HDL Translation With X-HDL

For X-HDL 4.1.4

July 5, 2010

Contents

Figures

Introduction

X-HDL 4 is the premier VHDL <=> Verilog translator. Unlike other translators, X-HDL performs intelligent translation of your HDL code, not just syntax conversion. Along with top-notch translation abilities, X-HDL helps you do the entire translation job by providing user-definable translations, an integrated design hierarchy browser to evaluate design organization and a built-in link checker to validate code structure and form.

X-HDL is supported on the most popular engineering design platforms including:

* Sun Sparc Solaris 8 and 10
* X86 Linux
* MS Windows 2000/XP/Vista/7

Initial setup

Upon initial startup after installation, X-HDL will invoke the License Wizard. The License Wizard can be used to request an evaluation license for X-HDL, install a new license received from X-Tek Corporation for X-HDL or just to continue running without a license, known as Demo mode. Demo mode is fully functional but restricts the source file size to 1K bytes.

To request an evaluation license, simply select the "Request Evaluation License" option on the Wizards first page and select 'Next'. On the second page, fill out the requested information and select 'Next'. A license request will be automatically sent to X-Tek Corporation for processing. Please allow up to 24 hours for a license to be sent to you via e-mail.

To install a license received from X-Tek via e-mail, first, save the license which is attached to the e-mail to a location within your filesystem. Then, from the License Wizard select "Install a new license" and select 'Next'. On the second page enter the full pathname to the license file and select 'Next'. The license will be automatically installed into the license server.

If X-HDL starts up and finds a valid license, then the License Wizard will not be invoked and the main application will be started.

For installation of a floating -license, please see the XLM User's manual.

X-HDL Overview

X-HDL 4 is the premier Verilog <=> VHDL bi-directional translator. X-HDL performs translation of even the most complex RTL/gate-level code efficiently and requiring few, if any, "hand tweaks" of the translated code. X-HDL also contains specialized algorithms which are very effective in translating behavioral-level code to functionally equivalent target-language code.

Key Features

- Provides both GUI and command-line modes
- Performs automatic hierarchical translations as well as file-at-time translations.
- Translates structural, RTL and behavioral code
- Preserves comments with placement nearly identical to the source
- Consistent code formatting with user customizations
  - VHDL'87 or VHDL'93 syntax generation
  - Verilog or Verilog-2001 syntax generation
  - Code alignment controls
  - Indentation controls
  - Line wrap controls
- Supports component libraries
- Smart overloaded subprogram handling for VHDL
- Intelligently determines if translated Verilog tasks/functions are local or global within the VHDL.
- VHDL conversion function filtering
- Conversion definitions to support user-defined translation
- Support for pre- and post-processing scripts to enable user-specific translation needs.

The following sections describe in detail the capabilities of the Verilog-to-VHDL and VHDL-to-Verilog features within X-HDL.

Verilog-to-VHDL Translation

X-HDL provides push-button translation of RTL and gate-level Verilog code to VHDL. Except for user-defined primitives, the entire language is parsed by X-HDL, but not all constructs can be translated to VHDL. The following table provides an overview of the supported constructs. Note that some constructs may be more or less translatable depending on their context and use within the source code.

Component Instantiations	
Gate Primitives	
Wires/Regs/Arrays	
Parameters	
Always blocks	
Initial blocks	
If/Case statements	
Looping statements	
Continuous assign statements	
Blocking/non-blocking assignments	

Compiler directives	✓
Tasks/Functions	✓
System Tasks/Functions	✓
LHS concatenation	✓
Delays	✓
System timing checks	✓
Verilog-2001 Generates	✓
Verilog-2001 Ansi-ports	✓
Verilog-2001 indexed part selects	✓
Verilog-2001 multi-dimensional arrays	✓
Verilog-2001 sensitivity list options	✓
Verilog-2001 parameter passing by name	✓
Verilog-2001 Initial values	✗
Fork/join blocks	✗
Force/disable/release	✗
Hierarchical identifiers	✗
User-defined Primitives	✗
File I/O	✗

Table 1 - Verilog-to-VHDL supported constructs

The Verilog-to-VHDL translator is also extensible allowing the user to extend and customize the translation process. X-HDL accepts user-defined conversion definitions which allow the user to change names, modify function and task calls, and override the translation of the standard operators. Furthermore, the user can define pre- and post-processing scripts which will automatically be executed during the translation. This provides the user with near complete control over the translation process.

