

Xilinx ISE/WebPack: Introduction to Schematic Capture and Simulation

Revision: February 7, 2003

Overview

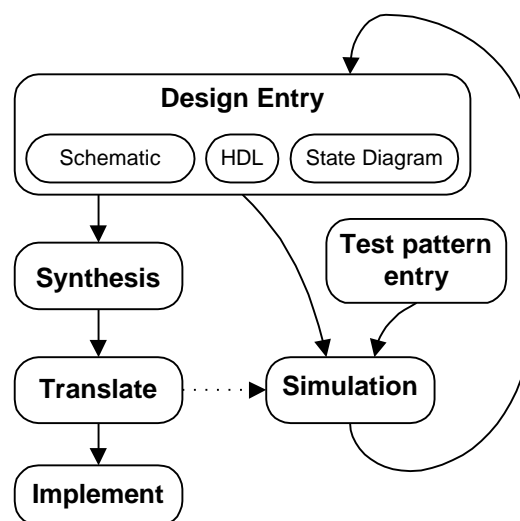
This document is intended to assist new entry-level users of the Xilinx ISE/WebPack software. It uses simple logic circuits to illustrate the various CAD tools in the ISE environment. This document should be referenced while the reader has access to a computer running the Xilinx tools, so that all procedures can be preformed as the document proceeds.

Background

Over the past several generations, engineers have created Computer Aided Design (CAD) tools to assist in all aspects of modern design. From architectural design, to mechanical engineering, to circuit design, CAD tools have revolutionized the way in which engineers work. Since the 1960's, CAD tools have been used in digital circuit design to “capture” a virtual copy of a circuit on a computer, and then to simulate the circuit so that various behaviors could be investigated and modified before the circuit was actually built. Since their inception, CAD tools have been continuously evolving. Modern tools allow very precise simulations (down the picosecond), they allow circuits to be automatically synthesized from an easily written high-level definition, and they allow designs to be reformatted so they can be implemented in a variety of technologies.

Starting in the 1980's, digital engineers could use powerful new technologies to implement complex digital systems on a single chip, right on the desktop. These chips, called “Field Programmable Gate Arrays” (or FPGAs), and the software used to program them, has revolutionized digital design and ushered in a new class of CAD tools.

FPGA CAD software typically includes schematic capture, simulation, implementation, and device programming tools. All of these tools can be started from a single “navigator” tool that coordinates the files and processes associated with a given design project. The navigator shows all source files, all CAD tools that can be used with the source files, and any output or status messages and files that result from running a given tool. The remainder of the document presents the Xilinx tools, starting with the **Project Navigator**.

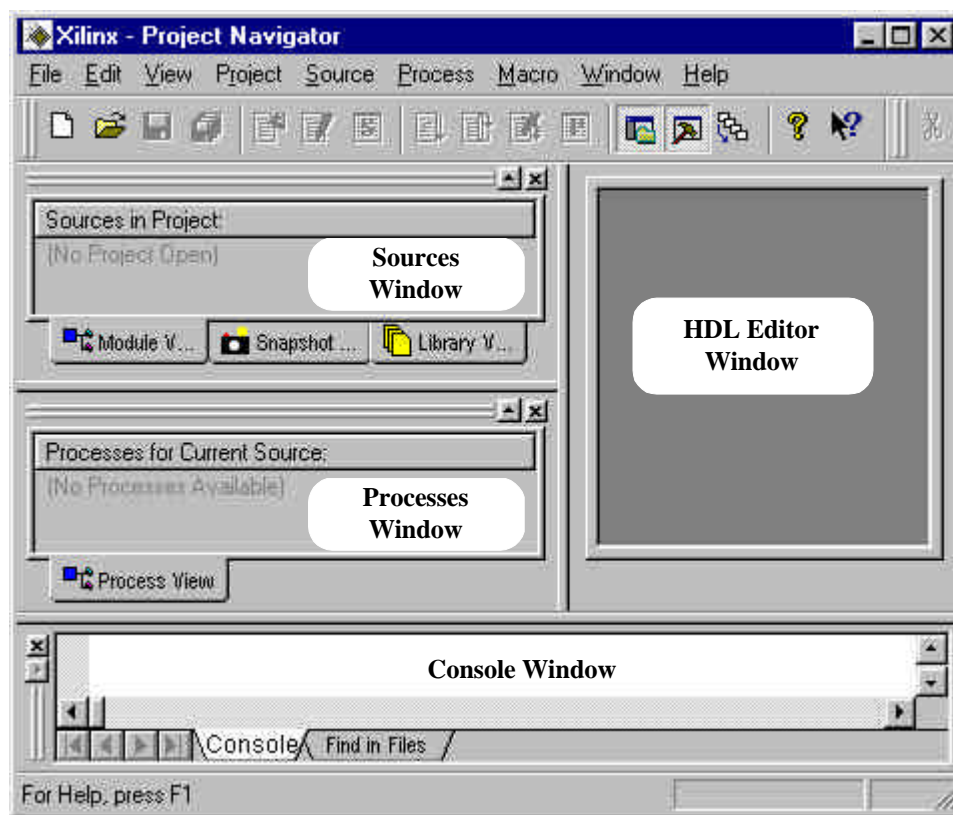


CAD tool general design flow

The Project Navigator



The entry point to the Xilinx ISE or WebPack tool is the **Project Navigator**. The Project Navigator provides an user interface that organizes all files and programs associated with a given design. It is divided into four main panels as shown. The **sources** panel shows all source files associated with a given design. Double-clicking on a file name shown in this panel will open the file in the appropriate CAD tool. The **processes** panel shows all processes that are available for a given source file (different source files have different process options). Double-clicking on any process name will cause that process to run. The **console** panel shows process status, including all warnings and errors that result from running a given process on a given source file. The **HDL editor** panel shows the HDL source code for any selected HDL source file. The project navigator will also open other windows as needed for some applications (for example, the schematic capture tool opens in a separate window).

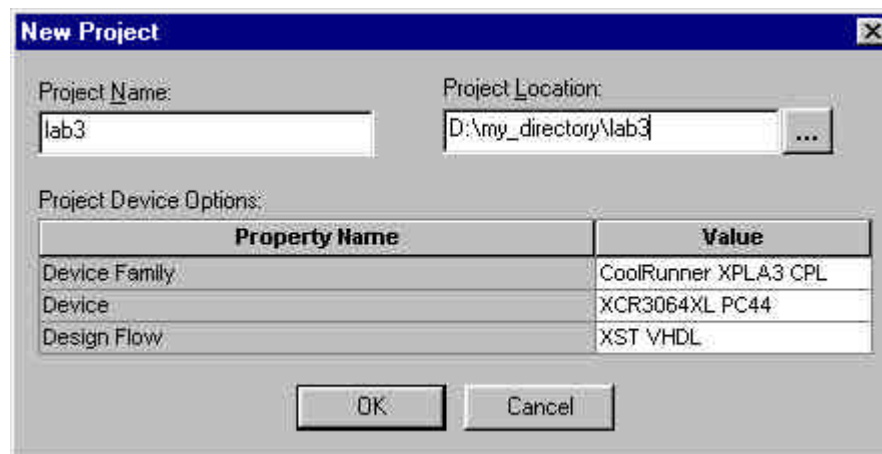


When working with Digilab boards and the ISE tool, you never need to leave the Project Navigator environment. All required steps, from design creation to ultimately programming the board, can be completed from within the Project Navigator framework.

Starting a new project

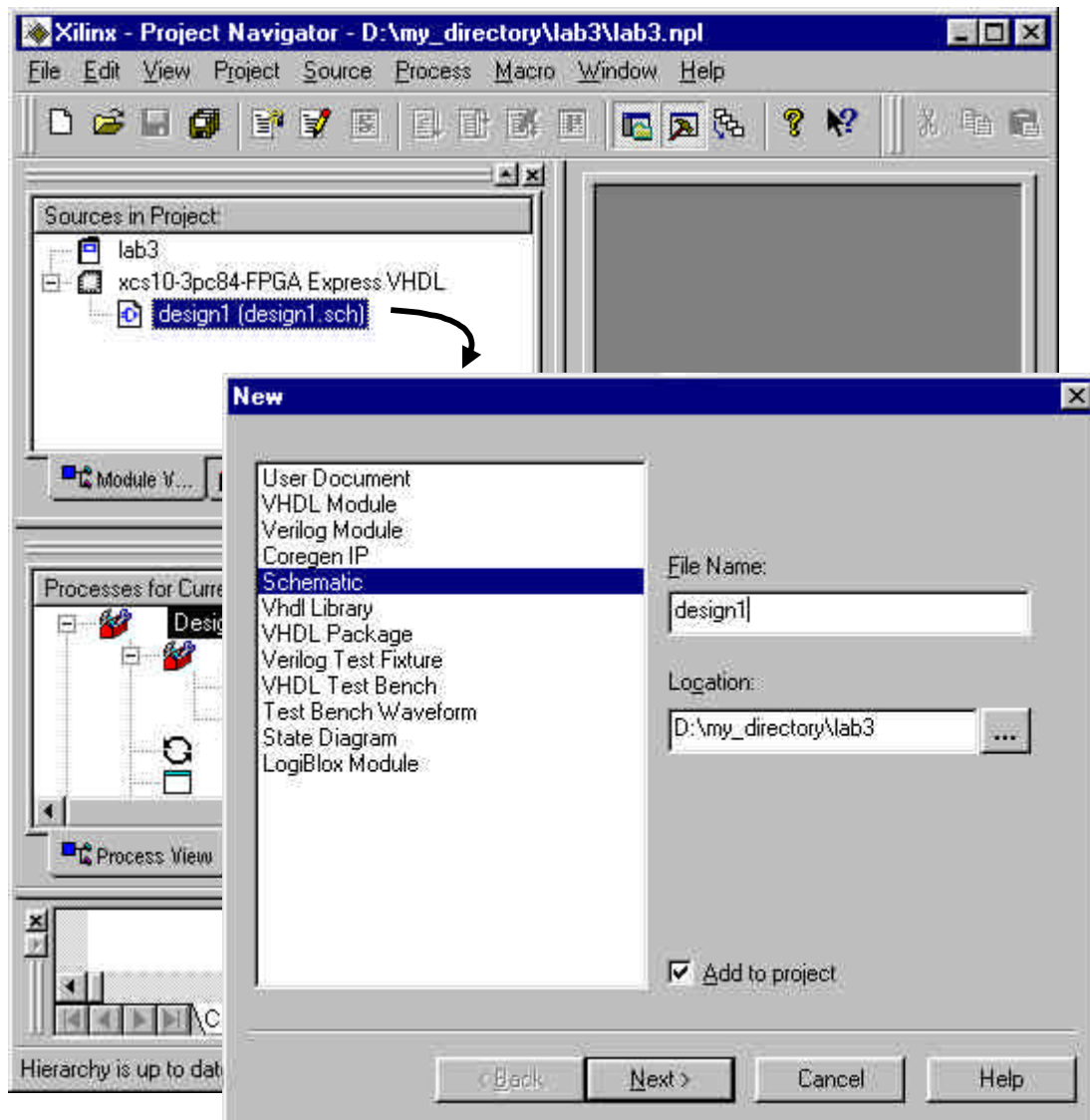
New projects can be defined from within the project navigator. The project navigator can be started from the windows Start menu, or by double-clicking the desktop icon. After the Project Navigator window opens, a new project can be created, or an existing project can be opened. In general, a new project should be created for each new lab exercise or each new design. The project navigator can be configured to automatically load the last project used, or to not load any project (see the “properties” dialog box).

