ISE Simulator (ISim) In-Depth Tutorial

UG682 (v 13.2) July 6, 2011

XILINX®

Table of Contents

Chapter 4: Using the ISim Graphical User Interface

About This Tutorial

About the ISE Simulator (ISim) In-Depth Tutorial

The ISim In-Depth Tutorial provides Xilinx PLD designers with a detailed introduction of the ISE Simulator (ISim) software. After you have completed the tutorial, you will have a thorough understanding of how to analyze and debug your design via HDL simulation using ISim.

This tutorial is designed for running the ISim software on a Windows environment. Some modifications might be required to run certain steps successfully in other operating systems.

Tutorial Contents

This tutorial covers the following topics:

Chapter 1, Overview of the ISE Simulator (ISim), introduces the ISim software environment, including the ISim compilers, linker, simulation executable and Graphical User Interface (GUI).

Chapter 2, Running ISim from ISE Project Navigator, explains how to launch a functional simulation through the ISE Project Navigator software.

Chapter 3, Running ISim Standalone, guides you through a typical procedure for launching a functional simulation using the ISim compiler, linker and simulation executable outside of the ISE Project Navigator environment.

Chapter 4, Using the ISim Graphical User Interface, introduces you to the ISim GUI by examining, debugging, and verifying a functional simulation.

Tutorial Flows

This tutorial presents two flows in which ISim can be used for performing a functional (behavioral) simulation.

Using ISim from ISE Project Navigator

In this flow, you will launch ISim via one of the simulation processes available in ISE Project Navigator. This flow works best when an ISE Project Navigator project is created in order to implement the design in a Xilinx® FPGA or CPLD. This flow is intended for designs that contain sources that are not HDL, such as schematics and cores, which require Project Navigator to properly convert these sources to HDL source files which ISim can compile.

Follow these chapters if you are interested in this flow:

Chapter 1, Overview of the ISE Simulator (ISim)

Chapter 2, Running ISim from ISE Project Navigator

Chapter 4, Using the ISim Graphical User Interface

Using ISim Standalone

In this mode, you will simulate your design by creating your own ISim project files and running the HDL linker and simulation executable in a command line or batch file mode. This flow is intended for users who do not need to use Project Navigator for HDL design management.

The following chapters will help you understand this flow:

Chapter 1, Overview of the ISE Simulator (ISim)

Chapter 3, Running ISim Standalone

Chapter 4, Using the ISim Graphical User Interface

Additional Resources

To find more detailed information and discussions on ISE Simulator (ISim) topics covered in this tutorial, refer to the following documents:

ISim User Guide (UG660), accessible from the Software Manuals page on the Xilinx website:

- http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/plugin_ism.pdf

Note: *ISim Help* is available from the ISim software by pressing **F1** or from the **Help** menu.

Software Manuals - To find additional documentation, see the Xilinx website at:

- http://www.xilinx.com/support/documentation/index.htm

Tutorials Page - To find the *ISim In-Depth Tutorial* design files and other ISE Design Suite tutorials, see the Xilinx website at:

- http://www.xilinx.com/support/documentation/dt_ise13-1_tutorials.htm

Answer Records - 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/

Xilinx Forums - To discuss topics of interest with other Xilinx users, see the Xilinx User Community Forum at:

- http://forums.xilinx.com/xlnx/

XILINX®

Chapter 1

Overview of the ISE Simulator (ISim)

Overview of ISim

The Xilinx® ISE Simulator (ISim) is a Hardware Description Language (HDL) simulator that enables you to perform functional (behavioral) and timing simulations for VHDL, Verilog and mixed-language designs.

This ISE Simulator environment is comprised of the following key elements:

- vhpcomp (VHDL parser)
- vlogcomp (Verilog parser)
- fuse (HDL elaborator and linker)
- Simulation Executable
- isimgui (ISim Graphical User Interface)

Note: More information about ISim is available in the *ISim User Guide (UG660)*.

vhpcomp, vlogcomp

vhpcomp and vlogcomp parse and compile VHDL and Verilog source files respectively. The parsed dump generated by the parsers is used by fuse to generate object code and link object code with simulation kernel library to create a simulation executable.

fuse

The fuse command is the Hardware Description Language (HDL) elaborator and linker used by ISim. fuse effects static elaboration on the design given the top design units and then compiles the design units to object code. The design unit object files are then linked together to create a simulation executable.

The fuse command can automatically invoke vlogcomp and vhpcomp for each VHDL or Verilog source code in a project file (.prj), allowing you to compile sources “on-the-fly”.

Simulation Executable

The Simulation Executable is generated by the fuse command. To run the simulation of a design in ISim, the generated simulation executable needs to be invoked. When ISim is run inside the ISE Project Navigator interface, ISE takes care of invoking the generated simulation executable. A command-line user needs to explicitly invoke the generated simulation executable to effect simulation. The simulation executable effects event-driven simulation and has rich support for driving and probing simulation using Tcl.

Note: The ISE Simulation Executable has a .exe extension in both Linux and Windows. The default executable naming format is `x.exe`.

isimgui.exe

isimgui.exe (isimgui on Linux) is the ISim Graphical User Interface. It contains the wave window, toolbars, panels, and the status bar. In the main window, you can view the simulation-visible parts of the design, add and view signals in the wave window, utilize ISim commands to run simulation, examine the design, and debug as necessary.

XILINX®

Chapter 2

Running ISim from ISE Project Navigator

Overview of ISE Simulator (ISim)-Project Navigator Integrated Flow

The Xilinx® ISE Design Suite provides an integrated flow with the ISE Simulator (ISim) that allows simulations to be launched directly from the Project Navigator (ISE). All simulation commands that prepare the ISim simulation are generated by ISE Project Navigator and automatically run in the background when simulating a design using this flow.

Getting Started

Software Requirements

To use this tutorial, you must install one of the following software:

- ISE WebPACK™ 13.1
- One of the ISE Design Suite 13.1 Editions (Logic, DSP, Embedded, System)

For more information about installing Xilinx software, see the *Xilinx ISE Design Suite: Installation, Licensing, and Release Notes*.

Installing the Tutorial Design Files

Design files for this tutorial are available from the Tutorials page on the Xilinx Website.

- Download the ug682.zip Design File associated with the ISE Simulator In-Depth Tutorial from the website.
- Unzip the design files into an easily accessible directory with full read and write permissions.

The contents of the tutorial design files are as follows:

Table 2-1: **Tutorial Design Files**

Folder	Description
sources	Contains all the HDL files necessary for a functional simulation of the design.
scripts	Contains incomplete script files to run the simulation. These script files will be completed as you go through the tutorial.
completed	Contains completed script, simulation and wave configuration files, as well as a completed ISE 13 project of the tutorial design, for comparison purposes.

Design Description

The tutorial design is a simple demonstration of the Dynamic Reconfiguration feature of the Virtex®-5 Digital Clock Manager (DCM).

Using the Virtex-5 DCM, the design generates an output clock using the following relationship:

Output Clock = Input Clock * (Multiplier / Divider)

Using the Dynamic Reconfiguration Ports (DRP) in the DCM, the design allows you to re-define the Multiplier and Divider parameters to generate different output frequencies.

Functional Blocks

The tutorial design consists of the following functional blocks.

drp_dcm (drp_dcm.vhd)

Virtex-5 DCM macro with internal feedback, frequency controlled output, duty-cycle correction, and Dynamic Reconfiguration ability.

The CLKFX_OUT output provides a clock that is defined by the following relationship:

CLKFX_OUT = CLKIN_IN * (Multiplier/Divider)

For example, using a 100 MHz input clock, setting the Multiplier factor to 6, and Divider factor to 5, produces a 120 MHz CLKFX_OUT output clock.

Using the DRP ports of the DCM, the Multiplier (M) and Divider (D) parameters can be dynamically redefined to produce different CLKFX_OUT frequencies. For the purposes of this tutorial, it suffices to show how the Multiply and Divide parameters are provided to the DCM via the 16-bit wide DI_IN port:

DI_IN[15:8] = M – 1

DI_IN[7:0] = D – 1

For example, for an M/D factor of 6 / 5, DI_IN = 0504h.

drp_stmach (drp_stmach.vhd)

This module describes a Dynamic Reconfiguration Controller. The DRP controller asserts and monitors the DCM DRP signals in order to perform a dynamic reconfiguration cycle.

A dynamic reconfiguration cycle is started by asserting the drp_start signal. Following this step, the DRP Controller asserts the appropriate DCM DRP pins in order to complete a full Dynamic Reconfiguration cycle.

Signal drp_done indicates a successful completion of a dynamic reconfiguration cycle.

drp_demo (drp_demo.vhd)

This is the top module of the tutorial design which connects the DCM macro and the DRP controller modules to the external I/O ports.

drp_demo_tb (drp_demo_tb.vhd)

Self-checking HDL test bench. Refer to Design Self-Checking Test Bench.

Design Self-Checking Test Bench

To test the functionality of this design, a self-checking test bench has been provided. (Refer to source file `drp_demo_tb.vhd` in the `sources/` folder.) A self-checking test bench contains a validation routine or function that compares sampled values from the simulation against expected results. The self-checking test bench provided for this design performs the following functions.

- Generates a 100 MHz input clock for the design system clock (clk_in).
- Performs four different tests in order to dynamically change the output frequency of the design. In each test, a DRP cycle is started (using the drp_start signal) to set the output clock to a different frequency. The following table shows the desired output frequency and Multiplier/Divider parameters used for each test.

Table 2-1: **Desired Output Frequency and Multiplier/Divider Parameters**

Test	Freq. (MHz)	Period (ps)	Multiplier (M)	Divider (D)
1	75	13,332	3	4
2	120	8,332	6	5
3	250	4000	5	2
4	400	2,500	4	1

- In each test, the test bench compares the expected clock period and the clock period measured during simulation. Based on the comparison results, messages are written to the simulator indicating success or failure.
- Upon completion of the simulation, a summary report provides a list of tests, both passed or failed.

For more details on the functionality of this design, refer to the in-line comments included in the sources of the design.

Tip: To create an HDL test bench in Project Navigator, select **Project > Create New Sources** and select **VHDL Testbench** or **Verilog Text Fixture**.

Note: To learn more about designing HDL test benches, check out Application Note XAPP199 "Writing Effective Testbenches."

Simulating the Design

The ISE-ISim integrated flow enables you to quickly perform behavioral and timing simulations of your design in the ISE Project Navigator software.

In this tutorial flow, in the ISE Project Navigator you will create an ISE project for the tutorial design first. You will then set some behavioral simulation properties and launch the ISim simulator to perform a behavioral simulation of the design.

Creating a Project in ISE Project Navigator

We will use the New Project Wizard in ISE Project Navigator to quickly create an ISE project for the tutorial design.

Note: Read Installing the Tutorial Design Files to obtain the files required for this design.

Launching Project Navigator and Using New Project Wizard

Follow these steps to launch Project Navigator software and create an ISE project.

1. Launch ISE Project Navigator from the Xilinx ISE Design Suite.
2. Select **File > New Project** to launch the New Project Wizard.
3. In the Create New Project window, provide a name (for example, isim_tutorial) and an appropriate location for the project.
4. Click **Next** to continue.
5. In the Project Settings window, you select the device and project properties. Change the settings to match the settings shown in Figure 2-1.

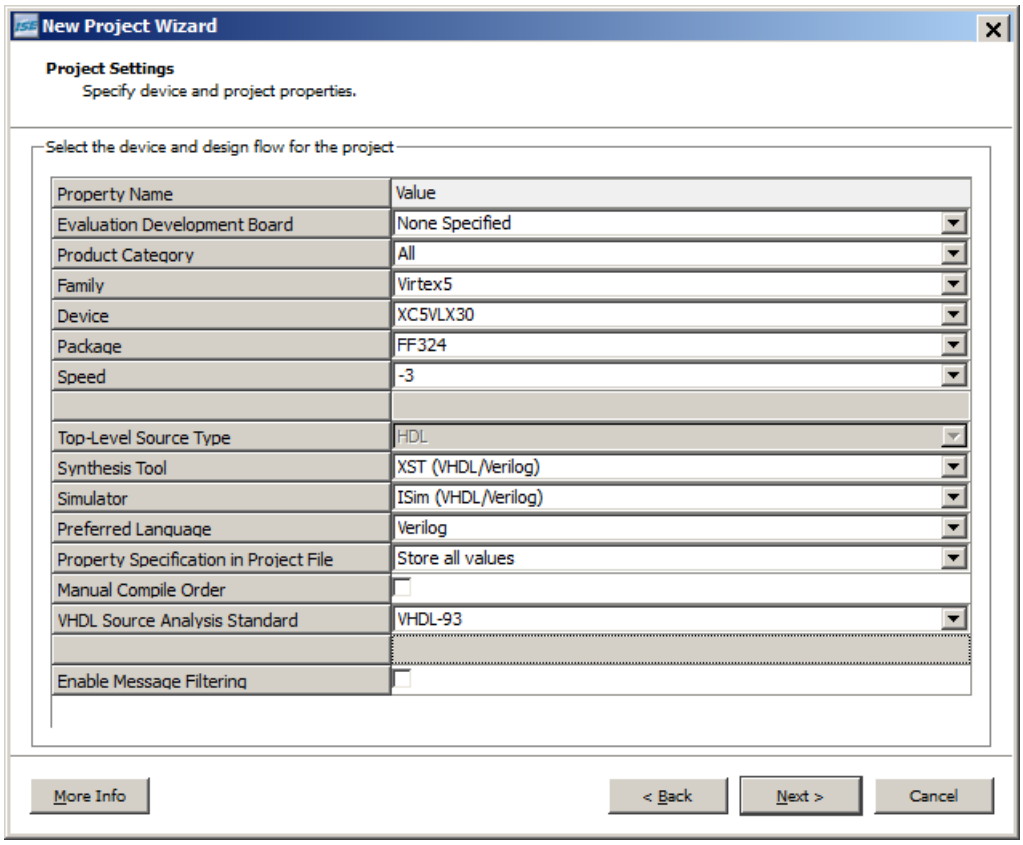


Figure 2-1: **New Project Wizard: Project Settings**

6. Click **Next** to continue.
7. Review the Project Summary page and click **Finish** to continue.

Adding Tutorial Source Files to the Project

1. Select **Project > Add Source** to add a source file. For this tutorial, navigate to the location where you saved your source files in Installing the Tutorial Design Files.
2. In the **sources** subfolder, select all of the files and open them.
3. In the next window, make sure that the association and libraries have been properly specified for the tutorial sources. Compare your settings with those in Figure 2-2.