VHDL-to-Verilog Translation

X-HDL provides push-button translation of RTL and gate-level VHDL code to Verilog. Although the entire language is parsed by X-HDL, not all constructs can be translated to Verilog. The following table provides an overview of the supported constructs. Note that some constructs may be more or less translatable depending on their context and use within the source code.

Component Instantiations	✓
Signals/Variables/Constants	✓
Multi-dimensional arrays	✓
Generics	✓
Processes	✓
If/case statements	✓
Loop statements	✓
Continuous assign statements	✓
Generates	✓
Report	✓

Procedures/Functions	✓
Overloaded functions	✓
Types/subtypes	✓
Records	✓
Delays	✓
Packages	✓
VHDL'87 & VHDL'93	✓
Physical types	✗
File I/O	✗
Allocators	✗
Array ports	✗

Table 2 - VHDL-to-Verilog supported constructs

The VHDL-to-Verilog translator is also extensible allowing the user to extend and customize the translation process. X-HDL accepts user-defined conversion definitions which allow the user to change names, modify function and procedure calls, and override the translation of the standard operators. Furthermore, the user can define pre- and post-processing scripts which will automatically be executed during the translation. This provides the user with near complete control over the translation process.

Hierarchical Browser

The X-HDL hierarchical browser displays a hierarchy tree of a VHDL or Verilog design. The browser also can optionally display the filename and path of each element of the design.

To browse a VHDL design, first ensure that the VHDL->Verilog translation tab is selected in the Main window. Then, invoke the hierarchical browser from the file menu (File->Browse Hierarchy). A directory selection box will appear. Navigate to the desired VHDL directory tree and select 'Ok'. X-HDL will recursively process the files from the selected directory on down.

To browse a Verilog design, first ensure that the Verilog->VHDL translation tab is selected in the Main window. Then, invoke the hierarchical browser from the file menu (File->Browse Hierarchy). A directory selection box will appear. Navigate to the desired Verilog directory tree and select 'Ok'. X-HDL will recursively process the files from the selected directory on down.

Once the design files are processed, the Hierarchy Window will appear, showing the design hierarchy. Any element of the design may be selected for translation by double-clicking on it. The associated file will be entered in the Source entry box of the translation window.

GUI Mode

X-HDL's graphical user interface provides an easy to learn, consistent interface for the translation process. Graphical user interface mode is entered when X-HDL is invoked by the 'xhdl3' command from the Unix command line or from the 'xhdl3' icon from the Windows interface.

When invoked, the GUI begins with the main menu as shown below:

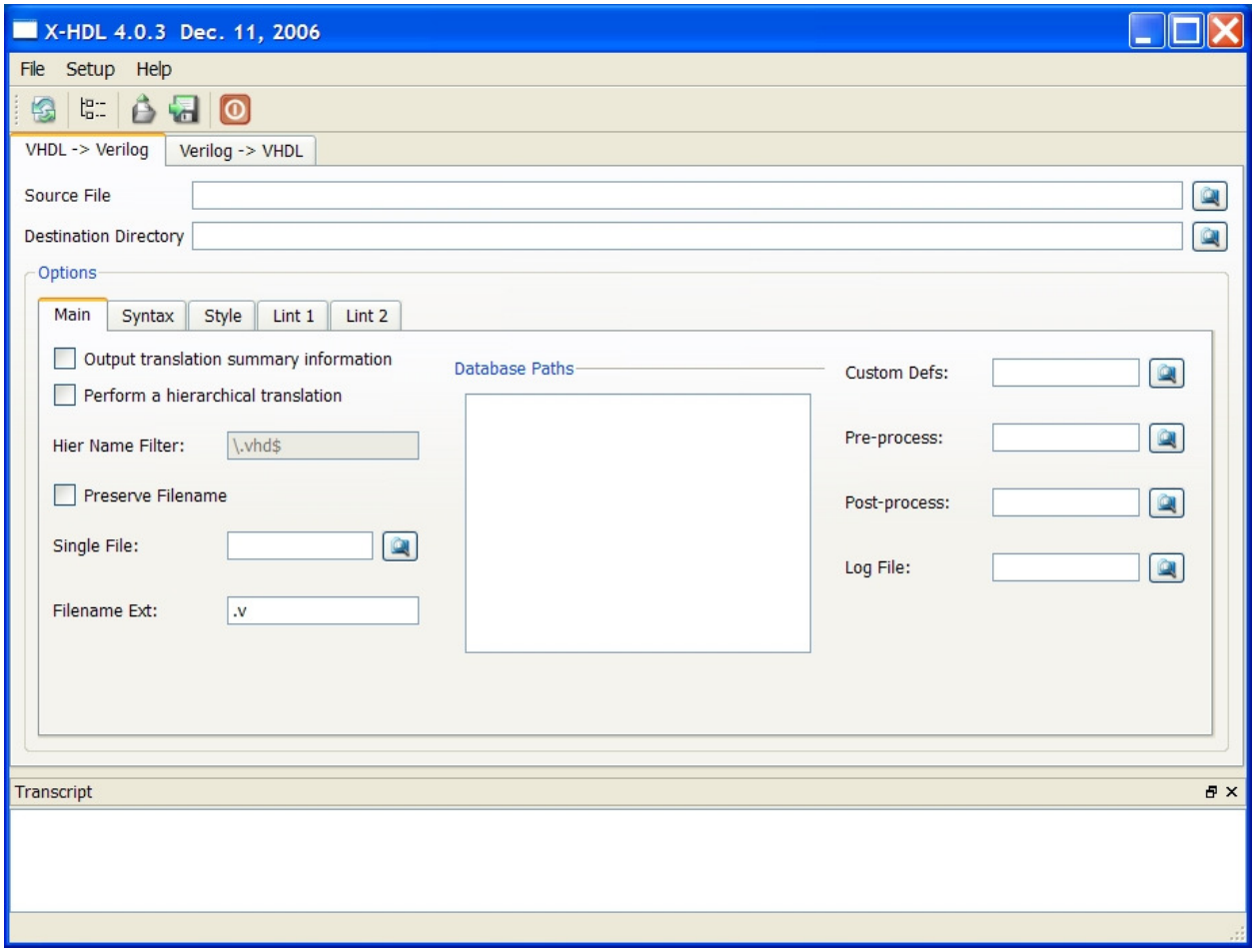

Figure 1 - X-HDL main window

The main GUI window contains 3 pull-down menus (File, Setup and Help), a toolbar, a tabbed window containing pages of configuration controls for each translator and a log window at the bottom. The log window displays any messages generated during the translation process.

File Menu

The File Menu is invoked by the selecting the ‘File’ option in the menu toolbar.

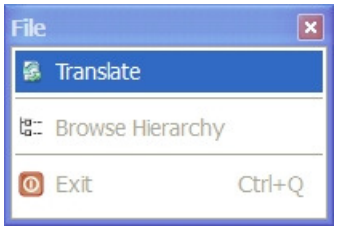

Figure 2 - X-HDL file menu

The File Menu provides controls to perform various operations. These operations are:

Translate: Translate the selected source file into the selected destination directory using the specified translation configuration.

Browse Hierarchy: Invoke the hierarchy browser. This browser is used to view the hierarchy of a design and to select files to be translated.

Exit: Exit the application. All window positions will be saved and used to setup the application on the next start up.

Setup Menu

The Setup Menu is invoked by the selecting the 'Setup' option in the menu toolbar.

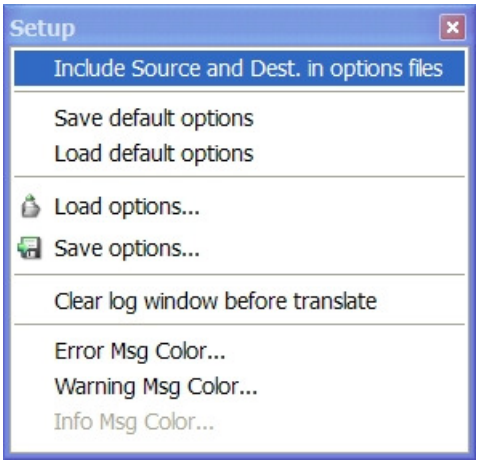

Figure 3 - X-HDL setup menu

The Setup Menu provides controls to save and load translation configurations and to configure the log window. The setup menu controls are:

Include Source and Dest. in options files: Normally when X-HDL saves option information it does not save the source file to be translated nor the destination directory. Enabling this control directs X-HDL to save that information along with the other translation options.

Save default options: This control directs X-HDL to save the current configuration information into a standard location options file (the location depends on the particular platform).

Load default options: This control directs X-HDL to load the configuration information from a standard location options file into the application (the location depends on the particular platform).

Load options: This control directs X-HDL to load the configuration information from a user-specified options file into the application.

Save options: This control directs X-HDL to save the current configuration information into a user-specified options file.

