MAT file into MATLAB, then write the content

back out in ASCII format. For example, assume that your design function has the following line:

```
coeff = load('coefficients.mat');
```

Do the following:

1. Open MATLAB and set the working directory to your design directory
2. Execute the following commands:

```
clear all
coeff = load('coefficients.mat');
save ('coefficients.txt','-ASCII','-double')
```

3. Change the load command in your design function to:

```
coeff = load('coefficients.txt');
```

E-CODESTYLE-0051

Message: The prefix of the variable indicated collides with an AccelDSP reserved prefix. Please rename the variable.

Refer to the topic AccelDSP Reserved Keywords for a complete list of reserved keywords.

E-CODESTYLE-0058

Message: The specified variable is a **struct** data type. Currently, AccelDSP only supports direct assignments and uses between struct members inside the design function.

The AccelDSP synthesis tool provides the following limited support for the **struc** data type:

- Individual elements of a structure can be passed in to or out of a design function port as long as a real-valued scalar or vector is assigned to the element. The passing of a whole structure array is not allowed.
- Inside the design function, individual elements of the structure array can be directly assigned a real-valued data type and used individually. However, a whole operation on a structure array is not allowed. For example, y = mystruct.a + mystruct.b is allowed; y = sum(mystruct) is not allowed.

E-COMMAND-0013

Message: Unable to execute command because target language is not set. Use command SetProjectOption -targetlanguage <language> to set an output language.

Refer to the topic SetProjectOption for complete command syntax.

E-COMMAND-0014

Message: Inner matrix dimensions are not equal for multiply expression.

Refer to the topic Unrolling a Matrix Multiply Operation to get more information on the requirements for inner matrix dimensions.

E-COMMAND-0015

Message: The Design Function was not found. The most common reason for this message is that the name of the streaming variable or design function has changed.

Another reason might be that the instance path specified in the DESIGNFUNCTION directive has not been updated. To re-establish the proper design function instance path, do the following:

1. From the Project Explorer window, right-click on the script file, select **Remove Script**, then click **OK**.
2. Click on the **Verify Floating Point** icon.
3. Select the script file in the **Add Script File** dialog box, then click **OK**.

Verify Floating Point should finish successfully and the design should be ready to Analyze.

E-COMMAND-0019

Message: The argument is not valid for the -targetlanguage switch. Please use a valid argument.

Refer to the topic SetProjectOption for complete command syntax.

E-COMMAND-0023

Message: Unable to set the indicated project option. There is an invalid argument for the currently selected technology. Try using a valid argument or try setting -techvendor, -technology, or -device before setting the indicated argument. Valid devices are listed in the Target toolbar in AccelDSP GUI.

Refer to the topic SetProjectOption for complete command syntax.

E-COMMAND-0024

Message: The indicated environment variable is not set to a valid pathname. Make sure the environment variable is set correctly.

You can use the **Help > System Environment** pulldown menu to bring up the System Environment page. Refer to the topic Verifying the Setup of Other Tools for a list of common settings for environment variables used with the AccelDSP synthesis tool.

E-COMMAND-0025

Message: The indicated environment variable is not set. Please set the environment variable.

Refer to the topic Verifying the Setup of Other Tools for a list of common settings for environment variables used with the AccelDSP synthesis tool.

E-COMMAND-0026

Message: Please specify one of the following switches when running this command -fixedpoint|-rtl|-system_generator_rtl.

Refer to the topic Generate for complete command syntax.

E-DIR-0001

Message: Could not apply the directive to specified instance path. The directive is not applicable to the type of object specified by the instance path.

Refer to the topic SetDirective for complete command syntax.

E-DIR-0006

Message: Directives file does not exist.

The project ADD file is identified by the command SetProjectOption -directivesfile. This command is located in the AccelDSP project file (ACC). Most likely the name of the ADD file specified by this command has changed or the ADD file has been moved or deleted. To recover from this error you can do one of the following:

1. Verify that a valid project ADD file is located in the project directory.
2. Open the project ACC file and change the ADD file name to the valid name. If a valid project ADD file does not exist, delete the command **SetProjectOption -directivesfile** from the project ACC file.

Refer to the topic SetProjectOption for more information on project options that are set in the ACC file.

E-DIR-0009

Message: Could not set directive. No instance path specified.

This message typically appears when you are trying to set a directive on an object from the command console or a Tcl script. One method you can use to identify the correct instance path is to set the directive using the AccelDSP GUI. You can then copy the command line, including the instance path, from the transcript to the Tcl script you are creating or modifying.

Refer to the topic SetDirective for complete command syntax.

E-DIR-0010

Message: Could not set SHAPE directive. No dimensions specified.

Refer to the topic SetDirective for complete command syntax.

E-DIR-0012

Message: Could not apply the MEMMAP Directive. The variable specified appears outside of the design function. Can only map to memory variables that are inside the design function.

Refer to the topic SetDirective for complete command syntax.

E-DIR-0014

Message: Unable to locate a loop in the Script M-file to associate as the input stream.

The top-level design function to be synthesized must be located inside a for loop or a while loop in the script file. Refer to the topic Examining the Floating-Point Model for more information on the relationship between the script file and the design function file.

E-DIR-0029

Message: Could not set shape directive. Could not set shape directive as a list of integers was expected.

Refer to the topic SetDirective for complete command syntax.

E-DIR-0031

Message: MEMMAP directive is not associated with a variable inside the hardware function. Please specify the instance path to a variable inside the design function.

Refer to the topic SetDirective for complete command syntax.

E-DIR-0033

Message: Could not delete ignorefunction. A function name was not specified. Please specify a function name that has been previously set to ignore.

MATLAB functions to ignore are set with the command SetGlobalOption. A default list of functions to ignore can be found in the initialization file at pathname **<AccelDSP_tree>/configure/AccelDSPInit.tcl**. You can get a list of all functions to ignore by typing the command GetGlobalOption -ignorefunction in the Tcl Console.

E-DIR-0035

Message: Invalid sublist in quantizer directive, or invalid number in format sublist. Sublists can only be associated with the format portion.

Refer to the topic SetDirective for complete command syntax and valid options.

E-DIR-0037

Message: The specified mode string specified is not a valid quantizer mode string.

Refer to the topic SetDirective for complete command syntax and valid options.

E-DIR-0038

Message: The quantizer specified is not a valid quantizer.

Refer to the topic SetDirective for complete command syntax and valid options.

E-DIR-0044

Message: FOR Loop cannot be unrolled. The Begin, Step, or End Values are not constants.

Refer to the topic Unrolling a Loop to Increase Hardware Performance for more information.

E-DIR-0045

Message: FOR Loop cannot be unrolled. The number of iterations is NOT an integer value.

Refer to the topic Unrolling a Loop to Increase Hardware Performance for more information.

E-DIR-0046

Message: FOR Loop cannot be unrolled. The Index of the FOR Loop is redefined inside the body of the FOR Loop.

Refer to the topic Unrolling a Loop to Increase Hardware Performance for more information.

E-DIR-0053

Message: Wrong number of parameters specified to unroll of an element operation. You need to specify row and column.

Refer to the topic SetDirective for complete command syntax and valid options.

E-DIR-0054

Message: Wrong number of parameters specified to unroll a matrix multiplication. You need to specify row, column, and inner product.

Refer to the topic SetDirective for complete command syntax and valid options.

E-DIR-0055

Message: Initial value of the specified variable is not zero. The -connectreset directive can only be applied to a variable that has an initial value of zero.

Refer to the topic SetDirective for complete command syntax and valid options.

