Symphony EDA

VHDL Simili

User's Manual

© Symphony EDA
10844 SW Nutcracker Ct.
Beaverton OR, 97007

Version 3.0
April 2005

License and Copyright

LICENSE. Symphony EDA hereby grants you, as a Customer and Licensee, non-exclusive license to use the accompanying software programs on a single CPU at any given point in time. The Licensee is bound by the terms of the license purchased and may use the software accordingly. A "Node-locked" license must be used on the computer for which the license was granted. A "Floating" license may be used on any compatible computer in a computer network in a local area network (LAN) configuration. Each "Floating" or "Node-locked" license grants the user permission to run a single instance of the "program" on the designated computer as described above.

Product(s) provided under this agreement are copyrighted and licensed (not sold). Symphony EDA and its Licensors retain title to the software and any patents, copyrights, trade secrets, and other intellectual property rights therein. Symphony EDA does not transfer title to the Products to Licensee.

TERM AND TERMINATION: This Agreement is effective from the date the software is received using any medium (downloaded from the internet/intranet, transferred via disks, etc.) until this Agreement is terminated. Any unauthorized reproduction or use of the Program and/or documentation will immediately terminate this Agreement without notice. Upon termination you are required to destroy/un-install the installed software and documentation.

COPYRIGHT AND PROPRIETARY RIGHTS: Both United States Copyright Laws and International Treaty provisions protect the Program and documentation. This means that you must treat the documentation and Program just like a book, with the exception of making archival copies for the sole purpose of protecting your investment from loss. The Program may be used by any number of people, and may be moved from one computer to another, so long as there is No Possibility of its being used by two people at the same time.

RESTRICTIONS: The Software contains copyrighted material, trade secrets, and other proprietary information. You may not de-compile, reverse engineer, disassemble, or otherwise reduce the Software to a human-perceivable form. The Software manages the licensing using license management software from GLOBEtrotter Software, Inc. You may not reverse engineer the license management software. You may not modify the hardware or software configuration of your computer to create a false identity (host-id) for your computer. You may not modify or prepare derivative works of the Software in whole or in part.

DISCLAIMER AND LIMITATION OF LIABILITY: THIS SOFTWARE AND DOCUMENTATION ARE LICENSED "AS-IS", WITHOUT WARRANTY AS TO PERFORMANCE. SYMPHONY EDA EXPRESSLY DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS OF THIS PROGRAM FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL SYMPHONY EDA BE HELD LIABLE FOR ANY DAMAGES DUE TO THE USE OR INABILITY TO USE OF THE SOFTWARE DISTRUBUTION OR PARTS THEREOF, EVEN IF SYMPHONY EDA HAS BEEN ADVISED OF THE POSSIBILITIES OF SUCH DAMAGES

RESELLING: Only Symphony EDA authorized distributors can do Reselling or distribution of this product.

EXPORT RESTRICTION: You agree that you will not export or re-export the Software, reference images or accompanying documentation in any form without the appropriate government licenses.

Table of Contents

Chapter 1

1 Introduction

This manual includes documentation for both the free and commercial versions of VHDL Simili. VHDL Simili is a collection of tools that facilitate the design, development and verification of hardware models using VHDL, a Hardware Description Language. There may be sections of this manual that are not applicable to you depending on the particular type of license you have purchased (including the free license).

The manual is not intended to serve as a VHDL language reference. However, to follow these documents, you will need only very basic knowledge of VHDL - for example, you only need to know about keywords such as "`entity`", "`architecture`", "`package`" and "`library`".

VHDL Simili is supported on the following platforms

- Red Hat Linux 7.1 or higher running on Intel x86 platforms
- Windows XP/2000 or higher running on Intel x86 platforms

This manual is organized into the following chapters:

1.1 Conventions used in this manual

Throughout this manual, the '/' (slash) and '\' (backslash) characters are used interchangeably in path names. While this same convention is followed by the Simili toolset on Windows, only forward slashes are understood on the Unix platform as directory separators. The use of *italicized text* (***sometimes in bold***) indicates emphasis and **normal bold text** is used for commands and keywords. A `monospaced type` is used to illustrate code examples and path names.

1.2 Getting Started - Quick Start

While you are encouraged to read the entire documentation, if you are looking for a quick start please follow the recommendations listed below:

- ***Before you begin***. After installing the software for the first time, if you are interested in the Graphical Tools (demo or purchased), use the command "**`sonata -L`**" on Unix or use "`Start->Programs->Symphony EDA->VHDL Simili 3.0->License Management->Symphony EDA Licensing Wizard`" on Windows to acquire a license file. You do not need to do anything special if you only wish to use the free edition (unless you are using a very old release). Once you have a license file, Windows users are highly encouraged to read the section "Configuring the license server software" before configuring the License Management Software (LMTOOLS). Complete details of license-management are documented in the "License Management" chapter - users of floating licenses may need to read this chapter.

- ***Please take the time to go through the** [Sonata tutorial](#)*. It should take you approximately 15-30 minutes to complete this tutorial. It will give you a pretty good feel for finding your way through the Graphical User Interface and potentially save you hours of frustration later.

- ***Need to know all about Sonata***? Please read chapters "Sonata" and "Simulation/Waveforms with Sonata". This chapter is recommended to know more about Sonata's capabilities.

- ***What about command-line tools***? The command line tools are very useful for automated batch runs. Please go through the tutorials documented in <install-dir>/examples/index.htm. Also, read chapters "VHDL Simili Environment", "VHDL Compiler Reference (vhdlp)" and "Simulator Reference (vhdle)".

- ***What about my vendor libraries?*** This may be important if you third party vendors such as Altera, Xilinx, etc. After installing the software and enabling the appropriate license, you will have to compile the vendor libraries to be able to use Simili with designs targeted for those vendors. Please refer to `<install-dir>/lib/index.htm` for more details.

1.3 Other VHDL Resources

There are some excellent books on the market that can teach you VHDL. If you are a VHDL beginner, a good book may be an invaluable resource alongside VHDL Simili. These books can be immensely helpful in teaching you syntax, explaining the semantics and concepts. Some of these books also have practical points of view -- such as coding styles compatible with synthesis, techniques for writing effective test-benches, etc.

You can also check out other valuable resources available on the Internet - http://www.vhdl.org, news://comp.lang.vhdl, the famous VHDL FAQ, http://www.ieee.org, etc.

Chapter
2

2 Installation

Installation of VHDL Simili creates the appropriate directories, commands and shortcuts to be able to use Simili. ***However, the installation may not unlock the product you have purchased.*** After installation, you may have to acquire and install the appropriate license file(s) before the software can be used to its fullest extent. Refer to the license management section of this manual for more details.

☑ The free edition license file for VHDL Simili is pre-packaged so there is typically no need to read the license management section. However, if you have lost this license file or if this license file has expired, you may need to request a new one.

☑ The installation includes support for several third party or vendor libraries (Altera, Xilinx, etc.). To avoid incompatibilities and to minimize the download size, libraries that change often are not pre-compiled and do not ship with the software. Refer to the documentation for vendor libraries for information on each of the supported vendors. Although Simili ships with support for a few vendors you can just as easily use one of the vendor directories under "`<installation-dir>/lib`" directory as a template to create support for the vendor of your choice. It is recommended however that you create your vendor library directories outside the installation directory whenever possible.

2.1 Windows

All modern Windows platforms (Windows 98 or newer) are supported. Although Simili tools may install and run in Windows 95, this is not an officially supported platform. On Windows 2000/NT4, you will need administrative privileges to install VHDL Simili properly.

The Windows installation procedure is completely automated. Simili's installation uses the popular InstallShield program to perform installation. Simply double click on the downloaded EXE file and following the instructions of the installation wizard. You can also unzip the downloaded EXE file using WinZIP into a temporary empty directory and then run setup.exe from that directory. The following are some additional notes for Windows users:

- Expert users: If you use the command line tools and if you do not want to use the "VHDL Simili Shell" provided during installation, then you may consider the following:
    - Optional: Chose an installation directory that has a simple path name (path name that is short and without spaces) instead of accepting the default path during installation. This can make it easier to embed the path into your environment.

    - Modify your PATH environment variable to include the "`<installation-dir>\tcl\bin>`" directory. This directory contains all relevant executables and DLLs. Note that on Windows 98/ME, you will have to modify the c:\autoexec.bat file while in Windows NT/2000 you can change your user environment to accomplish the same using "`Start->Settings->Control Panel->Advanced->Environment Variables`". Please contact your system administrator for further help on this.

- To un-install the software, use "`Start->Control Panel->Add/Remove Programs`". After un-installing VHDL Simili, it is possible that some files and directories are not removed because they were created after installation. This is typically a result of compiling vendor libraries, running a tutorial or installing a new license file, etc. It is safe to remove any left over files or directories if you so desire. However, you may wish to backup your license file in case you need it afterwards.

- Some versions of virus checkers interfere with the installation process. If you are experiencing problems installing Simili, try disabling the virus detection software for the duration of the installation.

2.2 Unix/Linux

Currently, the only Unix/Linux platform supported is Red Hat 7.1. Other versions or distributions are currently not supported. While you are free to try other versions/distributions of Linux, we do not have the resources to test/debug/support the many combinations of Linux-kernel/glibc/gcc/pthread available today.

The installation procedure requires the following steps:

1. Create a temporary directory with enough room for storing approximately 30MB. For the sake of these instructions, we will assume that this directory is called **/tmp/Simili**.

2. Download the appropriate file for your platform to **/tmp/Simili** directory. For these instructions, we will use the file "`Simili20b34-linux-x86.tar`" as an example and we will assume that this file was placed in **/tmp/Simili**. **Note:** The typical filename format of a Simili download file is "`SimiliXXbYY-os-machine.tar`" where `XX` is the version of Simili; `YY` is the build number for that version and `operating-system/machine` describe the actual platform.

3. Execute the following commands (when prompted for the installation directory, remember to select a directory with at least 50MB of available disk space). The install script will step you through the installation process.

```
cd /tmp/Simili
tar xfz /tmp/Simili20b34-linux-x86.tar
./install.sh
```

4. For each user who wishes to use the Simili toolset add the following command(s) in your shell initialization files (replacing "`/usr/local/Simili30`" with your actual installation directory):

   ```
   bash, sh users (File: ~/.bashrc or ~/.profile):

          export SYMPHONYEDA=/usr/local/Simili30
          . $SYMPHONYEDA/bin/init.sh

   csh users (File: ~/.cshrc):

          setenv SYMPHONYEDA /usr/local/Simili30
          source $SYMPHONYEDA/bin/init.csh
   ```

   These init scripts simply set the PATH and the LD_LIBRARY_PATH environment variables based on the instllation location given by $SYMPHONYEDA. If you chose not to use these init scripts, please atleast ensure that the equivalent of the init scripts is done to your environment (and such customizations become your responsibility and will not be supported by Symphonyt EDA).

5. Logout and log back in to ensure that the Simili tools are in your environment. ***Now you are ready to use the VHDL Simili toolset***. Strictly speaking, you do not have to logout and then login but it is safer to do so, so that all your shell windows are initialized the same way.

6. You need a license file to run the tools. Please refer to the License Management chapter for more details.

If you use a shell other than the ones listed above (ksh, tsh, tcsh, etc.), please inspect the `init.*` files used in the above instructions and create an equivalent one for your shell. Symphony EDA and other users who use this same shell will appreciate it if you report back these details so that future installations can include support for this shell.

To un-install the software on a UNIX/LINUX platform, simply delete the Simili installation directory. Please remember to backup your license file (typically in `<install-dir>/bin/symphony.lic`) if you are planning on using it later.

Chapter 3

3 License Management

Before VHDL Simili can be used to its fullest extent, a license file has to be acquired and the license management software has to be properly configured. The process of downloading and installing the software simply copies all the necessary software on to your computer. However, unless the license management software is configured properly and a proper key is used to enable the software, Simili will not be fully functional.

Symphony EDA has created tools to work in a client/server mode to make license fulfillment and installation as smooth and fast as possible. Using our license management tools, you can be up and running in less than 5 minutes from the point you make a license request.

☑ Windows and Linux users on Intel x86 platforms must have a Network Information Card (NIC) installed. This NIC (its MAC address) provides a unique way of identifying your computer. The licensing mechanism does not use hardware dongles. A NIC is not a requirement for the "Free" version.

☑ With Version 2.1, the license mechanism has been upgraded and as such, older license files are not compatible with version 2.1. You need to request/download a newer license file that is compatible with Version 2.1 (or higher) using one of the methods outlined in this chapter. Version 3.0 is fully compatible with license files generated with version 2.1/2.2/2.3.

The license management software used by VHDL Simili is the industry standard FLEX*lm* from GLOBEtrotter Software Inc. FLEX*lm* is capable of supporting many models of licensing and is very configurable. While some system administrators may need to know the intimate details of this software this chapter attempts to provide the simplest possible set of instructions to get past the license installation process.

If you have purchased a license for Version 2.1 or older or if you have purchased floating licenses, then it is highly recommended that you read this entire chapter before proceeding with the Licensing Wizard. While the Licensing Wizard is self-documenting, it may still be helpful to read this chapter to understand the overall process.

If you install the license file in the default location (<install-dir>/bin/symphony.lic), it is highly recommended that you backup the license file.

3.1 The Licensing Wizard

The VHDL Simili toolset ships with a special utility called the "Licensing Wizard". This utility can be used to get most types of licenses (free, demo, node-locked, floating, etc.). By answering a few questions you will be able to step through the process of downloading an appropriate license file and even installing it in the pre-designated location.

Symphony EDA has taken many steps to ensure smooth and quick processing of license fulfillment. However, there are things that can go wrong in some networking configurations. If the Licensing Wizard fails for you, you can still resort to using email for license fulfillment. Refer to the manual license fulfillment section for more information.

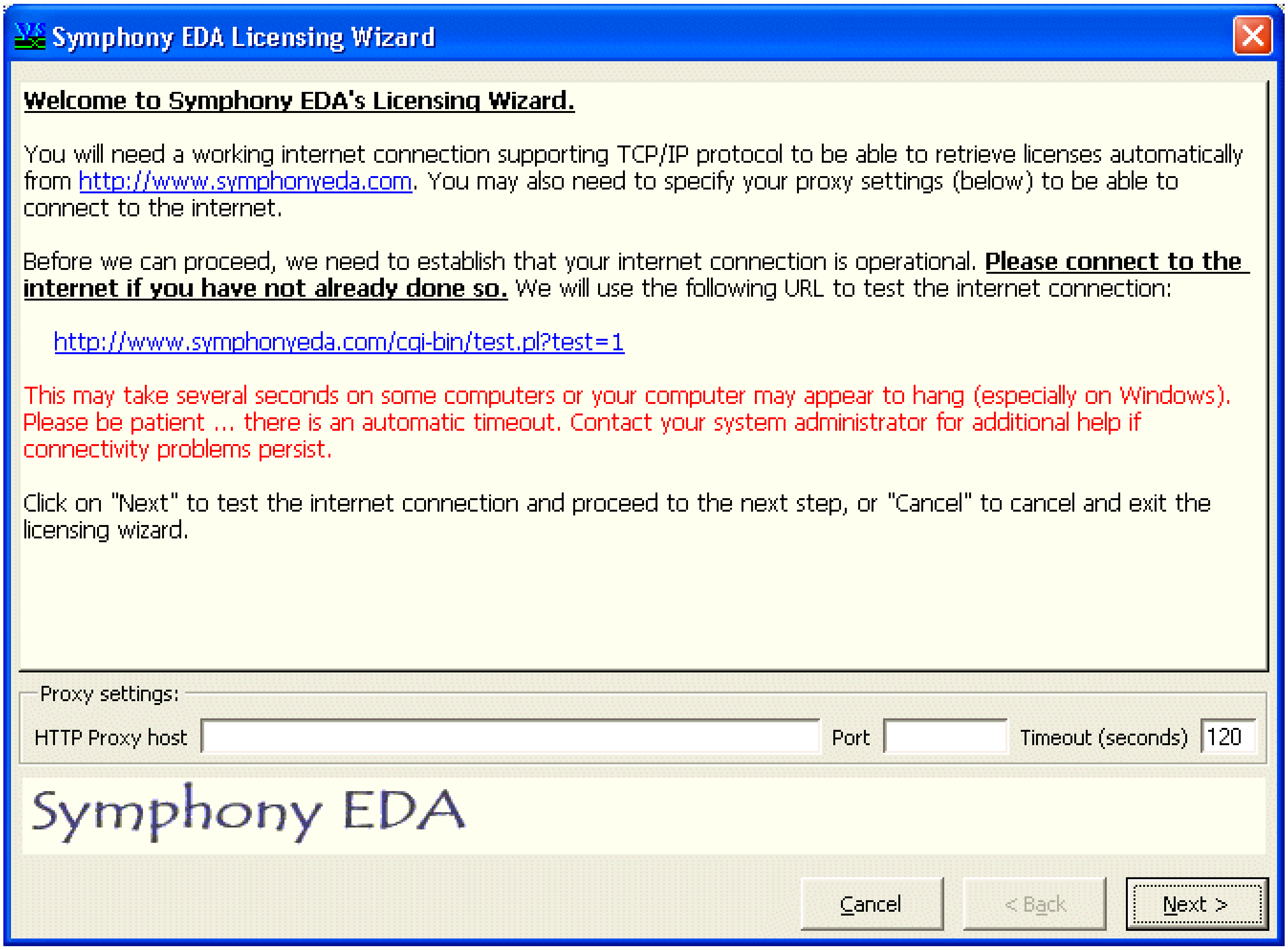
To save some work installing the license file, you may wish to run the Licensing Wizard as a user who has write-permission to the installation directory. The Licensing Wizard can be accessed in ***many*** ways.

- Windows users: Using "`Start->Run`", enter "**`sonata -L`**".
- Windows users: Launch Sonata by using the "`Start->Programs->Symphony EDA->VHDL Simili X.Y->License Management->Symphony EDA Licensing Wizard`" (where `X.Y` is the version number).
- Unix users: Make sure the program "`sonata`" is in your path (following the instructions in the installation manual) run

  ```
  sonata -L
  ```

- Windows and Unix users: If you already have "`sonata`" running, then you can access the Licensing Wizard under the "`Help`" menu.

The following is the initial screen that you will see:



Symphony EDA Licensing Wizard

Welcome to Symphony EDA's Licensing Wizard.

You will need a working internet connection supporting TCP/IP protocol to be able to retrieve licenses automatically from http://www.symphonyeda.com. You may also need to specify your proxy settings (below) to be able to connect to the internet.

Before we can proceed, we need to establish that your internet connection is operational. **Please connect to the internet if you have not already done so.** We will use the following URL to test the internet connection:

http://www.symphonyeda.com/cgi-bin/test.pl?test=1

This may take several seconds on some computers or your computer may appear to hang (especially on Windows). Please be patient ... there is an automatic timeout. Contact your system administrator for additional help if connectivity problems persist.

Click on "Next" to test the internet connection and proceed to the next step, or "Cancel" to cancel and exit the licensing wizard.

Proxy settings:

HTTP Proxy host

Port

Timeout (seconds) 120

Figure 3-1: Licensing Wizard's initial screen that collects information that allows the wizard to connect to the Internet.

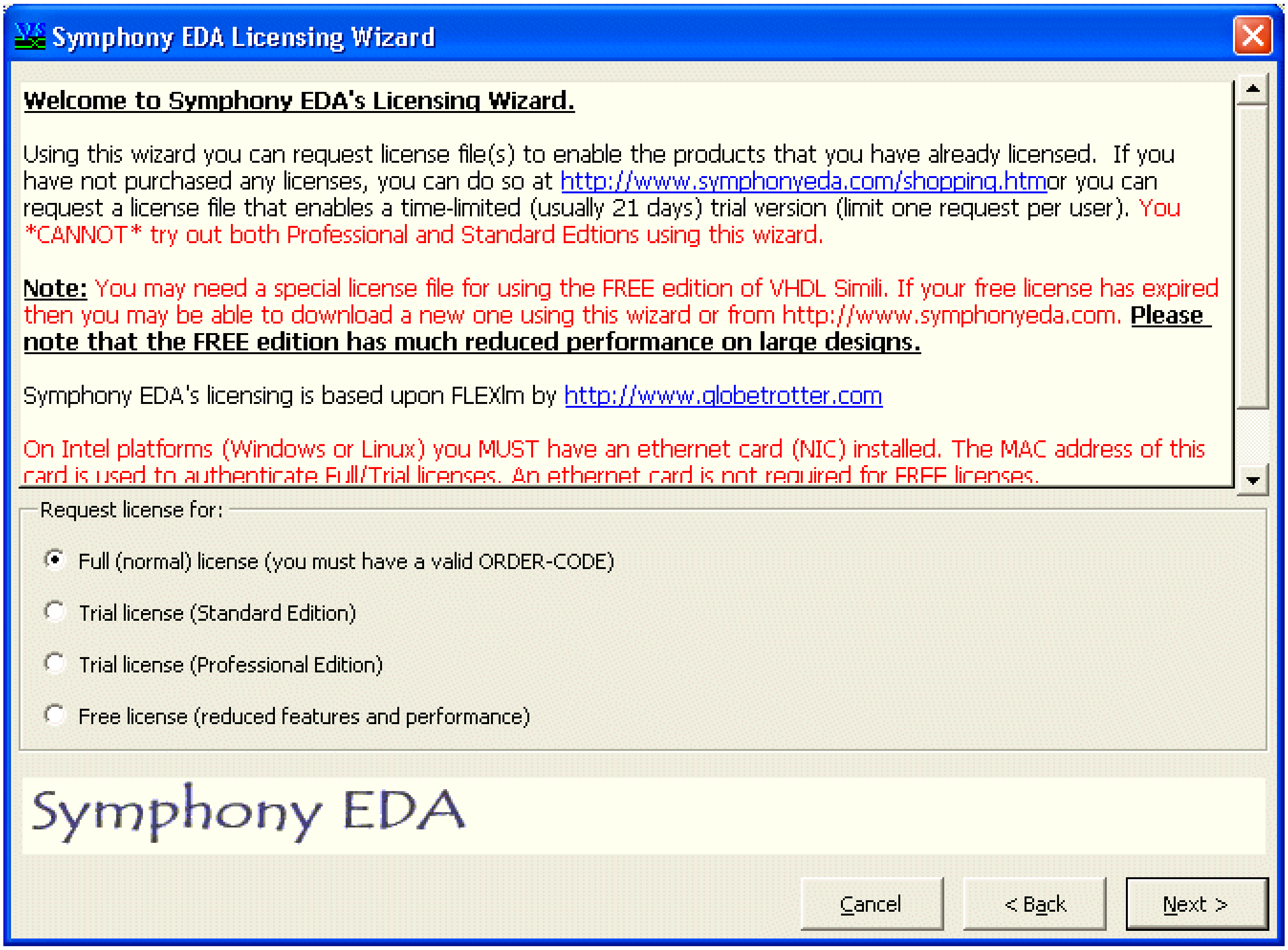
The main purpose of the initial screen is to try to communicate with the license fulfillment server at www.symphonyeda.com. If you use a dialup-connection, go ahead and connect to the Internet. The Licensing Wizard does not function without an active Internet connection.

☑ If you are a Windows Internet Explorer user, the very first time the Licensing Wizard is launched, it will automatically detect your proxy settings from your Internet Explorer settings. This is not done on subsequent runs.

☑ If you need a proxy server to access servers on the Internet, you must specify your proxy settings. Most home users do not need any proxy settings. If you have a slow connection to the Internet, make sure that you specify an adequate timeout. Using a 56K modem approximately 30 seconds should be more than adequate in the Continental United States. Use a larger timeout at peak times.

The Licensing Wizard will now determine if it can access the license fulfillment server at www.symphonyeda.com using your Internet connection settings. Downloading a test HTML file with specific (expected) contents does this. If this fails, the Licensing Wizard cannot be used to fulfill license requests. In case a connection cannot be established, please contact your system administrator and as a last resort follow the "Manual License Fulfillment" process.

The next step is selecting the correct type of license you are trying to request from the License fulfillment server



Welcome to Symphony EDA's Licensing Wizard.

Using this wizard you can request license file(s) to enable the products that you have already licensed. If you have not purchased any licenses, you can do so at http://www.symphonyeda.com/shopping.htmor you can request a license file that enables a time-limited (usually 21 days) trial version (limit one request per user). You *CANNOT* try out both Professional and Standard Edtions using this wizard.

Note: You may need a special license file for using the FREE edition of VHDL Simili. If your free license has expired then you may be able to download a new one using this wizard or from http://www.symphonyeda.com. **Please note that the FREE edition has much reduced performance on large designs.**

Symphony EDA's licensing is based upon FLEXlm by http://www.globetrotter.com

On Intel platforms (Windows or Linux) you MUST have an ethernet card (NIC) installed. The MAC address of this card is used to authenticate Full/Trial licenses. An ethernet card is not required for FREE licenses.

Request license for:

- ● Full (normal) license (you must have a valid ORDER-CODE)
- ○ Trial license (Standard Edition)
- ○ Trial license (Professional Edition)
- ○ Free license (reduced features and performance)

Figure 3-2: Select the license type

☑ You must have a valid ORDERCODE to proceed with the "Full" version. Also, you must have a NIC card installed on Windows and Linux platforms for the Demo and Full licenses (even if your computer is the only computer on the network). ORDERCODEs are issued when you purchase a License for VHDL Simili.

There are three categories of licensing for VHDL Simili: You can skip directly to the appropriate licensing category:

- Full/Licensed (purchased) license: You must have a valid ORDERCODE and a Network Card installed on your computer.

- Demo/Trial license: You must have a Network Card installed on your computer to enable a Demo/Trial version. You are limited to one license request. You can either try out the "Standard Edition" or the "Professional Edition" but not both.

