Precision RTL Synthesis Users Manual

2003c Update1

March 2004

End-User License Agreement

Trademark Information

Table of Contents

About This Manual ix

Chapter 1
Introducing Precision RTL Synthesis 1-1

Precision RTL Synthesis Features 1-1
  Intuitive User Interface 1-1
  Project Manager 1-1
  Advanced Synthesis Algorithms 1-2
  Constraint Driven Synthesis 1-2
  Boundaryless Optimization 1-2
  FSM Optimization 1-2
  PreciseTimeTM Timing Analysis 1-3
  Interactive Timing Analysis 1-3
  Register Retiming 1-3
  Schematic Viewing with Critical Path Fragment Filtering 1-3
Integrated Creation and Analysis Tools 1-3
  Design Bar 1-4
  HDLInventor 1-4
  Project Browser 1-5
  Design Browser 1-5
  Schematic Viewing 1-6
  P&RIntegrator 1-6
Documentation Available Online 1-7
  Context-Sensitive Help 1-7
  Product Manuals 1-8

Chapter 2
Setting Up a Design and Compiling 2-1

Invoking Precision RTL Synthesis 2-1
  Invoking the Precision GUI 2-1
  Invoking Precision from a Shell 2-2
Setting up the Design Environment 2-2
  The Project Directory and the Results Directory 2-4
  Setting the Input Directory 2-5
  Setting the Technology 2-5
  Adding Input Files to the Project 2-6
  Setting Synthesis Options 2-12
Compiling the Design 2-13
  Evaluating the Results 2-13
Setting Timing and Design Constraints 2-17
Synthesizing the Design 2-18
  Evaluating the Results 2-19

Table of Contents (cont.)

Performing Initial Place and Route 2-22

Chapter 3
Setting Constraints and Synthesizing 3-1

Managing a Master Constraint File 3-1
Creating a Master Constraint File 3-1
Maintaining a Master Constraint File 3-1
Setting Timing Constraints 3-2
Setting Global Timing Constraints 3-2
Clocks Overview 3-3
Derived Clocks 3-5
Report Timing and Clock Names 3-6
Specifying Clock Constraints 3-7
Handling Multiple Clocks 3-9
Specifying Input Delay 3-10
Specifying Output Delay 3-13
Setting False Paths 3-15
Setting Multicycle Paths 3-17
Setting Mapping Constraints 3-18
Mapping Ports to Pin Numbers and IO Pads 3-18
Mapping Operators, Counters, and Memory to Chip Resources 3-21
Setting Other Constraints 3-21
Controlling Hierarchy 3-21
Protecting Blocks from Change (dont_touch) 3-23
Controlling Fanout on Data Nets 3-23

Chapter 4
Managing Projects 4-1

The Project Manager 4-1
Bypassing the Project Manager 4-2
Using the Project Manager Interactively 4-3
Creating a Project 4-3
Opening a Project 4-8
Closing a Project 4-9
Setting Up the Implementation 4-10
Creating a New Implementation 4-10
Creating an Implementation from the Command Line 4-10
Saving the Active Implementation 4-11
Renaming an Implementation 4-12
Activating an Implementation 4-13
Copying an Implementation 4-14
Commenting an Implementation 4-15

Table of Contents (cont.)

Deleting an Implementation 4-17
Creating a Script from a Precision.log File 4-17
Scripts for Creating New Projects or Reusing Existing Projects 4-18
Exporting your Settings 4-20
Actions to Take if the Project, Input Directory, or Input Files Move 4-20
Backward Compatibility with Pre-2003c Projects 4-21

Chapter 5
Viewing a Schematic 5-1

A Quick Way to Bring Up a Schematic 5-1
Viewing the RTL Schematic 5-2
Viewing the Technology Schematic 5-3
Viewing the Critical Path Schematic 5-4
Understanding the Left Mouse Button Actions 5-5
Traversing the Schematic with Strokes 5-5
Zooming to Area 5-5
Zooming Out 5-6
Zooming In 5-6
Zooming to Fit 5-6
Moving Up and Down the Hierarchy 5-7
Paging Forward ( ) and Paging Back ( ) 5-7
Tracing a Signal to the Next Page 5-8
Panning the Schematic 5-8
Centering an Object in the Schematic View 5-9
Viewing the Internals of a Hierarchy Block 5-10
Traversing the Schematic Using Keyboard shortcuts 5-11
Using the Cross Probe Features 5-12
Cross Probing from Schematic to the HDL Source 5-12
Cross Probing to HDL Designer from Precision RTL Synthesis 5-12
Viewing Schematic Fragments 5-13
Tracing a Signal 5-13
Viewing a Trace Schematic 5-14
Click and Sprout the Trace Path 5-14
Using the Find Window Features 5-15
Conducting a Simple Search for Objects 5-15
Searching for Objects Using Regular Expressions 5-16
Setting Schematic Viewing Options 5-18
Understanding the Term “Bused Instance” 5-18
Viewing Bundled Instances and Net Buses 5-19
Changing the Schematic Printing Defaults 5-19
Changing the Way Symbols are Handled 5-20
Changing the Placement of Output Devices 5-20
Changing the Viewer Display and Color Options 5-21

List of Figures

Figure 2-1. The Design Center Window 2-3
Figure 2-2. Specifying Input File Properties 2-8
Figure 2-3. Preparing the Input UCF File 2-18
Figure 2-1. The Constraint Relationship 2-21
Figure 3-1. Clocks Example 3-4
Figure 3-2. The Clock Period Defines the Maximum Delay Between Flip Flops 3-7
Figure 3-3. Setting the Clock Constraints from the GUI 3-8
Figure 3-4. Input Delay Defined 3-10
Figure 3-5. Setting Input Delay from the GUI 3-11
Figure 3-6. Output Delay Defined 3-13
Figure 3-7. Setting Output Delay from the GUI 3-14
Figure 3-8. Setting a False Path from the GUI 3-16
Figure 3-9. Multicycle Path Defined 3-17
Figure 4-1. Organization of Project Files by the Project Manager 4-2
Figure 4-2. Initial Session Window and New Project Dialog Box 4-3
Figure 4-3. Newly Created Project and Implementation 4-5
Figure 4-4. Hierarchical Project File Structure 4-6
Figure 4-5. Setting the Input Directory 4-8
Figure 4-6. Project Manager Icons on the Design Bar 4-9
Figure 4-7. Project Popup Menu 4-10
Figure 4-8. New uart_xilinx Implementation 4-11
Figure 4-9. Menu and Dialog for Renaming the Active Implementation 4-12
Figure 4-10. Select Implementation Dialog Box 4-13
Figure 4-11. Dialog Box Listing all Inactive Implementations 4-14
Figure 4-12. Menu and Dialog for Editing Implementation Comment Property 4-16
Figure 5-1. A Quick Way to Bring Up a Schematic 5-1
Figure 5-2. Design Browser with RTL Schematic 5-2
Figure 5-3. Design Browser with Technology Schematic 5-3
Figure 5-4. Viewing the Critical Path Schematic 5-4

List of Tables

Table 3-1. Technology Pin Name Attributes 3-18
Table 3-2. Handling Different IO Pad Scenarios 3-19
Table 3-3. Handling Different Clock Buffering Scenarios 3-21
Table 4-1. Deprecated Commands 4-21

List of Tables (cont.)

About This Manual

Precision™ RTL Synthesis is a comprehensive tool suite, providing design capture in the form of VHDL and Verilog entry, advanced register- transfer-level logic synthesis, constraint-based optimization, state-of-the-art timing analysis, schematic viewing and encapsulated place-and-route. This manual describes the synthesis design process using the Precision RTL Synthesis Graphical User Interface (GUI) and provides information on how to perform synthesis tasks and analysis procedures.

Chapter 1
Introducing Precision RTL Synthesis

Precision™ RTL Synthesis is a synthesis platform the maximizes the performance of both existing programmable logic devices (CPLDs and FPGAs) and next-generation, multi-million gate field programmable system-on-chip (FPSoC) devices. Precision RTL Synthesis is a comprehensive tool suite, providing design capture in the form of VHDL and Verilog entry, advanced register- transfer-level logic synthesis, constraint-based optimization, state-of-the-art timing analysis, schematic viewing and encapsulated place-and-route. Precision RTL Synthesis runs on PC platforms using Windows 2000/NT/XP and Linux RedHat; and UNIX Sun and HP platforms. Refer to the Precision Synthesis Installation Guide for detailed information about supported system configurations and requirements.

Precision RTL Synthesis Features

Intuitive User Interface

The graphical Design Bar and the Design Center window guides both novice and expert users alike easily through the synthesis process. A progressive disclosure paradigm presents only valid next steps reducing frustration and confusion. When optimization completes, additional design bars appear to assist you with design analysis and place and route.

Project Manager

The Project Manager offers a framework that supports and promotes a design process based on experimentation. The Project Manager system is based on *projects* which contain multiple *implementations*. Each implementation within a project is a separate workspace in which you can experiment with different synthesis strategies and configurations. Implementations keep track of all of the design settings, input files, and output files that comprise its configuration. The Project Manager facilitates your design efforts by allowing you to create, delete, copy, edit and save implementations. For more information refer to Chapter 4, Managing Projects.

Advanced Synthesis Algorithms

Precision RTL Synthesis includes a suite of unique algorithms called Architecture Signature Extraction (A.S.E.) optimization that automatically focus specific optimizations on areas of the design that are most likely to hinder overall performance, such as finite state machines (FSM), cross-hierarchical paths, or paths with excessive combinational logic. ASE uses an automated, heuristic approach to deliver smaller and faster designs without the need for iterative manual user intervention.

Constraint Driven Synthesis

As FPGAs continue to grow in size and complexity, designers need a synthesis tool that delivers excellent results the first time. Timing constraints, based on the industry standard Synopsys Design Constraint (SDC) format, are all the information Precision needs to deliver correct results without endless iterations.

Missing timing constraints result in incomplete timing analysis and may allow timing errors to go undetected. Precision eliminates this by performing a complete constraint analysis prior to synthesis to insure that designs are fully and accurately constrained. First time success in the lab requires a fully and accurately constrained design during synthesis.

Boundaryless Optimization

Precision's advanced optimization technology breaks down performance limiting design barriers such as register, hierarchy and operator boundaries. A powerful retiming algorithm balances logic across register boundaries; hierarchy optimization minimizes logic between modules and pipelining moves registers into multipliers. Precision understands when and where to employ these algorithms, which can improve circuit performance by up to 70%.

FSM Optimization

Finite state machines are automatically detected and optimized by Precision. Complete analysis is performed on each FSM to locate and eliminate all unnecessary states. A variety of encoding styles are then evaluated to determine the best implementation for the design and target technology. If you choose, you can override the automatic feature and specify a specific state machine encoding. You can even specify a “safe” state machine for radiation environments. Refer to the Precision Synthesis RTL Style Guide for more detailed information on state machine synthesis.

PreciseTime™ Timing Analysis

Precision's PreciseTime™ timing analysis is capable of finding the true critical paths in even the most complicated designs and clocking schemes. PreciseTime employs the concept of clock domains, a combination of timing analysis technologies to correctly handle even the most complex circuits.

Interactive Timing Analysis

A designer's information about device timing should not be limited to a few pre-selected paths in a synthesis report. Interactive timing analysis can instantaneously generate detailed timing reports from any port, pin or instance in the design. Timing queries can be initiated throughout the user interface including selected objects in the schematic viewer.

Register Retiming

Precision Synthesis includes a powerful optimization algorithm called *register retiming* for improving performance in FPGA designs. Retiming allows the optimizer to move registers across combinatorial logic to improve circuit performance. Improvements of up to 50% are not uncommon when using this algorithm. Refer to the topic Register Retiming on page 4-12 of the Precision Synthesis Reference Manual for more information.

Schematic Viewing with Critical Path Fragment Filtering

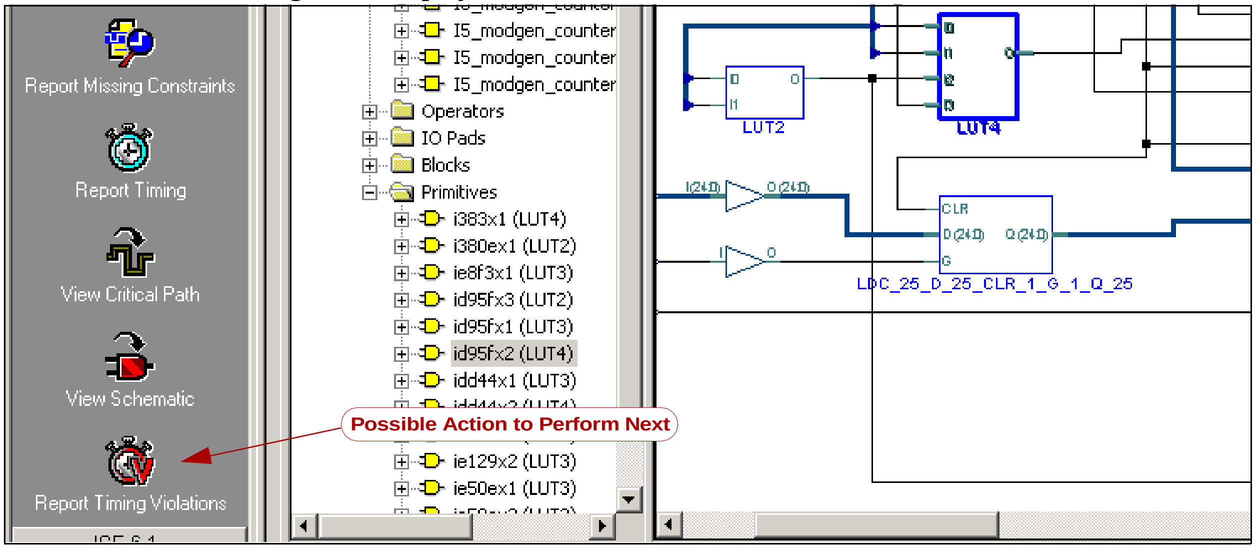
An integrated schematic viewer provides a clear visualization of the synthesis process. High-level RTL schematics help you determine the impact of coding styles while detailed technology schematics show where and how device specific resources such as RAM and ROM are utilized. Patented path viewing and filtering technology displays concise fragments of timing critical logic.

Integrated Creation and Analysis Tools

The following additional standard features allow you to complete the entire synthesis task within Precision RTL Synthesis.

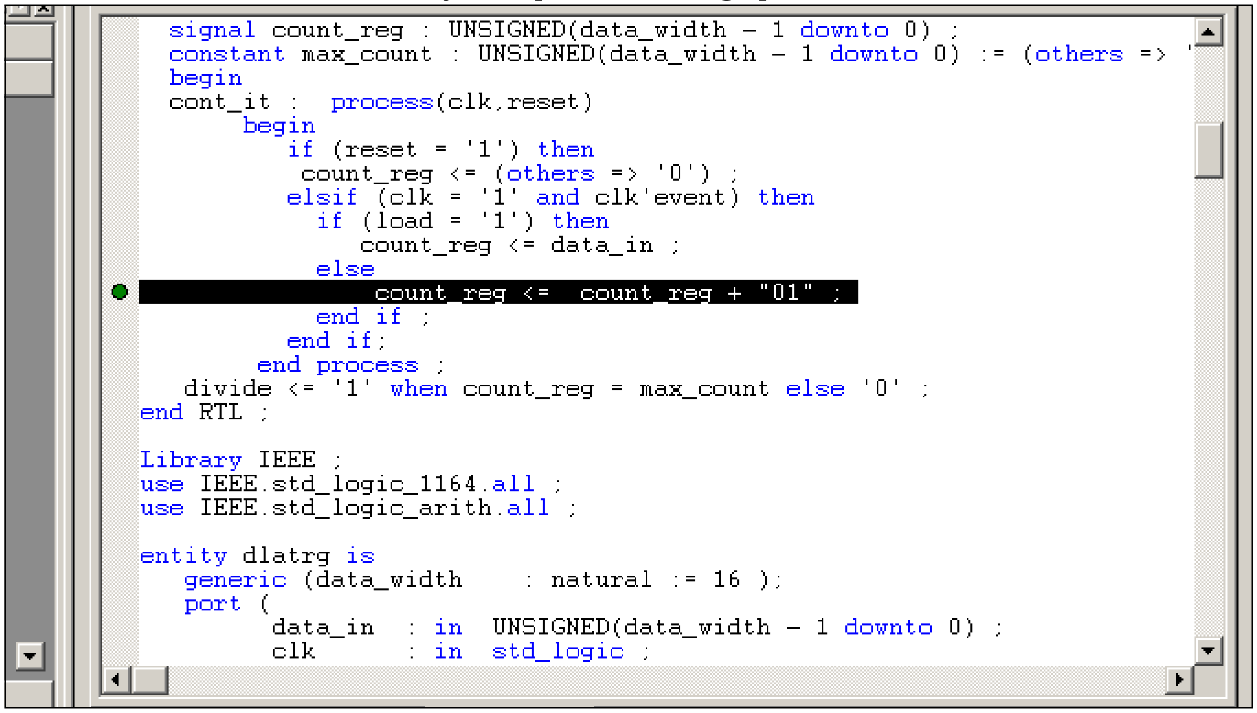
Design Bar

The Design Bar provides you with a visual path through the synthesis flow. Progressive discloser in the Design Bar helps you determine what to do next.



HDLInventor

The HDLInventor is an interactive source code editor in Precision RTL Synthesis. You can double click on errors, warnings, and information (red, green, and blue dots) in the Transcript window or click on the name of your input file to bring up the HDLInventor.

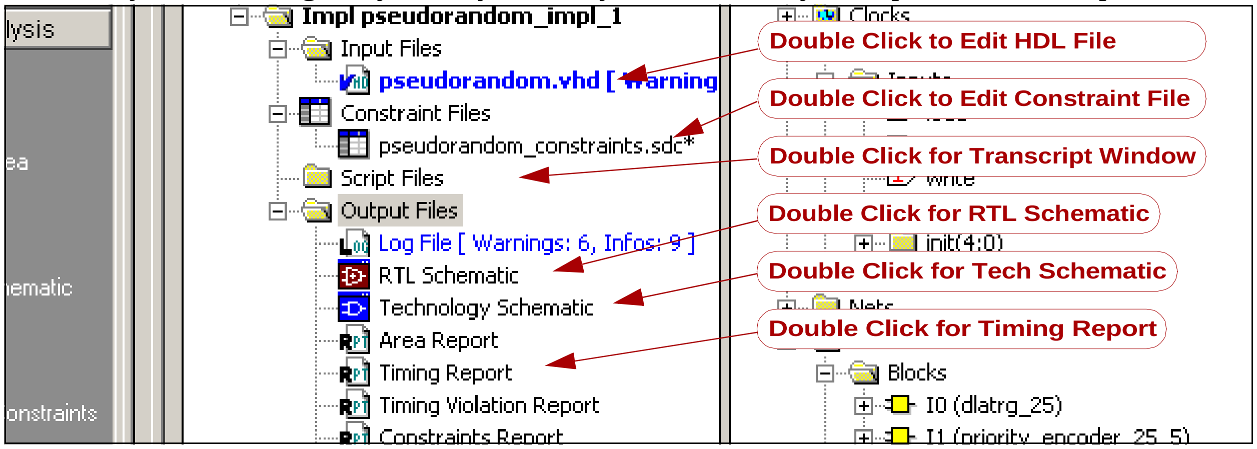


```
    signal count_reg : UNSIGNED(data_width - 1 downto 0) ;
    constant max_count : UNSIGNED(data_width - 1 downto 0) := (others => '
    begin
    cont_it :  process(clk,reset)
         begin
            if (reset = '1') then
             count_reg <= (others => '0') ;
            elsif (clk = '1' and clk'event) then
              if (load = '1') then
                count_reg <= data_in ;
              else
                  count_reg <=  count_reg + "01" ;
              end if ;
            end if;
          end process ;
     divide <= '1' when count_reg = max_count else '0' ;
  end RTL ;

  Library IEEE ;
  use IEEE.std_logic_1164.all ;
  use IEEE.std_logic_arith.all ;

  entity dlatrg is
     generic (data_width     : natural := 16 );
     port (
           data_in   : in  UNSIGNED(data_width - 1 downto 0) ;
           clk       : in  std_logic ;
```

The HDLInventor interactively highlights syntax and synthesis construct errors found during the Compile process. You can make your edits in this window and, if required, insert template(s) of HDL code that you frequently use.

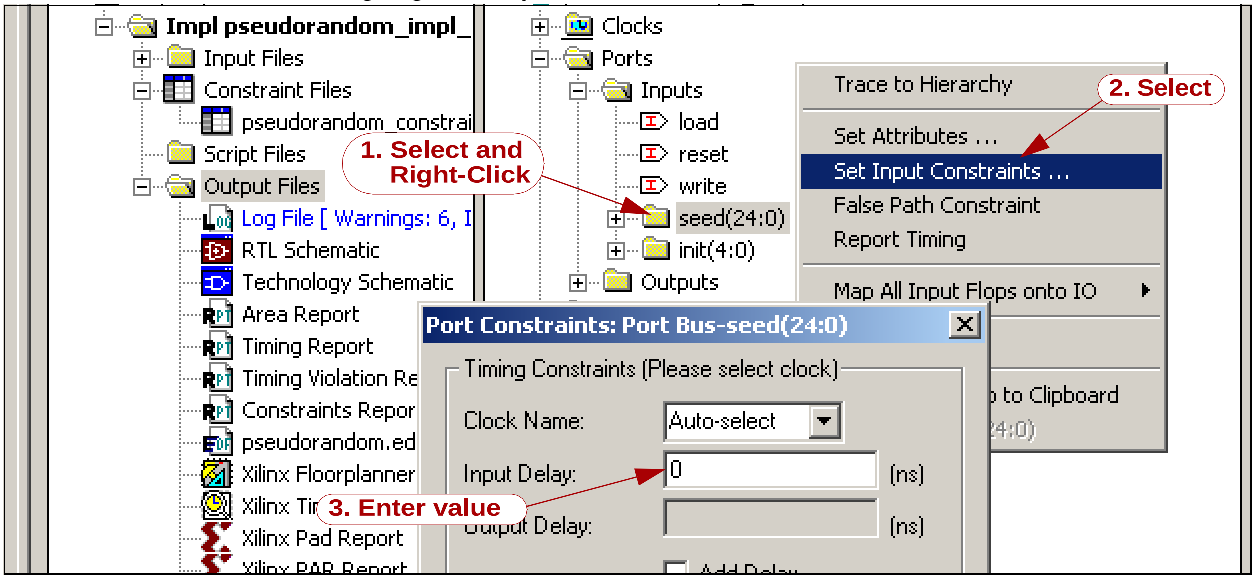
Project Browser

The Project Browser gives you fully visibility and access to your input files and output files.


Impl pseudorandom_impl_1
Input Files
pseudorandom.vhd [Warning
Constraint Files
pseudorandom_constraints.sdc*
Script Files
Output Files
Log File [Warnings: 6, Infos: 9]
RTL Schematic
Technology Schematic
Area Report
Timing Report
Timing Violation Report
Constraints Report
Double Click to Edit HDL File
Double Click to Edit Constraint File
Double Click for Transcript Window
Double Click for RTL Schematic
Double Click for Tech Schematic
Double Click for Timing Report
Clocks
init(4:0)
Blocks
I0 (dlatrg_25)
I1 (priority_encoder_25_5)

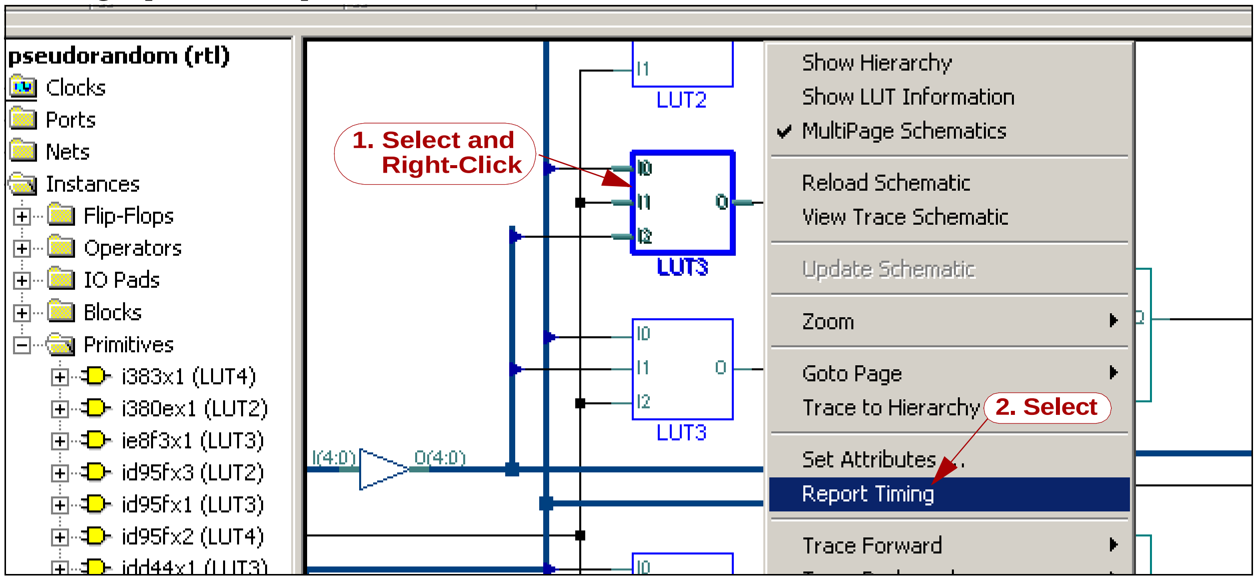
Design Browser

The Design Browser is a graphical representation of the in-memory design database and allows you to traverse through the design hierarchy to observe and set constraints such as Input Delay on ports and “Don’t Touch” attributes on modules. You can also use the Design Browser to flatten or preserve the design hierarchy. Objects selected and highlighted may also be highlighted in the schematic viewer. Furthermore, if the selected object initiates cross probing, then that line of code is highlighted in your HDL source code.



Schematic Viewing

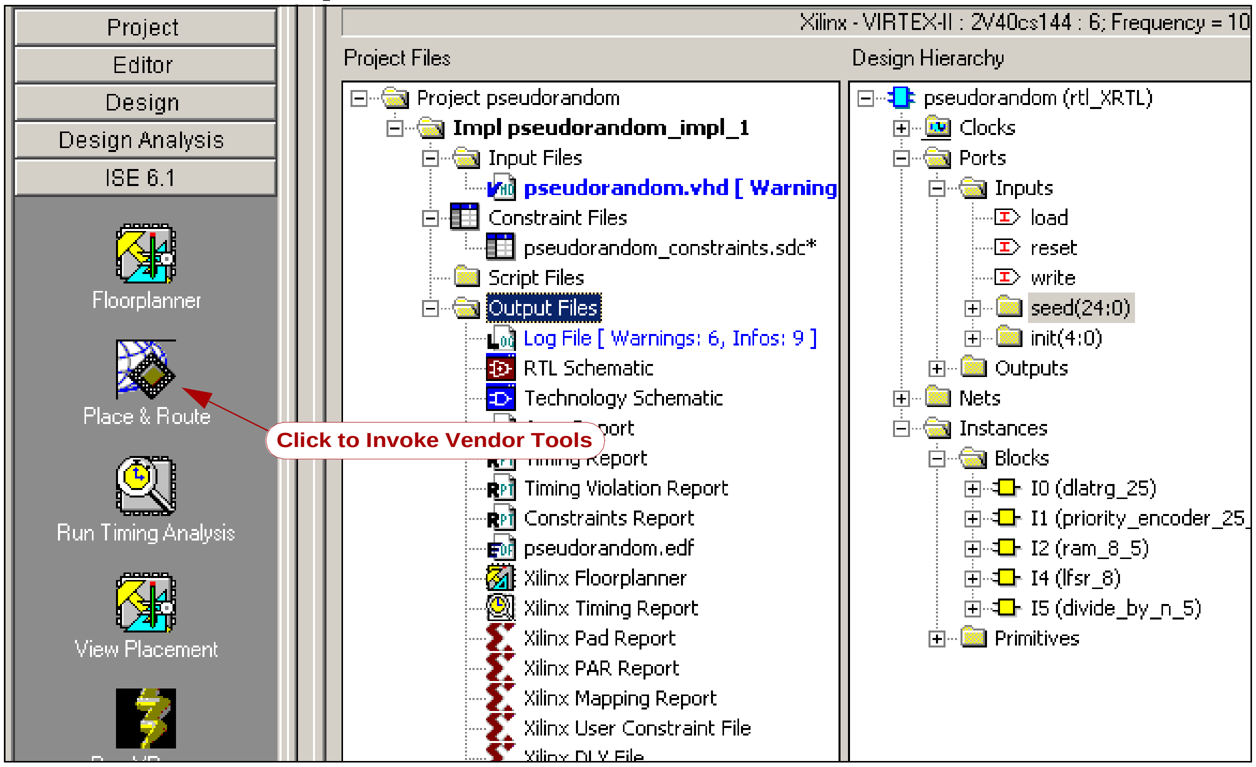
Schematic Viewing is integrated with the advanced debug and analysis environment. As shown below, the advanced features allow you to select an object in a path and generate an instant timing report on that path.



The schematic viewer also allows you, for example, to: (1) Cross-probe between HDL source code, RTL schematic, and Technology schematic. This correlation allows for easy debugging. In addition, you can cross probe a schematic generated in HDL Designer with a schematic generated in Precision RTL Synthesis. (2) You can view the whole critical path in one window, even if the path traverses multiple levels of hierarchy. (3) You can view fanout and fanin cones of logic from a selected net or instance. (4) When the critical path viewer is in query mode, detailed timing popup information is displayed for the objects in the critical path. (5) The schematic viewer search utility allows you to search for instance, net, and port; and lists these items for you in a window.

P&RIntegrator

P&RIntegrator automatically sub-invokes vendor backend place and route tools from within Precision RTL Synthesis. Currently the following environments are supported: Actel Designer, Altera MAX+PLUS II, Altera Quartus II, Lattice ispLEVER, Lattice ispLEVER ORCA and Xilinx ISE. The vendor's backend tools then create a binary program file which is used to program FPGA and CPLD devices.Refer to Chapter 4 Running Physical Implementation for more information on a particular Vendor environment.



```
Project pseudorandom
  Impl pseudorandom_impl_1
    Input Files
      pseudorandom.vhd [ Warning
    Constraint Files
      pseudorandom_constraints.sdc*
    Script Files
    Output Files
      Log File [ Warnings: 6, Infos: 9 ]
      RTL Schematic
      Technology Schematic
      ...port
      Timing Report
      Timing Violation Report
      Constraints Report
      pseudorandom.edf
      Xilinx Floorplanner
      Xilinx Timing Report
      Xilinx Pad Report
      Xilinx PAR Report
      Xilinx Mapping Report
      Xilinx User Constraint File
      Xilinx DLY File
```

```
pseudorandom (rtl_XRTL)
  Clocks
  Ports
    Inputs
      load
      reset
      write
      seed(24:0)
      init(4:0)
    Outputs
  Nets
  Instances
    Blocks
      I0 (dlatrg_25)
      I1 (priority_encoder_25
      I2 (ram_8_5)
      I4 (lfsr_8)
      I5 (divide_by_n_5)
    Primitives
```

Documentation Available Online

Context-Sensitive Help

Precision RTL Synthesis has context-sensitive help throughout the GUI. You can press **F1** to open a context-sensitive help or press the HELP button. The GUI window must be selected first to be in current focus when using **F1**. **Note: F1** does not work on UNIX.

Online help is available in Windows 95 format. You can view frames of help text and graphics by moving your cursor to the Help pulldown menu and selecting:

Help > Help Contents...

You can expand the Table of Contents and select from a variety of topics or do a full index search for keywords.

Product Manuals

All Precision RTL Synthesis product manuals are available for on-screen viewing and printing with the Adobe Acrobat Reader after Precision RTL Synthesis and the Adobe Acrobat Reader are installed from the Precision RTL Synthesis CD-ROM.

You can view the manuals by selecting the following pulldown menu from the Main menu:

Help > Open Manuals Bookcase

The PDF manuals and the Manuals Bookcase also contain HyperText links that guide you to related vendor documentation on the Web, provided your web browser is operational and properly configured.

Note

To ensure that the product manuals come up full size, you should verify/set the Acrobat Default Zoom preference to "Actual".

File > Preferences > General... Magnification Default Zoom: Actual

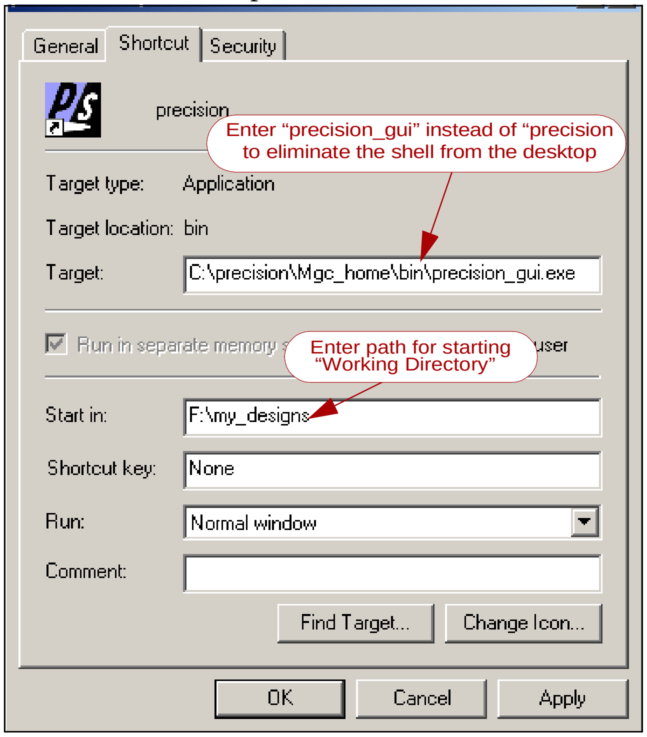
Chapter 2
Setting Up a Design and Compiling

Invoking Precision RTL Synthesis

You can run Precision RTL Synthesis from the Graphical User Interface (GUI) or from a Shell command line. Typically, you will use the GUI while you are learning to use the tool and for setting up your initial synthesis runs. Once your design is setup and the initial constraints are specified, you can easily generate a Project File and a Master Constraint File. You will typically do the remaining synthesis runs by opening the Project File and making minor adjustments to the Master Constraint File.