Selecting “new project” from the File pull-down menu will open the **New Project** dialog box, where all information for a new project can be entered. Enter a descriptive name (such as *lab3*) in Project Name box, and choose an appropriate directory in the Location box. This directory will store all design files and all intermediate files, so you will want to choose a directory that is protected and backed-up. Choose the appropriate device family (CoolRunner for the XCRP board, or Spartan 2 for a “Digilab 2” FPGA board), and the appropriate device type (e.g., xcr3064XL PC44 for the XCRP board). Finally, choose XST VHDL as the design flow and click OK. This same new-project definition procedure is used for any new design. The remainder of this exercise presents schematic capture in the ISE environment.



Schematic Capture in ISE/WebPack

To create a new schematic, right-click on the *device* entry in the source window of the project navigator (the device entry is below the project name in the sources window) and select **New Source**. In the window that appears, select **Schematic** and enter a descriptive File name. Make sure to select *add to project*, and then click *Next* and *Finish* to bring up the schematic editor window. The schematic editor is simply a blank palette to which shapes (representing circuit components) and lines (representing wires) can be added. The schematic tool can be used effectively using tool-bar buttons or pull-down menu choices. In general, the tool-bar buttons and pull-down menus offer the same functions, but the pull-down menus offer some unique features; you are encouraged to experiment with them.

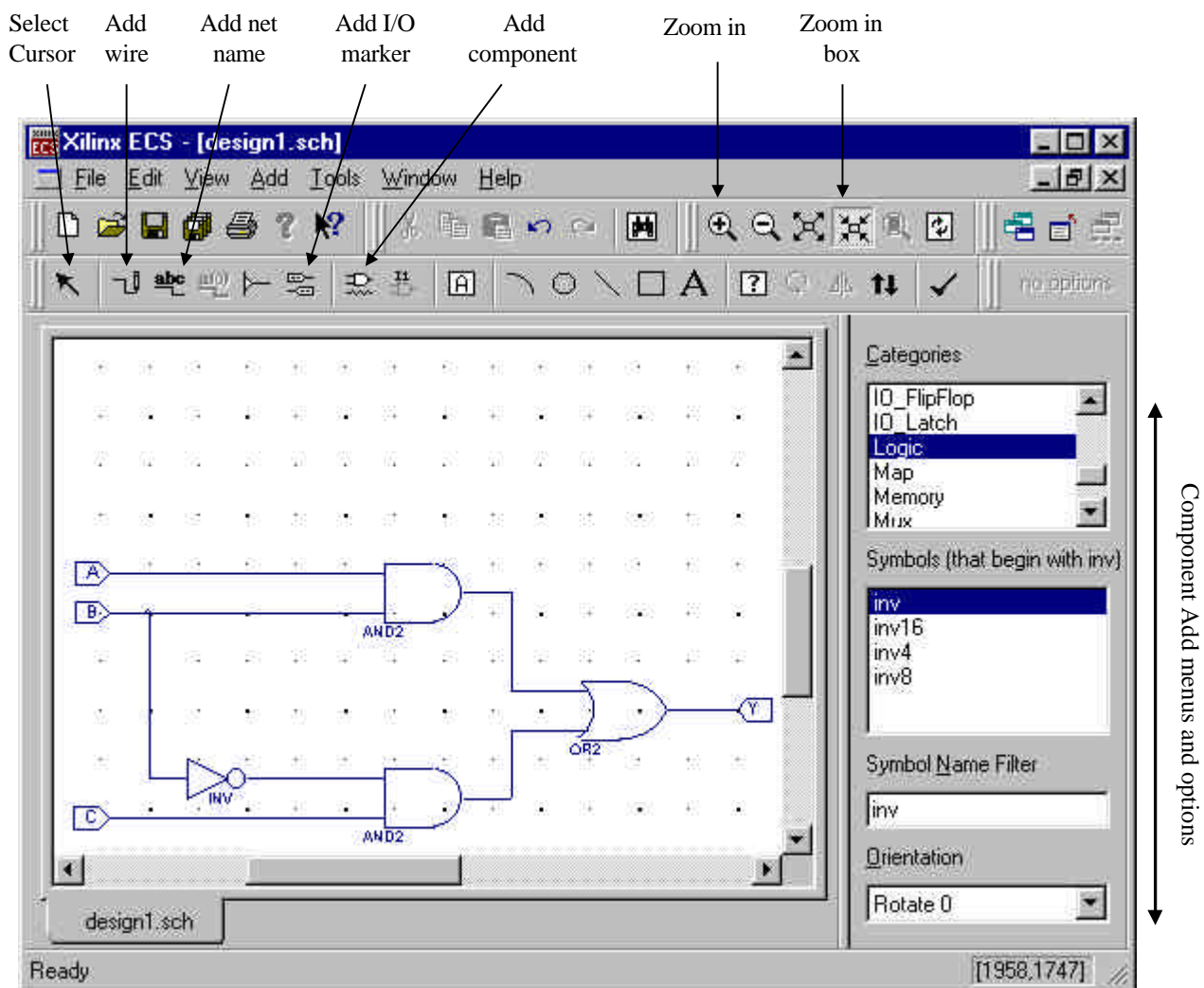


Creating a new schematic source file

To draw a schematic, components must be added and interconnected with wires. To add components, click the *Add Symbol (or component)* tool-bar button to cause the component library to be displayed in a menu on the right of the schematic entry window. The components shown in the menu depend on which *device family* was selected in the new project setup window – different families use different schematic symbol libraries. Under *Categories*, select “Logic”, which restricts the *Symbol* menu to displaying only the more basic logic components like AND and OR gates. To add a particular component, scroll through the menu to locate it, or type its name in the box at the bottom of the menu. Components can be moved after they have been added, so it’s generally a good idea to add all needed components first, and then to rearrange them into a neater circuit once they are all present. Selected components can be “dragged and dropped” onto the schematic drawing palette.

In this example, we’ll create the circuit specified by: $Y = A \cdot B + B' \cdot C$. This circuit requires two **and2** gates, an **or2** gate, and an **inv** gate. These components can be added the schematic by selecting them

from the component menu as described, and then dragging-and-dropping them to place them on the schematic palette.



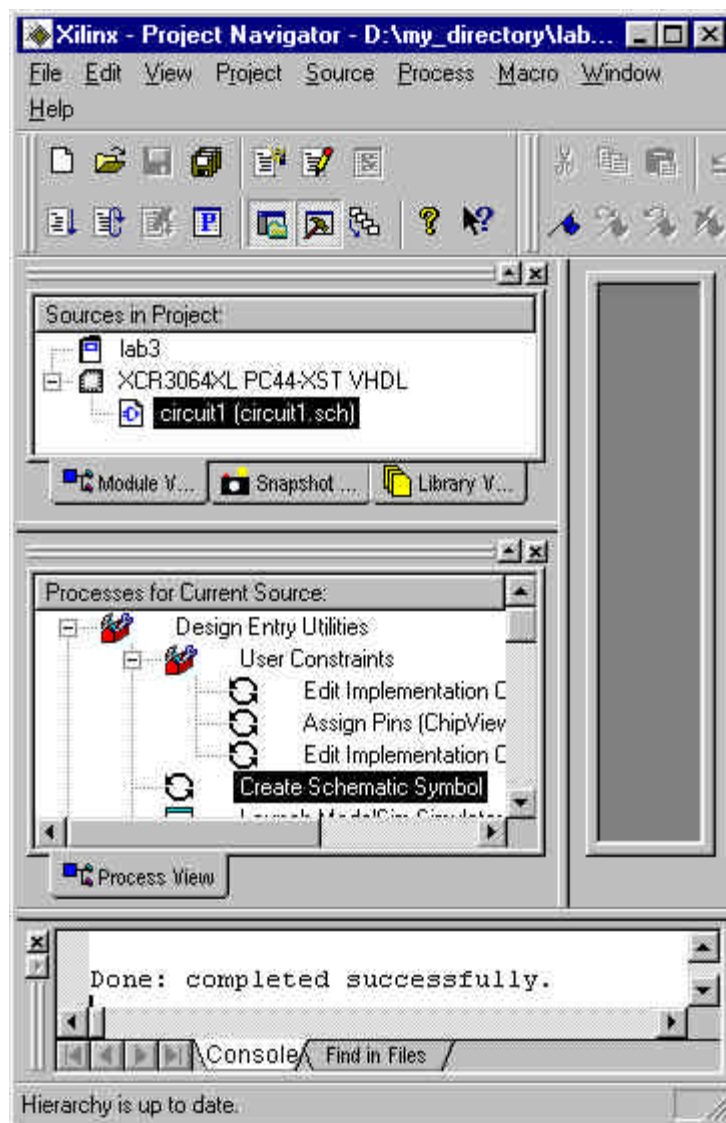
Once the needed components are in place, wires can be added by pressing the *add wire* tool button, and then clicking on the source and destination component pins. When connecting components with wires, be sure some amount of wire exists between all component pins. Note that it is difficult to tell whether a wire segment exists between the inverter and the AND gate. In general, enough wire should be used so that it is obvious that the pins are not directly touching. Wires can be ended in “space” by double clicking the screen area where the wire is to be terminated. Labels can be added to wires by selecting the *Add Wire Name* button, and then selecting the wire, or by double-clicking on the wire. Circuit inputs and outputs (as opposed to internal nodes) are identified by selecting the *Add I/O marker* button and clicking on the end of each input or output wire. Unique default names are automatically assigned to I/O markers. To change the default names, click on the *select cursor* toolbar button, and then double-click on each I/O marker in the schematic. In the window that appears, enter a new name, choose whether the net is an input or output, and click **OK**.