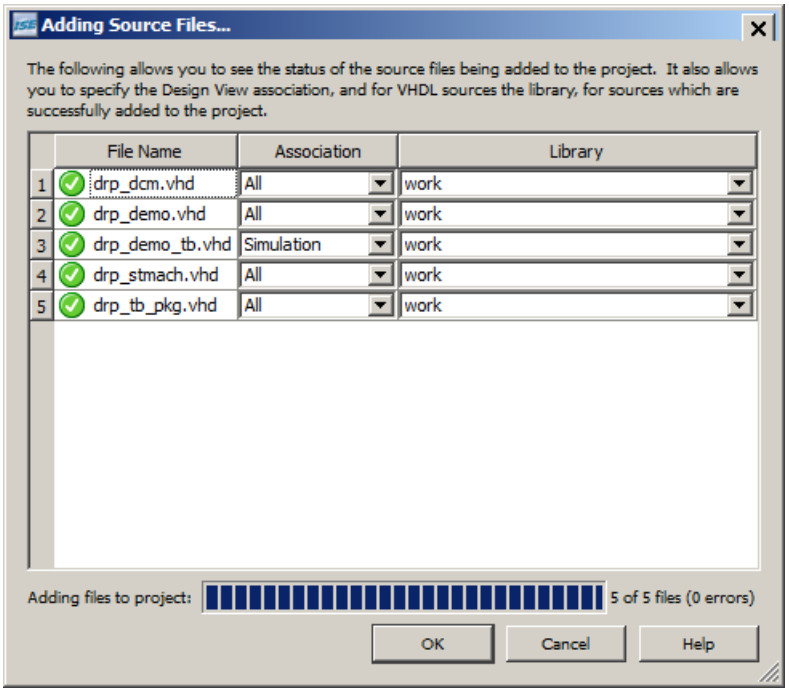


Figure 2-2: **Status of Source Files and Associations**

4. Click **OK**.

The source files are added to the project. Your Project Navigator window should now look like Figure 2-3.

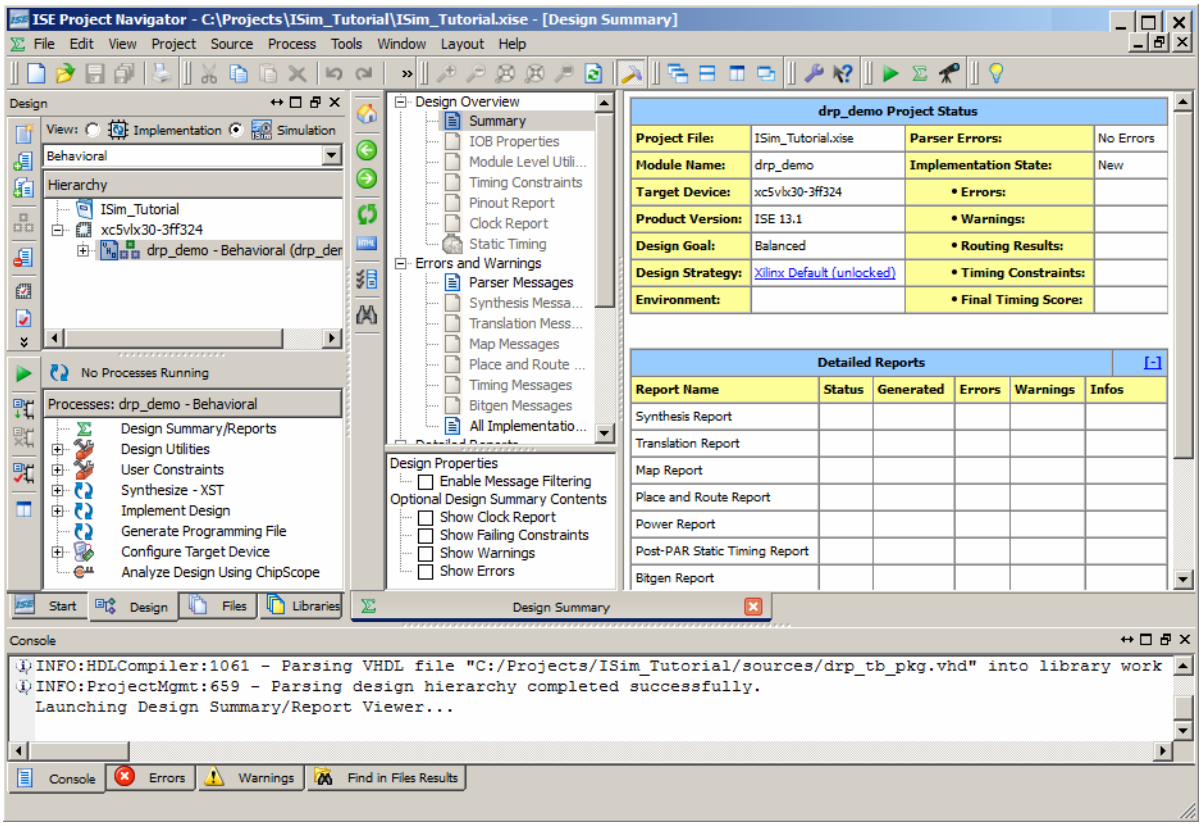


Figure 2-3: **ISE Project Navigator Design Summary**

Creating VHDL Library

Next, you need to create a user VHDL library for a VHDL package (`drp_tb_pkg.vhd`) that will be used by the test bench of this design. The VHDL package contains VHDL functions used by the test bench to perform verification routines. Once the VHDL library is created, you will move the VHDL package file from the work library to the newly-created VHDL library.

Follow these steps to create a VHDL library.

1. In Project Navigator, select **Project > New Source**. The New Source Wizard opens.
2. Select **VHDL Library** as the source type.
3. Type **`drp_tb_lib`** for the VHDL library name. (Refer to Figure 2-4).

   Note: Leave the **Add to project** check box selected.
4. Click **Next** to continue.

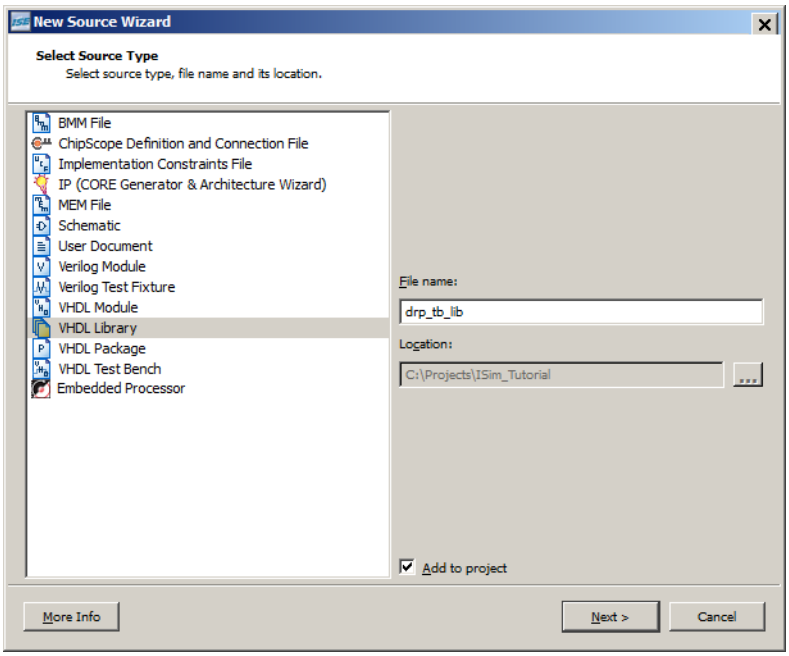


Figure 2-4: **Select Source Type**

5. In the Summary page, click **Finish** to complete the New Source Wizard.

Moving VHDL files to a Library

Follow these steps to move the VHDL package file to the `drp_tb_lib` VHDL library.

1. In the Sources Panel, click the **Libraries** tab to switch to the Libraries Panel.
2. Expand the work library. (Refer to Figure 2-5.)

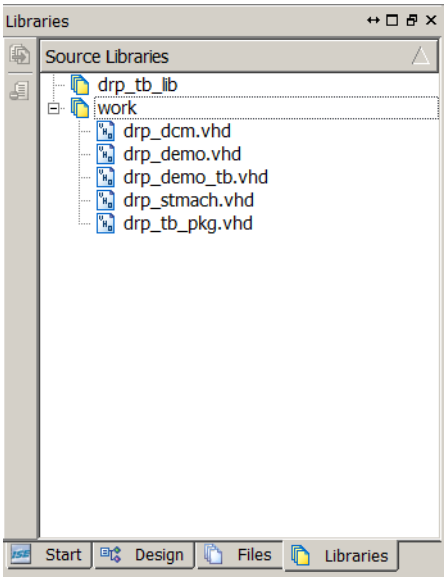


Figure 2-5: **Libraries Panel**

3. Right-click the `drp_tb_pkg.vhd` file, and select **Move to Library**.

4. In the Move to Library dialog box, select `drp_tb_lib` as the library into which you will move drp_tb_pkg.vhd, the VHDL package file.
5. Click **OK**.

You can now observe that a new VHDL library, `drp_tb_lib`, contains the VHDL package file `drp_tb_pkg.vhd`. (Refer to Figure 2-6.)

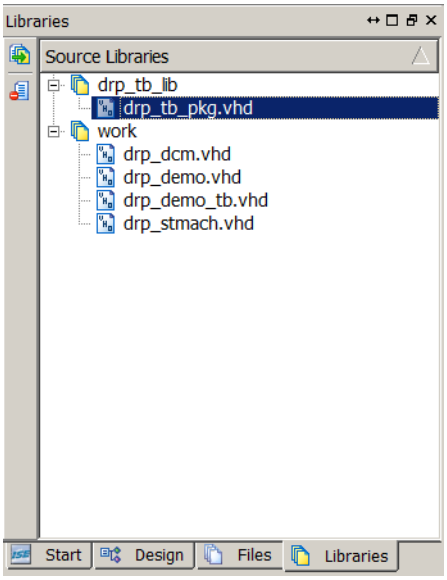


Figure 2-6: **Source Libraries**

Launching a Behavioral Simulation

Now that the ISE project has been created for the tutorial design, we can proceed to set up and launch a behavioral simulation using ISim.

Setting Behavioral Simulation Properties

Follow these steps to set behavioral simulation properties in ISE:

1. In the Design Panel, select the **Simulation** radio button.

   The Simulation type drop-down list appears.
2. Select **Behavioral** Simulation from the drop-down list.
3. Select the tutorial design test bench file, `drp_demo_tb`.

   You should now see the simulation processes available for the design in the Processes pane. (Refer to Figure 2-7)

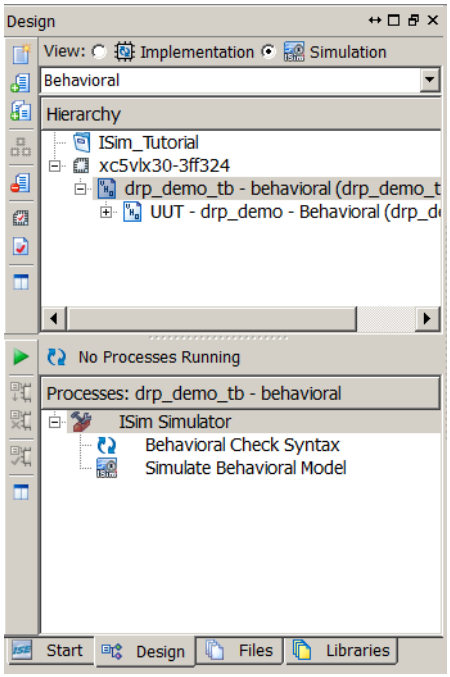


Figure 2-7: **Behavioral Simulation Processes**

4. Right-click **Simulate Behavioral Model** under the ISim Simulator process and select **Process Properties**. The ISim Properties dialog box opens (Refer to Figure 2-8).

   In this window you can set different simulation properties, such as simulation runtime, waveform database file location, and even a user-defined simulation command file to launch the simulation. For the purposes of this tutorial, we will disable the feature that runs the simulation for a specified amount of time.

5. Uncheck the property **Run for Specified Time** and click **OK**.

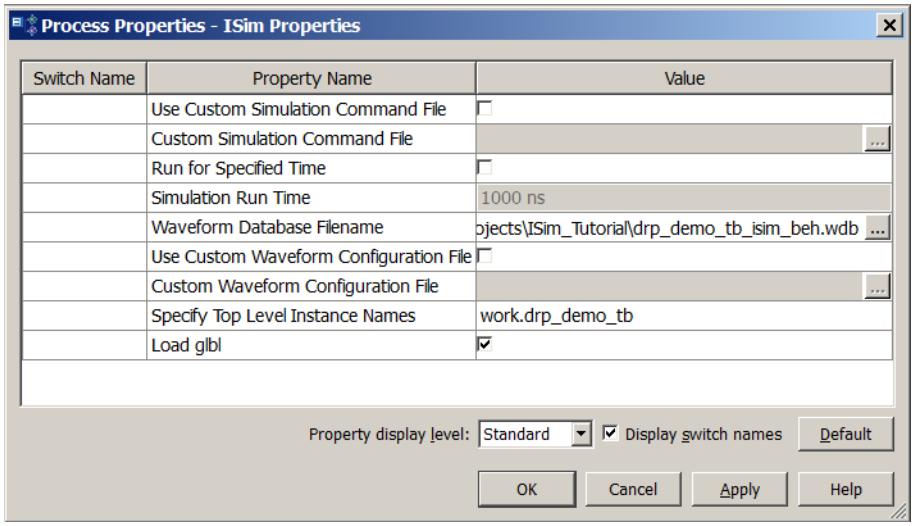


Figure 2-8: **ISim Properties Dialog Box**

Launching Behavioral Simulation

You are now ready to launch the ISE Simulator to perform a behavioral simulation of the tutorial design. To launch the simulator:

In the Processes panel, double-click **Simulate Behavioral Model**.

The ISim interface (Figure 2-9) appears after the design successfully parses and compiles.

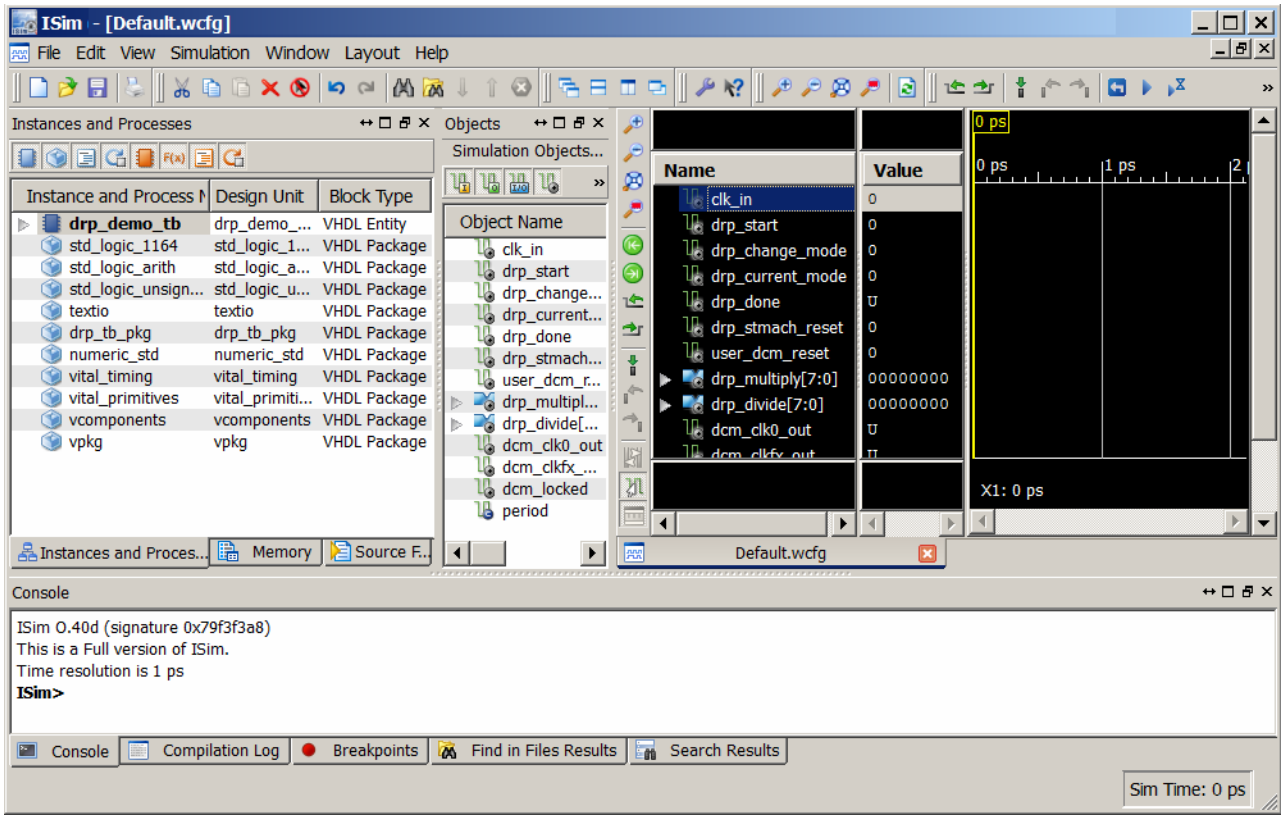


Figure 2-9: **ISim Graphical User Interface**

What's Next?