Clear log window before translate: X-HDL directs all translation output into the log window located at the bottom of the application window. The control directs X-HDL to clear the window contents prior to any translation.

Error Msg Color: Error messages are color coded within the X-HDL log window. This control provides the user the ability to select the color of the error messages. The default color is red.

Warning Msg Color: Warning messages are color coded within the X-HDL log window. This control provides the user the ability to select the color of the warning messages. The default color is dark yellow.

Info Msg Color: Informational messages are color coded within the X-HDL log window. This control provides the user the ability to select the color of the informational messages. The default color is black.

Help Menu

The Help Menu is invoked by the selecting the 'Help' option in the menu toolbar.

Figure 4 - X-HDL help menu

The Help Menu provides controls to access various help information for the X-HDL application. The help menu selections are:

About: This control invokes a dialog box which displays X-HDL version and copyright information

Host ID: This control invokes a dialog box which displays the Host ID of the local machine. This Host ID is needed for licensing operations.

License info: This control invokes a dialog box which displays the current license issued to the application. If no license is available, the application runs in demo mode.

User's manual: This control invokes a dialog box which displays the location of the X-HDL documentation file.

Verilog-to-VHDL Configuration

To begin a Verilog-to-VHDL translation, select the Verilog->VHDL tab in the main X-HDL window.