Hierarchical design

For all but the simplest circuits, schematics can be made much more readable if certain well-defined parts of the circuit are grouped inside of a “wrapper” called a **macro** or **symbol** (just like in computer programming, where often-used code is placed inside of a subroutine). When creating a macro, it is important to make sure all inputs and outputs have I/O markers, and that all I/O markers are named. These names will appear as pin labels on the macro symbol. A macro can be created from any schematic page, and everything on the schematic page will be placed inside the macro symbol. To create a macro for a given schematic source, select the Xilinx Project Navigator window. From within that window, select the schematic source file name in the *Sources in Project* panel, and then select the “Create Schematic Symbol” process available under the “Design Entry Utilities” entry in the *Processes for Current Source* panel. The screen shot on the right shows a macro being created for a circuit named “circuit1”.

After a macro has been created, it is available as a component in the schematic capture tool. The new macro component can be added to a schematic by selecting the circuit macro name in the symbols panel in the schematic editor (in this case, a macro named circuit1 would be selected). If it is difficult to find the macro, the symbols search field can be narrowed by selecting the project name in the Categories panel of the schematic capture window, or by typing the macro’s name in the Symbol Name Filter box.

Using just these basic methods, schematics for circuits of arbitrary complexity can be created.



Basic Logic Simulation

A logic simulator allows a designer to observe circuit outputs in response to all combinations of inputs before the circuit is implemented in hardware. Simulating a circuit is perhaps the best technique an engineer can use to ensure that all required features are present, and that no unintended behaviors have been inadvertently designed in. For larger designs, simulation is far cheaper and far less error prone than designing and testing a hardware prototype. If errors are observed in the simulator's output, the circuit can easily be corrected and re-simulated as often as necessary.

The simulator requires two kinds of inputs: the circuit description source file, and a set of stimulus values that define all input logic inputs for the duration of the simulation. The circuit description source must be an HDL file; if a schematic source is created, an HDL file is automatically generated whenever the schematic is saved. No matter what type of source file is used to describe a circuit, the designer must define the stimulus inputs.

The simulator functions by dividing the overall simulation into very small time steps (typically 10ps, but this value can be changed by the user). At each time step, the simulator finds all signals that have changed during the preceding time step, and processes those signals as dictated by the circuit's HDL source file. If output signals must change as a result of that processing, then changes to these signals are "scheduled" for a later time step (signal changes are scheduled for a later time step because signals can't change voltage values instantaneously).

Different simulators provide various methods for designers to define input signals over time. Most simulators provide at least three methods, including a graphical interface, a text file based interface, and a command line interface. Any of these methods can be used with the ModelSim simulator included with the Xilinx ISE/WebPack CAD tools. Graphical interfaces are most useful when defining small numbers of inputs (up to 20 or so) that require relatively few changes over time (e.g., each of the 20 signals might need 20 or 30 changes between '0' and '1'). When dealing with a greater number of input signals (possibly numbering in the hundreds), or a greater number of signal changes over time (possibly in the tens of thousands), a graphical interface is too cumbersome. In this case, a text file based interface is used. The third method using the command line interface is most useful when changing a few signals once or twice to make some quick adjustments to the end of a graphical or text file based simulation.

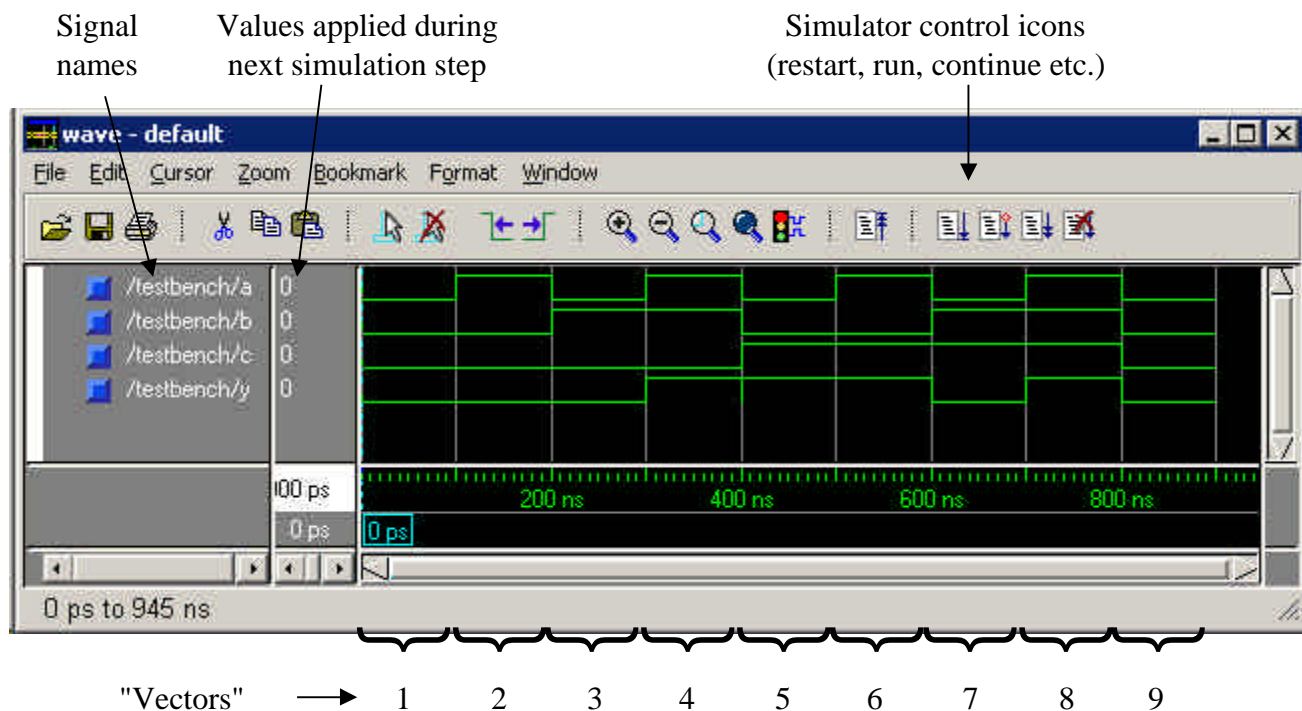
The ModelSim simulator is a state-of-the-art tool that has many features to assist engineers in creating stimulus inputs, editing circuit descriptions, and analyzing circuit outputs. Only the most basic features of the ModelSim graphical interface and command line interface are presented here – more involved features will be introduced in a later tutorial. A later section of this document (dealing with creating HDL source files) will present creation of text-based stimulus files.

ModelSim graphical user interface

The ModelSim tool uses two different graphical interfaces. The "waveform viewer" interface shows simulator state and all past simulator outputs, but it does not allow any waveforms to be modified (it is, as the name implies, strictly a waveform viewer). The waveform editor allows input stimulus to be created using a "point and click" interface to set logic levels on input signals. These separate tools can be used together to create a simple yet powerful simulator interface.

The figure below shows an example of the waveform viewer. From this figure alone, it is not possible to determine which of the **a**, **b**, **c**, and **y** signals are inputs and which are outputs, nor is possible to discern whether the inputs were defined using a graphical interface, a text-based interface, or the command line.

In fact, the signals **a**, **b**, and **c** are inputs, and they were defined using the waveform editor. The simulator created the output signal **y** by processing the inputs. Note that by assigning a '0' to a LLV signal and a '1' to a LHV signal, the signal values can easily be mapped to a truth table. Each row in a truth table represents a unique combination of inputs, and each "vector" time-slice in the figure below represents a unique combination of inputs. A vector (or more properly, a "test vector") is simply a label given to the set of all circuit inputs and outputs at a given point in the simulation. Vector duration is defined by the smallest amount of time that all inputs are stable; thus, the border between consecutive vectors is defined by one or more signal changes. By definition, vectors are non-overlapping and seamless throughout the entire simulation. In general, all vectors have the same duration, but this is not a requirement. A good simulation contains enough vectors so that all signals in the design are driven to both LHV and LLV.