Continue on to Chapter 4, Using the ISim Graphical User Interface to learn more about the ISim GUI features and tools for analyzing and debugging HDL designs.

XILINX®

Chapter 3

Running ISim Standalone

Overview of ISE Simulator (ISim) Standalone Flow

You can use the ISim standalone flow to simulate your design without setting up a project in ISE® Project Navigator. In this flow, you:

- Prepare the simulation project by manually creating an ISim project file in order to create a simulation executable using **`fuse`**.
- Start the ISim Graphical User Interface (GUI) by running the simulation executable generated by **`fuse`**.

Getting Started

Software Requirements

To use this tutorial, you must install one of the following software:

- ISE WebPACK™ 13.1
- One of the ISE Design Suite 13.1 Editions (Logic, DSP, Embedded, System)

For more information about installing Xilinx software, see the *Xilinx ISE Design Suite: Installation, Licensing, and Release Notes*.

Installing the Tutorial Design Files

Design files for this tutorial are available from the Tutorials page on the Xilinx Website.

- Download the tutorial design zip file from the website.
- Unzip the design files into an easily accessible directory with full read and write permissions.

The contents of the tutorial design files are as follows:

Table 3-1: **Tutorial Design Files**

Folder	Description
sources	Contains all the HDL files necessary for a functional simulation of the design.
scripts	Contains incomplete script files to run the simulation. These script files will be completed as you go through the tutorial.
completed	Contains completed script, simulation and wave configuration files, as well as a completed ISE 13 project of the tutorial design, for comparison purposes.

Design Description

The tutorial design is a simple demonstration of the Dynamic Reconfiguration feature of the Virtex®-5 Digital Clock Manager (DCM).

Using the Virtex-5 DCM, the design generates an output clock using the following relationship:

Output Clock = Input Clock * (Multiplier / Divider)

Using the Dynamic Reconfiguration Ports (DRP) in the DCM, the design allows you to re-define the Multiplier and Divider parameters to generate different output frequencies.

Functional Blocks

The tutorial design consists of the following functional blocks.

drp_dcm (drp_dcm.vhd)

Virtex-5 DCM macro with internal feedback, frequency controlled output, duty-cycle correction, and Dynamic Reconfiguration ability.

The CLKFX_OUT output provides a clock that is defined by the following relationship:

CLKFX_OUT = CLKIN_IN * (Multiplier/Divider)

For example, using a 100 MHz input clock, setting the Multiplier factor to 6, and Divider factor to 5, produces a 120 MHz CLKFX_OUT output clock.

Using the DRP ports of the DCM, the Multiplier (M) and Divider (D) parameters can be dynamically redefined to produce different CLKFX_OUT frequencies. For the purposes of this tutorial, it suffices to show how the Multiply and Divide parameters are provided to the DCM via the 16-bit wide DI_IN port:

DI_IN[15:8] = M – 1

DI_IN[7:0] = D – 1

For example, for an M/D factor of 6 / 5, DI_IN = 0504h.

drp_stmach (drp_stmach.vhd)

This module describes a Dynamic Reconfiguration Controller. The DRP controller asserts and monitors the DCM DRP signals in order to perform a dynamic reconfiguration cycle.

A dynamic reconfiguration cycle is started by asserting the drp_start signal. Following this step, the DRP Controller asserts the appropriate DCM DRP pins in order to complete a full Dynamic Reconfiguration cycle.

Signal drp_done indicates a successful completion of a dynamic reconfiguration cycle.

drp_demo (drp_demo.vhd)

This is the top module of the tutorial design which connects the DCM macro and the DRP controller modules to the external I/O ports.

drp_demo_tb (drp_demo_tb.vhd)

Self-checking HDL test bench. Refer to Design Self-Checking Test Bench for more information.

Design Self-Checking Test Bench

To test the functionality of this design, a self-checking test bench has been provided. (Refer to source file `drp_demo_tb.vhd` in the `sources/` folder.) A self-checking test bench contains a validation routine or function that compares sampled values from the simulation against expected results. The self-checking test bench provided for this design performs the following functions.

- Generates a 100 MHz input clock for the design system clock (clk_in).
- Performs four different tests in order to dynamically change the output frequency of the design. In each test, a DRP cycle is started (using the drp_start signal) to set the output clock to a different frequency. The following table shows the desired output frequency and Multiplier/Divider parameters used for each test.

Table 2-1: **Desired Output Frequency and Multiplier/Divider Parameters Used For Each Test**

Test	Freq. (MHz)	Period (ps)	Multiplier (M)	Divider (D)
1	75	13,332	3	4
2	120	8,332	6	5
3	250	4000	5	2
4	400	2,500	4	1

- In each test, the test bench compares the expected clock period and the clock period measured during simulation. Based on the comparison results, messages are written to the simulator indicating success or failure.
- Upon completion of the simulation, a summary report provides a list of tests, both passed or failed.

For more details on the functionality of this design, refer to the in-line comments included in the sources of the design.

Preparing the Simulation

The ISim standalone flow enables you to simulate your design without setting up a project in ISE Project Navigator. In this flow, you will manually create an ISim project file which **`fuse`** will use to create a simulation executable. Following completion of this step, the ISim Graphical User Interface (GUI) can be launched by running the simulation executable.

Creating an ISim Project File

The typical syntax for an ISim project file is as follows:

```
verilog|vhdl <library_name> {<file_name_1>.v|.vhd}
```

where:

- *verilog* | *vhdl* indicates that the source is a Verilog or VHDL file. Include either verilog or vhdl.
- <*library_name*> indicates the library that a particular source on the given line should be compiled. work is the default library.
- <*file_name*> is the source file or files associated with the library.

Note: While one or more Verilog source files can be specified on a given line, only one VHDL source can be specified on a given line.

Complete the following steps to build an ISim project file for the tutorial design:

1. Browse to the `scripts` folder from the downloaded files.
2. Open the `simulate_isim.prj` project file with a text editor.

   The project file is incomplete.
3. List the missing sources using the syntax guidelines shown above.

   Missing sources:

   - `drp_dcm.vhd`: VHDL source file. It should be compiled to `work` library.
   - `drp_tb_pkg.vhd`: VHDL package file. It should be compiled to `drp_tb_lib` library.

   Note: You do not need to list the sources based on their order of dependency. **`fuse`** automatically resolves the order of dependencies and processes the files in the appropriate order.

   For comparison purposes, you can browse to the `completed` folder of the tutorial files for a completed version of the project file.
4. Save and close the file.

Building the Simulation Executable

In this simulation step, **`fuse`** will use the project file created in the previous section to parse, compile and link all the sources for the design. Following completion of these steps, a simulation executable will be created which will enable you to run the simulation in the ISim GUI.

Using fuse

The typical fuse syntax is as follows:

```
fuse –incremental –prj <project file> -o <simulation executable>
<library.top_unit>
```

where:

- `-incremental`: requests fuse to compile only the files that have changed since the last compile
- `-prj`: specifies an ISim project file to use for input
- `-o`: specifies the name of the simulation executable output file
- `<library.top_unit>`: specifies the top design unit

Complete the following steps to parse, compile and elaborate the tutorial design using fuse:

1. Browse to the `scripts` folder from the downloaded files.
2. Open the `fuse_batch.bat` batch file using a text editor.
3. This fuse command is incomplete. Using the syntax information provided above, edit the command line so it includes the following options:
   a. Use incremental compilation.
   b. Use `simulate_isim.prj` as the project file.
   c. Use `simulate_isim.exe` as the simulation executable.
   d. Use `work.drp_demo_tb` as the top design unit for simulation.
4. Save and close the batch file.
5. Using the ISE Design Suite Command prompt, navigate to and run the `fuse_batch.bat` file to run fuse.

   Note: To open the ISE Design Suite Command prompt, go to **Start > Programs > Xilinx ISE Design Suite > Accessories** and click the **ISE Design Suite Command Prompt** item.

Once fuse completes compiling source code, elaborating design units, and linking the object code, a simulation executable (`simulate_isim.exe`) should be present in the scripts folder.

For comparison purposes, you can browse to the `completed` folder for a completed version of the fuse batch file.

Simulating the Design

In this simulation step you will launch the ISim GUI by running the simulation executable which was generated by the **`fuse`** tool in the previous section, Building the Simulation Executable. After this step is complete, you will be able to use the ISim GUI to explore the design in more detail.

Running the Simulation Executable

The typical syntax used when launching the simulation executable is as follows:

```
Simulation_executable –gui –view <wave_configuration_file> -wdb
<waveform_database_file>
```

where:

- `-gui`: launches ISim in GUI mode.
- `-view`: opens the specified waveform file in the ISim GUI.
- `-wdb`: specifies the file name of the simulation database output file.

Complete the following steps to launch the simulation:

1. Browse to the `scripts` folder from the downloaded files.
2. Open the `simulate_isim.bat` batch file using a text editor. The batch file is intentionally blank.
3. Using the syntax information provided above, edit the batch file so it includes the following settings:
   a. Simulation Executable name: `simulate_isim.exe`.
   b. Launch in GUI mode.
   c. Set simulation database output name to `simulate_isim.wdb`.

   Note: A wave configuration file is not provided in the tutorial files. This file will be created
4. Save and close the file.
5. Using the ISE Design Suite Command prompt, navigate to and run the `simulate_isim.bat` file to run the simulator.

The ISim GUI opens and loads the design. The simulator time remains at 0 ns until you specify a run time.

For comparison purposes, you can browse to the `completed` folder for a completed version of the `simulate_isim.bat` batch file.

What's Next?

Continue on to Chapter 4, Using the ISim Graphical User Interface to learn more about the ISim GUI features, and tools for analyzing and debugging HDL designs.

Using the ISim Graphical User Interface

Overview of the ISE Simulator (ISim) Graphical User Interface

The ISim Graphical User Interface (GUI) contains the wave window, toolbars, panels, and the status bar. In the main window, you can view the simulation-visible parts of the design, add and view signals in the wave window, utilize ISim commands to run simulation, examine the design, and debug as necessary.

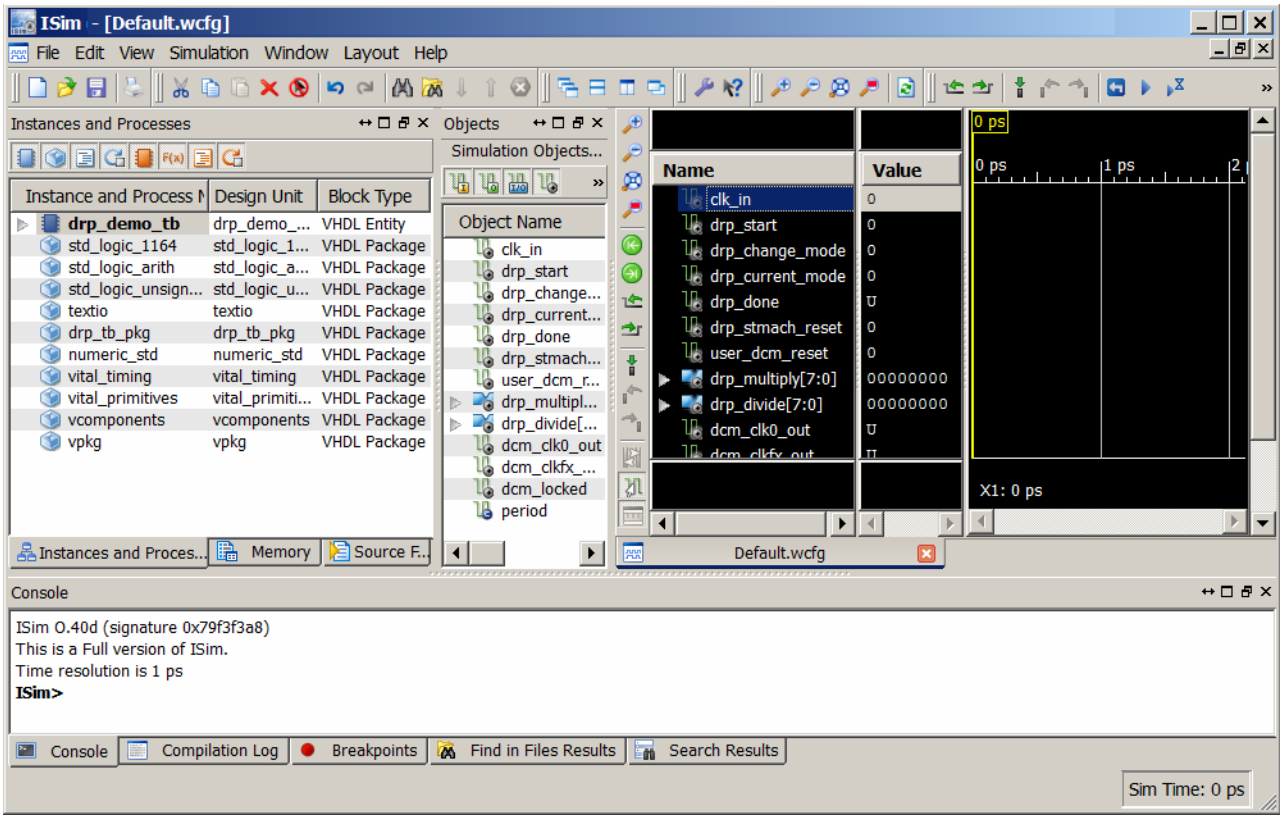


Figure 4-1: **ISim Graphical User Interface**

Exploring the User Interface

Main Toolbar

The toolbars available in the ISim main window consists of many functionally different toolbars. Each of these toolbars offers access to frequently used commands that can also be accessed from the menus.

The main window toolbar icons are located near the top of the user interface.

Figure 4-2: **Main Toolbar**

Instances and Processes Panel

The Instances and Processes panel displays the block (instance and process) hierarchy associated with the wave configuration open in the Wave window. Instantiated and elaborated entities/modules are displayed in a tree structure, with entity components being ports, signals and other entities/modules.