Invoking the Precision GUI

You invoke the Precision RTL Synthesis GUI by entering the `precision` command from a Windows shell or a Unix shell. In a Windows environment, you can optionally create a Shortcut on your Desktop and set the Shortcut Properties similar to that shown in the following figure.


General
Shortcut
Security
Enter “precision_gui” instead of “precision to eliminate the shell from the desktop
Enter path for starting “Working Directory”
Find Target...
Change Icon...
OK
Cancel
Apply

precision

Target type: Application

Target location: bin

Target: C:\precision\Mgc_home\bin\precision_gui.exe

[✓] Run in separate memory s... user

Start in: F:\my_designs

Shortcut key: None

Run: Normal window

Comment:

Invoking Precision from a Shell

You can invoke Precision in non-GUI mode by using the command `precision -shell.` In this mode you can source Tcl scripts or you can interactively enter commands from the shell prompt. A typical command line entry might be the following where a Tcl command file is specified to set constraints and guide the tool through the synthesis process:

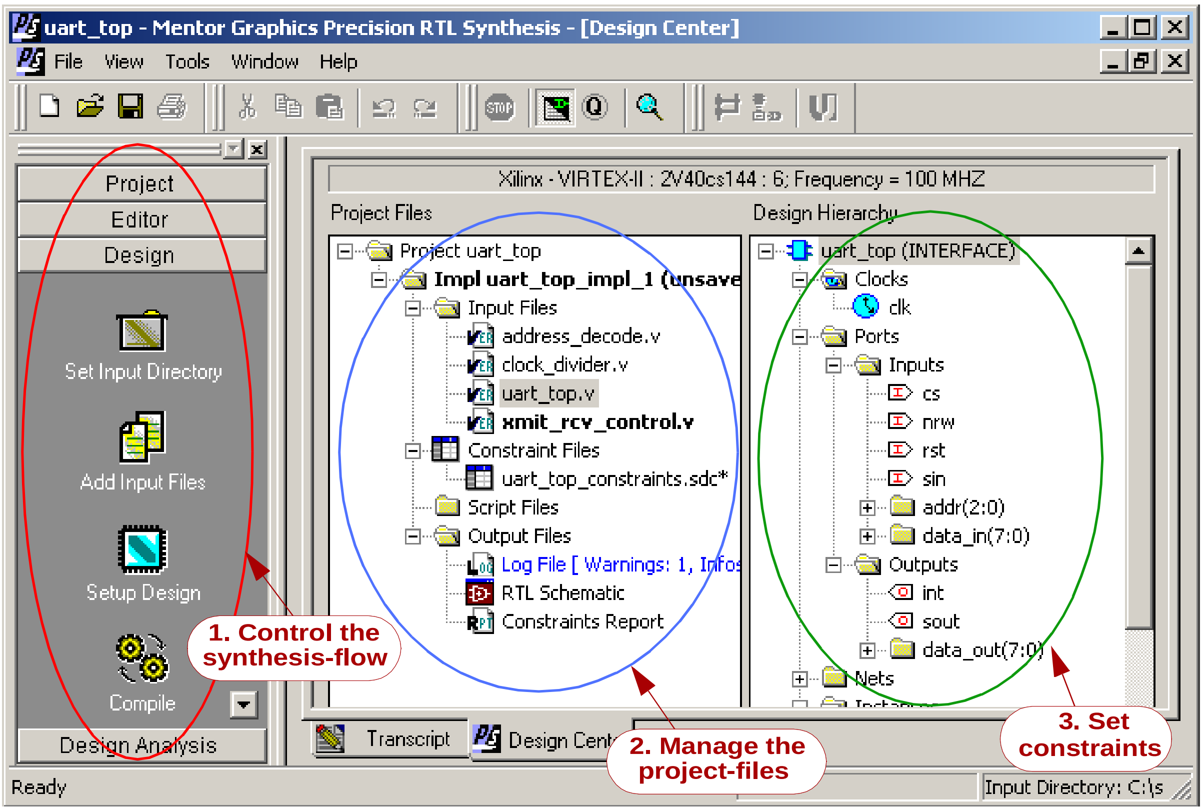
```
% precision -shell -file dofile.tcl
```

You can also export your script settings. For more information, see Exporting your Settings. The `precision` command and its optional switches is fully documented in the section titled Commands in the Precision Synthesis Reference Manual.

Setting up the Design Environment

As shown Figure 2-1, the Main window has three primary control areas. The Design Bar (on the left) controls the synthesis flow. You will manage your project files from the Project pane (center) and apply constraints to design objects in the Design Hierarchy pane (right). The Design Bar provides a visual indicator of the sequential steps in the synthesis flow. Using progressive disclosure, the icons in the Design Bar change depending on where you are in the flow and reveal the possible next steps.

Figure 2-1. The Design Center Window



The following list shows some hints for using the Precision GUI:

- **If you don't know what to do, try right-clicking on a folder, file, or object.** The Design Center window is right-click driven. Right-clicking on the folders and files in the Project Files pane will pop up a menu of commands specifically for the selected object. Also, you can set constraints by right-clicking on design objects in the Design Hierarchy pane. Thus, you can drive the entire flow from the Design Center window.

- **View messages by double-clicking on the Log File.** Doubling clicking on the Log File brings up the *Transcript* window (and its associated Interactive Command Line pane). You can scroll up and down the transcript to read messages and review the history of the current session. Double-clicking on an error message in the transcript brings up the associated HDL source file for review and editing.

  Precision commands executed from the GUI are recorded in the transcript. These commands are highlighted in blue. The transcript is saved in Precision log files (`precision.log`). A session log records all commands executed during an entire Precision session, and is saved in the project directory. In addition, an implementation log is saved inside the active implementation directory which records only the commands executed while that implementation is active.

Note If the results directory was set using the set_results_dir command, only the session log file is created, and it is saved in the results directory.

- **View Schematics/Reports by double-clicking on Schematic/Report Files.** Doubling clicking on a Schematic File or a Report File in the Design Center window brings up the the content of that file in a separate window.
- **Constraints that are Set in the GUI are Saved to an SDC File.** The synthesis process is "constraint driven" and your objective is to create a Master Constraint File that can be read as part of the Input File List. The constraint file format is Synopsys Design Constraints (SDC). One of the easiest ways to create an initial constraint file is to manually set constraints on design objects in the Design Hierarchy window. As you do so, the constraints are automatically saved to a generated `.sdc` file and placed in your *implementation* directory. For future synthesis runs, you can include this generated constraint file as one of your input files.

The Project Directory and the Results Directory

Generally speaking, a *project directory* and a *results directory* are user specified locations where Precision saves output files during a particular session. A project directory is created by the Precision's Project Manager when you create a new project. On the other hand, if you want to work without using the Project Manager, you must set the results directory location by calling the set_results_dir command. In this case, all output files go into the results directory.

The structure and organization of each type of output directory is different and reflects purpose of the directory. The purpose of the project directory is support the organizational functions of Precision's Project Manager. Therefore, the project directory is hierarchical in structure. The top-level is the project level, and it contain multiple sub-directories, each of which is dedicated to a different implementation of your design. For detailed information about the file structure of a project directory, refer to Creating a Project on page 4-3.

The purpose of the results directory is simply a repository for single implementation usage of Precision Synthesis. This type of usage is typically carried out by batch scripts or from the command line interface. Because this usage model deals with only a single implementation, it does not require the organizational structure provided by the Project Manager. Therefore, the file structure is flat. All output files are stored in the results directory without hierarchy.

When working with the project, synthesis design work is not automatically saved to the implementation directory but is held in a temp directory until you save your work. For more information, see The Precision Temp Directory.

Setting the Project Directory

You specify the location of a project directory when you first create the project. Once the project is created, the location is implicitly set each time you reopen the project (because the project file resides inside the project directory). You cannot open a project if a results directory is currently set.

Setting the Results Directory

You set the location of the results directory by calling the set_results_dir command. This command is not available if a project is already open. Within scripts, it must be the first command called in order for Precision to run without using the Project Manager. When the results directory is set, no Project Manager commands are available until you call the close_results_dir command.

Setting the Input Directory

The input directory is the location where Precision will look for input files by default. The input directory can be independent of the project directory. When you ask Precision to access an input file, you can give it the full pathname of the file or a partial pathname. If you specify a partial pathname, that pathname must be relative to the input directory. In other words, the full path to the file must be equivalent to the combination of the input directory path and the partial file path (<input_dir>/<partial_path>).

You can change the input directory at any time, either from the GUI or by calling the set_input_dir command. Resetting the input directory while you are working in Precision can be a useful strategy. For example, you can create multiple input directories that contains variations of the same set of input files. Then you can easily switch back and forth as you experiment. For this strategy to work, all of the input files that have the same name must be added to the project using a relative pathname (relative to the input directory).

Setting the Technology

In order for Precision RTL Synthesis to map your design to a specific technology, you must specify a target technology. Precision RTL Synthesis provides a number of CPLD and FPGA libraries from major vendors. These libraries includes the technology-specific cell definitions and custom operator implementations. They also contain global library defaults such as route tables and fanout constraints.

To specify the technology and global options, you should follow these steps:

1. Click on the Setup Design icon in the Design Bar (left side of the GUI)

   This action displays the Setup Technology dialog box containing all technology libraries that are included with Precision RTL Synthesis.

2. Select the target technology by clicking on the vendor and technology name

   This action adds a set of default technology settings that you can modify.

3. Verify and or modify the default technology settings

4. Optionally set a global design frequency

   This value serves as a starting point for clock constraints on all clocks. After the Compile step, you will be able to select each clock in the Design Hierarchy pane and adjust the constraints accordingly.

5. Optionally set the default Input Delay and Output Delay

   Input Delay refers to the delay consumed outside the design before a data signal reaches the input port of your design. Output Delay is the delay required outside the design in order to properly clock the device being driven by your design. These optional default values serve as a starting point for setting I/O constraints. After the Compile step, you will be able to select each I/O port in the Design Hierarchy pane and adjust the constraints accordingly.

6. Apply the Setup Technology dialog box

   This action executes a series of `setup_design` commands that inform Precision RTL Synthesis about the technology and the global options you entered. Precision RTL Synthesis does not actually load the synthesis library into memory until the command to Compile is executed.

After you specify the technology and verify the options, you can add your design files to the project.

Adding Input Files to the Project

Precision RTL Synthesis does not read or reference pre-compiled HDL libraries, packages, or designs from disk. Instead, the library and package source files and the design source files are read directly into memory where Precision builds an EDIF-like in-memory database. The design source files may reside in any location and may even reside in more than one location.

To add your input files to the project from the GUI, follow these steps:

1. Click on the Add Input Files icon in the Design Bar

   This action opens the Add Input Files dialog box.

2. Navigate to and select a source file(s), then click Open.

Since Precision analyzes all of the files in the input file list together, the order that you add the files is usually not important. Precision RTL Synthesis will auto-detect the top level. If your design files reference a standard library or package, such as an IEEE library, then Precision RTL Synthesis will automatically open and load that library or package file which is located in the `<precision install directory>/pkgs/techdata/vhdl` directory.

If you include Xilinx Coregen files in this list, Precision will read the coregen block and mark it as a dont_touch block (so Precision will not optimize it but will consider it for fanout and timing analysis. For more information, refer to "Including Xilinx Coregen-Generated Modules" in the Precision Synthesis Reference Manual.

3. (optional) Adjust the file type by right-clicking on the file name and selecting Properties.

   Precision detects the file type based on the file extension. Valid values are `vhdl (.vhd/.vho)`, `verilog (.v,.vo)`, `edif(.edif, .edn, .edf)`, `syn`, `tcl`, `xnf`, `xdb`, and `sdf`. If a file does not use a valid extension, then you can use this form to specify the file type.

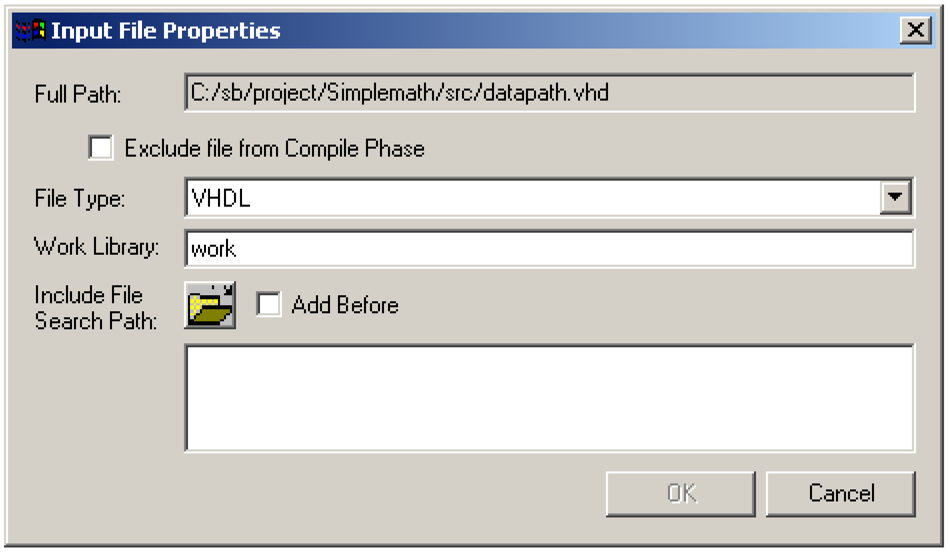
4. Apply the Add Input Files setting by clicking on the Apply button.

   This dialog box may execute a series of `add_input_file`, `move_input_file` and `set_input_file` commands (depending on how you add the files).

Setting Input File Properties

Once you include a file in the Input File List, you can right-click on the file in the Project Files pane and specify the Properties of the file. Figure 2-2 illustrates the dialog box options and the text following the Figure explains the options in detail.

Figure 2-2. Specifying Input File Properties



Full Path: C:/sb/project/Simplemath/src/datapath.vhd
File Type: VHDL
Work Library: work

File Type

Allows you to specify the file type for file names that don’t have the proper extension. If this option is not used and a valid extension exists, then the file type will be automatically detected.

Passing Files from Input to Output

You can use this option when you wish to add one or more files to the Input File List, but exclude them from the Compile phase. Files marked with the exclude property are copied into the implementation directory after the synthesis phase is complete. Also, files of an unknown type are automatically marked as “Exclude” and copied to the implementation directory. This mechanism is handy for passing a file, such as a place and route control file, around the synthesis process and onto the physical implementation tools.

Work Library

Specifies the name of the work library for compiling the content of the VHDL file. If not specified, then the work library name `work` is assumed.

Include File Search Path

Specifies additional directories that are pre-pended to the global search path that is specified from the pulldown menu **Tools** > **Set Options** > **Input**.

Searching for Verilog ‘include’ Files

If a Verilog file is being added and additional files are referenced via the ‘include’ directive, then the search for the include file is conducted in the following order:

1. The directory of the file that specifies the include directive
2. The directories that are specified for this file in the Input File Properties dialog box.
3. The directories that are specified for global searches in the pulldown menu **Tools > Set Options... Input**

Assume, for example, that the file being added is located in the directory `F:/design/src` and this search path is set to the following:

```
{“C:/my_include_files” “F:/more_include_files”}
```

During the compile operation for this file, Precision RTL Synthesis first searches for any specified include files starting in directory `F:/design/src`, then `C:/my_include_files` then directory `F:/more_include_files.` If the file is not found, the directories specified in the pulldown menu **Tools > Set Options... Input** are searched. As soon as the file is found, the search ends.

Searching for VHDL files

When the file being added is compiled and it references a VHDL library or a package that has not yet been compiled, a search is conducted for this package file by that library or package name so it can be compiled first. Assume, for example, that this input file contains the following the clause:

```
use lib.my_package.all;
```

When the file is compiled, Precision RTL Synthesis looks in the library ***work*** to see if `my_package` has been compiled. If not, a search begins in the directory where this input file resides, then the search directories that are specified for this file. The search continues in the global directories that are specified in the pulldown menu **Tools > Set Options... Input** and finally the directory `<precision install directory>/pkgs/techdata/vhdl` is searched. As soon as the file is found, the search ends, the package file is compiled and the specified input file is compiled. If the package file is not found, Precision RTL Synthesis issues an error message.

VHDL Considerations

Before you use Precision to perform VHDL synthesis, you should already have a good understanding of the syntax and semantics of VHDL. If your design references custom libraries and packages, then you must include these package source files for reading before your design files are read. The methods for doing this are fully discussed in the Precision RTL Synthesis Style Guide.

The following list shows some common issues when loading VHDL designs into memory:

- **Read source (not compiled) VHDL files** Precision RTL Synthesis can not read compiled HDL databases from any simulator or synthesis tool. You must read the VHDL source code, including libraries and packages.

- **Auto-Loading IEEE Libraries** Precision RTL Synthesis automatically loads the IEEE standard `std_logic_1164` and `numeric_std` packages. It also loads (if necessary) the Synopsys `std_logic_arith`, `std_logic_unsigned`, and `std_logic_signed` packages in the IEEE library. You do not have to include these packages in the Input File List in Precision RTL Synthesis.

- **Auto-Top Detection** In most simulators, you must compile the VHDL code using a bottom-up order. Precision RTL Synthesis should auto-detect the top-level of the design and automatically determine the compile order of the input VHDL files.

- **Library/Use Clauses** for referring to technology cells. Because synthesis tools use a different scheme to locate technology cell instantiations (compared to simulators), you should pragma out any VHDL Library and Use clauses for these cells using `translate_on` and `translate_off`. Precision RTL Synthesis will still locate these cell instantiations because Precision RTL Synthesis loads the technology library prior to compiling the VHDL source code.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;
-- pragma translate_off
USE a42mx.components.all;
-- pragma translate_on
```

- **Case Sensitivity** Although VHDL is not case sensitive, in order to handle mixed language designs (VHDL and Verilog), Precision RTL Synthesis considers VHDL to be case sensitive. Since most cell and port names in the technology libraries are uppercase, the case of instantiated technology cells must also be uppercase.

Verilog Considerations

Before you use Precision to perform Verilog synthesis, you should already have a good understanding of the syntax and semantics of Verilog. Using Verilog synthesis also requires knowledge of the guidelines presented in the Precision RTL Synthesis Style Guide.

- **Auto-Top Detection** Precision RTL Synthesis auto-detects the top-level of the design and automatically determined the compile order of the input Verilog files.
- **Include Files** If additional Verilog files are referenced via the 'include' directive in an input file, then the file is searches for in a pre-defiled order. This search order was described in the previous topic Include File Search Path.
- **Full Case** When a case statement is used in your Verilog design, the Full Case switch tells Precision that all conditions of the case statement are specified. If a default assignment is not used, for example, then this option prevents the implementation of extraneous latches. This switch is set from the pulldown menu **Tools** > **Set Options** > **Input** > **Verilog**.
- **Parallel Case** When using a CASE statement in your Verilog design and the case conditions are mutually exclusive, a multiplexer is often the preferred implementation (instead of priority encoding a state machine). You should select Parallel Case to direct Precision RTL Synthesis to treat case conditions as mutually exclusive and implement a multiplexer logic structure. This switch is set from the pulldown menu **Tools** > **Set Options** > **Input** > **Verilog**.
- **Verilog 2001 Support** Precision Synthesis provides support for Verilog 2001 constructs. Refer the to Precision RTL Synthesis Style Guide for details.

EDIF Considerations

Precision RTL Synthesis only supports the reading of EDIF files that were generated by Mentor Graphics synthesis tools or FPGA place/route tools. If Precision determines that the EDIF file is from Xilinx CoreGen, Precision RTL Synthesis automatically applies a `dont_touch` attribute to the instantiated CoreGen block.

Although Precision RTL Synthesis supports the EDIF format, you may not be able to read EDIF from other tools (e.g. schematic capture) because the cell sets and pin names may not match between the two tools.

XDB Considerations

Precision RTL Synthesis can only read and properly interpret an `.xdb` file that was first generated by Precision RTL Synthesis.

SDC considerations

By default, Precision RTL Synthesis saves all timing constraints and attributes generated from the GUI in a file named `<design_name>.sdc` located in the current implementation directory. If you added any additional constraints or attributes via the command line, you must manually add these commands to the Master Constraint File if you want them included in the next synthesis run.

Setting Synthesis Options

You direct and refine the behavior of the synthesis process by selecting options from the pulldown menu **Tools** > **Set Options...** The options are described online if you click on the HELP button at the bottom of each dialog box. The following list describes some of the common options that you should verify the default settings.

- Set State Machine encoding.

  You can allow Precision Synthesis to automatically select a Finite State Machine encoding style for you (the default) or you can specify a specific encoding style. Precision Synthesis also supports the ability to create a Safe State Machine. A safe state machine is a state machine in which a transition will always be to a valid state. Refer to the Precision Synthesis RTL Style Guide for more information on how to set these options.

- Setting Output Options

  Precision saves the output netlists for the implementation tools along with area and timing reports in the current *implementation* directory. These file are also listed in the Project File pane of the Design Center window.

  By default, the base name (leaf name) of the generated output file is taken from the name of the last HDL input file to be read. You can change this name by specifying a new name from the pulldown menu **Tools** > **Set Options** > **Output** > **Output File Base Name**.

- Generating Output Files in Different Formats

  If you need Precision RTL Synthesis to generate additional netlists (e.g. an HDL netlist for simulation), you need to set this additional output netlist option from the pulldown menu **Tools > Set Options > Output**. A separate file is generated for each type that is selected (EDIF, Verilog, VHDL).

Compiling the Design

Once you have specified the target technology and your source HDL files, Precision RTL Synthesis can compile the design into a technology-independent implementation. To compile your design, click on the Compile icon in the Design Bar.

If the Compile icon does not appear, verify that you on using the Design Center tab and that you have specified a target technology in the design setup.

If you get any errors or warnings, you can edit the source file by double-clicking on either the message in the Transcript window or on the file name in the Design Center.

An in-memory design database is created by reading one or more source files into memory. Precision converts any of the initial input formats into an intermediate representation which permits the tool to efficiently optimize designs. Within the views, Precision RTL Synthesis uses the following naming convention for different types of logic:

- ix* = instance name for boolean gate
- reg_*signal* = flip-flop with the q output driving *signal*
- ? or "unknown" = power/ground net
- lat_*signal* = latch with the q output driving *signal*
- modgen_* = cell used to implement an operator

Evaluating the Results

By examining the transcript and viewing the RTL Schematic, you can determine whether Precision has created the generic gate-level implementation that you expect. You can analyze both the warnings in the transcript and the quality of the initial results. Even though the output of the `compile` command is technology independent, the quality of the compiled design is a precursor to the quality of the final technology design.

Reviewing Warnings and Errors in the Transcript

The following list shows some typical warning messages and their causes:

- Messages about overwriting designs—

  ```
  Warning: Overwriting netlist work.comb (INTERFACE)
  ```

  Typically, you expect these warnings when you read a new block into the overall design and that block matches an empty or existing cell. However, you should investigate these messages to make sure you have not unintentionally overwritten an in-memory block.

- Messages about the sensitivity list—

  ```
  “arbiter.v”,line 873: Warning, interleave should be present in the
  'always' condition.
  ```

  Because sensitivity lists are ignored during synthesis, Precision builds combinational logic as if such missing signals are present in the sensitivity list. This can lead to simulation mismatches.

- Messages about unused nets—

  ```
  “arbiter.v”, line 441: Warning, extranet is never used
  ```

  These warnings indicate local nets that were declared but never used within the design. Such nets are dissolved during constant propagation. Often the indicated nets lead to expected unused output ports from instantiated blocks. Consider removing such unused internal nets. However, you should analyze the nets to ensure that a design error does not exist.

- Messages about unused ports—

  ```
  “arbiter.v”, line 441: Warning, input extraport is never used
  ```

  Warnings about unused ports are more specific cases of the preceding warning about unused nets. You should consider removing any ports that are declared but never used within the Verilog code.

- Messages about latches—

  ```
  “arbiter.v”,line 638: Warning, start_dir_ras_tmp is not always
  assigned. latches could be needed.
  ```

  Warnings about latches may be given when Precision inserts transparent latches for combinational processes. This circuitry is inferred from code (the lack of `else` statements, typically) that does not account for all the possible conditions within a conditional construct. Precision infers that it needs to store the states for these conditions, which results in unexpected circuitry.

- Messages about parallel case—

  ```
  “arbiter.v”,line 1028: Warning, case choices are not mutually
  exclusive, parallel_case pragma may induce simulation mismatch.
  ```

  Warnings about parallel case may be given when Precision encounters case structures that you have specified as having conditions that are not to be prioritized during synthesis. This can lead to simulation mismatches.

- Messages about black boxes—

  ```
  Warning: cell xcve.DLL (INTERFACE) marked black box.
  ```

  Make sure that the black boxes represent empty modules for incremental synthesis or vendor cell instantiation (use the Area Report to examine results). Otherwise, these messages indicate that Precision could not find certain cells in the design.

  If the design is read in, but you see unexpected black boxes on technology cell instantiations, verify that the cell and port names in the HDL code match the cell and port names in the technology library in Precision RTL Synthesis. All port names must be included. You can display this information using the `get_lib_cell` and `get_lib_ports` commands. The case of these names must also match (even for VHDL which is not case sensitive). Precision RTL Synthesis must maintain case sensitivity on all objects to support mixed-HDL designs. If the instantiation uses positional association to map the net to the cell port (compared to including the port name and net name in the instantiation), you should verify that Precision RTL Synthesis is making the same assumptions about pin ordering.

Evaluating Quality of Initial Results

After you have fixed any errors and have a satisfactory explained of the warnings produced during the compile run, you should examine the quality of the compiled design. Although you can not generate a timing report on the technology-independent design, examining the compiled design will give you an early opportunity to address any Quality of Results issues.

To help you evaluate the design, the following features are available.

- The session transcript in the Transcript Window (double-click on the Log File)
- The RTL schematic (double-click on the RTL Schematic file)
- Cross-probing features

To evaluate the quality of the initial results, double-click on the Area Report in the Output File list and examine the following issues:

- Check for latches (in the transcript during the compiling of each cell)

  In FPGA designs, synthesis tools infer latches due to incomplete signal assignments. Unintended latches have two negative affects on the design. One, they increase the size of the design because these sequential cells prevent the optimizer from combining boolean logic on either side of the latch. Two, unintended latches block timing analysis.

- Verify the number of inferred flip-flops in each block (in the Area Report)

  Since synthesis tools only optimize boolean logic, it is very important that you verify that each block is inferring the approximate number of flip-flops that you expected. Once a flip-flop or latch is part of the design, the synthesis tool will not optimize it away as long as it affects the design.

  During optimization, Precision RTL Synthesis will only add/remove flip-flops in your design in the following cases:

  - Register replication for timing or DRC fixes
  - Register removal due to D-input being driven by a constant (these situations will be reported in the transcript).

- Verify the number and type of inferred operators (in the Design Browser in the Operators folders)

  Since the critical path within most designs propagates through operators, it is very important that you verify you are inferring the correct number/size of operators.

- Verify the number of inferred memories and counters (in transcript after the pre_optimize operation)

  Precision RTL Synthesis utilizes special mapping algorithms to target counters and memories from your HDL code. If Precision does not infer the correct number of RAMs, ROMs or Counters, refer to the Precision RTL Synthesis Style Guide to verify that the HDL code matches the coding style that Precision expects. For RAMs that do not infer properly, try to cut and paste one of the examples. For counters, verify that the reset condition is either all zeros or all ones. The Precision RTL Synthesis counter operator only has an asynchronous reset or set (no asynchronous load). You can work around this issue by muxing the data and reset value.

If your compiled design meets expectations, you can continue by setting constraints on your design as described in the next chapter. You may also want to save the present state of the implementation. Simply right-click on the implementation in the Project Files pane of the Design Center window, then choose **Save Implementation** from the popup menu.

If your compiled design does not meet your expectations, you will probably need to make changes to your HDL source code and follow the recommendations in the Precision RTL Synthesis Style Guide.

Setting Timing and Design Constraints

After the design has been compiled, you can set design constraints on the design so that Precision can address the critical timing paths in your design. At a minimum, you should set the global frequency. Since Precision automatically detects all of the clocks in the design after compile, it is highly recommended that you specify clock constraints on each detected clock. This method will insure that Precision focuses on the critical timing paths in the design.

Constraints can be set from the using any combination of the following methods:

- Adding an SDC File to the Input File List

  This file is called the Master Constraint File, but the filename may be any name you specify as long as it have an `.sdc` extension. As previously described, if you don't have a Master Constraint File, you can use Precision RTL Synthesis to generate one for you on the first synthesis run. Technically, you can have more than one `.sdc` file in the Input File List, however, it is a common practice to maintain only one Master Constraint File.

- Entering Constraints from the Precision GUI

  You can manually add constraints to the in-memory design objects with the GUI. A record of the changes are saved in a generated `.sdc` file that is placed in the current implementation directory.

- Entering Constraints from the Command Line

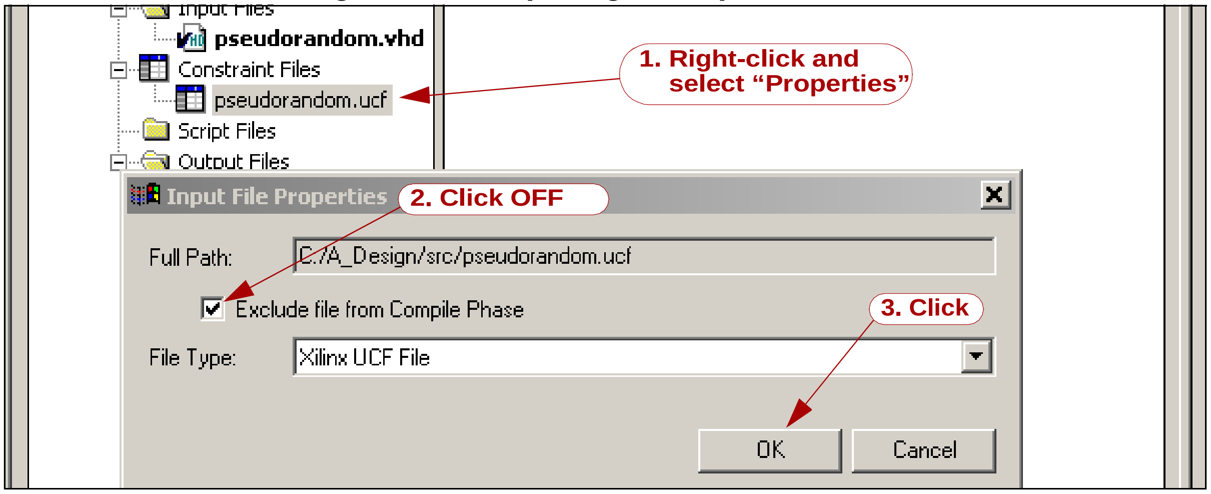
  You can use the Interactive Command Line Shell to add constraints to the in-memory design. You may do this while experimenting with different constraint sets. You must realize, however, that any constraints entered this way will not be saved to the generated .sdc file. If you want to save one of these constraints for the next synthesis run, you must manually type the constraint directly into the Master Constraint File.

- Adding a Xilinx UCF File to the Input File List

  Precision Synthesis has the ability to read the timing constraints from a Xilinx UCF file and apply those constraints to the in-memory design. This is handy, for example, if you have a design that was generated by another synthesis tool and you have timing constraints that are expressed in Xilinx UCF format. You can include only one UCF file in the Input File List and to read the UCF timing constraints, you must remove the "Excluded" attribute from the file as shown in the figure below. The procedure is to right-click on the file from the Design Browser and select "Properties". Click off the

"Exclude file from Compile Phase", then click OK. When the Design is compiled, the UCF timing constraints will be applied to the in-memory design.

Figure 2-3. Preparing the Input UCF File



Full Path: C:/A_Design/src/pseudorandom.ucf

☑ Exclude file from Compile Phase

File Type: Xilinx UCF File

You may include other SDC constraint files with the UCF input file, however, you should make sure that the constraints in different files don't conflict. If a conflict occurs, the constraints in the last file read (last file in the list) prevail.

- Specifying Attributes or Pragmas in the HDL Source

  Constraints that rarely change can be added directly to the HDL source files by using VHDL attributes or Verilog pragmas. An example might be a physical pin number assignment that is set by the system specification or a `dont_touch` attribute that is set on a block of IP (Intellectual Property). Refer to the chapter titled Attributes in the Precision Synthesis Reference Manual for details and examples.

For more information on setting a specific timing or design constraint, refer to "Setting Constraints and Synthesizing" on page 3-1.

Synthesizing the Design

During synthesis, Precision performs the following high-level tasks:

1. Implement operators
2. Perform intelligent auto-dissolve on smaller hierarchical blocks
3. Perform initial quick optimization of each block
4. Determine the blocks that do not meet timing and perform additional optimizations where necessary.

5. Write out netlists and reports to the implementation directory

Evaluating the Results

After the synthesis run is complete, you should evaluate the quality of the results. Although the natural inclination is to look at the maximum frequency, this value can be mis-leading until you verify that your clock distribution is correct. To get a complete view of the synthesis quality, you should check the areas described below.

Analyzing the Cell Usage

As part of the evaluation of your synthesis results, you should use the Area Report and Schematic Viewer to verify that key resources were utilized to their fullest potential. The following list shows some areas that you should check:

- Empty IP blocks.
- Clock Buffers.
- IO cell usage
- Memories.
- Unexpectedly removed ports or instances

Analyzing the Timing Results

After you run Synthesize, you can view a timing report. Timing reports display timing paths through hierarchical boundaries. The following list provides the typical results of *initially* analyzing critical paths and the actions that you should take based upon the results:

- Positive slack time

  If the design produces a positive slack time for critical paths, you do not have to perform any further performance optimization steps and you can proceed to Place and Route.

- Small negative slack time

  If your design produces a negative slack time for critical paths that is about 10% or less than the overall cycle time, you should perform a quick check of the hints in the following two sections and proceed to Place and Route. Since the initial synthesis pass uses pre-layout timing estimates, it is possible that small negative violates can be fixed in placement during Place/Route.

- Large negative slack time

  If your design produces a negative slack time for critical paths that is 10% or greater than the overall cycle time, you need to verify you are reporting a *true critical path* (not a multicycle, false path, or invalid clock interaction) and then analyze the elements within the critical paths. Using the hints in the following two subsections, you need to identify the HDL code that is causing the critical path and possibly make adjustments in the HDL code.

  If your design is still not meeting performance requirements, you can either loosen your timing constraints or re-synthesis using a faster speed grade for the device. If you are within 10% of your timing goals, you may want to run the design through the vendor implementation tools to get more accurate post-place/route timing values. Precision RTL Synthesis calculates delays using a conservative pre-layout estimate. If you are close to meeting your timing constraint (according to Precision's estimate), you may have actually met the timing constraint according to the accurate post-layout value produced by the physical implementation tools.

To obtain the fastest design, consider doing the following:

- *Don't over-constrain paths.* Precision RTL Synthesis will spend a lot of CPU time trying to obtain an impossible goal while greatly increasing the size of the design.
- *Check for asynchronous signals exist on the critical path.* Asynchronous signals (such as set or reset) cause negative slack because they typically arrive well before the data to the synchronous device is available for clocking.

  Examine your timing report for the following statement with the *Source* or *Des*t fields in order to determine the existence of asynchronous signals:

  ```
  Combinational path through sequential primitive.
  Probably an asynchronous set/reset.
  ```

  Since asynchronous signals are usually connected to external ports, you can correct the negative slack by setting the input constraint on the port to -1 clock cycle (e.g., -100ns). Then the asynchronous signal is a static value 1 clock cycle before the input to the synchronous device is available.
- *Multi-cycle paths exist on the critical path*. You can detect multi-cycle paths in timing reports by observing a negative slack value that is greater than the clock cycle. You will also notice that the path starts with and ends with a synchronous device.
- *Examine the type of logic (boolean or operators).*
- Effective use of operators
- Check fanout

- *Consider using Retiming.* Precision includes a highly effective register balancing algorithm. This algorithm is disabled by default in RTL synthesis and enabled by default in physical synthesis.

  To enable retiming in the entire design:

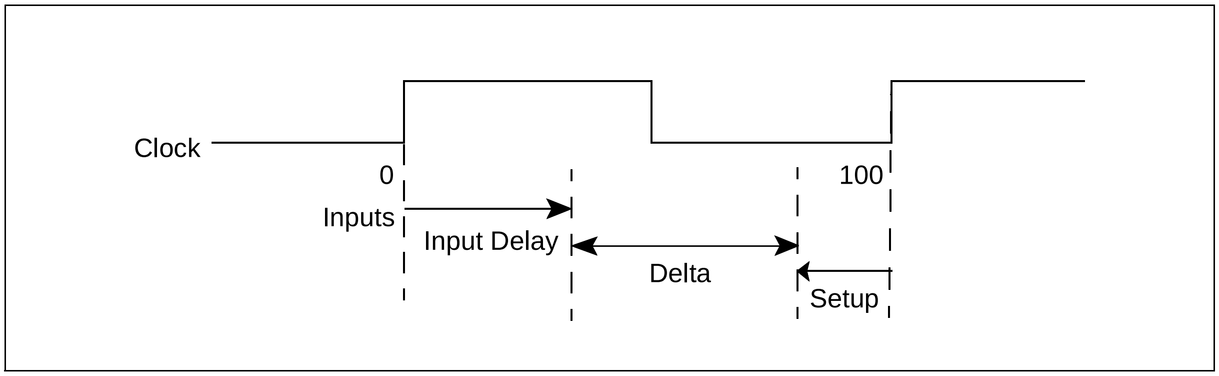
  ```
  setup_design -retiming=true
  ```

  Retiming will affect the observability of internal registers. If observability of a particular internal register is an issue then set a "don't_retime" attribute on those specific registers. For example:

  ```
  set_attribute -name DONT_RETIME -value TRUE -instance U1/reg_pipe(*)
  ```

- *Incorrect constraints*. Typically, setting the input/output constraints incorrectly may over-constrain the design such that Precision reports a negative slack that can never be corrected. Setting constraints incorrectly is usually the result of not understanding the relationship between Input Delay and setup time, as shown in Figure 2-1. Basically, the Input Delay is defined with respect to the first clock edge, and the setup time is defined with respect to the last edge of *one* clock cycle. The difference (delta) between these two times is the delay available for the circuit. When this delta is zero or when the times overlap, Precision reports an un-correctable negative slack.

Figure 2-1. The Constraint Relationship



- *Fine tune constraints*. Refine the constraints to meet the “real world” conditions of the complete design.

- *Flatten blocks*. If the area results are acceptable, begin to flatten hierarchical blocks in order to improve timing results on subsequent synthesis runs.

- *Turn off resource sharing.* If you have a large operator in the critical path that is resource shared, try turning off resource sharing. Click off the radio button **Tools** > **Set Options... Input** > **Resource Sharing** and click **Synthesize** again. The additional operators may increase the area, but the eliminated mux circuitry may solve the timing problem.

- *Return to the HDL source design and re-implement*. You might have to re-design a critical path in the HDL source in order to create a parallel (faster) design or you may have to increase the latency of the path by a cycle or two.
- *Switch speed grades*. Last, check with your vendor to determine if there is a faster version of the chip available that represents a higher performance technology.

Determining the Next Step

If your design meets the specified performance requirements, then move on to Place & Route.

If your design is still not meeting performance requirements, you can either loosen the timing constraints or select the Register Retiming option and re-run Synthesis. If you are within 10% of your timing goals, you may want to run the design through the physical implementation tools to get accurate post-place/route timing values. Precision RTL Synthesis calculates delays using a conservative pre-layout estimate. If you are close to meeting your timing constraints (according to Precision's estimate), you may have actually met the timing constraints according to the accurate post-layout value produced by the physical implementation tools.

Performing Initial Place and Route

Precision RTL Synthesis has a built in Place & Route environment that allows you to move seamlessly from synthesis to the physical implementation of your design. Once your design is synthesized, icons for the Vendor's implementation tools appear in the Precision Design Bar. The current implementation directory becomes the *project directory* for the Vendor tools and you can proceed to Place and Route.

Like the synthesis process, the place and route process is controlled by two primary commands. Synthesis options are set by the `setup_design` command and the flow is executed with the `synthesize` command. In a parallel way, place and route options are set by the `setup_place_and_route` command and the flow is executed with the `place_and_route` command.

The Precision GUI interface to the `setup_place_and_route` command appears after you select the target technology. The pulldown menu **Tools** > **Set Options** > **(Vendor Environment Name)** becomes available.

After Precision output files are written to the implementation directory, the Vendor Tools icons are revealed in the Design Bar. You can invoke the automated Place & Route flow, or you can invoke the Vendor's project browser and manually step through the implementation process. You can also view Vendor-generated reports and invoke Vendor tools such as timing analyzers and power analyzers.

Chapter 3
Setting Constraints and Synthesizing

When you Compile a design, a technology-independent database is created in memory. The next step involves setting constraints and mapping the design to technology-specific cells. By default, Precision RTL Synthesis maps the design to the fastest possible implementation that meets your timing constraints. In order to accomplish this, you must, at a minimum, specify timing on the automatically determined clock sources. With this information, Precision performs static timing analysis to determine the location of the critical timing paths. Although Precision will map your design to the target technology without any timing information, Precision will produce the best results with this information.

Managing a Master Constraint File

Typically, you will make more than one synthesis run before you achieve the best results. During this process, you guide Precision toward the ideal solution by setting constraints. The constraints will be timing constraints, mapping constraints, and constraints that control the structure of the implemented design. The constraints you specify first are usually estimates that you'll want to change and refine on future synthesis runs.

Creating a Master Constraint File

The recommended methodology is to create a Master Constraint File (`.sdc` file) and add this file as the last item in the Input File List. You can create this file with a common text editor or use Precision to generate one for you on the first synthesis run. You then add this generated file to the Input File List as your Master Constraint File. During the Compile phase on a future synthesis run, the in-memory design is created from the HDL source files, then the Master Constraint File is read and the constraints are applied to in-memory design objects.

Maintaining a Master Constraint File

Typically, you will add and change constraints on future synthesis runs. Constraint changes that that are made to the RTL in-memory design from the GUI are logged to a newly generated constraint file. This `.sdc` file is saved to the current implementation directory. As previously stated, on the first run you can use this file as a template for your Master Constraint File. On subsequent runs, you should regard this file only as a record of constraint changes. Some of the constraints will be a record of your experimentation while other constraints are changes you'll want to keep by adding them to your Master Constraint File. The methodology is to use a text editor to manually "copy and paste" the new or modified constraints to the Master Constraint File.

For the purpose maintaining the integrity of the `.sdc` file, the only time the Precision GUI will save constraint settings in the `.sdc` file is when you apply the constraints to the in-memory RTL design. If you add or edit constraints on the gate-level in-memory design, your changes are applied to the design, but not saved in the `.sdc` file. For your convenience, the Design Hierarchy pane of the Design Center window always displays the RTL design. To view a gate-level design after synthesizing, double-click on the Technology Schematic file in the Project Files pane of the Design Center window.

Likewise, when you manually enter constraints into a `.sdc` file, be sure to use the **-design** option that is available on all constraint related commands. That option takes one of two arguments, **rtl** or **gatelevel**, that specifies the design file to which the constraint will be applied. The default value is **rtl**.

Continually managing the Master Constraint File is a primary task and you should be diligent to maintain the accuracy and content of the file. To properly edit the file, you'll need to learn the syntax of the SDC commands. These commands are fully documented in the section titled Commands in the Precision Synthesis Reference Manual. In addition, examples of SDC commands will be presented in this Chapter.

Setting Timing Constraints

If you do not specify timing constraints, Precision will map your design to technology cells but will not perform timing optimization. When you specify timing constraints, you only need to specify constraints on the top-level ports and Precision will characterize the top-level timing constraints through the lower-level blocks of hierarchy.

With timing constraints, Precision can make informed decisions about how to implement logic using technology cells. After the design is mapped, PreciseTime preforms static timing analysis to identify the paths that do not meet timing. If there are violations, Precision performs additional synthesis on these timing paths to increase the speed. This usually results in more hardware on the critical paths due to an increase in parallel structures, but only the critical paths are affected, not the whole design.

Setting Global Timing Constraints

As a starting point, you can set the default global timing during the Design Setup step. After you Compile the design, you can add, refine and finalize the constraints before the Synthesize step.

You do this by selecting design objects in either the Design Browser or Schematic Viewer and use the right mouse button to display the timing constraint dialog box. Constraints that you apply to the an in-memory RTL design from the Precision GUI are automatically saved in the Precision generated `.sdc` file. The constraints are later reapplied to subsequent compile/synthesis iterations.

> ***NOTE:*** For the purpose maintaining the integrity of the `.sdc` file, only constraint settings applied to the in-memory RTL design are saved to the `.sdc` file. If you add or edit constraints on the gate-level in-memory design, your changes are applied to the design, but not saved in the `.sdc` file. Based on this policy, the Design Hierarchy pane of the Design Center window always displays the RTL design.
>
> Additionally, all of the constraint related commands now provide the **`-design`** option. This option takes one of two arguments, **`rtl`** or **`gatelevel`**, that specifies the design file to which the constraint will be applied. The default value is **`rtl`**.

Although setting constraints can be as simple as specifying a target design frequency or as powerful as indicating several multi-cycle paths, at a minimum, you should define the clock period on every detected clock pin. Precision displays these clocks in the right side of the Design Center.

By default, Precision does not check timing paths from inputs to registers or registers to outputs unless you specify timing constraints on the top-level input and output ports.

> ***NOTE:*** As defined by the industry standard SDC format, since timing constraints must be set relative to defined clocks, you must specify your clocks prior to applying any other timing constraints.

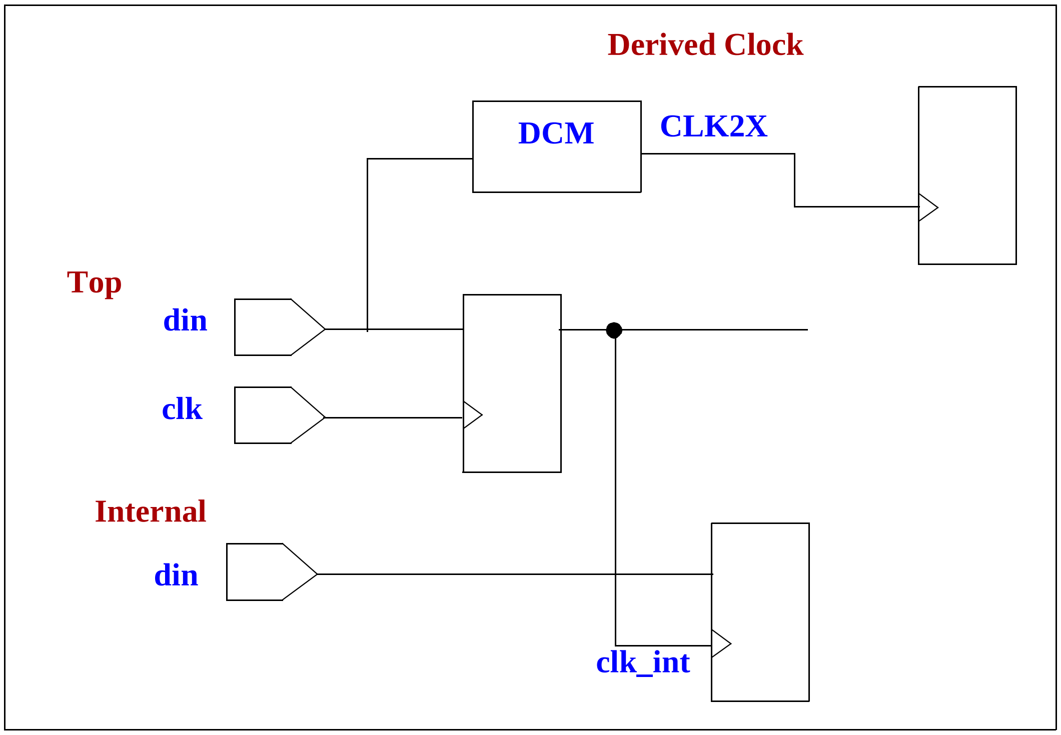
Clocks Overview

A clock is an abstract design object created within your design database to allow you to constrain your design for timing. Ideal clocks are not part of your design, but clocks can be associated with signals and ports in your design. Defining a clock defines the period and edge positions of a recurring waveform. When a clock is associated with a port or signal, it is referred to as a “real clock.” If no such assignments are made, the clock is considered a virtual clock. Virtual clocks are useful for describing timing synchronized by external circuitry.

In addition to user defined clocks, the system will automatically create clocks for you to model the behavior of clocking components (e.g. Xilinx DCM components or Altera DLL components). The clocks are referred to as derived clocks, since their properties are determined based on the input clock properties and the function of the clocking component. Derived clocks can be used everywhere a defined clock is used, but they cannot be modified by the user.

They will appear in the clock folder of the design browser along with the user defined clocks. See the following figure for an example of clocks.

Figure 3-1. Clocks Example



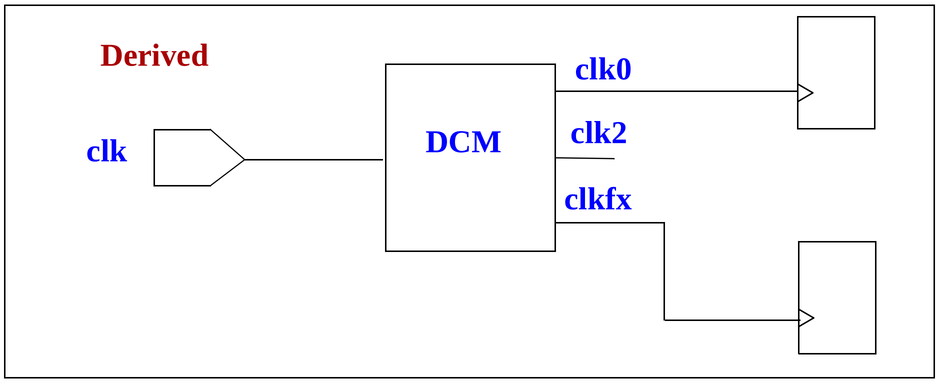
Clocks are used as reference objects for constraint commands. The `set_input_delay` and `set_output delay` commands are defined with respect to a defined clock. Clocks can also be used in exception commands to specify a group of endpoints. For example, the command `set_false_path -from clk1` will remove timing violations which begin at all registers clocked by that particular clock. The `report_timing` command also accepts clock names, which are used similarly.

Note: If a clock and a port in the design have the exact same name, an ambiguity occurs. In most cases, the Precision Synthesis tool will resolve this ambiguity in favor of the clock name. You may need to pick unique names for your clocks to allow some operations.

Signals that require clocks are automatically found by the system. Before clocks are defined, clocks are displayed in the design browser with no properties. This is done to help you find the points that must be constrained with clock definitions. Clocks will be traced back to primary design ports when possible. If clocks are gated, sometimes this is impossible, so in this case, internal signals will appear in the clock folder.

Derived Clocks

The Precision tool automatically generates derived clocks for clocks that feed a Xilinx Digital Clock Manager (DCM). When a clock feeds a DCM, the DCM has clock output such as clk0, clk2, clkfx, which clock other registers. Precision automatically calculates the delay value of clocks generated by the DCM. See the following figure for an example of the derived clock.



Within the Precision tool, you can specify constraints on clock inputs to a DCM. If you specify global clock constraints on an input to the DCM, for example, Precision RTL Synthesis will propagate the timing constraints to the outputs of the DCM, including transferring the proper constraints to the clock multiplier and divider ports.

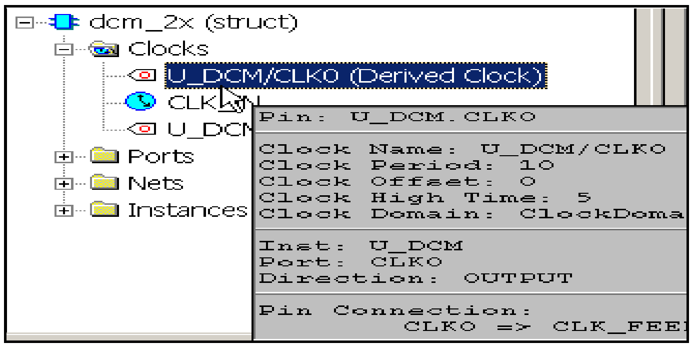
The Precision tool uses a Common Timing Engine to construct a “graph” of all clocks that indicates the parent/children clocks of every clock. Every time any clock is updated with a lower clock frequency, the parent and child clocks will be updated as well, depending on their relationship. For example: Suppose you have a design that has DCM o/p clocks CLK0 and CLK2X, whose parent clock is CLKIN. If Precision RTL reports CLKIN to be run at 100Mhz, then CLK0 will also be run at 100Mhz, and CLK2X will be run at 200Mhz.

Displaying Derived Clocks within the Graphical User Interface

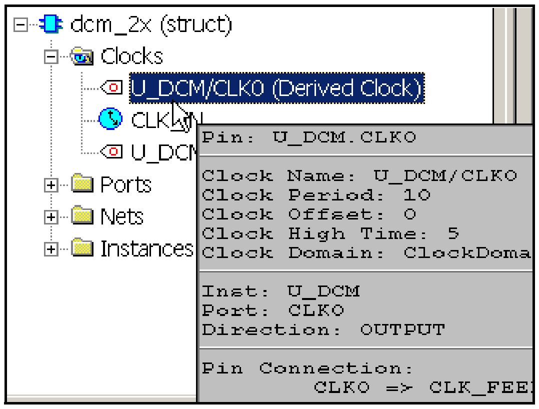
The tool displays the design in the Design Input windows. Once you load the design, you set Architectural Constraints and compile the design. Once the design is compiled, the generated clocks display in the design window. Clocks that were derived from the DCM are identified by the label “Derived Clock.” To display information associated with the derived clock, select the clock and hover the cursor over it. Information associated with the clock is display in a pop-up window. For example, a derived clock might be named “U_DCM_CLK0.”

Note: You cannot edit the constraints on a derived clock.

For a sample of how a design and a derived clock appear in the graphical user interface, see the following figure:



```
dcm_2x (struct)
  Clocks
    U_DCM/CLK0 (Derived Clock)
    CLK_IN
    U_DCM
  Ports
  Nets
  Instances

Pin: U_DCM.CLK0

Clock Name: U_DCM/CLK0
Clock Period: 10
Clock Offset: 0
Clock High Time: 5
Clock Domain: ClockDoma

Inst: U_DCM
Port: CLK0
Direction: OUTPUT

Pin Connection:
          CLK0 => CLK_FEE
```



```
dcm_2x (struct)
  Clocks
    U_DCM/CLK0 (Derived Clock)
    CLK_IN
    U_DCM
  Ports
  Nets
  Instances

Pin: U_DCM.CLK0

Clock Name: U_DCM/CLK0
Clock Period: 10
Clock Offset: 0
Clock High Time: 5
Clock Domain: ClockDoma

Inst: U_DCM
Port: CLK0
Direction: OUTPUT

Pin Connection:
          CLK0 => CLK_FEE
```

Commands for Derived Clocks

The following commands relate to derived clocks:

- find_clock -derived
- set_false_path
- set_multicycle_path
- remove_propagated_clock
- report_timing

For more information, see the text associated with these commands in the *Precision Synthesis Reference Manual*.

Report Timing and Clock Names

It is a general practice within the tool for clocks to share the same name as the object on which they are set. If a port and clock have the same and you enter the report_timing command, then the tool applies the command with regard to the clock and not on the physical item associated with the clock. There are extra steps you need to take when a clock and a port have the same name so that the tool can provide the correct output information using the report_ timing command or when you use the set_false_path command. For example, let’s assume that you have a design where both the port and clock are named CLK1. In this case, you would need to do the following:

1. You should rename the clock using the create_clock command and also specify the desired clock period.
2. You should constrain the clock port like a data port with the appropriate input delay using the set_input_delay command. Entering “report_timing -from clk1” should report

timing paths that begin at the primary input “clk1” and end at the constrained (data) pins on flip-flops. It is likely that paths won’t fit this criteria. The Precision tool would only report this path if clk1 had a set_input_delay constraint defined on it or if the max_delay constraint was defined with clk1 in the “from” list.

Example:

If you have both a clock and a port named CLK1 and you want to set a false path on the PORT named CLK1, then you need to issue a create clock command to rename the clock and to specify the period setting for this clock.

```
   create_clock clk1 -name myclk -period 10,
```

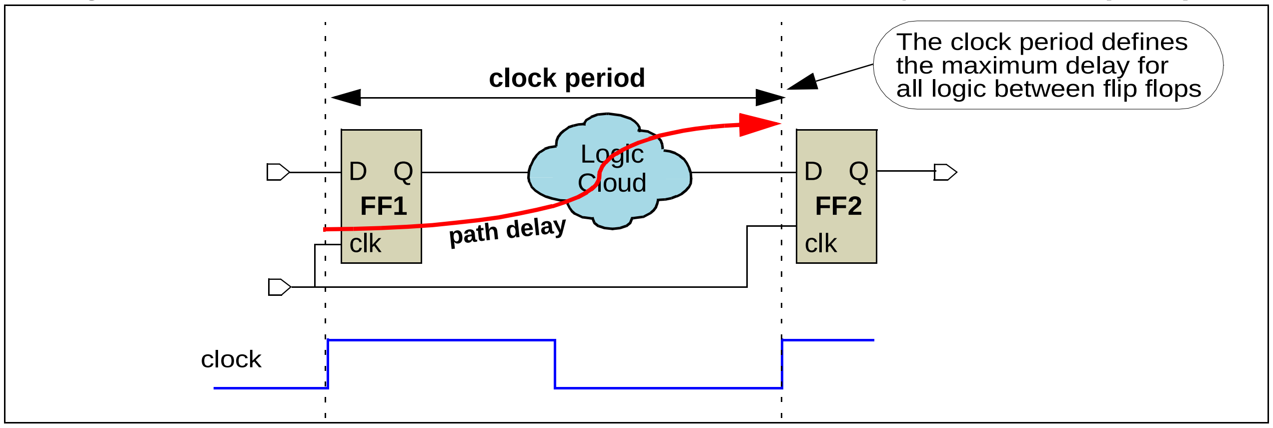
This will cause any register or input clock referenced by “clk1” to reference the source of the timing path for the clock you have renamed “myclk”.

Specifying Clock Constraints

Precision uses clock constraints to define the timing between registers. Precision displays the auto-detected clock sources in the *clocks* folder in the Design Hierarchy pane.

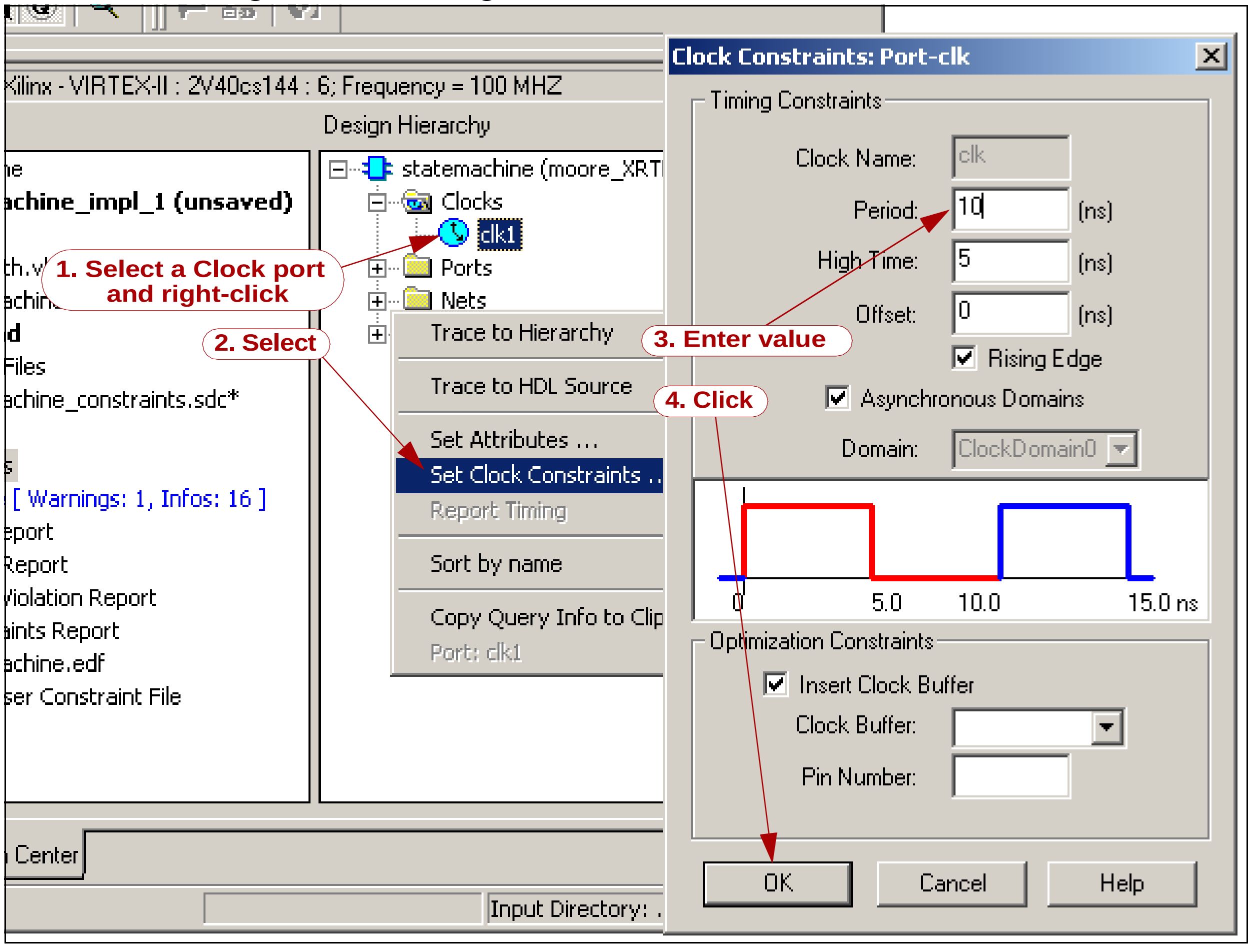
When a clock is not defined, all combinational logic between the registers driven by the undefined clock is ignored during timing optimization. Thus, if you do not define any clocks, Precision will not run timing analysis or attempt to increase the speed of your design. As soon as you define a clock, you have effectively constrained the combinational logic between all registers to one clock period, as shown in Figure 3-2.

Figure 3-2. The Clock Period Defines the Maximum Delay Between Flip Flops


clock period
The clock period defines the maximum delay for all logic between flip flops
D
Q
FF1
clk
Logic Cloud
path delay
D
Q
FF2
clk
clock

To constrain a clock using the GUI, follow the steps in Figure 3-3.

Figure 3-3. Setting the Clock Constraints from the GUI



Clock Constraints: Port-clk

Timing Constraints

Clock Name: clk

Period: 10 (ns)

High Time: 5 (ns)

Offset: 0 (ns)

[✓] Rising Edge

[✓] Asynchronous Domains

Domain: ClockDomain0

Optimization Constraints

[✓] Insert Clock Buffer

Clock Buffer:

Pin Number:

When you set a clock constraint from the GUI, the default Clock Name is the same as the port name. You can change the Clock Name later if you wish by changing the `-name` switch in the `create_clock` command in the Master Constraint File.

If you want to set the same constraint on all clocks, you can select just the Clock folder (which selects all objects in the contents of the folder). Within the folder, you can select more than one port by pressing the Control key when you select addition ports.

(optional) Specify the clock domain. By default, Precision assumes that registers that are triggered by different clocks do not interact. If you have multiple clocks in your design that contain logic that may interact, you should define these clocks in the same domain. Clock domains do not exist within standard SDC. This characteristic is included in Precision for efficiency, and serves as more intuitive way to describe purely asynchronous clocks (clocks that do not interact).

The Clock Constraint dialog box adds a `create_clock` command to the generated `.sdc` file and applies the constraint to the specified input port or instance pin in the in-memory design. The basic syntax of the `create_clock` constraint is:

```
create_clock -design <view_type> -name <clock_name> -period <value>
                                   [-waveform <edge_list>] <port_list>
```

Using Figure 3-2 as an example, the logic between FF1 and FF2 is constrained to one clock period. If the clock period is 10ns, then the Logic Cloud has approximately 10ns minus the clk to D timing arc of FF1 and the setup constraint of FF2 to meet timing. Therefore, to accurately define the clock, you must apply the following constraint:

```
> create_clock -design rlt -name sysclk -period 10 clk
```

If you are entering the `create_clock` command directly into the Master Constraint File, you can use the `find_clocks` command to quickly define all clocks in the design (before you have time to determine whether they may be driven by other logic (e.g. clock dividers). For example:

```
> create_clock -design rlt -name sysclk -period 10 [find_clocks]
```

Handling Multiple Clocks

Multiple clocks are common in today's circuit designs. The proper handling of multiple clocks is a major feature of Precision RTL Synthesis. For clocks of different frequency, Precision finds the clock period where the clocks most closely approach each other. The timing report shows the clock period where this occurs when reporting violations. For example, worst case timing might be found at the 3rd rising edge of the source clock and the 6th rising edge of the destination clock. In general, the greatest common divisor of the 2 clock periods is usually the clock period used for analysis.

Clocks without small integer common multiples are probably not related and any paths between these clocks are probably false paths. If there is no guaranteed timing relationship between them, then Precision considers the clocks to be asynchronous (or in different clock domains). Generally, this occurs when clocks are driven by different oscillators. Two clocks in different domains may have exactly the same frequency, may have identical latencies, etc., but they cannot be analyzed, because there is no common "zero-point" to which these edges and latencies are measured. By default, Precision places these clocks in different domains.

Since SDC does not support the concept of clock domains, Precision RTL Synthesis has added a switch to the `create_clock` command to model this behavior. See the following syntax:

```
create_clock -period <value> -name <value> -domain <domain_name>
```

The `<domain_name>` argument can be any name that you want. If two clocks have the same domain name, they are modeled as synchronous. If no `<domain_name>` is specified, the domain name "ClockDomain0" is used.

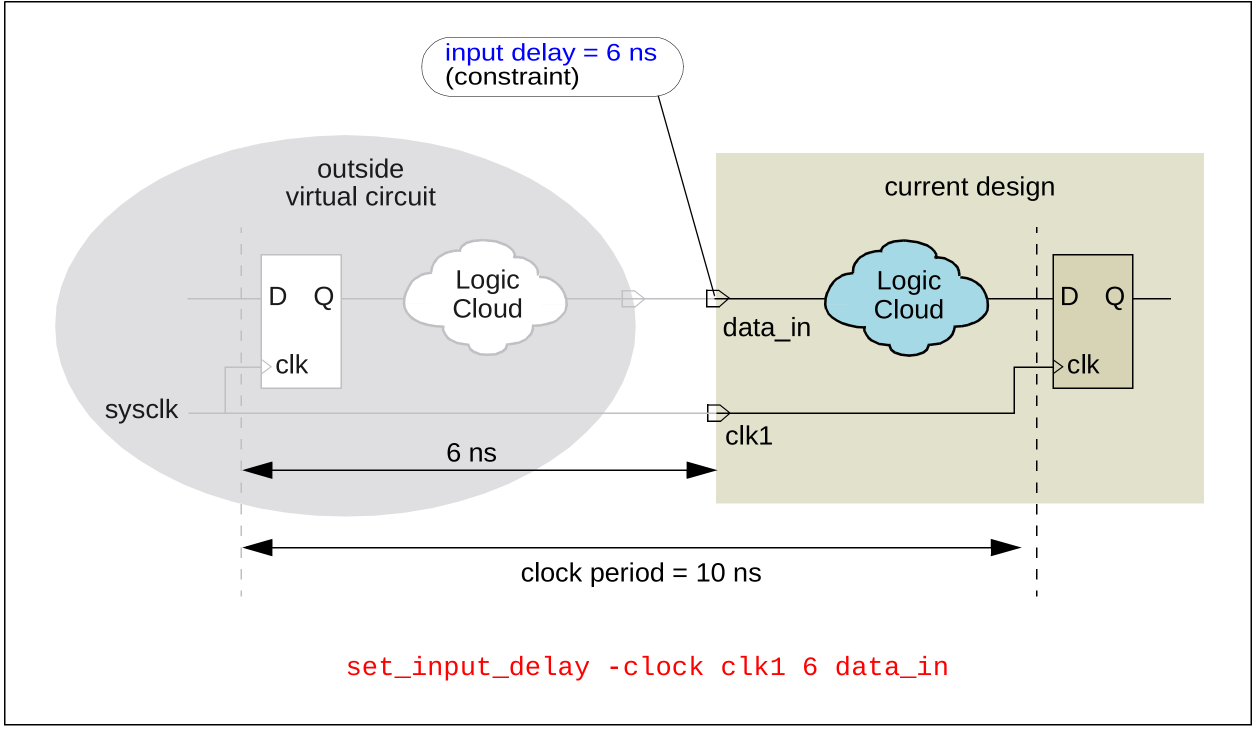
Other tools that use SDC constraints may require you to use False Path definitions to model asynchronous clocks. Although Precision RTL Synthesis supports this method, it is much less efficient in terms of runtime and memory utilization. For large designs it is well worth the effort to convert False Path definitions to domains.

Specifying Input Delay

What is Input Delay?

Input Delay is the delay consumed outside of the current design before a data signal arrives at the input port (or pin). Input Delay is **equivalent** to the term *arrival time* that may be used in other Mentor Graphics tool environments. As shown in the following illustration, if the reference clock period is 10 ns and the Input Delay is specified as 6 ns, then Precision RTL Synthesis will constraint the combinational path from the input port (`data_in`) to the first register to 4 ns (minus the register setup time).

Figure 3-4. Input Delay Defined

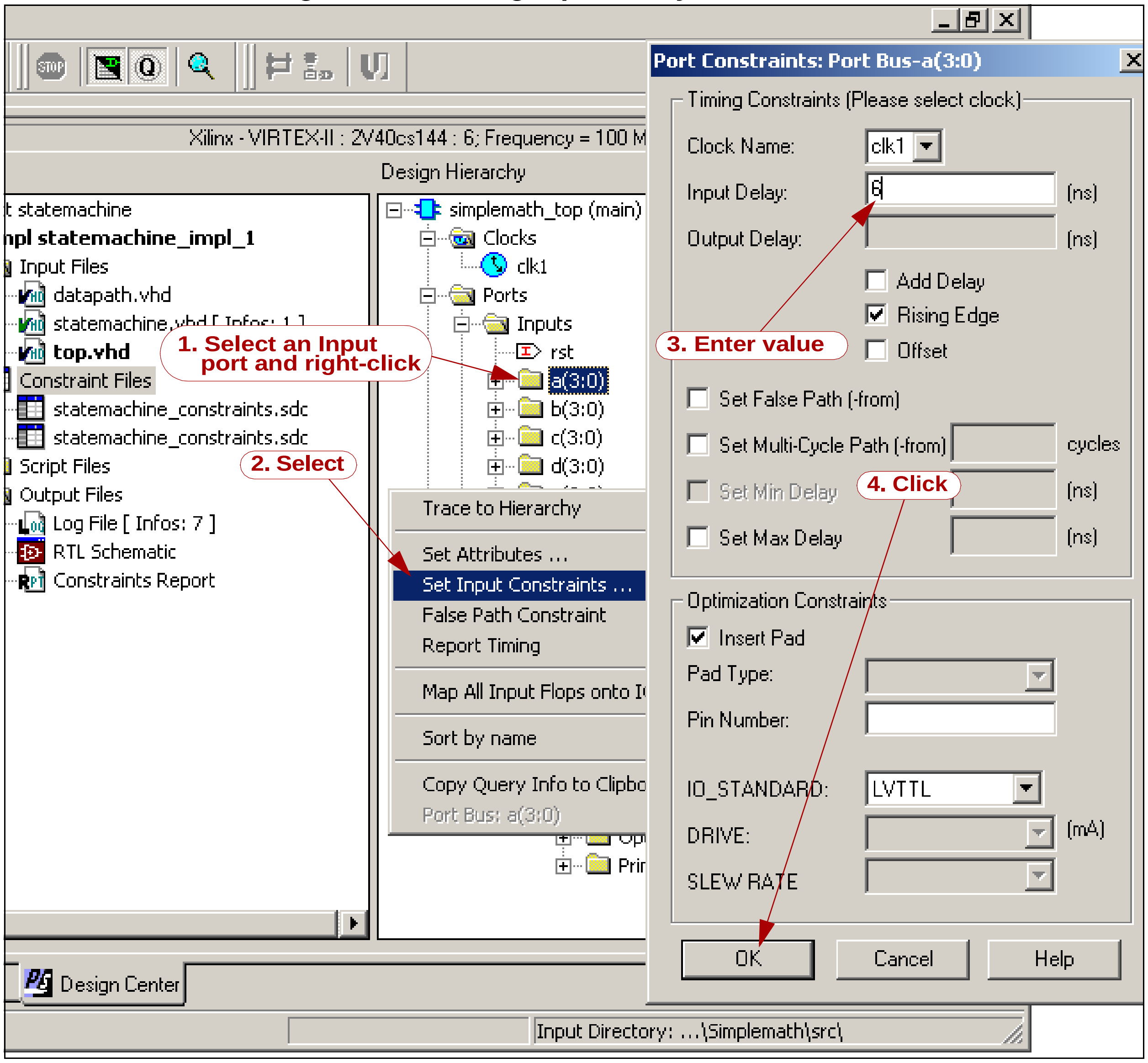


```
set_input_delay -clock clk1 6 data_in
```

Setting Input Delay from the GUI

To specify Input Delay on an input port using the GUI, follow the steps in Figure 3-5.

Figure 3-5. Setting Input Delay from the GUI


1. Select an Input port and right-click
2. Select
3. Enter value
4. Click
Xilinx - VIRTEX-II : 2V40cs144 : 6; Frequency = 100 M
Design Hierarchy
Trace to Hierarchy
Set Attributes ...
Set Input Constraints ...
False Path Constraint
Report Timing
Map All Input Flops onto I
Sort by name
Copy Query Info to Clipbo
Port Bus: a(3:0)
Design Center
Input Directory: ...\Simplemath\src\
OK
Cancel
Help

Port Constraints: Port Bus-a(3:0)

Timing Constraints (Please select clock)

Clock Name: clk1

Input Delay: 6 (ns)

Output Delay: (ns)

☐ Add Delay
☑ Rising Edge
☐ Offset
☐ Set False Path (-from)
☐ Set Multi-Cycle Path (-from) cycles
☐ Set Min Delay (ns)
☐ Set Max Delay (ns)

Optimization Constraints

☑ Insert Pad

Pad Type:

Pin Number:

IO_STANDARD: LVTTL

DRIVE: (mA)

SLEW RATE

This action adds a `set_input_delay` command to the generated .sdc file and applies an Input Delay constraint to the specified port. The basic syntax of the `set_input_delay` command is:

```
    set_input_delay -design <view_type> -clock <defined_clock> <value> <port>
```

After the action in Figure 3-5, you should see the following command in the Transcript Window and the generated constraint file:

```
set_input_delay -design rtl -clock clk1 6 a(3:0)
```

The set_input_delay Command

The `set_input_delay` command applies an Input Delay constraint to the specified input port (or instance pin). The reference clock must be defined prior to executing this command.

Input ports are assumed to have *undefined* Input Delay, unless otherwise specified. This means that if you don't specify the Input Delay constraints, no timing will be checked for input-to-register paths. This can be a handy way to declare False Paths. Often a reset input is left unconstrained for this reason.

For inout (bidirectional) ports, you can specify the Input Delay with the `set_input_delay` command and specify the Output Delay with the `set_output_delay` command. To describe a path delay from a level-sensitive latch, you should use the **`-level_sensitive`** option in the Master Constraint File. If the latch is positive-enabled, set the Input Delay relative to the rising clock edge; if it is negative-enabled, set the Input Delay relative to the falling clock edge.

Assuming that the current design is a module in a bigger design, the Input Delays must be obtained from the system designer, based on the chip environment described in the system functional spec. If you don't know the Input Delay, it is common to specify zero delay at first to ensure that the circuit successfully completes Place & Route on the first run. On future synthesis runs, you can tighten up the Input Delays to more closely model the real environment. This is a much better technique than leaving Input Delays undefined.

If you are you manually entering a `set_input_delay` command into the Master Constraint File, you can use wildcards with the `get_ports` command to simplify the specification of external delays on bus pins. For example:

```
> set_input_delay -design rtl -clock clk1 6[get_ports waddr(*)]
```

And you can utilize the `all_inputs` command to set a default value on all top-level input ports:

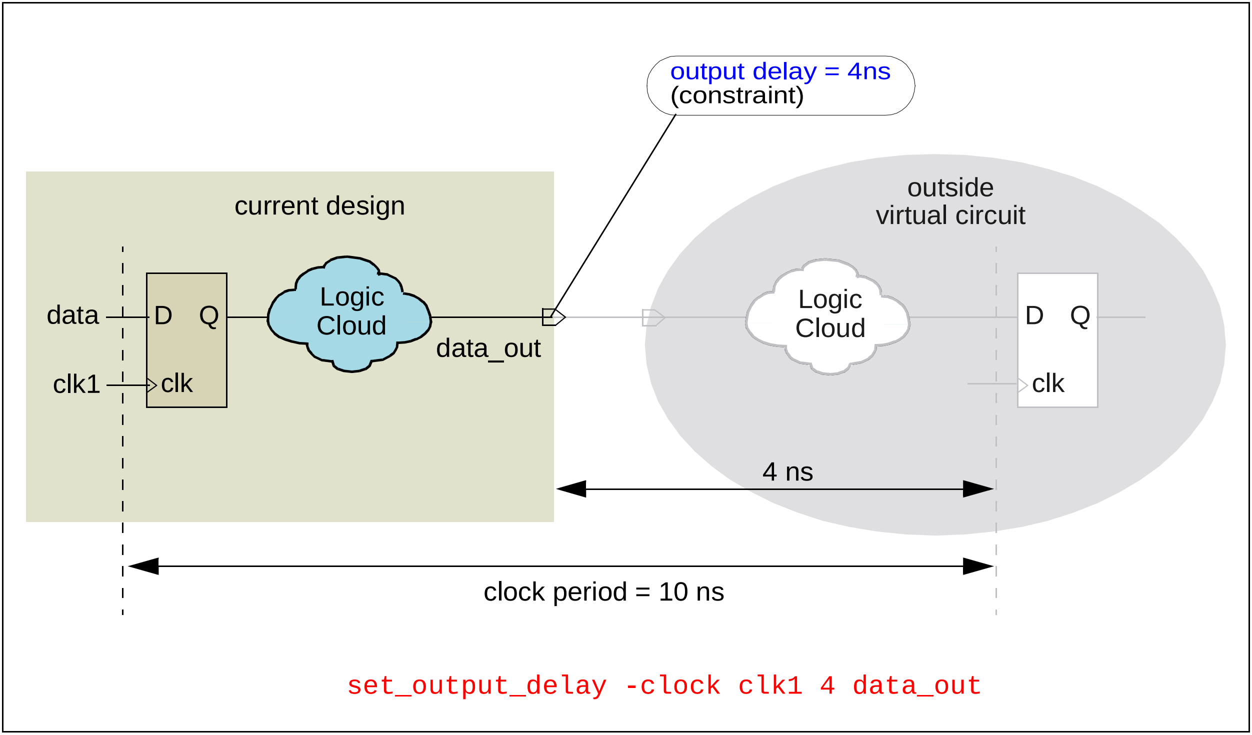
```
> set_input_delay -design rtl -clock sysclk 3 [all_inputs]
```

Specifying Output Delay

What is Output Delay?

Output Delay is the delay required outside of the current design in order to properly clock the driven device. Output Delay is the **inverse** of the term *required time* that may be used in other Mentor Graphics tool environments. As shown in Figure 3-6, if the reference clock period is 10 ns and the Output Delay is specified as 4 ns, then Precision RTL Synthesis will constrain the combinational path from the clock pin of the internal register of the current design to the specified output port to 6 ns

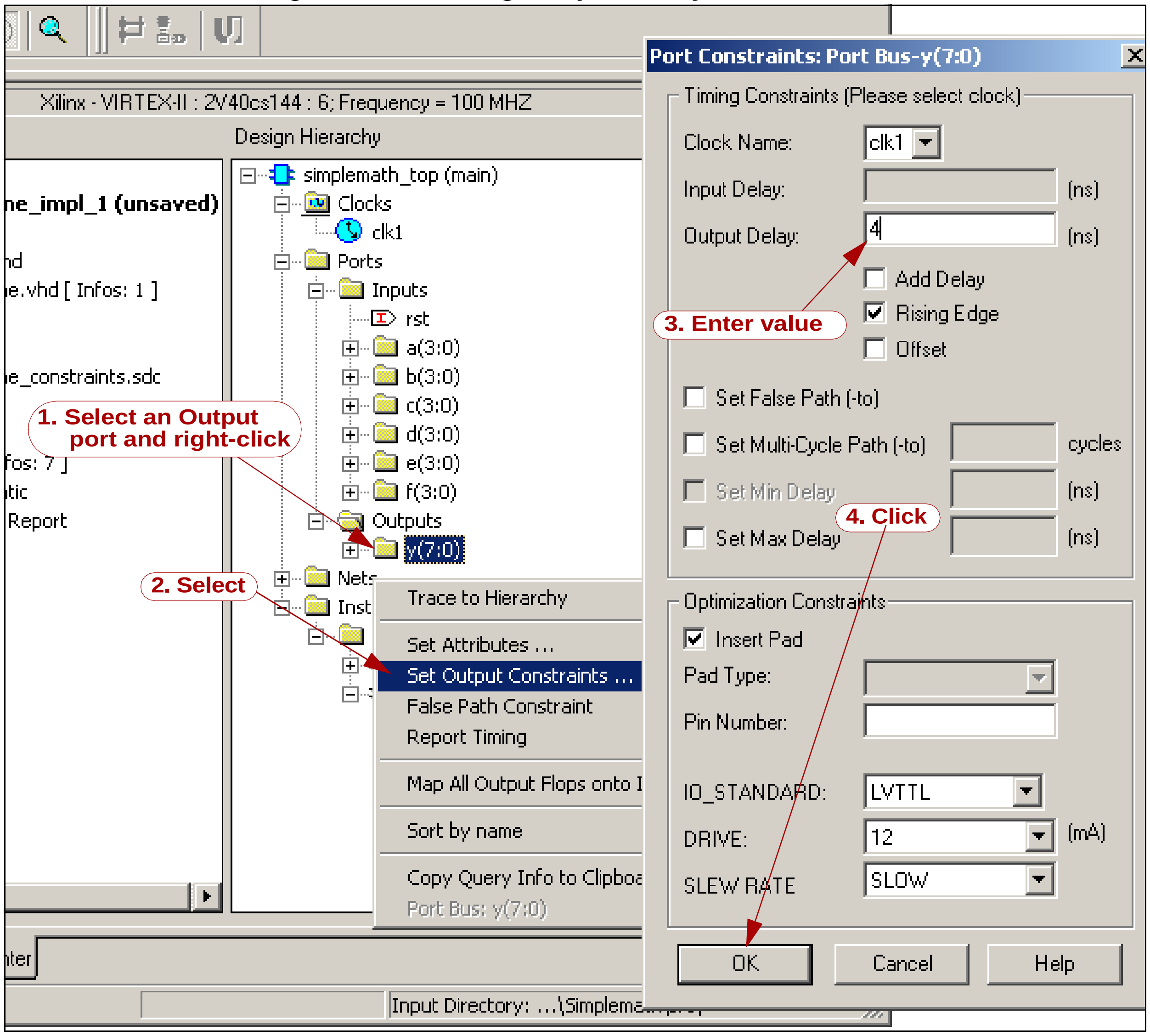
Figure 3-6. Output Delay Defined



Setting Output Delay from the GUI

To specify Output Delay on an output port using the GUI, follow the steps in Figure 3-7.

Figure 3-7. Setting Output Delay from the GUI



This action adds a `set_output_delay` command to the generated .sdc file and applies an Output Delay constraint to the specified port. The basic syntax of the `set_output_delay` command is:

```
set_output_delay -design <view_type> -clock <defined_clock> <value> <port>
```

After the action in Figure 3-7, you should see the following command in the Transcript Window and the generated constraint file:

```
set_output_delay -design rtl -clock clk1 4 y(7:0)
```

The set_output_delay Command

The `set_output_delay` command applies and Output Delay constraint to the specified output port (or instance pin). The reference clock must be defined prior to executing this command.

Output ports are assumed to have *undefined* Output Delay, unless otherwise specified.This means that if you don't specify the Output Delay constraints, no timing will be checked for register-to-output paths.

For inout (bidirectional) ports, you can specify the Output Delay with the `set_output_delay` command and specify the Input Delay with the `set_input_delay` command. To describe a path delay from a level-sensitive latch, you should use the **`-level_sensitive`** option.

Output Delays must be obtained from the system designer, based on the chip environment described in the system functional spec. If you don't know the Output Delay, it is common to specify zero delay at first to ensure that the circuit successfully completes Place & Route. On future synthesis runs, you can tighten up the Output Delays to more closely model the real environment. This is a much better technique than leaving Output Delays undefined.

If you are you manually entering a `set_output_delay` command into the Master Constraint File, you can use wildcards with the `get_ports` command to simplify the specification of external delays on bus pins. For example:

```
> set_output_delay -clock clk1 6[get_ports data_out(*)]
```

And you can utilize the `all_outputs` command to set a default value on all top-level output ports:

```
> set_output_delay -clock sysclk 3 [all_outputs]
```

Setting False Paths

False paths are design paths that you want Precision to ignore for timing optimization purposes. Timing optimization focuses it's efforts on paths in the design that violate timing constraints. If a path is seen in violation, but is in fact not a problem, then Precision RTL Synthesis will waste time trying to fix the false violation. In addition, Precision will likely insert unnecessary circuitry and may view real violations as a lower priority. You can use a False Path definition in these cases to avoid the false violation.

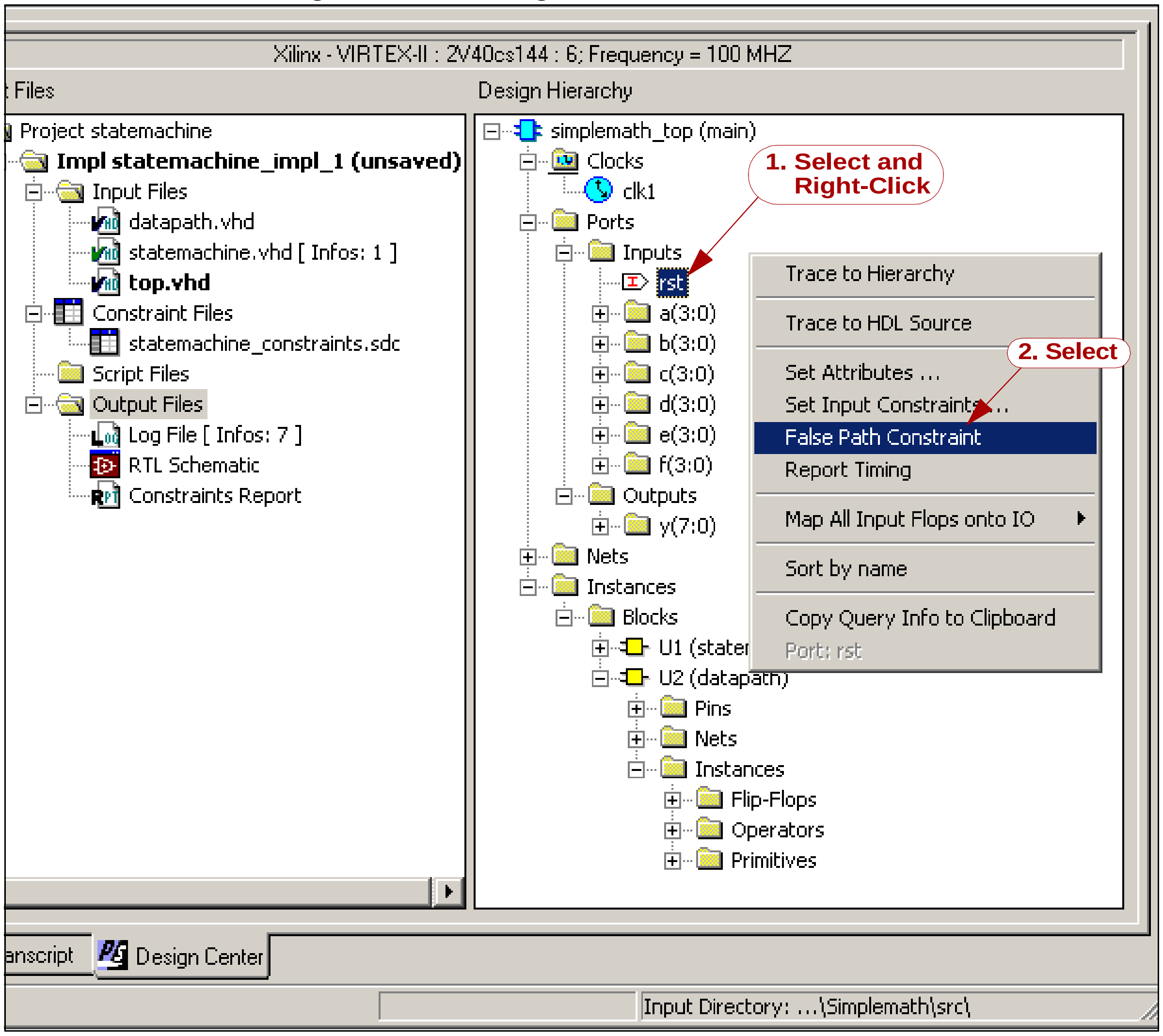
False paths occur for a variety of reasons. A path may be false because you know that the path will never pass data forward or that the circuitry involved will operate slower than your constraints indicate. For example, test logic can usually be operated at a lower frequency if necessary. Paths from master reset signals are often false for a similar reason. Often reset sequences occur over many clock cycles, and it doesn't matter exactly when a particular register is reset.

Another possible reason for using a False Path definition is to exclude paths that exist in the circuit, but because of the operation of the system, they never interact. Static analysis cannot not know the relationship of states of your circuit, so False Path definitions are used to help PreciseTime ignore these paths.

Setting a False Path from the GUI

To specify a False Path using the GUI, follow the steps in Figure 3-8.

Figure 3-8. Setting a False Path from the GUI



This action adds a `set_false_path` command to the generated `.sdc` file and applies a `false_path` attribute to the specified port or pin. The basic syntax of the `set_false_path` command is:

```
set_false_path -design <view_type> [-from <from_list>[ [-to <to_list>]
```

After the action in Figure 3-8, you should see the following command in the Transcript Window and the generated constraint file:

```
set_false_path -design rtl -from rst
```

If you identify a false path between two registers, FF1 and FF2 for example, and you are you manually entering a `set_false_path` command into the Master Constraint File, the command might look like the following:

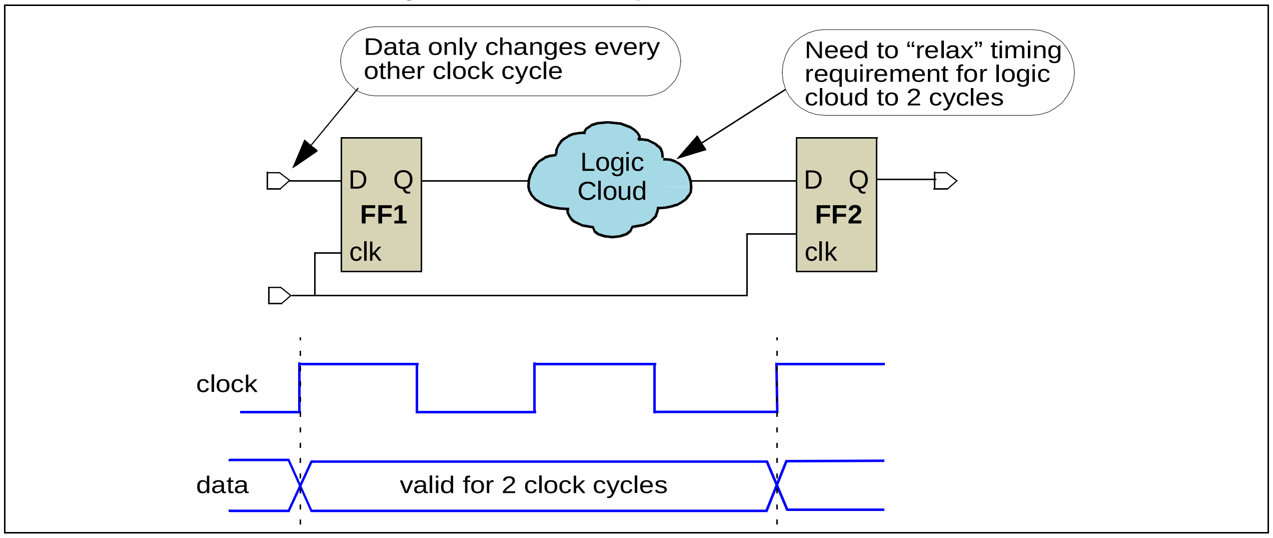
```
> set_false_path -from U1.FF1.clk -to U1.FF2.data_in
```

Setting Multicycle Paths

Logic that you know will take more than one clock to propagate through is quite common. When you set a multicycle constraint, you identify a path to PreciseTime to prevent the tool from reporting invalid violations, and to prevent timing optimization from wasting time and resources trying to correct valid (but falsely reported) paths.

Single-point multicycle path definitions are much more efficient for PreciseTime to process, both in terms of analysis time and memory utilization. If you have a design with hundreds of multicycle paths, it is sometimes worthwhile to see if a single point constraint would suffice.

Figure 3-9. Multicycle Path Defined


Data only changes every other clock cycle
Need to “relax” timing requirement for logic cloud to 2 cycles
D Q FF1 clk
Logic Cloud
D Q FF2 clk
clock
data
valid for 2 clock cycles

To appropriately constrain the design in Figure Figure 3-9, you should place the following `set_multicycle_path` command in the Master Constraint File.

```
set_multicycle_path -from FF1 -to FF2 -value 2
```

Setting Mapping Constraints

To get the best results from Precision RTL Synthesis, you should verify the default assumptions that Precision makes during the synthesize process to meet your specific design needs. The following subsections describe the four common areas that you should check.

Mapping Ports to Pin Numbers and IO Pads

Mapping to Pin Numbers

Most place/route tools allow you to assign pin numbers to ports in either the EDIF netlist file or in a separate constraint file. Precision supports assigning device pin numbers to top-level ports. These pin numbers are transferred to the synthesized netlist as a technology specific attribute which is recognized by the place-and-route tool. Like all attributes, you can specify the pin number constraints in your HDL code or use the `set_attribute` command in the Master Constraint File. You can use either the technology specific attribute name (e.g. Xilinx uses LOC) or you can use the generic `pin_number` attribute and Precision will map this generic attribute to the technology specific attribute during synthesis.

Table 3-1. Technology Pin Name Attributes

Vendor	Pin Number Attribute Name
Actel	ALSPIN
Altera	Use Altera “pin_lock” attribute
Xilinx	LOC

Mapping to IO Pads

Since some Vendor implementation tools require that IO pads be added to the design prior to running Place and Route, Precision assigns IO pads to all ports in the top level of a design by default. Precision can map input buffers, output buffers, tri-state buffers, bi-directional buffers, and complex I/O (IOs with registers) cells. The IO pads are defined in the target technology library. If the technology includes more than one IO cell of a particular class (e.g. multiple output buffers for different voltage standards), the technology vendor will specify a default IO pad to use for that class of IO pad.

Although moving registers into the IO ring can decrease the input-to-register and register-to-output delay, it can also increase the delay from the complex IO to the registers in the core of the chip. You should consider this trade-off and make sure that you are not trading off one timing problem for another.

If these IO assignments do not meet your design constraints, then you can override the default assignments by adding attributes to the ports or, with some vendors, you can manually instantiate the IOs in your design. Since each technology has different constraints on the usage of complex I/Os (for manually assigned I/Os) you are responsible for the validity of the output design. Precision does not preform checks to verify that instantiated IO cells meet the constraints of the technology.

Table 3-2. Handling Different IO Pad Scenarios

Situation	Action
To prevent Precision from adding any IO pads	Use the **Tools** > **Set Options** > **Optimization** dialog box. Click off the **Add IO Pads** radio button, then click **Apply.** This adds the command `setup_design -addio=false` to the Project File.
To prevent Precision from adding IO pads on a top-level port	A) Select an IO port in the Design Hierarchy pane or the Schematic Viewer, right-click and select Set Input Constraints. Click off the **Insert Pad** radio button. B) Attach a `nopad` attribute to the port in the HDL source. This situation typically occurs when you are using IP blocks that includes special instantiated IO pads in the IP core (e.g. PCI cores)
To assign a non-default IO pad to a top-level port	A) Select an IO port in the Design Hierarchy pane or the Schematic Viewer, right-click and select Set Input Constraints. Select one of the options from the **Pad Type** dialog box. B) Attach a `buffer_sig` attribute to the top-level port with a value set to the name of the IO pad
To force the first register in the path into the Complex IO	A) Select an IO port in the Design Hierarchy pane or the Schematic Viewer, right-click and select Set Input Constraints. Click on the radio button **Force Register into IO**. B) Attach a `buffer_sig` attribute to the top-level port with a value set to the name of the IO pad
To infer a complex IO when the associated register is not in the same level of hierarchy.	Precision only moves internal registers to complex IOs when the registers exists on the top-level of hierarchy. If the register is buried in hierarchy, you must either instantiate the complex IO in the lower-level block or flatten the hierarchy (using the `hierarchy` attribute) so that the buried registers are moved to the top-level of the design.

You use the `set_attribute` command to set attributes in the Master Constraint File.

Mapping Internal Registers to IO Block Registers

By default, Precision Synthesis maps all internal candidate registers to registers located in IO Pads. This maximizes the performance at the interface of the chip, but may result in a longer delay path from the IO block to a register in the chip core. You can disable this automatic IO Pad mapping on a selective basis to increase the performance on internal critical paths. From the GUI, you Compile the design, select an IO port and right-click to bring up the popup menu. Select **Force Input Flop onto Input Pad > FALSE**. This sets an attribute on the port that tells Precision how to map the IO register. One of three attributes are set depending on the type of port, `inff` for input ports, `outff` for output ports, and `triff` for tri-state ports. The commands that are recorded in the SDC constraint file might look like the following:

```
set_attribute -design rtl -name INTFF -value FALSE -port data_in(1)
set_attribute -design rtl -name OUTFF -value FALSE -port data_out(1)
set_attribute -design rtl -name TRIFF -value FALSE -port data_inout(1)
```

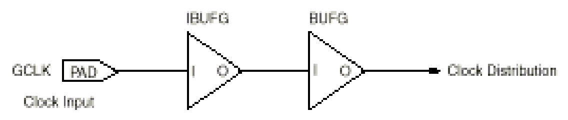
You may also set these attributes on ports in the source HDL. Refer to the Precision Synthesis Reference Manual for details and examples. This functionality currently applies to all Xilinx Virtex-I/II/Pro and Spartan technologies. The equivalent functionality for other technologies is exercised by selecting the popup menu item **Force Register into IO** which sets the attribute named *map_complex*.

Mapping Clocks to Dedicated Clock Resources

Most FPGA technologies contain dedicated clock routing channels to handle high fanout nets such as clock and enable lines. These device resources are usually hard wired into the chip so using the global buffers typically does not utilize any extra area. Since these global buffers are designed to handle high fanout loads, you should attempt to use all of these resources for clock signals (to minimize skew and slew) and then use the remaining global buffering resources for high fanout signals in the design. If you do not have extra global buffering resources available, Precision will fix the fanout using either buffering or logic replication.

Precision assigns clock buffers to top-level and internal signals that drive clock pins. Since the technology contains the maximum number of global clock buffers for each technology and die size, Precision does not exceed the maximum number of available global clock buffers.

Typically, you will let Precision automatically insert clock buffers on the initial synthesis run. As shown in the figure below, Precision automatically infers the IBUFG and BUFG clock buffers when the corresponding input signal is used as a clock in the VHDL or Verilog code.



Note: The actual cell inserted is a BUFGP which is equivalent to an IBUFG and a BUFG combination in Virtex-II.

After viewing the Area Report and the transcript, you can determine how many and where the clock buffers were used. If you have some global buffers still available, you can use the `buffer_sig` attribute to specify a hierarchical net pathname(s). These attributes should be added to your design constraint (`.sdc`) file so that you can consistently reproduce the results.

Table 3-3. Handling Different Clock Buffering Scenarios

Situation	Action
To assign a clock buffer to a top-level port or internal net	A) Select a clock port in the Design Hierarchy pane or the Schematic Viewer, right-click and select Set Clock Constraints. Select one of the options from the **Clock Buffer** dialog box. B) Attach a `buffer_sig` attribute to a top-level port net or internal net with a value of the global buffer name

Mapping Operators, Counters, and Memory to Chip Resources

Although you can instantiate technology-specific modules into your HDL source code, Precision RTL Synthesis has a module generator (called Modgen) that creates efficient technology-specific implementations of arithmetic and relational operators, counters, and memory (RAM and ROM). For detailed explanations and coding examples, refer to the following topics in the Precision RTL Synthesis Style Guide:

Inferring Operators and Counters
Inferring a Pipelined Multiplier
Mapping Operators to Dedicated Resources
Mapping Blocks to Altera LogicLock Regions
Mapping to Altera Memory Resources
Mapping to Xilinx Memory Resources

Setting Other Constraints

Controlling Hierarchy

Synthesis tools create a block of hierarchy for every component instantiation in an HDL-based design. Although hierarchy is an excellent method for reducing a complex engineering problem into reasonable, understandable blocks, it does constrain the optimizer from obtaining the best possible results. For example, if an `AND` gate and inverter are in separate blocks of hierarchy, synthesis tools would traditionally not be able to optimize the logic into a single `NAND` gate. Designers want to keep as much hierarchy as possible to aid in debugging the design. The synthesis tools wants to flatten all of the design hierarchy (within the constraints of the RAM and CPU speed of the computer) so they will have the best chance of obtaining the fastest, smallest design. Unfortunately, flat designs with their inherently long, abstract instance and net names may be difficult to examine.

To try to achieve both of these conflicting hierarchy goals, Precision performs intelligent auto-dissolving of smaller blocks of hierarchy. By examining the instance count and type of logic within a hierarchical block, Precision will only flatten the hierarchy of a lower-level block if the block contain less than ~50 instances *and* there is a high likelihood that removing the hierarchy will improve the synthesis results. Precision reports a message (prior to optimizing each hierarchical block) to the transcript whenever it auto-dissolves a block of hierarchy.

Prior to running synthesize, you can override the hierarchy control on any block in the design by placing a `hierarchy` attribute on a specific instance. Before you create or change the design hierarchy, you should consider these ramifications:

- *Gate counts in leaf blocks should not exceed 50K gates.*

  Optimization can be performed on much larger designs provided that the sub-hierarchy falls within this guideline.

  The maximum size of a single level of hierarchy is difficult to define since it is so dependent on the design and technology. Number of operators, type, style of RTL code, amount of random logic, etc. all play a role in the maximum size that Precision can handle.

  Precision RTL Synthesis characterizes constraints down through hierarchy. And, if you wish, you can constrain each block individually if you want to override the characterized constraints through hierarchy. Typically, applying constraints to lower level blocks is only necessary if the instance is a blackbox (without any underlying logic, Precision can not determine the timing through the block).

- *Keep related critical paths in the same block of hierarchy.*

  Keep in mind that Precision RTL Synthesis performs area and timing optimization separately. By separating timing critical logic into one block, it may be possible to perform aggressive area optimizations on a greater percentage of the design (thus creating a smaller circuit that meets timing).

- *Place registers at end of hierarchical boundaries.*

  Since optimization tools can only reduce combinational logic, there are two “barriers” that constrain optimization, hierarchical boundaries and registers. When designing hierarchically, you should attempt to place registers at either the front or back end of the hierarchical boundary. This register placement essentially combines the two “barriers” into one thus minimizing the impact to overall results when optimizing hierarchical designs.

- *Keep state machines into separate blocks of hierarchy.*

  This partitioning will speed the optimization and provide greater control over encoding.

- *Place tristate drivers at the same level of hierarchy.*

  This allows you (and Precision RTL Synthesis) more flexibility in handling/implementing tristates. By default, Precision RTL Synthesis will not move tristate drivers across hierarchy (because it would require changing the port interface on the hierarchical block to pass the enable signal). You should also be aware of whether your target technology has tristate cells available internally and/or in the IO ring. For example, Altera technologies only have tristates on the IO ring (no internal tristates).

The entire design hierarchy can be controlled by applying the hierarchy attribute to the top-level of the design and then compiling the design. Then in the Design Center window, you can right-click on the top-level and choose either **Flatten Hierarchy** or **Preserve Hierarchy**.

To flatten/preserve one hierarchical block in the design, select the block, then right-click and select **Flatten Hierarchy** or **Preserve Hierarchy**.

Protecting Blocks from Change (dont_touch)

You may have portions of your design that have been hand-implemented and you want to prevent Precision RTL Synthesis from performing any optimizations on those blocks or instances. For example, you would not want Precision to optimize an technology-specific IP blocks (e.g. CoreGen or ACTGen cells) or you may want to create a fast-transition clock signal. To do this, you could specify the buffer as a technology instantiation and retain it through optimization. This method of protecting hierarchy is also very useful when you are “gluing” lower-level blocks together into a larger design. You can optimize a low-level block then in the top-level design, you can set a `dont_touch` attribute on the technology mapped blocks and only optimize the level above.

Controlling Fanout on Data Nets

Fanout is the maximum number of pins being driven by an instance or top-level port. High fanout nets can potentially cause significant delays on wires and/or make a net un-routable. On a critical paths, high fanout nets can cause significant delays in a single net segment and cause the timing constraints to fail. To limit these issues, technology libraries have a global fanout value set in the library with possible individual overrides on specific cells designed for high fanout situations (e.g. global buffers).

To eliminate routability and timing issues associated with high fanout nets, Precision also allows you override the library default value on a global or pre-net basis. You can override the library value by setting a `max_fanout` attribute on the net.

Depending on the technology, Precision can address fanout violations by splitting the net and replicating the driving cell. If replication is not possible (e.g. the driver is an input port), Precision will add buffers. By using the technology default values with selective user-defined overrides, Precision can control the fanout on a net.

Chapter 4
Managing Projects

Performing FPGA synthesis can be an iterative process where constraints and optimization settings are constantly adjusted until the desired result is achieved. Each of these iterations will generate a result based on a specific set of inputs, constraints and synthesis options. When you perform these iterations, often you will save multiple versions of your design for future reference. Precision Synthesis includes the Project Manager to help you manage the data created during the iterative synthesis process.

The Project Manager

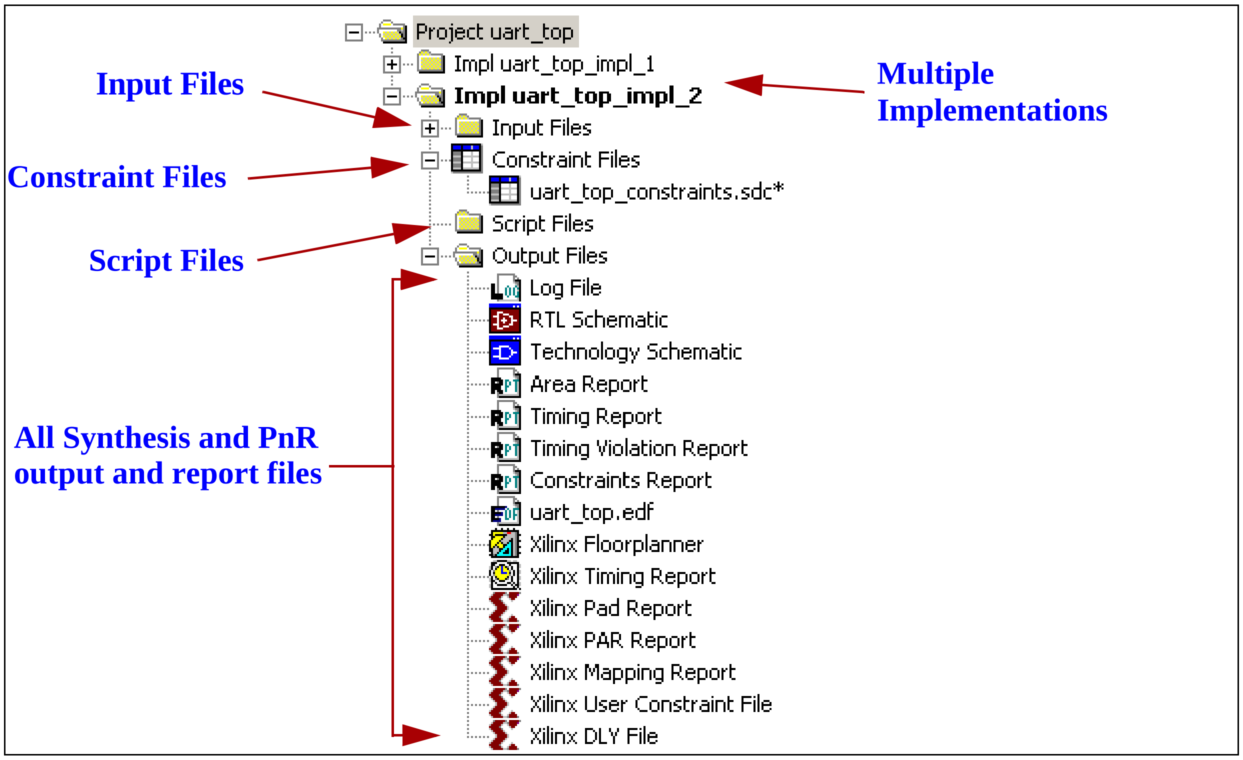
The Project Manager offers a framework that supports and promotes a design process based on experimentation. The Project Manager system is based on *projects* which contain multiple *implementations*. Each implementation within a project is a separate workspace in which you can experiment with different synthesis strategies and configurations. Implementations keep track of all of the design settings, input files, and output files that comprise its configuration. The Project Manager facilitates your design efforts by allowing you to create, delete, copy, edit and save implementations.

After creating a project and its implementation(s), proceed with your synthesis design flow. Before closing the project, save the current state of the active implementation. Then the next time you open the project, it will be restored to the state it was in when you closed it.

The Project Manager is accessible in all user interface modes; Graphical User Interface (GUI), Tcl command line in the Transcript window or the shell window, and Tcl scripts when running Precision in batch mode. Precision provides Tcl commands that correspond to all of the Project Manager menus and Design Bar icons, as well as other supporting commands. The Tcl commands are detailed in Chapter 3, Commands, in the Precision Synthesis Reference Manual. Refer to Table 3-2, Project and File Management Commands.

Figure 4-1 shows an example of how the contents of a project are displayed in the Project Files pane of the Precision Design Center window. The example project name is “uart_top,” and it has 2 implementations, “uart_top_impl_1” and “uart_top_impl_2”.

Figure 4-1. Organization of Project Files by the Project Manager



Bypassing the Project Manager

As mentioned earlier, Precision’s Project Manager is well suited for interactive sessions in which you are exploring and trying multiple versions (implementations) of the same design. In other situations, you might prefer to run Precision without using the Project Manager. Bypassing the Project Manager is desirable when you are using scripted flows in which a known set of inputs and the outputs of a single pass are all that is required. For example, if you are using Precision to automatically setup a known design and generate output files for downstream tools.

You can work interactively without the Project Manager by entering commands at the shell command line or the Precision Transcript command line. However, the Design Center GUI window will not be available. Also, you will be responsible for managing all of the input and output files of your synthesis work.

To bypass the Project Manager, simply set the results directory by calling the `set_results_dir` command. All Project Manager commands will be unavailable until the results directory is unset by the `close_results_dir` command. After the results directory is set, you can proceed with your synthesis flow; setup the design, compile, synthesize, place and route, and so on. The set_results_dir command is not available if a project is open.

Using the Project Manager Interactively

The following sections describe how to use the various Project Manager commands. In each section, the graphical user interface is presented first, followed by the equivalent Tcl command interface.

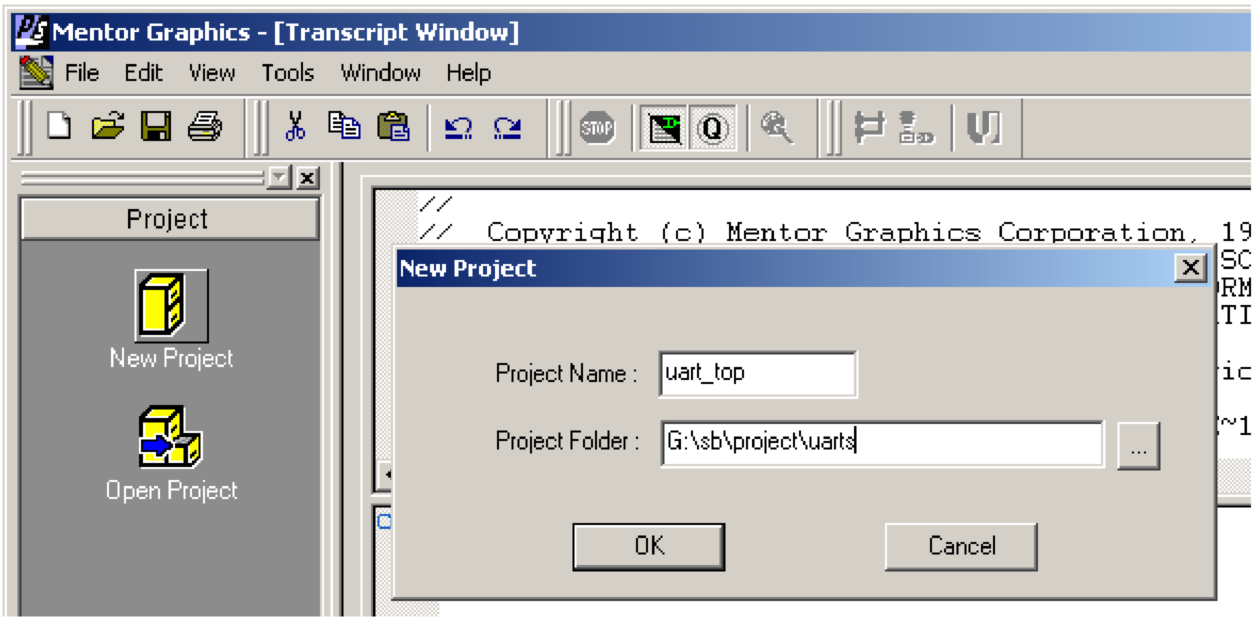
Creating a Project

The Project Manager is enabled by opening or creating a project. From that point until you close the project, all of your work is carried out in the context of the active implementation in that project.

When you invoke Precision, the session window displays the Design Bar and the transcript window. The Design Bar displays the “Project” pane which is where all project related icons are found. Initially it contains only 2 icons: **New Project** and **Open Project**. More appear after a project is open.

To create a project either click on the **New Project** icon on the Design Bar, or choose the **File > New Project** menu. Precision opens the New Project dialog box, as shown in Figure 4-2.

Figure 4-2. Initial Session Window and New Project Dialog Box


Mentor Graphics - [Transcript Window]
File Edit View Tools Window Help
Project
New Project
Open Project
New Project
OK
Cancel

```
//
//  Copyright (c) Mentor Graphics Corporation, 19
```

Project Name : uart_top

Project Folder : G:\sb\project\uarts

When you OK the dialog box, the project and an initial *implementation* are created and opened in the Design Center window, as shown in Figure 4-2.

The default project name is project_n. The project name will change automatically as you change the project folder. You can rename the project by typing the name of the project in the Project Name field. If you rename the project, the project name will not change as you change the project folder.

The system will warn you when you press the OK button on the New Project dialog box when the following occurs:

- Blank: If the name field is blank, you must enter a name for the project.
- Folder does not exist: If the folder does not exist, then you have the option of letting the tool create the folder. You may also use the Browse (...) button to display a directory and then scroll to the desired folder.
- Project name already exists: If you receive a message that the project already exists, enter a new project name in the Project Name field.

Creating a Project from the Command Line

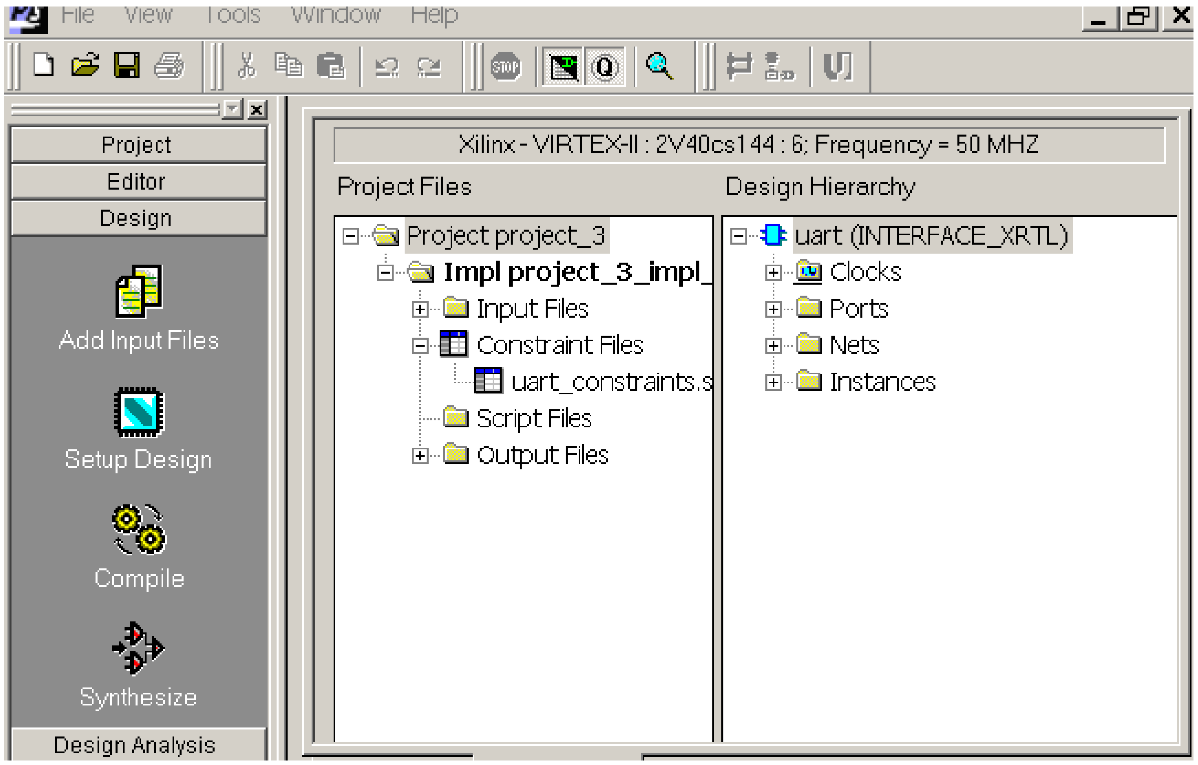
The following `new_project` command example creates the same project as in Figure 4-2.

```
new_project -name uart_top -folder G:/sb/project/uarts -createimpl
```

When executed from the Precision Transcript window, it opens the Design Center window. When executed from the shell command line, the GUI is not used. The `-createimpl` option will create a default implementation. If omitted, only the project is created. You can create an implementation later by calling the new_impl command.

New Project and Implementation

Figure 4-3. Newly Created Project and Implementation

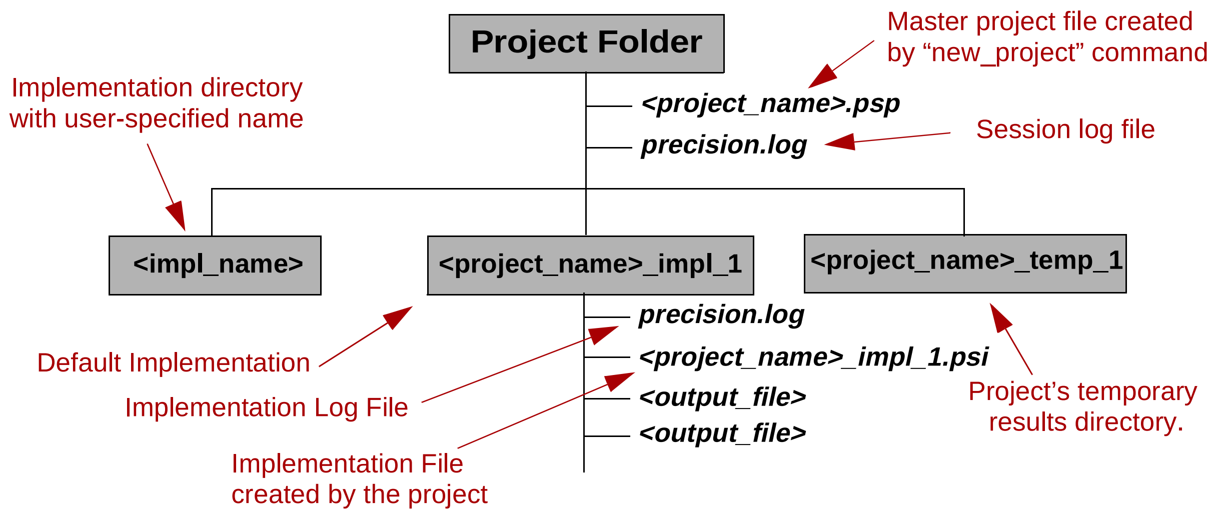


As the project is created, the project directory structure is also created in the specified Project Folder of your file system. All Project Manager commands operate directly on the project files and folders in your file system. No “save project” command exists. Note that the actions performed by your synthesis design work (files output to the implementation folder) are not automatically saved to the implementation directory, but are held in a temporary directory in the project folder. Refer to Saving the Active Implementation on page 4-11 for more information.

Figure 4-4 shows the file structure of a project directory. The project directory contains the project file (.psp). Each implementation is stored in its own subdirectory that has the same name as the implementation. Inside each implementation directory is the implementation file (.psi). The temporary directory is created each time a project implementation is activated, and deleted when it is deactivated (when the implementation is closed, or another implementation is activated or created).

You can rename implementations to more descriptive names if you wish. Refer to Renaming an Implementation for instructions.

Figure 4-4. Hierarchical Project File Structure


Project Folder
<project_name>.psp
precision.log
Master project file created by "new_project" command
Session log file
Implementation directory with user-specified name
<impl_name>
<project_name>_impl_1
<project_name>_temp_1
precision.log
<project_name>_impl_1.psi
<output_file>
<output_file>
Default Implementation
Implementation Log File
Implementation File created by the project
Project's temporary results directory.

In the example above, the master project folder contains the project file (.psp). The implementation directories "<impl_name>" and "<project_name>_impl_1" are where work is performed.

By default, the name of a new implementation is composed of the project name with the _impl_<n> suffix, where <n> is an integer that is incremented to ensure the name is unique within the project. A directory is created in the project folder for each implementation. The project's temp directory consists of the project name_temp_<m> to identify it as a temp directory and includes a unique number.

Warning: The Precision RTL Synthesis tool cannot track if more than one user is using the same project. If two users have the same project open and both are making changes, then it is possible that one of the users will lose their changes. We recommend that you do not let two users work on the same project at the same time.

The Precision Temp Directory

When working on a project using Precision Synthesis, your synthesis design work (files output to the implementation folder) are not automatically saved to the implementation directory, but are held in a temporary directory in the project folder until you save your work using the Save Active Implementation command. For more information see Saving the Active Implementation. Once you save the implementation, the Precision Synthesis tool empties the implementation directory and copies the temp directory into the implementation directory. For example, if you are working in a project with an implementation named impl_1, Precision stores the information for this implementation in the temp directory. Once you save the implementation, Precision Synthesis copies the contents from the temp directory to impl_1.

The sdc and log files are always updated and current. However, the database file (.xdb) is only written after synthesis and the .ixdb is only written after a compile. Because of the how files are written, you should always save after the synthesis and compile steps.

Temp Directories

The tool contains a “preference” that determines whether new projects use temp directories. When a new project is created, the preference is stored in the .psp file as a project property. When the project is loaded, the value is read and used to determine whether to use temp dirs when impls are activated. The default is that projects use temp directories.

If you do not want the project to use a temp directory, use the following command:

```
set_project_property -usetempdir -value false
```

Use the following command to set the global preference for new projects.

```
set_preference -pref project.usetempdir -value true
```

Turning Off the Temp Directory

You can turn off the option that causes the Precision RTL Synthesis tool to save to the temp directory. Large projects can take a long time to save, you can turn off saving to the temp directory by issuing the following global preference command. The `set_preference` command sets all *new* projects so that they do not use the temp directory. If you want to change the *current* project so that it either saves or does not save to the temp directory, use the `set_project_property` command.

```
set_preference -pref project.usetempdir -value false
```

Warning: If your session is interrupted and you are using a temp directory, you can delete the temp directory and start work where you last saved. If there is no temp directory, then the implementation will not be recoverable.

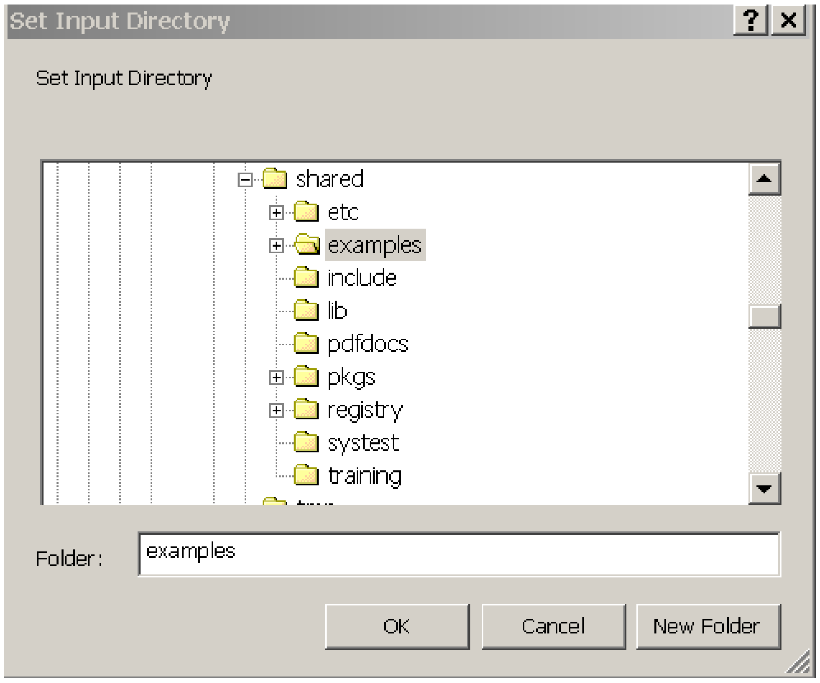
Projects Created with a Script

Projects that are created when a script runs without using `new_project` or `open_project` commands do not automatically use temp dirs and that fact is persisted in the .psp file. For more information on using scripts, see the Scripts for Creating New Projects or Reusing Existing Projects or Creating a Script from a Precision.log File sections.

Setting the Input Directory

You can set the input directory for files loaded into the project from the graphical user interface or by entering the set_input_dir command. Select File > Set Input Directory... to display the following dialog box.

Figure 4-5. Setting the Input Directory



- If the directory displayed is the directory where you want input files stored, then click **OK**.
- If you want to select another directory, find and select the directory, then click **OK**. If you want a new directory, click **New Folder** a new folder is added with the name New Folder. Overtype this name with the desired folder name.

You can also set the input directory by entering the `set_input_directory` command. See the *Precision Synthesis Reference Manual* for information on this command.

Opening a Project

Opening a project restores the project to the state it was in when it was closed. Restoring a project will reactivate the last active implementation, and set the *input directory* and *results directory* settings. The input directory is reset to the value saved in the implementation file. The results directory is set to the temporary output directory in the Project Folder. It also restores all synthesis binary databases and place and route files. If the project being opened is a pre-2003c format file, Precision will automatically convert it to the current format as it is loaded.

To open a project, either click on the **Open Project** icon in the Design Bar, or choose the **File > Open Project...** pulldown menu. Only one project can be open at a time. If a project is currently open, Precision will prompt you to close the project before it will open another project.

Opening a Project from the Command Line

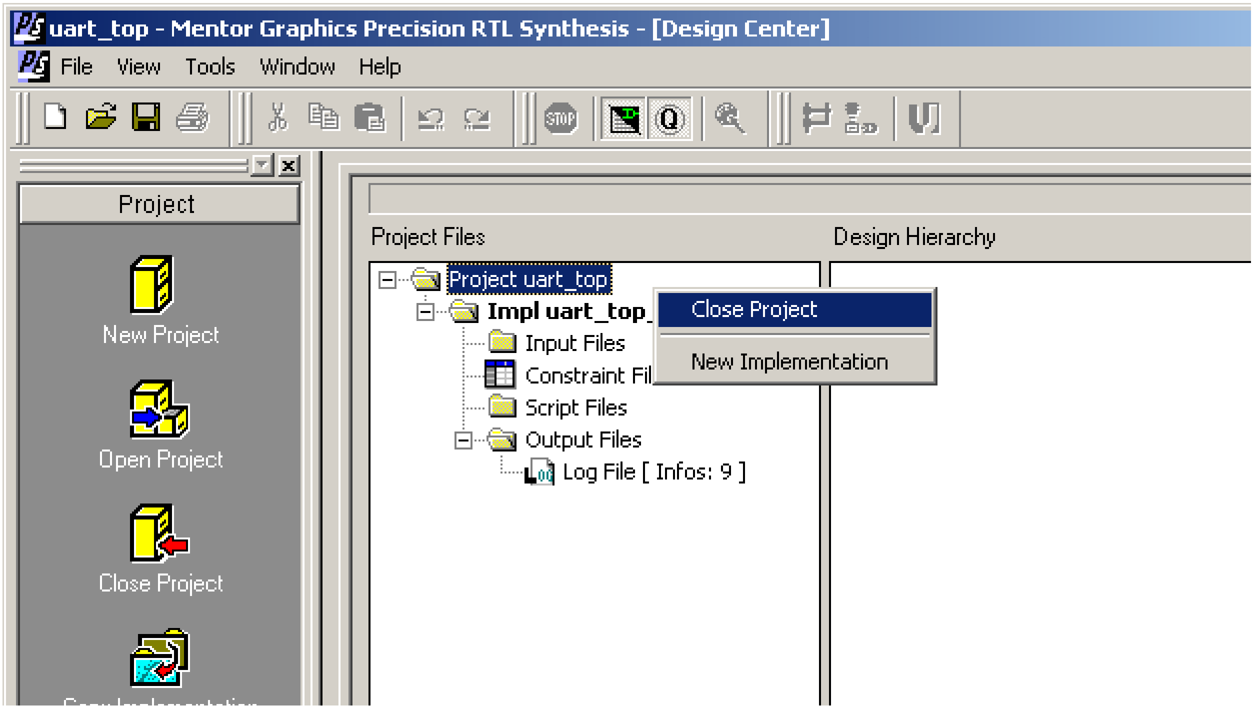
To open a project by using the Tcl command line interface, call the `open_project` command and specify the full path to a project file (.psp). The following command example open the project uart_top.psp and sets the current working directory to `g:/sb/project/uarts`:

```
open_project g:/sb/project/uarts/uart_top.psp
```

Closing a Project

The Precision GUI provides the **Close Project** command in 3 places; as an icon in the Project pane of the Design Bar, on the **File** > **Close Project...** pulldown menu, and on the popup menu when you right-click on the project object in the Project Files pane of the Design Center window. If the active implementation has unsaved changes, Precision will prompt you to save those changes before closing the project. Figure 4-6 shows the additional project management icons that appear in the Project pane of the Design Bar when a project is open. It also shows the project popup menu.

Figure 4-6. Project Manager Icons on the Design Bar



You can also close the project by calling the `close_project` command.

Setting Up the Implementation

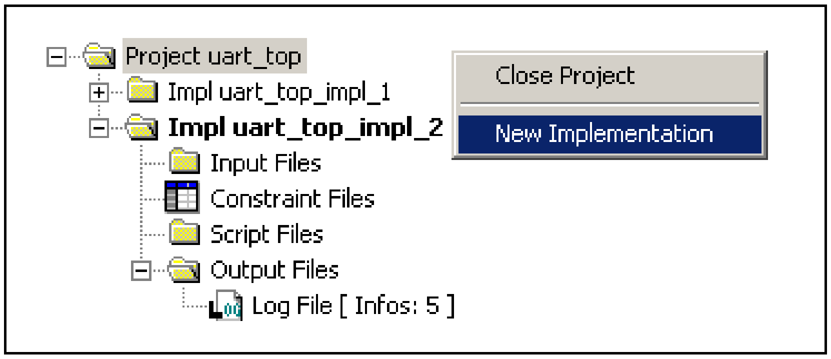
Setting up the implementation consists of two steps that implement the add_input_files and setup_design commands, which are shown graphically as icons in the Design Bar in Figure 4-2. For detailed information about these steps, refer to Setting up the Design Environment on page 2-2.

Creating a New Implementation

Precision provides a few ways for you to create a new implementation. You can choose the **File > New Implementation** menu, or right-click on the *project* in the Project Files pane of the Design Center window and choose **New Implementation** from the popup menu. The New Implementation command first closes the currently active implementation, then creates and activates the new implementation. If the current implementation has unsaved changes, Precision will prompt you to save or discard the changes before creating the new implementation.

Figure 4-7 shows the Project popup menu and a new implementation named `uart_top_impl_2`.

Figure 4-7. Project Popup Menu



The default name of the new implementation is composed of the project name with the **_impl_<*n*>** suffix, where <*n*> is an integer that is incremented to ensure the name is unique within the project. The new implementation is automatically activated. Precision creates a new implementation directory of the same name in the Project Folder.

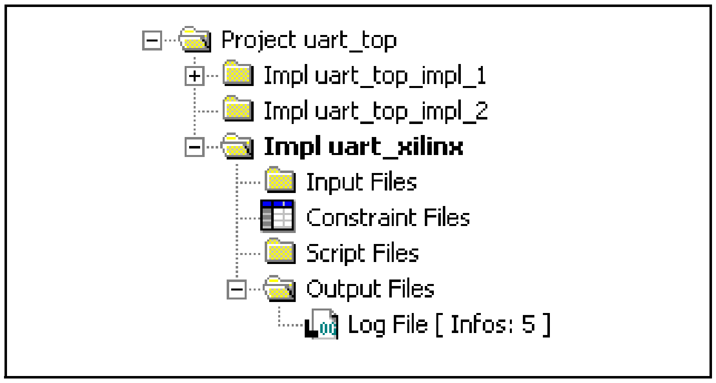
Creating an Implementation from the Command Line

When creating a new implementation by using the `new_impl` command you can supply a name for the new implementation or not. If you don't specify a name, a default name is provided. The default name of the new implementation is composed of the project name with the **_impl_<*n*>** suffix, where <*n*> is an integer that is incremented to ensure the name is unique within the project.

The following command example does three things: (1) Deactivates the active implementation, if one exists. (2) Discards any unsaved changes in the active implementation before closing it. (3) Creates and activates a new implementation named `uart_xilinx`.