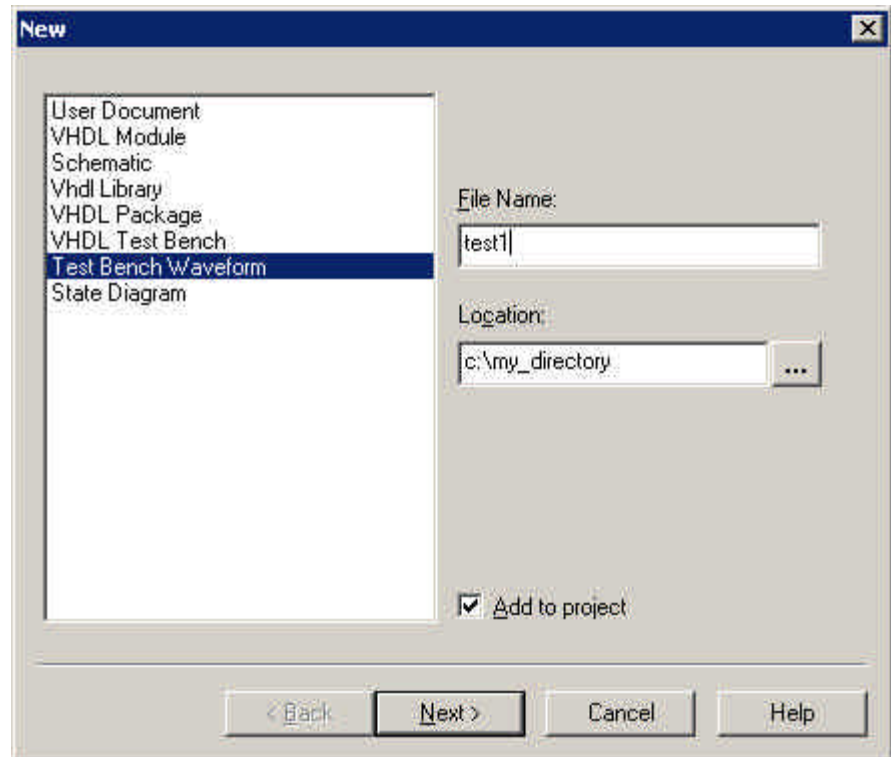


Running a simulation

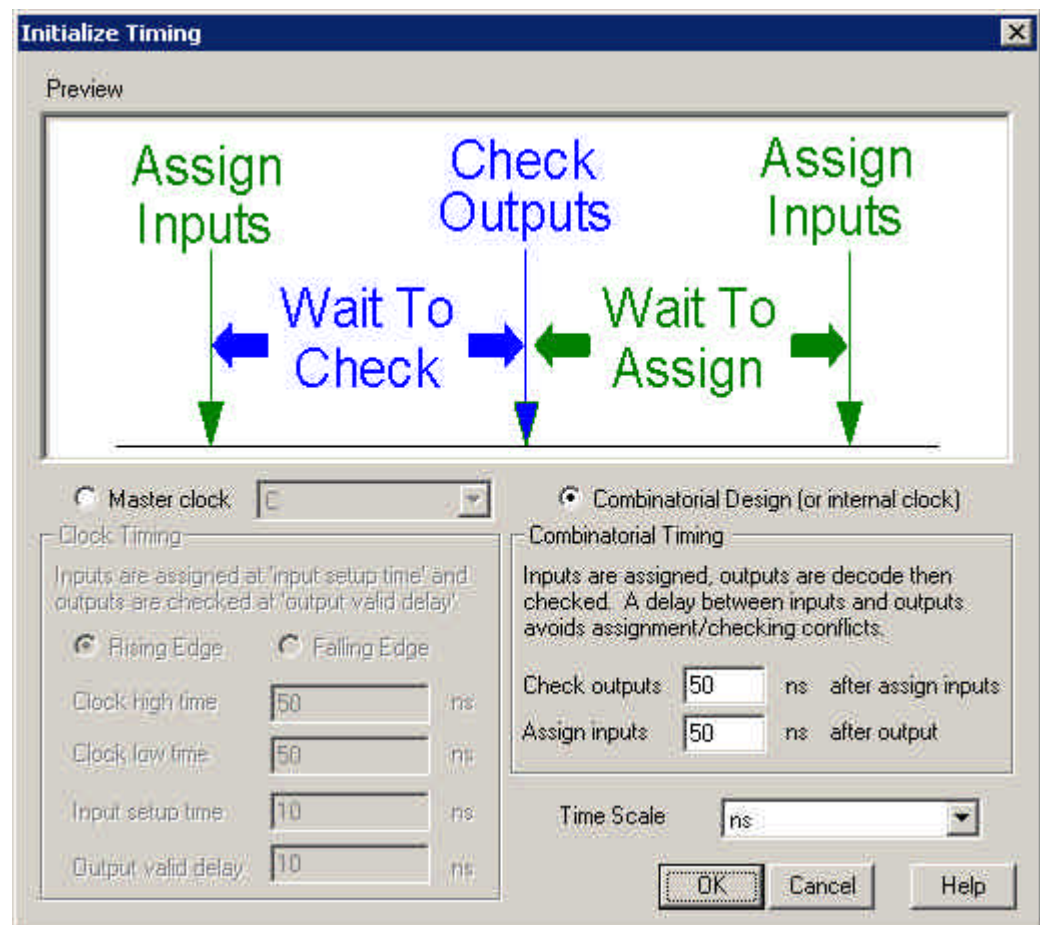
Since Xilinx has incorporated ModelSim into their CAD tool framework, the simulator can be run without leaving the project navigator. But prior to running the simulator, a stimulus file should be created (the simulator can be run without a stimulus file using the command line interface, but it is a good design practice – and easier – to create a stimulus file first). Xilinx includes a graphical tool that can be used to create stimulus files. To start this tool, right-click on the source to be simulated in the *Sources in Project* panel of the Project Navigator. Then, select New Source, choose the Test Bench Waveform file type, and enter a file name (do not choose the same name as your circuit source file).

Click next, and then select the source file that the Test Bench Waveform should be associated with (which in this case is the default choice). Clicking finish will start the waveform editor.

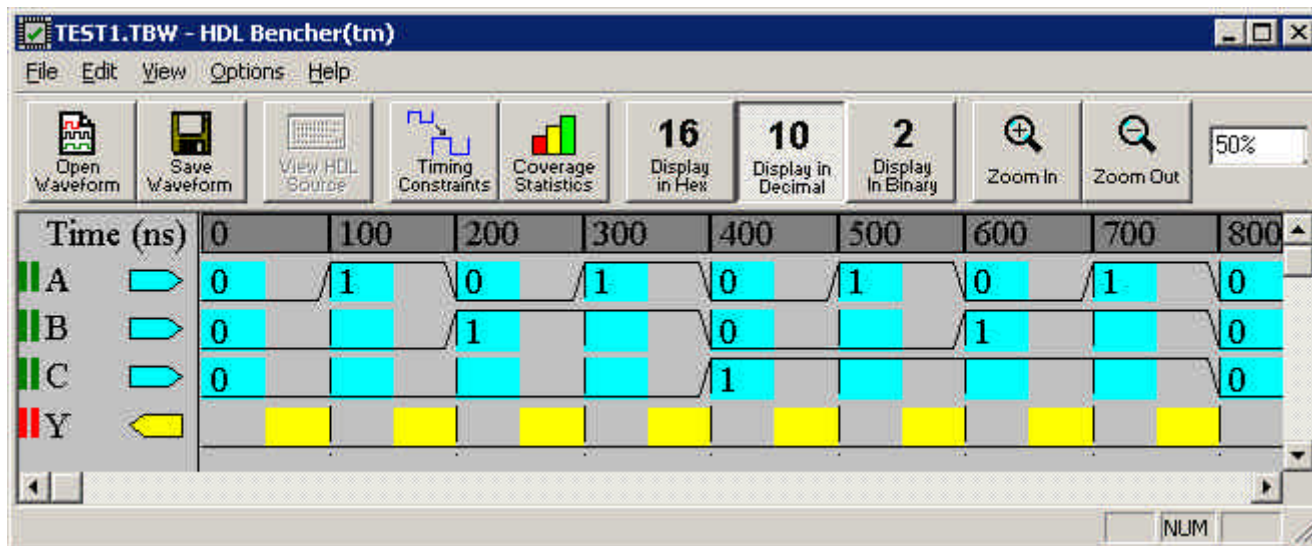
The waveform editor may open with a dialog box that allows users to modify several parameters as shown in the figure below. In this example, three parameters can be set: the time scale, the time at which output values should be checked, and the time at which inputs should be assigned. These parameters are useful for more advanced users; for now, simply click OK to accept the defaults. We will return to these features in a later tutorial.



When the waveform editor opens, all “top level” signals are shown on the far left of the window (these are the signals to which input or output connectors have been attached, or, for future reference, the signals listed in a VHDL port statement). To the right of the signal names, stretching to the far end of the window, are grid positions underneath a time base that defaults to 100ns per simulation step. Left-clicking a signal under any time step will cause that signal to toggle from a ‘0’ to a ‘1’ or vice-versa. By clicking



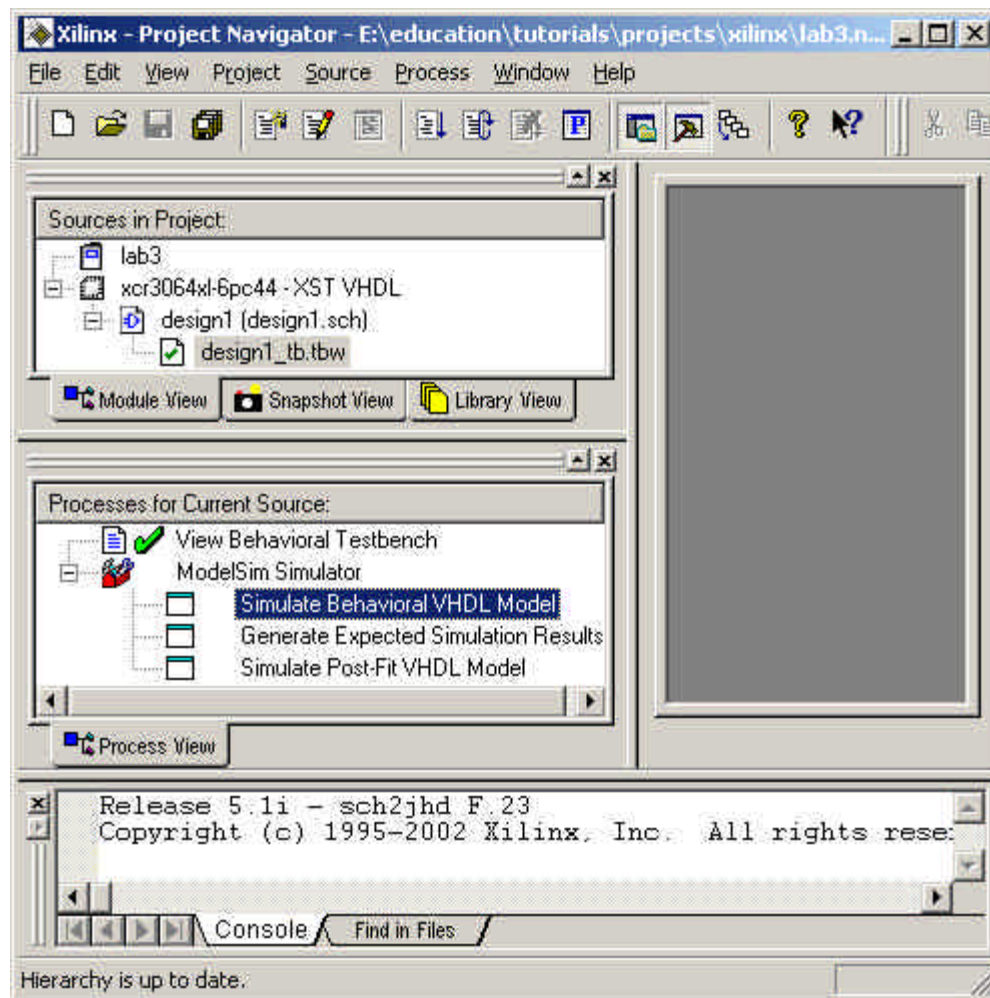
signal names at the appropriate time steps, simulation input patterns can readily be defined. Note that output signals are also shown in the waveform editor. Recall that the simulator, and not the user, defines outputs in response to input stimulus. Output signals are shown so the user can enter **expected values** for each input vector. During the course of the simulation, the simulator compares the value entered by an user with the values generated by the simulator. If at any point the values do not match, the simulator issues an error statement. If no values are assigned to output signals, or if all correct values are assigned to output signals, then the simulator will not issue an error message.