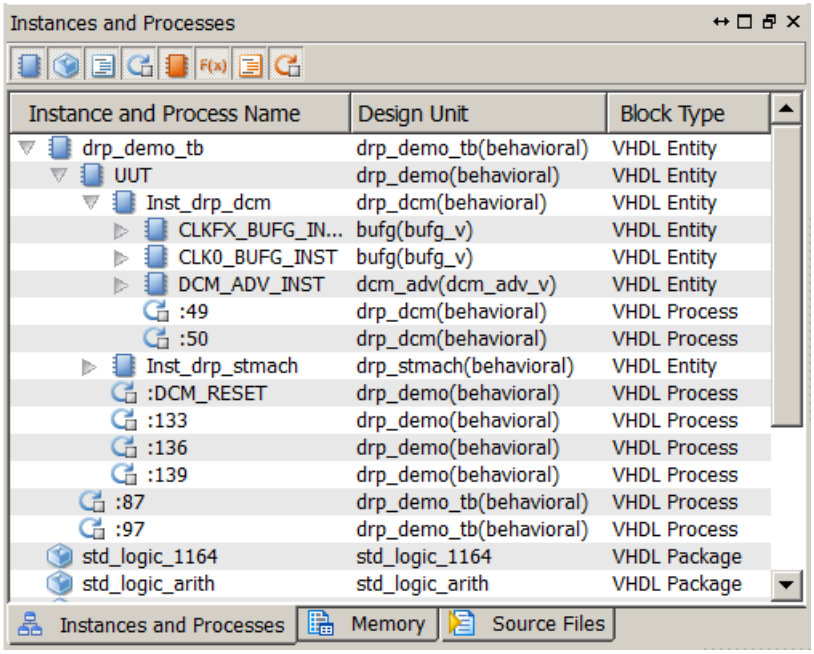


Figure 4-3: **Instances and Processes Panel**

Source Files Panel

The Source Files panel displays the list of all the files associated with the design. The list of files is provided by the **`fuse`** command during design parsing and elaboration, which is run in the background for GUI users. The HDL source files are available for quick access to the read-only source code.

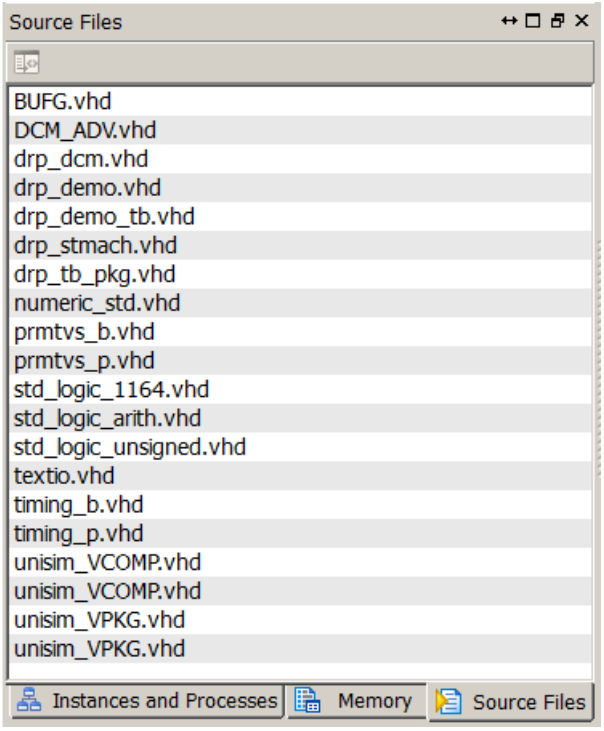


Figure 4-4: **Sources Files Panel**

Objects Panel

The Objects panel displays all ports and signals associated with the selected instances and processes in the Instances and Processes panel.

At the top of the panel, the Simulation Objects displays which instance/process is selected in the Instances and Processes panel and the corresponding objects and their values are listed in the Objects panel.

The table columns are defined as follows:

- **Object Name** - Displays the name of the signal, accompanied by the symbol which represents the type of object it is.
- **Value** - The value of the signals at the current simulation time or at the main cursor, as determined by the Sync Time toolbar button.
- **Data Type** - Displays the data type of the corresponding simulation object, logic or an array.

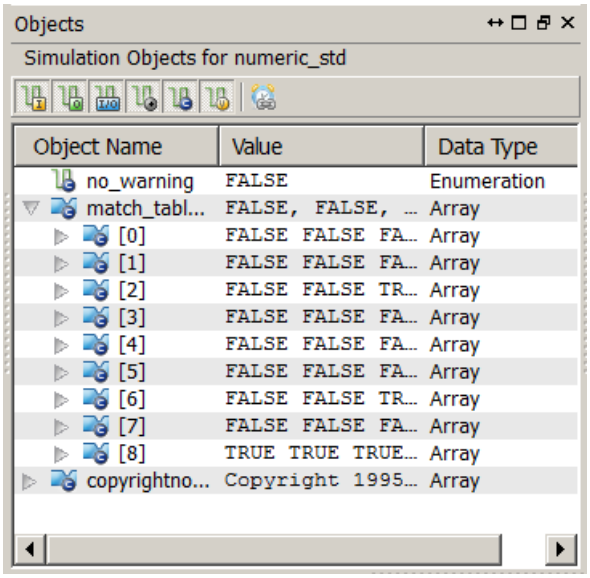


Figure 4-5: **Objects Panel**

Wave Window

The Wave window displays signals, buses and their waveforms. Each tab in the Wave window represents a wave configuration, which consists of a list of signals and buses, their properties, and any added wave objects, such as dividers, cursors, and markers.

In the user interface, the signals and buses in the wave configuration are traced during simulation, and therefore, the wave configuration is used to drive the simulation and to then examine the simulation results. Since design and simulation data are contained in a database, simulation data is not affected when adding signals to- or removing signals from the wave configuration.

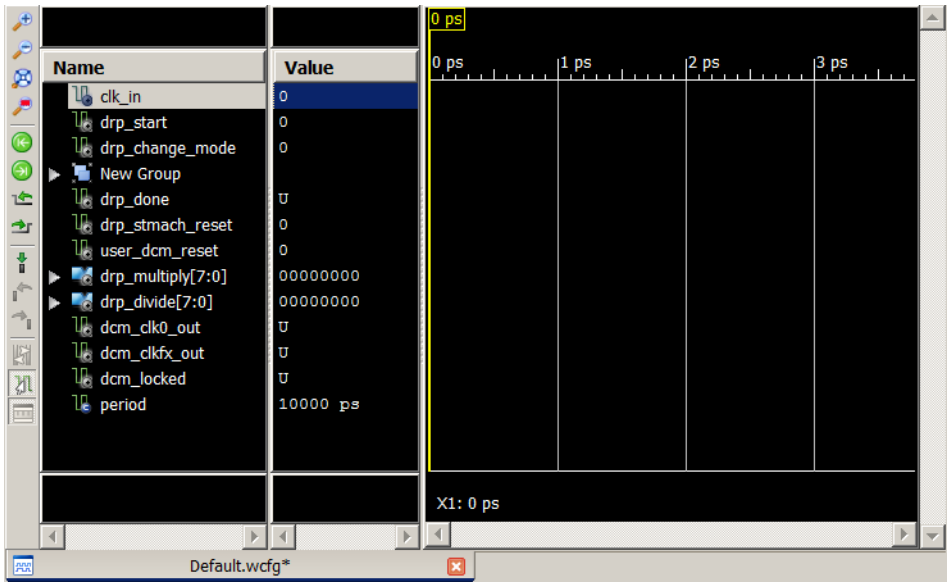


Figure 4-6: **Wave Window**

Text Editor

The text editor window is available for easy access to the HDL source files used in the simulation. Basic steps available are:

- Opening HDL source files for viewing and editing
- Setting breakpoints to source files for debugging
- Stepping through the source code

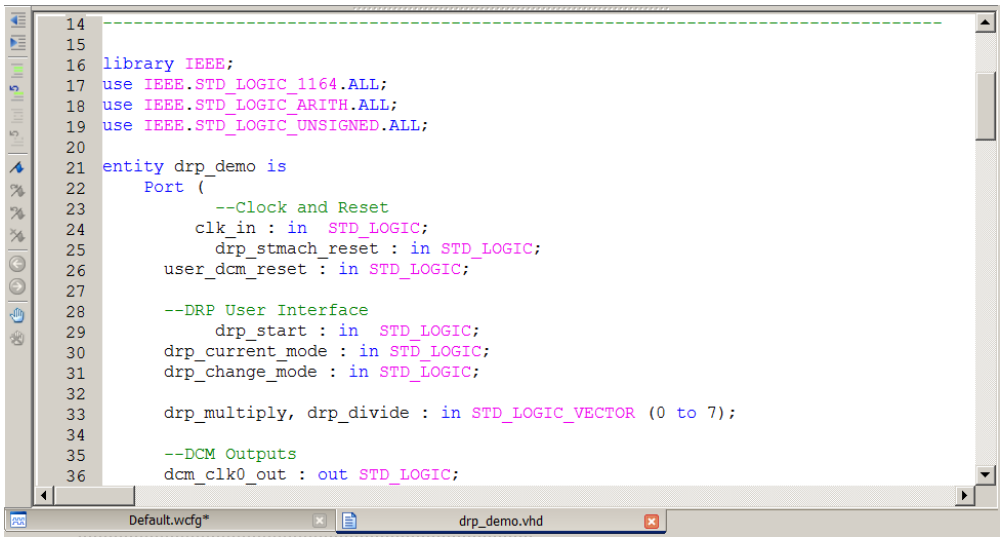


Figure 4-7: **Text Editor**

Breakpoints Panel

The Breakpoints panel displays a list of all breakpoints currently set in the design. For each breakpoint set in your source files, the list in the Breakpoints panel identifies the file location, file name and line number. You can delete a selection, delete all breakpoints, and go to the source code from the Breakpoint panel toolbar buttons or context menu.

For more information, see Chapter 4, "Debugging the Design" in the *ISim User Guide (UG660)*.

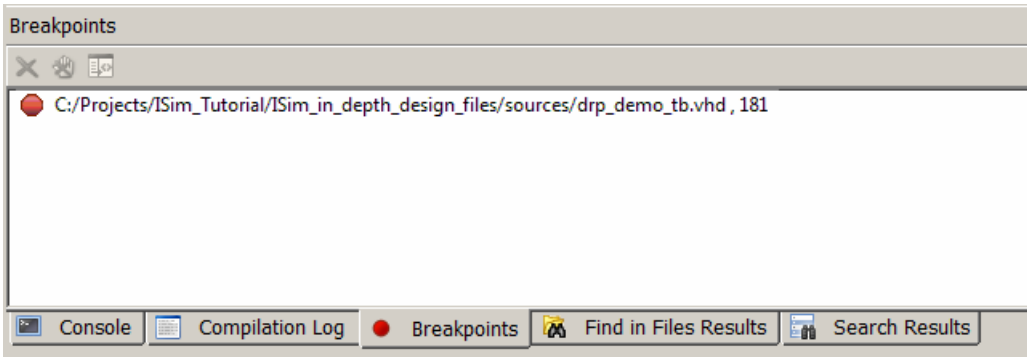


Figure 4-8: **Breakpoints Panel**

Console Panel

The Console panel enables you to view a log of messages generated by ISim, and to enter standard Tcl and ISim-specific commands at the command prompt.

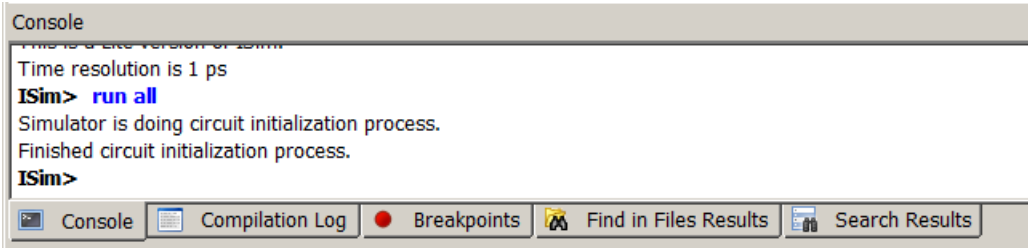


Figure 4-9: **Console Panel**

Examining the Design

In this section, you will perform several steps to further analyze the functional behavior of the tutorial design. These include:

- Running and restarting the simulation to review the design functionality, using signals in the wave window and messages from the test bench shown in the Console panel.
- Adding signals from the test bench and other design units to the Wave window so their status can be monitored.
- Adding groups and dividers in order to better identify signals in the Wave window
- Changing signal and Wave window properties to better interpret and review the signals in the Wave window.
- Using markers and cursors to highlight key events in the simulation and to perform zoom and time measurement features.
- Using multiple Wave window configurations to further enhance your ability to review multiple signals in one simulation session.

Adding Signals

Note: Skip this step if you completed Chapter 2, Running ISim from ISE Project Navigator. All visible simulation objects from the test bench have been added to the Wave window.

Prior to running simulation for a specified time, you must add signals to the Wave window in order to observe the signal status.

You will add all available simulation objects from the test bench to the Wave window, which include:

- **Input Clock (clk_in)**: This is a 100 MHz clock generated by the test bench and will be the input clock into the Digital Clock Manager (DCM).
- **Dynamic Reconfiguration Ports (DRP) (drp_*)**: These are signals associated with the DCM DRP feature. The test bench asserts and monitors these signals to control and review the DCM DRP functionality.
- **DCM Output signals (dcm_*)**: These are output clocks from the DCM.

To add these signals to the Wave window:

1. In the Instances and Processes panel, right-click the `drp_demo_tb` instance unit.
2. Select **Add to Wave Window**.

   All visible simulation objects from the `drp_demo_tb` test bench display in the Wave window. (Refer to Figure 4-10).

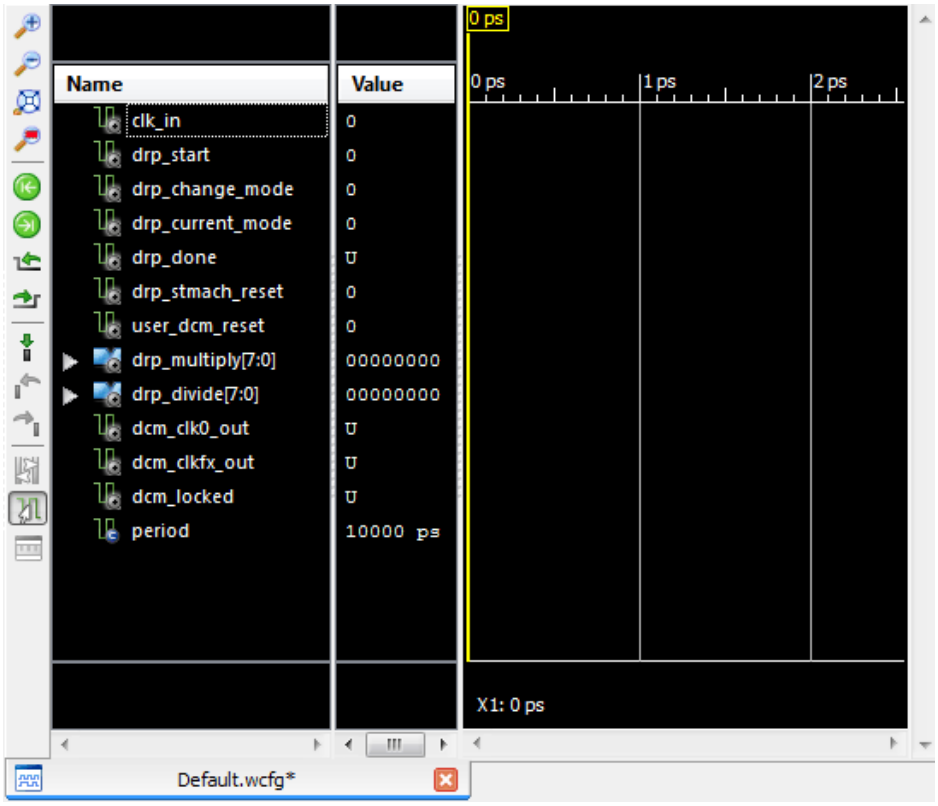


Figure 4-10: **Wave Window**

Running the Simulation for a Specified Time

You can now run the simulator for a specified time. Run the simulation for 5 microseconds (us).