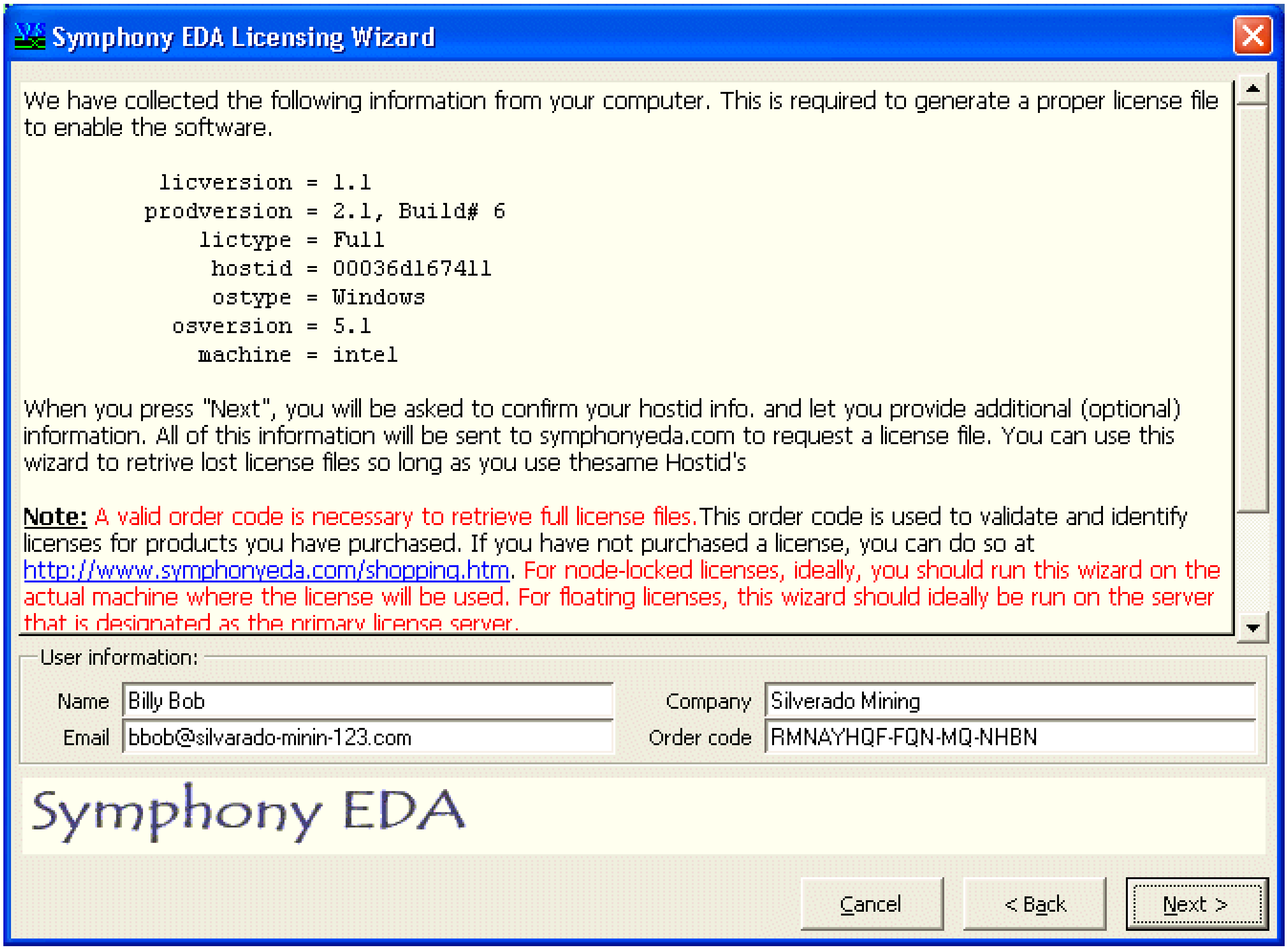
- Free edition license: You do not need a Network Card.

3.2 Full/Licensed (purchased) license

There are primarily two kinds of licenses currently supported by the VHDL Simili licensing policy. The ORDERCODE determines the type of license you own. ONCE A LICENSE IS GRANTED TO A PARTICULAR COMPUTER'S HOSTID, THIS CANNOT BE ALTERED, SO PLEASE MAKE SURE THAT YOU ARE REQUESTING A LICENSE FOR THE CORRECT COMPUTER. The two types of license are

1. Node-locked license: ***For node-locked licenses, it is very important that you run this Wizard on the computer that you intend to run the software on***. Node-locked licenses will use the current computer as the license server as well as the client. You can request licenses repeatedly and the License fulfillment server will honor such a request in most instances so long as the host-id of the current request matches the host-id of the very first request for your ORDERCODE.

2. Floating license: You can run this wizard on any computer on the network but you must have valid HOSTIDs available for your licensing server(s) if you are running the wizard on a computer other than the computer intended to be the license server.



We have collected the following information from your computer. This is required to generate a proper license file to enable the software.

```
         licversion = 1.1
        prodversion = 2.1, Build# 6
            lictype = Full
             hostid = 00036d167411
             ostype = Windows
          osversion = 5.1
            machine = intel
```

When you press "Next", you will be asked to confirm your hostid info. and let you provide additional (optional) information. All of this information will be sent to symphonyeda.com to request a license file. You can use this wizard to retrive lost license files so long as you use thesame Hostid's

Note: A valid order code is necessary to retrieve full license files.This order code is used to validate and identify licenses for products you have purchased. If you have not purchased a license, you can do so at http://www.symphonyeda.com/shopping.htm. For node-locked licenses, ideally, you should run this wizard on the actual machine where the license will be used. For floating licenses, this wizard should ideally be run on the server that is designated as the primary license server.

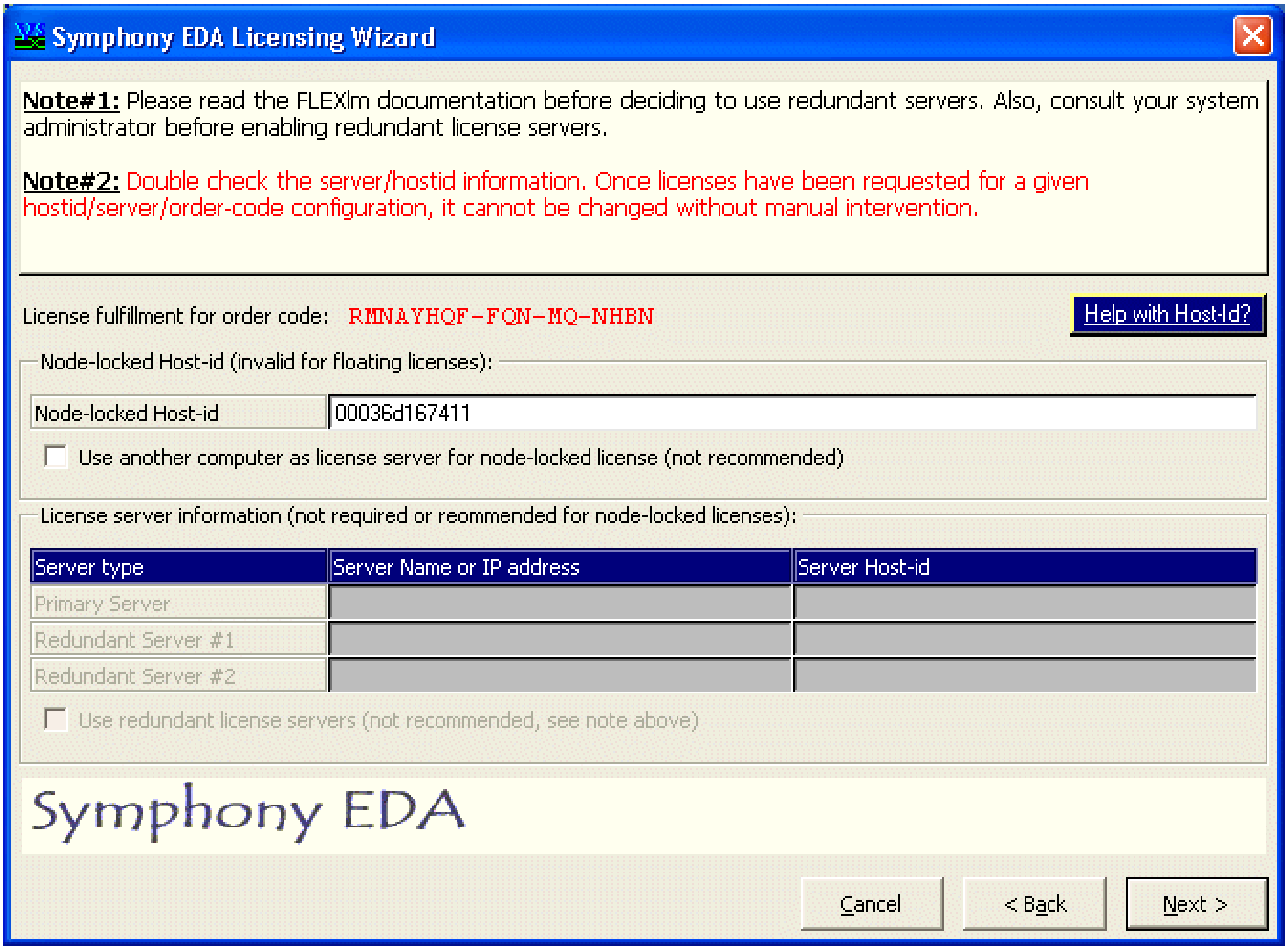
User information:

Name	Billy Bob	Company	Silverado Mining
Email	bbob@silvarado-minin-123.com	Order code	RMNAYHQF-FQN-MQ-NHBN

Figure 3-3: Enter the ORDERCODE for a full/purchased license. The order code is the key to downloading the license.

3.2.1 Node-locked Licenses

Management of node-locked licenses is very simple. The same computer can serve as a client and a server for the license management (in this case, license management is the same procedure used for demo/trial mode). If your order code is valid, you advance to the next screen in the Licensing Wizard that resembles the following:



Note#1: Please read the FLEXlm documentation before deciding to use redundant servers. Also, consult your system administrator before enabling redundant license servers.

Note#2: Double check the server/hostid information. Once licenses have been requested for a given hostid/server/order-code configuration, it cannot be changed without manual intervention.

License fulfillment for order code: RMNAYHQF-FQN-MQ-NHBN

Node-locked Host-id (invalid for floating licenses):

Node-locked Host-id: 00036d167411

□ Use another computer as license server for node-locked license (not recommended)

License server information (not required or reommended for node-locked licenses):

Server type	Server Name or IP address	Server Host-id
Primary Server		
Redundant Server #1		
Redundant Server #2		

□ Use redundant license servers (not recommended, see note above)

Figure 3-4: Page that allows you to enter host-id/server information.

It is highly recommended that you copy/install the license information to the pre-designated license file "`<installation-dir>/bin/symphony.lic`". ***Note: You are free to install the license file wherever you like so long as you configure the FLEXlm tools properly and use the environment variable SYMLM_LICENSE_FILE to point to the appropriate location.***

If you wish to use a license server that is different from the client computer where the software will be used, the procedure is the same as that used for managing "Floating" licenses. See below. Otherwise, you need to simply download and install the license file and restart Sonata to start using all the tools.

3.2.2 Floating Licenses

Provided the order code is correct, you will see the dialog box resembling Figure 3-4. Use one of your most reliable computers to serve as the license server. Once the license manager is enabled, any computer on your local-area-network (LAN) will now be able to checkout licenses. FLEXlm allows for redundant servers to manage and serve licenses (at least two out of three servers must be running). *Symphony EDA does not recommend the three-server configuration*. Please read the FLEXlm documentation about this topic before opting for the three-server configuration.

It is highly recommended that you copy/install the license information to the pre-designated license file "<installation-dir>/bin/symphony.lic". ***Note: You are free to install the license file wherever you like so long as you configure the* FLEX*lm tools properly and use the environment variable SYMLM_LICENSE_FILE to point to the appropriate location***.

When any of the Simili executables start up, the first thing they need to know is which license server they need to talk to get a license. This is done in one of the ways described below:

1. By default, all Symphony EDA programs examine the file "symphony.lic" located in the "<install-dir>/bin" directory. The SERVER lines in this file indicate which servers they have to contact to retrieve licenses. If you have installed the software on several computers, you will need to copy the license file to all the installation locations.

2. Use the SYMLM_LICENSE_FILE environment variable to point to the exact path-name to the license file. This will allow you to place the license file in a central location but letting individual users install the software wherever they wish.

3. You can use the SYMLM_LICENSE_FILE environment variable to point to a particular server and TCP/IP port. This is the fastest way of getting a license but it requires that you make a small modification to the default license file to include the exact TCP/IP port. The default license information supplied by Symphony EDA has a SERVER line in the that looks as follows

   ```
   SERVER    server-name-or-ip-address    server-hostid
   ```

   This line starts the license server on the first available TCP/IP port in the range of 2700-27009. You can add the exact port that you wish to use (typically any number above 2000 so long as no other application/service needs it). Consult your system administrator to choose an appropriate TCP/IP port. An example SERVER line that has a custom SERVER line looks as follows:

   ```
   SERVER    myserver    myhostid    myport
   ```

   For example,

   ```
   SERVER bullwinkle 00b0d08ada08 27009
   ```

   Once the license file is modified and the lmgrd server is restarted, the client computers' environment needs to be modified such that the environment variable SYMLM_LICENSE_FILE to myport@myserver. In the example above, the value of this variable would be 27009@bullwinkle.

4. Alternatively, if you chose not to modify the license file to add a specific port, you can set the SYMLM_LICENSE_FILE to @myserver. In the example shown above this value

would be `@bullwinkle`. This would cause the license request to search for all ports in the range 27000 to 270009.

5. If you are managing FLEXlm licenses from multiple vendors, it is recommended that you read the FLEXlm end user documentation.

On Windows, if you are configuring "lmgrd" using LMTOOLS, this is needed only on the server. The client computers only need to know the identity of the server (either via the license file in its default location or via SYMLM_LICENSE_FILE environment variable).

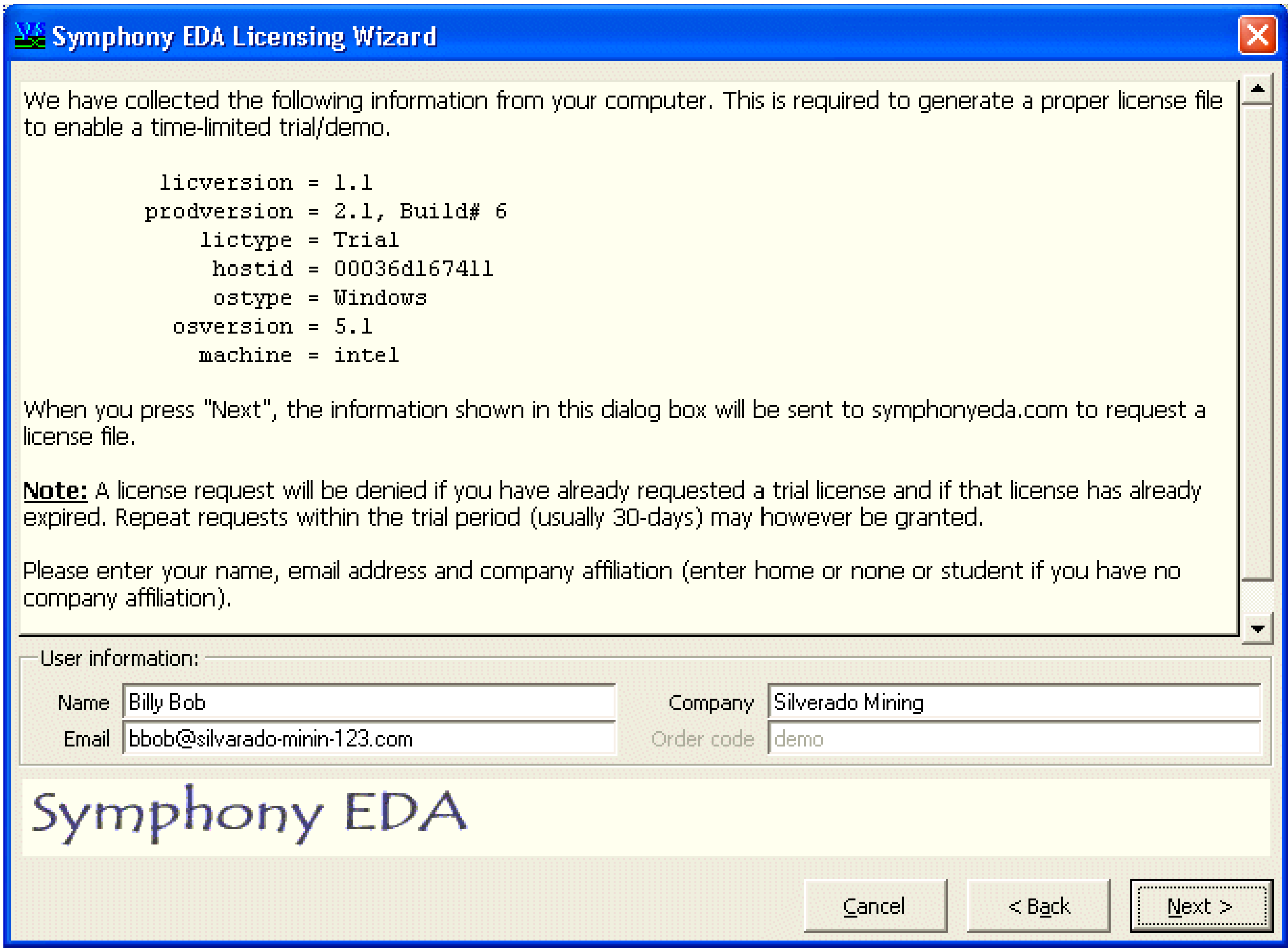
On Unix/Linux, the lmgrd daemon needs to be running on the server only. The client computers only need to know the identity of the server (either via the license file in its default location or via SYMLM_LICENSE_FILE environment variable).

Once the license file is installed and you have determined the method in which the client computers would access the license server, you need to configure the FLEXlm license manager. This is discussed in section 3.6.

If your DNS name-lookup is not configured properly, you can replace host-names with their IP addresses in the license file. This will avoid a DNS query but avoid using IP address for computers that are served via DHCP (if IP addresses are likely to change).

3.3 Demo/Trial license

You can request a demo or a trial license that enables the software for a limited amount of time. To get a demo license, you must have a Network Card installed (Windows and Linux). The Licensing Wizard requests a demo license on your behalf and if the demo program is still in effect, the License Server at Symphony EDA will grant such a license. You are allowed to request a demo license only once. To request a demo license the Licensing Wizard collects some information from your computer and before sending it to the license fulfillment server, displays it in the following dialog box:



We have collected the following information from your computer. This is required to generate a proper license file to enable a time-limited trial/demo.

```
  licversion = 1.1
 prodversion = 2.1, Build# 6
     lictype = Trial
      hostid = 00036d16741l
      ostype = Windows
   osversion = 5.1
     machine = intel
```

When you press "Next", the information shown in this dialog box will be sent to symphonyeda.com to request a license file.

Note: A license request will be denied if you have already requested a trial license and if that license has already expired. Repeat requests within the trial period (usually 30-days) may however be granted.

Please enter your name, email address and company affiliation (enter home or none or student if you have no company affiliation).

User information:

Name: Billy Bob

Email: bbob@silvarado-minin-123.com

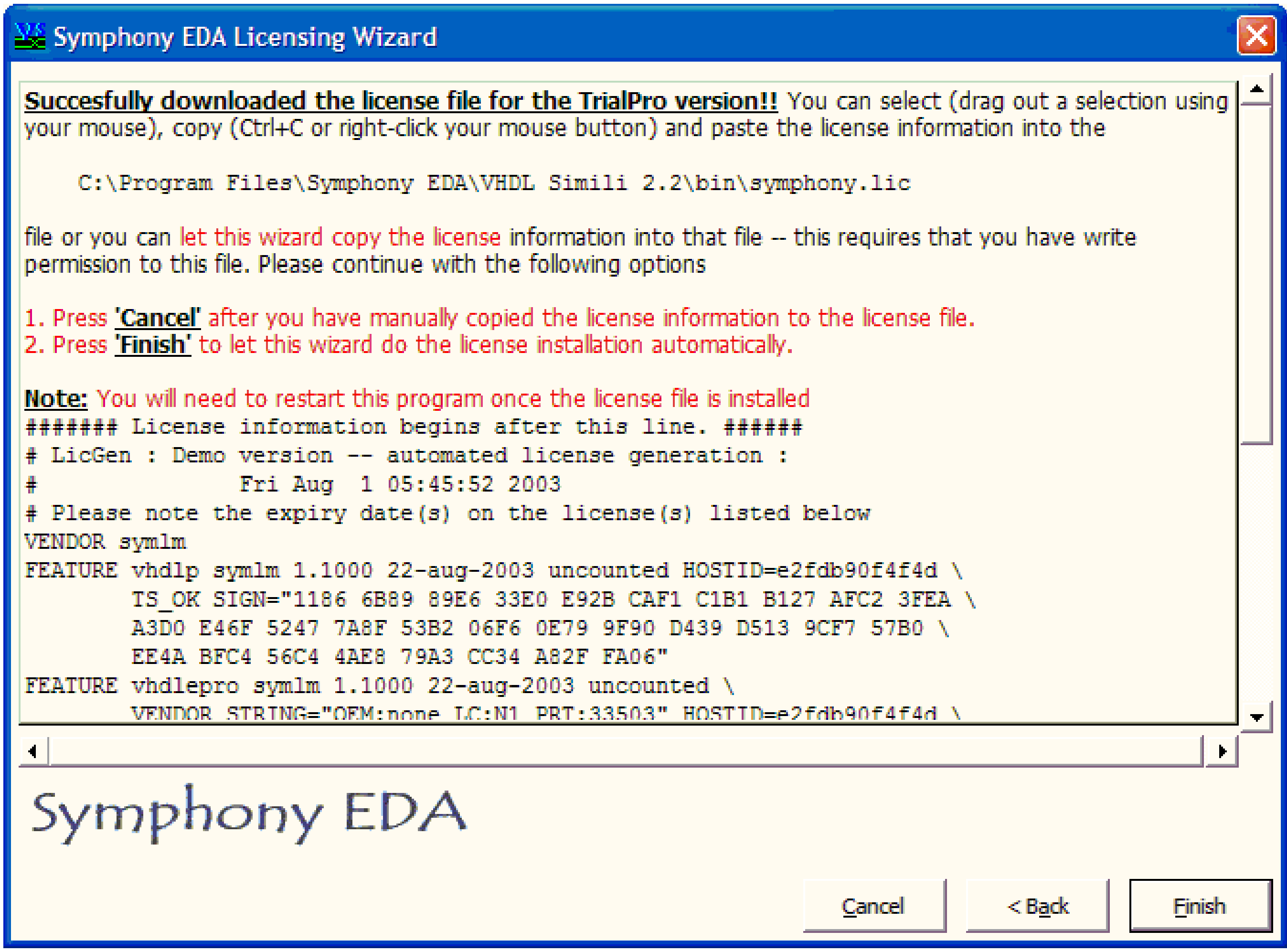
Company: Silverado Mining

Order code: demo

Figure 3-5: Demo license request screen

Using the license wizard, you can request a demo license only once. You must choose between the Professional and Standard editions for a trial license. Once you have downloaded a license for the given edition, you **cannot** request another kind of trial license using this wizard.

When the license fulfillment server issues a license, the following screen will appear:



Succesfully downloaded the license file for the TrialPro version!! You can select (drag out a selection using your mouse), copy (Ctrl+C or right-click your mouse button) and paste the license information into the

```
    C:\Program Files\Symphony EDA\VHDL Simili 2.2\bin\symphony.lic
```

file or you can let this wizard copy the license information into that file -- this requires that you have write permission to this file. Please continue with the following options

1. Press **'Cancel'** after you have manually copied the license information to the license file.
2. Press **'Finish'** to let this wizard do the license installation automatically.

Note: You will need to restart this program once the license file is installed

```
####### License information begins after this line. ######
# LicGen : Demo version -- automated license generation :
#                Fri Aug  1 05:45:52 2003
# Please note the expiry date(s) on the license(s) listed below
VENDOR symlm
FEATURE vhdlp symlm 1.1000 22-aug-2003 uncounted HOSTID=e2fdb90f4f4d \
        TS_OK SIGN="1186 6B89 89E6 33E0 E92B CAF1 C1B1 B127 AFC2 3FEA \
        A3D0 E46F 5247 7A8F 53B2 06F6 0E79 9F90 D439 D513 9CF7 57B0 \
        EE4A BFC4 56C4 4AE8 79A3 CC34 A82F FA06"
FEATURE vhdlepro symlm 1.1000 22-aug-2003 uncounted \
        VENDOR_STRING="OEM:none LC:N1 PRT:33503" HOSTID=e2fdb90f4f4d \
```

Figure 3-6: A successful retrieval of the demo license

At this point, you have the option to either let the Licensing Wizard copy the licensing information to the license file or you can copy and paste this information into the license file yourself. It is recommended that the Wizard do this automatically. If the Wizard is allowed to edit the license file, it will create a backup of any existing license file and update the license file with this new information.

Once the license file is updated, you need to restart Sonata. Note that the demo/trial license is usually a time-limited license, which expires in a pre-determined number of days.

3.4 Free Edition license

No special license file is needed for the free edition. When you download and install VHDL Simili, you are ready to run the tools because there is a default license file pre-packaged with Simili. Unless the download is very old, this license file should be current (it does have an expiry date). If for some reason you have lost this file or it has expired, you have the following options.

1. Re-install the VHDL Simili software. This will re-install the original license file (remember to remove any old licensee files before re-installing the software).