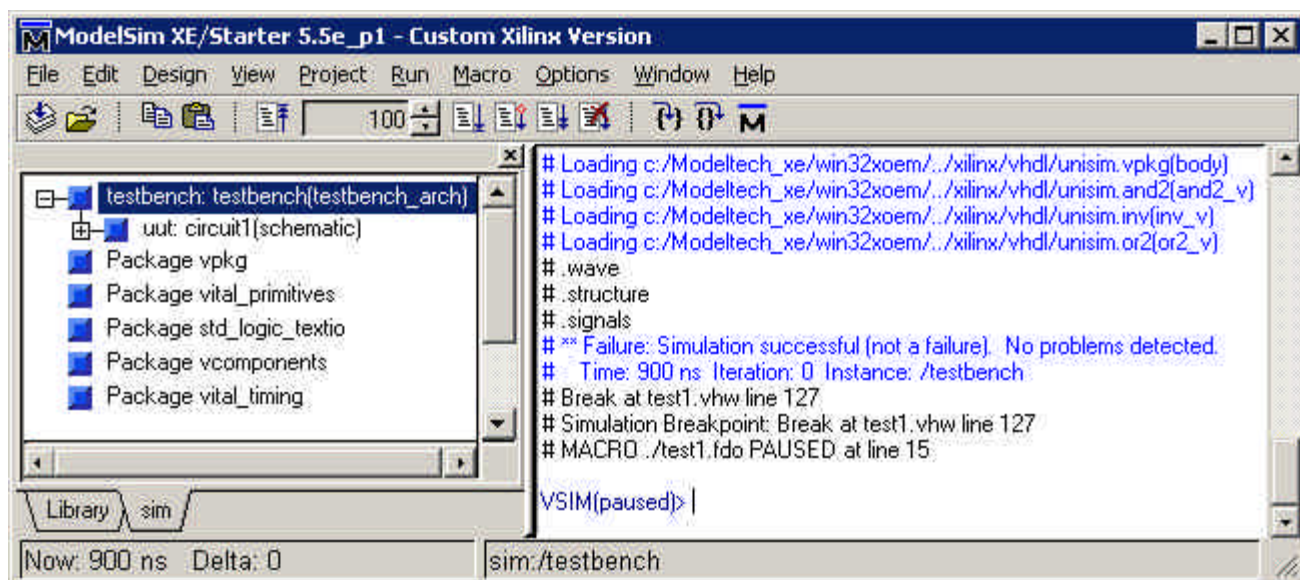
When the waveform editor window opens, a second window showing the source code for the circuit to be simulated is opened as well. For this design, the source file is shown as a Xilinx VHDL netlist of the schematic (because XST VHDL was selected as the project type). VHDL source files will be presented in later.

Once the desired values have been entered into the waveform editor, click “save waveform”. A dialog box may open asking how many time steps (or vectors) should pass between the last input change and the end of the simulation – for now, select the default (‘1’).

To run the simulator, select the test bench waveform filename in the “Sources in Project” panel of the Project navigator (this file will have a .tbw extension). Then, under the “ModelSim Simulator” entry in the “Processes for Current Source” panel, select “Simulate Behavioral VHDL Model”. This will start and run the ModelSim simulator after automatically loading both the circuit netlist and stimulus waveform (see figure below).

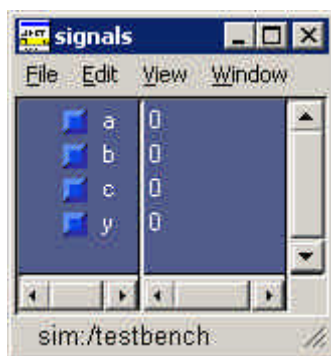


When ModelSim starts, four different windows are displayed, including the *command*, *structure*, *signals*, and *wave viewing* windows. The command window (below) shows status, error and warning messages, and it provides a prompt at which command line instructions can be entered.



The structure window shows all files used by the simulator in outline form. This window is useful in more complex circuits when a designer wants to find a component that might be buried under layers of hierarchy (we will use this feature in later labs). In this case, expanding the UUT entry (UUT stands for “Unit Under Test”) will show all component files required by the current design.

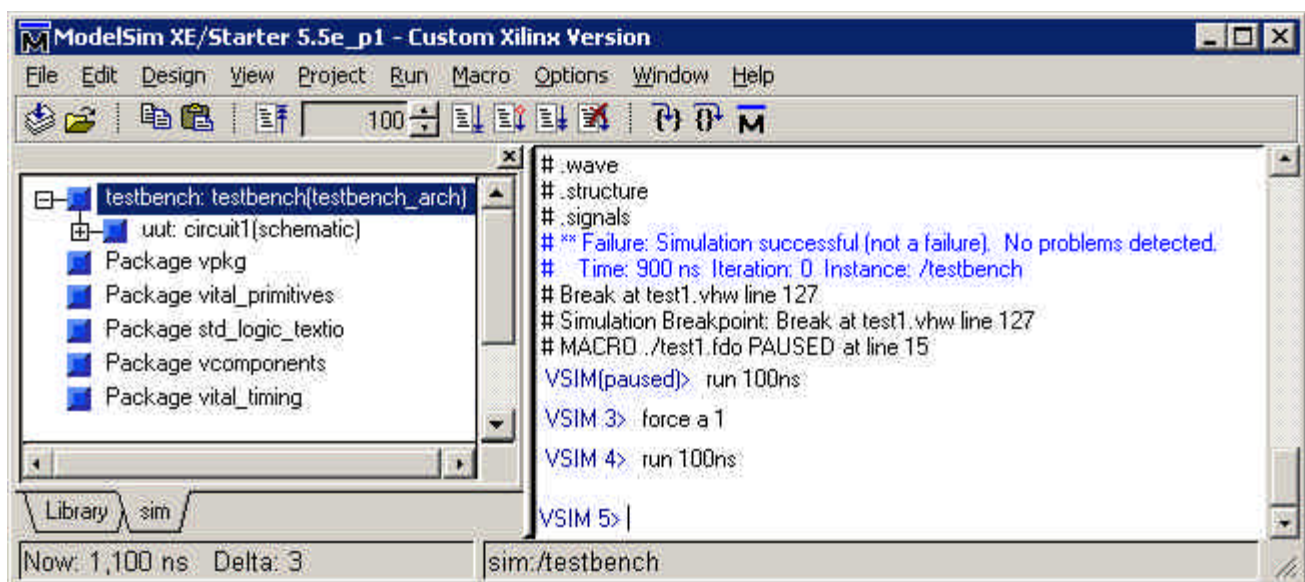
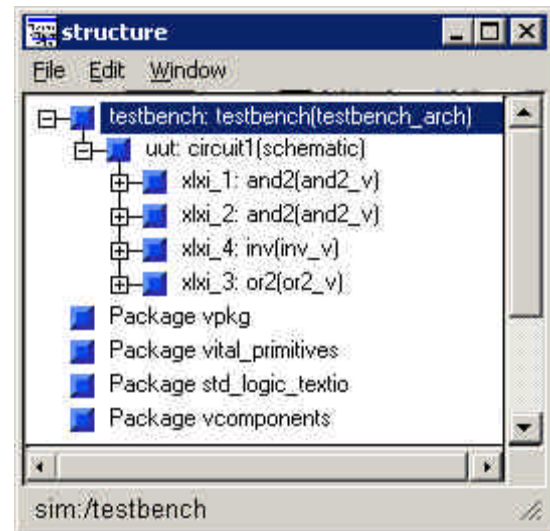
The signals window shows all signals that are available to the simulator. This window can be used to add or remove signals from the waveform viewer, or to “force” logic values onto signals in an override fashion.



The fourth window is the waveform-viewing window shown earlier. The waveform-viewing window shows the current waveforms, and provides panning, zooming, and cursor functions. This window also offers some toolbar buttons that can be used to restart and run the simulator.

When the simulator finishes running with the netlist and stimulus values, it is typically zoomed in and panned to the far right. To see the entire simulation, click the “zoom full” toolbar button (the rightmost magnifying glass icon on the waveform viewer toolbar).

As mentioned above, the simulator can also be run in command line mode. This is useful for tacking on a few more vectors to the end of a simulation without modifying the stimulus file and restarting the simulation. One very useful command is “**run XXYY**”, where XX is some number of time steps, and YY provides the units (the yy field can be fs, ps, ns, us, or ms).



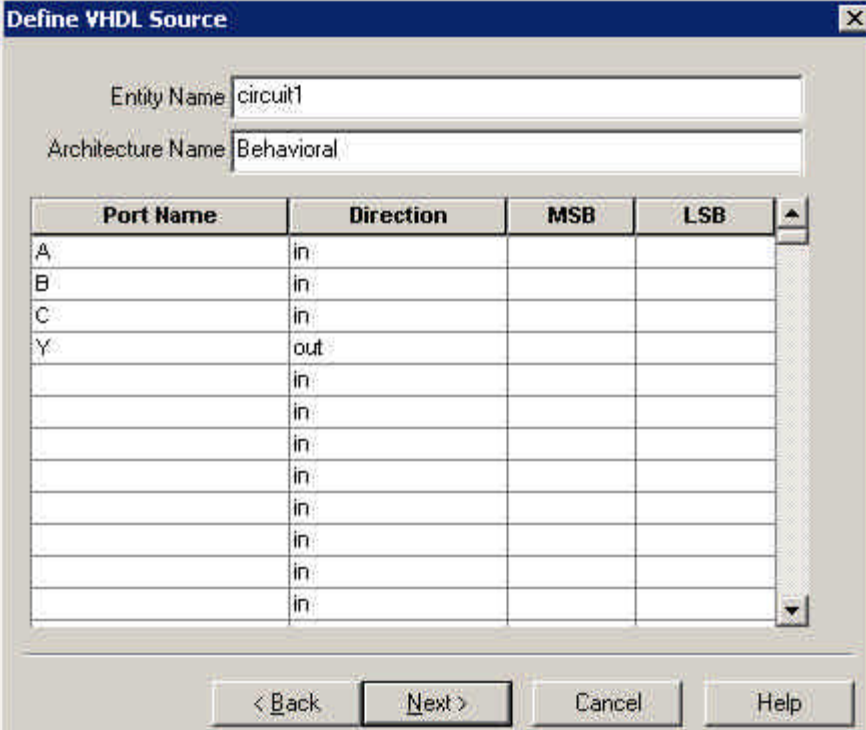
A second useful command can set a signal to a given value. Called a “force function”, its syntax is “**force** signal_name value”, where signal_name is the name of a signal to be driven to some value (e.g., ‘0’ or ‘1’), and value is the value the signal should assume during the next simulator time step (see preceding figure).