1. In the ISim menu toolbar, type **5 us** in the Simulation Time field and click the **Run** toolbar button .

   Note: You can also type **run 5 us** at the Tcl prompt and press **Enter**.

   The wave window now shows traces of the signals up to 5 microseconds in simulation time (Refer to Figure 4-11).

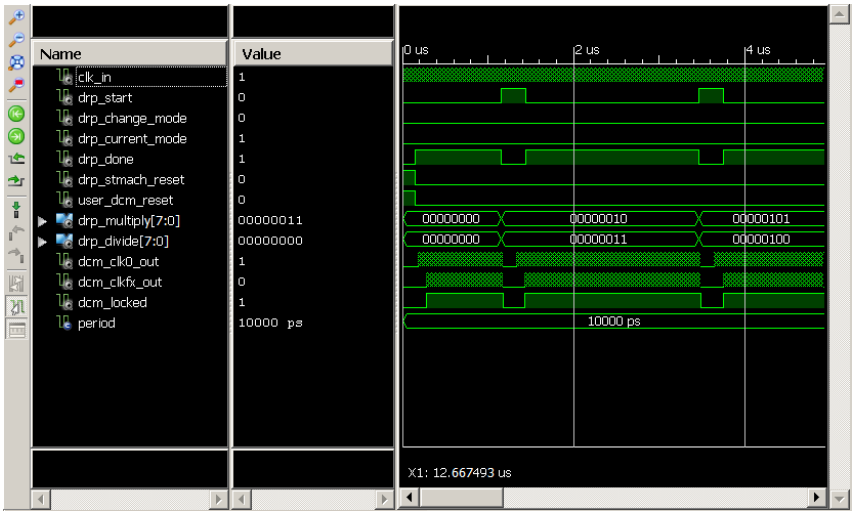


Figure 4-11: **Wave Window**

2. To display the full time spectrum in the Wave window, select **View > Zoom > To Full View** or click the **Zoom Full View** button .
3. You can use the horizontal and vertical scroll bars to view the full wave configuration.
4. There are assertions from the test bench during the time of simulation. Review the Console panel for messages from the test bench (Refer to Figure 4-12).

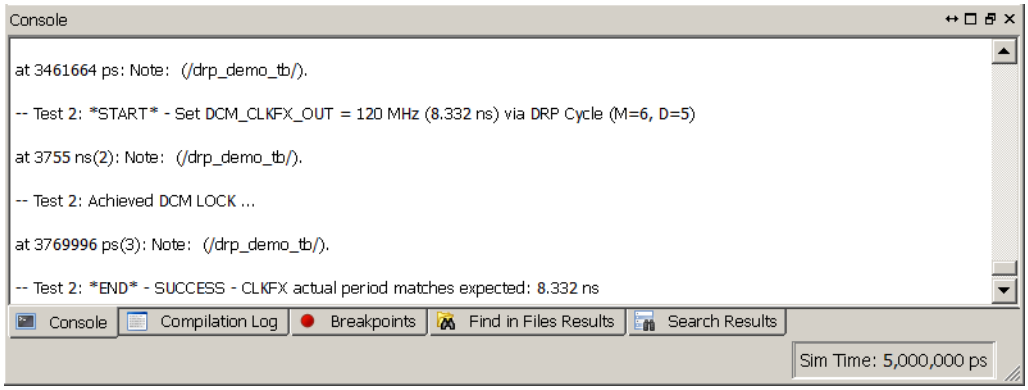


Figure 4-12: **Console Panel**

Restarting the Simulation

1. Before you continue, Click the Restart button  to restart the simulation to clear the Wave window and set the simulation time to 0 picoseconds (ps).

   Note: You can also restart the simulation by typing **`restart`** at the Tcl prompt.

   The wave window appears as shown in Figure 4-13.

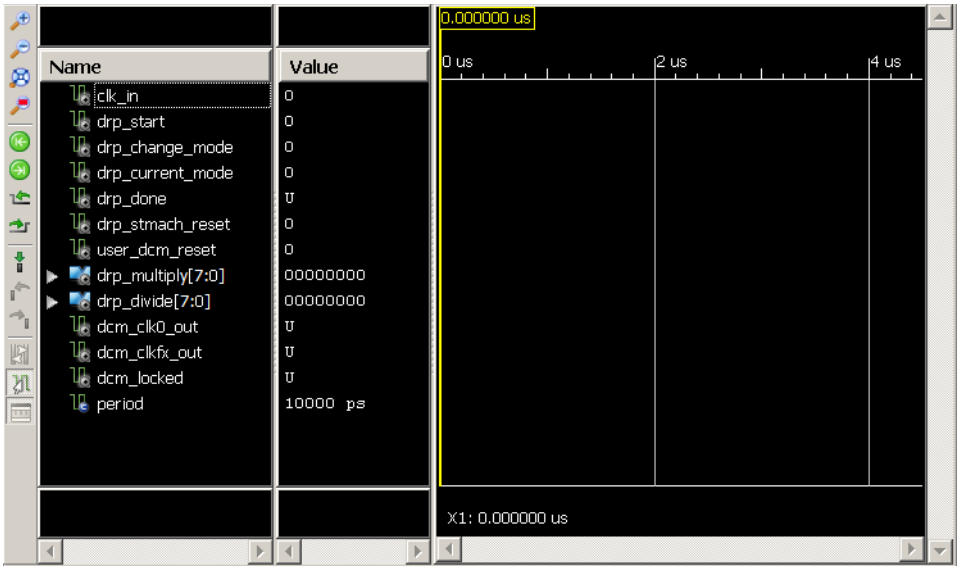


Figure 4-13: **Wave Window**

In the next section, you will be analyzing the simulation of the tutorial design in more detail using features from the Wave window, such as dividers, groups, cursors and markers.

Adding Groups

In the next steps, you will be adding signals from other design units in order to better analyze the functionality of this design. However, soon after you add additional signals to the wave window, the size of the wave window will not be large enough to display all signals in the same view. Reviewing all signals would require the use of the vertical scroll bar in the Wave window repeatedly, making the review process rather tedious.

We can remedy this situation by collecting signals into a group. With a group, you can collectively show or hide signals of similar purpose. To group signals in the wave configuration:

1. In the Wave window, click and hold the **Ctrl** key, and select all signals in the `drp_demo_tb` design unit that start with "drp_".
2. Right-click any selected signal and select **New Group**.
3. Type a name for the new group. For this example, name the group "DRP Test Signals".
4. A collapsed group is created in the Wave window. To expand the group, click once to the left of the group name.
5. Follow steps 1-3 above to make a group for all signals in the `drp_demo_tb` design unit that start with "dcm_". Name the group "DCM Test Signals".
6. Expand all the created groups. Your wave window should be similar to Figure 4-14.

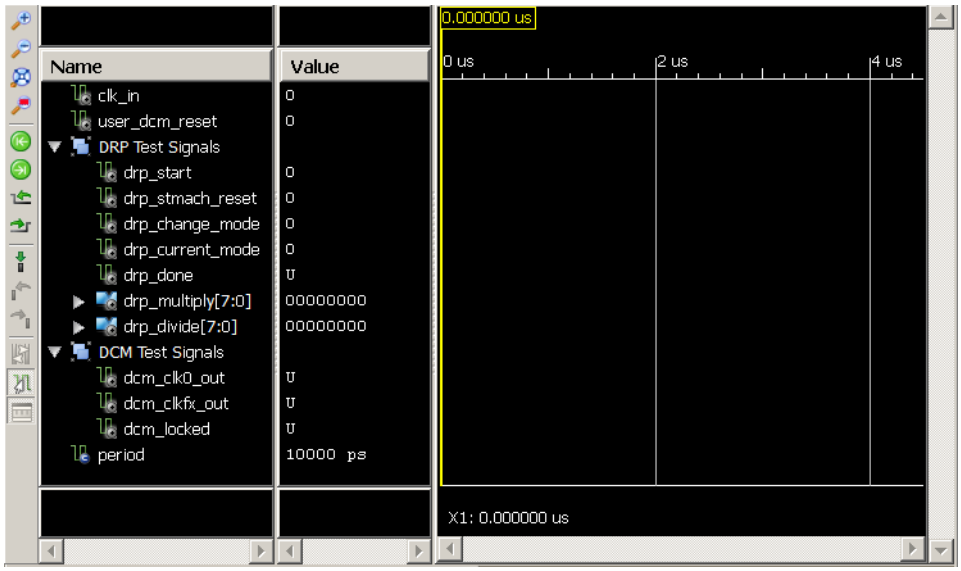


Figure 4-14: **Adding Groups**

If your signal groups do not match Figure 4-14, you can do the following to fix them:

- If you included an unrelated signal, use *cut and paste* to move it into the main list.
- If you created the group but missed a signal in the main list, use *drag and drop* to move the signal into the group. The signal will then be placed inside the group.
- You can undo the group using the **Edit > Undo** menu command.
- You can start over by ungrouping a group. Right-click it and select **Ungroup**.

Adding Dividers

Soon you will be adding signals from other design units in order to better analyze the functionality of this design. To better visualize which signals belong to which design units, you can add dividers to separate the signals by design unit.

To add dividers to the Wave window:

1. Right-click anywhere on the Wave window and select **New Divider**.
2. Enter a name for the divider.
3. Use the instructions above to add three dividers named:
   - TEST BENCH
   - DCM
   - DRP CONTROLLER
4. Click and drag the TEST BENCH divider to the top of the list.
5. Move the other dividers to the bottom of the list.

Note: Divider names can be changed at any time by double-clicking on the divider name or pressing the F2 function key, and entering a new name.

Your Wave window should be similar to the one shown in Figure 4-15.

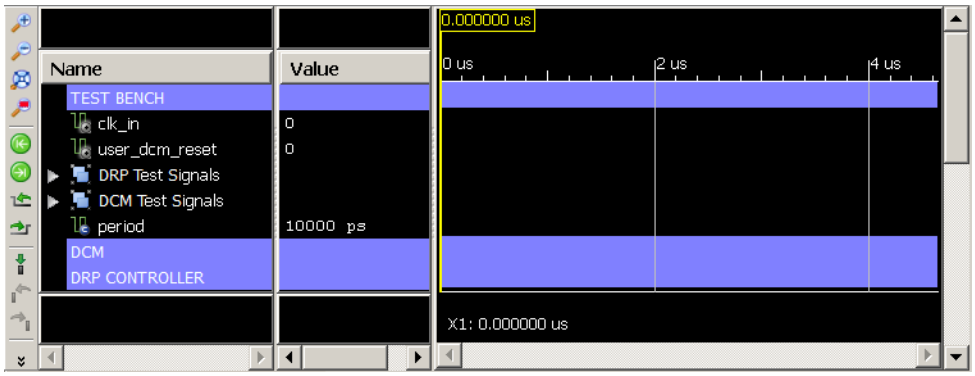


Figure 4-15: **Adding Dividers**

Adding Signals from Sub-Modules

You will now add signals from the instantiated DCM module (`Inst_drp_dcm`) and the instantiated DRP controller module (`Inst_drp_statmach`) in order to study the interactions between these sub-modules and the test bench test signals.

Follow these steps to add the necessary signals:

1. In the Instances and Processes panel, expand the hierarchy under drp_demo_tb as shown in Figure 4-16.
2. Click the Inst_drp_dcm entity.

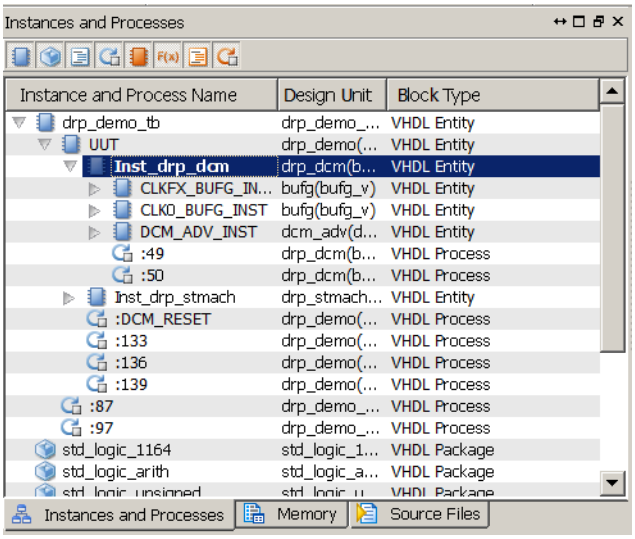


Figure 4-16: **Instances and Process Panel**

3. Simulation objects associated with the currently highlighted design unit appear in the Objects panel (refer to Figure 4-17).

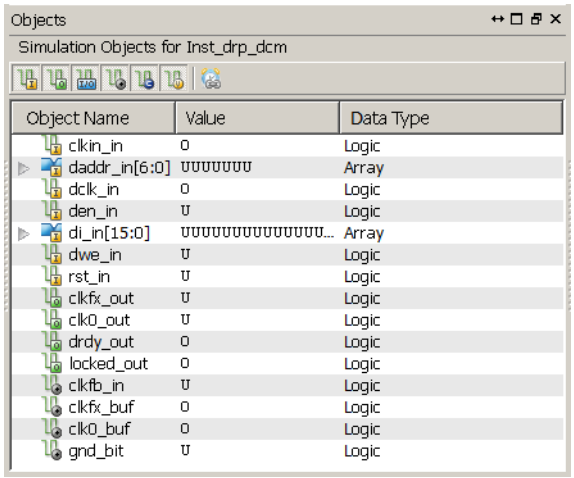


Figure 4-17: **Simulation Objects Panel**

Next, you will add all input and output ports, and internal signals from the `Inst_drp_dcm` design unit instantiation to the Wave window.

By default, all types of simulation objects (variables, constants, etc.) are displayed in the Objects panel. The types of objects are shown in Figure 4-18.