2. If your downloaded software is too old (expired) it may refuse to work even with newer licenses, you may wish to download and install the latest version.

3. Request a new license file using the Licensing Wizard. If the free edition is still being offered, the Licensing Wizard will help you get the license file and install it for you.

3.5 Manual License Fulfillment

If for some reason, the automated license fulfillment method (Licensing Wizard) cannot download the license for you, you may want to try again in a couple of hours (in case the problem is due to a temporary problem with the Internet connection).

If for some reason, the license fulfillment server refused to give a license, the most probable reason is that the ORDERCODE you have entered is wrong. Please double check the ORDERCODE.

When the license server at symphonyeda.com denies a license request, the license wizard in most instances will print instructions on how to proceed with license fulfillment using email.

If none of the above applies, continue reading the following sections. Since the license fulfillment via email is not automated (and thus time consuming), things can be expedited if the instructions listed below are followed very closely. Once the email for a license request is made, you can expect a reply in most cases, within one working day.

3.5.1 Free License

Download and install the latest release of VHDL Simili. If the free license policy is still in effect, you will get new licenses using this method.

3.5.2 Demo License

Demo licenses are essentially node-locked licenses. You have to decide between trying out the "Professional Edition" or the "Standard Edition" – by default, it is not possible to try out both editions. Follow the procedure listed for Node-Locked licenses below - except that the ORDERCODE is called "demo" for the Standard Edition and "demopro" for the Professional Edition.

3.5.3 Purchased Node-Locked License

We have to determine the HOST-ID of your computer. The HOST-ID can be determined using the following commands

3.5.3.1 WHAT IS MY HOSTID?

Windows:

```
        cd <install-dir>\bin
        .\lmhostid
```

Unix:/Linux (any hardware platform):

```
        cd <install-dir>/bin
        ./lmhostid
```

The HOST-ID on Windows and Linux (Intel platform) is a twelve (12) hexadecimal-digit address of your Network Card (NIC). One benefit of this is that a dual/multi-boot machine (Linux and Windows) can use the same license file. Other platforms differ but the HOST-ID must be a non-empty string of characters.

Prepare a text file/email with the following information (replacing the text between <> with actual information:

```
              lictype = <Enter Full or Trial>
               hostid = <Important: enter exact HOST-ID>
               ostype = <Windows or Linux>
             username = <Enter your username here>
              company = <Enter your company name here or enter none>
                email = <Important: enter your email address>
            ordercode = <Important: enter ORDERCODE>
```

The exact formatting (spaces) is not important so long as each line contains a "property = value" syntax. A sample request may look something like

```
              lictype = Full
               hostid = 01234567890ab
               ostype = Windows
             username = Billy Bob
              company = Silvarado Mining Co.
                email = bbob@silvarado-mining.com
            ordercode = XYZZY123456-93
```

Once this information is collected, email it to license@symphonyeda.com?subject=Full-license for purchased licenses or to license@symphonyeda.com?subject=Trial-license for demo/trial licenses. Please double check to ensure that the ORDERCODE, HOSTID and your EMAIL address are correct. In case the email hypertext links shown above do not work, ensure that the subject reads "Full-license" for purchased/full licenses and "Trial-license" for trial/demo licenses.

3.5.4 Purchased Floating License

1. First you have to identify the name (HOSTNAME or IP Address) and id (HOSTID) of the computer that is going to serve the floating licenses for your network. This is typically a server that can provide good response time to its client computers and has a tendency of

staying online and functioning for the most part. If your DNS is not properly configured, you can use the IP address of the server instead of its name. The procedure for determining your HOSTID is documented above. ***The server and client computers do not have to have the same operating system/hardware platform.***

2. **Optionally**, you can identify two additional servers that act as redundant servers. Symphony EDA does not recommend this but you are welcome to specify redundant servers. FLEXlm requires that two out of three servers must be up and running. Please read the FLEX*lm* documentation so that you understand how the redundant server policy is enforced before choosing redundant servers. If you decide you have redundant servers, collect their HOSTNAMEs and HOSTIDs.

3. Prepare a text file/email with the following information (replacing the text between <> with actual information:

```
   lictype = Full
   server1 = <Important: hostname-or-ip-address hostid>
   server2 = <Optional: hostname-or-ip-address hostid>
   server3 = <Optional: hostname-or-ip-address hostid>
  username = <Enter your username here>
   company = <Enter your company name here or enter none>
     email = <Important: enter your email address>
 ordercode = <Important: enter ORDERCODE>
```

   The exact formatting (spaces) is not important so long as each line contains a "property = value" syntax. A sample request may look something like

```
   lictype = Full
   server1 = myserver 01234567890ab
  username = Billy Bob
   company = Silvarado Mining Co.
     email = bbob@silvarado-mining.com
 ordercode = XYZZY123456-93
```

4. Email these details to mailto:license@symphonyeda.com?subject=Full-license. Please double check to ensure that the ORDERCODE, server information and EMAIL address is correct before emailing your request. The subject of the email should read "Floating"

3.6 Configuring the license server software

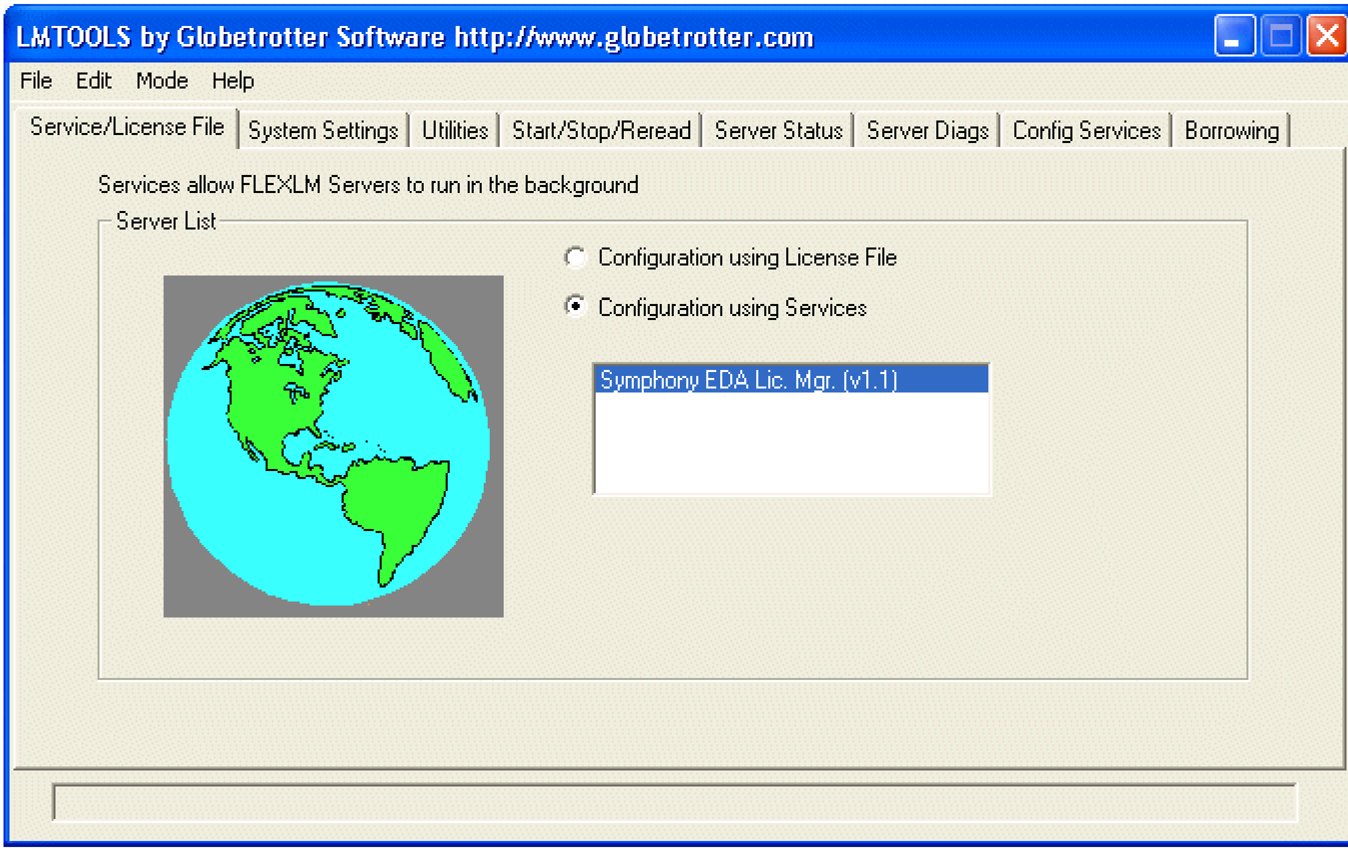
If your license file contains the keyword SERVER (this exists for floating licenses and licenses issued for pre-2.2 versions of VHDL Simili), then the FLEX*lm* license server (**`lmgrd`**) has to be configured appropriately before the licensing of the Simili is functional. You can skip this section nif you are using trial/demo or purchased node-locked licenses in normal mode. Although there are many ways of configuring the license server, the following sections document our recommendations. Skip to your platform specific instructions -- Windows or Unix/Linux.

If you are using FLEXlm license files from multiple vendors, then please read the section 3.7 before proceeding with the following instructions.

3.6.1 Windows

FLEX*lm* provides a handy little utility (LMTOOLS) to manage the software licenses. It can be invoked using "`Start->Programs->Symphony EDA->VHDL Simili X.Y->License Management->FLEXlm Configurator`". The Licensing Wizard may also launch this utility after it installs the license file. Once invoked follow these steps (you only need to do this once).

1. In the "`Service/License File`" tab, make sure that you have selected "`Configuration using Services`" and then select the "`Symphony EDA Lic Mgr.(v1.1)`" as shown below



Services allow FLEXLM Servers to run in the background

Server List

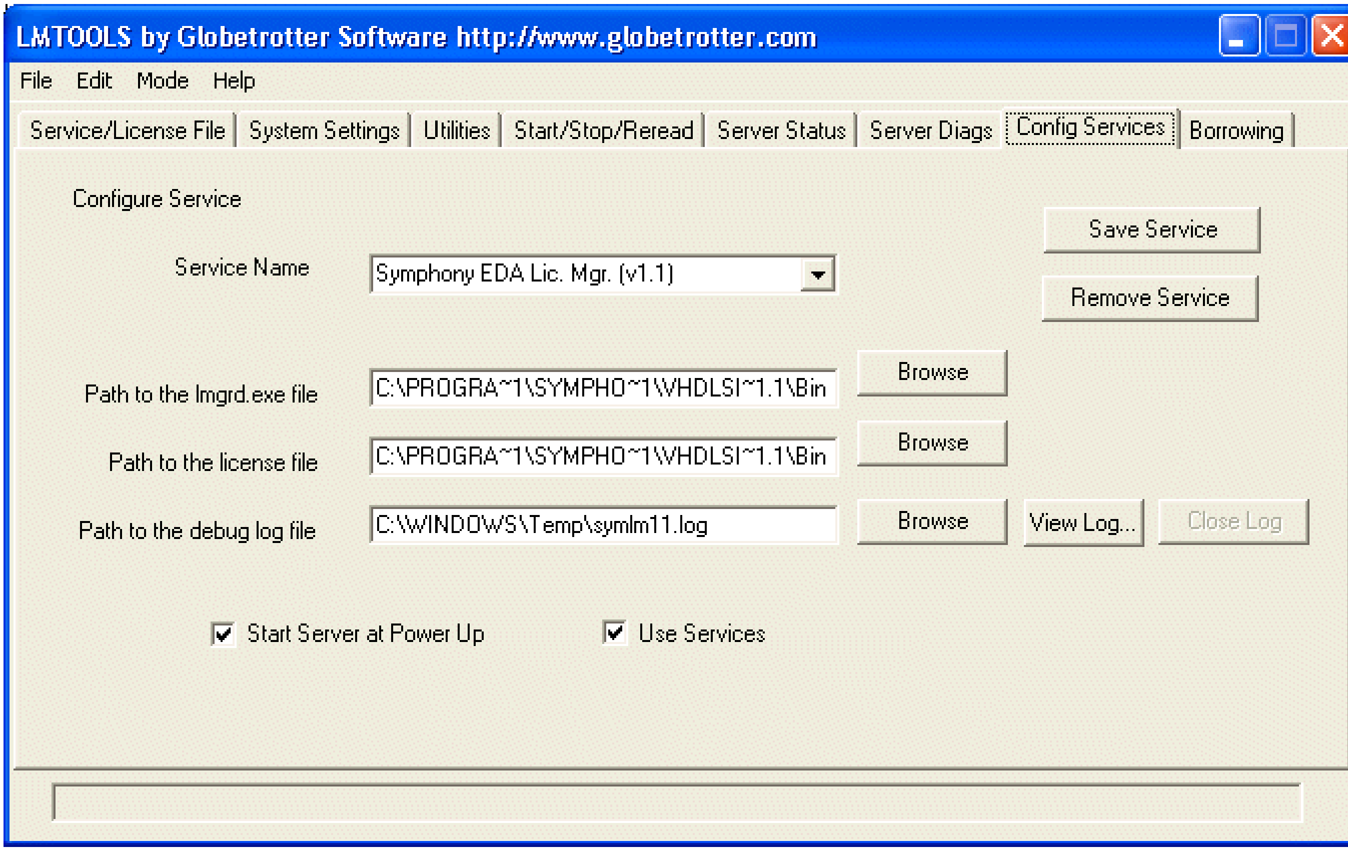
○ Configuration using License File

● Configuration using Services

Symphony EDA Lic. Mgr. (v1.1)

Figure 3-7: Selecting the Symphony EDA License Manager as a service

2. Select the "`Config Services`" tab. The entries in this tab should already be correct based on the last installation. If you have chosen an alternate license file name or a log-file name, you can modify these settings appropriately.



Configure Service

Service Name: Symphony EDA Lic. Mgr. (v1.1)

Path to the lmgrd.exe file: C:\PROGRA~1\SYMPHO~1\VHDLSI~1.1\Bin

Path to the license file: C:\PROGRA~1\SYMPHO~1\VHDLSI~1.1\Bin

Path to the debug log file: C:\WINDOWS\Temp\symlm11.log

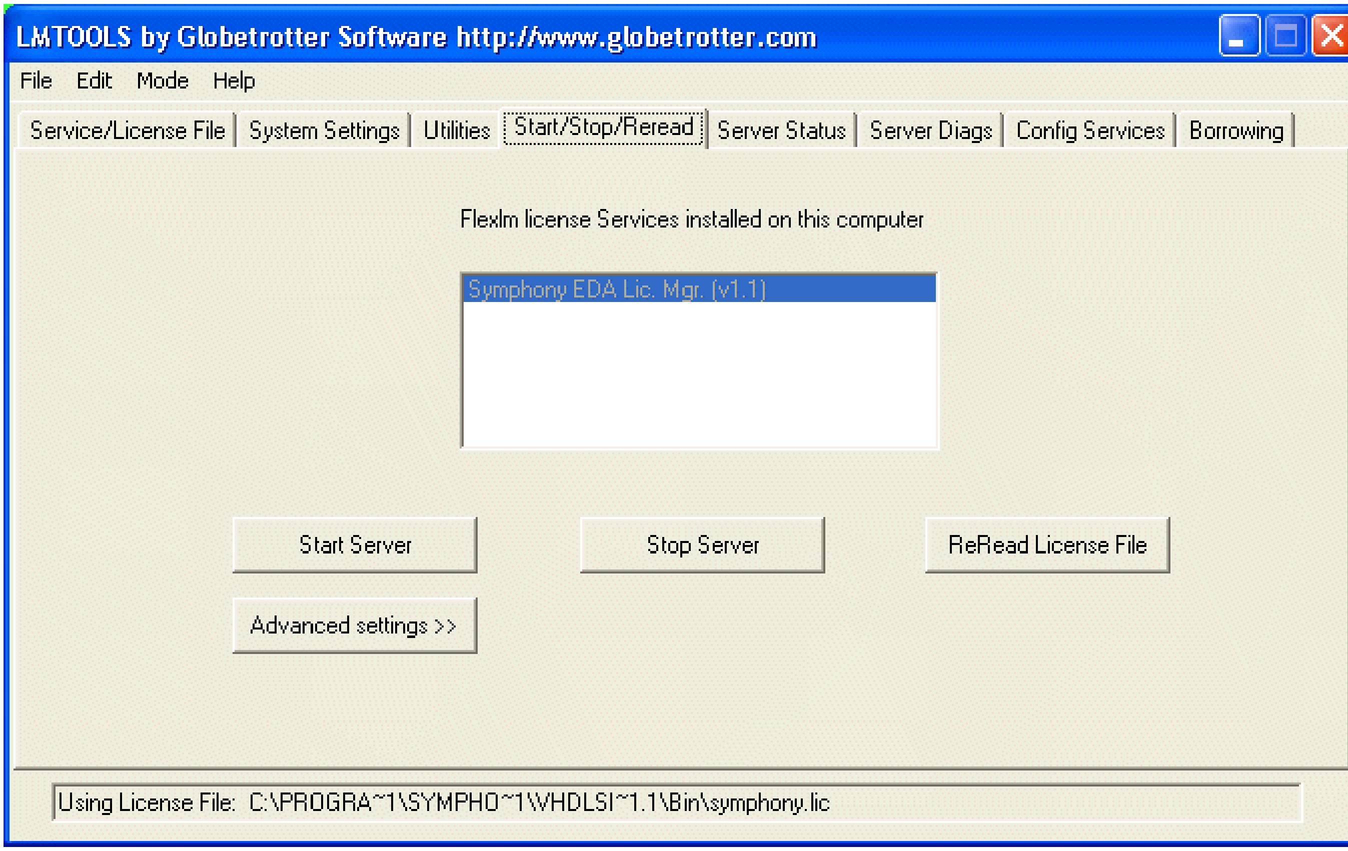
☑ Start Server at Power Up ☑ Use Services

Figure 3-8: Configure the "Symphony EDA License Manager"

3. Make sure that you enable the "`Start Server at Power Up`" checkbox so that the license manager starts after a reboot. On Windows NT/2000, you may wish to also enable "`Use Services`" checkbox to run the license manager **`lmgrd`** as a service.

4. Click on the "`Save Service`" button to save these settings and go to the last step.

5. Although we have configured the license manager, the license manager is not yet running. Click on the “`Start/Stop/Reread`” tab and then click on the “Start Server” button. This simply avoids a reboot of your computer before you can begin using the Simili.



Flexlm license Services installed on this computer

Symphony EDA Lic. Mgr. (v1.1)

Using License File: C:\PROGRA~1\SYMPHO~1\VHDLSI~1.1\Bin\symphony.lic

Figure 3-9: Start the license manager

3.6.2 Unix/Linux

On UNIX/Linux, you can start **`lmgrd`** (the license daemon) manually or start it at boot time. To start **`lmgrd`** manually do the following.

```
cd <install-dir>/bin
./lmgrd -c ./symphony.lic -l /tmp/symlmgrd11.log
```

Alternatively, edit the appropriate boot script, which may be `/etc/rc.boot`, `/etc/rc.local`, `/etc/rc2.d/S`**`xxx`**, `/sbin/rc2.d/S`**`xxxx`**, etc. On **Red Hat systems** the following is recommended in `/etc/rc.d/rc.local` (add it at the end).

```
sh -c “cd <install-dir>/bin ; ./lmgrd -c ./symphony.lic -l
/tmp/symlmgrd11.log”
```

The above examples assume that the license file is placed in the installation directory (the default location).

Please see <install-dir>/doc/flexuser/TOC.htm for more details.

3.7 Combining FLEX*lm* license from other vendors

There are a myriad of possibilities for configuring the FLEX*lm* tools. Please refer to the FLEX*lm* documentation at http://www.globetrotter.com or <install-dir>/doc/flexuser/TOC.htm for a complete discussion on this topic. If you are using license files from multiple venbdors, you can use one of the following two methods described below.

1. If you intend to run multiple license managers, then each license manager must be assigned a separate TCP/IP port number. By default the license manager (lmgrd) uses port 27000 if no port number is specified in the license file. The port number can be specified at the end of the SERVER line. The format of the SERVER line in the license file is

   ```
   SERVER host-name host-id [port]
   ```

   It is usually safe to pick a port number between 27001 and 27009 (any other number greater than 1024 may do as well so long as no-other program is using it). An example SERVER line may look as follows

   ```
   SERVER lulu 1300ab43 27009
   ```

   Once you have added unique port numbers to all the license files, you can start individual license managers for each of the vendors. For Symphony EDA, we recommend the procedure documented in sections 3.6.1 (Windows) and 3.6.2 (Unix).

2. Another way to manager multiple licenses from multiple vendors is to run one license manager for all the vendors. When chosing this option, you may run into compatibility problems because of the various versions of license FLEXlm used by the various vendors. You are usually safe to pick the latest license manager. To run this method, pelase follow these steps

   a. Copy all the license files intended for a given server into a common directory

   b. Copy all the VENDOR daemons to this same directory as well. The VENDOR daemon for Symphony EDA can be found in `<install-dir>/bin/symlm.exe`. Alternatively, you can edit the license file and add the full path name to the vendor daemon in the VENDOR line in the license file (sometimes this line is also called the DAEMON line for older license files). A sample VENDOR line may look as follows

      ```
      VENDOR symlm C:\SymphonyEDA\bin\symlm.exe
      ```

   c. Start the license manager (using instructions specified in sections 3.6.1 (Windows) and 3.6.2 (Unix) but use the path-name to this common directory is the name of the license file.

Chapter 4

4 VHDL Simili Environment

VHDL Simili is a collection of tools that include a powerful graphical interface. While each tool within the Simili toolset has its own name, Simili represents the entire collection of tools. For instance, Sonata is the name of the graphical interface to the simulation environment.

4.1 Introduction

4.1.1 Windows Users

When you install VHDL Simili, the setup program would have created the following items in the `"Start->Programs->Symphony EDA"` program group (unless you have instructed Setup to name it something else).

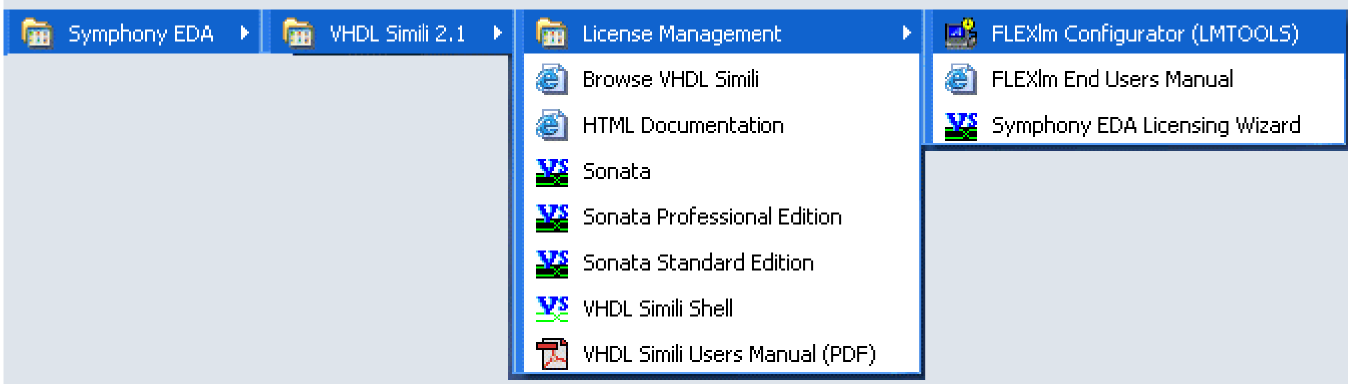


Figure 4-1: Start menu program group

Use the `"Browse VHDL Simili"` item to become familiar with the entire VHDL Simili environment. The `"Documentation"` link opens top-level documentation page (in HTML). The `"Sonata"` item launches the Symphony EDA's Integrated Development Environment, which is a graphical user interface to VHDL Simili. The `"Sonata"` item will attempt to check out one of `"sonatapro"`, `"sonata"` or `"sonatafree"` licenses. Alternatively, you can use the `"Sonata Professional Edition"` or the `"Sonata Standard Edition"` shortcuts to check out the edition you have purchased removing any guesswork (see option –uselicense).

The `"VHDL Simili Shell"` item creates an environment for command-line use of the Simili tools. This is an alternative to Sonata or may be used by power users who prefer to automate their tasks using the command-line tools. This item will launch a command line environment from where you can compile and simulate your VHDL models (it essentially sets up your PATH in a reliable way). The Shell window is actually a DOS window. As such, you can change its properties (font, window size, etc.) and these settings will be remembered the next time you launch the Shell. You may even copy this item to your desktop and modify it there so that future installations do not override your settings.

The "VHDL Simili Shell" always starts in the installation's Bin subdirectory. You can append a "cd" command to the <install-dir>/bin/Simili.bat file to take you to the directory of your preference. You can also chose to not use the SHELL at all if you prefer to edit your PATH settings (either using the autoexec.bat or autoexec.nt or your Environment Settings on Windows NT or some other way). If you modify the Simili.bat file, these changes may be lost if you reinstall VHDL Simili.

If you use some other shell (such as bash, sh, etc.) all you need to do is to ensure that you have set your PATH to emulate the contents of the Simili.bat file.

4.1.2 Unix/Linux Users

There is nothing special that occurs for Unix users. Following the installation directions, you must have the VHDL Simili tools in your PATH. To launch the graphical user interface, simply use the command "**sonata**" which will attempt to check out one of "sonatapro", "sonata" or "sonatafree" licenses If you have purchased the "Professional Edition", then use the command "**sonatapro**" or use the –uselicense option on **sonata**.