E-DIR-0060

Message: Unable to source hierarchical directives file.

Refer to the topic Understanding Hierarchical Directives for more information on using and overriding hierarchical directives.

E-DIR-0062

Message: Unable to load hierarchical directives file. No relative instance path specified. Refer to the topic Understanding Hierarchical Directives for more information on creating and overriding hierarchical directives.

E-DIR-0063

Message: Unable to delete hierarchical directive at specified instance path. Currently the tool does not support the deletion of hierarchical directives. Please remove the directive from it's hierarchical directives file and redo the Analyze flow step.

Refer to the topic Understanding Hierarchical Directives for more information on using and overriding hierarchical directives.

E-DIR-0070

Message: Invalid **-insertpipestage** number for specified instance path.

Refer to the topic Improving Performance with an Insertpipestage Directive for more information on using this directive.

E-DIR-0071

Message: Setting -insertpipestage directive to a variable is no longer supported for the specified object. Try to set the directive to a binary expression on the right-handed side of the assignment.

Refer to the topic Improving Performance with an Insertpipestage Directive for more information on using this directive.

E-DIR-0073

Message: The setting of project options in a hierarchical add file is not allowed.

You should set the project options from the AccelDSP GUI, then save the project or enter SetProjectOption commands directly into the project ACC file.

E-DIR-4008

Message: Unrecognized memory resource for memory-mapping the variable. Please provide a valid memory resource identification or double-check the MEMMAP directive syntax.

Refer to the topic SetDirective for complete command syntax and valid options.

E-EXPORT-0004

Message: In System Generator clk, rst, and ce are reserved port names and cannot be used. Please change any port names that conflict with the reserved names.

Refer to the topic Selecting the System Generator Flow for more information about exporting to System Generator.

E-MAT-0001

Message: Unable to infer the shape of the indicated variable. The synthesizer was unable to infer the shape of the variable purely based on context analysis. Please add a zeros call at the beginning of the file to define the shape of the variable.

For example, if the AccelDSP synthesis tool is having trouble determining the shape of X, you should include a zeros statement similar to the following prior to assigning values to X.

```
X = zeros(4,1);
```

This statement defines X as a column vector with four elements.

This error message may also commonly appear after the following Warning message:

```
#(W-MAT-0701):Unable to open or load the file referred to by "load()"
statement. Ignoring the "load()" statement.
```

The AccelDSP synthesis tool could not determine the shape of the variable because the file specified in the load() command is missing or its name has changed. You can restore the file to be loaded or you can add a zeros call at the beginning of the file to define the shape of the variable.

E-MAT-0015

Message: Expecting the arguments to be an array. The synthesizer was unable to infer the data type of the argument.

Remedy: Please provide a SHAPE directive for the argument if appropriate.

Refer to the topic SetDirective for complete command syntax of the SHAPE directive.

E-MAT-0021

Message: No Script M-file provided. Unable to proceed.

Refer to the topic Examining the Floating-Point Model for more information on the relationship between the script file and the design function file.

E-MAT-0033

Message: Unable to evaluate the shape of the indicated variable during synthesize time. Please provide a SHAPE directive.

Refer to the topic SetDirective for complete command syntax of the SHAPE directive.

E-MAT-0037

Message: Unable to evaluate the specified UNROLL value at synthesize time.

Refer to the topic How to Apply the Unroll Directive to a for Loop for information on how to set the unroll value from the AccelDSP GUI. Refer to the topic SetDirective for complete command syntax of the UNROLL directive.

E-MAT-0068

Message: Unable to proceed with synthesis. Please provide SHAPE directive(s) for the indicated item(s) or use supported data types.

Refer to the topic SetDirective for complete command syntax of the SHAPE directive.

E-MAT-0069

Message: The identifier indicated may conflict with a VHDL keyword. Please rename it.

Refer to the topic VHDL Reserved Keywords for a complete list of reserved keywords.

E-MAT-0070

Message: The identifier indicated may conflict with a Verilog keyword. Please rename it.

Refer to the topic Verilog Reserved Keywords for a complete list of reserved keywords.

E-MAT-0072

Message: Mismatch in analyzed shape and directive shape for the indicated variable. Directive shape must be equal to or larger than the analyzed shape.

Refer to the topic SetDirective for complete command syntax of the SHAPE directive.

E-MAT-0077

Message: "Unable to infer shape of the indicated expression. Please provide a SHAPE directive for the expression's context.

Refer to the topic SetDirective for complete command syntax of the SHAPE directive.

E-MAT-0081

Message: Unable to infer shape of the indicated variable.

Refer to the topic SetDirective for complete command syntax of the SHAPE directive.

E-MAT-0093

Message: Unrecognized bitwise operation.

Refer to the Chapter AccelDSP Base Library Functions for details on valid accel_bitwise functions.

E-MAT-0105

Message: Expecting two variables on LHS of accel_bitsplit assign.

Refer to the topic accel_bitsplit for details on this function.

E-MAT-0108

Message: Unable to infer the complete-unroll factor for the 'for' loop range expression.

Refer to the topic How to Apply the Unroll Directive to a for Loop for information on how to set the unroll value from the AccelDSP GUI. Refer to the topic SetDirective for complete command syntax of the UNROLL directive.

E-MAT-0132

Message: Incorrect shape for accel_bitunpack output.

Refer to the topic accel_bitunpack for details on this function.

E-MAT-0601

Message: Explicit quantizing the outputs of a function is illegal. Please quantize the output variables inside the function instead.

Refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value for information on how to set a quantize value from the AccelDSP GUI. Refer to the topic SetDirective for the complete command syntax of the quantize directive.

E-MAT-0750

Message: Unable to analyze the shape of multiplication operands. The synthesizer was unable to infer the shape of matrices being multiplied.

Remedy: Please provide SHAPE directives for each operand if applicable.

Refer to the topic SetDirective for complete command syntax of the SHAPE directive.

E-MAT-0751

Message: Unable to analyze the shape of the left operand in multiplication. The synthesizer was unable to infer the shape of left matrix in the multiplication.

Remedy: Please provide SHAPE directives for the left operand if applicable.

Refer to the topic SetDirective for complete command syntax of the SHAPE directive.

E-MAT-0752

Message: Unable to analyze the shape of the right operand in multiplication. The synthesizer was unable to infer the shape of left matrix in the multiplication.

Remedy: Please provide SHAPE directives for the left operand if applicable.

Refer to the topic SetDirective for complete command syntax of the SHAPE directive.

E-MAT-0951

Message: Could not find an initial value for the specified persistent variable. Please ensure that you are using a conditional assignment on “isempty”.

Incorrect:

```
persistent A;
isempty(A)
A = 1;
```

Correct:

```
persistent A;
if isempty(A)
A = 1;
end
```

E-MAT-0952

Message: Could not find an initial value for the specified persistent variable.

Message: Please include an isempty() clause within a conditional statement that sets the initial value.

For hardware synthesis, every persistent variable must have an associated isempty() statement that sets the initial value.

Incorrect:

```
persistent A;
```

Correct:

```
persistent A;
```

```
if isempty(A)
A = 1;
end
```

E-MAT-1004

Message: Cannot use different shapes for the indicated persistent variable. In each instance of the function all persistent variables must have the same shape.

Refer to the topic SetDirective for complete command syntax of the SHAPE directive.

E-QTZ-0001

Message: Unable to fully quantize the design. Found constructs in the design whose quantizations could not be calculated or encountered errors.

Remedy: Please apply directives to the items that were not quantized (see fixed point report or previous warnings).

Refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value or you can use the SetDirective command to set a valid quantize value from the Tcl Console.

E-QTZ-0007

Message: Unsupported quantizer in MATLAB source. Explicit quantization of input function arguments within the function definition is currently not supported. Please modify your MATLAB source and remove the explicit quantizer and use a directive instead.

Consider the simple example below:

Incorrect

```
function [outdata] = hw_design (indata)
q = quantizer( 'ufixed', 'floor', 'wrap', [8 0] );
indata = quantize(q,indata + 1); quantization applied to var on left s
outdata = indata;
```

Input port cannot be explicitly quantized inside the design function

Correct

```
function [outdata] = hw_design (indata)
outdata = indata + 1;
```

In this example, the correct method is to remove the explicit quantization of indata, then apply a quantize directive using the AccelDSP GUI or the Tcl Console. In this case, the correct command from the Tcl Console is as follows:

```
SetDirective -quantize {for n.hw_design.Ports.Inputs.indata} {ufixed
floor wrap {8 0}}
```

Refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value or you can use the SetDirective command to set a valid quantize value from the Tcl Console.

E-QTZ-0012

Message: Invalid unsigned quantizer directive applied. The word length is greater than the tool's allowed design function I/O size of 53 bits unsigned or 54 bits signed.

Refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value or you can use the SetDirective command to set a valid quantize value from the Tcl Console.

E-QTZ-0033

Message: Invalid parameter for SetDirective -quantize command.

Refer to the SetDirective command for valid quantize parameters.

E-QTZ-0034

Message: Wrong number of parameters for word and fractional length.

Refer to the SetDirective command for valid quantize parameters.

E-QTZ-0035

Message: Wrong number of parameters for SetDirective -quantize.

Refer to the SetDirective command for valid quantize parameters.

E-QTZ-0100

Message: Re-quantizing the bitwise output is not supported in this release. The bitwise function call can accept a separate output quantizer as one of its trailing arguments.

Remedy: Please relocate the output quantization as part of bitwise function call argument instead of using the quantize() wrapper.

E-SYN-0002

Message: Cannot use more than one Script M-file in the same design. A design may not consist of more than one Script M-file.

Refer to the topic Examining the Floating-Point Model for more information on the relationship between the script file and the design function file.

E-SYN-0399

Message: Unknown directive. Please review the directive syntax.

Refer to the SetDirective command for valid syntax and options.

E-TOOLGEN-0005

Message: Could not find correct file for place and route. Please ensure that your XILINX environment variable is setup correctly.

Refer to the topic Verifying the Setup of Other Tools for settings common to the XILINX environment variable.

E-VERIFY-0005

Message: Simulation of the current design Failed. This is most likely due to a pipeline register in a feedback loop.

Remedy: Remove one or more pipeline registers in the feedback path until the Verify RTL step passes.

E-VERIFY-0007

Message: Simulating VHDL design: ISE Simulator Failed. This is due to a pipeline register in a feedback loop.

Remedy: Enable the Array Access Guard for feedback variable(s) indicated in the accel_verify.log file and rerun Generate -fixedpoint.

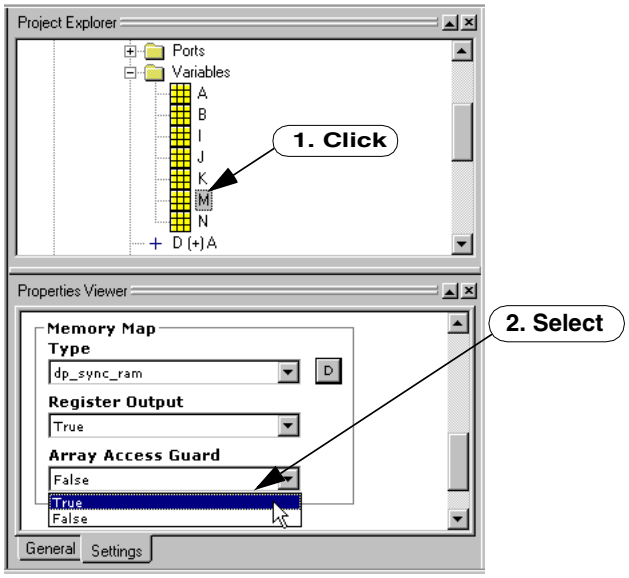
Assume that you have several array variables in your design and you disable the Array Access Guards to allow AccelDSP to more aggressively pipeline the design. This automatic pipelining may improve the design throughput, but may also cause simulation mismatches. You then re-run the AccelDSP flow and see the following message after the Verify RTL step:

```
#( E-VERIFY-0007): Simulating VHDL design: Modelsim Failed.  This is due
to a pipeline register in a feedback loop.
# Remedy: Enable the Array Access Guard for feedback variable(s)
indicated in the accel_verify.log file and rerun Generate -fixedpoint.
```

Following the directions in the message, you scroll up in the Transcript window and see the following message from the ISE Simulator:

```
# ** Warning: Encountered array conflict - Please enable the Array
Access Guard for variable 'M'.
```

To remedy the conflict, you then select the variable M in the Project Explorer window, as shown below, and set the Array Access Guard in the Properties Viewer window back to **True**.



Disabling the Array Access Guard on Array Variables may or may not cause simulation conflicts. Running the Verify RTL step after disabling the guard is the only way to tell.

Warning Messages

The following are common warning messages that appear in the AccelDSP transcript.

W-BUF-0200

Message: Function argument is neither array nor quantizer type. The function argument must be either array type or quantizer type.

Remedy: Please provide a SHAPE directive if appropriate, or use a supported data type.

Refer to the SetDirective command for valid SHAPE directive syntax.

W-CODESTYLE-0001

Message: The use of "a cell array" in the script file can lead to unexpected behavior.

Remedy: Comment out, change, or remove the use of the cell array in the script file.

The use of a MATLAB cell array in the script file is not supported at this time. AccelDSP issues a warning when encountered but not error. The use of a cell array to modify data to or from the design function can cause incorrect behavior and should be avoided.