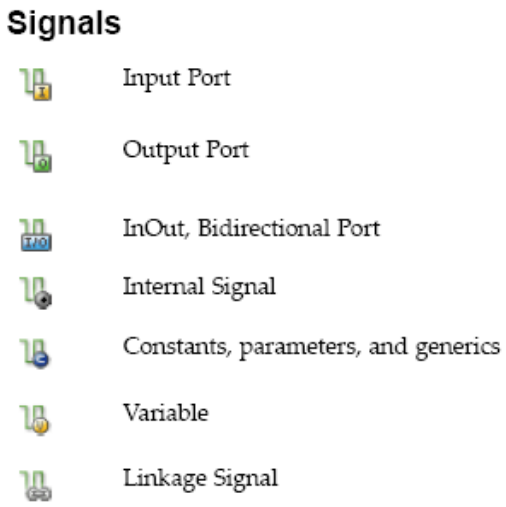


Figure 4-18: **Signals and Icons**

You can filter the type of simulation objects shown in this panel. Use the Objects panel toolbar to filter by inputs, outputs, bi-directional, internal, constants and variables. Toggle the desired object type by clicking on the corresponding button.

Figure 4-19: **Objects Panel Toolbar Buttons**

1. With the `Inst_drp_dcm` design unit in the Instance and Process panel selected, use the Objects panel toolbar buttons to display only input and output ports, and internal signals.
2. In the Objects panel, select all of the displayed objects, then drag and drop them into the Wave window, under the DCM divider.

   Note: You can also add these signals to the Wave window using the **`wave add`** Tcl command at the ISim Tcl prompt. For example:

   ```
   wave add /drp_demo_tb/uut/inst_drp_dcm
   ```
3. Repeat steps 1-3 to add the input/output ports of `Inst_drp_stmach` instantiated design unit below the DRP CONTROLLER divider.
4. Create groups for the signals you added. Define the groups as “Inputs”, “Outputs”, and “Internal” for each set of signals.

Your Wave window should be similar to the one shown in Figure 4-20 (with groups collapsed).

Figure 4-20: **Configuring the Wave Window**

Changing Signal and Wave Window Properties

Next, you will change the properties of some of the signals currently shown in the Wave window in order to better visualize the behavioral simulation.

Changing the Signal Name Format

By default, ISim adds signals to the waveform using a short name with the hierarchy reference removed. For some signals, it is important to know to which module they belong.

Do the following to change the signal name format for the drp_multiply and drp_divide bus signals.

1. In the wave window, **Ctrl**-click to select the drp_multiply and drp_divide signals listed in the DRP Test Signals group.
2. Right click and select **Name > Long**.

   The name format changes from Short to Long.

Changing the Signal Radix Format

Some signals are better interpreted if seen in hexadecimal rather than in binary. For example, the signals `drp_multiply` and `drp_divide` are bus signals that are best interpreted in hexadecimal format, rather than binary.

Do the following to change the radix options for these signals.

1. In the wave window, select the drp_demo_tb/drp_multiply and drp_demo_tb/drp_divide signals.
2. Right-click and select **Radix > Hexadecimal**.

Changing the Signal Color

ISim allows you to change the signal color in the Wave window to help you quickly distinguish similar signals from each other.

Do the following to change the color of the drp_multiply and drp_divide signals.

1. In the Wave window, right-click a signal name in the Name column.
2. Select **Signal Color**, and pick a color from the color palette or a custom color by clicking on the Browse (**...**) button.

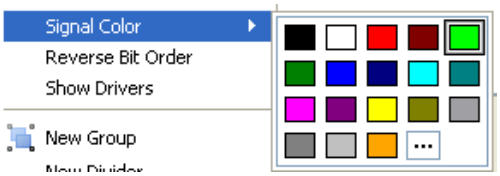


Figure 4-21: **Changing the Signal Color**

Note: You can also use the **Divider Color** option to change the color of the dividers you created in the Wave window.

Floating the Wave Window

Depending on your screen resolution, you might notice that the wave window has been populated with more signals than the interface can show at one time. To increase the viewable area, you can float the wave window. This opens a new window with just the waveform contents.

To float a window, click the **Float Window** button  on the main toolbar.

You are done making modifications to the Wave window. The Wave window should now look similar to Figure 4-22 when test bench groups are expanded.

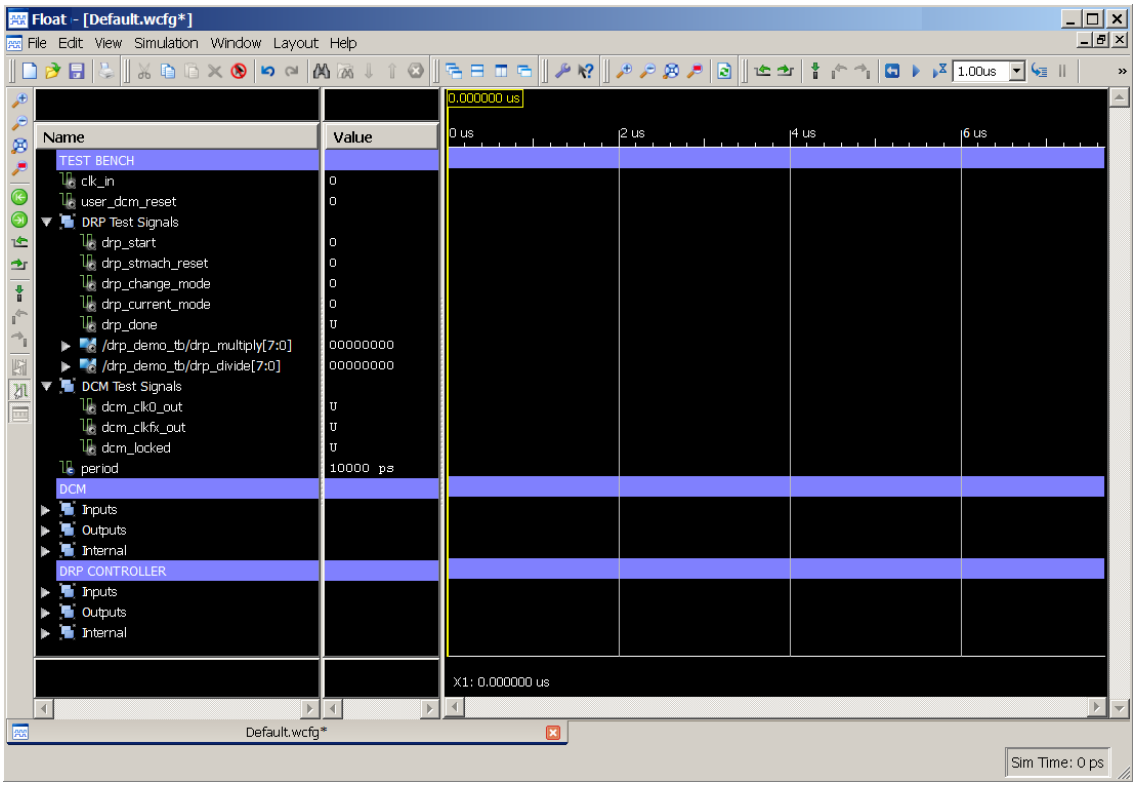


Figure 4-22: **Fully Configured Floating Wave Window**

Saving the Wave Window Configuration

You can save the current Wave Window configuration so it is available for use in future ISim simulation sessions of your design.

To save the wave configuration:

1. Select **File > Save As** to assign a name to the current wave configuration (Refer to Figure 4-23).
2. Save the current wave configuration to the filename **`tutorial_1.wcfg`**.

The wave configuration is now saved for future use.

Note: You can load the saved Wave window configuration using the menu command **File > Open**.

Simulating the Design

You are ready to simulate the design again with the updated wave configuration. Click the **Run All** button to re-run the simulation.

Note: You can also re-run the simulation by typing **`run all`** at the Tcl prompt.

The simulation will run for about 13 microseconds.

After the simulation is complete, click the **Zoom to Full View** button to zoom to full view.

The wave configuration should look similar to Figure 4-23.

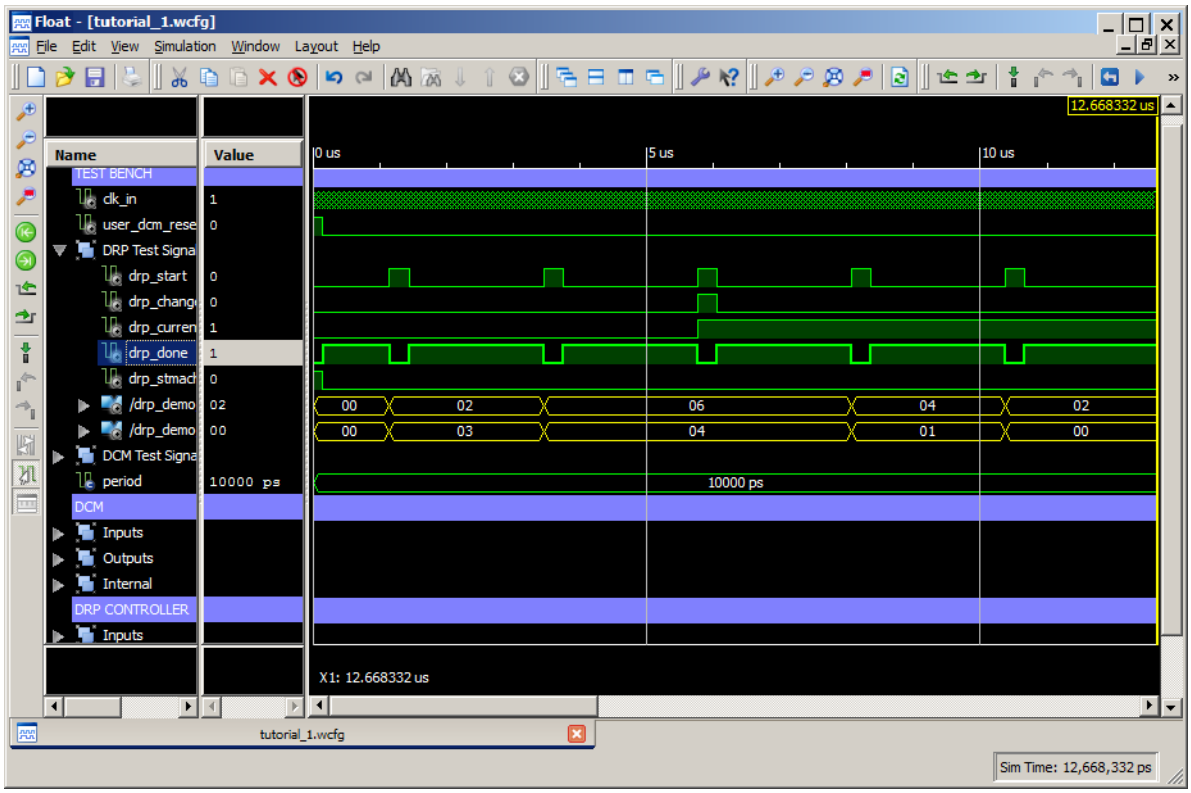


Figure 4-23: **Wave Window After 13 us Simulation Time**

Using Markers

The self-checking test bench used in this design performs four different tests to showcase the functionality of the DCM Dynamic Reconfiguration feature. Follow the next steps to use markers to mark each time a new test has started.

1. In the Console panel, identify the simulation times when each test has started. For example, Test 1 starts at about 1,150 ns, as shown by this segment of the ISim Console:

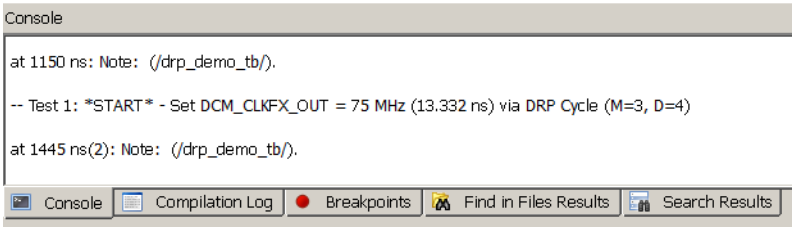


Figure 4-24: **Console Window**

2. From the ISim main menu, select **Edit > Go To** and enter **1150 ns** in the **Go To Time** field to move the main (yellow) cursor to the first test bench test.

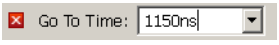


Figure 4-25: **Go To Time Field**

3. Click the **Add Marker** toolbar button  to add a marker at this time.

   Note: Although you typed the time in nanoseconds, the Wave window displays the time unit in microseconds. You could have also typed the measurement in picoseconds. You can use whichever measurement you prefer, and the Wave window will adjust accordingly.

4. Repeat steps 2-3 for all four tests performed by the test bench. The Wave window should look similar to Figure 4-26.

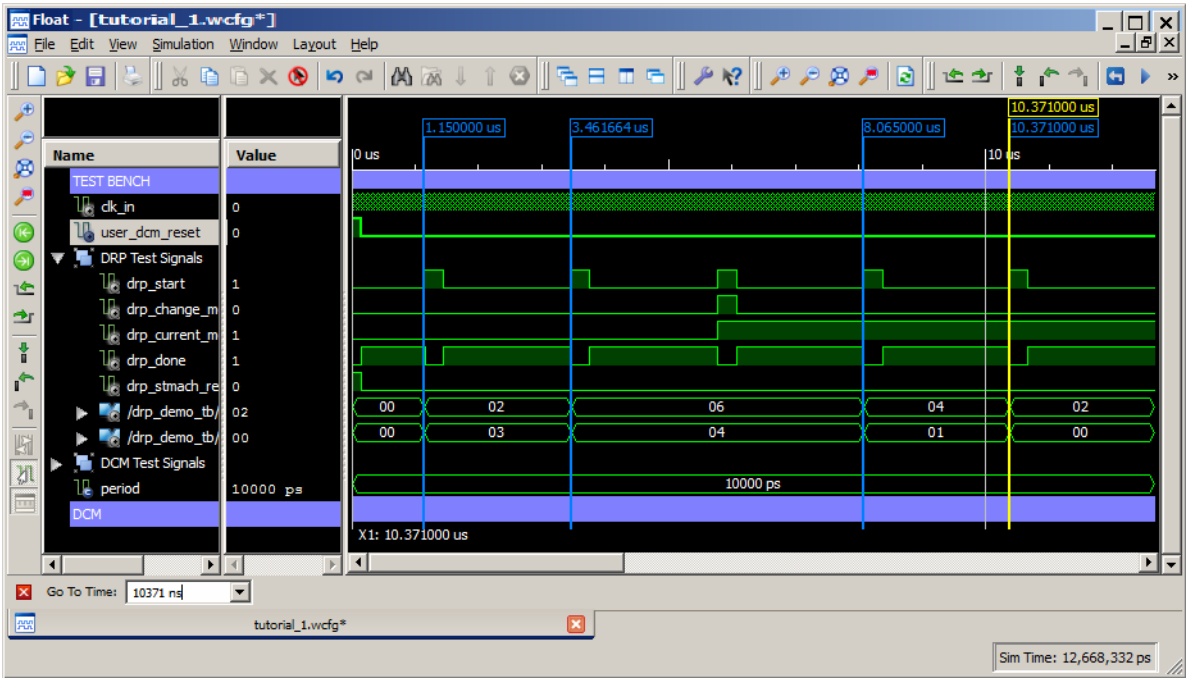


Figure 4-26: **Using Markers to Identify Start of Tests**

Using Cursors

The ISim Console reports that Test 2 and Test 4 failed (Figure 4-27).