4.2 VHDL Simili Directory Structure

The following is a table of important files and directories that are available after VHDL Simili installation under the root of the installation directory:

./bin	Contains the Simili.bat, the default symphony.ini file and a couple of miscellaneous files. It also contains programs related to license management as well as the license file itself (symphony.lic). Symphony EDA does not recommend putting this directory in your PATH (use <install-dir>/tcl/bin instead).
./lib	Contains all the VHDL libraries. There is a sub-directory for each of the vendor libaries. The script compile.bat (Windows) and compile.sh (Unix) can be used to compile all vendor libraries installed on your system that Simili knows about. There is documentation available in <install-dir>/lib/index.htm
./examples	Contains examples/tutorials for the command line tools (<install-dir>/examples/index.htm).
./doc	Contains documentation for FLEXlm and VHDL Simili itself (this documentation)

`./tcl`	This is where the Tcl/Tk is installed. This is a private installation of Tcl/Tk intended only for VHDL Simili's use. The VHDL Simili program files (executables and shared object files) are also stored under this directory.
`./tcl/bin`	This is where the VHDL Simili and FLEX*lm* executables reside. You can place this directory in your PATH environment for easy access to the executables from the command line.
`./tcl/lib`	This is where all the TCL packages that are required by Simili are stored
`./tcl/lib/symphony`	This directory contains the Tcl/Tk code for Sonata (Simili's graphical interface).
`./index.htm`	This is an HTML file that allows you to browse the installation directory.

Most sub-directories under the installation directory contain a file index.htm, which can give you more information about the contents of a given directory (and its sub-directories).

4.3 Basic VHDL Concepts

There are a handful of VHDL concepts that you need to know before VHDL Simili makes sense to you. A typical VHDL source file contains zero or more design units. Examples of design units are entity, architecture, package, etc. When a VHDL source file is compiled, the results of a successful compilation are placed in a library. In other words, the design units contained within a VHDL source file are compiled into a library.

A design unit that has been compiled into one library may reference other designs units in other libraries (due to use clauses, library statements, etc.).

In VHDL, the current working library is always called `"work"`. When using a VHDL compiler or simulator, there is always a concept of a current working library. If no particular library is specified as the current working library, the current working library is assumed to be `"work"`. You can associate the "work" library with any other library. When using Sonata, work is always associated (mapped) with an actual library that is registered with the project.

There are two kinds of design units -- primary design units and secondary design units. The design units of type entity, package and configuration are primary design units. Design units or type architecture and "package body" are secondary design units. Secondary design units are always associated with a particular primary design unit. Secondary units typically contain the implementation of their respective primary units.

A few simple rules to remember

- All primary units in a given library must have unique names. **Note:** VHDL language actually allows an entity to have the same name, as one of its configurations but VHDL Simili requires that all primary units have unique names in a given library.

- All secondary units for a given primary unit must also be named uniquely.

- A primary design unit and all of its secondary design units must both reside in the same library.

More on libraries: Library names have a logical name and a physical name. In the VHDL source file, all references to library names are logical library names. Each logical library name is associated with some physical library name (a path-name to some physical storage mechanism). This association is made through controlling the environment of the tools via the use of the `symphony.ini` or your Sonata project file described in the next section.

The STD library: There is also a library with a special name `"std"`. This library and its contents (the packages `standard` and `textio`) are built into the tools and cannot be controlled. This also means that you cannot have a user-defined library called `"std"`.

The `symphony.ini` file controls access to libraries. The following section discusses this in greater detail.

4.4 Controlling your environment

Libraries are about the only things that cause external dependencies on how a given VHDL source file is analyzed. Some vendors use some of the same common names (such as DFF, AND2, etc.) for the design units within their libraries. If you want VendorA's components, you definitely do not want to unintentionally pick up VendorB's components. You as a user may also have multiple libraries with some design units in those libraries using the same names. Within the VHDL source file, a user typically uses library names and use clauses to make things more specific.

To avoid any confusion and to allow flexibility, the mapping of libraries occurs through a file named `"symphony.ini"`. All VHDL Simili tools use this file. Unless an explicit file name is given on the command line, the Simili tools look for this file in the current-working-directory and if they do not find one there, they use the default INI file located in the installation directory's “`bin`” sub-directory. **Note:** When using the Simili tools from Sonata, the project file serves as the `symphony.ini` file. This file can be used to map library logical names to physical library names.

You do not need a custom `symphony.ini` file if you meet the following requirements:

- You are using the pre-compiled libraries shipped with VHDL Simili or you are using vendor libraries that Simili supports under the installation directory.

- Any other libraries that you have created are in the current directory.

- You have used the default physical-directory naming convention (the physical-directory name is the logical-name with a .sym extension).

- None of your logical library names are extended VHDL identifiers.

The tools will automatically search for a library by looking first in the current working directory and then in the set of pre-compiled (or supported) libraries shipped with the tool. Any library mappings that appear in an INI file override the default search mechanism. ***Unless you have special needs that cannot be served by Sonata, you can skip the rest of this section***.

You will need the INI file if you have libraries that are not in the current directory or if the base-name of the physical-directory containing the library is not the same as the library name. You may also associate multiple logical library names to the same physical library.

Depending on the vendor you use, the very first thing you will need is your own `symphony.ini` file (you may use the default one if you use nothing but the IEEE library).

The following are the contents of the default `symphony.ini` file.

```
#
# Setup the libraries
# You can use environment variables of the form $var or %var%
# and on Windows slashes and back-slashes are treated the same
#
# The SYMPHONYEDA environment variable is automatically
# created to reflect the installation directory.
#
[libraries]
ieee = $SYMPHONYEDA/lib/ieee/ieee.sym
```

The `'#'` marks the beginning of a comment and the entire line after the `'#'` character is treated as a comment. Empty lines are ignored. The keyword `"[libraries]"` begins the library mapping section. It should be followed by a zero or more lines (non-comment or empty lines) where each line represents a library mapping. The format of a library-mapping file is as follows

```
logical-library-name = physical-library-directory
```

The `"logical-library-name"` is the name that is used to refer to this library in your VHDL source. The physical directory name is the name of the directory where the library resides. Note that typically, the VHDL Simili library directories have a suffix of `".sym"`. Once such a line is added to the `symphony.ini` file, the directory mentioned in the mapping is automatically created when needed.

Example#1: If you want to add your own library called `"mylib"` assuming that the library exists one directory above the current directory, then you would add the following line to the `symphony.ini` file.

```
        mylib = ..\mylib.sym
```

Since the '/' character is treated the same as the '\' character, the following line would have been the equivalent. On Unix, it is important to only use '/'.

```
        mylib = ../mylib.sym
```

Example#2: Sometimes, it may become necessary to create a mapping of a library where the name of the logical library is not the same as the name that was used when the library was created (using the default conventions). The following line

```
        mylib = ./somelib.sym
```

associates the logical-library-name "mylib" with the physical library directory "./somelib.sym".

☑ In the future, additional sections may be added to the symphony.ini file format to allow for tool customization. The command line tools allow the use of the -ini option that allows one specify the path to an INI file explicitly. Since Sonata project files use the same format as the INI file, any Sonata project file can be used to specify an INI file using the -ini option.

☑ In the INI file (or in a Sonata project file), you can associate multiple logical names with the same physical library name.

4.5 What is in a .sym directory?

VHDL Simili tools create VHDL Libraries with a “.sym” extension in the directory name. The contents of this directory are exclusively used by the Simili toolset. To delete a VHDL Library, simply delete its directory (and its contents). This is the easiest way to clean up. Note that you should delete libraries that you can easily re-build - do not delete pre-packaged libraries or libraries supplied by others unless you can re-install them. These directories do not need to be backed up if you can easily re-create them from their source VHDL.

4.6 VHDL Simili tools

Sonata is a powerful environment to help you manage large VHDL projects as well as serve as an interface to the compiler and the simulator. The compiler and the simulator themselves are actually command line tools and if you using Simili from the command line, there are only two commands you have to deal with - the compiler (**vhdlp**) and the simulator (**vhdle**).

Chapter

5

5 Sonata

Sonata is an application that provides an Integrated Development Environment (IDE) for designing, verifying and managing HDL projects. If you do not have a license for running Sonata, the Licensing Wizard is automatically invoked when Sonata is launched. Sonata is a graphical user interface (GUI) to the Simili simulation environment. It aids in project management tasks, and serves as an interface to the compiler and simulator.

This chapter deals with the general uses of the Sonata framework. The actual process of simulation and interacting with waveforms is discussed in a separate chapter.

Windows users: Sonata can be invoked using "`Start->Programs->Symphony EDA->VHDL Simili X.Y->Sonata`" (where `X.Y` is the version number). You can also do a "Start->Run" and enter **`sonata`** as the command to run and this will run the last installed version of Sonata.

Unix users: Sonata can be invoked on the command-line using the command "**`sonata`**". Since this is a graphical tool, you must be running under X-Windows.

Sonata contains many windows, panes and tabs. Most of these areas support a context-sensitive menu that can be useful. A context-sensitive menu is invoked by pressing on the right mouse button (Button-3 on Unix). Sonata also supports the wheel mouse for scrolling.

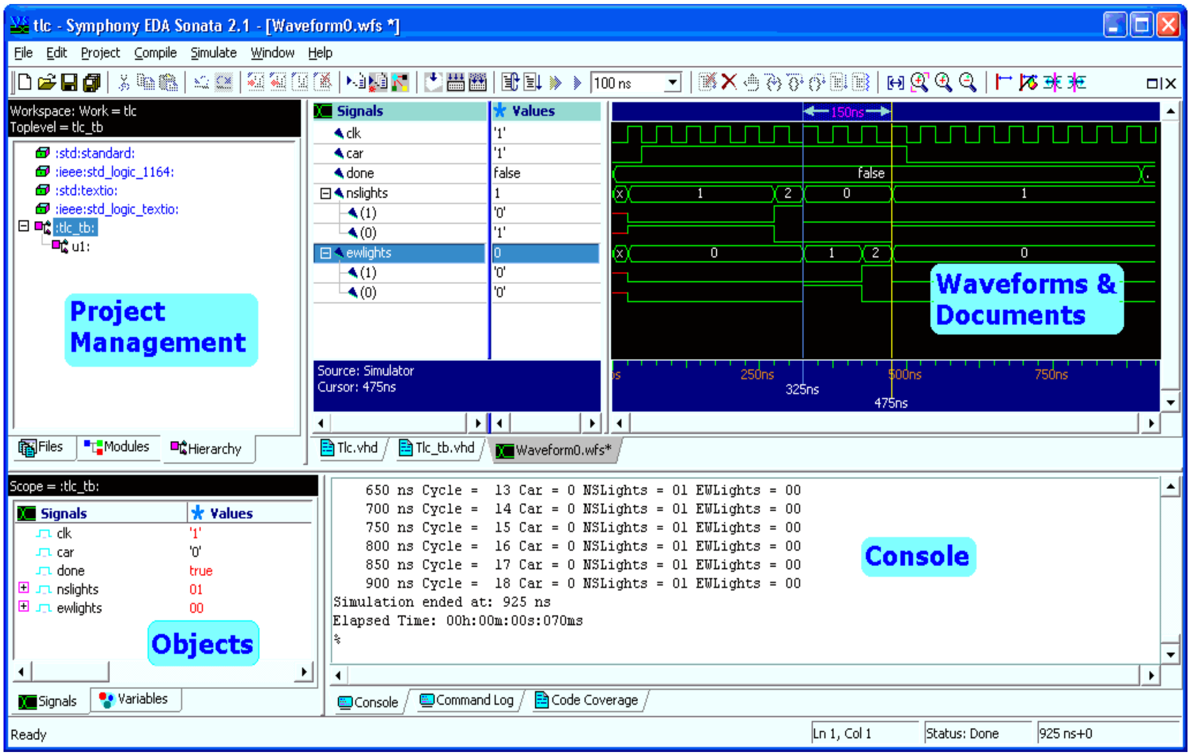
Many areas of Sonata uses forward slashes as a directory separators instead of backslashes on Windows. This is intentional. However when using the Windows Common Controls (File Open, Save As, Directory Chooser), use of forward slashes does not work.

Sonata project files are stored in files with the extension "`.sws`". It is highly recommended that you backup the project files along with your source code.

Sonata is a high capacity, high performance graphical framework capable of dealing with very large designs (hundreds of files) and very large waveform databases. Sonata user interface is liable to change as a result of user input and we will try our best to keep pace with accurate documentation. The best way to learn Sonata is to read this chapter and go through the tutorial. This chapter explains the basic framework, important concepts related to workspaces and libraries, various options and settings, etc. In most cases, Sonata's dialog boxes are self-documenting.

5.1 Sonata - Overview

This section explores the various parts of the Sonata framework. Once you are familiar with the general layout of Sonata and the purpose/functionality of the various parts, you are ready to learn about Sonata's project management concepts. The following is a snapshot of Sonata:



Signals	Values
clk	'1'
car	'1'
done	false
nslights	1
(1)	'0'
(0)	'1'
ewlights	0
(1)	'0'
(0)	'0'

Source: Simulator
Cursor: 475ns

Signals	Values
clk	'1'
car	'0'
done	true
nslights	01
ewlights	00

```
    650 ns Cycle =  13 Car = 0 NSLights = 01 EWLights = 00
    700 ns Cycle =  14 Car = 0 NSLights = 01 EWLights = 00
    750 ns Cycle =  15 Car = 0 NSLights = 01 EWLights = 00
    800 ns Cycle =  16 Car = 0 NSLights = 01 EWLights = 00
    850 ns Cycle =  17 Car = 0 NSLights = 01 EWLights = 00
    900 ns Cycle =  18 Car = 0 NSLights = 01 EWLights = 00
Simulation ended at: 925 ns
Elapsed Time: 00h:00m:00s:070ms
%
```

Ready Ln 1, Col 1 Status: Done 925 ns+0

Figure 5-1: Snapshot of the Sonata framework showing the four major panes.

Sonata is divided into four major re-sizable panes (areas). It is helpful to be familiar with the functions of these four panes.

5.1.1 Project management pane

This window (top-left) displays the status of the current project being edited in Sonata. This pane contains the following tabs arranged in a tabbed window:

- **Files tab**: This tab displays the project using the "File" view. In the "File" view, you can see the various libraries registered with the project and for each library, the source files associated with it are also shown. Nested inside each file, you can see the various design units (if any) contained in the file. These design units are displayed based on a quick scan of the file and this information is updated each time the file is saved.

☑ You can use the drag and drop mechanism to change the order in which files and libraries are displayed. However, note that you cannot re-order files if auto-ordering is enabled; if you try to re-order when auto-ordering is enabled, Sonata will re-order the files only after promting the user to disable the auto-ordering.

☑ You can double click your left mouse button on a file or a design unit to open the file in a text editor.

- **Modules tab**: This tab displays the project in its compiled form. Whereas the "Files" tab displays project information from a source code point of view, this tab displays the project information based on what was actually compiled into each library.

- **Hierarchy tab**: This tab is only used when a simulation session is currently active. It displays the structure (hierarchy) of the current design unit being simulated.

☑ Double-clicking your left mouse button on a specific section of the hierarchy will display the objects contained in that section of the hierarchy in the "Signal/Variable" display area.

5.1.2 Document pane

This window (top-right) is responsible for managing all open documents. A document is either a text file or a waveform file. All documents are arranged in a tabbed notebook window with each tab hosts a single document of a given type. Text files are displayed using a text editor that is capable of performing syntax highlighting. Waveforms are displayed using a waveform viewer that is capable of displaying millions of signal transitions.

5.1.3 Console pane

This window (bottom-right) displays all the messages from various tools (including the IDE itself). Symphony EDA envisions a bright future for the “Console” pane. There are many things possible using the Tcl/Tk interface but this documentation intentionally excludes such documentation as the API is still evolving and may cause compatibility problems for users relying heavily on the Tcl/Tk commands. The console area contains the following tabs:

- **Console tab**: The console is actually a customized TCL (Tool Control Language) command window. One can enter various commands on the command line and the console will evaluate these commands as if they are TCL commands. If the command entered is not a valid TCL command, then the console will look for an external command with the same name and execute that if possible. The console also has the following features.

   1. Maintain a history of the commands being executed and also log the commands to the "Command Log". Use Up/Down arrow keys to navigate through the command history. Pressing the F9 function key displays the command history list.

   2. Display all messages resulting from the commands that are executed.

3. Color-code various messages (like Errors, Warnings, etc.), whenever possible double-clicking on these special messages will attempt to open the file (in the Document pane) that was the source of the message.

On Windows, the console can be very useful if you have installed command line tools such as MKS or Cygnus tools which provide many helpful Unix-like utilities. On Unix, this is a natural thing to do. You can execute any program that does not require input from "stdin".

It is important to remember that when entering commands in the console window that the command is actually interpreted by a Tcl interpreter. As such, a backslash, single and double quotes, a semi-colon, {} and [] are special characters. Where possible, use a forward-slash instead of a backslash within pathnames in Windows (most tools understand this convention). Alternatively, you can use two backslashes to represent a single backslash. Single and double quotes are helpful to combine words with embedded spaces into a single argument for a command.

- **Command Log tab**: This window logs all commands entered in the console tab. While the "Console tab" shows all input and output, the "Command Log tab" only shows the commands entered. This window can be helpful in copying commands into a script file. You can use the TCL command source to execute commands stored in that file.

5.1.4 Objects pane

This window contains valid data only when a simulation session is currently active. The contents of this window are synchronized to the current scope selected in the "Hierarchy tab". This window contains the following tabs:

- **Signals tab**: This tab displays all the signals and their current values in the current scope. You can drag and drop the selected signals from this tab to the waveform window that is associated with an active simulation.

- **Variables tab**: This tab is similar to the "Signals tab" except it displays all non-signal objects that have values -- these include generics, constants, variables, etc.

Each time simulation time advances, the values displayed in the objects pane may change color from black to red or vice-versa. A value marked in red indicates a change in value from the previous display and a change from red to black indicates no-change.

5.2 Sonata - Project Management

Sonata offers advanced project management features designed to handle complex projects consisting of multiple libraries where each library is built with zero or more HDL files. However, it is also capable of handling small/simple projects with very little user setup. Note that the top-left part of the GUI is used to perform project management tasks.

Sonata project management tasks can be broken down into the following broad categories:

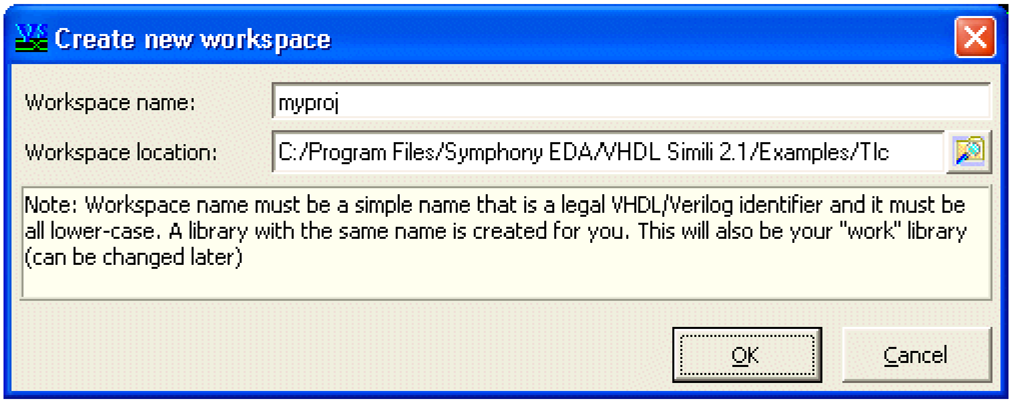
- The workspace
- Working with libraries
- Compilation
- Simulation

5.2.1 The workspace

Sonata is an Integrated Development Environment (IDE) for managing HDL projects. Each project is represented by a workspace (also called a project). Before Sonata can be used for anything, a workspace must be created.

Workspace functions such as opening, saving, closing and creating workspaces can be accessed using the "**File**" menu.

A workspace is created using the "File->New Workspace..." menu item. This menu item will display the following dialog box.



Workspace name: myproj

Workspace location: C:/Program Files/Symphony EDA/VHDL Simili 2.1/Examples/Tlc

Note: Workspace name must be a simple name that is a legal VHDL/Verilog identifier and it must be all lower-case. A library with the same name is created for you. This will also be your "work" library (can be changed later)

Figure 5-2 : Create New Workspace Dialog.

Using the above dialog box, you can create a workspace in a directory of your choice. Note that all files and libraries will be maintained in this workspace relative to this directory regardless of their actual location on disk.

Creating a workspace will also create a library with the same name. This library becomes the current working library (which can be changed later). When a workspace is first created, it is initialized with the " `symphony.ini`" file. If this file exists in the current directory, it is used or else the " `symphony.ini`" file located in the installation-bin directory is used.

Once a workspace is created, all other tasks become possible.

All compilation and simulation will occur relative to the directory specified for the workspace. This is important to remember when accessing files using relative pathnames from within your VHDL source files.

5.2.2 Working with libraries

The function of libraries can be summarized using the following key points:

- Every Sonata project is a workspace that manages one or more libraries.
- Each library is a library in the VHDL sense.
- A library contains a set of design units. Design units are entities, architectures, configurations, packages and package bodies.
- There is always a concept of the current working library. The current working library is displayed in the top left area of the project management window. It is also shown in the "Files" and "Modules" tab in red.
- A library can be marked read-only. A read-only library simply does not allow compilation of HDL files to create/modify design units. A read-only library is displayed in the "Files" and "Modules" tab with a gray color (unless it is also the work library).
- All actions related to project management typically apply to the current working library.
- Each library can be associated with a set of source files (VHDL files). Files can be added using the context sensitive menu on the library (right click on the library) or by using the "Project" menu.
- You can detach (remove) a file associated with a library by selecting it in the “Files” tab and pressing the Delete key or using your right mouse button and selecting the appropriate menu item in the context sensitive menu. Note that detaching a file does not physically remove the file from the file system.
- The "**Project**" menu contains functions to manage libraries. Using this menu, you can

  1. Add/Remove files to the current library

  2. Force a re-ordering of the files

3. Change the current working library

4. Delete the contents of the current working library (deletes compiled objects, not source files)

5. Toggle (change) the read-only status of the current library

6. Attach/Detach an existing library to the current workspace

7. Create a new library and attach it to the current workspace

- Most project management functions are also available using the context sensitive menu in the "Files" tab.

- As files are added to the project or whenever files associated with a given library are modified, the order of files is automatically adjusted. Sonata does a quick scan of the files and tries to determine their relationship to each other. Although the heuristics used by Sonata are very sophisticated, it is still possible that every once in a while (with VHDL files that use complex configuration statements), it may suggest a wrong ordering.

☑ You may want to consider disabling automatic-file-ordering for libraries that are associated with large files (several megabytes). You can do this using the "Project->Settings..." menu item on a library-by-library basis.

☑ You do not have to associate files with libraries that you are not modifying. For instance, there is not reason to add any files to the "ieee" library because you are unlikely to compile any files into this library. We merely want to point to it to gain quick access to modules defined within a given library.

- Change the settings for the workspace.

- You can drag and drop files/libraries to change their order in the workspace.

- The "Modules" tab displays the modules actually compiled into a library whereas the "Files" tab will display modules that exist in each file. This information is updated with each compile and with each file save and upon loading a workspace.

5.2.3 Compilation

The “Compile" menu provides access to compilation tasks. Using this menu, you can do the following tasks

- Compile the current file being edited: You can compile the current file (if associated with the current working library).

- Compile selected files: Files selected in the project management window ("Files" tab) are compiled.

- Build: This will compile all files that need to be compiled in the current working library. It looks at the timestamps of the files and considering dependencies between the various design-units in that library; it will attempt to compile the minimum set of files. Please ensure that the clock on your machine is correct and with a networked environment, also ensure that your PC clock is properly synchronized with other computers in your network.

- Compile All: This will force the compilation of all files regardless of timestamps. This is one way of bypassing the "Build" (smart compilation).

A VHDL file is compiled using **`vhdlp`**. All messages from the compiler go to the console window and where necessary, act as hypertext links into the source files that caused the messages. Use your mouse (double-click left mouse button) to track errors to your source file(s). You can also use the "Next error..." or "Previous error.." menu items in the "Compile" menu.

Each time a compilation session begins; any old output from previous commands in the console window will turn gray (while all new messages take on natural colors).

When a large number of files need to be compiled, Sonata will automatically split the compilation into a multiple invocations of **`vhdlp`** to avoid exceeding command-line limits.

Compilation settings are part of the workspace settings.

5.2.4 Simulation

The "Simulation" menu provides access to simulation tasks. The topic of simulation is covered in a separate chapter.

When a simulation is active, the "Hierarchy" tab in the "Project Management" window is activated. The companion "Objects" window (lower left area of the application frame) is activated. Note that these two windows will be blanked out once the connection to the simulator (**`vhdle`**) is severed.