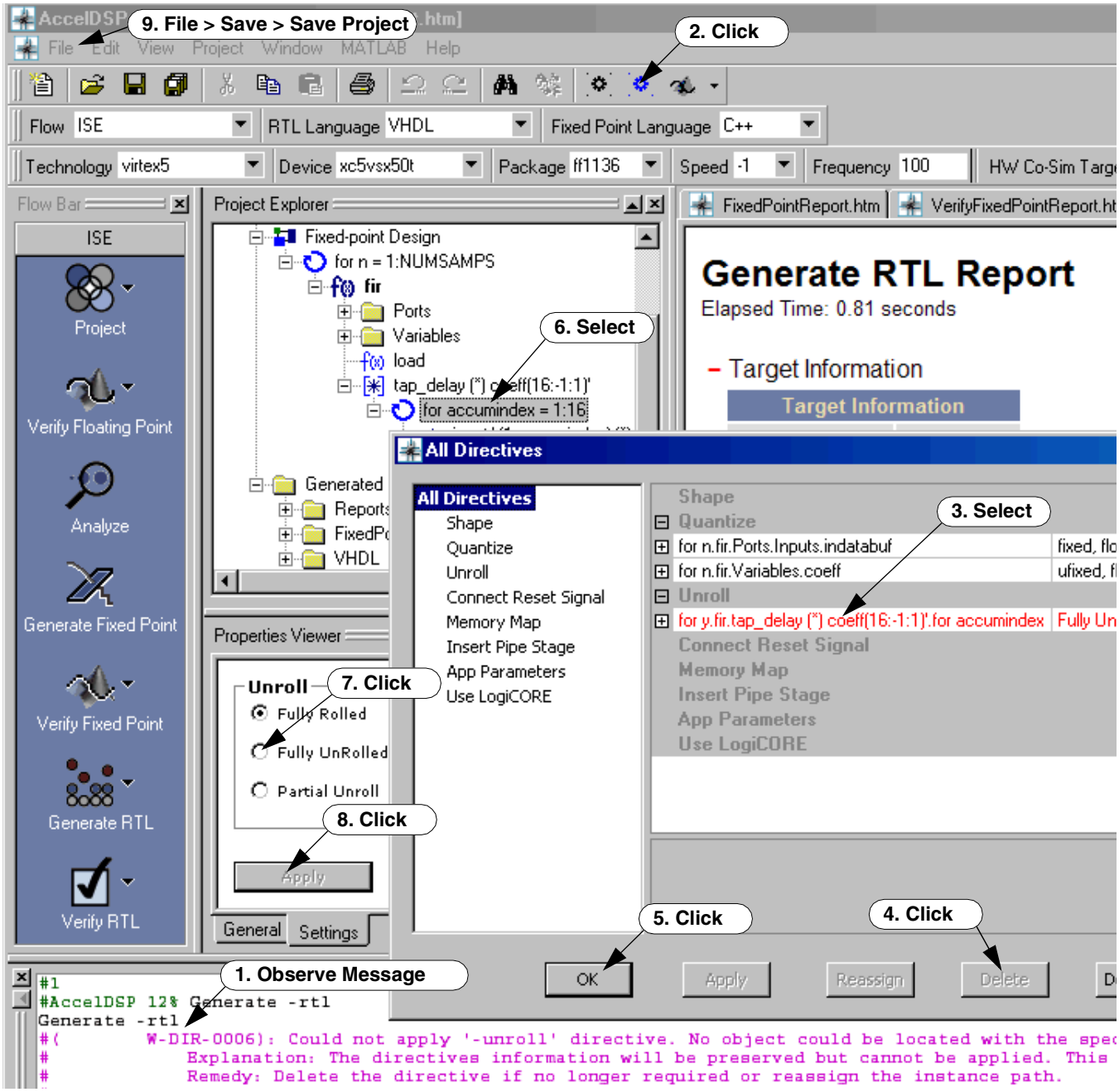
W-DIR-0006

Message: Could not apply <named> directive. No object could be located with the specified instance path: <instance_path>

Explanation: The directives information will be preserved but cannot be applied. This can be a side effect of name and/or hierarchy changes.

Remedy: Delete the directive if no longer required or reassign the instance path.

The figure below shows a design that has an unroll directive defined in the ADD file with an incorrect instance path. To see this directive more closely, you can clicked on the "blue" snow flake to bring up the All Directives dialog box. As shown in the figure, the uncorrectly specified directive is highlighted in red.



To correct the directive instance path, you can select the instance path highlighted in red, click on **Delete**, then click **OK**. You can then re-select the loop in the Project Explorer window, click **Fully UnRolled** in the Properties Viewer, then click **Apply**. To save the correct instance path to the ADD file, select **File** > **Save** > **Save Project**.

W-DIR-0013

Message: Ignoring the specified Unsupported Argument after the name of the memory in the MemMap Directive. The syntax for the MEMMAP directive is SetDirective -memmap <memory_name> {<instance_path>}.

Refer to the SetDirective command for valid MEMMAP directive syntax and options.

W-DIR-0016

Message: The specified Memory in the SetDirective -memmap command is not a supported memory. Will replace the memory "dp_sync_ram", which is a Technology Independent Dual-Port RAM. In the future, please specify one of the support memories.

Refer to the SetDirective command for valid MEMMAP supported memories.

W-DIR-5001

Message: The specified unroll factor is not a factor of the number of iterations of the loop. The 'for' loop will not be unrolled.

Refer to the topic Unrolling a Loop to Increase Hardware Performance for more information.

W-MAT-0008

Message: Unable to analyze the shape of the specified variable as an element of array expression.

Remedy: SHAPE directives will be required on the elements of the expression.

Refer to the SetDirective command to set a SHAPE directive from the Tcl Console.

W-MAT-0301

Message: Project Option specified is not supported. This option will be ignored.

Refer to the topic SetProjectOption for a list of valid options.

W-QOR-0400

Message: The specified constant expression results in a large quantization value during Multiplication. This can result in a mismatch between the MATLAB Simulation and the VHDL/Verilog Simulation.

Remedy: It is recommended to apply a quantize directive to the constant to reduce the number of bits required to represent it.

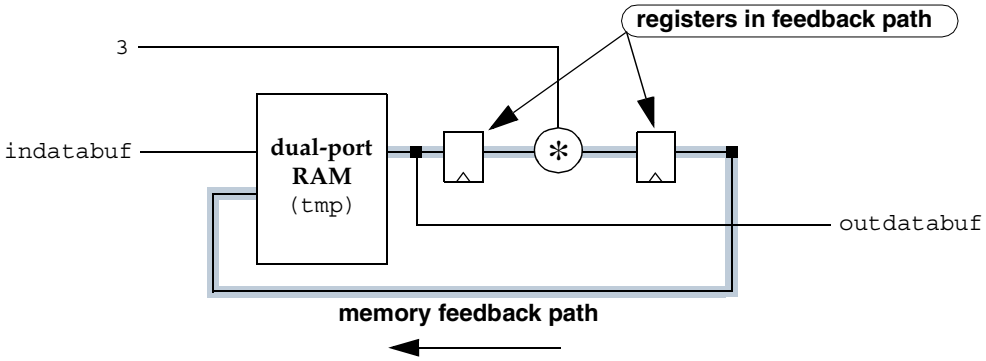
Refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value or you can use the SetDirective command to set a valid quantize value from the Tcl Console.

W-QOR-0901. ***Message:*** Pipeline registers have been detected in a feedback loop containing a memory-mapped variable. This may cause a verification mismatch.

Explanation: Consider the feedback path that is created for the design function below:

```
function outdatabuf = simplemath(indatabuf)
  tmp = indatabuf;
  for n = 1:5
    tmp(n+1) = tmp(n) * 3
  end;
  outdatabuf = tmp;
```

In this example, `indatabuf` is a 10-element vector. The design function assigns the input vector to `tmp` (which is memory-mapped variable). Each iteration of the `for` loop then reads an element of tmp, multiplies by the constant value `3`, then assigns the result to the next location in memory. This is done for the first 5 elements in the vector.



Assume that the two registers in the feedback path around the multiplier have been added with an `insertpipestage` directive. When AccelDSP detects the feedback path containing registers, this W_QOR_0901 Warning message is issued.

Remedy: Proceed to the Verify RTL step and if it fails, remove one or more pipeline registers in the feedback path until the Verify RTL step passes.

W-QTZ-0006

Message: The quantization word length on the specified variable is greater then the MathWorks accuracy limit of 53 bits.

Apply a quantize directive to reduce the word width to less than 53 bits.

For more information, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value.

W-QTZ-0010

Message: Invalid quantizer directive applied to the specified variable. The word length is greater then the MathWorks accuracy limit of 53 bits.

Apply a quantize directive to reduce the word width to less than 53 bits.

For more information, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value.

W-QTZ-0011

Message: Invalid quantizer directive applied to the specified variable. The word length is greater than AccelDSP's allowed design function I/O size of 53 bits unsigned or 54 bits signed.