```
new_impl -name uart_xilinx -discard
```

Figure 4-8. New `uart_xilinx` Implementation



If the `-discard` option is omitted and the current implementation has unsaved changes, the `new_impl` command aborts and displays an error message in the transcript. If you want to save the changes, call the save_impl command.

Saving the Active Implementation

As you work in the active implementation, all of your work is held in the project's temporary directory until you choose to save it. Precision indicates that you have unsaved work by displaying "(unsaved)" next the active implementation name in the Project Files pane of the Design Center window. The **Save Implementation** command copies the contents of the temporary directory into the active implementation directory.

Caution

Do not save files directly into an implementation directory because those files will be deleted during the save_impl operation. That is because save_impl deletes all files inside the implementation directory before it copies the files from the temporary directory.

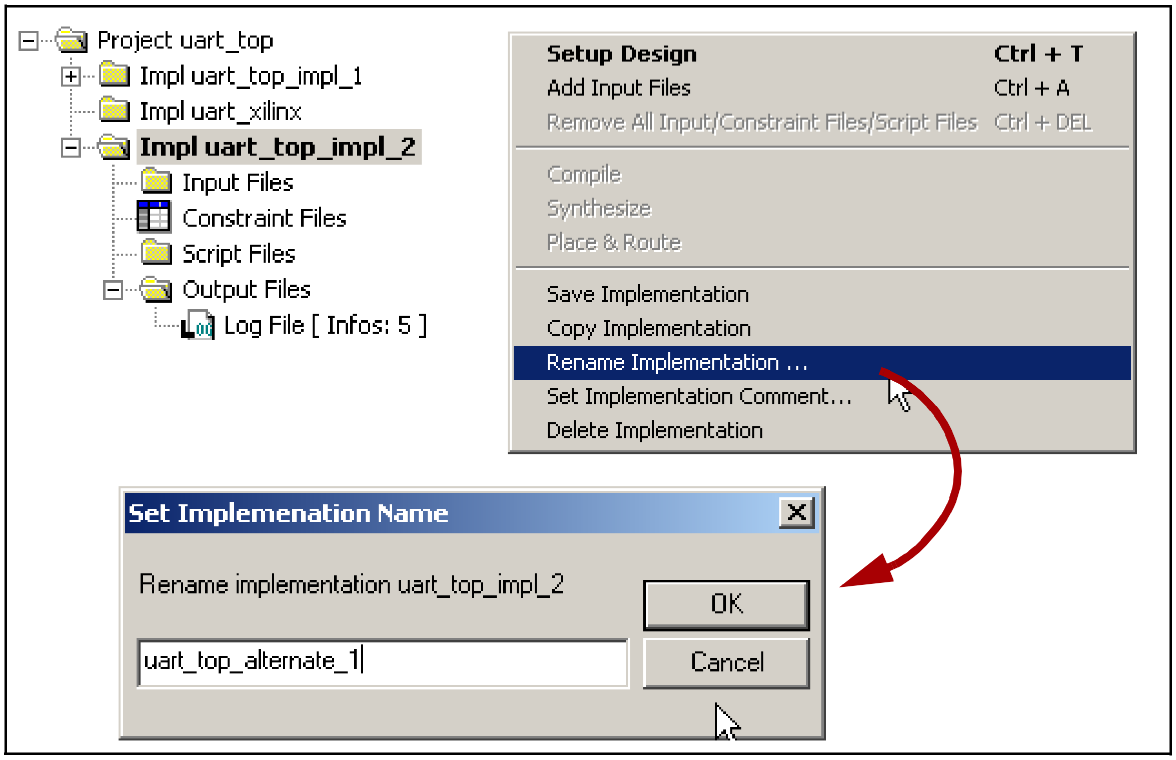
If you want to write files into the active implementation directory, save them to the results directory (use get_results_dir to obtain the pathname) which is actually the temporary directory when an implementation is active.

To save your work, either right-click on the active implementation in the Project Files pane of the Design Center window and chose **Save Implementation** from the popup menu, or choose the **File** > **Save Active Implementation** pulldown menu. From the command line, call the save_impl command.

Renaming an Implementation

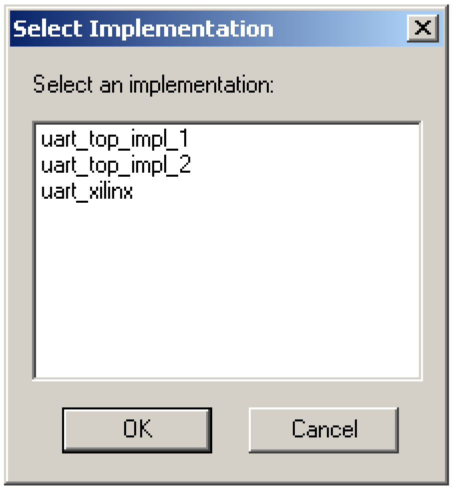
Implementation names are arbitrary. You can use any name that is meaningful to you. The project keeps track of all file associations. To rename an implementation, right-click on the implementation in the Project Files pane of the Design Center window and choose **Rename Implementation** from the popup menu. That opens the Set Implementation Name dialog box. Enter the new implementation name and OK the dialog box. (Figure 4-9)

Figure 4-9. Menu and Dialog for Renaming the Active Implementation



Alternatively, you can use the **File** > **Rename Implementation** pulldown menu item. This menu choice opens the Select Implementation dialog box which, when in "rename" context, lists the implementations in the project that you can rename (see Figure 4-10).

Figure 4-10. Select Implementation Dialog Box



When you OK the Set Implementation Name dialog box, the name change appears in the Project Files pane of the Design Center window. Precision also immediately renames the implementation folder and the implementation file (.psi) in the file system. Any unsaved work is retained in the temporary directory and will be saved to the renamed directory when you save the implementation.

Renaming an Implementation from the Command Line

An implementation's name is actually a property of the implementation. To change that property, use the `set_impl_property` command. The following example sets the name property of implementation `uart_top_impl_2` to `uart_top_alternate_1`. If you do not use the `-impl` option, `set_impl_property` operates on the active implementation.

```
set_impl_property -impl uart_top_impl_2 -name uart_top_alternate_1
```

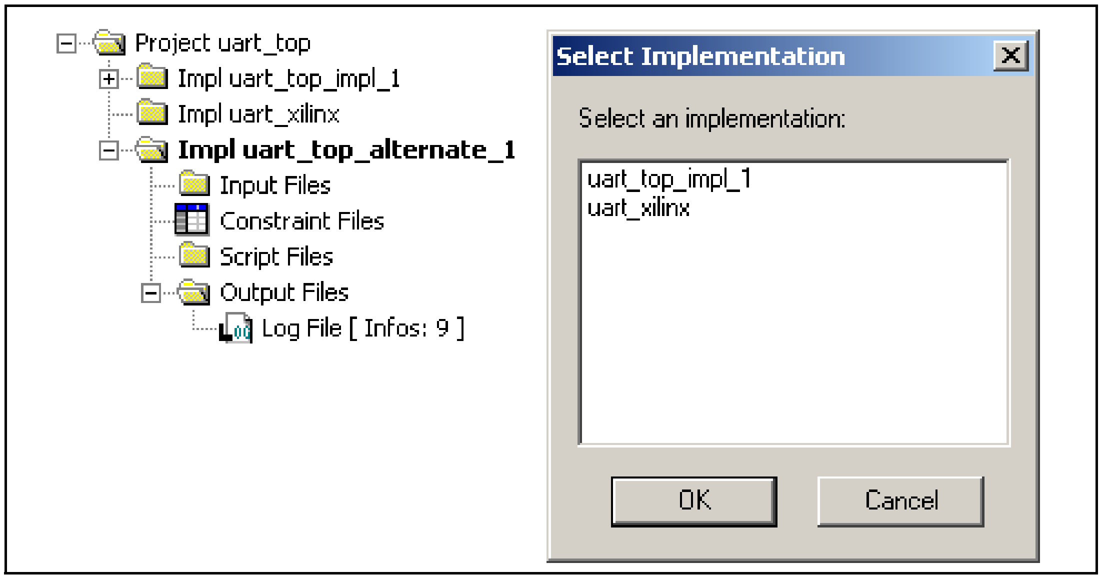
To retrieve the property values of an implementation, call the `get_impl_property` command. The following example gets the name property of the active implementation.

```
get_impl_property -name
```

Activating an Implementation

Inactive implementations are displayed in the Project Files pane of the Design Center window with a *closed folder* icon next to them, and their hierarchy is collapsed. You can expand the hierarchy and view the output files of inactive implementations by clicking on its '+' icon, but you cannot modify the contents. The easiest way to activate an implementation is to double-click on it. Alternatively you can right-click on the implementation you want to activate, and choose **Activate Implementation** from the popup menu. A third method that doesn't require you to have an implementation already selected is also available. Click on the **Activate Implementation** icon in the Project pane of the Design Bar. This command opens the **Select Implementation** dialog box (Figure 4-11) which lists the inactive implementations within the project. Select one and OK the dialog box.

Figure 4-11. Dialog Box Listing all Inactive Implementations



When you activate an implementation, Precision first closes the currently active implementation. If you have unsaved work in the active implementation, Precision prompts you to either save or discard the work. Then the implementation hierarchy is collapsed and the selected inactive implementation is opened.

Activating an Implementation from the Command Line

The following `activate_impl` command example will activate the implementation named `uart_top_impl_1` and discard any unsaved changes in the currently active implementation.

```
activate_impl -impl uart_top_impl_1 -discard
```

You can use `get_project_impls` to get a list of all implementations in the current project.

Copying an Implementation

The copy command creates and activates a new implementation that is in exactly the same state as the original implementation. The easiest way to make a copy of an implementation is to right-click on the implementation to be copied and choose **Copy Implementation** from the popup menu. Alternatively, click on the **Copy Implementation** icon in the Design Bar or the **File** >

Copy Implementation pulldown menu item to open the Select Implementation dialog box and choose an implementation to be copied.

Precision creates a new implementation of the same name as the source implementation, but increments the "_<*n*>" suffix. At the same time, a new implementation directory of the new name is created in the Project Folder, and all of the files in the source implementation are copied into the new implementation folder. The new implementation becomes the active implementation. If the currently active implementation has unsaved changes, Precision prompts you to either save or discard the changes before closing it.

Note If the source implementation name does not end with "_<*n*>", where <*n*> is an integer, Precision will append "**_1**" to the new implementation name.

Copying an Implementation from the Command Line

When calling the `copy_impl` command, if you don't specify the name for the new implementation, a default name is provided. The default name is composed of the source implementation name, but its **_<*n*>** suffix is incremented to the next higher integer that makes the name unique within the project. Alternatively, you can specify a name for the new implementation. The name you choose is arbitrary.

The following `copy_impl` command example will copy the implementation named `uart_top_impl_2` to a new implementation named uart_top_alternative. Any unsaved work in the active implementation is discarded. The new implementation is automatically activated. If you want to save the changes, call the save_impl command.

```
copy_impl -name uart_top_alternative -from uart_top_impl_2 -discard
```

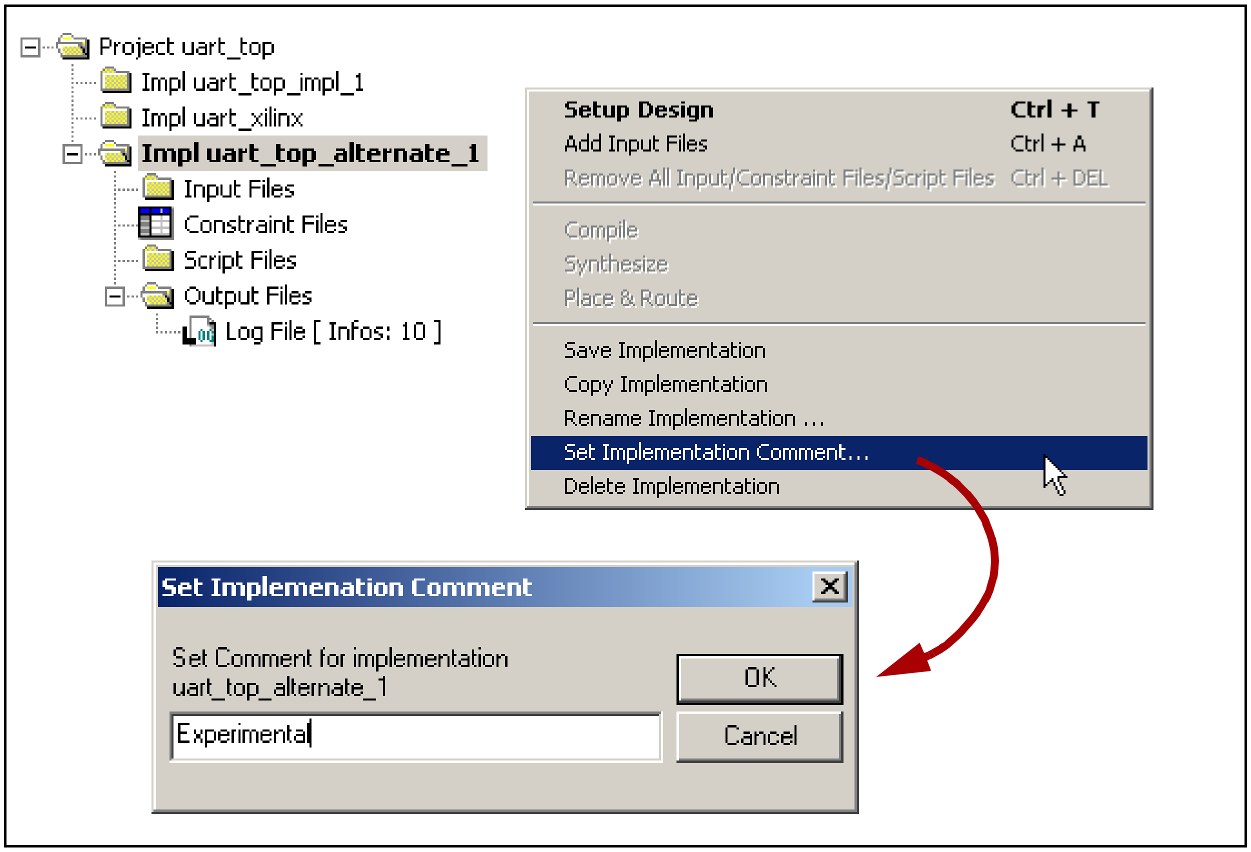
You can use `get_project_impls` to get a list of all implementation in the current project.

Commenting an Implementation

Each implementation has a comment property which takes a string value. To edit the comment property, right-click on an implementation and choose **Set Implementation Comment...** from the popup menu. Alternatively, choose the **File > Comment Implementation...** pulldown menu item to open the Select Implementation dialog box and choose an implementation, as shown in Figure 4-12.

In the Set Implementation Comment dialog, edit the comment string and click OK to set the comment property. The comment setting is immediately saved to the project implementation file in your file system. To view an implementation's comment, hover the curser over the implementation in the Project Files pane of the Design Center window and a popup box will display the comment. The popup box will display only if the comment property has been set.

Figure 4-12. Menu and Dialog for Editing Implementation Comment Property



Setting an Implementation Comment from the Command Line

To edit an implementation comment property, use the `set_impl_property` command. The following example sets the comment property of implementation `uart_top_alternate_1` to the phrase “First alternatives of uart_top_impl_1”. If you do not use the `-impl` option, `set_impl_property` operates on the active implementation.

```
set_impl_property -impl uart_top_alternate_1 -comment “First alternatative
of uart_top_impl_1”
```

To retrieve the property values of an implementation, call the `get_impl_property` command. The following command example gets the comment property of implementation `uart_top_alternate_1`.

```
get_impl_property -impl uart_top_alternate_1 -comment
```

Deleting an Implementation

To delete an implementation, do either of the following. Right-click on the implementation in the Project Files pane of the Design Center window and choose **Delete Implementation...** from the popup menu. Or choose the **File** > **Delete Implementation...** pulldown menu item to open the Select Implementation dialog box. Select and implementation and OK the dialog box. Precision prompts you to confirm the deletion before executing it. Choosing **Yes** removes the implementation from the project and immediately deletes the implementation directory from file system. Note that all files in the directory will be deleted, not just those placed there by Precision.

Deleting the active implementation leaves no implementation active. You must explicitly activate or create another implementation. If you close the project with no active implementation, it will be restored in the same state when reopened.

Deleting an Implementation from the Command Line

The `delete_impl` command deletes the specified implementation. If no implementation is specified, the active implementation is deleted. The following command example deletes the implementation `uart_xilinx`. Note that Precision does not prompt you to confirm the delete command when executed from the command line.

```
delete_impl -impl uart_xilinx
```

You can use `get_project_impls` to get a list of all implementation in the current project.

Creating a Script from a Precision.log File

A quick way to create a preliminary script is to convert the current session log file into a Tcl command file. First open the Precision Transcript window, then choose the **File** > **Save Command File...** pulldown menu. In the dialog box, specify the name of the script to be created, and click OK. Precision takes the contents of the current session log file, removes all lines that are not commands, and saves the results in the named file.

You can also invoke a log file as a script from the command line. The following 2 examples show how to invoke the session log file and an implementation log file:

```
dofile <project_dir>/precision.log

dofile <project_dir/impl_dir>/precision.log
```

You can run this file by opening the Precision Transcript window and selecting the **Run Script...** option. See the Scripts for Creating New Projects or Reusing Existing Projects section for information on scripts you can use to create new projects, reuse existing projects, and produce results.

Scripts for Creating New Projects or Reusing Existing Projects

Some users like to use scripts to automate commonly performed tasks. The following are examples of scripts that can be used to 1) create new projects, 2) reuse existing projects, or 3) create a script that combines the two methods.

Script for Creating a New Project

The sample script below clears the project and its impls and then creates the a new project and its impls. You would use this script if you intend to rebuild the project and go through the entire flow every time you run the script.This may be preferable if you don't intend to reuse the project but only intend to maintain your script. For example, if you want to set a constraint, you'll add a tcl command to do this to your script.

```
# Tcl script to clean up, create project and produce results

   # parameters
   set proj_folder c:/temp/projects
   set proj_name a_project
   set impl_name a_project_impl_1
   set input_dir c:/HDL

   # clean up
   file delete -force $proj_folder/$proj_name.psp
   file delete -force $proj_folder/$impl_name

   # create project and impl
   new_project -name $proj_name -folder $proj_folder
   new_impl -name $impl_name

   # configure settings
   set_input_dir $input_dir
   add_input_file ...
   setup_design ...
   setup_place_and_route ...

   # produce results
   compile
   synthesize
   place_and_route

   # save results to impl folder
   save_impl

   # results are now in directory $proj_folder/$impl_name/
```

Script for Reusing an Existing Project

You would use the following script if you have a project file and an impl folder and you intend to build the project and impl once and then reuse them each time. The script assumes the project exists, opens it, and then uses it. This may be preferable if you intend to maintain the project and its settings. For example, if you want to set a constraint, you open the project in the GUI, set the constraint, save the project then copy off the constraints file to a location where it can be preserved and reused by others.

```
# Tcl script to open project and produce results

   # parameters
   set proj_folder c:/temp/projects
   set proj_name a_project
   set impl_name a_project_impl_1

   # open project and activate impl
   open_project $proj_folder/$proj_name.psp
   activate_impl -impl $impl_name

   # produce results
   compile
   synthesize
   place_and_route

   # save results to impl folder
   save_impl
   # results are now in directory $proj_folder/$impl_name/
```

Script for Creating a Project and Reusing the Project

The following script combines the methods used in the previous scripts. This script creates a project if one does not exist, opens a project if it does exist, and then produces results.

```
# Tcl script to create project if necessary, open it if it exists and
produce results

   # parameters...

   # open project or create it
   if { file exists $proj_folder/$proj_name.psp } {
      # open project and activate impl...
   } else {
      # create project and impl...
      # configure settings...
   }