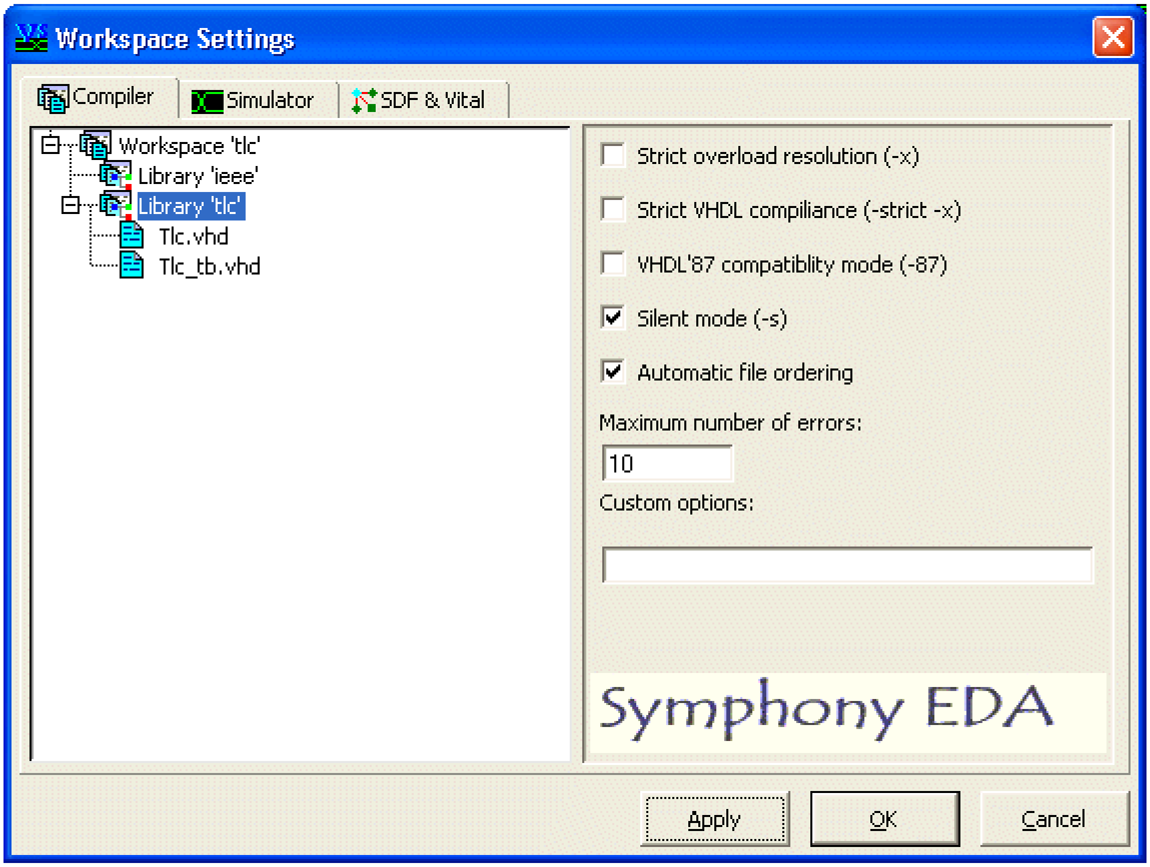
5.3 Sonata - Workspace settings

Workspace settings affect the compiler and simulator options for the current workspace. Changing workspace settings only affect the current workspace. There settings are not considered as personal preferences (these are maintained on a per-user basis and are the same across all workspaces for a given user). There are three broad categories of settings. They are compilation settings, simulation settings and SDF annotation settings.

Workspace settings can be modified using `"Project->Settings..."` or `"Compile->Settings..."` menu items. The following sections discuss the three types of settings.

5.3.1 Compiler settings

The "Compiler" settings tab displays the workspace contents in a tree form (on the left side) and the options of a given object in workspace are shown on the right side.



☐ Strict overload resolution (-x)

☐ Strict VHDL compiliance (-strict -x)

☐ VHDL'87 compatiblity mode (-87)

☑ Silent mode (-s)

☑ Automatic file ordering

Maximum number of errors: 10

Custom options:

Figure 5-3 : Compiler Settings Tab

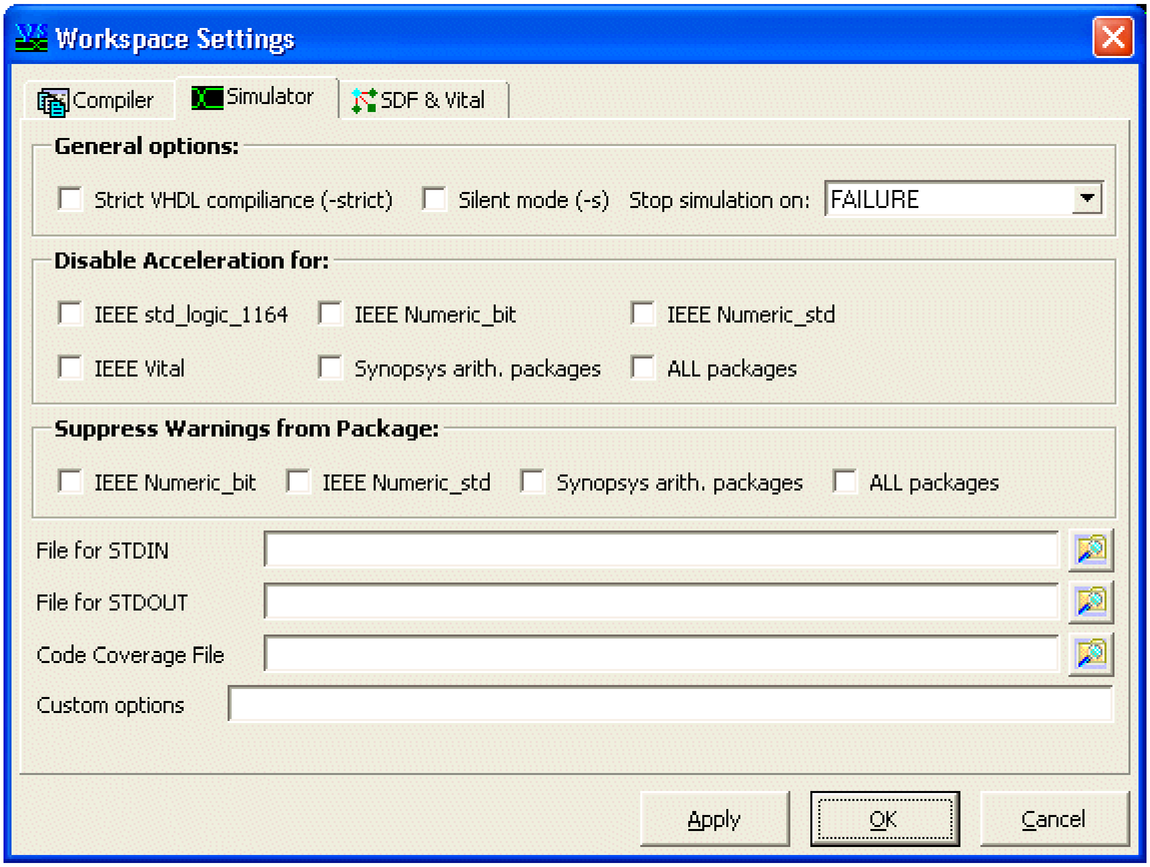
Options can be set in a hierarchical manner where you can have a particular set options specified for a given library but individual files can have different (superseding) settings. Also, each library can have a different set of options. ***In effect, each library and file can have its own settings***. The options shown on the right side of the panel represents the current settings for the selection of library/file in the tree control shown on the left.

While the detailed meaning of each of these options is documented in the compiler chapter the following is a brief summary of options shown above.

Automatic file ordering and the maximum-number of errors are options that are valid only at a library level. Automatic file ordering can be controlled on a library-by-library basis. Whenever a file registered with a given library changes, the modified files are scanned to determine the proper order of compilation. Note that this is done without actually compiling the VHDL files and as such can rarely be wrong (this can happen due to complicated configurations that are not easy to determine by trivial examination of the VHDL source files).

5.3.2 Simulator settings

The "Simulator" settings tab displays the options and controls for the simulation environment.



General options:

□ Strict VHDL compiliance (-strict) □ Silent mode (-s) Stop simulation on: FAILURE

Disable Acceleration for:

□ IEEE std_logic_1164 □ IEEE Numeric_bit □ IEEE Numeric_std

□ IEEE Vital □ Synopsys arith. packages □ ALL packages

Suppress Warnings from Package:

□ IEEE Numeric_bit □ IEEE Numeric_std □ Synopsys arith. packages □ ALL packages

File for STDIN

File for STDOUT

Code Coverage File

Custom options

Figure 5-4 : Simulator Settings

While all of these options are documented in the simulator reference chapter, the following is a brief summary of these options.

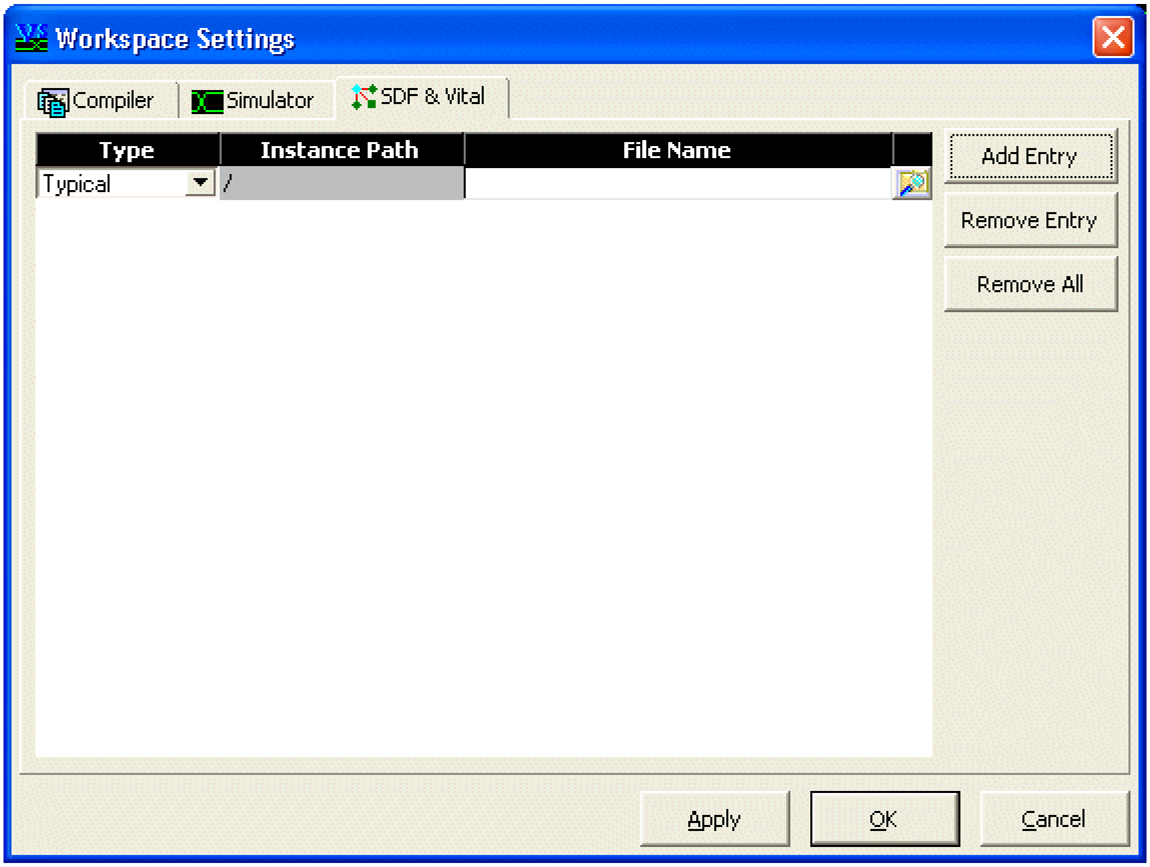
- The "General options" section as the name implies are general options controlling the simulator behavior
- The "Disable Acceleration for" section allows disabling acceleration for various built-in packages. Note that disabling acceleration will slow down your simulation speed by an order of magnitude.
- The "Suppress Warnings from Package" section allows you to suppress warning message that are built-in to these built-in packages. If you have disabled acceleration for a given package, you cannot suppress messages that result from that package. It is rather dangerous to suppress these messages because real problems may go un-noticed. The usual reason for most of these messages is due to non-initialization of signals/variables involved in arithmetic operations at time zero.

- You can specify a file for "STDIN" and "STDOUT". If your model expects input on the TEXTIO.INPUT FILE handle, you must specify a file name for it. If no file for STDOUT is specified, all output written to the TEXTIO.OUTPUT FILE handle appears in the console window.

- The "Custom options" text entry box can be used to specify arbitrary options. The user is responsible for ensuring the correctness of these options. A good use of this is to provide "generic overrides".

5.3.3 SDF & Vital settings

Using this tab, one can specify SDF back-annotation parameters



Type	Instance Path	File Name
Typical	/	

Figure 5-5: SDF/Vital Settings

Use the buttons on the right hand side to add/remove SDF entries

The SDF entries are shown in a spreadsheet form where the first column specifies the operating conditions (min, max, typ), the middle column is used to specify the instance-path and the right column specifies the path-name of the SDF file on disk. Note that these path-names are relative to the location of the project file (unless full-path names are used)

5.4 Sonata - User preferences

The user can control many of the visual and behavioral aspects of Sonata. Visual settings and certain kinds of behavior are in reality, user preferences and as such, are maintained on a per-user basis. On startup, Sonata will display the location of the preference file. Although the preference file is a text file, we do not recommend editing this file manually (unless you know what you are doing). Instead, use "`Edit->Preferences...`" menu item to edit preferences.

On Windows, most applications use the Windows registry for this purpose. With Sonata, all preferences and settings (window positions, recent documents, etc.) are recorded in the preferences file. Sonata (and other tools in the Simili toolset) uses the registry very sparingly if at all. To create the preferences file, Sonata considers the following alternatives:

1. Unix and Windows: If an environment variable HOME exists, the value of this environment variable is used to as the base name. Even on Windows it is recommend to use the HOME environment variable.

2. Unix: Try the users home directory (~)

3. Unix: Try using `/tmp`

4. Windows: Try using "`%APPDATA%`"

5. Windows: Try using "`%USERPROFILE%`"

6. Windows: Try using "`%HOMEDRIVE%%HOMEPATH%`"

7. Windows: Try using "`%WINDIR%\Profiles\<USERNAME>\Application Data`"

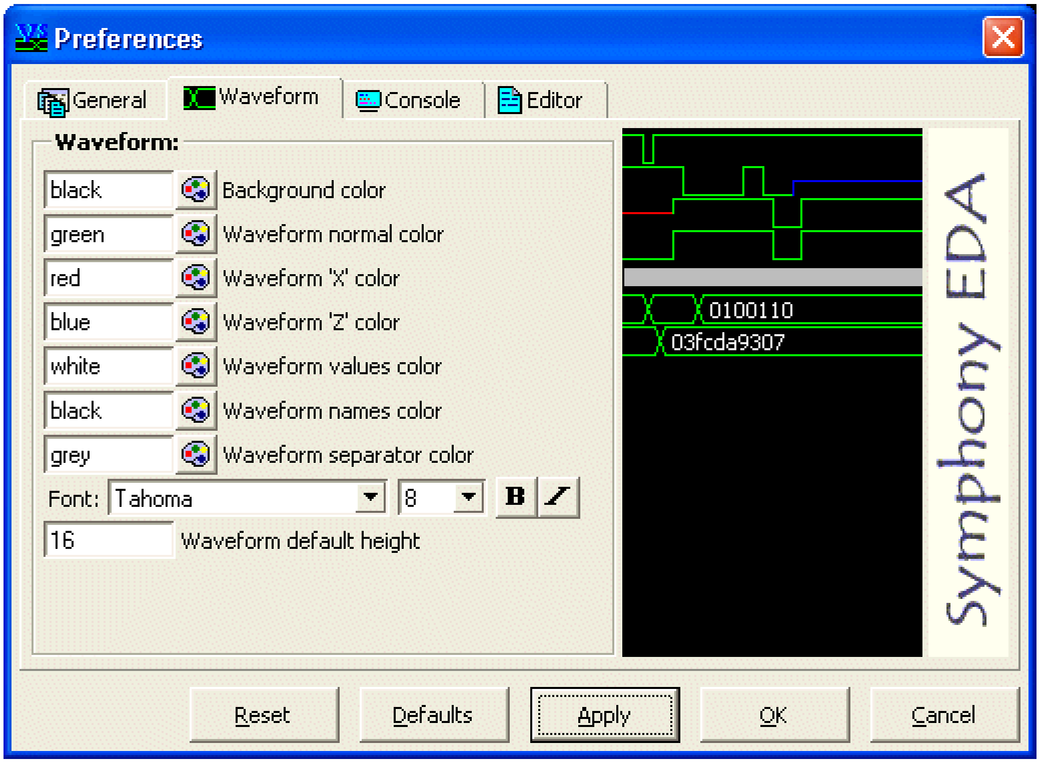
8. Windows: Try using "`%WINDIR%\Profiles\All Users\Application Data`"

9. Windows: Try using "`%WINDIR%\Profiles`"

10. Windows: Try using environment variables `TMP` or `TEMP`

It will search all the above directories for a sub-directory called SymphonyEDA (.symphonyeda on Unix) for a writable preferences file and if one is found, it will use that. If none is found, it will create this sub-directory (and the preferences file) in the frist writable directory from the list above. The preferences file is named "prefs.tcl". This subdirectory is also used to store other information (bookmarks, list of open text files, current workspace, etc.).

The following is a screen-shot of the preferences dialog



Waveform:

black Background color

green Waveform normal color

red Waveform 'X' color

blue Waveform 'Z' color

white Waveform values color

black Waveform names color

grey Waveform separator color

Font: Tahoma 8

16 Waveform default height

Figure 5-6: Waveform Preferences

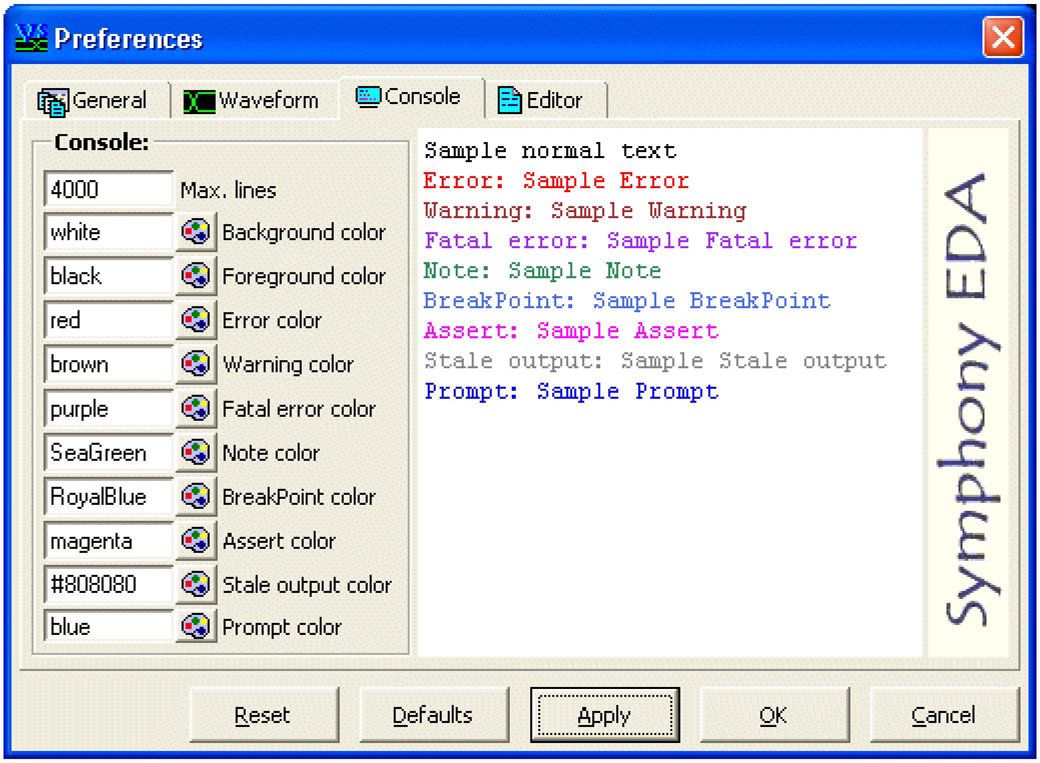
We expect that with time, this dialog box will have more additions and may even have a slightly different organization. Wherever colors are required, common color names are accepted in place of an RGB value. At any time, you can go back to factory defaults by using the "Defaults" button. Note that preferences in *ALL* the tabs will revert to the factory defaults. The following are some important points to remember (available in the "General" tab):

- The "Reload last workspace at startup" only works if the previous session that was editing that workspace exited normally. This is done so that a problem in a workspace reload does not cause Sonata to become useless (creating a situation where Sonata can never be started successfully).

- The "Autoload externally modified files" option if enabled will make using an external editor with Sonata much easier if your workspace contains many files that are constantly changing. Even if this option is enabled, Sonata will still check for editing conflicts but will silently revert a file (to its contents on disk) loaded into the editor if there are no conflicts.

- The "Disable smart compile" as the name implies, will disable minimal compiles. This is not a recommended option unless for some reason you are having trouble with the Sonata heuristics that determine compilation rules.

The following screen shots display the "Console" settings and the text editor settings: The right side of the panel shows the preview of what the settings may look like. Note that the preview functionality only works when you are finished changing an option.

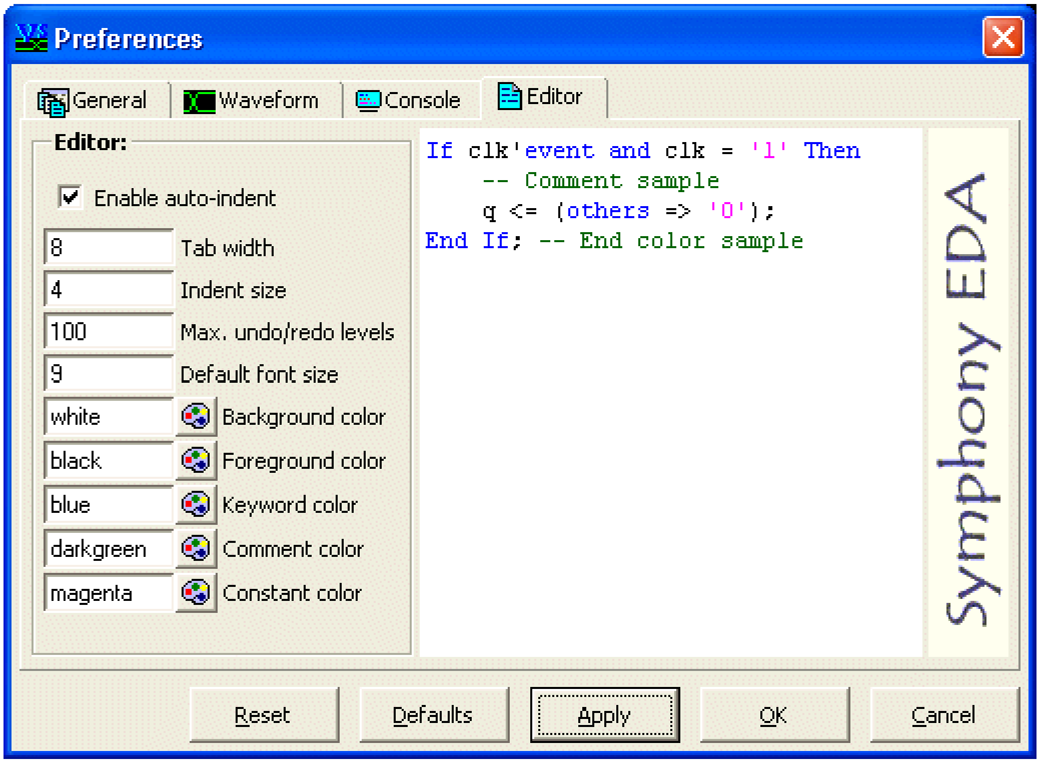


Console:

4000	Max. lines
white	Background color
black	Foreground color
red	Error color
brown	Warning color
purple	Fatal error color
SeaGreen	Note color
RoyalBlue	BreakPoint color
magenta	Assert color
#808080	Stale output color
blue	Prompt color

```
Sample normal text
Error: Sample Error
Warning: Sample Warning
Fatal error: Sample Fatal error
Note: Sample Note
BreakPoint: Sample BreakPoint
Assert: Sample Assert
Stale output: Sample Stale output
Prompt: Sample Prompt
```

Figure 5-7: Console Window Preferences



Editor:

☑ Enable auto-indent

8	Tab width
4	Indent size
100	Max. undo/redo levels
9	Default font size
white	Background color
black	Foreground color
blue	Keyword color
darkgreen	Comment color
magenta	Constant color

```
If clk'event and clk = '1' Then
    -- Comment sample
    q <= (others => '0');
End If; -- End color sample
```

Figure 5-8: Text Editor Preferences

The “Indent size” indicates the number of spaces that have to be inserted when left/right indentation is performed in the editor or when the users presses the enter-key while the auto-indent feature of the text-editor is enabled. The tab-width however controls how Tabs are to be interpreted in text files. In some editors, these two values are interpreted to be the same way. We recommend leaving the “Tab” width at 8 because this is supported universally in all viewers, editors, terminals, etc. and use the “Indent size” to control the formatting of your code. However, this is a matter of personal taste, so Sonata provides the option to override Tab-width.

5.5 Command line arguments

On Windows, you rarely need to use the command line arguments. However there are some useful ways of controlling Sonata at startup. The general format of Sonata command-line is as follows:

```
sonata [options] [--] [file1 ... fileN] [workspacename]
```

The following command-line arguments are supported by Sonata on all platforms:

`-uselicense license-feature-name`

> This causes Sonata to checkout the specified license feature name. The feature-name must be one of “sonatapro”, “sonata” or “sonatafree”. Without this option, Sonata tries them all and checks out the first one available. Using this option, you will force sonata to checkout a specific feature saving time and removing any guesswork. Whichever license type (free,

standard or professional) is used by Sonata, Sonata uses the corresponding license type for "`vhdle`" when launching the simulator. This means that if you use "sonatapro" for Sonata, then it will force the use of "vhdlepro" for the simulator.

On Windows, the shortcuts "Sonata Professional Edition" and "Sonata Standard Edition" make use of this option to directly invoke the appropriate edition.

`-L`

This option forces Sonata to invoke the licensing wizard.