Apply a quantize directive to reduce the word width to less than 53 bits unsigned or 54 bits signed.

For more information, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value.

W-QTZ-0020

Message: Invalid MATLAB source quantizer applied to the specified variable. The word length is greater then the MathWorks accuracy limit of 53 bits.

For more information, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value.

W-QTZ-0021

Message: Invalid MATLAB source quantizer applied to the specified variable. The word length is greater than AccelDSP's allowed design function I/O size of 53 bits unsigned or 54 bits signed.

For more information, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value.

W-QTZ-0035

Message: Circular dependencies of quantizations detected between the variables specified. The Auto-Quantizer cannot determine optimal quantization for these variables.

Remedy: A quantization directive on at least the following specified variable(s) is advised.

Refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value or you can use the SetDirective command to set a valid quantize value from the Tcl Console.

W-QTZ-0036

Message:: The result of expression exceeds 53 bits. Additional hardware will be added to perform convergent rounding as specified in the IEEE 754 standard.",

Remedy: To avoid this situation, quantize the expression explicitly.

Refer to SetProjectOption **-ieee_754_rounding_threshold** for a more indepth discussion of this topic. For setting quantize values, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value or you can use the SetDirective command to set a valid quantize value from the Tcl Console.

W-QTZ-0037

Message: The result of specified expression exceeds 53 bits. Additional hardware will be added to perform convergent rounding as specified in the IEEE 754 standard.

Remedy: Set the number of fractional bits on the expression to no more than the amount specified or insure that the quantization on the operands is optimal.

Reduce quantization so the number of dropped bits is greater than the IEEE 754 Rounding Threshold project option. You may also decrease the IEEE 754 rounding threshold but you may cause the RTL not to be bit true.

Refer to SetProjectOption **-ieee_754_rounding_threshold** for a more indepth discussion of this topic. For setting quantize values, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value or you can use the SetDirective command to set a valid quantize value from the Tcl Console.

W-QTZ-0100

Message: Cannot infer quantization. Please apply a directive. If this is a sub-expression then assign it to a temporary then apply the quantization.

For setting quantize values, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value or you can use the SetDirective command to set a valid quantize value from the Tcl Console.

W-QTZ-0101

Message: Cannot infer quantization from the specified variable. Please apply a directive.

For setting quantize values, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value or you can use the SetDirective command to set a valid quantize value from the Tcl Console.

W-QTZ-0102

Message: Cannot infer quantization from the specified undefined variable. Please apply a directive.

For setting quantize values, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value or you can use the SetDirective command to set a valid quantize value from the Tcl Console.

W-QTZ-0401

Message: The Auto-Quantizer has reduced the word width of this top-level design function input or output argument to 53 bits unsigned or 54 bits signed to conform to the AccelDSP limit. In so doing, all fractional bits (and possibly some integer bits) have been removed. To avoid this automatic truncation, you can apply one or more quantize directives to selectively reduce the word width to 53 bits unsigned or 54 bits signed. For more information, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value.

W-QTZ-0402

Message: The Auto-Quantizer has reduced the word width of this data value to '53' bits in order to conform to the MATLAB fixed-point arithmetic limit of '53' bits for bit-true MATLAB simulations. In so doing, all fractional bits (and possibly some integer bits) have been removed. To avoid this automatic truncation, you can start with the input ports and selectively apply quantize directives to reduce the word width and bit growth of data feeding into this computation. For more information, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value.

W-QTZ-0403

Message: In order to minimize quantization error, the Auto-Quantizer would have selected the specified number of bits for input, however the design-function arguments are limited to 53 bits unsigned or 54 bits signed and fewer.

To avoid this automatic truncation, you can apply one or more quantize directives to selectively reduce the word width to 53 bits unsigned or 54 bits signed. For more information, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value.

W-QTZ-0404

Message: In order to minimize quantization error, the Auto-Quantizer would have selected the specified number of bits for the argument, however bit-true MATLAB fixed-point arithmetic is limited to 53 bits and fewer.

To avoid this message, you can start with the input ports and selectively apply quantize directives to reduce the word width and bit growth of data feeding into this computation. For more information, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value.

W-TBG-0002

Message: Port specified requires more than 53 bits unsigned or 54 bits signed representation. RTL simulation is not guaranteed to produce bit-true results.

Refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value or you can use the SetDirective command to set a valid quantize value from the Tcl Console.

Information Messages

The following are common information messages that appear in the AccelDSP transcript and reports.

I-ANALYZE-0011

Message: Loading hierarchical directives file for function instance.

Refer to the topic Understanding Hierarchical Directives to learn how to create a hierarchical directives file and override a hierarchical directive with a project-level directive.

I-DESIGNFUNC-0001

Message: No design function specified; will assume function with the same basename as the project. To override use SetDirective -designfunction.

You have not explicitly indicated to the AccelDSP synthesis tool which function call in the script is the top-level design function. In this case, the tool assumes that the name of the design function is the same name as the project. If this is not the case, you can specify the correct top-level design function by entering a SetDirective -designfunction command in the AccelDSP ADD file. A typical entry might be as follows:

```
SetDirective -designfunction {for n.fir}
```

I-DESIGNFUNC-0002

Message: The Design function is currently set to the function indicated.

Refer to the topic Examining the Floating-Point Model for more information on the relationship between the script file and the design function file.

I-EXPORTSIMULINK-0002

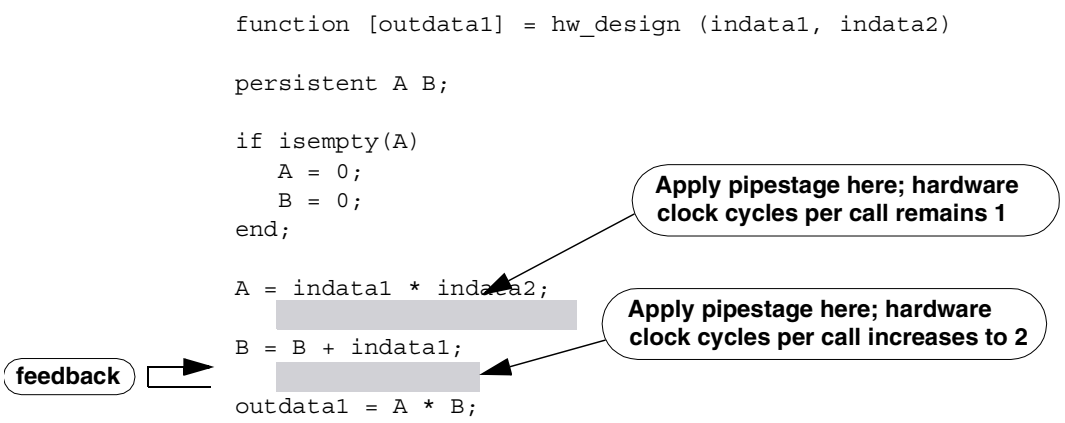
Message: Successfully exported current design to a Simulink block. To use this new Simulink block, add the block to your MATLAB path.

For information on how to add a block or a function to the MATLAB search path, refer to the topic Adding AccelDSP MATLAB Functions to the MATLAB Search Path.

I-QOR-0201

Message: The number of hardware cycles per function call will be affected because an Insert Pipe Stage directive was applied inside of the feedback loop caused by a variable assignment. You may correct the problem by apply the Insert Pipe Stage directive outside of the feedback loop or remove the feedback altogether.