   # produce results...
   # save results to imple folder...
   # results are now in directory $proj_folder/$impl_name/
```

Exporting your Settings

This option exports all the project settings in your project log and saves them to a .tcl file. First open the Precision Transcript window, then choose the **File > Export Settings Script ...** pulldown menu. In the dialog box, specify the name of the script to be created, and click OK.

You can then edit the script file to modify existing implementation settings or add information for a new implementation. You can run this file by opening the Precision Transcript window and selecting the **Run Script...** option.

Actions to Take if the Project, Input Directory, or Input Files Move

The Precision RTL Synthesis tool automatically tracks the location of the project, input directory, and input files. You should follow these steps if you have moved the project location, input directory, or input files:

- **If the Project Directory Moves**

  If you move the project but the input directory has not moved, then browse to the new project directory using **File > Open** from within the Precision Synthesis tool. The project opens and the files are accessible.

- **If the Input Directory Moves**

  If you move the input directory, the Precision RTL Synthesis tool displays with the files greyed out (inaccessible). Select **File > Set Input Directory** from the File menu to set the new location for the input directory. All relative file paths will now be relative to this new location, and Precision will resolve all input files automatically.

- **If the Input Files Move**

  If the input file has been moved or renamed, it will display greyed out. If you hover over a file, the tool displays the message “file cannot be found.” Remove the file by selecting it, right-clicking to display a menu, and then select **Remove File**. Click the **Add Input Files** icon to browse to the desired input file location and select the file.

When the project is completed, the Precision RTL Synthesis tool automatically saves the project, input directory, and input files to the new locations. We recommend that you save the implementation after changing the input directory or removing then adding an input file.

Backward Compatibility with Pre-2003c Projects

Precision version 2003c introduced a number of enhancements to the Project Manager. The Project Manager is more flexible and more tightly integrated into the work flow. As described throughout this chapter Precision now provides new graphical user interface features specifically for the Project Manager, such as new command icons, menus, and dialog boxes. In addition, an assortment of new Project Manager Tcl commands are now provided.

In most cases, the Project Manager is backward compatible with pre-2003c project files and scripts. The format of project files and implementation files has changed. Now when you open a pre-2003c project, Precision automatically converts it to the new format and creates a project directory structure for it. The new project file is not a script of Tcl commands as was the old format. The new format is more like a registry of objects in the project. A good usage model with the new Project Manager is to always open (or create) a project as your first step. Once the project is open, you can run any or your scripts as you did in the past.

Your pre-2003c scripts will run, with only a few minor differences. The differences are due to modifications made to older Precision commands so that they will behave properly in the new environment. In some cases the new Project Manager commands replace functionality that was provided by old commands. The old commands (and certain options for the setup_design command) are now *deprecated*, and are labeled as such in the command reference pages in the Precision RTL Synthesis Reference Manual. Deprecation means they are being phased out. Precision continues to support them at this time, but support may be dropped in the future. The table below lists the commands that deprecated and the new replacement commands.

Table 4-1. Deprecated Commands

Deprecated	Replacement
setup_design -impl	activate_impl or set_impl_property
setup_design -impl_comment	set_impl_property -comment
setup_design -increment	copy_impl
setup_design -reset	close_project or close_results_dir
remove_design -all	close_project or close_results_dir
set_working_dir	set_results_dir or set_input_dir or cd
load_project	open_project

Many existing scripts were written to configure a design's inputs and settings, as a means of saving time and eliminating errors. Typically these script do nothing more than add input files to the design, choose technology settings, set constraints and attributes, and so on. These scripts will run just the same as they did before.

The scripts that you might want to adjust are those that begin by calling set_working_dir. The old behavior of set_working_dir was threefold; set the input directory, set the current working directory, and set the results directory to the path specified with set_working_dir. The three functions are now separate commands, and you should call each command explicitly. (You will only call set_results_dir when you want your script to run without the Project Manager.)

Furthermore, depending on the current state of Precision Synthesis, set_working_dir will behave differently. The following is a list of the three states of Precision Synthesis and the corresponding behavior of `set_working_dir`:

1. Running with Project Manager (a project is open):
   Behavior: Set the current working directory and the input directory to the specified path. (The output directory has already been set by the project.)

2. Running without Project Manager (a results directory is set):
   Behavior: Set the working directory and the input directory to the specified path. (The output directory has already been set by set_results_dir.)

3. Running without Project Manager and no results directory is set:
   Behavior: Set the current working directory and the input directory to the specified path, and create a default project and implementation in the current working directory.

The following is a simple example of how a pre-2003c script can be adjusted to use the new Project Manager commands:

Pre-2003c Script	2003c Script

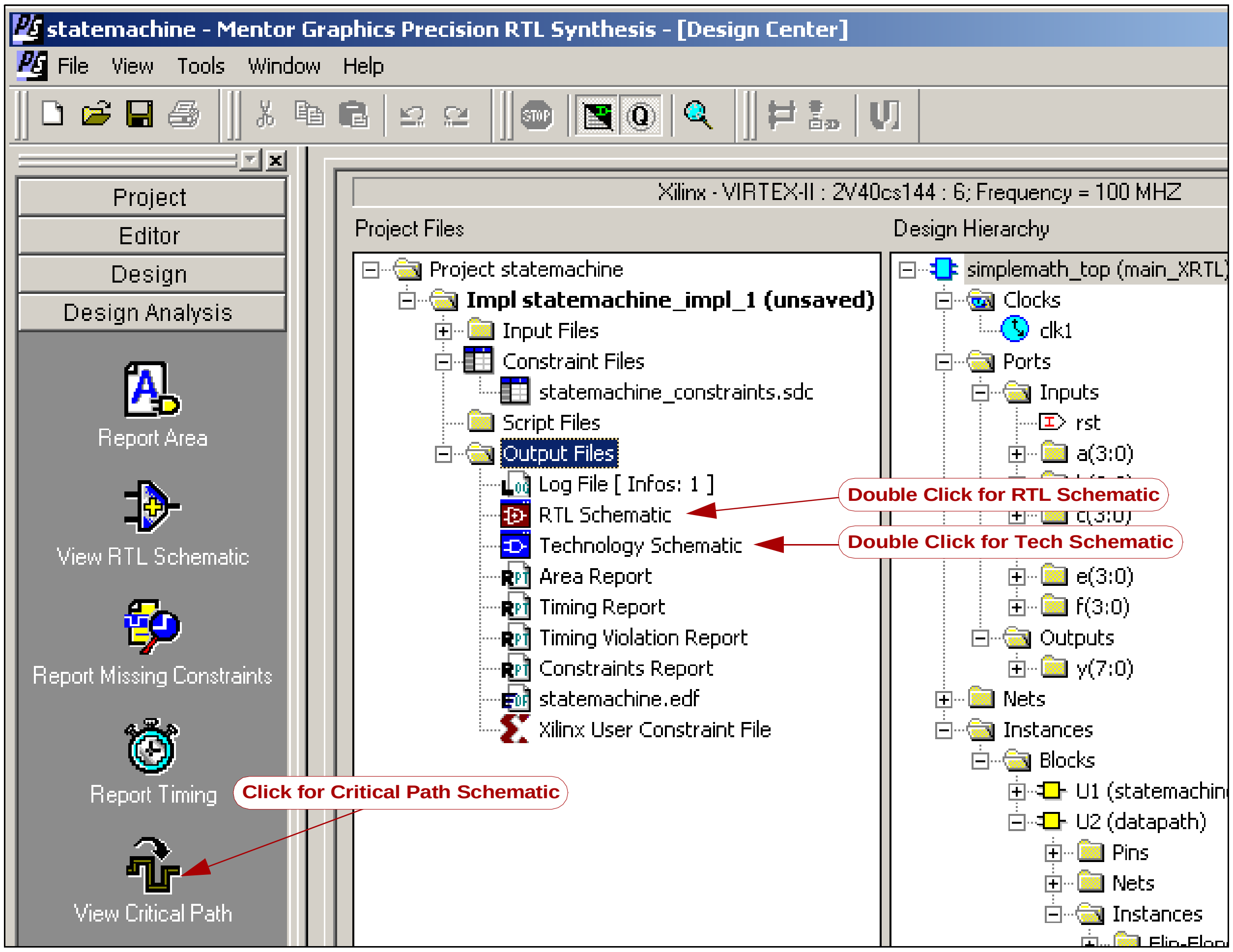
`set_working_dir {.}`
`add_input_file ...`
`setup_design ...`
`compile`
`synthesize`
`setup_design -increment`
`place_and_route`
	`new_project -name test -folder {.}`
	`new_impl -name my_test_impl_1`
	`add_input_file ...`
	`setup_design ...`
	`compile`
	`synthesize`
	`save_impl`
	`copy_impl -name my_test_impl_2`
	`place_and_route`
	`save_impl`
	`close_project`

Chapter 5
Viewing a Schematic

A Quick Way to Bring Up a Schematic

Typically, you'll bring up a number of windows as you explore the in-memory design. The figure below illustrates the quickest way to bring up views on various parts of your design, including and RTL schematic, a Technology-mapped schematic, and a Critical Path schematic.

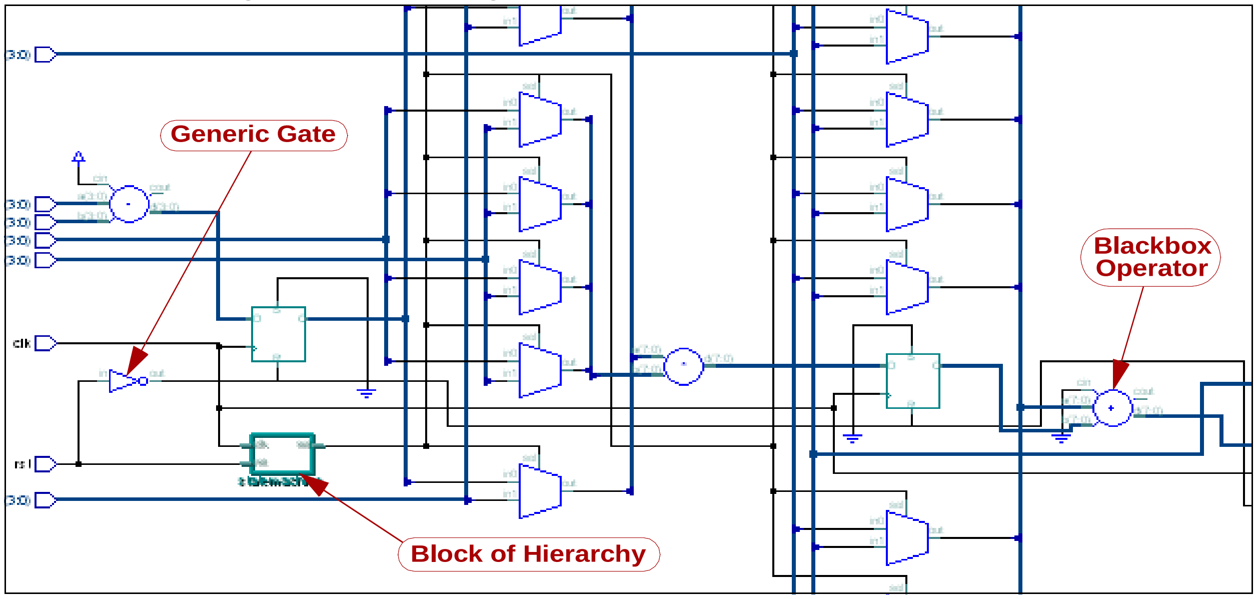
Figure 5-1. A Quick Way to Bring Up a Schematic



Viewing the RTL Schematic

When you initially compile a design, the in-memory database is created with generic gates. As shown below, if you invoke the Design Browser at this point, the RTL Schematic is a reflection of this database.

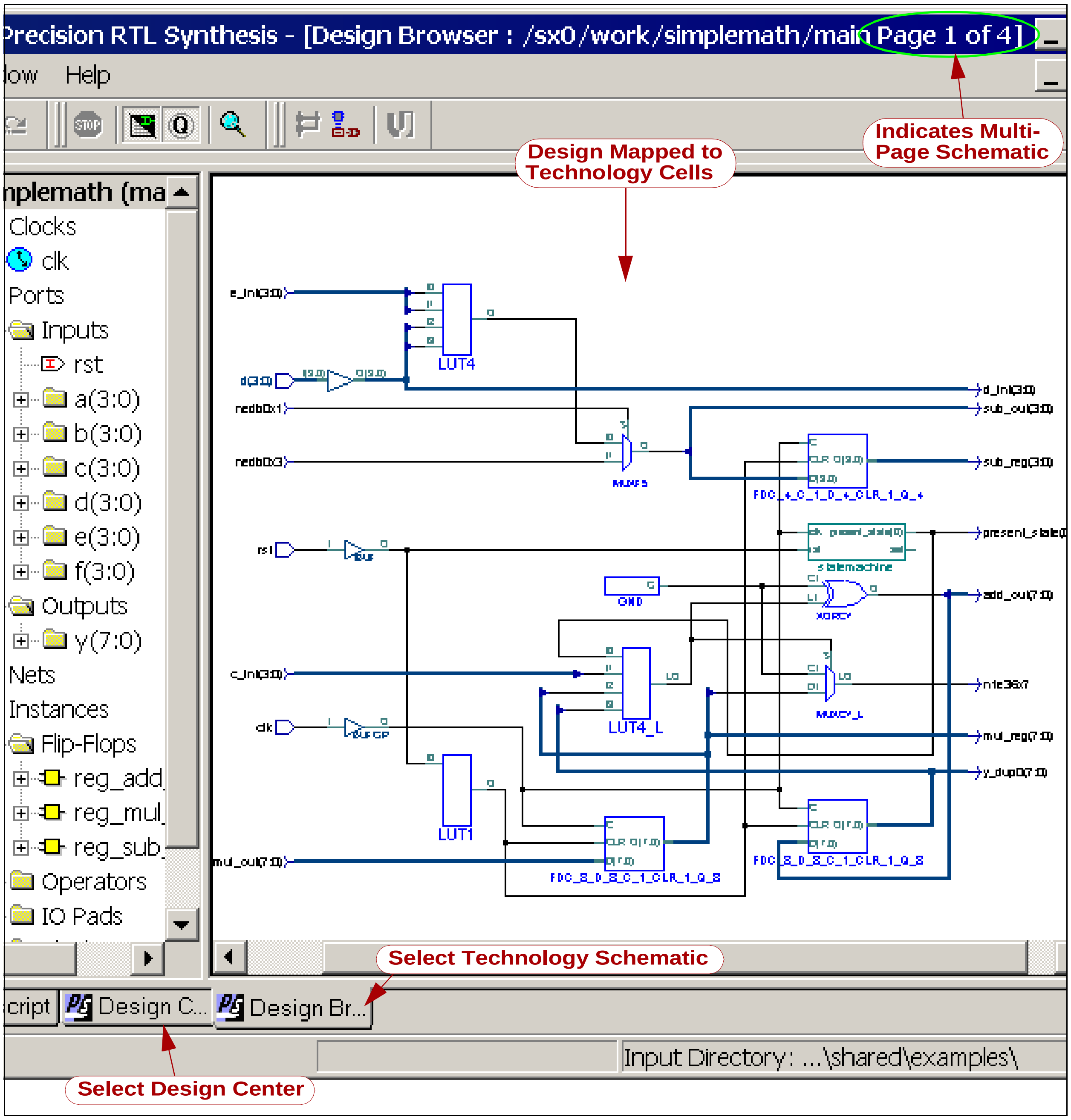
Figure 5-2. Design Browser with RTL Schematic



Viewing the Technology Schematic

After Synthesis, a technology-mapped database is created in addition to the generic database. When you double click on the Technology Schematic file, a schematic of the technology-mapped design comes up. You can choose to view either schematic by selecting the appropriate view as shown below.

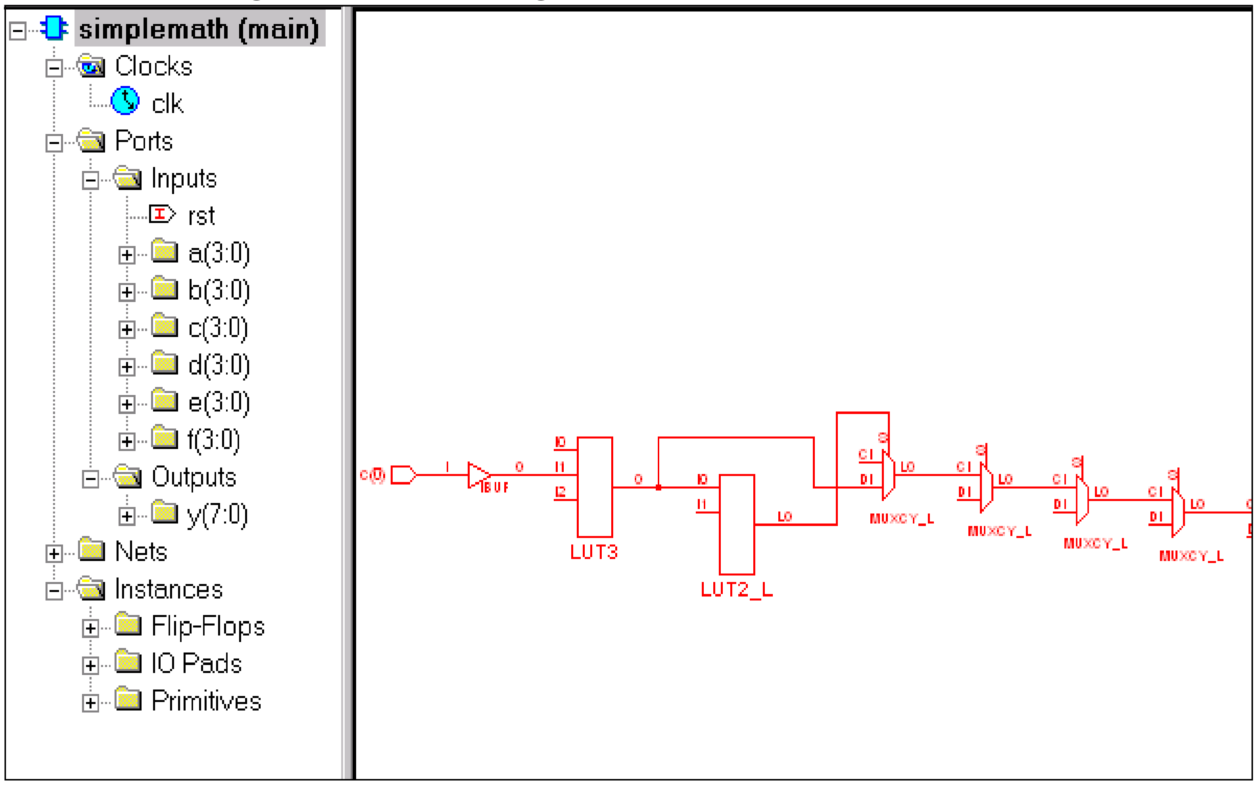
Figure 5-3. Design Browser with Technology Schematic



Viewing the Critical Path Schematic

After Synthesis, you may also view a Critical Path Schematic by clicking on the Critical Path icon in the Design Analysis pane of the Design Bar.

Figure 5-4. Viewing the Critical Path Schematic



Understanding the Left Mouse Button Actions

ZOOM TO AREA	Left Click, Drag, release - draws bounding box to zoom in on
SELECT OBJECT	Left Click on Object - selects and highlights a single object (like instance, net, pin, port)
SELECT AREA	SHIFT, Left Click, Drag, and Release - selects all non-filtered objects in the bounding box
UNSELECT ALL	Left Click in Window (not on an object)
UNSELECT OBJECT	SHIFT, Left Click on Object - unselects a single object without unselecting others

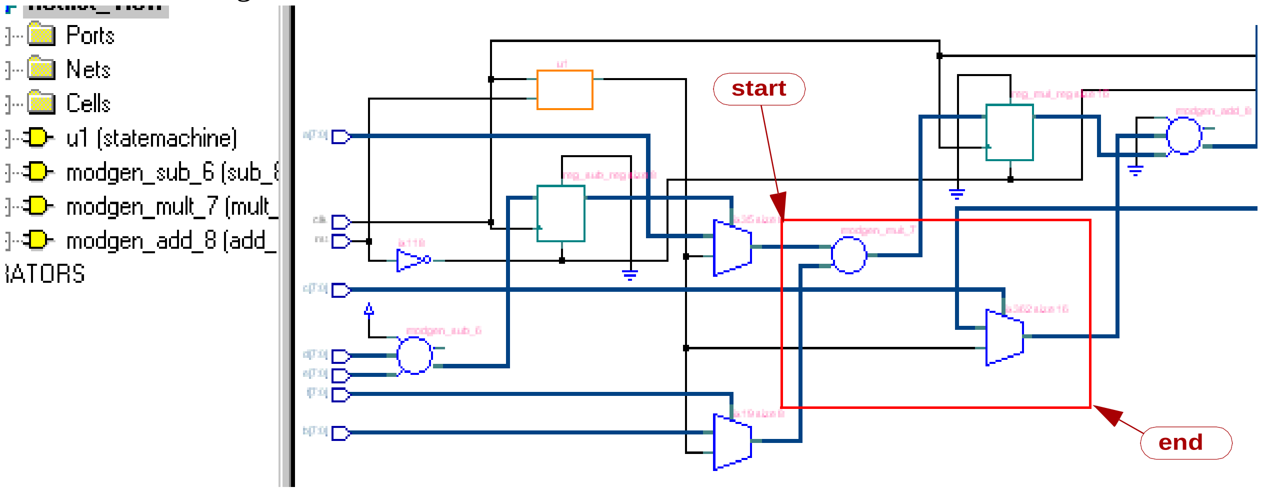
Traversing the Schematic with Strokes

You can click on the Design Bar icons to accomplish many of the actions described in this section. However, many people find strokes the fastest and most convenient method of traversing a schematic.

A stroke is an action where you press the Left Mouse Button, draw a line or bounding box, then release the button. The direction of the line indicates what action you want to take. For zooming strokes, the length of the line controls the amount of zoom. The longer the line, the more zoom.

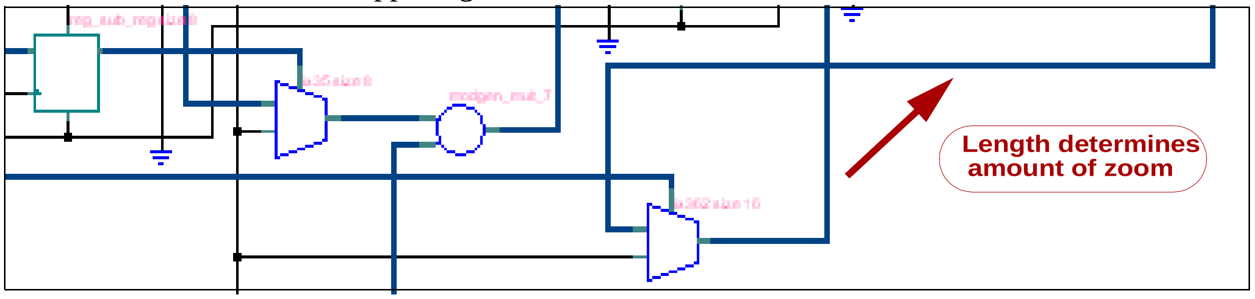
Zooming to Area

Draw a bounding box with the Left Mouse Button



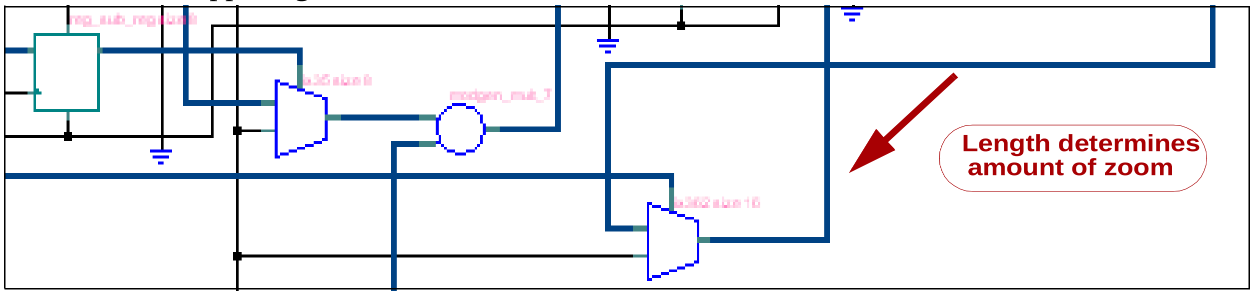
Zooming Out

Draw a line, lower left to upper right.



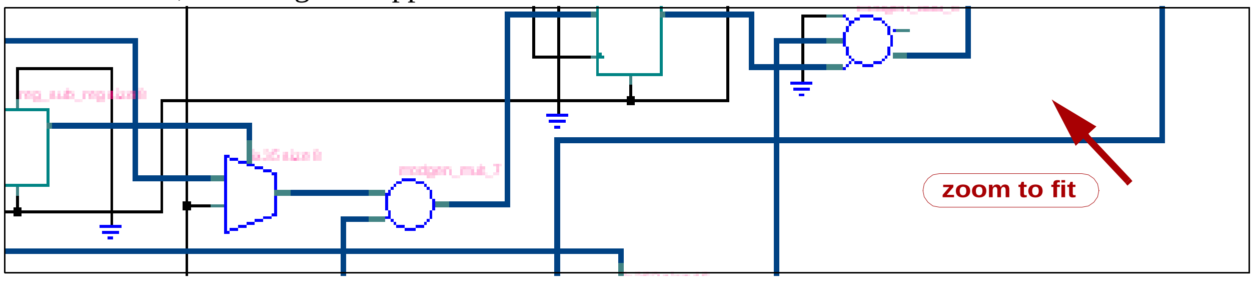
Zooming In

Draw a line, upper right to lower left.

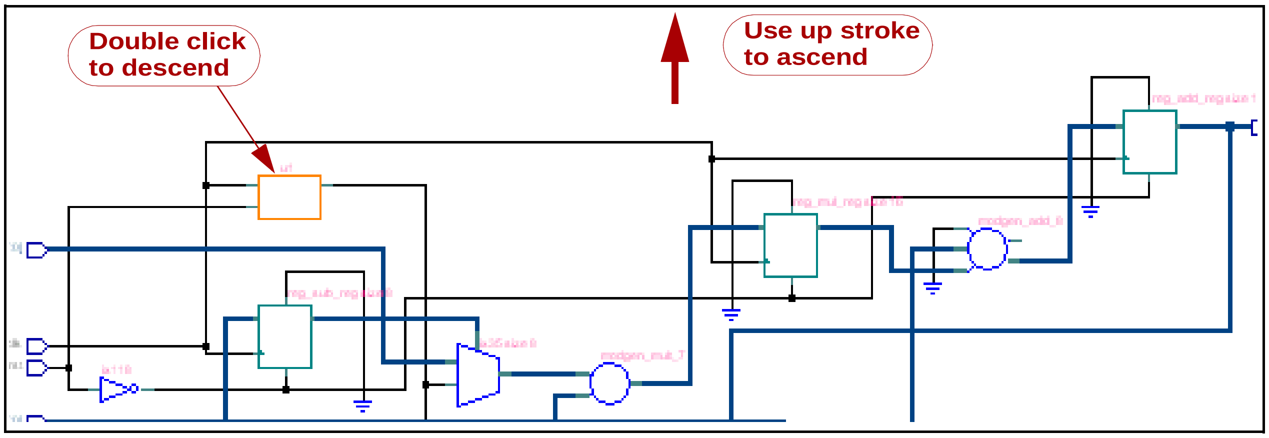


Zooming to Fit

Draw a line, lower right to upper left



Moving Up and Down the Hierarchy

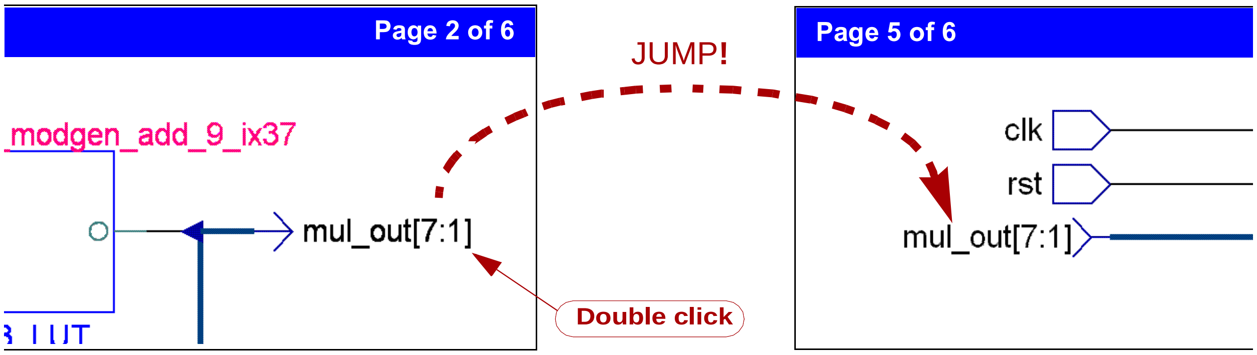


Paging Forward (→) and Paging Back (←)

If you have a multi-page schematic, you can move to the next page by using a horizontal stroke → (left to right), then move back to the previous page with a horizontal stroke ← (right to left). If there is a multi-page schematic, the toolbar indicates page 1 of x, where x indicates the number of pages in the schematic. See Figure 5-3 for an example of how the number is displayed for multi-page schematics.

Tracing a Signal to the Next Page

When you are viewing a page in a multi-page schematic, you can double click on an output port and the viewer will jump to the connected input port on the next page.



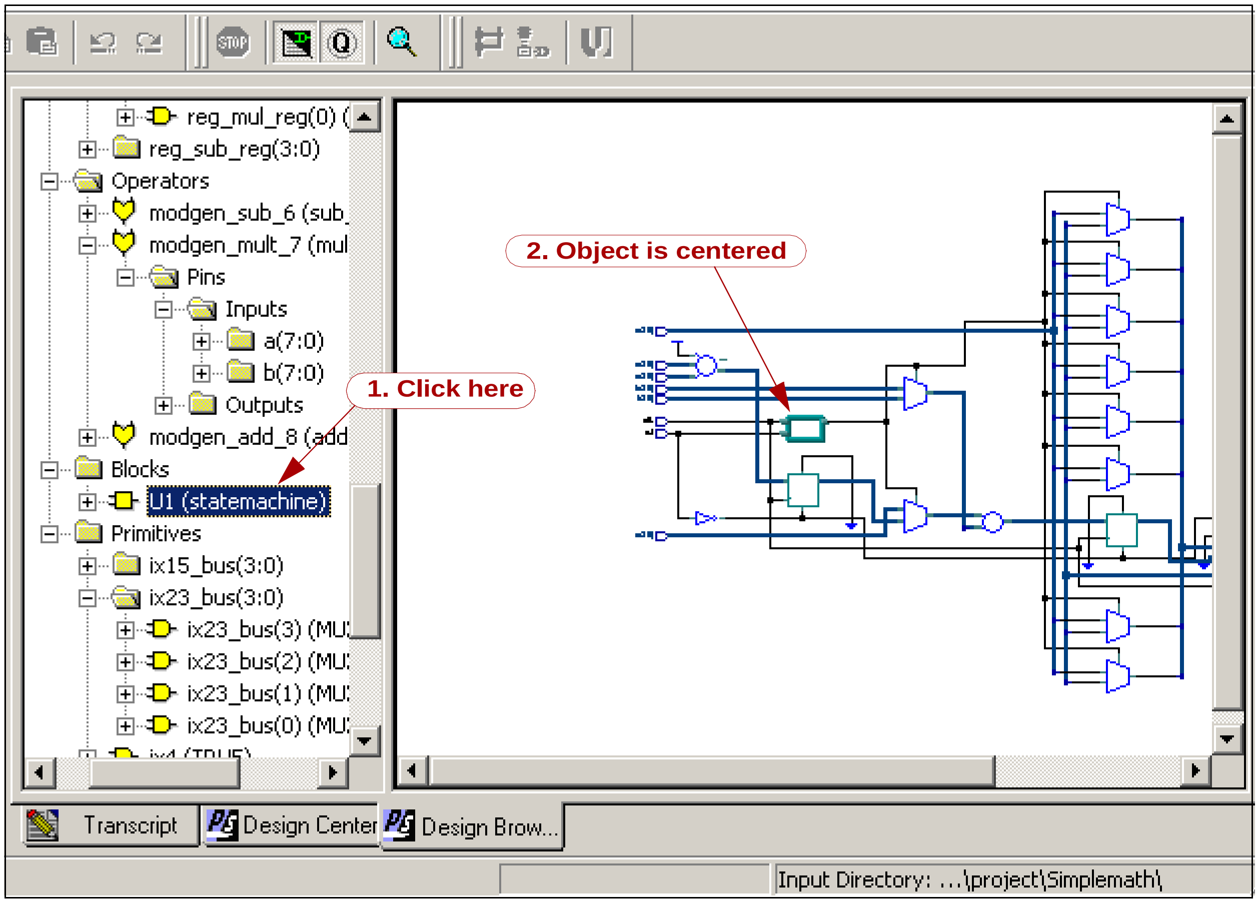
Panning the Schematic

There are two ways to pan the schematic:

1. Press the center mouse button on a 3-button mouse and drag the schematic around underneath the window. (Press and hold both buttons on a two-button mouse.)
2. Use the side scroll bars to change the view

Centering an Object in the Schematic View

When you double click on an object in the Hierarchy pane, the design object is centered in the Schematic pane.

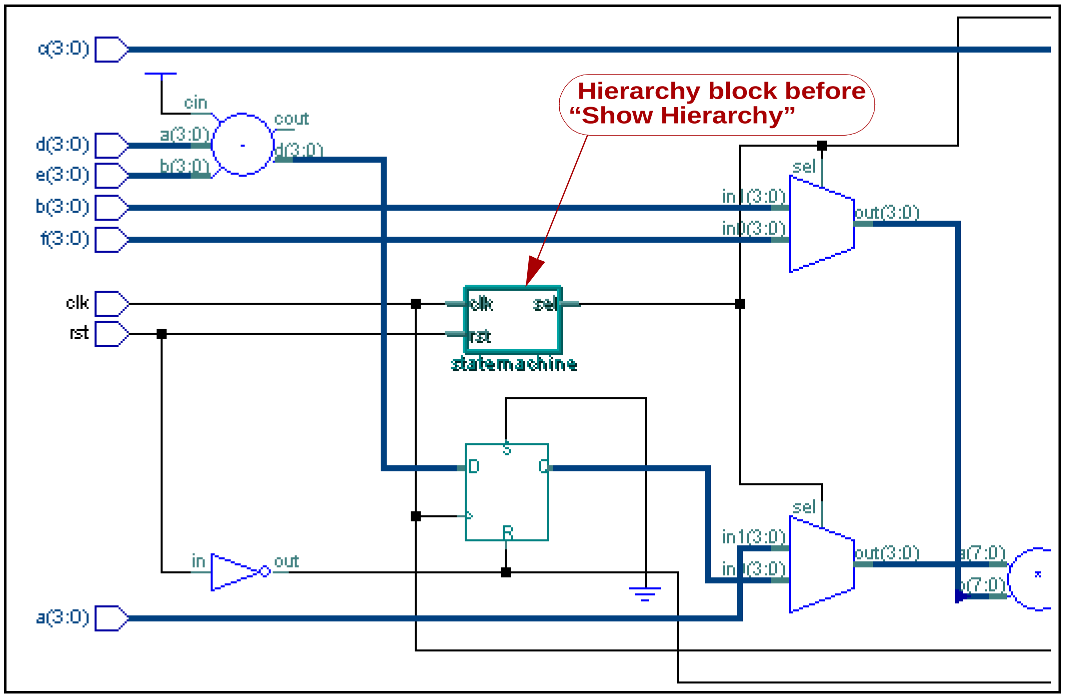


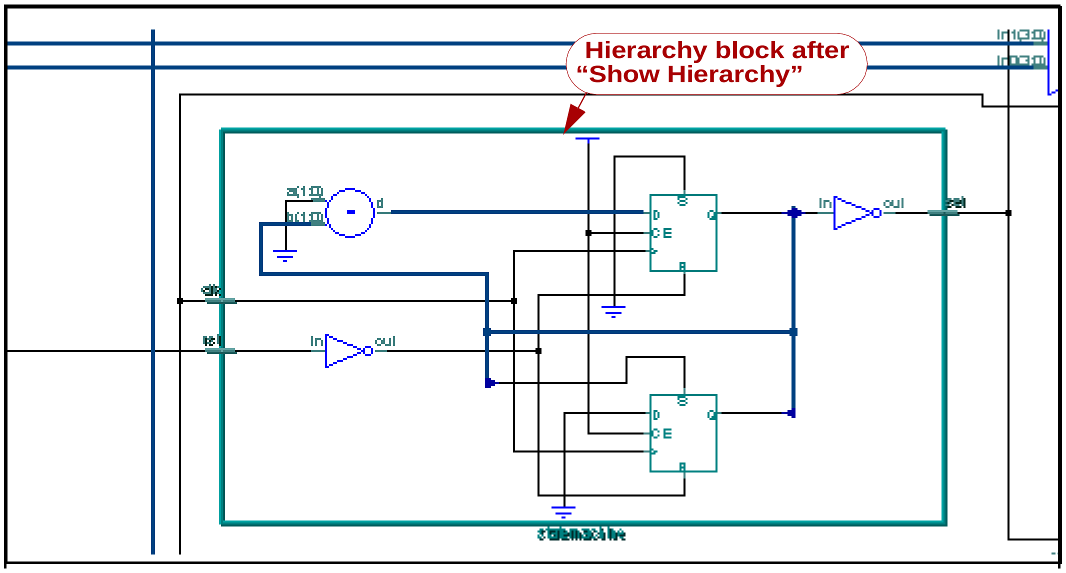
Viewing the Internals of a Hierarchy Block

The Schematic Viewer has the capability to display the internal schematic of a block in the context of the current schematic.

To display the internal schematic, do the following:

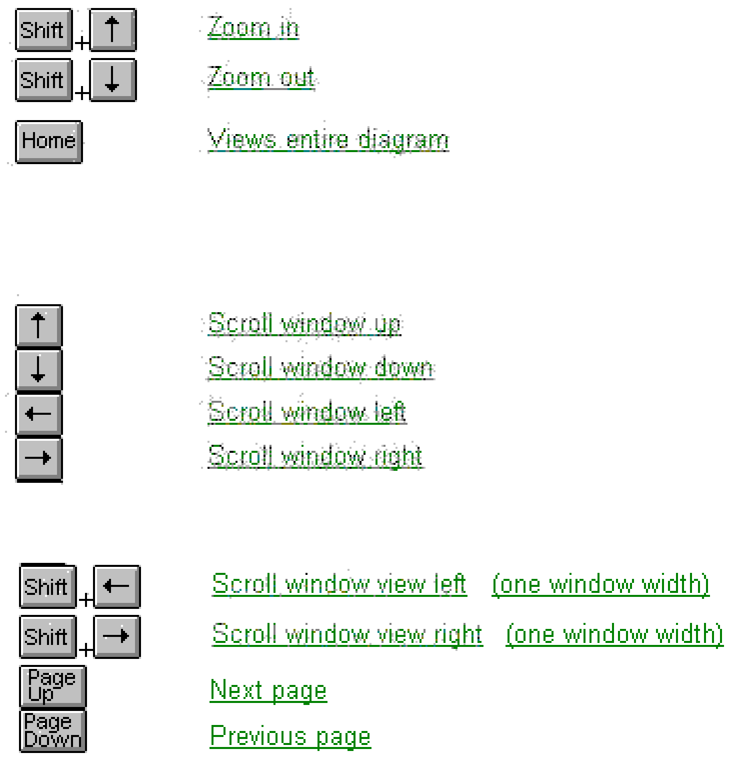
1. Right click anywhere in the schematic window and select **Show Hierarchy**





Traversing the Schematic Using Keyboard shortcuts

You can click on the Design Bar icons to accomplish many of the actions described in this section. However, you can also use the keyboard to traverse a schematic. The following keyboard shortcuts for Precision's Schematic Viewer are based on the shortcuts from the Block Diagram editor in HDLDesigner.



Shift + ↑	Zoom in
Shift + ↓	Zoom out
Home	Views entire diagram
↑	Scroll window up
↓	Scroll window down
←	Scroll window left
→	Scroll window right
Shift + ←	Scroll window view left (one window width)
Shift + →	Scroll window view right (one window width)
Page Up	Next page
Page Down	Previous page

Using the Cross Probe Features

Cross Probing from Schematic to the HDL Source

1. Select an object on the schematic (like an INV instance)
2. Right click, then select **Trace to HDL Source**

The HDL source comes up highlighted in an edit window. At this point you can edit the source, save, and re-Read the file to correct errors. If changes cause the schematic to be “grayed-out”, just click in the Schematic window, right click, then select Reload Schematic.

NOTE: Not all objects initiate cross probing. The following is a list of objects that do initiate cross probing:

1. Instances
2. IF statements and Conditional Signal Assignment
3. CASE statements and selected signal assignment
4. Expressions with arithmetic, boolean, or logical operators
5. Declaration statements
6. Procedure calls
7. Variable array indexing

Cross Probing to HDL Designer from Precision RTL Synthesis

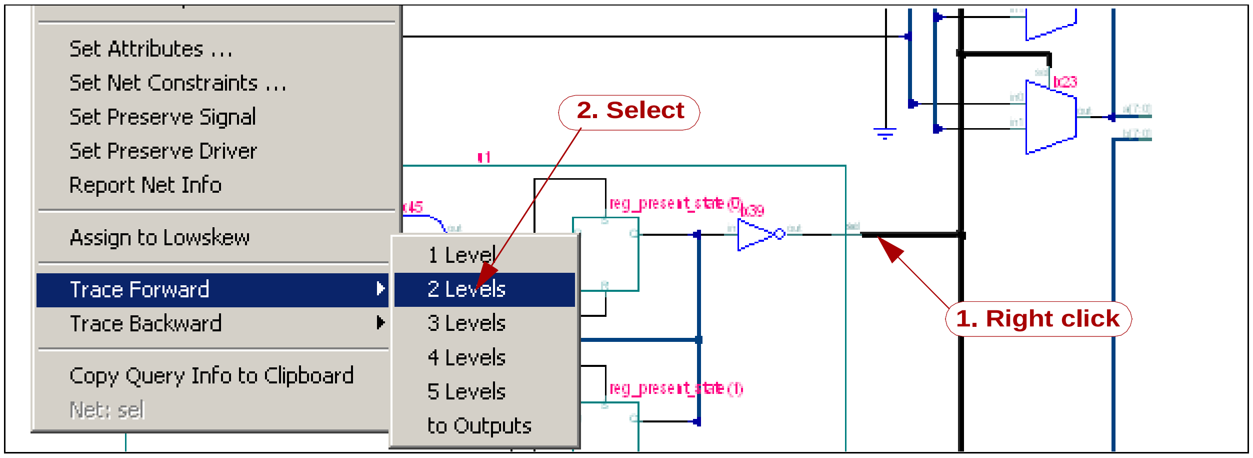
1. Open a schematic in Precision RTL Synthesis that was created in HDL Designer
2. Click on an instance and from the RMB menu select Trace to HDL Designer

   The same instance will highlight in the HDL Designer schematic.

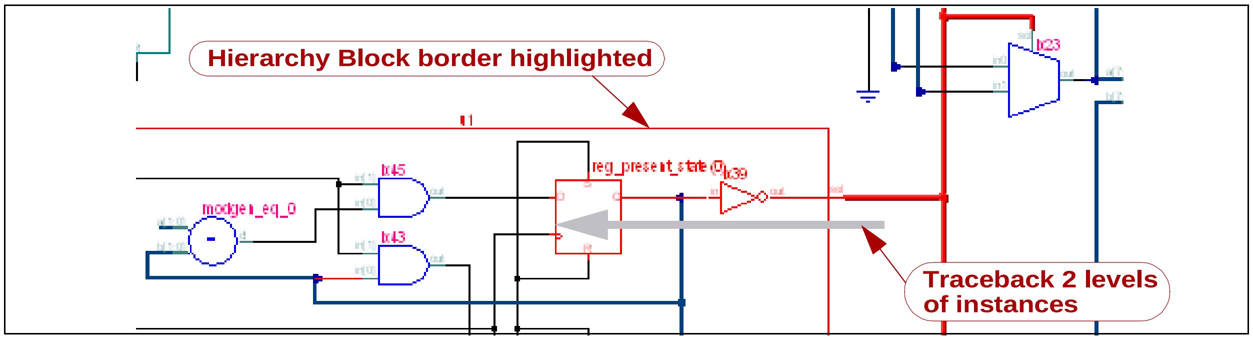
Viewing Schematic Fragments

Tracing a Signal

Schematic viewing allows you to select a signal and trace that signal forward or backward through one or more levels of instances. Once traced, you can View Trace Schematic and bring up a schematic view that only contains the traced elements. Examine the following procedure:

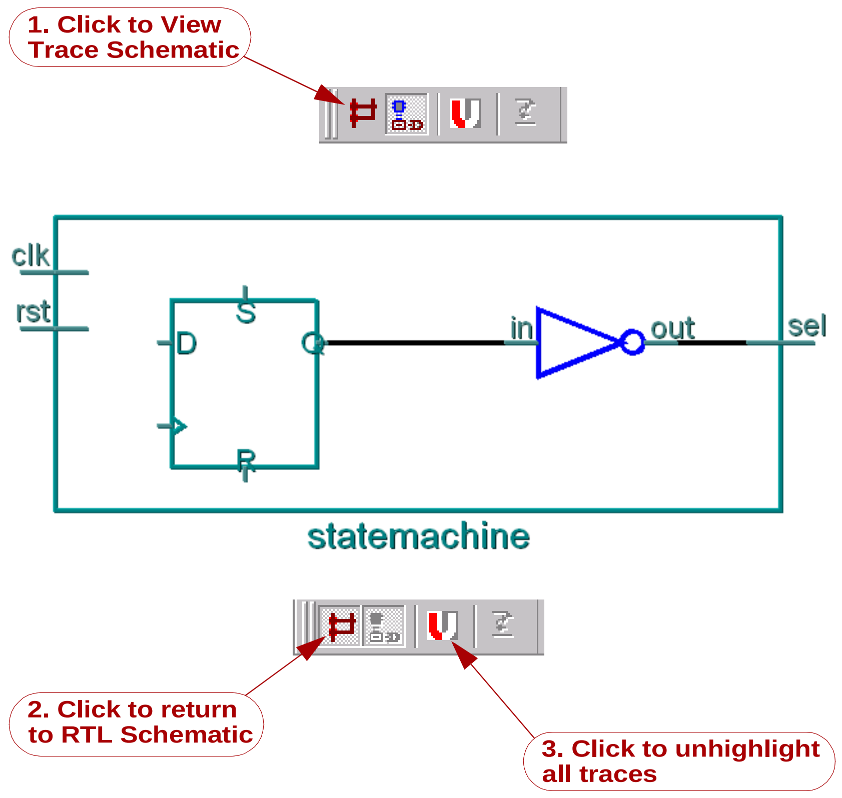


In the frame above, the `sel` net is selected and the signal is traced back two levels of instances. The results are shown below. Notice that the signal and the first two levels of instances are highlighted in orange. Notice also that the signal crosses a hierarchy block, so the border of that block is highlighted.


Hierarchy Block border highlighted
Traceback 2 levels of instances
reg_present_state(0)
ix39
ix23
ix45
ix43
modgen_eq_0

Viewing a Trace Schematic

After you have traced a signal one or more levels, you can view just the traced elements by clicking the View Trace Schematic icon on the Standard Schematic toolbar or you can right click and select View Trace Schematic from the menu. The results are shown below:


1. Click to View Trace Schematic
clk
rst
S
D
Q
R
in
out
sel
statemachine
2. Click to return to RTL Schematic
3. Click to unhighlight all traces

Click and Sprout the Trace Path

You may want to examine the fan in and fan out of a block in more detail.

Use the menu pick method:

1. Select an object in the trace path.
2. Right click and select Trace Forward or Trace Backward (one or more levels). Each time you do this, you will see the circuitry around the object grow and grow.

Use the double-click method

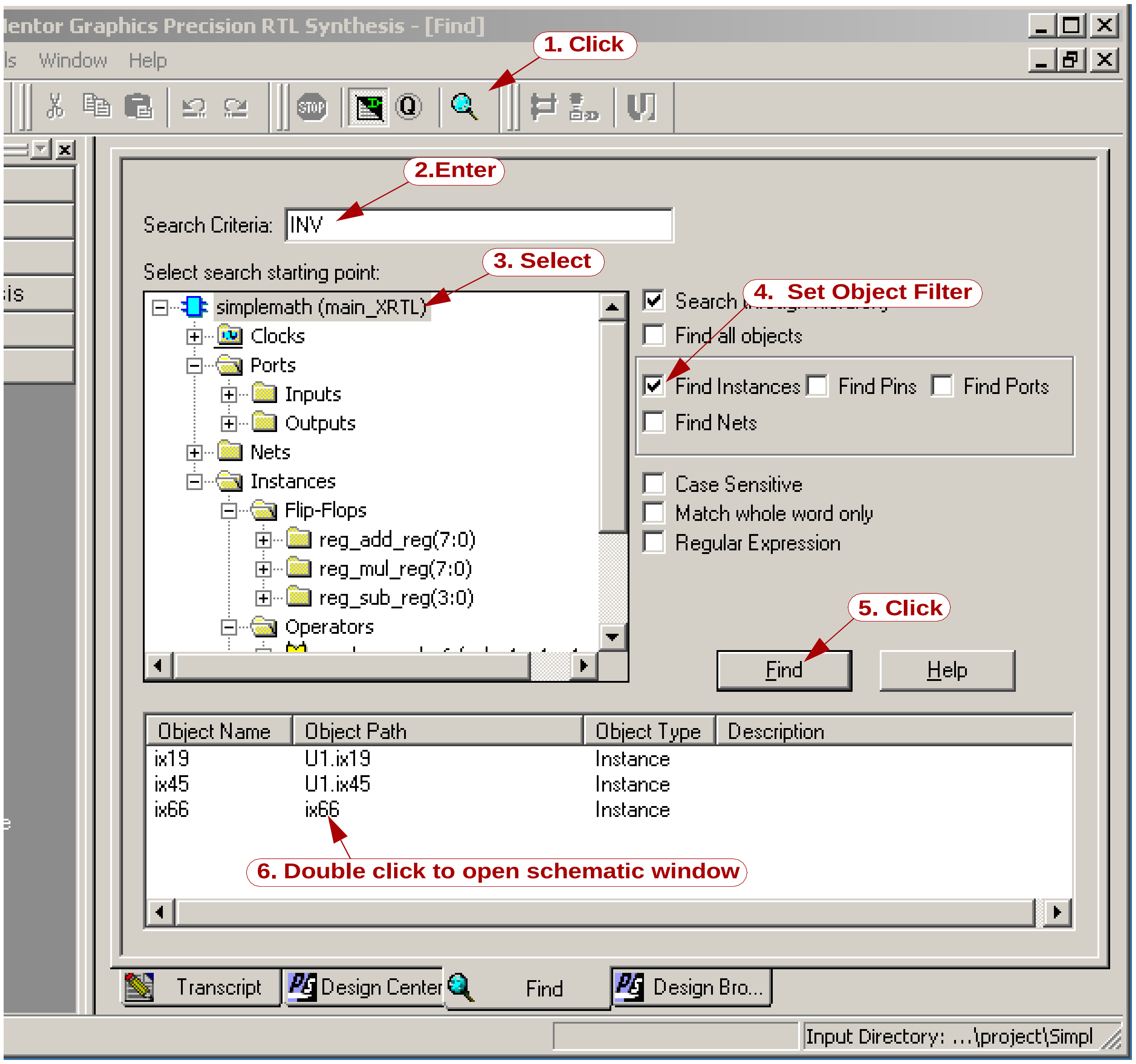
Double-click on a primitive to trace back one level

- Shift-double-click on a primitive to trace forward one level

Using the Find Window Features

Conducting a Simple Search for Objects

The Find Window allows you to search the in-memory database for the name(s) of one or more objects. When the objects are found, a list of object pathnames are displayed at the bottom of the window. This list is active and by double clicking on a pathname, you can view the object in a schematic window. The graphic below provides an example of searching the *statemachine* block of this design for all the inverters.


lentor Graphics Precision RTL Synthesis - [Find]
ls Window Help
1. Click
2.Enter
3. Select
4. Set Object Filter
5. Click
6. Double click to open schematic window
Find
Help
Transcript
Design Center
Find
Design Bro...
Input Directory: ...\project\Simpl

Search Criteria: INV

Select search starting point:

- simplemath (main_XRTL)
  - Clocks
  - Ports
    - Inputs
    - Outputs
  - Nets
  - Instances
    - Flip-Flops
      - reg_add_reg(7:0)
      - reg_mul_reg(7:0)
      - reg_sub_reg(3:0)
    - Operators

[✓] Search through hierarchy
[ ] Find all objects
[✓] Find Instances [ ] Find Pins [ ] Find Ports
[ ] Find Nets
[ ] Case Sensitive
[ ] Match whole word only
[ ] Regular Expression

Object Name	Object Path	Object Type	Description
ix19	U1.ix19	Instance	
ix45	U1.ix45	Instance	
ix66	ix66	Instance	

The sequence is as follows:

1. Knowing that inverter primitives are named INV, enter INV in the Search Criteria entry box.

2. Select the *statemachine* object as the place to start the search.

3. Set the search filters to only look for instances.

4. Click FIND

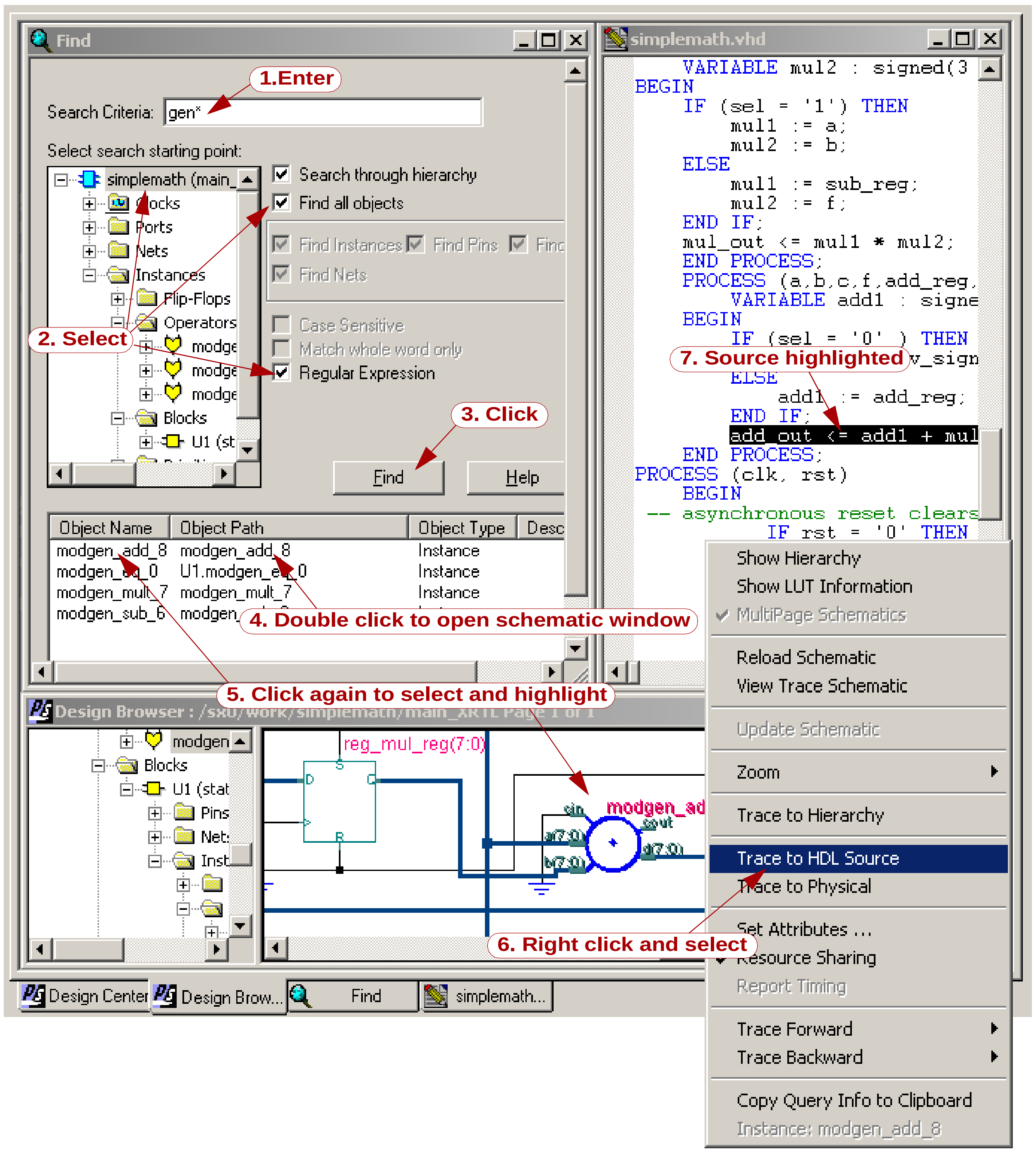
The Find function returns the pathnames of two inverters in the *statemachine* block. At this point, you can double click on an object's pathname to view it in a schematic window. If you select the object again, it will highlight in the schematic window.

Searching for Objects Using Regular Expressions

The concept of regular expressions in the Tcl language is a powerful pattern matching capability. You can think of a regular expression as a template that matches a group of strings. A regular expression can contain special characters that represent subparts of a string. For example, if the wildcard asterisk character (*) is placed at the beginning of an expression, the asterisk represents any number of characters preceding it; if placed at the end of the expression, the asterisk represents any number of characters to the end of the string. To illustrate, the regular expression ***or*** tells the find function to find all object names that contain the substring **or** no matter how many characters come before or after the substring.

NOTE: You can find a complete description of regular expressions and the special characters they contain in a Tcl reference manual.

The graphic below illustrates a regular expression search for all the generated operators in the design. Because every operator's name contains the string "modgen", you can search using the regular expression ***gen***



Object Name	Object Path	Object Type	Desc
modgen_add_8	modgen_add_8	Instance	
modgen_add_0	U1.modgen_add_0	Instance	
modgen_mult_7	modgen_mult_7	Instance	
modgen_sub_6	modgen		

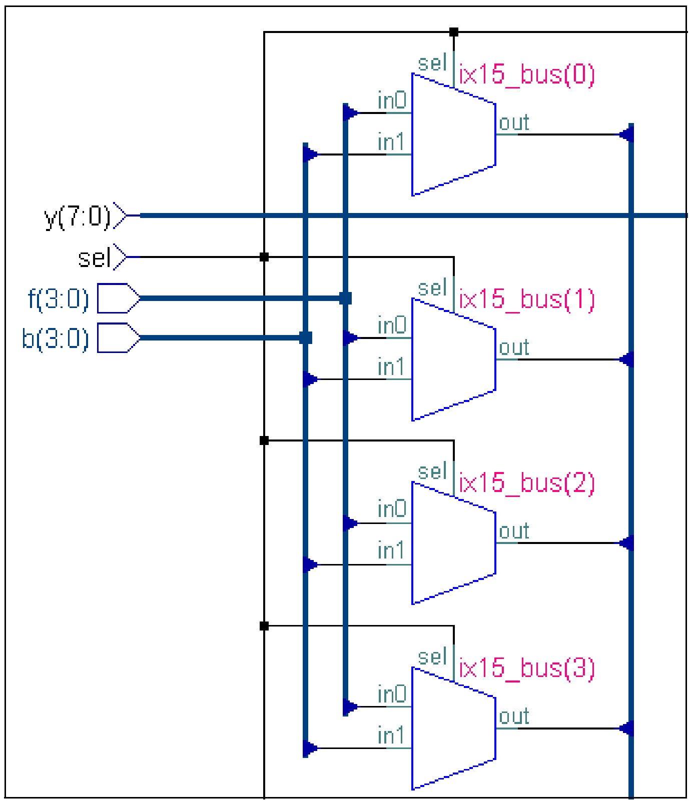
```
    VARIABLE mul2 : signed(3
BEGIN
    IF (sel = '1') THEN
        mul1 := a;
        mul2 := b;
    ELSE
        mul1 := sub_reg;
        mul2 := f;
    END IF;
    mul_out <= mul1 * mul2;
    END PROCESS;
    PROCESS (a,b,c,f,add_reg,
        VARIABLE add1 : signe
    BEGIN
        IF (sel = '0' ) THEN
                               v_sign
        ELSE
            add1 := add_reg;
        END IF;
        add_out <= add1 + mul
    END PROCESS;
PROCESS (clk, rst)
    BEGIN
  -- asynchronous reset clears
        IF rst = '0' THEN
```

Setting Schematic Viewing Options

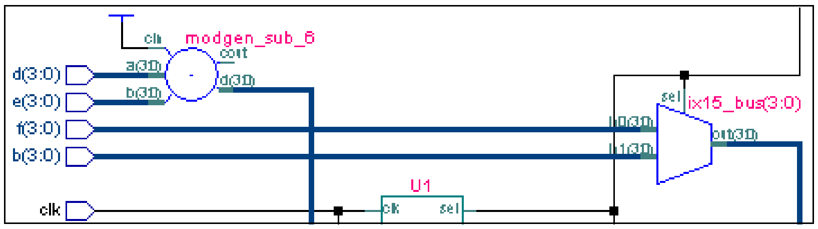
Understanding the Term "Bused Instance"