`-guisource filename`

This option can be used to source a Tcl/Tk file in the context of the Sonata Graphical environment automatically at startup. Note that this environment is not the same as the one used by the console or the simulator where only Tcl commands are allowed. This option is intended for advanced users wanting to implement their own graphical interfaces.

`-source filename`

This option can be used to source a Tcl file in the normal context during startup. Note that you cannot use Tk commands in this context.

`-tclcmd command`

This option can be used to run any Tcl command at startup. The "command" must be a valid Tcl command string.

`--`

This option marks the end of option arguments. All subsequent options are considered to be filenames. You typically do not need this option unless you have filenames beginning with a '-'.

`workspacename`

You can specify an optional workspace file name (must have a .sws extension) which will be invoked instead of the last open workspace.

`file1 . . . fileN`

Any argument that is not an options (does not begin with a '-' or appears after a '--'), and is not a workspace name, is considered a text filename is is opened using the Sonata text editor at startup.

Chapter

6

6 Simulation/Waveforms with Sonata

This chapter deals with the task of simulation and debugging your simulation using the waveform viewer. While this chapter will go through each feature of the tool in a clinical way, the Sonata tutorial is better-suited get an overall idea of the user interface.

6.1 The simulation process

Before we delve into the individual features, menu items and buttons, it would probably help to describe the steps involved in a typical simulation run. Assuming that all the necessary files have been compiled successfully, when simulation begins there are a series of steps that are executed:

1. *First, the top-level of the design hierarchy must be identified.* A top-level design is the name of an entity or architecture or a configuration that will serve as the root of your design. In most instances, this is actually your test bench. If a top-level design is already established, you will see it listed in the top-left area of the Workspace pane. To change or set the top-level design unit, use the "`Simulate->Select toplevel…`" menu item.

2. *Start the simulator*: Based on the settings for the workspace (use "`Project->Settings…`" to change the settings), Sonata launches the simulator. Some of the more important options may be SDF settings for back-annotated delays (if your vendor supports this). Sonata works with **`vhdle`** in a client server fashion where the simulator (**`vhdle`**) works in a server mode in the background and Sonata becomes the client. All your workspace settings are transferred to the background simulator.

   The button will stop any existing simulation session, perform a Build (smart-compile) of the current working library to ensure that the compiled contents of the library are up-to-date with respect to their source HDL files and then starts the simulation.

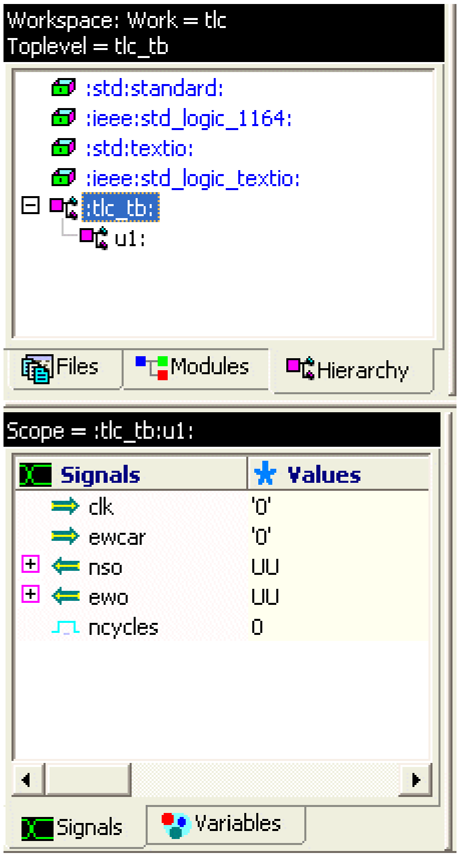
   The button is the "Go" button that is context dependent. It has the following behavior

   - If there is no currently active simulation, then start the simulator

   - If the simulator is in stopped state, advance the simulation time by 100 ns

3. *Elaboration*: The very first thing that **`vhdle`** has to do is to read the compiled form of your design from disk and then, beginning from the top-level, has to read in and configure any and all instantiated design units (component instances). When elaboration is complete your entire design has been initialized and the hierarchical structure is

established. Once elaboration is complete, **`vhdle`** is now waiting for instructions from the client (the user/sonata) for proceeding with the simulation.



Workspace: Work = tlc
Toplevel = tlc_tb

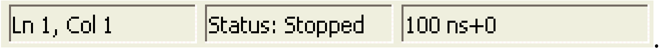
- :std:standard:
- :ieee:std_logic_1164:
- :std:textio:
- :ieee:std_logic_textio:
- :tlc_tb:
  - u1:

Scope = :tlc_tb:u1:

Signals	Values
clk	'0'
ewcar	'0'
nso	UU
ewo	UU
ncycles	0

Figure 6-1: Hierarchy and Objects Pane after elaboration

4. *Initialize Sonata*: Once elaboration is complete, Sonata updates itself with the contents of the hierarchy of the design by querying the background simulator. Since some designs contain a large number of signals and component instantiations, Sonata does this on demand (except for the very top-level which is done as soon as elaboration is complete). The hierarchy is shown in a tree format in the "Workspace" pane while the lower-left pane (Objects pane) shows the objects in the current scope (as shown in Figure 6-1). Note that when simulation begins, the current scope is always the top-level scope. Sonata also creates a Waveform window (in the Document pane) if one does not already exist. The lower right area of the Sonata status bar always displays the current status of the simulator (simulation run-status and current simulation time)



Ln 1, Col 1 | Status: Stopped | 100 ns+0

.

When the simulator is in stopped state, you are allowed to interact with the Hierarchy/Object panes. Interacting with these panes is not allowed while a simulation is running for performance reasons. However, at any time, you can use the ⏸ button to pause (stop) the simulator and then interact with these panes. You can use the standard ▶ or ⏩ buttons to continue simulation after stopping.

5. *Begin simulation.* At the end of elaboration, the simulation is paused just prior to time zero. This means that no simulation time has elapsed and advancing the simulation in any way would potentially advance time. Simulation can be advanced in one of two ways.

   - Advance simulation by a fixed amount: The 1 us ▾ item in the toolbar advances simulation by the specified time.

   - Run the simulation until there are no more events. This can be used if your test-bench and design are done in a way that runs through your tests and then stops creating events. If you have a free running clock in your test-bench for example, the simulation will run to infinity and is not considered an example of a self-stopping design. The ⏩ button runs the entire simulation (until all events have been consumed or until TIME'HIGH has been reached).

6.2 The Objects Pane

Objects are items in your VHDL design that are allowed to have values. Examples of Objects are signals, variables, generics, constants, etc. Note that items such as types, subtypes, subprograms, etc. are not considered objects.

The "Objects Pane" displays the current state of the simulation objects declared in the current scope. While the waveform window contains the current status of an object along with all previous history, the "Objects Pane" provides only a snapshot (to maintain a history of all objects in the "Objects Pane" would require Gigabytes of memory even for a moderate sized simulation). However, unlike the waveform window, you can examine the contents of any part of your hierarchy. You can also examine objects that are global in scope (object declared in packages). You can also examine the values of objects of any type (including FILE, ACCESS types) or any class (signal, variable, generic, etc.)

The current scope for the "Objects Pane" can be set in the "Hierarchy" tab of the "Workspace Pane". To set the current scope, use the following steps.

- Expand the tree control that displays the hierarchy so that the scope you are interested in is visible.

- Double click on the scope item or use the context-sensitive menu (right-click).

The "Objects Pane" is updated every time the simulator is paused. If the same scope is used between two consecutive simulator stops, items that have changed (with respect to the previous value at the previous stop time) are displayed in red.

6.3 Waveforms

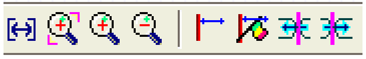
A "waveform" is a graphical representation of a record of the history of changes that have occurred to a given simulation object.

When a simulation is active, it is virtually attached to a single Waveform window. Note that you can have other Waveform windows that represent files on disk (similar to text-editor windows that represent files on disk). The lower left area of the waveform displays the source of the waveform information. A waveform window attached to a simulator is a dynamic waveform window (changes as simulation time advances) while other waveform windows that represent files on disk are static (never change).

A waveform for one or more simulation objects can be displayed in the waveform window. Once an object is selected for inspection, all history is collected for that signal from the time the signal was added to the waveform window to the current simulation time. It is perfectly legal to add a new object to the waveform window after some simulation time has elapsed. You can also add a given object more than once to a waveform window. You can also add partial HDL objects (such as an array element or a record field) to the waveform windows.

If there is enough room between transitions, the value of the signal between transitions is displayed. You also have a choice of the radix used to display values. You can set the radix by selecting one or more signals and using the context-sensitive menu (right-click) to set its radix. The default radix is "auto" which tries to do something reasonable automatically for a given type of the object.

You can also save the current waveform window to disk for later retrieval. Other than the fact that the waveforms loaded from disk are static (do not change), you can use such windows just like a waveform window attached to a simulator. Saved waveform files are stored in an extremely compact form (however, their memory image when loaded into a window can be somewhat larger).

Most waveform operations are performed using either the context sensitive menu or the waveform related buttons  on the toolbar.

6.3.1 Waveform context sensitive menu

The single most important feature in the waveform window is the context sensitive menu. Most features described below are accessible from this menu. This menu is invoked by holding the right mouse button down in the waveform window. *Note that the context sensitive menu may not look exactly like as shown since features are added to Sonata all the time*

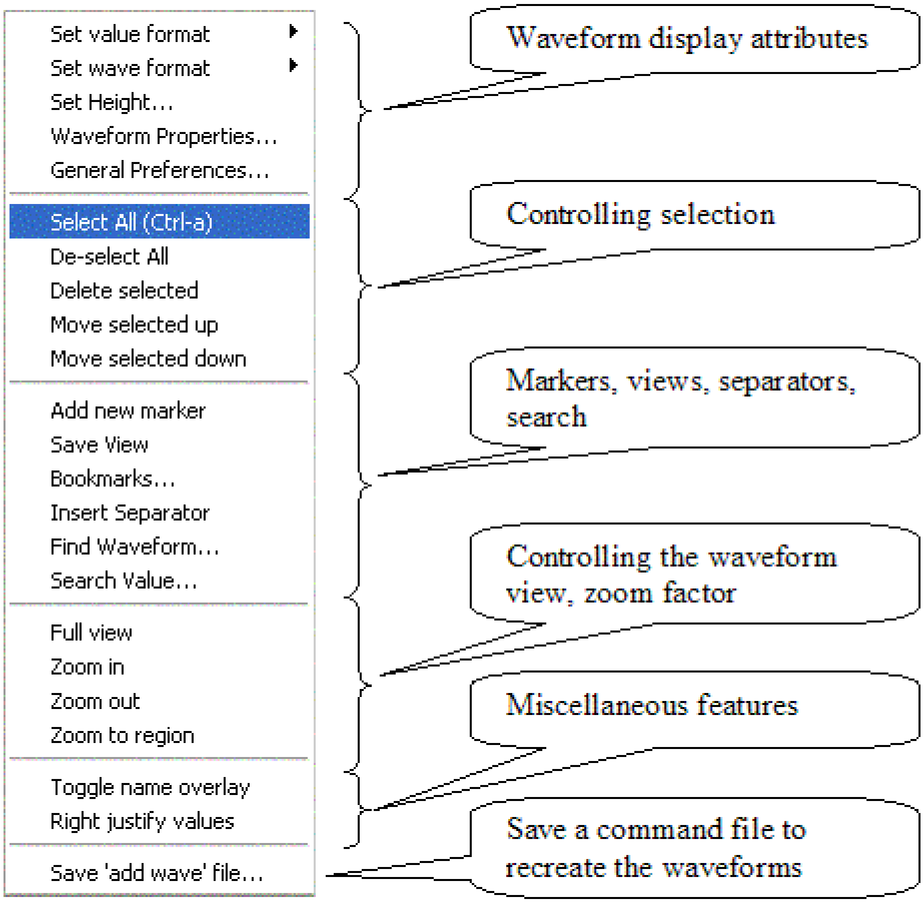


Figure 6-2: Context sensitive menu in the waveform window

6.3.2 Waveform Views

The following view-related operations are supported

- Zoom in using the 🔍button

- Zoom out using the 🔍 button

- Zoom to a specific region using the 🔍 button. To use this button, drag out an area using your left mouse button to create a region between the primary and secondary waveform cursor.

- Quick Zoom: You can drag out a region using your middle mouse button as well. This method does not disturb the position of the primary/secondary cursors.

- Full View: Use the ↔ button to see the entire waveform.

- Panning: You can scroll/pan either using the left/right arrow keys on your keyboard or using the scrollbar. The left/right arrow keys pans one unit (about 10% of the current window) at a time and holding the Control key down while using the left/right arrow keys pans a page at a time. You can also pan vertically using the up/down right arrows (which also have the side effect of changing selection) or using the vertical scrollbar.

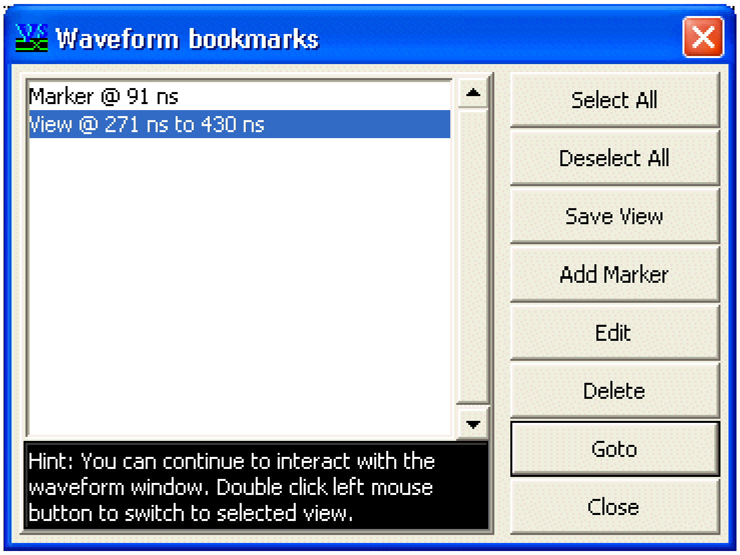
6.3.3 Waveform bookmarks

Unlike the text document bookmarks, waveform bookmarks are maintained on a per wavefrom-file basis and are saved with the waveform file. A waveform bookmark is essentially a way to remember and restore a particular view of the waveform window. Sonata supports two kinds of bookmarks

- Markers: A marker (also called a Waveform Cursor) is a user-defined marker placed in the waveform. A user-defined marker/cursor is created using the button. Typically, when the left mouse button is pressed, the primary cursor is moved to that location and if the mouse drags without the left button being released, then the secondary cursor moves with the mouse. To move a user-defined marker/cursor, you can select the cursor (the cursor becomes fat when the mouse is positioned over it indicating that it is selected), press the left mouse button and drag it to the desired location.

- Views: A "view" is a view into the waveform document. It is defined by the left and right extents of the waveform window. Views are created by using the right mouse button and pressing

Waveform views and bookmarks can also be created and manipulated following dialog box (invoked using the button on the toolbar with the context sensitive menu item "`Bookmarks`...")



Marker @ 91 ns
View @ 271 ns to 430 ns

Hint: You can continue to interact with the waveform window. Double click left mouse button to switch to selected view.

Figure 6-3: Waveform bookmark management dialog

6.3.4 Saving and restoring waveforms

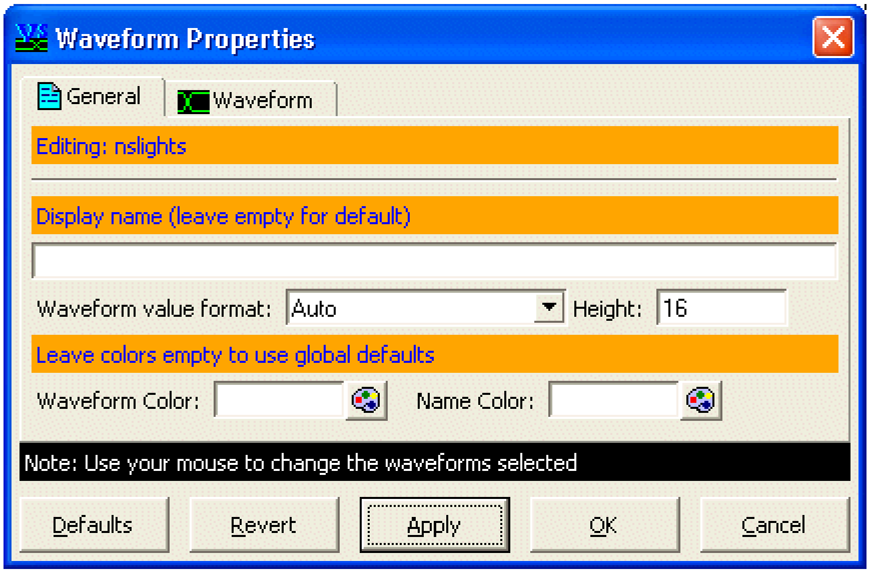
Once you have the waveforms displayed the way you want (with customized display attributes, separators, etc), you can save a command file that will let you restore the waveforms in a future Sonata session. To create such a command file, use the context sensitive menu item “`Save ‘add wave‘ file…`”. Once you have a command file saved, you can restore the waveforms in a future Sonata session using the Tcl command

```
        source filename.tcl
```

in the console window. Note that you may have you use double quotes or use the escape character (backslash) if the file-name contains spaces or other characters that are special to Tcl.

6.3.5 Controlling waveform display properties

While the preferences dialog allows one to set global preferences for colors and fonts used for displaying waveforms, you can override the global display attributes for individual waveforms. Quick changes are provided on the context sensitive menu to directly affect the display attributes and full control is provided using the “Waveform Properties…” menu item. Using this menu item, you will see the following dialog box



Editing: nslights

Display name (leave empty for default)

Waveform value format: Auto

Height: 16

Leave colors empty to use global defaults

Waveform Color:

Name Color:

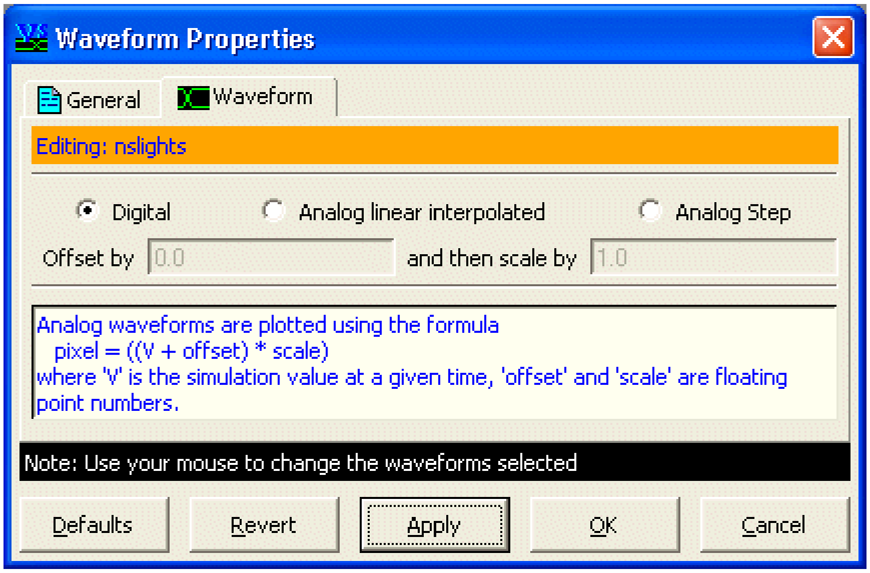
Note: Use your mouse to change the waveforms selected

Figure 6-4: Controlling Waveform Display Properties

Using this dialog box, you can control the display name, the waveform colors, the format used to print the values and the height of the waveform. Leave entries blank to allow the general preferences take effect. Note that any item that is not left blank is saved with the waveform file and will affect other users viewing this file.

6.3.6 Digital or Analog waveforms

The dialog shown in the section above can also be used to control the waveform format using the "Waveform" tab. The choices for the waveform format are "Digital", "Analog linear interpolated" and "Analog Step". Note that when using the analog format, depending upon the scale and offset used, it is possible that it cause the waveform to write over neighboring waveforms.



Editing: nslights

● Digital ○ Analog linear interpolated ○ Analog Step

Offset by 0.0 and then scale by 1.0

Analog waveforms are plotted using the formula
pixel = ((V + offset) * scale)
where 'V' is the simulation value at a given time, 'offset' and 'scale' are floating point numbers.

Note: Use your mouse to change the waveforms selected

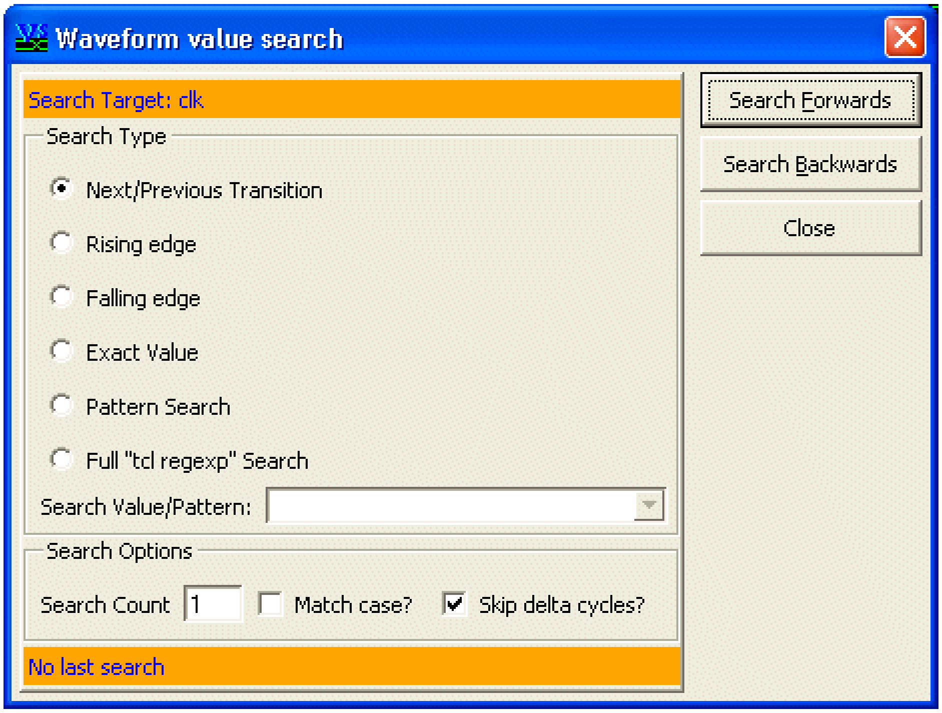
Figure 6-5: Waveform format -- digital or analog

When displaying logic array values (bit_vector, std_logic_vector, unsigned, signed, etc.), normally the vectors are treated as unsigned bits. However, if you set the display format to "Decimal" format then the vector is treated as a signed quantity. To select the "Decimal" format, select the signal in the waveform window and use the "Set value format" or the "Waveform Properties" menu items.

.

6.3.7 Search waveform values

Sonata allows for various kinds of searches on a selected waveform. You can search forwards or backwards in time for a transition, or a particular value. Note that the format of the value (unsigned, hex, decimal, etc.) affects the search results. The following dialog box (invoked with the context sensitive menu) shows the full capability.



Waveform value search

Search Target: clk

Search Type

● Next/Previous Transition

○ Rising edge

○ Falling edge

○ Exact Value

○ Pattern Search

○ Full "tcl regexp" Search

Search Value/Pattern:

Search Options

Search Count 1 □ Match case? ☑ Skip delta cycles?

No last search

Figure 6-6: Search for a waveform value

You can also use ⇥ and ⇤ buttons in the toolbar to go to the next/previous transition without using this dialog box.

Chapter

7

7 Sonata Code Coverage

The Professional Edition allows powerful code coverage analysis. The main use for the kind of analysis provided herein is to be able to measure that the test-benches being developed adequately cover the RTL VHDL code. Code coverage may not be very useful on a post-synthesis/post-layout netlists, which typically have only a gate-level netlist (while you can run the tool on such a netlist, it is unclear how useful such an analysis will be).

This coverage tool was designed for two very usage models

1. **Single Test Bench Usage**: This is the case where the developer of a test bench is targeting a particular device under test (DUT) and wants to measure how effective the test-bench is. In this usage model, simply delete the code-coverage database before starting a new simulation.

2. **Multiple Test Bench Usage**: Another usage model is to be able to have a test suite comprising of many (even hundreds or thousands) of individual test-benches that test various functional aspects of the DUT and the objective is to measure the cumulative coverage. In this usage model, all simulations update a single code-coverage database and once the simulations are completed, the code-coverage database is analysed.

   ☑ It is highly recommended that the simulations be run serially so as to avoid contesting for write access to the database. It is also recommended that the multiple runs be done in batch mode (using vhdle) and once the simulations are completed, Sonata can be used to analyse the results.

7.1 Coverage Database Initiatialization

Code coverage is enabled by the option `-coverage` option on `vhdle`. The code coverage options are documented in section 11.3. The most important aspect of the initialization phase to understand is how the global and project exceptions are managed.