The purpose of applying an Insert Pipe Stage directive is to increase the frequency of the design by breaking the critical data path into two or more parts. Consider the design function below:



The data path in this design contains a multiplier, an adder, and another multiplier. When you apply an insertpipestage directive to the subexpression containing the first multiplier, the tool places a register between that multiplier and the adder; hardware clock cycles per function call remains at 1. When you apply an insertpipestage directive to the subexpression containing the adder, the tool places the register between the adder and the second multiplier. Because this subexpression is in a statement containing feedback, hardware clock cycles per function call increases to 2. In both cases, frequency is increased because the multipliers are placed in difference clock cycles, however, in the second case, the number of hardware cycles per function call is also increased because of feedback in the statement.

I-QTZ-0024

Message: The Auto-Quantizer cannot determine the quantization necessary to minimize the quantization error of "sum". Quantization of persistent variables that are assigned to themselves (feedback) currently cannot be accurately determined.

Remedy: Please check the quantization of "sum" and apply a quantize directive if necessary.

Consider the persistent variable sum in the following design function:

```
function [outdata1] = simple(A)
persistent sum
if isempty(sum)
  sum = 0;
end;
sum = sum + A;
outdata1 = sum;
```

When the tool analyzes this function, there is no way to determine how many times the function will be called, so the persistent variable sum may grow to any size. the tool makes an attempt to assign a reasonable word width, however, there is a danger that sum will overflow if the function is called too many times. As the designer, it is best for you to estimate the maximum word width of sum and apply a quantize directive to the variable. To learn how, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value.

I-QTZ-0025

Message: The Auto-Quantizer cannot determine the quantization necessary to minimize the quantization error of "tmp". Quantization of variables that are assigned to themselves (feedback) inside a loop currently cannot be accurately determined.

Remedy: Please check the quantization of "tmp" and apply a quantize directive if necessary.

Consider the variable *tmp* in the following design function:

```
function [outdata1] = simple(A,B,C)
tmp = B;
for I = 1:A
  tmp = tmp + C;
end;
outdata1 = tmp;
```

When the tool analyzes this function, there is no way to determine how many times the for loop will iterate, so the variable tmp may grow to any size. the tool makes an effort to assign a reasonable word width to tmp, however, there is a danger that tmp will overflow if the loop iterates too many times. As the designer, it is best for you to estimate the maximum word width of tmp and apply a quantize directive to the variable. To learn how, refer to the topic Using Variables by Hierarchy Tab to Adjust a Quantizer Value.

Accounting for Latency in the Hardware Co-Sim Script File

The Hardware Co-Sim Script File is a copy of the original MATLAB script file that compares the Floating Point Simulation Results with the Fixed Point Simulation Results. In the example shown below, the script file produces an error if the difference is more than 0.1%, a relative difference of "0.001". If you are using the Hardware Co-Simulation flow, then you may want to use and modify this file after the first simulation run to add additional test vectors, since simulation speed is very fast. In the file shown below, this can be accomplished by changing "len" to be a value greater than "256", e.g. "len = 1024;"

Under most conditions, you will not have to modify the Hardware Co-Sim Script File to adjust for hardware latency. However, if your design has a Latency whose value is greater than the Throughput, you MUST modify the script file to prevent the comparison of invalid-to-valid data. For example, suppose the design has Latency (Number of Startup Cycles) of 3, and a Throughput of 1. Within the hardware simulation, the first piece of valid output data will be on the third clock cycle. The first two cycles output invalid data. As such, the comparison between the Floating Point data and the Fixed Point (hardware) will be skewed. The first set of data from the Floating Point has to be compared against the third set of data from the hardware. Therefore, the comparison becomes the difference between "<golden_from_floating>(1:end-2) - <data_from_hardware>(3:end)".

```
% Generate input stimulus
len = 256;
indata1 = [0.9*sin([0:(pi/64):(pi*len/64)]) ;
           0.9*cos([0:(pi/48):(pi*len/48)])];
num_vectors = size(indata1,2);
for n = 1:num_vectors
    outdata1(:,n) = hw_design( indata1(:,n) );
end
%Save golden vectors
%save golden_output.txt outdata1 -ascii -double
% Load golden vectors
golden_output = load('golden_output.txt');
% Compute difference vs. golden data
%diff1 = golden_output(1,1:end)-outdata1(1,1:end);
%diff2 = golden_output(2,1:end)-outdata1(2,1:end);
diff1 = golden_output(1,1:end-2)-outdata1(1,3:end);
diff2 = golden_output(2,1:end-2)-outdata1(2,3:end);
num_diff_vectors = length(diff1);
% Generate plots
subplot(3,1,1);
plot(1:num_vectors,indata1(1,:),1:num_vectors,indata1(2,:));
title('Inputs');
subplot(3,1,2);
plot(1:num_vectors,outdata1(1,:),1:num_vectors,outdata1(2,:));
title('Outputs');
subplot(3,1,3);
plot(1:num_diff_vectors,diff1,1:num_diff_vectors,diff2);
title('Difference vs. Golden');
% Error if actual results are not close to (+/- quantization noise)
golden
error_tolerance = 0.001;
if (max(abs(diff1)) > error_tolerance) | (max(abs(diff2)) >
error_tolerance)
   error('\nActual output differs from golden vectors by more than %f',
error_tolerance);
end
```

Last two data samples in the golden source are ignored.

Comparision starts by subtracting output sample 3 from sample 1 in the golden source.

XILINX®

Chapter 11

Reserved Words

AccelDSP Reserved Keywords

The following list of AccelDSP keywords cannot be used as MATLAB variable names for synthesis:

ac_fixed
ac_ufixed
accel_resize
clock
pad
reset

MATLAB Reserved Keywords

The following list of MATLAB keywords are not valid variable names for synthesis:

eye
false
ones
true
zeros

VHDL Reserved Keywords

The following list of VHDL keywords are not valid variable names for synthesis:

Table 11-1: **Reserved VHDL Keywords**

abs	else	label	positive	srl
access	elsif	library	postponed	std_logic
after	end	linkage	procedure	std_logic_vector
alias	entity	literal	process	std_ulogic
all	error	loop	pure	std_ulogic_vecto
and	exit	map	range	r
architecture	failure	mod	real	subtype
array	false	nand	record	string
assert	file	natural	register	then
attribute	file_open_kind	new	reject	time
begin	file_open_status	next	rem	to
bit	for	nor	report	transport
bit_vector	function	not	return	type
block	generate	note	reverse_range	unaffected
body	generic	null	rol	units
boolean	group	of	ror	until
buffer	guarded	on	select	use
bus	if	open	severity	variable
character	impure	or	severity_level	wait
case	in	others	signal	warning
component	inertial	out	shared	when
configuration	inout	package	sla	while
constant	integer	pad	sll	with
disconnect	is	port	sra	xnor
downto				xor

Verilog Reserved Keywords

The following list of Verilog keywords are not valid variable names for synthesis:

Table 11-2: **Reserved Verilog Keywords**

always
and
assign
attribute
automatic
begin
buf
bufif0
bufif1
case
casex
casez
cell
cmos
config
deassign
default
defparam
design
disable
edge
else
end
endattribute
endcase
endconfig
endfunction
endgenerate
endmodule
endprimitive
endspecify
endtable
endtask
event
for
force
forever
fork
function
generate
genvar
highz0
highz1
if
ifnone
initial
inout
input
instance
integer
join
large
liblist
localparam
macromodule
medium
module
nand
negedge
nmos
nor
noshowcancelled
not
notif0
notif1
or
output
parameter
pmos
posedge
primitive
pull0
pull1
pulldown
pullup
pulsestyle_ondetect
pulsestyle_onevent
rcmos
real
realtime
reg
release
repeat
rnmos
rpmos
rtran
rtranif0
rtranif1
scalared
showcancelled
signed
small
specify
specparam
strength
strong0
strong1
supply0
supply1
table
task
time
tran
tranif0
tranif1
tri
tri0
tri1
triand
trior
trireg
unsigned
use
vectored
wait
wand
weak0
weak1
while
wire
wor
xnor
xor

Glossary

Click on a letter, or scroll down to view the entire glossary.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

AccelDSP ISE Synthesis Flow

The flow consisting of the following basic steps:
1) Examine the Style of the Floating-Point Model 2)Open/Create Project 3) Analyze 4) Verify Floating Point 5) Generate Fixed Point 6)Verify Fixed Point 7) Generate RTL 8) Verify RTL 9) Synthesize RTL 10) Implement and 11) Verify Gate Level

ACC file

AccelDSP Control file. Also called a Project file. This control file is created and placed in the project directory when an AccelDSP project is created. This file keeps track of project details like the pathnames to MATLAB design files and project options that the user may have set. There can be only one ACC file for each project.