The term "bused instance" refers to an instance that is connected in parallel with one or more identical instances. Each bused instance must be a group of instances of the same cell and the same view, and each pin on the instance must be connected in parallel to the same scalar net or net array (bus) as the other instances in the group. The following illustration shows a group of four multiplexors connected in parallel. Each instance in the following illustration is a "bused instance". This group of four bused instances are "bundled" in the second illustration.

Bused Instances that are "unbundled"



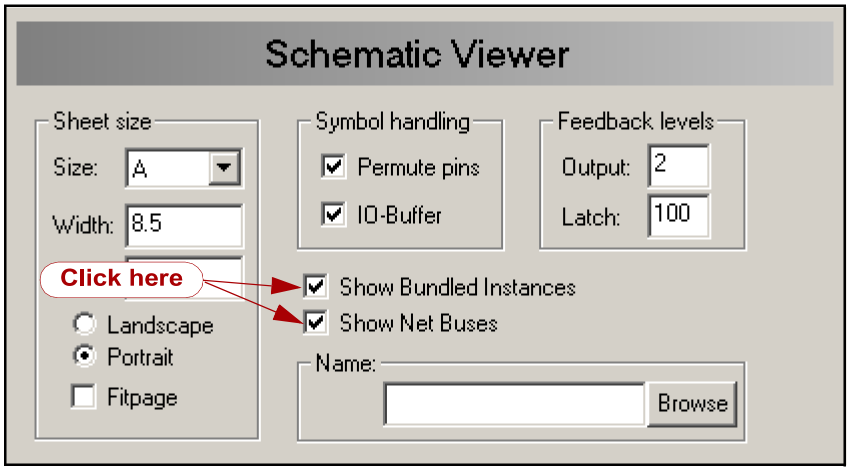
Bused Instances that are "bundled"



Viewing Bundled Instances and Net Buses

By default, bused instances are automatically bundled and buses are displayed as a bold line to make viewing easier. You can turn off the bundling feature by following these steps:

1. From the Menu Bar, select **Tools** > **Set Options...**
2. Select the **Schematic Viewer**
3. Click off **Show Bundled Instances** and/or **Show Net Buses**
4. Right click and select **Reload Schematic**


Schematic Viewer

Sheet size
Size: A
Width: 8.5
( ) Landscape
(•) Portrait
[ ] Fitpage

Symbol handling
[✓] Permute pins
[✓] IO-Buffer

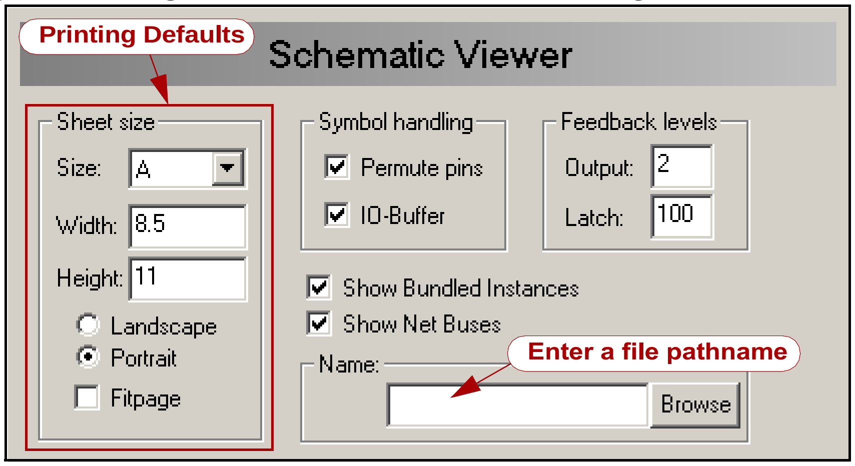
Feedback levels
Output: 2
Latch: 100

[✓] Show Bundled Instances
[✓] Show Net Buses

Name:

Changing the Schematic Printing Defaults

1. From the Menu Bar, select Tools > Options...
2. Select the **Schematic Viewer**
3. Change the Printing Defaults are shown in the following illustration:


Schematic Viewer

Sheet size
Size: A
Width: 8.5
Height: 11
( ) Landscape
(•) Portrait
[ ] Fitpage

Symbol handling
[✓] Permute pins
[✓] IO-Buffer

Feedback levels
Output: 2
Latch: 100

[✓] Show Bundled Instances
[✓] Show Net Buses

Name:

4. Enter a file pathname to save the schematic to a file.

Changing the Way Symbols are Handled

The Symbol handling options allow you to control the viewers ability to do the following:

Permute pins

Switch the placement of pins on a symbol to reduce net congestion. The default is true. You may want to turn this off if you are using a symbol standard that doesn’t allow the changing of pin positions.

IO-buffer

Specifies that IO buffers should be placed in the outside edges of the schematic sheet. The default is true.

Changing the Placement of Output Devices

The Feedback levels dialog controls the placement of output devices on the right side of the schematic sheet. Forcing output devices to the right side may make the schematic easier to read, but may also increase the page count of a multi-page schematic.

Output

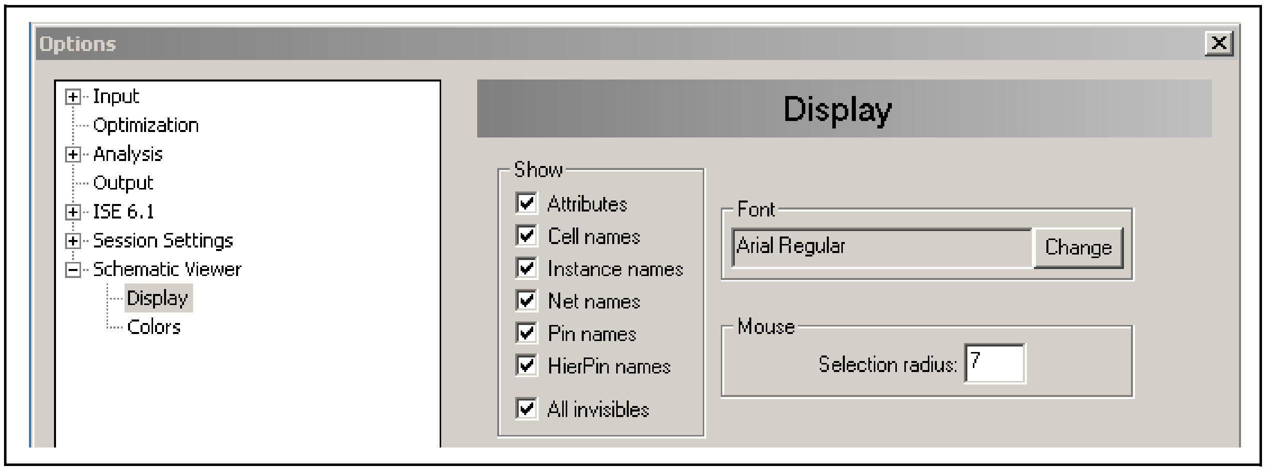
Specifies the number of logic levels from the right side where output drivers (components directly connected to an output connector) are placed. The number of levels can be specified from 0 to 500 and defaults to 2.

Latch

Specifies the number of logic levels from the right side where latches are placed. The number of levels defaults to 100 (effectively disabled).

Changing the Viewer Display and Color Options

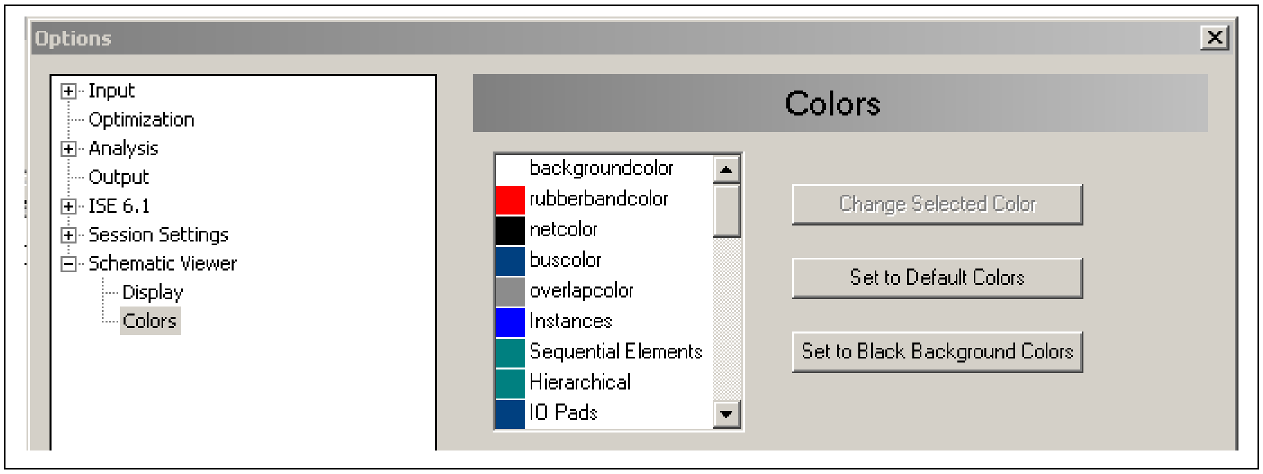
As shown below, the Schematic Viewer Display options dialog allows you to filter out certain objects from the schematic view, change the font style and specify the mouse selection radius. The Color options dialog allows you to change the schematic color scheme.



Show
- [✓] Attributes
- [✓] Cell names
- [✓] Instance names
- [✓] Net names
- [✓] Pin names
- [✓] HierPin names
- [✓] All invisibles

Font: Arial Regular

Mouse
Selection radius: 7



⬜	backgroundcolor
🟥	rubberbandcolor
⬛	netcolor
🟦	buscolor
▨	overlapcolor
🟦	Instances
🟩	Sequential Elements
🟩	Hierarchical
🟦	IO Pads

Index

.sdc File
  generated, 2-4
.ucf File, 2-17

A
Activating an Implementation, 4-13
add_input_file command, 2-7
Adding IO Pads, 3-19
Adobe Acrobat Reader, 1-8
Altera
  mapping blocks to LogicLock regions, 3-21
  memory mapping, 3-21
Asynchronous signals, detecting, 2-20
Attributes
  buffer_sig, 3-19
  dont_touch, 2-11, 2-18
  false_path, 3-16
  hierarchy, 3-19
  nopad, 3-19
  VHDL, 2-18
Auto-Top detection, 2-10

B
buffer_sig attribute, 3-19
Buffers
  global, 3-20

C
Case Sensitivity, 2-10
Clocks oOverview, 3-3
Command Line
  entering constraints, 2-12
Commands
  add_input_file, 2-7
  compile, 2-13
  get_lib_ports, 2-15
  move_input_file, 2-7
  precision, 2-1
  set_attribute, 3-18, 3-19
  set_false_path, 3-16
  set_input_delay, 3-11
  set_input_file, 2-7
  set_output_delay, 3-14
  setup_design, 2-6
Commands for derived clocks, 3-6
Commenting an Implementation, 4-15
Compatibility with Pre-2003c Projects, 4-21
compile command, 2-13
Constraints
  conflicts between files, 2-18
  Incorrect, 2-21
  input delay, 3-11
  ouput delay, 3-14
  reading from UCF file, 2-17
Copying an Implementation, 4-14
Counter Inferrencing, 3-21
  Inferring
    Counters, 3-21
Creating a new Implementation, 4-10
Creating a New Project, 4-3

D
Deleting an Implementation, 4-17
Derived clocks, 3-5
Derived clocks commands, 3-6
Derived Clocks within the Graphical User
    Interface, 3-5
Design Bar, 2-2
Design Browser
  display options, 5-21
Design Center Window, 2-2
Design Files
  opening, 2-6
dont_touch attribute, 2-11, 2-18

E
Encoding
  state machines, 2-12
Excluding

Index (cont.)

files from Compile phase, 2-8
Exporting your Settings, 4-20

F
False Paths
setting constraints, 3-15
false_path attribute, 3-16
Fastest Design
strategy for obtaining, 2-20
Files
.sdc, 2-4
.ucf, 2-17
in the project directory, 4-5
passing from input to output, 2-8
Find Window
simple search, 5-15
Finding Objects
using regular expressions, 5-16
FSM
encoding, 2-12
safe, 2-12
FSM Options, 2-12
Full Case
Verilog, 2-11

G
Generating Reports, 2-20, 2-22
get_lib_ports command, 2-15
Global Buffers, 3-20

H
help
online, 1-7
hierarchy attribute, 3-19

I
I/O Mapping, 3-18
IEEE Libraries
auto-loading, 2-10
Implementation
Activating, 4-14
Commenting, 4-15
Copying, 4-14
Creating, 4-10
Deleting, 4-17
Renaming, 4-13
Saving, 4-11
Include Files
specifying search path for, 2-9
Verilog, 2-11
Inferring
counters, 3-21
operators, 3-21
pipelined multipliers, 3-21
Input Delay
constraint, 3-11
setting the global default, 2-6
Input Directory Moves, 4-20
Input Files
adding to the project, 2-6
excluding from Compile phase, 2-8
specifying properties of, 2-8
types, 2-8
Input Files Move, 4-20
Interactive Command Line Shell
how to bring up, 2-3
Introducing, 1-1

L
Latches, transparent, 2-14
Library Clauses, 2-10
Loading
a technology library, 2-5
LogicLock
mapping blocks to regions, 3-21

M
Mapping
I/O mapping, 3-18

Index (cont.)

Mapping Operators to Dedicated Resources, 3-21
Master Constraint File, 2-12
Memory Mapping
  Altera, 3-21
  Xilinx Virtex, 3-21
ModGen, 3-21
Module Generation, 3-21
move_input_file command, 2-7
Mulitplier
  pipelined, 3-21

N

nopad attribute, 3-19

O

Opening a Project, 4-8
Operators
  inferring a pipelined multiplier, 3-21
  mapping to dedicated resources, 3-21
Output Delay
  constraint, 3-14
  setting the global default, 2-6

P

P&RIntegrator, 1-6
Parallel Case
  Verilog, 2-11
Pipelined
  multiplier, 3-21
Pragmas
  Verilog, 2-18
precision command, 2-1
precision -shell command, 2-2
Precision Synthesis
  invoking from a shell, 2-2
  Invoking the GUI, 2-1
precision.log File, 2-3
Project
  save and restore, 1-7
Project Directory
  file structure, 4-5
Project Directory Moves, 4-20
Project Management, 4-1
  actvating implementations, 4-13
  backward compatibility, 4-21
  bypassing, 4-2
  closing a project, 4-9
  commenting implementations, 4-15
  copying implementations, 4-14
  creating a new implementation, 4-10
  creating a new project, 4-3
  deleting implementations, 4-17
  opening a project, 4-8
  output directory, 2-4
  overview, 1-1, 4-1
  renaming implementations, 4-12
  saving implementations, 4-11
  setting the project directory, 2-5
  setting the results directory, 2-5
Project, Input Directory, or Input Files Move, 4-20
Projects Created with a Script, 4-7

R

Reading
  VHDL files, 2-10
Register Retiming, 1-3
Renaming an Implementation, 4-12
Reports
  generating, 2-20, 2-22
Retiming, 1-3
RTL Schematic
  viewing, 5-2

S

Safe FSM, 2-12
Saving the Active Implementation, 4-11
Schematic Viewing
  bused instance defined, 5-18

Index (cont.)

centering an object, 5-9
changing the display colors, 5-21
changing the printed sheet size, 5-19
changing the selection radius, 5-21
click and sproutl, 5-14
cross probing from HDL Designer to Precision Synthesis, 5-12
cross probing from schematic to HDL, 5-12
how to unbundle instances, 5-19
how to unbundle net buses, 5-19
invoking Critical Path Schematic, 5-1
invoking RTL Schematic, 5-1
invoking Technology Schematic, 5-1
left mouse button, 5-5
moving up the hierarchy, 5-7
object selection, 5-5
paging forward/backward, 5-7
panning around the schematic, 5-8
permuting symbol pins, 5-20
placement of latches, 5-20
placement of output drivers, 5-20
placing IO buffers, 5-20
saving a schematic to a file, 5-19
select area, 5-5
selection modes, 5-5
showing the internals of blocks, 5-10
tracing a signal to the next page, 5-8
tracing through schematic fragments, 5-13
unhighlight all traces (U icon), 5-14
unselect all, 5-5
unselect object, 5-5
view trace schematic (VT icon), 5-14
XRTL extension, 5-3
zooming in, 5-6
zooming out, 5-6
zooming to area, 5-5
zooming to fit, 5-6

Scripts
Creating from a Precision.log File, 4-17
Creating New Projects or Reusing Existing Projects, 4-18

Search Path
for include files, 2-9
for VHDL files, 2-9

set_attribute command, 3-18, 3-19
set_false_path command, 3-16
set_input_delay command, 3-11
set_input_file command, 2-7
set_output_delay command, 3-14

Setting
synthesis options, 2-12
the default Input Delay, 2-6
the default Output Delay, 2-6
the global frequency, 2-6

Setting Constraints
on false paths, 3-15

Setting the Input Directory, 4-7

Setting Up
the technology, 2-5

setup_design command, 2-6

Strategy
for obtaining fastest design, 2-20

Synthesis
setting options, 2-12

T

Tcl
startup scripts, 2-13

Technology Library
loading for a synthesis run, 2-5

Technology Schematic
viewing, 5-3

Temp Directories, 4-7

Timing analysis
Asynchronous signals, 2-20
Incorrect constraints, 2-21
Multi-Cycle paths, 2-20

Transparent latches, 2-14

Index (cont.)

U

Use Clauses, 2-10

V

Verification
  Reports, *see* Reports
  VHDL synthesis, 2-13
Verilog
  Full Case, 2-11
  Parallel Case, 2-11
Verilog 2001 Support, 2-11
VHDL
  attributes, 2-18
  Examining synthesis results, 2-13
VHDL Files
  specifying search path for, 2-9
VHDL Libraries
  reading, 2-10
VHDL Packages
  reading, 2-10

W

Work Library
  specifying, 2-8

X

Xilinx
  using UCF file as input, 2-17
Xilinx Virtex
  memory mapping, 3-21

Index (cont.)

End-User License Agreement

Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this subsection shall survive termination or expiration of this Agreement.

4. **RESTRICTIONS ON USE.** You may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. You shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. You shall not make Software available in any form to any person other than employees and contractors, excluding Mentor Graphics' competitors, whose job performance requires access. You shall take appropriate action to protect the confidentiality of Software and ensure that any person permitted access to Software does not disclose it or use it except as permitted by this Agreement. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, you shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive from Software any source code. You may not sublicense, assign or otherwise transfer Software, this Agreement or the rights under it, whether by operation of law or otherwise (“attempted transfer”) without Mentor Graphics' prior written consent and payment of Mentor Graphics then-current applicable transfer charges. Any attempted transfer without Mentor Graphics’ prior written consent shall be a material breach of this Agreement and may. at Mentor graphics’ option, result in the immediate termination of the Agreement and licenses granted under this Agreement. The provisions of this section 4 shall survive the termination or expiration of this Agreement.

5. **LIMITED WARRANTY.**

   5.1. Mentor Graphics warrants that during the warranty period, Software, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Software will meet your requirements or that operation of Software will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. You must notify Mentor Graphics in writing of any nonconformity within the warranty period. This warranty shall not be valid if Software has been subject to misuse, unauthorized modification or installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND YOUR EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF SOFTWARE TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF SOFTWARE THAT DOES NOT MEET THIS LIMITED WARRANTY, PROVIDED YOU HAVE OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) SOFTWARE WHICH IS LICENSED TO YOU FOR A LIMITED TERM OR LICENSED AT NO COST; OR (C) EXPERIMENTAL BETA CODE; ALL OF WHICH ARE PROVIDED “AS IS.”

   5.2. THE WARRANTIES SET FORTH IN THIS SECTION 5 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, WITH RESPECT TO SOFTWARE OR OTHER MATERIAL PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

6. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT PAID BY YOU FOR THE SOFTWARE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER.

7. **LIFE ENDANGERING ACTIVITIES.** NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF SOFTWARE IN ANY APPLICATION WHERE THE FAILURE OR INACCURACY OF THE SOFTWARE MIGHT RESULT IN DEATH OR PERSONAL INJURY.

8. **INDEMNIFICATION.** YOU AGREE TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE, OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH YOUR USEOF SOFTWARE AS DESCRIBED IN SECTION 7.

9. INFRINGEMENT.

9.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against you alleging that Software infringes a patent or copyright or misappropriates a trade secret in the United States, Canada, Japan, or member state of the European Patent Office. Mentor Graphics will pay any costs and damages finally awarded against you that are attributable to the infringement action. You understand and agree that as conditions to Mentor Graphics’ obligations under this section you must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to defend or settle the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

9.2. If an infringement claim is made, Mentor Graphics may, at its option and expense: (a) replace or modify Software so that it becomes noninfringing; (b) procure for you the right to continue using Software; or (c) require the return of Software and refund to you any license fee paid, less a reasonable allowance for use.

9.3. Mentor Graphics has no liability to you if infringement is based upon: (a) the combination of Software with any product not furnished by Mentor Graphics; (b) the modification of Software other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of Software as part of an infringing process; (e) a product that you make, use or sell; (f) any Beta Code contained in Software; (g) any Software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by you that is deemed willful. In the case of (h) you shall reimburse Mentor Graphics for its attorney fees and other costs related to the action upon a final judgment.

9.4. THIS SECTION 9 STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS AND YOUR SOLE AND EXCLUSIVE REMEDY WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY SOFTWARE LICENSED UNDER THIS AGREEMENT.

10. **TERM.** This Agreement remains effective until expiration or termination. This Agreement will automatically terminate if you fail to comply with any term or condition of this Agreement or if you fail to pay for the license when due and such failure to pay continues for a period of 30 days after written notice from Mentor Graphics. If Software was provided for limited term use, this Agreement will automatically expire at the end of the authorized term. Upon any termination or expiration, you agree to cease all use of Software and return it to Mentor Graphics or certify deletion and destruction of Software, including all copies, to Mentor Graphics' reasonable satisfaction.

11. **EXPORT.** Software is subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products, information about the products, and direct products of the products to certain countries and certain persons. You agree that you will not export any Software or direct product of Software in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.

12. **RESTRICTED RIGHTS NOTICE.** Software was developed entirely at private expense and is commercial computer software provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement under which Software was obtained pursuant to DFARS 227.7202-3(a) or as set forth in subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, as applicable.

Contractor/manufacturer is Mentor Graphics Corporation, 8005 SW Boeckman Road, Wilsonville, Oregon 97070-7777 USA.

13. **THIRD PARTY BENEFICIARY.** For any Software under this Agreement licensed by Mentor Graphics from Microsoft or other licensors, Microsoft or the applicable licensor is a third party beneficiary of this Agreement with the right to enforce the obligations set forth in this Agreement.

14. **AUDIT RIGHTS.** With reasonable prior notice, Mentor Graphics shall have the right to audit during your normal business hours all records and accounts as may contain information regarding your compliance with the terms of this Agreement. Mentor Graphics shall keep in confidence all information gained as a result of any audit. Mentor Graphics shall only use or disclose such information as necessary to enforce its rights under this Agreement.

15. **CONTROLLING LAW AND JURISDICTION.** THIS AGREEMENT SHALL BE GOVERNED BY AND CONSTRUED UNDER THE LAWS OF OREGON, USA, IF YOU ARE LOCATED IN NORTH OR SOUTH AMERICA, AND THE LAWS OF IRELAND IF YOU ARE LOCATED OUTSIDE OF NORTH AND SOUTH AMERICA. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of Dublin, Ireland when the laws of Ireland apply, or Wilsonville, Oregon when the laws of Oregon apply. This section shall not restrict Mentor Graphics' right to bring an action against you in the jurisdiction where your place of business is located.

16. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

17. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions, except valid license agreements related to the subject matter of this Agreement (which are physically signed by you and an authorized agent of Mentor Graphics) either referenced in the purchase order or otherwise governing this subject matter. This Agreement may only be modified in writing by authorized representatives of the parties. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse. The prevailing party in any legal action regarding the subject matter of this Agreement shall be entitled to recover, in addition to other relief, reasonable attorneys' fees and expenses.

Rev. 020826, Part Number 214231

Trademark Information