7.1.1 Global Exceptions

When the coverage database is initialized, you can register a set of global exceptions. Global exceptions cannot be added once the database is initialized (without re-creating the database). Global exceptions are typically maintained in a site-wide file. The default global exceptions are documented in `"<install-dir>/bin/coverage.ini"`. No coverage data is collected for global exceptions

7.1.2 Project Exceptions

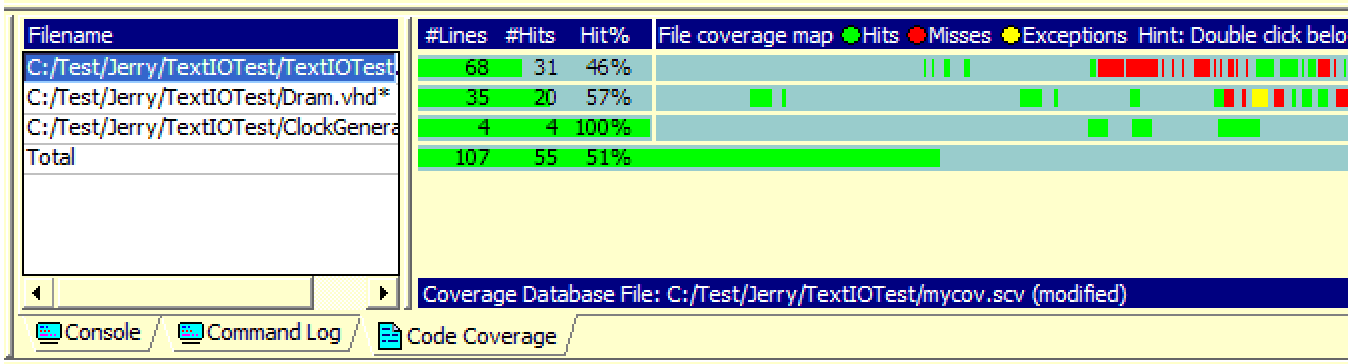
Once a coverage database is created and one or more simulation runs have completed, you can create project level exceptions using Sonata. Project level exceptions change how the data is analyzed or viewed but does not actually change the database itself. This means that if you disable coverage for a file and later enable it, the coverage information for that file is still intact. Using project exceptions, you can specify either file level exceptions or exclude individual lines (or ranges of lines) in a particular file.

7.2 Code coverage analysis

There are three ways to analyze/view the data collected during simulation. You can use Sonata to analyze the results, you can convert the database to a text format (helpful for post processing the data), or you can view the results in a HTML file (mainly for presentation and printing purposes).

7.2.1 Coverage Analysis using Sonata

The primary way is to load the code-coverage database into Sonata (use `File->Open Coverage Database…` menu item). This will show the coverage map and over-all statistics in the "Code Coverage" tab of the console.



Filename	#Lines	#Hits	Hit%	File coverage map ● Hits ● Misses ● Exceptions Hint: Double click belo
C:/Test/Jerry/TextIOTest/TextIOTest.	68	31	46%	
C:/Test/Jerry/TextIOTest/Dram.vhd*	35	20	57%	
C:/Test/Jerry/TextIOTest/ClockGenera	4	4	100%	
Total	107	55	51%	

Coverage Database File: C:/Test/Jerry/TextIOTest/mycov.scv (modified)

Figure 7-1: Code Coverage tab showing a small coverage database

Figure 7-1 illustrates a sample coverage database. The right pane shows the coverage for each file as a bar graph overlaid with a table of numbers. Each row in the table contains the number of lines of code contained in the file, the number of lines actually hit during the simulation and finally, the percentage. Note that the numbers in the '#Lines' do not include comments, blank lines or declarations. It consists of actual statements that have can be executed and may also exclude lines that have been optimized out by the compiler/simulator. After the 'Hit%', the coverage data is shown graphically with green areas representing hits, red areas representing misses, yellow areas representing the exceptions and the background color representing the entire file. You can double click in this region to open the text-editor.

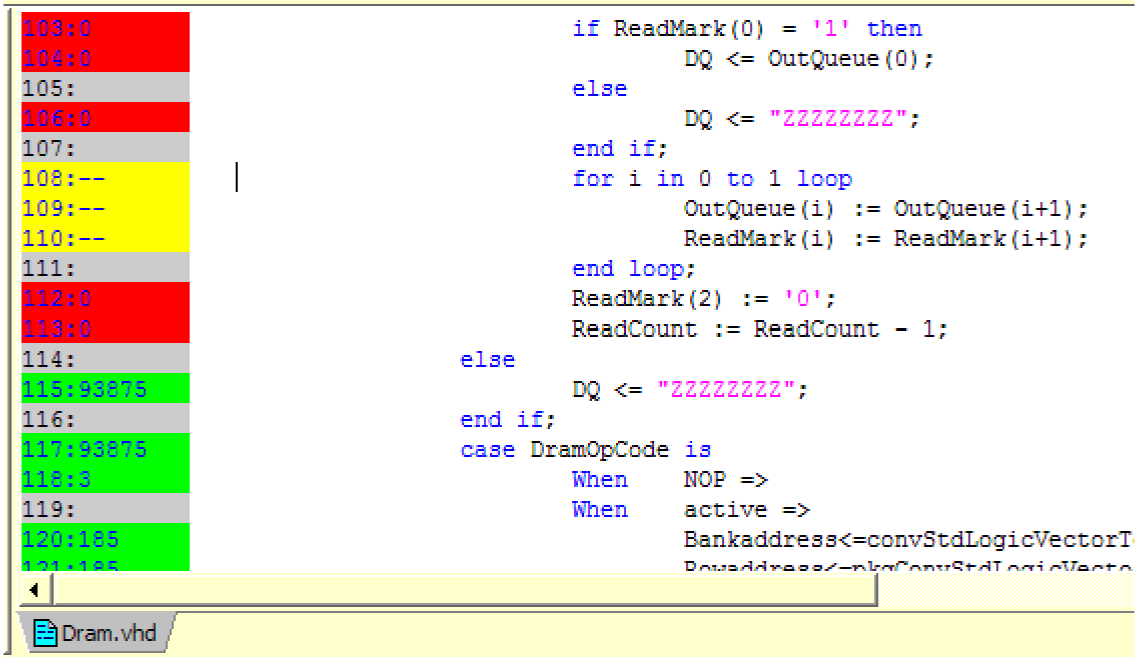
The following are some important aspects of this window

1. The right mouse button can be used to invoke a context sensitive menu that contains many options.

2. You can sort by either the "Filename" column or one of the columns in the right pane. Place your mouse on the title for the column and click with your left mouse button

3. You can manually position the files by selecting, dragging and dropping using your left mouse button.

4. Double click in the coverage map area (right pane) to view the file in the text editor. When the coverage database is open, the text editor annotates the detailed coverage information whenever it can (if the file in the editor is participating in the coverage analysis). The following is a sample text editor with coverage data.



```
103:0                   if ReadMark(0) = '1' then
104:0                           DQ <= OutQueue(0);
105:                    else
106:0                           DQ <= "ZZZZZZZZ";
107:                    end if;
108:--                  for i in 0 to 1 loop
109:--                          OutQueue(i) := OutQueue(i+1);
110:--                          ReadMark(i) := ReadMark(i+1);
111:                    end loop;
112:0                   ReadMark(2) := '0';
113:0                   ReadCount := ReadCount - 1;
114:            else
115:93875                   DQ <= "ZZZZZZZZ";
116:            end if;
117:93875       case DramOpCode is
118:3                   When    NOP =>
119:                    When    active =>
120:185                           Bankaddress<=convStdLogicVectorT
121:185                           Rowaddress<=pkgConvStdLogicVecto
```

Figure 7-2: Text editor showing coverage data

7.2.2 Convert to Text format

You can convert the coverage database to a human readable format (text). Use the context sensitive menu in the coverage window and invoke the "`Generate text report file…`" menu item. When this menu item is invoked, a dialog box will allow access to various options (file name, detailed vs. summary only, sort by hits or line-numbers, sort order, etc.) that control the generation of the text report file. After generating the report file, it is automatically opened in the text editor

7.2.3 Convert to HTML format

You can convert the coverage database to a HTML format. This format is very useful for presentation purposes as well as printing. Use the context sensitive menu in the coverage window and invoke the "`Generate Html report file…`" menu item. When this menu item is invoked, a dialog box will allow access to various options (file name, detailed vs. summary only, sort by hits or line-numbers, sort order, etc.) that control the generation of the text report file. After generating the HTML file, it is automatically opened in your favourite browser.

Chapter

8

8 Sonata Tutorial

The primary intent of this chapter is to explore various aspects of Sonata using a small design as a simulation target. There are other tutorials that serve as examples for the command line tools. This tutorial focuses on using Sonata.

☑ In this tutorial, there are some references to context-sensitive menus. A context sensitive menu is a menu that is invoked when the user right clicks on the mouse -- a popup menu appears with items that are appropriate for the object or area in the vicinity of the mouse pointer. Using these menus can make many tasks easier and causes less crowding in the main menu-bar.

As mentioned in earlier chapters, many aspects of Sonata uses forward slashes as directory separators. Other than Microsoft native dialog boxes (File Open, Save As, etc.) forward slashes are supported for directory separators universally.

8.1 Starting Sonata

The very first step in the tutorial is to start Sonata.

- Windows users: Use "`Start->Programs->Symphony EDA->VHDL Simili X.Y->Sonata`" on Windows. You can also do a "`Start->Run`" and enter **`sonata`** as the command to run and this will run the last installed version of Sonata.

- Unix users: Use the command "**`sonata`**" on Unix/Linux.

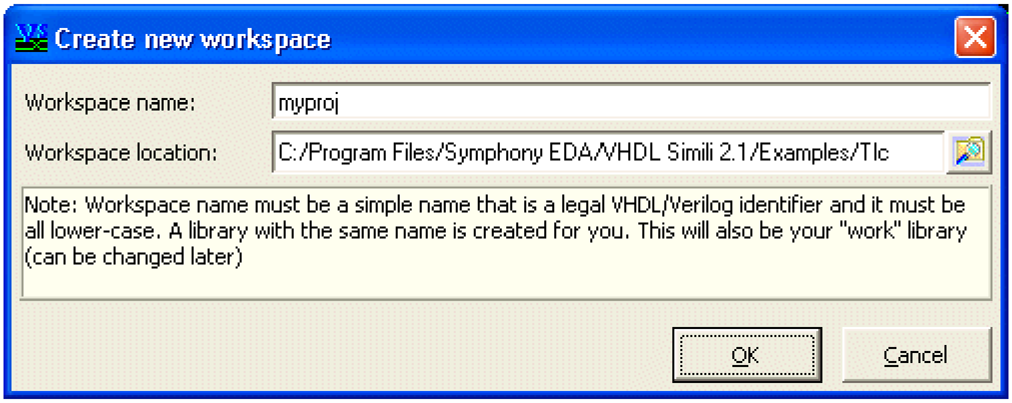
If you do not have a proper license, you will either see a dialog box from FLEX*lm*, the license manager (on Windows only) or the Licensing Wizard itself will start. On Windows, if the licensing is not properly configured, there may be a significant pause before you see a dialog box indicating failure. On Windows, on a failure to secure a proper license, you may also see a dialog box from FLEX*lm* license manager. If you do not have a license file, "Cancel"" the FLEX*lm* dialog box to start the Licensing Wizard. Please ensure that you have the license file installed and that the license management software is enabled (started).

8.2 Creating a Workspace

We need to create a workspace where the design with all its relevant files are managed and organized. The name of the workspace will also be the name of the "work" library. This "work" library will be used as the current working library into which all the tutorial's VHDL files will be compiled. Note that using Sonata, you cannot use "work" as the name of the workspace as this can conflict with Sonata's capability of managing multiple libraries in a given workspace and its ability to switch between different "working" libraries.

For this tutorial, we will use the Tlc tutorial (this is the same design that is also used for the command line tutorial). This tutorial is located in the "<install-dir>/examples/tlc" directory. If you do not have write permission in this directory, please copy this directory to a temporary place and use the copy of this directory instead. You can also create the workspace in any directory you wish but for this tutorial, create it in the same directory where the source files for the Tlc design are located.

To create a new workspace, use the "`File->New Workspace…`" menu item.



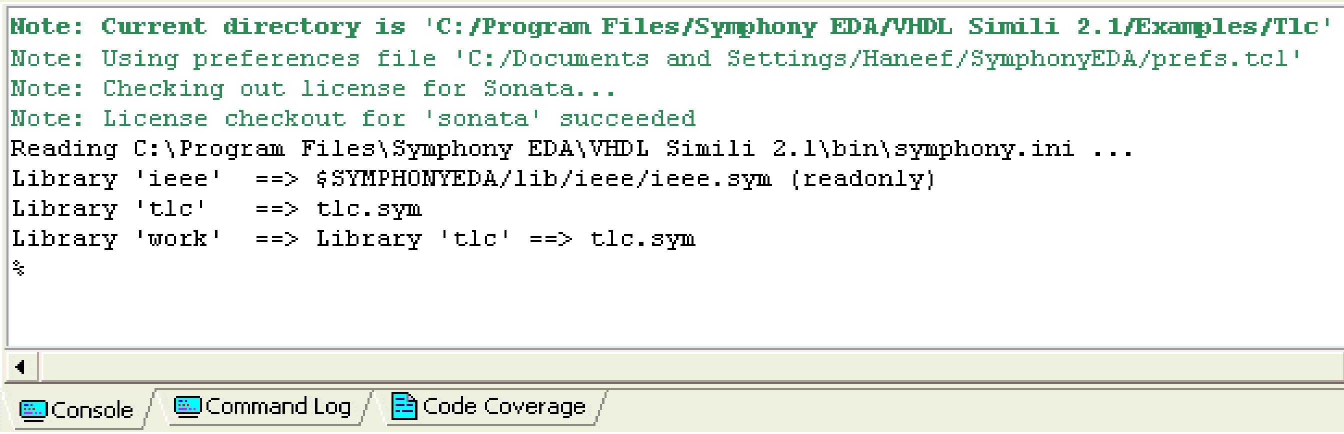
Workspace name: myproj

Workspace location: C:/Program Files/Symphony EDA/VHDL Simili 2.1/Examples/Tlc

Note: Workspace name must be a simple name that is a legal VHDL/Verilog identifier and it must be all lower-case. A library with the same name is created for you. This will also be your "work" library (can be changed later)

Figure 8-1 : Create New Workspace

Pressing OK in this dialog box will create a workspace (with a .sws extension) and a similarly named library (with a .sym extension). The Console window will contain several messages indicating the status of the libraries and the project management pane will list the libraries in a tree control. For each of the libraries included in the workspace, the project management pane lists its physical location on disk. Read-only libraries are displayed with a gray color and the current working library is displayed using a bold-red font.



```
Note: Current directory is 'C:/Program Files/Symphony EDA/VHDL Simili 2.1/Examples/Tlc'
Note: Using preferences file 'C:/Documents and Settings/Haneef/SymphonyEDA/prefs.tcl'
Note: Checking out license for Sonata...
Note: License checkout for 'sonata' succeeded
Reading C:\Program Files\Symphony EDA\VHDL Simili 2.1\bin\symphony.ini ...
Library 'ieee'  ==> $SYMPHONYEDA/lib/ieee/ieee.sym (readonly)
Library 'tlc'   ==> tlc.sym
Library 'work'  ==> Library 'tlc' ==> tlc.sym
%
```

Figure 8-2 : Console window after creating a new workspace

8.3 Add files to the Workspace

The next step is to associate a set of VHDL files with the current working library. There are several ways of doing this. You can use one of the buttons from the tool bar to add existing file(s) to the project. You can also use the "Project" menu or the context sensitive menu on the library name (in this case the tlc library in the project management pane). Using one of these methods, please add the files "tlc.vhd" and "tlc_tb.vhd". Figure 8-3 is a screen shot of the workspace status after the two files are added (you may have to expand the tree control crosses to the left of the tlc library to see the files and the design units they contain).

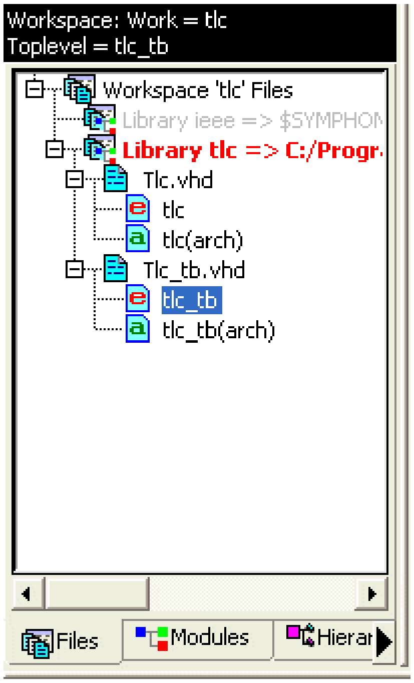


Figure 8-3 : Project management pane just after creating a new workspace

On Unix, the file names in the tutorial are all lower case. The screen shots shown here were captured on the Windows platform.

If you wish to examine the contents of one of these files, you can double click your left mouse button to open the files in the Document-Pane. Since by default, automatic file ordering is enabled, the two files added to the project will arrange themselves as shown in Figure 8-3 automatically.

Click on the "Modules" tab and view the contents of each of the libraries. You will notice that the "ieee" library has some design units listed for it but the "tlc" library is empty at this point. This is because we have not yet compiled the VHDL files. Adding a file to the project does not automatically compile it.

8.4 Compiling in Sonata

To compile all the files associated with the current working library, press F7. View the "Compile" menu to see alternate ways of compiling files. The process of compiling will result in the design units (entities, architectures, packages, etc.) to be added in compiled form to the current working library (tlc). Once compilation is completed, the "tlc" library in the "Modules" tab will now be populated with design units. You can double click on the design units to go directly to the source file (and line number) of the design unit.

8.5 Simulating in Sonata

Simulation control is easier done using the simulation/waveform toolbar. The left part of the tool bar has controls that start and advance simulation. The middle part contains controls to pause or stop a running simulation. The right side of this graphic contains buttons that control the waveform display.

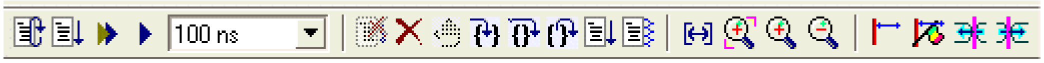
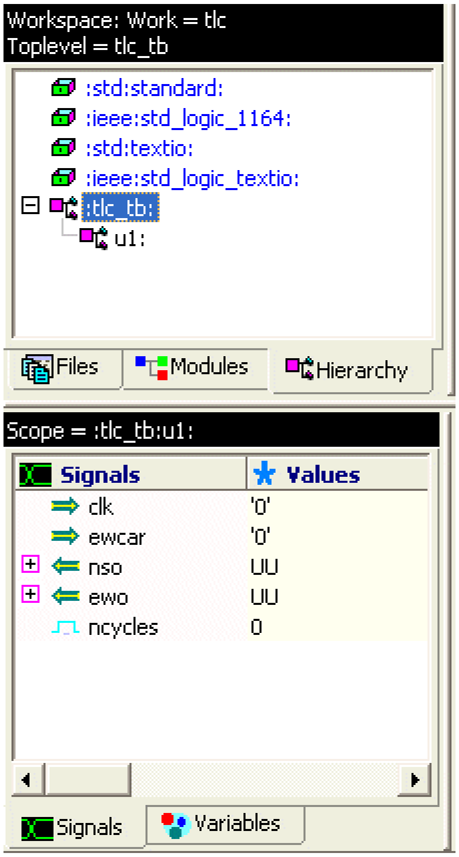


Figure 8-4 : Simulation and Waveform controls

Most of this functionality is also available from the "Simulate" menu.

To begin simulation, click on the ▤ or ▤ button. At this point, since we have not yet selected a design unit as the top-level design unit for simulation, you will be prompted with a dialog box to select one. Please select "tlc_tb" as your top-level design unit.



Workspace: Work = tlc
Toplevel = tlc_tb

:std:standard:
:ieee:std_logic_1164:
:std:textio:
:ieee:std_logic_textio:
:tlc_tb:
u1:

Scope = :tlc_tb:u1:

Signals	Values
clk	'0'
ewcar	'0'
nso	UU
ewo	UU
ncycles	0

Figure 8-5 : Hierarchy window and Object panes populated when simulation begins

When simulation begins, several things occur (see also Figure 8-5).

- A connection to the simulator (**vhdle**) is opened which runs in the background - This may not be important information at this point but it may help you understand the tool somewhat better in the future.
- The "Console" window shows the results of the elaboration. Elaboration is the process of initializing the simulation.

- The “Hierarchy” tab is now displayed in the “Project Management Pane”. This window will now be populated with a tree control representing the structure of the design being simulated.

- The “Objects Pane” at the bottom left area of the Sonata framework is now populated with the objects (Signals and Variables) of the top-level design. Note that while the “Signals” tab only contains signals (and ports), the “Variables” tab contains all other VHDL objects that have values associated with them. This includes constants, generics, shared-variables, loop variables, etc. The “Objects Pane” always displays the current values of objects in the selected scope. Also, newly changed values are shown in red. The Objects Pane also displays the scope for which the Objects belong to. You can change the scope displayed in the Objects Pane by selecting (double-clicking) the appropriate branch of the hierarchy in the “Hierarchy” tab. The object pane is refreshed every time the simulation pauses.

- The waveform window is created (if one does not already exist) and is attached to the simulator. Initially, the waveform window does not have any objects displayed.

8.6 Working with Waveforms

To display waveforms for VHDL signals (or non-subprogram variables), they have to be added to the waveform window. Before we proceed, make sure that your scope reads “:tlc_tb:” - this is the top-level. Waveforms can be added to the waveform window by using one of the following methods:

1. Use the context-sensitive menu (right click) in the Object Pane’s “Signals” tab. You can add all signals, only the selected ones or just the ports defined in the current scope (our test-bench does not have any ports).

2. You can select signals in the Objects Pane and drag and drop them into the waveform window.

3. You can use the command-line in the Console window (not recommended for this tutorial).

Please add all the signals listed for the top-level scope to the waveform window using one of the above methods. Note that you can add a given signal multiple times. You can remove a signal from the waveform window by selecting the signal(s) in the Waveform window and pressing the “Delete” key. You can also re-arrange the order in which signals are displayed by selecting the appropriate signals and then using context sensitive menu (right mouse button) or using the left mouse button to drag and drop them into position. Composite signals (arrays and records) are shown using a tree structure.

8.7 Running the simulation

Use one of the run buttons ⏩ or ▶ to advance simulation. Running the simulation will display the simulation behavior of the signals being probed in the waveform window. You can use the context sensitive menu in the waveform window to select an appropriate radix. Use either of the two buttons to run the simulation until the simulation ends. Since this test-bench is written in a way that automatically stops producing events when the simulation finishes, you can use the "Run All" ▶ button.

Any output created by the VHDL model, is displayed in the Console window. Such output can be a result of messages from the simulator itself or messages written explicitly by the VHDL model (using ASSERTs, REPORTs and FILE I/O).

8.8 Using the Waveform Cursors

The waveform window can optionally have zero or more cursors. Using cursors in Sonata may take some getting used to because the behavior may not be the same as the simulator you are accustomed to. The values shown to the left of the waveforms represent the values at the time point where the last cursor was drawn. In Sonata, there are three kinds of cursors. Two of the three cursors are dynamic cursors.

8.8.1 Dynamic Cursors

The dynamic cursors always follow your left mouse clicks. Among the dynamic cursors, there is a primary cursor and a secondary cursor. You can place the "Primary" cursor anywhere using your left mouse button (single click) and if you start dragging your mouse while holding the left button down, the secondary cursor starts measuring from the point of the primary cursor. All cursors snap to a waveform transition if the mouse pointer is within a few pixels (short distance) of a transition. You can disable the "snapping" behavior either by hold the Shift key down or by placing the mouse pointer where there is no waveform transition nearby.

8.8.2 Static (user defined) Cursors

Static cursors are user-defined cursors. These cursors do not move automatically. You can place a static cursor by clicking on the button. Once they are placed in the Waveform window, they can be selected using your left mouse button and dragged around to reposition them. Static cursors snap similar to the dynamic cursors (with the Shift key disabling snapping). Positioning your mouse pointer near a static cursor selects it (it drawn using a thicker line). Once selected, you can drag it with your mouse. A static cursor is also a type of a waveform bookmark and can be manipulated (deleted/edited) as one (see section 8.8.3).

Please practice measurements with the dynamic cursors before you shutdown the tutorial. Once you are comfortable the dynamic cursors, create one or two static cursors and observe their behavior.

8.8.3 Waveform bookmarks

In Sonata, there are two kinds of waveform bookmarks. A static cursor is one type of bookmark. The other kind of bookmark is a waveform view port. A waveform view port includes the left and right extents of the waveform window. You can also name individual bookmarks. Waveform bookmarks are manipulated (edited/deleted) using the button. Waveform bookmarks are saved along with the waveform file.

Chapter
9

9 Sonata Command Reference

This chapter deals with commands that can be executed inside the Sonata console environment. The command-line format and arguments are described in section 5.5. The basic syntax used in the Sonata console is actually the syntax dictated by the Tcl language. You can find more information about the Tcl language and the Tk toolkit at http://dev.scriptics.com.

9.1 Syntax summary

While there are many books and detailed documentation (http://dev.scriptics.com) available, this section gives a brief summary of the Tcl syntax.

All Tcl commands follow the format

```
command-name {arguments}
```

In the Sonata console, if "command-name" is a valid (defined) Tcl command-name, it will be executed with any arguments. If it is not a valid Tcl command, it will try to execute it as an external command if an executable or script with the same name is found (in your PATH). Note that you must not run any external command that requires input on its "stdin" (such commands may fail). Command arguments are always separated by white space; and the Return/Enter key (or the end-of-line) terminates the command.

When arguments contains spaces (for example 10 ns), they must be enclosed in quotes or {} (braces). The following are some examples

```
run {10 ns}
run ”10 ns”
vhdlp ”my file with spaces.vhd”
```

Without spaces or braces, each of the words may be interpreted as separate arguments, which is probably not the intent here.

The characters inside the quotes or braces are taken literally. However, quotes are terminated by a newline (unless escaped by a backslash at the end of the line) and braces will allow you to continue across multiple lines until a matching close brace is seen. Note that braces can also be nested. The backslash character is a general escape character and can be used either as a line continuation character or to escape characters that are special to Tcl (like braces, quotes and spaces).