ADD file

AccelDSP Design Directives file. This file contains SetDirective commands that guide AccelDSP Synthesis in synthesizing the design into hardware. You can use directives to change the architecture of the resulting hardware as well as map specific language constructs like memory elements to specific hardware structures like embedded RAM.

Analyze step

A step in the AccelDSP flow where the project design files are evaluated and an in-memory data structure is built that represents the design to be synthesized.

auto-quantize

An automatically executed step in the AccelDSP flow. In preparation for hardware synthesis, floating-point numbers in MATLAB are converted to fixed-point numbers.

B

bit-true

The output results of the RTL Model match bit-for-bit with the output results of the MATLAB fixed-point model.

C

D

design directive

A design choice that you pre-specify in the GUI to guide the AccelDSP synthesis tool during the hardware synthesis process. The choices include things like how much to unroll a loop as well as how to map storage constructs like variables to hardware memory elements like RAM (random access memory).

design function

This term refers to the top-level function in the script file that represents the design to be synthesized into hardware. Other functions at this level in the script file are referred to as support functions. The top-level design function may call other functions in the script file or other MATLAB function files.

design verification

In the AccelDSP Synthesis environment, this term refers to running an HDL simulation on the synthesized hardware (the DUT) using the generated Testbench. The fixed-point MATLAB verification you perform on the design before synthesis is called functional verification.

DUT

Design Under Test. In the AccelDSP environment, the design function (in MATLAB code) is synthesized into a hardware module which is referred to as the design under test (DUT). The DUT is written in a hardware description language (HDL), either VHDL or Verilog. The functionality of the DUT is verified by the HDL Testbench which is generated by AccelDSP.

E

F

fidelity

In the AccelDSP flow, a MATLAB fixed-point model is automatically generated from the “golden” MATLAB floating-point model. Within this context, the term “fidelity” refers to the degree to which the MATLAB simulation results of the fixed-point model matches the MATLAB simulation results of the floating-point model.

fixed-point model

A design function in MATLAB that uses numeric values in fixed-point notation. In the AccelDSP synthesis tool, a fixed-point model can be automatically generated from a floating-point model, but this process sometimes requires manual intervention to achieve the most accurate results.

floating-point model

A design function in MATLAB that uses numeric values in floating-point notation. A floating-point MATLAB design must be converted to fixed point before the AccelDSP synthesis tool can synthesize the design into hardware.

function M-file

A MATLAB M-file that defines a MATLAB function. In the AccelDSP synthesis tool, the body of the top-level design function M-file is synthesized into the main block of hardware. Function calls from this top-level M-file to other function M-files are synthesized into sub-blocks of hardware.

functional verification

In the AccelDSP design flow, you perform functional verification by including verification constructs in your MATLAB script file. When you run the script in MATLAB, the verification constructs apply stimulus to the design function inputs. Verification constructs are also used to capture and monitor the output and possibly compare the output to a known golden data.

G

Generate Fixed Point step

A step in the AccelDSP flow when the user generates a fixed-point MATLAB model of the in-memory design.

Generate Simulink

A feature in the AccelDSP synthesis tool that generates a Simulink Block from the Verified RTL design. Generatting the Simulink Block must occur after the Verify RTL step in order to acquire the Startup Clock Cycles and Hardware Clock Cycles Per Design Function Call data on the block.

Generate System Generator

A feature in AccelDSP synthesis tool that generates a Xilinx System Generator Block from the Verified RTL design. Generating the System Generator Block must occur after the Verify RTL step in order to acquire the Startup Clock Cycles and Hardware Clock Cycles Per Design Function Call data on the block.

Generate RTL step

A step in the AccelDSP flow where high-level behavioral synthesis is performed on the analyzed design function. RTL (Register Transfer Level) code is written out in Verilog and/or VHDL format along with a Testbench in Verilog and/or VHDL format.

GUI

Graphical User Interface.

H

Hardware Clock Cycles Per Design Function Call

Refers to the number of hardware clock cycles it takes the design to produce a valid output after the startup period is over. For example, if the number of clock cycles per call is 1, the design produces a valid output every clock cycle after the startup period is over. If the number is 8, the design produces a valid output every 8 clock cycles, and so on. Assume that throughput is defined as “number of samples per second”. If there are 2 Hardware Clock Cycles Per Design Function Call and the design is operating at 10Mhz, then the design throughput is 5 Mega Samples Per Second.

HDL

HDL (Hardware Description Language) generally refers to a model written in the VHDL or Verilog language. In the AccelDSP environment, an untimed MATLAB model is transformed into a timed HDL model that is written at the RTL (register transfer level) of abstraction. In the RTL model, data is moved from one storage element to another in rhythm with a clock.

I

if balancing

Refers to the technique of always assigning a value to a variable inside a conditional statement. The following code segment contains an unbalanced if statement

```
b = 0;
if a > 5
b = a;
end
```

When the above code is executed, the variable b may or may not be assigned a value. This will produce less efficient hardware.

Now consider the following balanced if statement:

```
b = 0;
if a > 5
b = a;
else
b = b;
end
```

In this code, b is always assigned a value and the AccelDSP synthesis tool will produce the most efficient hardware.

In most cases, the tool automatically does "if" balancing. If it cannot, you may need to do the "if" balancing manually in the MATLAB source code to produce the most efficient hardware.

iterative processing

Refers to the algorithmic concept of capturing a pre-defined portion of the incoming data stream, executing the algorithm on that portion, outputting the result, then repeating the process.

implementation

A down-stream process where Xilinx ISE tools read a gate-level netlist, place and route the design, then generate a bit file that programs the functionality into a FPGA.

implement step

A step in the AccelDSP flow where a generated Tcl script calls the ISE tools to read the generated gate-level netlist, then place and route the design. One of the output files from this step is a gate-level HDL model of the implemented design. This model can be verified against the results from the fixed-point MATLAB model to ensure a bit-true outcome.

IP

Intellectual Property. A broad category of materials that are legally recognized as proprietary to an organization. In the electronics field, IP refers to specific portions of an FPGA or "building blocks" that can be proprietary and/or patented designs of a particular company. These IP "blocks" can then be sold to customers as commodity parts for new designs.

J

K

L

M

MATLAB®

A software program from the MathWorks, Inc that is used for high-performance numerical computing and visualization.

MATLAB® Console

A window in the AccelDSP GUI consisting of a command line prompt and a transcript. This window enables you to directly enter MATLAB commands.