Using the Xilinx VHDL tools

To implement VHDL designs in the WebPack environment, a text editor is required to create the VHDL source file, a synthesizer is needed to translate the source file to a form that can be downloaded to a chip, and a simulator is needed to check the results. Other tools, like a floorplanner and/or timing analyzer, might also be needed for more complex designs (these tools will be discussed in a later tutorial).

Any text editor can be used to create a VHDL source file. Xilinx supplies an editor with WebPack, and this editor uses colors and auto-indents to make the source file more readable (its use is recommended). To start the Xilinx HDL editor, right-click on the device name in the *Sources in Project* panel, and select **New Source** (or, select **Project->Add New Source** from the Navigator window). In the window that appears, choose **VHDL Module** and enter a name for the source file, and then click **Next**. This opens a dialog box (shown below) where an *architecture* name can be chosen (the default name can always be used), and a table where input and output signals can be defined. This is actually an optional step – if signals are defined here, they will automatically appear in a port statement within an entity statement (and if signals are not defined here, the user must type that information into the editor).

For this example, we will define a circuit with the behavior $Y \leq A'B + C$. Enter A, B, and C in the dialog box table as inputs, and enter Y as an output (be sure to change the **direction** of Y to **out**). Change the architecture name if desired (any text string is acceptable). For now, the remaining fields in the dialog box can be ignored, so click **Next** and then **Finish** in the ensuing summary dialog box. The HDL editor opens with part of the required VHDL code present (see figure below).



The dialog box titled "Define VHDL Source" contains the following fields and table:

Entity Name: circuit1

Architecture Name: Behavioral

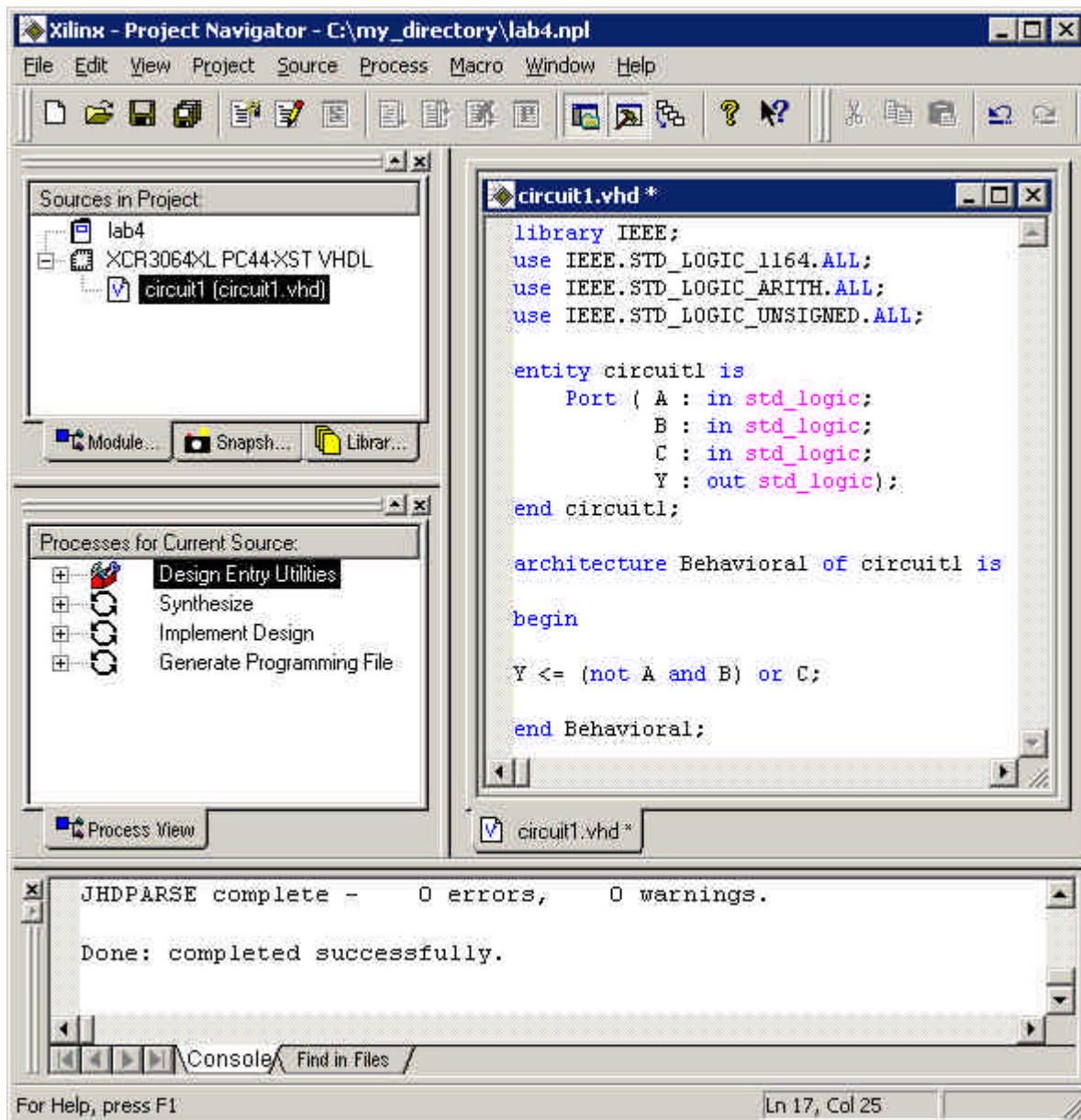
Port Name	Direction	MSB	LSB
A	in		
B	in		
C	in		
Y	out		
	in		
	in		
	in		
	in		
	in		
	in		
	in		
	in		

Buttons at the bottom: < Back, Next >, Cancel, Help

Referring to that VHDL source file, the library statements are needed to make various VHDL items visible to the current source file, and the entity and architecture statements were created using the information entered in the “define VHDL source” dialog box. All that remains is to complete the architecture statement with a description of the circuit. As shown in the source file, the VHDL behavioral statement

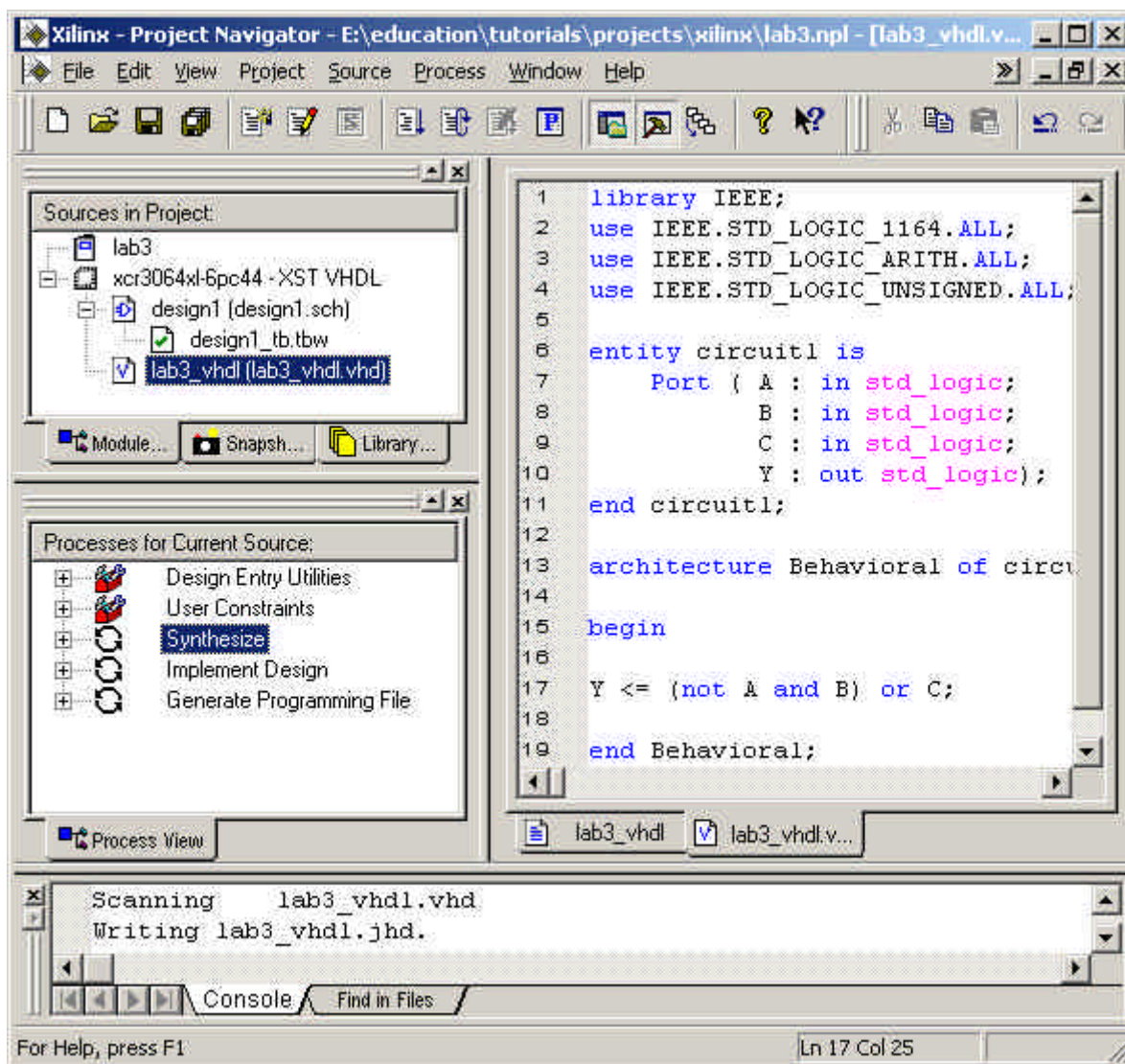
$$Y \leq (\text{not } A \text{ and } B) \text{ or } C$$

was entered to define the circuit for this exercise (you should type that same statement as well).