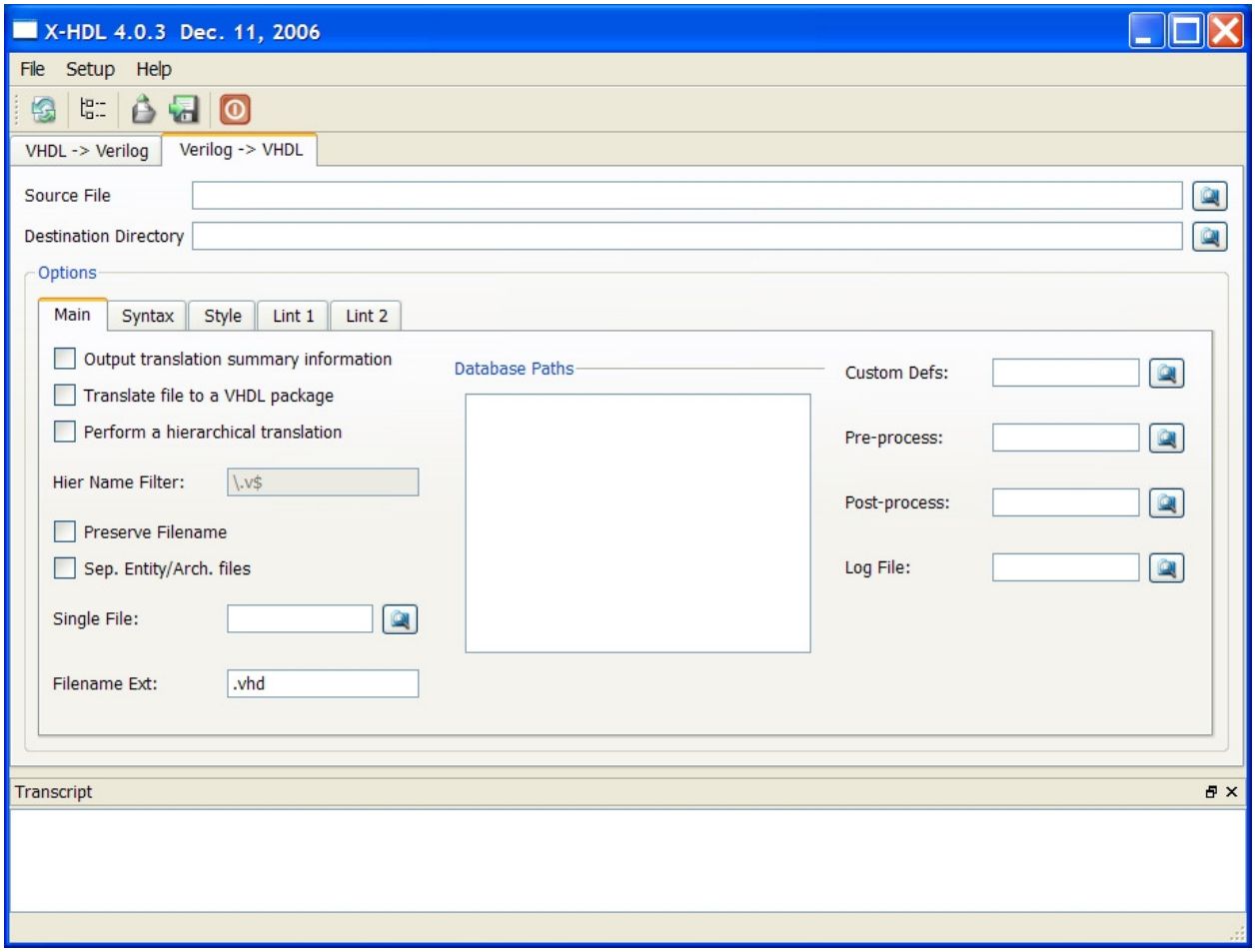

Figure 5 - Verilog-to-VHDL translation page

The next step is to select the source file to translate and the destination directory. These can be entered manually in the associated entry boxes in the translation page, or can be selected using the file-find button on the right side of the entry box.

Finally, options can be set to control the translation, then selecting File->Translate will perform the translation.

The Verilog-to-VHDL translator contains a myriad of options and controls. These are detailed in the following sections.

Main Options Page

The main options page selects various file output, database search paths and customization options for the translation.

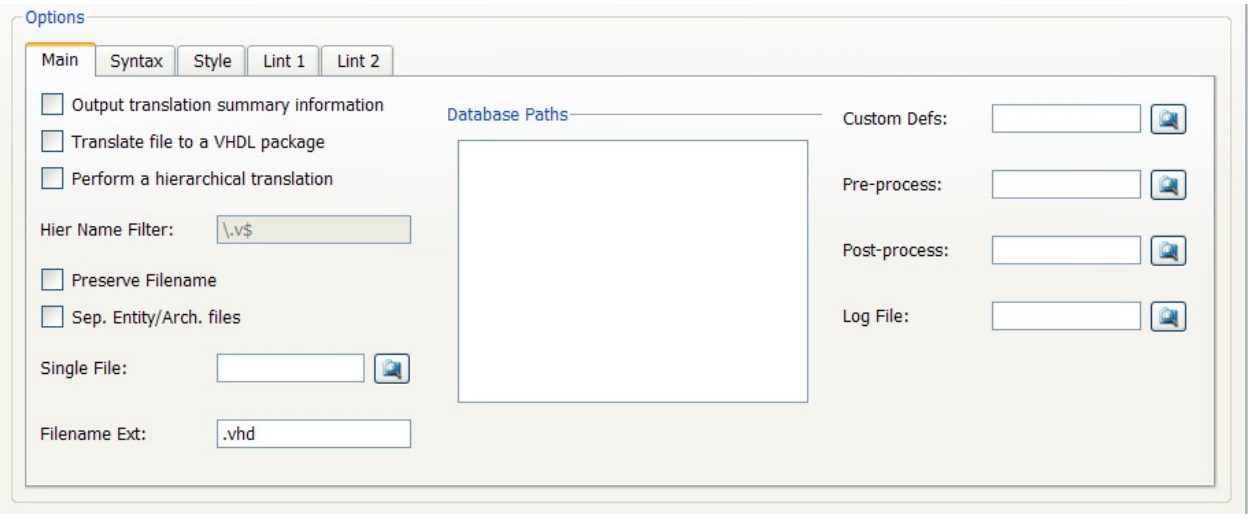

Figure 6 - Verilog-to-VHDL main options page

The main options page contains the following controls:

Output translation summary information: When this control is enabled, X-HDL will output translation summary information to the log window. Information provided in the summary includes:

- filename
- Number of errors
- Number of warnings
- Lines of code
- Number of input/output/bidir ports
- Number of parameters/regs/wires/events/integers/reals/times
- Number of concurrent statements
- Number of sequential statements
- Number of tasks/functions
- Number of instances
- Number of clock domains

Translate file to a VHDL package: Some Verilog files which are `include-d into other Verilog source files are good candidates to be translated to VHDL packages. This control enables translating a partial Verilog source file into a VHDL package. The partial Verilog file must contain only parameter declarations and subprograms. Other Verilog constructs are not appropriate for VHDL packages. If this control is not enabled, the `include-d code will be placed in-line at the point of the `include statement and translated.

Perform a hierarchical translation: When this control is enabled, X-HDL will automatically recurse through the directory specified in the Source entry and translate all Verilog source files matching the specified hierarchical filter (see Hier. Name Filter) in a botton-up order. All translated files will be placed in the location specified by the Destination directory.