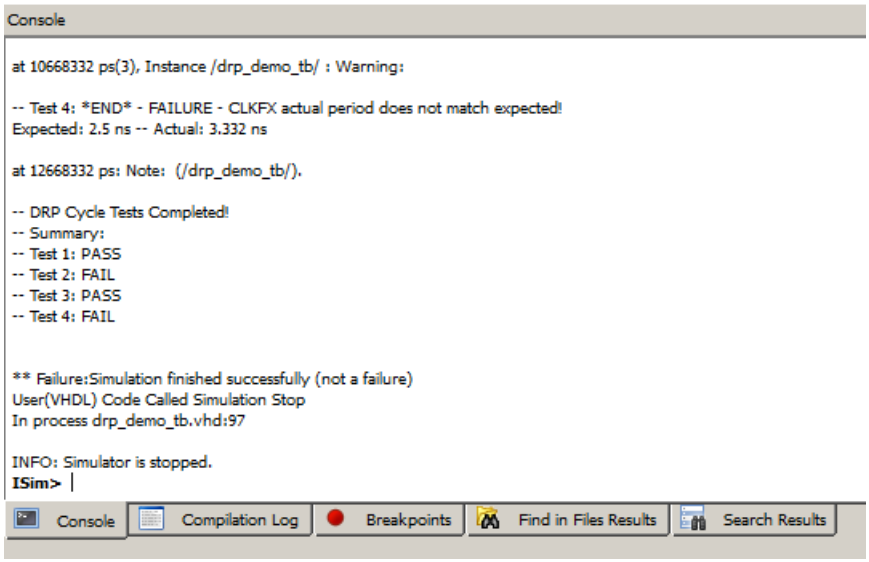


Figure 4-27: **Console Report: Tests 2 and 4 Failed**

In Tests 2 and 4, a Dynamic Reconfiguration (DRP) write cycle is performed in order to change the multiply and divide factors of the Digital Frequency Synthesizer and set new clock output (CLKFX) frequencies (120 MHz and 400 MHz, respectively). However, at the end of the DRP cycle, the test bench measured a period that did not match the expected period. Tests 2 and 4 failed due to the period discrepancy. You can view the failure for Test 2 in Figure 4-28.

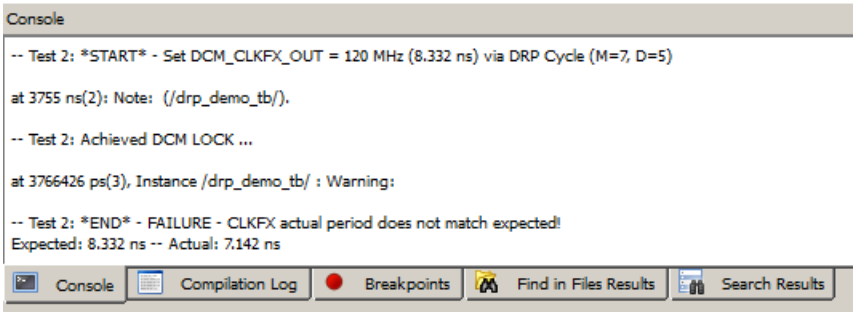


Figure 4-28: **Test 2 Fails Due To Period Discrepancy**

In the next few steps, you will use the ISim main cursor (yellow cursor) to zoom in the wave window when one of the failing tests takes place. You will also use the cursor to measure the period of signal dcm_clkfx_out and verify that the test bench is making accurate measurements.

Zooming In

First, in the Wave window, zoom in to where Test 2 starts to review the status of output clock dcm_clkfx_out.

To use a cursor for zooming in on a specific area:

1. Place the cursor on the desired area by clicking and dragging the main (yellow) cursor close to the marker that represents the start of Test 2 (marker at time 3.461664 µs). The cursor will snap onto the marker.

   Note: You can also click the **Previous Marker** and **Next Marker** toolbar buttons to quickly move the main cursor from marker to marker.

2. Zoom in using the **Zoom In** button .

   The Wave window zooms in around the area specified by the cursor.

3. Use step 2 above repeatedly until you can clearly see the rising and falling edges of DCM test signals `dcm_clk0_out` and `dcm_clkfx_out`.

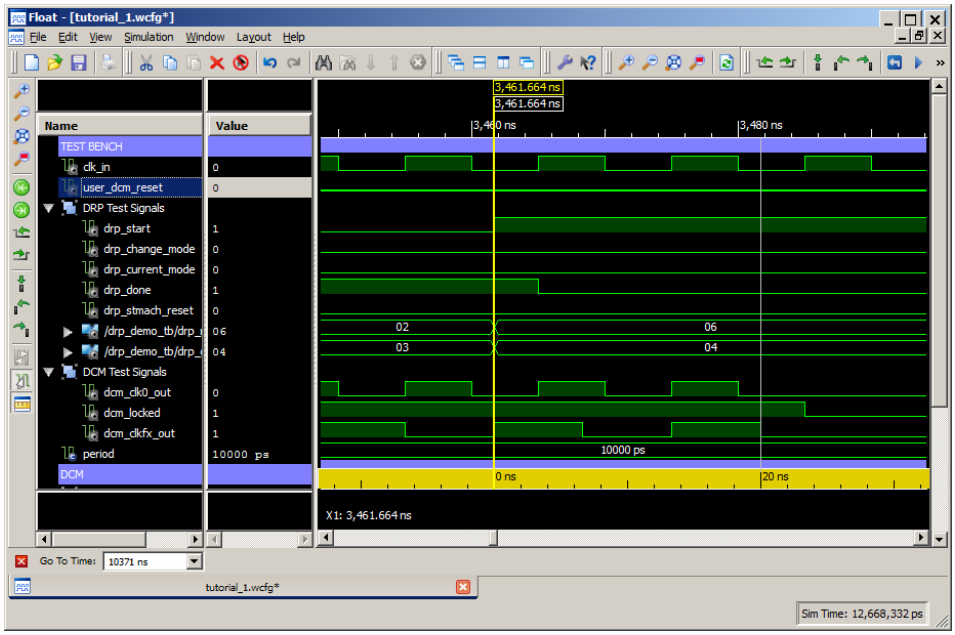


Figure 4-29: **Viewing dcm_clk0_out and dcm_clkfx_out**

Measuring Time

You can use your mouse cursor to measure time between two endpoints. You will use this feature to confirm the test bench calculations reported in the Console during Test 2 by measuring the period of `dcm_clkfx_out` after the DRP cycle has completed (signal `drp_done` is asserted).

To measure time using cursors:

1. Use the **Snap to Transition** toggle button to snap the cursor to transition edges.
2. Click and hold in an area around the first clock rising edge following DRP cycle completion (`drp_done` signal asserted). The main cursor will snap to the rising edge of `dcm_clkfx_out`.
3. While holding the button, move the mouse over to the next clock rising edge. A second marker should appear.

   The time between the two defined endpoints appears at the bottom of the wave window as a time delta (refer to Figure 4-30).

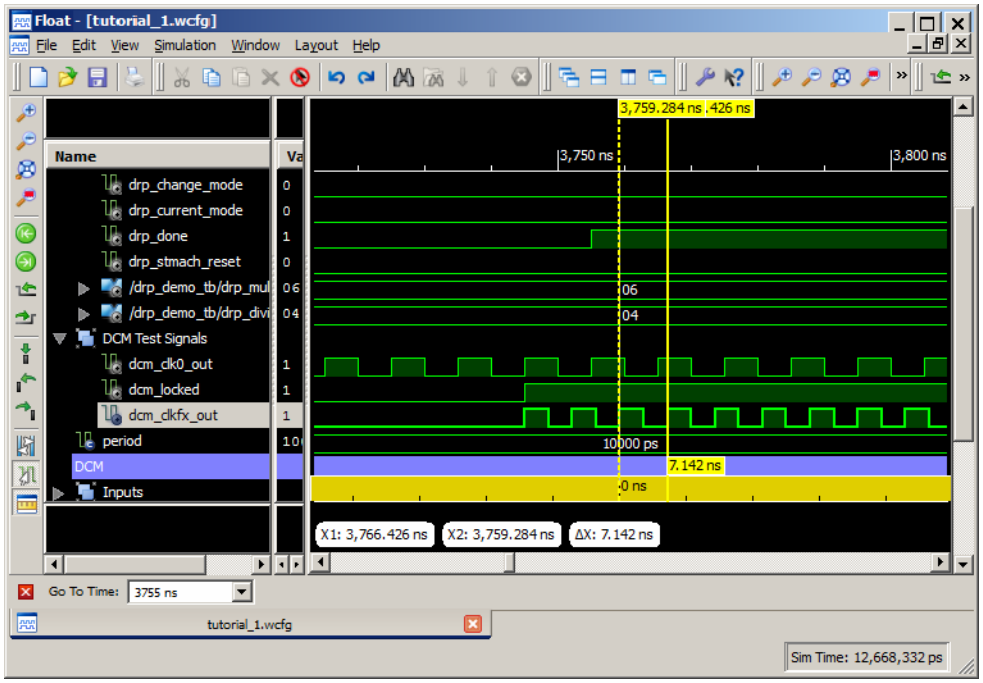


Figure 4-30: **Measuring Time in the Wave Window**

Using the cursors, we measure a 7,142 ps time difference between two rising edges of the `dcm_clkfx_out` output clock. This translates to a 140 MHz clock signal. Test 2 fails due to the frequency discrepancy (expected is 120 MHz).

4. Repeat the same steps above to analyze the Test 4 failure. You should observe that while the test bench expects a frequency of 400 MHz, the actual frequency measured is 300 MHz.

Note: You can use the Floating Ruler feature (available from the wave window toolbar) to display a hovering ruler over the wave configuration. This feature is available when performing a time measurement using cursors between two endpoints. The zero (0 ps) on the ruler is placed at the first time endpoint. This feature is useful when making multiple time measurements with respect to the first endpoint.

Using Multiple Wave Configurations

Depending on the resolution of the screen, a single Wave window might not display all the signals of interest at the same time. You can resolve this problem by opening multiple Wave windows, each with their own set of signals and signal properties.

To open a new Wave window:

1. In ISim, select **File > New**.
2. In the New dialog box, select **Wave Configuration** and click **OK**.

   A blank wave configuration opens.

You can move dividers, groups and simulation objects to the new wave configuration. Do the following to move all of the simulation objects associated with the DCM and DRP Controller units to a new wave window.

1. Press and hold the **Ctrl** key, and highlight objects (dividers, groups, and so on) you want to move to the new wave window.
2. Right-click any selected signals, and select **Cut**.

   Note: Although you are moving signals from one Wave Configuration to another for this tutorial, you can also use the **Copy** command to copy the signals so that they remain in both windows.
3. Click the window tab for the new wave configuration.
4. Right-click in the Name column area of the wave configuration, and select **Paste**.
5. Select **File > Save As** to save this wave configuration as `tutorial_2.wcfg`.

You should now have two wave windows that should look similar to Figure 4-31 and Figure 4-32.

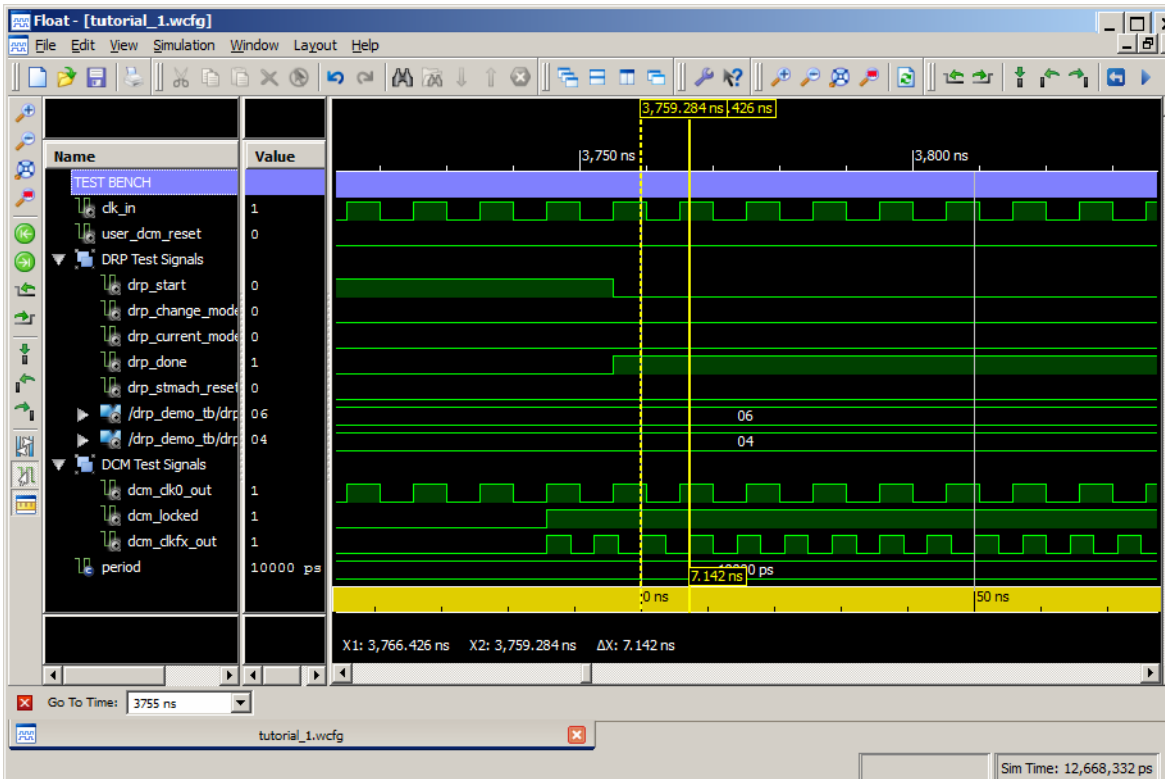


Figure 4-31: **tutorial_1.wcfg**

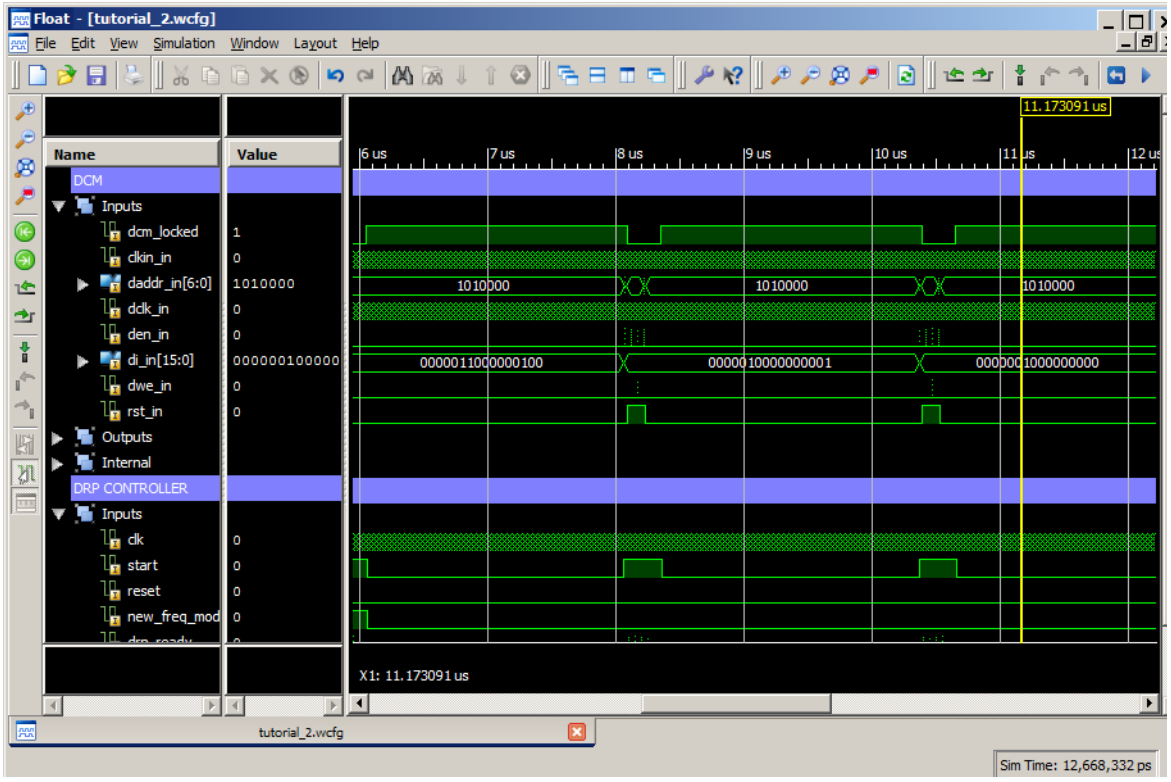


Figure 4-32: **tutorial_2.wcfg**

Debugging the Design

Now that you have examined the design using markers, cursors, and multiple wave configurations, you will now use ISim debugging features, such as setting breakpoints and stepping through source code, in order to debug the design and address the two failing DRP tests.