The VHDL source file is now complete, and it can be simulated or synthesized and then downloaded to the Digilab board. To simulate the source file, use the simulation procedures presented above (namely, create a stimulus file using WebPack's waveform editor, then run ModelSim using that stimulus file to check the results).

To synthesize a VHDL source, highlight its name in the *Source in Project* panel in the Project navigator, and then double-clicking the **Synthesize** process in the *Processes for Current Source* panel. If synthesis completes without errors, a green check mark will be displayed by the synthesize process, and a ‘process completed successfully’ message will appear in the status window at the bottom of the Project Navigator window. If errors are present, error messages in the status window will guide you towards appropriate corrective action. Note that if error messages are related to problems in the source file, double-clicking the red “error” icon in the status window will automatically jump to the offending line in the source file.



It is possible (and sometimes desirable) to add VHDL modules into schematics. To create a schematic symbol for the VHDL module, select the VHDL source file name in the *Source in Project* panel, and double-click on the “Create Schematic Symbol” process in the *Processes for Current Source* panel. This will create a symbol that can be added to any schematic.

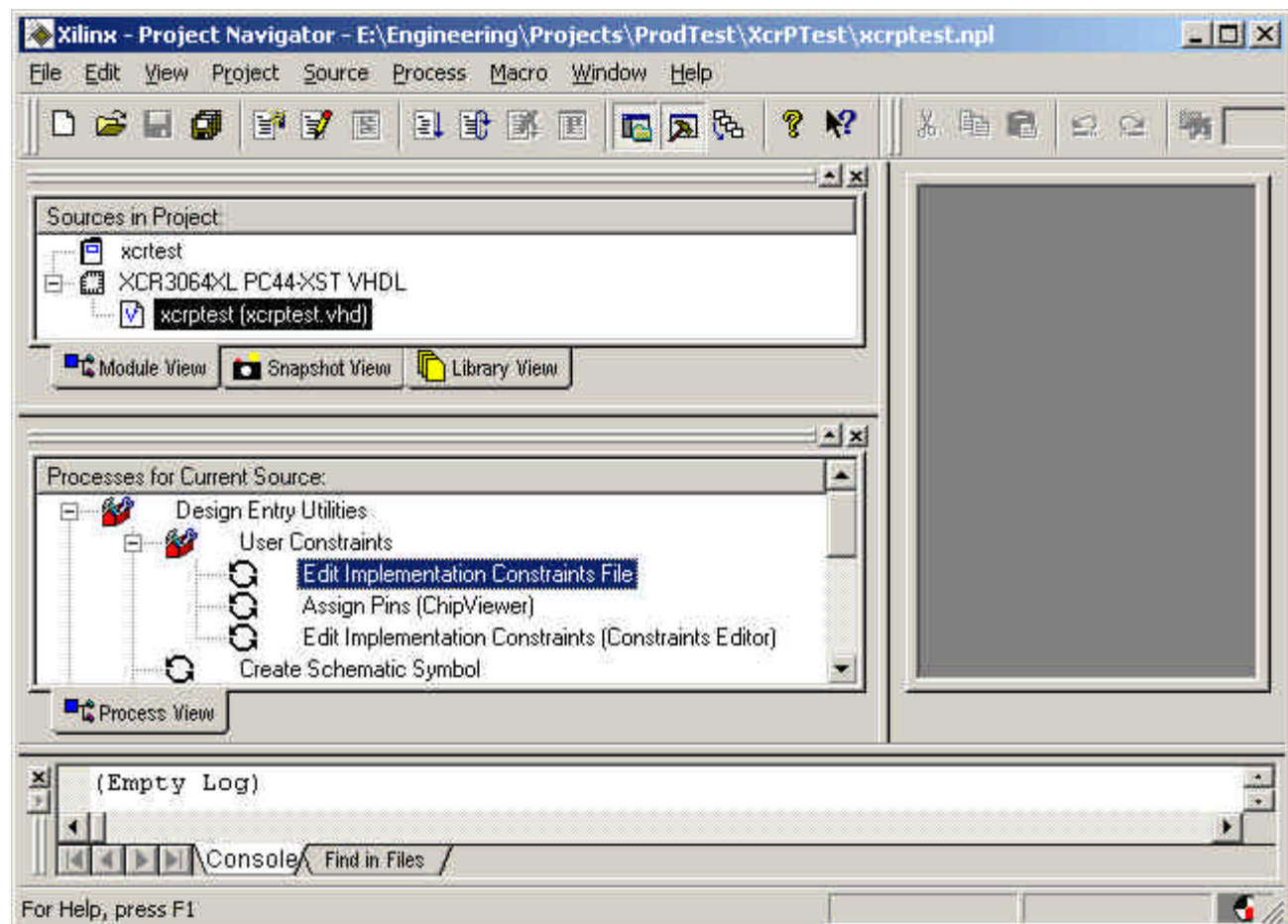
Many good on-line VHDL references exist. You are encouraged to search for them, or to purchase a reference manual (see the website for recommendations on good VHDL titles)

Downloading a design

The Xilinx programmable chip on the Digilab board can easily be configured with circuits that have been designed in the Xilinx CAD tools (or, for that matter, in CAD tools from other vendors like Mentor Graphics or Cadence). Circuits downloaded into the Xilinx chip can directly drive any of the I/O devices on the Digilab board. Before a circuit can be downloaded, input and output signals in the circuit must be associated with physical pins on the chip, and the Xilinx chip targeted by the design must be specified. Once this information is entered, the Xilinx backend tools can process the circuit source files into a chip-specific and design-specific file that can be transferred to the Digilab board.

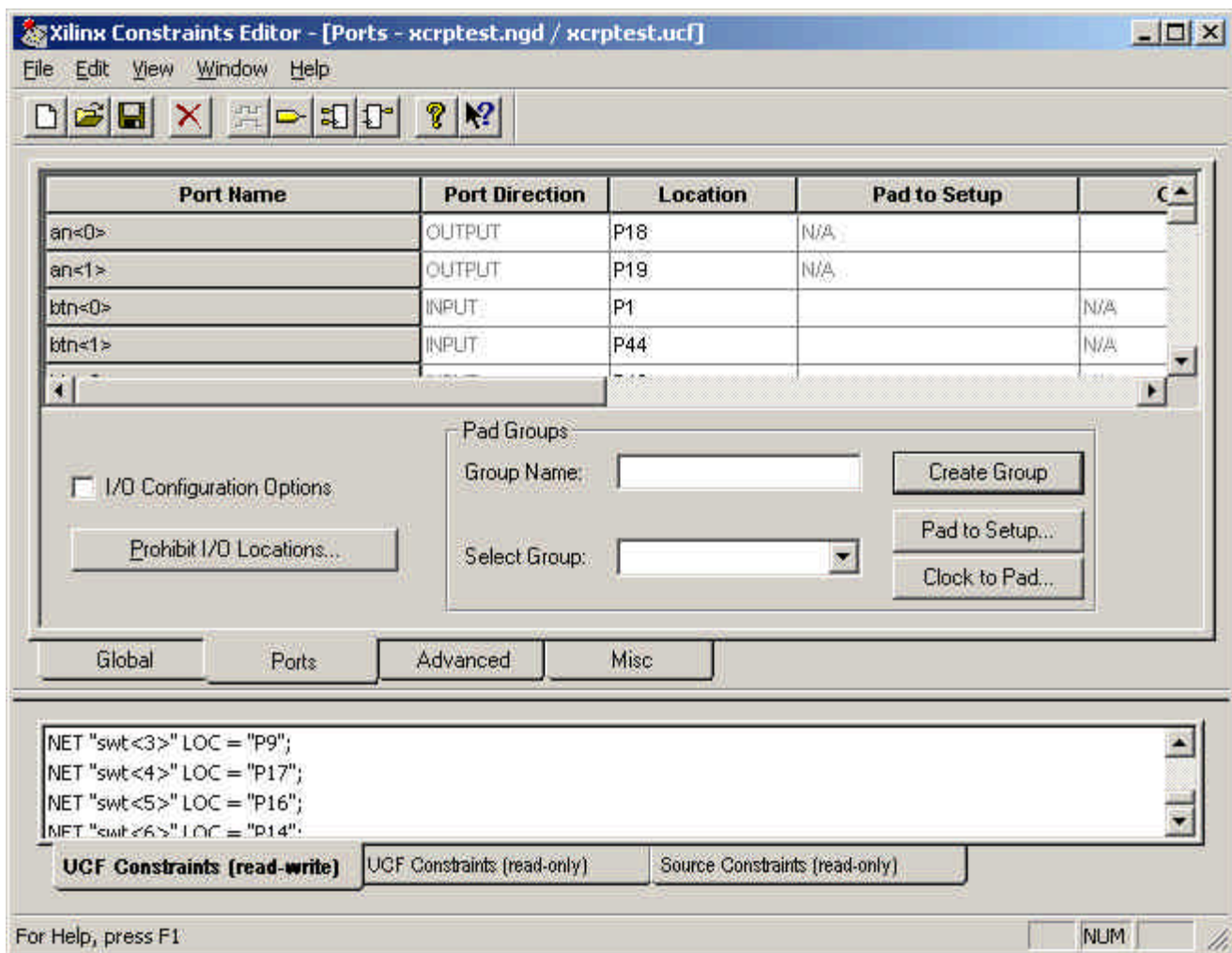
Assigning physical pin definitions

The Xilinx tools use an “User Configuration File” (or .ucf file) to define user constraints like physical pin to circuit net mappings. Information can be added directly to the .ucf file using a text editor, or indirectly using a windows interface called the Xilinx Constraints Editor. To use the text editor, select the top-level source file in the *Sources in Project* panel of the project navigator, and then select “Edit Implementation Constraints File” in the *Processes for Current Source* panel. This will open an editor, and load a standard .ucf file template that is full of useful information in comment fields. To associate a physical pin with a given net name, type: NET “netname” LOC = “PXX” on a separate line in the .ucf file. In the statement, “netname” (quotes included) is the name of the net to attach to pin number XX (quotes and the letter P included).