Hier. Name Filter: When performing a hierarchical translation, the value entered in this control will be applied to all filenames in the searched directories. Files matching the filter will be translated. The default filter is '\.v$'. See the section on Regular Expression Formats for information on valid filter specifications.

Preserve Filename: By default, X-HDL creates the translated filename using the module name within the source code. This option causes X-HDL to use the source filename (with different extension) for the translated code.

Sep. Entity/Arch. files: Specifies that X-HDL should generate separate entity/architecture files. By default, X-HDL places both the entity and architecture into the same file

Single File: Specifies that X-HDL should output all translated code to a single file. The filename will be the value entered into this control.

Filename ext: X-HDL will use the value in this control as the translated filename extension. The default extension is '.vhd'.

Database paths: When X-HDL instantiates a component, it requires information about the translated component name, port names, etc. X-HDL stores this information in a textual database during translation of the component. The entries in this control are used to direct X-HDL to the location of the translated component information. To enter a new path, right click on this control and select 'Add' from the popup menu. A database entry dialog will be invoked:

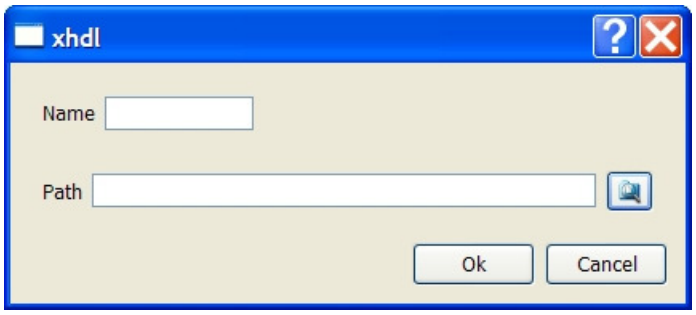

Figure 7 - Verilog-to-VHDL database path dialog

In the Name control, enter the name of the 'library' of components. In the Path control, enter the path to the translated components.

Database paths can also be edited or deleted by selecting the path in the control and right clicking the mouse. On the popup menu, select 'Edit' or "Delete', respectively.

Custom defs: X-HDL provides the capability to define custom translation of identifier names, including functions and tasks, and operators. The definitions are specified in the file entered in this control. The file format is straightforward. Each line of the file specifies a from/to translation customization separated by '=>'. For example:

```
vlog_id => vhdl_id;
+ => add($1, $2);
func => newfunc($1);
```

would change all occurrences of 'vlog_id' to 'vhdl_id', + to a function call to add (note: $1 and $2 represent the original parameters from the + operation) and change the function call 'func' to 'newfunc' using only the first parameter from the original 'func' call.

Note: Future version of X-HDL will provide custom translation of functions and tasks based on the paramter types of the function/task calls.

Pre-process: Specifies that the source code should be processed through the executable program entered in this control before being translated. This provides the capability to massage the source code prior to translation if necessary. The executable program must accept stdin data and provide output on stdout.

Post-process: Specifies that the translated code should be processed through the executable program entered in this control before being written to the output file. This provides the capability to further customize the translated output if necessary. The executable program must accept stdin data and provide output on stdout.

Log file: Specifies that X-HDL should output messages to the file entered in this control.

Syntax Options Page

The syntax options page provides controls for the user to select various VHDL syntax options.