N

O

overflow

Occurs when the magnitude of a number being assigned to a variable exceeds the number of bits allocated to the integer part of the fixed-point word. The excess in magnitude can happen with a negative number as well as a positive number.

P

partial unrolling

Normally, a loop in the design function may be partially unrolled by an even increment of the iteration-interval, if there are no data processing dependencies between iterations. For example, a loop with

an iteration interval of 16 may be unrolled 2, 4 or 8 times. Unrolling by 2 creates two parallel datapaths and takes eight cycles to complete. Unrolling by 4 creates four parallel datapaths and takes four cycles to complete. Unrolling by 8 creates eight parallel datapaths and takes two cycles to complete.

performance

Design performance refers to how fast data is moved through the design and is normally a function of the number of Hardware Clock Cycles Per Design Function Call and clock speed. Performance can be increased by increasing parallelism in the datapath; this also typically increases the size (area) of the hardware.

pre-analyze phase

A phase in the AccelDSP flow where the streaming loop and the design function call in the MATLAB script file are first identified. If these items cannot be automatically identified, the tool asks you to manually identify them through the GUI.

project

The collection of files and information that is relevant to the synthesis of the design function. The file collection includes the golden MATLAB design files, stimulus files, report files, source files, and generated output files (to name a few).

project directory

The physical directory (or folder) that contains the Project file. This directory receives the output files that are generated from the AccelDSP synthesis tool. Other project files may or may not reside in this directory. Non-project files may also reside in this directory, but are not viewable with the Project Explorer.

Project Explorer

A window in the AccelDSP GUI that allows you to view the content of the Project. Only files that are defined as part of the Project are viewable with the Project Explorer.

project file

Alternate name for an AccelDSP Control File.

project options

Information that you supply to the AccelDSP synthesis tool such as the pathname to the script file, the type of HDL language to generate as output, the HDL simulation tool to use for verification, and the RTL synthesis tool to use for optimization.

Q

quantization

The process of converting a very precise floating-point number to a less precise fixed-point number, usually through rounding or truncation.

quantization error

The loss of precision that occurs when a very precise floating-point number is quantized (converted) to a less precise fixed-point number.

quantized data

Floating-point numbers that have been converted to fixed-point numbers using the quantize() function.

R

RTL

Register Transfer Level. A level of modeling abstraction where the hardware design being modeled is divided into units of combinational logic and storage elements (registers). All operations (like multiply and sum) are scheduled to occur in specific clock cycles, but detailed timing delays below the cycle level are not modeled.

S

script M-file

A MATLAB M-file that is not a function definition. A script contains MATLAB language constructs that generate stimulus for the top-level design function as well as constructs to collect and monitor the results.

Simulink®

A software program from the MathWorks, Inc that is an extension to MATLAB. Simulink allows engineers to rapidly and accurately build computer models of dynamic systems using block diagram notation.

Startup Clock Cycles

Refers to the number of hardware clock cycles it takes to move data from the design input to the design output on startup. This number will always be greater than or equal to the number of Hardware Clock Cycles Per Design Function Call. The Startup Clock Cycles are greater than the Hardware Clock Cycles Per Design Function Call when the AccelDSP synthesis tool has automatically pipelined all or part of the design, causing a need for extra startup clock cycles to initialize the pipeline(s).

streaming loop

The outer-most for loop or while loop in a MATLAB script file that represents the iterative nature of the algorithmic design. A streaming loop manages the input data stream by partitioning the data into manageable groups for processing.

Synthesize RTL step

A step in the AccelDSP flow where a pre-specified RTL synthesis tool is called to synthesize the generated RTL design, map the logic to hardware-specific primitive elements, and then generate a gate-level netlist.

System Generator®

A software program from Xilinx® that allows engineers to rapidly and accurately build computer models of dynamic systems using block diagram notation.

T

Tcl

Tcl stands for "tool command language". Tcl is a simple scripting language for controlling and extending application programs. The command-line interface to the AccelDSP synthesis tool is Tcl.

Tcl Console

A window in the AccelDSP GUI consisting of a command line prompt and a transcript. This window enables you to directly enter Tcl commands that direct the activity of the tool.

Testbench

The HDL module that is generated (not synthesized) by the AccelDSP synthesis tool when the MATLAB design function is synthesized to an RTL model. When you execute the Verify RTL step in the AccelDSP flow, an HDL simulator runs a simulation on the Testbench which, in turn, applies stimulus to the inputs of the RTL model and compares the result to a known golden data.

U

underflow

Occurs when a very small fractional number gets rounded to zero. For example, if the number 0.00001365 gets assigned to a variable with only 4 bits allocated to the fractional part of the fixed-point word, then the number is rounded to zero. Underflows are usually more common and less serious than overflows.

uniquify

A term used in the AccelDSP environment that refers to replacing a single variable with two or more unique variables when the single variable is used multiple times. Consider the following code segment:

```
if a > 2
b = b + indata1;
end
a = b + 1;
```

Now consider the following code segment where b has be "uniquified":

```
if a > 2
b2 = b + indata1;
else
b2 = b;
end
a = b2 + 1;
```

The second code segment produces more efficient hardware. In most cases, the AccelDSP synthesis tool automatically "uniquifies" non-persistent variables. If it cannot, you may need to apply the techniques manually in the MATLAB source code to produce the most efficient hardware.

unrolling

Unrollig a loop. A loop construct is synthesized into a similar loop structure in hardware where data is applied to a single data path a specific number of times (called the iteration interval). A loop with an iteration interval of sixteen, for example, results in hardware where data is applied to a single data path sixteen times before processing ends. If the loop construct is completely unrolled, then the AccelDSP synthesis tool builds a different hardware structure where sixteen data samples are applied simultaneously to sixteen identical parallel data paths. The number of Hardware Clock Cycles Per Design Function Call is decreased to one at a cost of increasing the hardware area by sixteen times. If a full unroll consumes too much hardware, then a partial unroll may be a suitable trade off between area and performance. For a partial unroll, the unroll factor you specify must be an integer that is an integral divisor of the loop extent and should not exceed the loop extent.

V

verification constructs

MATLAB language constructs in the script file that are used to perform verification on the design function. Typically, these constructs are used to generate and apply stimulus to the design

function input(s) and compare the design function output(s) to a known golden data.

Verify Fixed Point step

A step in the AccelDSP flow where the user verifies the automatically-generated fixed-point MATLAB model with a MATLAB simulation run.

Verify Floating Point step

A step in the AccelDSP flow where the user may choose to verify the floating-point MATLAB model with a MATLAB simulation run. This step may be skipped if the floating-point model has already been verified outside of the AccelDSP synthesis tool.

Verify Gate Level step

The last step in the AccelDSP flow where the implemented gate-level HDL model is verified using the AccelDSP-generated Testbench and a pre-specified HDL simulator. The simulation results are compared to the original results from the generated fixed-point MATLAB design to ensure a bit-true outcome.

Verify RTL step

A step in the AccelDSP flow when the generated RTL HDL design is verified with the generated HDL Testbench using a pre-specified simulation tool.

Verilog

A hardware description language. An industry-accepted standard language used by electronic designers to describe and simulate their hardware designs prior to implementation. An alternative language to VHDL.

VHDL

VHSIC Hardware Description Language. VHSIC is an abbreviation for Very High Speed Integrated Circuit. An IEEE-standard hardware description language originally developed by the U.S. Department of Defense as a common means of documenting electronic systems. Specified in the IEEE 1076 standard and used by electronic designers to describe and simulate their hardware designs prior to implementation. An alternative language to Verilog.

W

X

Y

Z