Viewing Source Code

First, take a look at the test bench for the tutorial design and learn how each test is performed.

Do one of the following to open the source code for the tutorial design test bench (`drp_demo_tb.vhd`). The source file will be opened in read-only mode using the integrated text editor (Refer to Figure 4-33).

- Select **File > Open** to point to the file of choice.
- In the Instances and Processes Panel, right-click on the design unit described by the source file of interest, then select **Go to Source Code**.
- In the Objects Panel, right-click on any of the simulation objects declared in the source file of choice, then select **Go to Source Code**.
- In the Source Files Panel (viewable by clicking on the "Source Files" tab), double-click a source file.

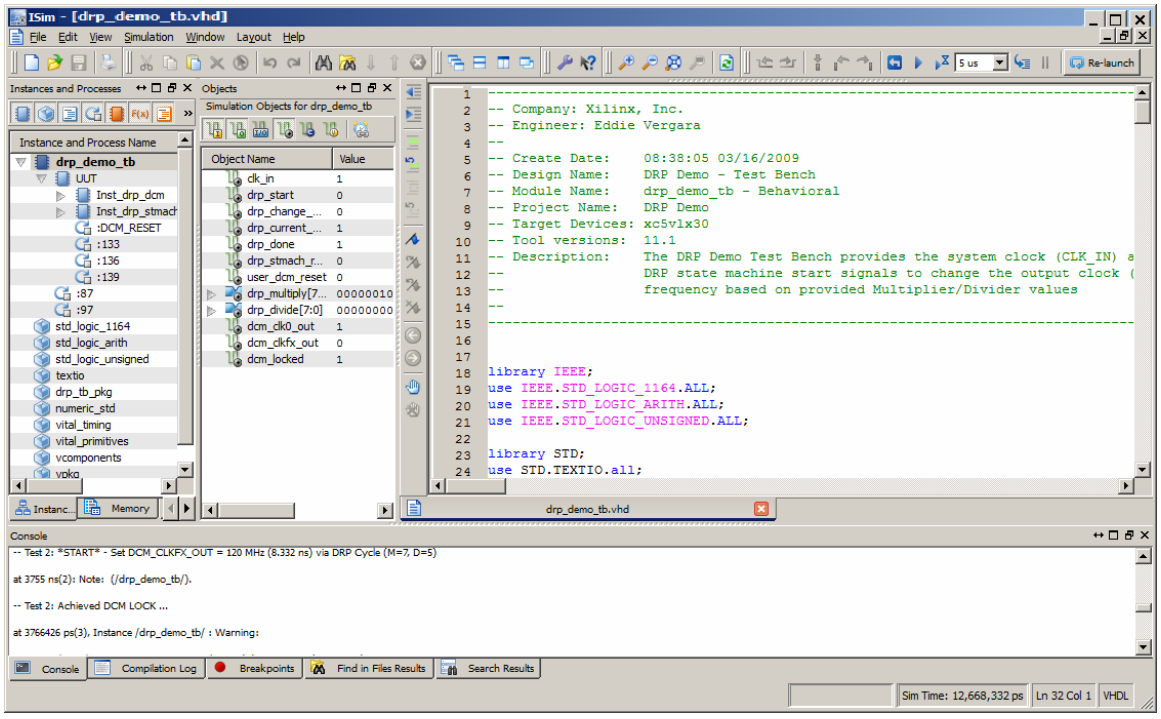


Figure 4-33: **Integrated Text Editor**

Using Breakpoints and Stepping

A breakpoint is a user-determined stopping point in the source code used for debugging the design with ISim. When simulating a design with set breakpoints, simulation of the design stops at each breakpoint in order to verify the design behavior. Once the simulation stops, an indicator is shown in the text editor next to the line of source code where the breakpoint was set, allowing you to compare the Wave window results with a particular event in the source code.

Another useful ISim debugging tool is the Stepping feature. With stepping, you can run the simulator one simulation unit at the time. This is helpful if you are interested in learning how each line of your source code affects the results in simulation.

We can use both of these debugging features to learn how the DRP cycle is performed during Test 2 in an attempt to debug the failing test.

Setting Breakpoints

Begin by first setting a breakpoint around the first signal assignment performed during each of the DRP cycle tests.

Do the following to set a breakpoint at line 185 in `drp_demo_tb.vhd` (Refer to Figure 4-34). Setting a breakpoint causes the simulator to stop every time the signal `drp_multiply` is assigned a value.

1. Open the source code in which you want to add the breakpoint.
2. Go to an executable line in the source code.
3. Right-click anywhere on the executable line and select **Toggle Breakpoint** to add the breakpoint.

   Note: You can also add a breakpoint by clicking the **Toggle Breakpoint** button  .

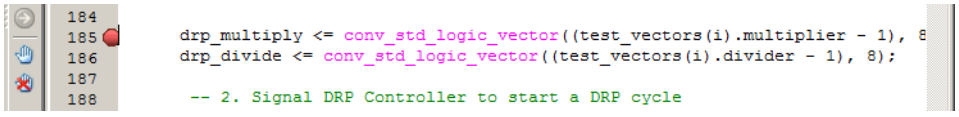


Figure 4-34: **Setting a Breakpoint at Line 185 in drp_demo_tb.vhd**

Note: You can manage breakpoints by clicking on the Breakpoints tab, located next to the Console tab. All breakpoints appear in this list. From here, you can:

- Delete a selected breakpoint or all breakpoints
- Go to the line of source code for a selected breakpoint

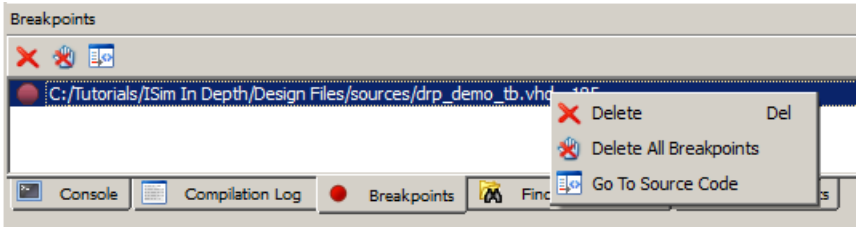


Figure 4-35: **Breakpoints Tab and Right-Click Menu Options**

Re-running the Simulation with the Breakpoint Enabled

Next, you will re-run the simulation with the breakpoint enabled.

Debugging with the breakpoints and stepping features work best when you are able to review the console output and the Wave window at the same time. Use the Float feature of the ISim panels, or resize the windows of the simulator, to best accommodate the windows so they can be viewed at the same time.

1. Click the **Restart** button to restart the simulation.
2. To run the simulation, click the Run All button .

The simulation runs near the start of the first test.

Focus changes to the text editor where it shows the yellow indicator ( ) at the last line of source code the simulator executed.

```
183        -- 1. Set Multiplier and Divider values from test vector
184
185      drp_multiply <= conv_std_logic_vector((test_vectors(i).multiplier - 1), 8
186      drp_divide <= conv_std_logic_vector((test_vectors(i).divider - 1), 8);
187
```

Figure 4-36: **The Last Line of Executed Source Code**

Additionally, a message appears in the Console to indicate that the simulator has stopped, including the line of source code last executed by the simulator.

We know that Test 1 finished successfully when we examined the design earlier. Therefore, we can skip debugging this test.

3. To continue forward to Test 2, click the Run All button .

The simulation now stops at the start of Test 2.

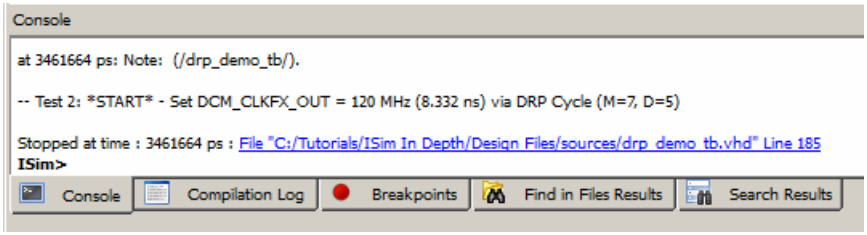


Figure 4-37: **Message in the Console Indicating That the Simulator Has Stopped**

Stepping through Source Code

You first need to verify that in Test 2, the appropriate Multiplier and Divider parameters are being set correctly via the `drp_multiply` and `drp_divide` bus signals. You will use stepping to step through the source code line by line and review how the `drp_multiply` and `drp_divide` bus signals are assigned to the DCM DRP ports.

To step through a simulation, click on the Step toolbar button  .

Note: You can also step through the simulation by typing **`step`** at the Tcl prompt.

1. Use this process to step through the design, paying close attention to each of these events:
   - `drp_multiply` and `drp_divide` bus signals are assigned values from a constant test_vectors.
   - `drp_start` asserts in order to start a DRP cycle.
   - `drp_multiply` bus signal is assigned to the 8 uppermost bits of bus signal DI_IN, while `drp_divide` bus signal is assigned to the 8 lowermost bits of the same bus.
   - The DRP controller (`drp_stmach.vhd`) leaves idle mode and moves to the next DRP cycle step, clearing the DCM status registers.
2. In the "tutorial_2" wave window, expand the DCM Inputs bus.
3. Continue stepping through the simulation until the `di_in` bus signal is updated with a new value (you might need to zoom in considerably in order to observe the change). At around 3,465 ns, the bus should be updated from 0203h to 0604h.

   Note: Change the radix of bus signal `di_in` to Hexadecimal to verify this value change.

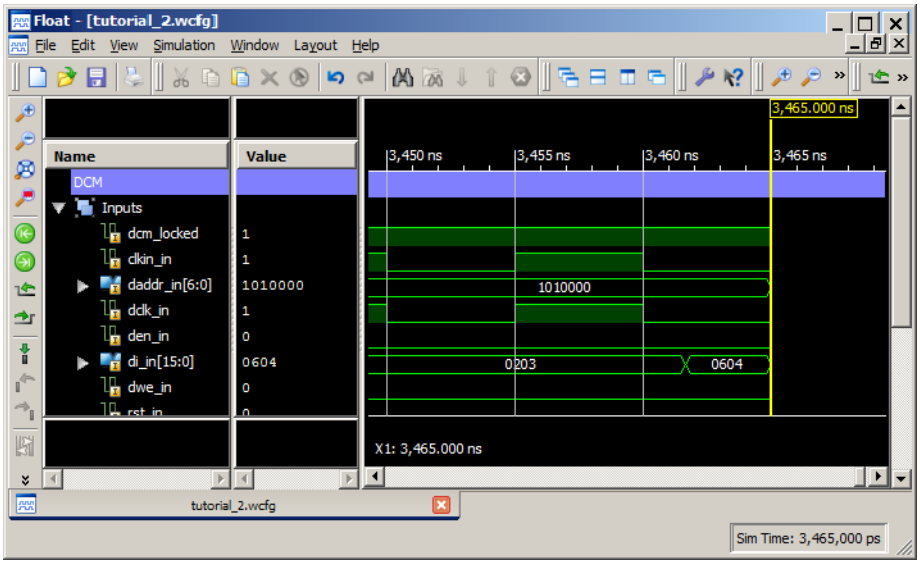


Figure 4-38: **Analyzing Output of DCM DI_IN Input Bus on the Wave Window**

The output clock frequency of this design (`dcm_clkfx_out`) is dependent on the multiply and divide factors you provide. For Test 2, we use the following parameters and expected output clock frequency:

Table 4-1: **Parameters and Expected Output Clock Frequency**

Test	Freq. (MHz)	Period (ps)	Multiplier (M)	Divider (D)
2	120	8,332	6	5

You might recall that for M=6 and D=5, `di_in[15:0]` bus value should be 0504h. Notice that the status of `di_in` in Test 2 is 0604h. Test 2 fails because an incorrect M/D factor is provided via the `drp_multiply` and `drp_divide` signals in the test bench.

You can repeat the steps above to determine the cause of failure for Test 4. You will determine that the failure is also due to incorrect assignments of the multiply and divide signals in the test bench.

Fixing Bugs in the Design

By using breakpoints and stepping, you have determined that the incorrect multiply and divide values are assigned to the `drp_multiply` and `drp_divide` signals in the test bench.

In the next steps, revise the test bench test vectors to use the correct Multiplier and Divider parameters in Tests 2 and 4.

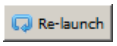
1. In the ISim main window, click the **Source Files** tab.
2. Right-click the `drp_demo_tb.vhd` file and select **Go To Source Code**.
3. In lines 117 through 127, the test vectors for the four DRP tests are defined. Revise the constant declaration to the values displayed below (changes highlighted in **bold**).

```
-------------------------------------------------
-- ** TEST VECTORS **
--  (Test, Frequency, Period, Multiplier, Divider)
-------------------------------------------------

constant test_vectors : vector_array := (

  ( 1, 75, 13332 ps, 3, 4),
  ( 2, 120, 8332 ps, 6, 5),
  ( 3, 250, 4000 ps, 5, 2),
  ( 4, 400, 2500 ps, 4, 1));
```

4. Save the file.

Verifying Bug Fix

Now that you have corrected the test bench source code, you need to re-compile the source code and build a new simulation executable.

1. Click the **Breakpoints** tab and remove the breakpoint that you previously set.
2. Click the **Re-launch** button 

to re-launch ISim.

   ISim recompiles the source files and reloads the simulation.
3. You are ready to simulate the design again with the updated test bench. Click the Run All button  to re-run the simulation.

   Note: You can also re-run the simulation by typing **`run all`** at the Tcl prompt.

If the test vectors in the test bench were properly revised, the simulation should run to completion, showing that all tests pass (Figure 4-39).

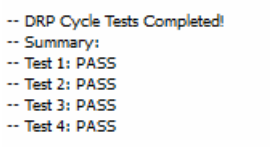


Figure 4-39: **Console Showing That All Tests Pass**

What's Next

This completes the ISE Simulator (ISim) In-Depth Tutorial. Refer to the Additional Resources section in the Preface for links to more detailed information about the ISE Simulator.