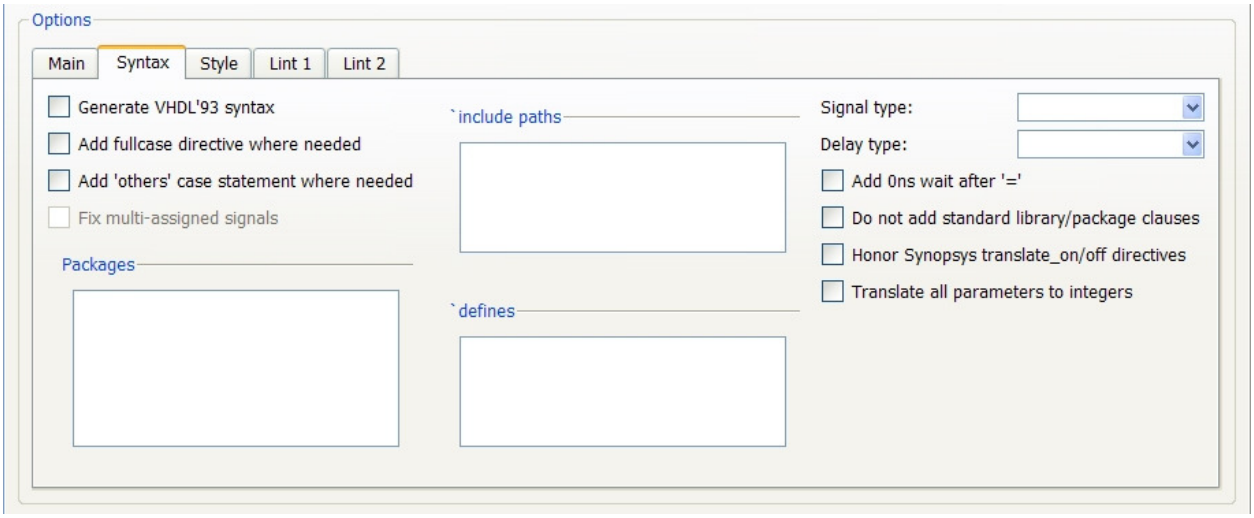

Figure 8 - Verilog-to-VHDL syntax options page

The syntax options page contains the following controls:

Generate VHDL'93 syntax: Specifies that X-HDL should generate VHDL'93 syntax. By default, X-HDL uses VHDL'87 syntax.

Add fullcase directive where needed: Specifies that X-HDL should add a 'fullcase' synthesis comment to any case statement that doesn't have one

Add 'others' case statement where needed: Specifies that X-HDL should add an 'others' case to any case statement that doesn't have a default.

Fix multi-assigned signals: Specifies that X-HDL should fix signals which are assigned in multiple procedural blocks, which causes multiple driver conflicts in VHDL. **Currently not implemented**.

Packages: This control contains a list of packages that should be added to the VHDL output code. To enter a new package, right click on this control and select 'Add' from the popup menu. A package entry dialog will be invoked:

Figure 9 - Verilog-to-VHDL package add dialog

Simply enter the package to be included, such as: mylib.mypkg.all, and select ‘Ok’. Packages can also be edited or deleted by selecting the package in the control and right clicking the mouse. On the popup menu, select ‘Edit’ or “Delete’, respectively.

***`include paths:*** This control contains a list of paths to search for `include files. To enter a new path, right click on this control and select ‘Add’ from the popup menu. A path entry dialog will be invoked:

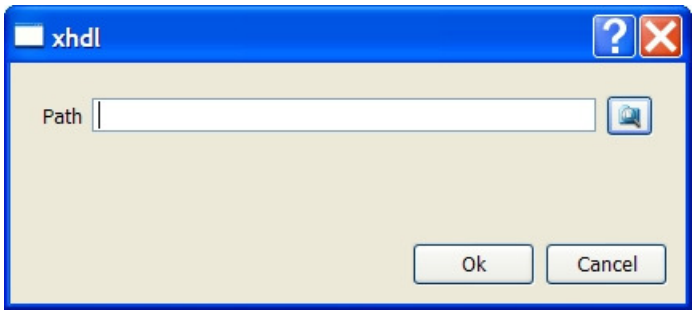

Figure 10 - Verilog-to-VHDL include path add dialog

Simply enter the path to be added and select ‘Ok’. Paths can also be edited or deleted by selecting the path in the control and right clicking the mouse. On the popup menu, select ‘Edit’ or “Delete’, respectively.

***`defines:*** This control contains a list of symbols to be defined in the Verilog context. This is equivalent to having a `define statement in the Verilog source code. To add a new define, right click on this control and select ‘Add’ from the popup menu. A define entry dialog will be invoked:

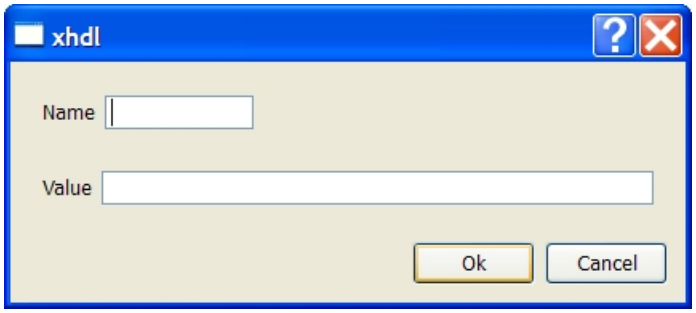

Figure 11 - Verilog-to-VHDL define add dialog

Simply enter the name to be defined and, optionally, the value to be associated with the name, then select ‘Ok’. Defines can also be edited or deleted by selecting the define in the control and right clicking the mouse. On the popup menu, select ‘Edit’ or “Delete’, respectively.