9.2 Simulator commands

While there are many Tcl commands this section documents only those commands that are relevant to the simulation task are explained.

```
vhdle [options] top-level-module-name
```

The command is used to start a new simulation session within Sonata. It prepares the Sonata GUI to receive data from the simulator and then launches the simulator (`vhdle`). The most important option to remember when running `vhdle` is the –ini option to force the simulator to use the same library setup as Sonata. For more information on xxx, please refere to the Simulator Reference (**`vhdle`**) chapter.

The following commands are only valid when a simulation session is active. **Note that your license may or may not allow some of these commands**.

```
add wave [options] signal-path1 [signal-path2 ... signal-pathN]
```

This command is used to add waveforms to the waveform window. Using this command, you can either add signals or VHDL variables to the waveform window. If the object being added is a VHDL variable, there are some special rules.

- The variable must be declared in a process, package, architecture or block declarative region. Specifically, variables declared inside functions and procedures cannot be added because they appear and disappear with time
- The variable cannot be a loop-variable
- It is okay for the variable class to be a constant or generic
- You cannot monitor a portion of a variable (for example vector(3)). Sonata only supports monitoring full variables
- The variable name cannot be an alias (this may be supported in a future release).

If the object is a signal type, the the following rules apply

- You can monitor just a portion of a signal (for example vector(3)).
- Any signal declared anywhere in the design heiarchy can be monitored.

Note that monitoring a large number of signals/variables can have a negative impact on the simulation performance.

The following options are support for waveforms:

`-alias alias`

This option can be used to use a display name that is different from the default full object name. When this option is not specified, the full name of the waveform object is used instead.

`-format digital|analog|analog-step`

The default waveform format is "digital". If you specify analog (linear interpolated) or analog-step, you may also wish to set the –offset and –scale options appropriately. To use the analog waveform formats, the type of the waveform must be of an appropriate type – a type that can be converted to a number. Bit-vectors and std_logic_vectors are supported and are interpreted as unsigned values.

The points (pixel offset) of the analog waveform are calculated using the following formula

```
pixel = ((V + offset) * scale) ;

Where 'V' is the value at a given time and offset and
scale are specified with the corresponding opions.
```

`-namecolor color`

This option can be used to display the waveform name in specified color. Colors can be specified using common color-names or using a hexadecimal color value of the format #rrggbb where 'rr', 'gg' and 'bb' represent two hex-digit values for redm blue and green intensities. When this option is not specified, the color specified in the Waveform preferences is used.

`-offset real-number`

The option is only useful when using one of the analog waveform formats. The default is 0.0. See –format.

`-pos integer`

With this option, you can specify the location where the newly added waveform is to be placed. By default, new waveforms are inserted at the end of the already existing waveforms. The 'integer' is the position (starting from 0) where the new waveform(s) are inserted.

`-radix binary|symbolic|hexadecimal|hex|octal|decimal|unsigned|auto`

Using this option, you can specify the radix to be used when displaying the waveform values. The default is "auto". To allow proper converstions to hex, octal, decimal and unsigned numbers, the object must be of an appropriate type (bit_vector and std_logic_vecots are supported along with integers).

`-scale real-number`

The option is only useful when using one of the analog waveform formats. The default is 1.0. See –format.

`-wavecolor color`

This option can be used to display the waveform in specified color. Colors can be specified using common color-names or using a hexadecimal color value of the format #rrggbb where 'rr', 'gg' and 'bb' represent two hex-digit values for redm blue and green intensities. When this option is not specified, the color specified in the Waveform preferences is used.

`add separator [options]`

This command is used to add a separator item to the waveform window. The display options used for "add wave" are also applicable to this command. Since the "separator" does not have a default name, use the –alias option to specify an arbitrary string to be used as the display string.

`bp set [options] filename linenum`

The "bp set" command creates a breakpoint. This command is only valid when a simulation session is active. The filename must be a valid filename that is readable on disk and must be a file that is actually participating in the current simulation session. If the file-name is a valid filename, the simulator will attempt to place the breakpoint at the closest actual line of code. Note that optimizations may cause certain lines of code to disappear and you cannot place breakpoints inside declarative regions.

When a break point is set successfully, it is associated with a integer id. This is printed to the console. This 'id' can be used in other breakpoint related commands. The following options are supported:

`-count count`

You can cause the break point to be invoked when the line is hit the specified number of times. The default is 1 (stops every time execution reaches the breakpoint line).

`-command command-string`

When a breakpoint is encountered, you can ask the simulator to execute the specified Tcl command. One of the commands you can use is the "run –continue" command

`-region region`

You can specify a specific region that this breakpoint refers to. Normally it applies to all regions the breakpoint is enclosed in. The region is a string representing the heriarchical path name (separated by ':'). The region name is not unlike the region names displayed in the Hiearchy window.

`bp clear id|ALL`

The "bp clear" command removes the specified breakpoint(s). This command is only valid when a simulation session is active. The argument to this command must be a valid breakpoint id or the keyword "ALL", which can be used to remove all breakpoints.

`bp disable id|ALL`

The "bp clear" command disables the specified breakpoint. This command is only valid when a simulation session is active. The argument to this command must be a valid breakpoint id or the keyword "ALL", which can be used to disables all breakpoints. Note that a disabled breakpoint can be re-enabled at a later time using the "bp enable" command.

`bp enable id|ALL`

The "bp clear" command enables the specified breakpoint(s). This command is only valid when a simulation session is active. The argument to this command must be a valid breakpoint id or the keyword "ALL", which can be used to enable all breakpoints.

`bp list`

The "bp list" command lists (prints to the console) the breakpoints currently defined in the simulator.

`drivers signal1 [signal2 ... signalN]`

This command prints information about all known drivers and sources of the given signals. For each driver, it prints the current value of the driver along with any projected future values. For each source that is a result of a component output/inout connection, it will only print the current value of that source.

`force [options] signal value [time] {value time}`

This command can be used modify/force signals to a particular value. The 'time' values must use valid VHDL time format and must be enclosed in quotes or braces to avoid confusion with any space characgers in the time value. If a time value is preceded by the '@' character, it is interpreted as an absolute time value. Otherwise, the time value is interpreted as a relative time value (relative to the current simulation time). You can also optionally specify multiple values for multiple time values (time must be in ascending order).

The following options are allowed for the "force" command:

`-freeze | -deposit | -drive`

These options are mutually exclusive and specify the kind of "force" to be applied to the target signal. The –freeze option freezes the signal at the specified value and any drivers that are trying to drive the signal from within the simulation will be overridded. The –deposit option simply deposits the specified value but may be overridded when a driver/source tries to set the value. For input signals that do not have any sources, this is effectively the same as the –freeze option. The –drive option creates an additional driver for the specified signal. The target signal must be a resolved type to be able to use the –drive option.

`-cancel time`

The "force" is automatically cancelled at the specified time.

`-repeat time`

The "force" is automatically repeated at the specified time. The time value must be a relative time for this option.

```
release signal1 [signal2 ... signalN]
```

This command releases any forces that are in place for the specified signals.

```
noforce signal1 [signal2 ... signalN]
```

The "noforce" command is an alias to the "release" command.

```
tb
```

The "tb" command lists (prints to the console) the current stack-strace (also called trace-back) if any. This command makes sense when the simulator breaks at a breakpoint or an ASSERT statement.

```
quit
```

This command ends the current simulation session. If the code-coverage is active, then the code-coverage database is updated

Chapter

10

10 VHDL Compiler Reference (vhdlp)

The **vhdlp** compiler performs the task of library creation as well as compilation. A few VHDL tools perform these two tasks using separate commands. The rest of this page contains three sections -- some background information about VHDL compilers, how to compile VHDL with **vhdlp**, and the various options available with **vhdlp**. If you are using "Sonata", the compiler options documented here can be set using a dialog box. Yet, it is still helpful to understand the various options and their exact effect.

10.1 Backgrounder

For any VHDL model to be simulated, the first step is to compile the model. In VHDL, a given VHDL model file does not have to be self-contained - it may contain external references. Examples of external references include things such as library names, package names, etc. It is very important that all external references be available when a VHDL file is being compiled - there are some exceptions but this is the general rule. This same rule (define before use) also applies to items within a given VHDL file.

10.2 How to compile

The **vhdlp** program performs the task of library management as well as compilation. The **vhdlp** program is a batch (command-line) program. The **vhdlp** program offers several options. You can get a synopsis of all the options by typing **vhdlp** without any arguments (or with the -h option). The **vhdlp** program can be used to refresh an existing library compiled with older versions of the compiler or libraries or it can be used to compile VHDL source files. The following is the syntax for **vhdlp**:

To compile VHDL source files:

```
vhdlp [options] file1 [file2 file3 ... fileN]
```

To compile refresh an existing library (see the **-refresh** option for more information)

```
vhdlp [-ini filename] [-work libname] [-nostderr] [-silent] -refresh
```

You can compile one or more files at a time using **vhdlp**. As mentioned above, unless the files are unrelated to each other, the order of compilation is important. The following demonstrates a typical usage of the compiler:

```
vhdlp myfile.vhd
```

When a VHDL file is compiled, the design units (entities, packages, etc.) contained within that file are compiled and stored in the "work" library. A library is typically a directory with a ".sym" extension. The contents of this directory are mostly of binary nature and do not contain anything of interest in a human readable form. Remove a ".sym" directory essentially removes all design units stored in that library.

10.3Compiler options

The following options are listed in order of their relative importance.

`-work library-name`

You can use this option to override the default library (work). Using this option, you can set the logical library name "work" to some other library name. For example:

```
vhdlp -work mylib myfile.vhd
```

The above command will compile the contents of myfile.vhd into the library mylib. The compiler will also create the library if it does not already exist (see Introduction).

`-ini initialization-file-name`

You can use this option to override the default INI file name used to setup library associations. When this option is not specified, a search is done to find the file "symphony.ini" in the current working directory first and then in the installation directory. You can use this option to specify any file to be read as an INI file (as long as the file has the expected format). See Introduction for more information on the INI file. Hint: You can use the Sonata project file (.sws file) as an INI file since it uses the same format as the INI file. It is an error if this option is specified but file could not be opened for reading.

`-s`

You can use this option to compile in silent mode. In the silent mode, very few informative messages appear as output. The compiler only issues messages about warnings, errors, etc.

`-x`

This option is intended for advanced users and is more than likely not required for most users. This option enables strict conformance to the VHDL93 standard with regards to overload resolution. More precisely, when you use the -x option, an explicit subprogram declaration cannot hide an implicit subprogram declaration if those declarations appear in different declarative regions.

VHDL allows users to have user-defined types as well as overloaded operations. However, according to the VHDL rules, when a type is defined, a set of default operations (such as "+", "=", etc.) for such types are also defined. The language also allows overriding such default overloading-definitions within the same region (the region where the type is defined). However, if such operators are overloaded in other packages, you might run into problems with overload resolution when you use this operator. For example, if a type "MYTYPE" is defined in package "SOMEPACKAGE", and if you overload a default operation in another package (or region), you might run into problems with overload resolution when you use this operator. However, this is precisely what some popular packages do and strict conformance with the VHDL 93 standard may cause either those packages or your VHDL files referring to those packages to fail to compile.

By default, **vhdlp** gives higher precedence to an explicit override of a pre-defined operator regardless of where the over-riding occurred. Use this option to disable this feature.

In certain obscure cases, it is possible that default behavior may cause certain VHDL files to not compile. The ModelSim std_developerskit library is an example of this behavior. In such cases, you will have to use the -x option (or -strict option) to use the stricter rules of VHDL operator overloading.

`-87`

VHDL Simili is primarily a VHDL 93 compiler. This option enables certain amount of compatibility with VHDL 87. It is highly recommended that you do not use VHDL'87 constructs and instead change your source code to comply with VHDL 93 standard. Where this is not possible, this option may be of some help.

This option allows VHDL'87 style FILE declarations in addition to the VHDL'93 style FILE declarations. Use of this option is not recommended since the semantics of FILE types are much better dealt with in the VHDL'93 specification. We recommend that you change your existing designs if possible instead of using this option. Note that this option does not compensate for other (non FILE related) incompatibilities between VHDL'93 and VHDL'87. VHDL Simili follows the VHDL'93 specification. Also note that this option merely enables the VHDL'87 FILE compatibility mode, which does not mean that the compiler checks for FULL VHDL'87 compliance.

Bit-string literals are treated strictly as bit_vector types. In VHDL'93, Bit-string literals (for example `b"101", x"fe"`) are treated as ordinary strings that can match any compatible array type (for example std_logic_vector).

`-h`

This option provides a short synopsis of the various options provided by the compiler.

`-refresh`

This option refereshes an existing pre-compiled library to make it current with the current version/build of the compiler or other pre-compiled libraries. It is useful if your library depends on another library (example: ieee) that was compiled with a different version of the compiler that yours. If you have the source code for the library, it is recommended that you re-compile the library instead of using this option to refresh the library. You will always be able to upgrade libraries to newer versions of the compiler/libraries (ie, you can upgrade your libraries that were compiled with version 2.1/2.2 to version 2.3). However, going backwards (from version 2.4 to 2.3) is not always guaranteed.

It is important to note that you “SHOULD NOT” use this option if the source VHDL files from which any of the libraries (your library or the library that you depend on) are have been modified.

You “MUST” refresh the libraries in order of ther dependence. For instance, if “library b” depends on “library b”, then “library b” must be refreshed before “library a” is refreshed. Please note that you “MUST NEVER” refresh libraries that are installed during the installation of the product.

`-strict`

This option forces the compiler into strict VHDL compliance. Currently, this option simply implies `-x` (see above).

`-maxerrors number`

By default, the compiler stops after having encountered 10 errors. This option allows you to control this threshold. Setting this value to zero implies infinite limit.

`-nowarnmsg name`

This option disables a specific warning message. The “name” is the identifier for the warning message (example CSVHE0053) or you can use the keyword ALL to disable all warnings.

`-vital2000`

By default, the compiler uses Vital95 (3.0) when vital specific packages are used in the VHDL source packages. Using this option, you can force the compiler to automatically substitute in the Vital 2000 packages (no explicit action is required for this to occur). The simulator itself supports the mixing of the Vital95 and Vital2000 packages in the same design. However, the compiler only allows Vital95 or Vital2000 on a given VHDL file.

10.4Exit code

The exit code is 0 if there were no errors and non-zero otherwise. This might be helpful for scripts/makefiles checking for a successful run.

Chapter

11

11 Simulator Reference (vhdle)

While Sonata can be used to interact with the simulator (**vhdle**) as well as control the various options for the simulator, this chapter serves as reference documentation for the actual simulator. This chapter is recommended for users who do not have access to Sonata, or users who intend to use **vhdle** in batch mode.

Once all the required models are compiled, the next task is to simulate. In VHDL, you must select a top-level entity (with an optional architecture) or a configuration to simulate. A process of elaboration precedes the process of simulation. Elaboration can be seen simply as a preparation step for simulation. In VHDL Simili, the process of elaboration and simulation occurs in one step using the command **vhdle** (elaboration does really happen but is transparent to the user).

vhdle is a batch program that runs without any user intervention. It has no mechanism (like some of the other simulators) to enter stimulus. We recommend the test-bench methodology for creating stimulus (and optionally to also check the results). You can use the full power of VHDL to create stimulus to drive the inputs, or even use VHDL file-IO to read/write test vectors from an external file. The simulation process automatically terminates when all activity ceases in the simulation model. VHDL Simili provides a facility for monitoring signals at any level in your design hierarchy using a command file. Please refer to the command file reference for more information.

11.1 Command line summary

vhdle supports a multitude of options. Most of these options are for advanced use. The following is a very simplified command summary

```
vhdle [options] top-level-design-unit
```

Assuming your top level entity name is myentity (in the work library), use the following command:

```
vhdle myentity
```

This will start the simulation of the myentity VHDL entity using the most recently compiled architecture for it. If you would like to simulate a particular architecture (for instance, myarch) of myentity, use the following command:

```
vhdle myentity(myarch)
```

You can also use a configuration as your top level. Assuming you have a configuration named myconfig, use the following command line:

```
vhdle myconfig
```

11.2Simulator options

The options listed below can help you control the behavior of the simulator. Certain options require arguments and certain arguments are optional. Optional items are specified using square brackets "[]" and a choice is separated by the character '|'.

`-h`

Prints a summary of all the available options with some examples.

`-s`

Enables silent mode. Most messages are suppressed.

`-work library-name`

You can use this option to override the default library (work). You will need this option if your top-level model is in a non-work library. For example, the following command:

```
vhdle -work mylib myentity
```

will allow you to simulate the model "myentity" from the library "mylib".

`-ini initialization-file-name`

You can use this option to override the default INI file name used to setup library associations. When this option is not specified, a search is done to find the file "symphony.ini" in the current working directory first and then in the installation directory. You can use this option to specify any file to be read as an INI file (as long as the file has the expected format). See Introduction for more information on the INI file. Hint: You can use the Sonata project file (.sws file) as an INI file since it uses the same format as the INI file. It is an error if this option is specified but file could not be opened for reading.

`-p`

This option will suppress the printing of progress messages to the screen during simulation. When this option is disabled, the current simulation time is not printed onto the screen (normally, this occurs at least once a second).

`-gName=Value`

Using this option, you can override values for generics for the top-level architecture/entity. Note that this option is invalid if your top-level design is a configuration. Using this option, you can override all scalar type generics (integers, enums, reals and physicals) and 1-dimensional character type arrays (such as bit_vector, std_logic_vector, string, etc.). Quotes are optional for arrays (double quotes) and enumerated (single quotes) types. Also, case is significant for the array and enumerated character literals. The generic name itself is treated in a case-insensitive manner. For numeric arguments, simple decimal constants are recommended although for integers and reals based literals are also allowed. For physical literals (such as time), only decimal literals can optionally precede the physical unit name and the physical unit name is required.

```
vhdle -gMyInt=32 -gMyTime=10.0ns -gMyReal=20.0 -gMyStr=hello myentity
vhdle -gMyStr=" this string has spaces " myentity
```

If the generic that is being set already has a default value (implicit or explicit), it will be never be evaluated and will be replaced by the one supplied by this option.

Note that spaces are not allowed unless they are enclosed in quotes.

```
-GName=Value
```

Using this option, you can override values for generics at any level of the hierarchy. This option is very similar to the '-g' option described above except that it can be used as a global override to override generics at all levels of the hierarchy.

`-do command-file`

This option is not supported while interfacing with Sonata *(only supported in batch mode)*.

Once the design is loaded into the simulator and elaborated, this command file specified by this option will be read. This option is primarily used to specify commands to monitor signals in your design and write the signal transitions to a simple text file. Please refer to the Command Reference Manual for more information about the syntax and features offered by such a command file. See examples below.

`-iterlimit n`

This option controls the maximum number of delta cycles allowed before time is advanced. It is an error if the simulation cannot advance even after the maximum number of iterations have been exhausted. Without this option, the iteration limit is set at 500.

`-list list-file`

This option is not supported while interfacing with Sonata *(only supported in batch mode)*.

This option is used in conjunction with the -do command and is ignored if no -do command is specified. This option specifies the name of the file that is used when writing out the signal transitions. If this command is not specified, the default file name is `"simili.lst"` in the current directory.

```
vhdle myentity -t 1us -do mycommands.cmd
vhdle myentity -do mycommands.cmd -list run1.lst
```

The first command simulates myentity for 1us, reads the commands specified in the file "mycommands.cmd" and creates the output file "simili.lst". The second command is similar to the first one except that the simulation will run until there is nothing to simulate and uses the file "run1.lst" as the output file. Note that the ".lst" extension is not special and you can choose any extension you would like.

`-t time[unit]`

Runs the simulation for the given amount of time. Normally, simulation automatically terminates when all simulation activity ceases. However, if your test bench runs forever, you may need to use this option to limit the amount of time simulation is run. Their valid time units are fs, ps, ns, us, ms, sec, hr and min. If unit is not specified, the default of fs is assumed. There must not be a space between the "time" and the "unit". Following are some examples of specifying a maximum simulation time.

```
vhdle -t 10ns myentity   -- Run for 10 nanoseconds
vhdle -t 100us myentity  -- Run for 100 microseconds
vhdle -t 100 us myentity -- !!!NOT VALID!!!
vhdle -t 100 myentity    -- Run for 100 femtoseconds
```

`-r time[unit]`

The format of this option is similar to the "`-t`" option above and can be used to restrict the simulation resolution. Simulation resolution can be restricted to any unit in multiples of 10. For example, 1ps, 10ps, 100ps, 1ns, 10ns, 100ns, 1us, 10us, etc. are all valid resultion limiters. The default resolution is 1ps. The resoulition can be set as low as 1fs. When performing gate level simkulations, you can significantly speed up your simulation by selecting a higher simulator resolution.

`-stdin filename`

Use this option to redirect the INPUT file defined in standard package TEXTIO to be any user-defined file. All read requests on the INPUT file are redirected to the file specified by "filename" (this file must exist).

`-stdout filename`

Use this option to redirect the OUTPUT file defined in standard package TEXTIO to be any user-defined file. All write requests on the OUTPUT file are redirected to the file specified by "filename". Note that the contents of the file specified by "filename" are overwritten.

`-sdf[min | typ | max] [InstancePath=]SdfFileName`

The -sdf option can be used to specify SDF timing data with a particular instance. The -sdfmin uses the minimum values, the -sdfmax option uses the maximum values and the -sdftyp option uses the typical values from the SDF file. The -sdf (without a min, typ, max suffix) option implies typical values. The optional "InstancePath" identifies the exact instance to which the SDF back-annotation values apply. If no InstancePath is given, or if the InstancePath of "/" is given, it implies the top level of the model. Note that you can use this option multiple times to back annotate data from several SDF files to the various instances of your model. The following applies the SDF file mysdf.sdf (typical values) to the instance "/u0/u1" of "myentity"

```
vhdle -sdf /u0/u1=mysdf.sdf myentity
```

The following is an example demonstrating the back-annotation of multiple SDF files.

```
vhdle -sdfmin /u0=musdf1.sdf -sdfmax /u1=mysdf2.sdf myentity
```

The following two examples are equivalent and the both apply to the top level "myentity".

```
vhdle -sdftyp /=mysdf.sdf myentity
vhdle -sdftyp mysdf.sdf myentity
```

Note: '/' represents a hierarchy separator

Note: Versions of SDF supported: 2.1, 3.0, 4.0

Note: Versions of Vital supported: Vital95, Vital 3.0, IEEE 1076.4-Oct-95, Vital2000

Note: The "InstancePath" is case sensitive -- for normal VHDL instance names, use lower case characters for instance path (unless you have extended identifiers in your path)

`-noaccel name`

To improve simulation performance, many of the industry standard VHDL packages have been accelerated. Using this option you can disable acceleration for a given type of VHDL package. The argument for -noaccel must be one of 1164, std_logic_arith, numeric_bit, numeric_std, vital or ALL. 'ALL' disables all package accelerations. 1164 disables acceleration for the IEEE std_logic_1164 package. std_logic_arith disables acceleration for synopsys libraries (std_logic_arith and related packages). You can specify this option multiple times to disable more than one package(s). **Note:** Disabling acceleration can lead to extremely slow simulations.

`-nowarn name`

This option relates to the -noaccel mentioned above. The numeric_bit and numeric_std packages offer a compile time constant that determines whether or not warnings should be printed under certain circumstances. By default, these warnings are enabled. When using the accelerated versions of these packages, disable these warnings. This option has no effect if you are using un-accelerated versions of these packages. The 'name' has to be one of numeric_bit, numeric_std, std_logic_arith or ALL. You can specify this option multiple times. Note that disabling these warnings may hide important bugs in your design. In a lot of cases, you can eliminate these warnings by simply initializing your variables/signals. The -nowarn option does not have any effect if acceleration has been disabled for the package.

`-nowarnmsg name`

This option disables a specific warning message. The “name” is the identifier for the warning message (example CSVHE0053) or you can use the keyword ALL to disable all warnings.

`-nocycleopt`

VHDL Simili uses some special optimizations to accelerate simulations. However, under rare circumstances, simulation results may differ from other VHDL simulators with respect to the exact delta cycle in which a given signal is updated. You can disable this kind of optimization using this option. We recommend not using this option unless you are very particular as to the exact delta cycle at which your signals are updated.

`-noifopt`

This option can be used for debugging purposes or get a more accurate code-coverage analysis. It prevents if-then-else statements from getting optimized out into simpler constructs.

`-uselicense featurename`

Without this option, vhdle first tries to checkout "vhdlepro", then tries "vhdle" and then finally "vhdlefree". Using this option, vhdle can be forced to checkout a specific license feature.

`-breakon severity`

'Severity' can be NOTE, WARNING, ERROR or FAILURE. By default, the simulation stops when a FAILURE (via an ASSERT or a REPORT statement) is triggered during simulation. However, there might be instances where you might want to stop the simulation for ASSERT/REPORTs of a lower severity. Use this option to make the simulation stop at a different severity level.

11.3Code coverage options

The following `vhdle` options relate to code-coverage

`-coverage filename.scv`

When this option is used, code-coverage is enabled and the results of the coverage are written to the specified filename. If the specified file does not exist, then the coverage database is initialized using either the file `"<install-dir>/bin/coverage.ini"` or the file specified in the –`coverageini` option (see below). This is only done once. While you can use any extention for the coverage database, the ".scv" extension is recommended for ease of use within Sonata.

`-coverageini filename.ini`

This option is used only during the initialization of the coverage database. It is ignored when the specified coverage database already exists. Use this option to override the default one in `"<install-dir>/bin/coverage.ini". See the default file for documention on the format of the initialization file.`

11.4Exit code

The exit code is 0 if there were no errors and non-zero otherwise. This might be helpful for scripts/makefiles checking for a successful run.