To use the more user-friendly Constraints Editor to assign pin mappings, select the top-level source file in the *Sources in Project* panel of the project navigator, and then select “Edit Implementation Constraints (Constraints Editor)” in the *Processes for Current Source* panel. This brings up the Constraints Editor tool shown below. When the “Ports” tab is selected, all top-level signals in the design are shown in the leftmost “Port Name” column in the window. The next two columns can be used to define signals that are to connect to physical pins on the chip. A signal can be connected to a physical pin on the Xilinx chip by simply providing the Port Direction (Input or Output) and Location (PXX, where XX is the pin number) in the appropriate boxes. Pin numbers for all boards can be found in the appropriate Reference Manual at the Digilent website.

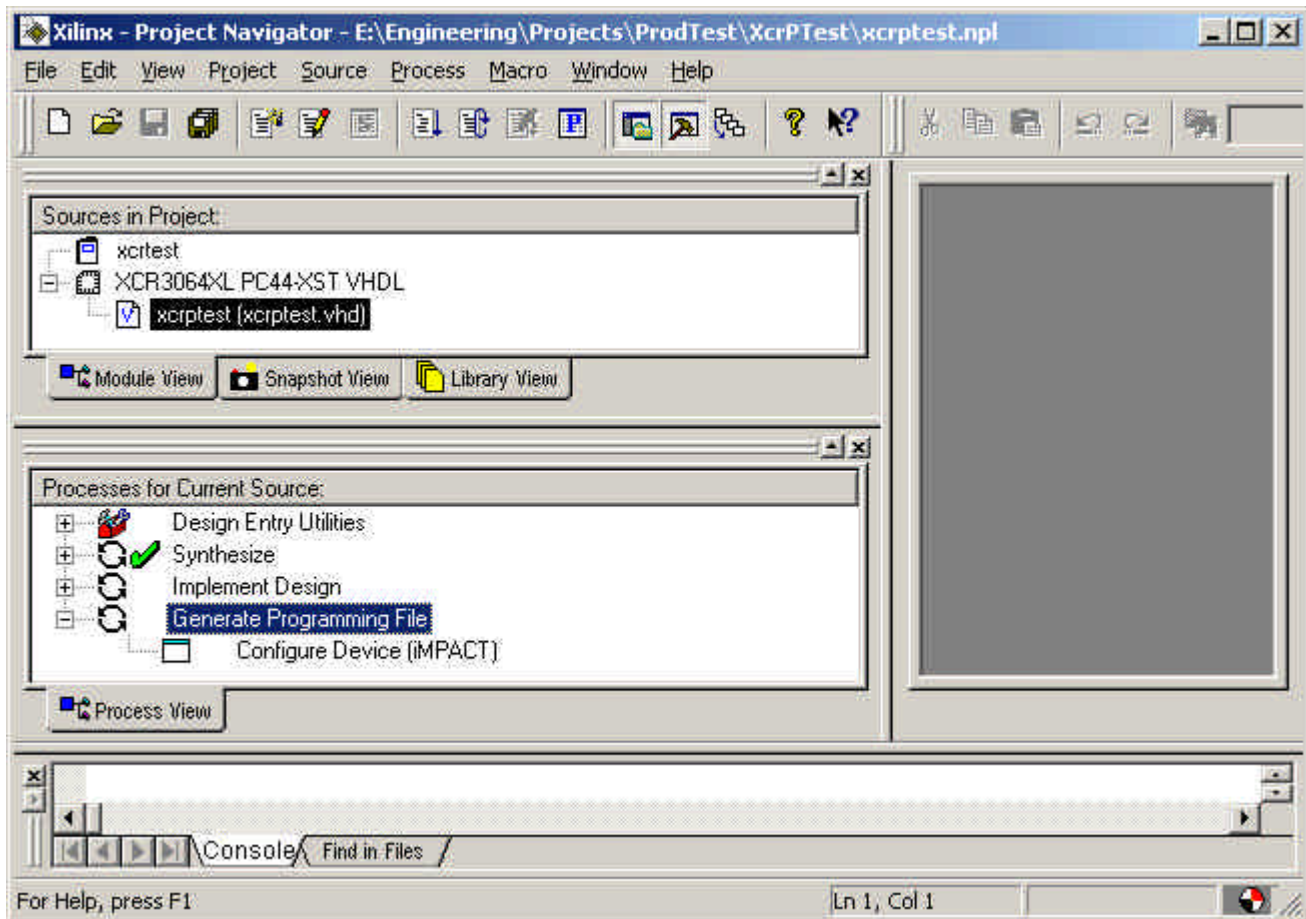
Whenever the .ucf file is modified, a dialog box is displayed that asks whether a “reset” should be performed. In general, a reset should be chosen so that any changes can be incorporated into the configuration file that will be downloaded to the Xilinx chip.



After pin mappings have been defined, ensure the proper device is targeted. This information is available as the second entry in the *Sources in Project* panel of the project navigator. If that entry shows the incorrect device type (i.e., if the device type shown does not match the Xilinx chip type on the Digilab board), then right-click on the device name, select “properties”, and then select the proper Xilinx device.

As a final step before running the backend tools (here, “backend” tools refers to all CAD programs that process the design after it has been captured), ensure the “JTAG” configuration mode is selected (this step should be skipped when using CPLD based boards like the XCRP board). To set the JTAG configuration mode, select the top-level source file in the *Sources in Project* panel of the project navigator, and then right-click on the “Generate Programming File” entry in the *Processes for Current Source* panel. Select “properties” in the box that appears, and then select “JTAG” in the startup mode tab.

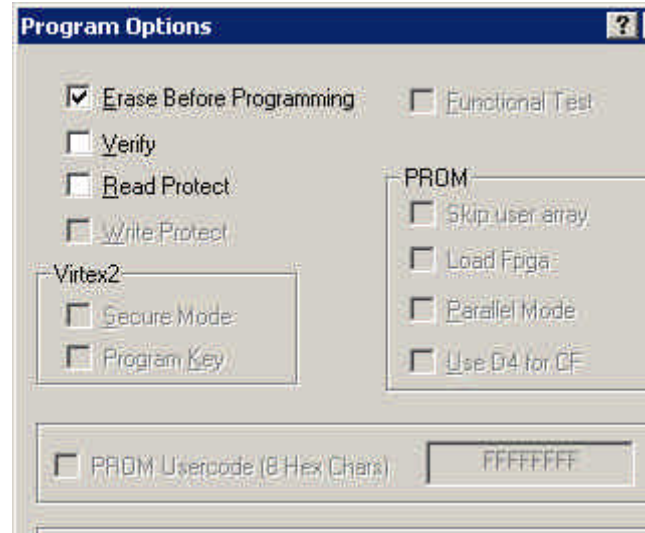
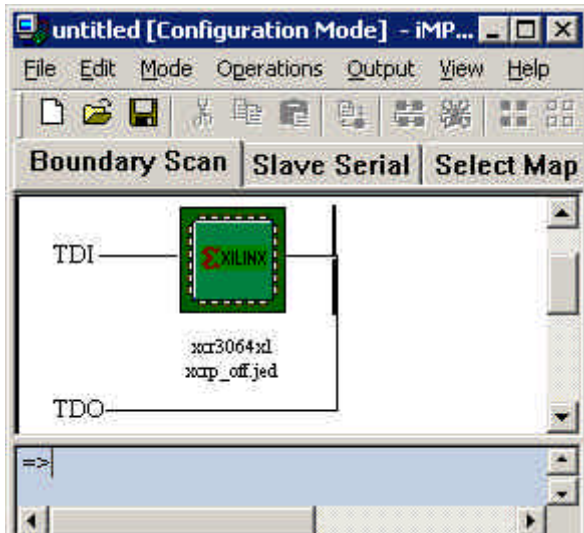
After the data discussed above has been entered, the backend tools can be run. These tools, “synthesize”, “implement design”, and “generate programming file” can be run individually by double-clicking on the appropriate process in the *Processes* panel, or they can be run together by clicking on the last process (i.e., “generate programming file”). The project navigator uses **make files**, which means that if any process is executed, the tools first check to see whether any data files required by that process have changed. If they have, any prerequisite processes are run before the selected process.



Programming the Xilinx chip

After generating a programming file, the final step is to configure the device using the Impact programmer. *Note that iMPACT 5.1 must be run with administrator privileges.* Before running the “Configure Device (iMPACT)” process, attach the programming cable to the PC and to the circuit board, and apply power to the board (this will allow the iMPACT programmer to automatically detect the board – otherwise, the board must be manually selected). Start the iMPACT programmer, and wait

until the board is identified. When the board and programming cable have been detected, an image of the Xilinx chip will be displayed in the programmer window. To download a design, right-click on the chip image, and select “Program” (the top-most entry). In the dialog box that opens, be sure the “Erase Before Programming” box is checked (if present), and click OK. This will transfer the configuration file from the PC to the Xilinx chip.



Using iMPACT as a stand-alone programmer

When ISE/WebPack is installed, the iMPACT programmer is installed as a stand-alone application, so that it can be used to download designs separately from the project navigator. Before starting iMPACT from the Windows start menu, ensure the file to be downloaded (a .jed file for a CPLD, or a .bit file for other Xilinx devices) is available somewhere in your directory structure. Also make sure the Digilab board is attached to the PC and powered on (and, if a JTAG/PORT switch exists, ensure it is in the JTAG position).

When iMPACT starts, it displays three dialog boxes (“Configure Device”, “Boundary Scan Mode”, and “Automatically connect to cable”); generally, the defaults should be chosen. Clicking finish in the third box displays a new dialog box where a configuration file can be chosen. Navigate to the desired file, and click “open”. Right click on the image of the Xilinx chip, and select “Program” (the top-most entry). For CPLD programming, be sure the “Erase Before Programming” box is checked, and click OK. The configuration file will be transferred from the PC to the CPLD.

More information

This presentation has been somewhat brief. For more information, see the tutorials available on the Xilinx website (www.xilinx.com and search on tutorial), and the ModelSim website (www.model.com).