Signal type: This control selects the VHDL type that should be used for the translation of all Verilog regs and wires. The available types are:

bit
std_logic
std_ulogic

The default type is ‘std_logic’

Delay type: This control selects the delay value in a Verilog triplet delay specification to be used in the translation. The available delay values are:

min

typ
max

The default delay is ‘typ’

Add 0ns wait after ‘=’: This control specifies that X-HDL should add a 'wait for 0 ns' after each blocking ('=') assignment to emulate the operation of the blocking assignment in VHDL. The default is to convert blocking assignments to VHDL non-blocking assignments

Do not add standard library/package clauses: By default, X-HDL automatically adds standard VHDL package/library clauses to the translated output. When enabled, this control specifies that these clauses should not be automatically added

Honor Synopsys translate_on/translate_off directives: This control directs X-HDL to honor Synopsys translate_on and translate_off directives. Code within this these directives is not translated nor contained in the output code. **Currently not implemented.**

Translate all parameters to integers: By default, X-HDL translated unsized parameters to VHDL integers and sized parameters to VHDL vectors. This control directs X-HDL to translate all parameters to VHDL integers regardless whether they are sized or not.

Translate stardard gates to comb. logic: (not shown in figure 8). By default, X-HDL preserves the hierarchical structure of the code. This control directs X-HDL to translate standard Verilog gate instances (and, or, nor, etc.) into combinational logic.

Style Options Page

The style options page provides controls for the user to select various VHDL code style options.

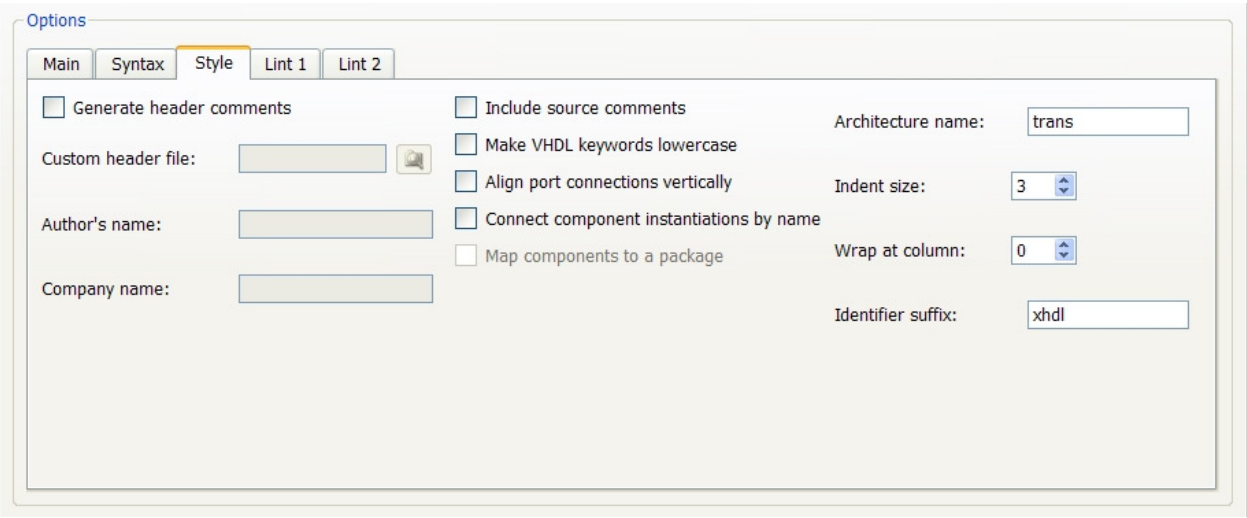

Figure 12 - Verilog-to-VHDL style options page

The syntax options page contains the following controls:

Generate header comments: Enables the generation of header comments. If the Custom header file control is not enabled (see below), a default header will be used.

Custom header file: When header comments are included in the translated output X-HDL will use the contents of file specified in this control for the header comments. The header comments file is free form, however, the following specifiers may be used to direct X-HDL to auto-insert system information:

%a	: author's name (see Author's name, below)
%c	: company name (see Company name, below)
%d	: current date
%f	: source filename
%n	: module name
%t	: current time
%v	: X-HDL version
%V	: X-HDL version date

Author's name: When the header comments are enabled, the author's name is set to value entered in this control. The default author's name is ''.

Company name: When the header comments are enabled, the company name is set to <name>. The default company name is ''.

Include source comments: When enabled, X-HDL will include source comments in the translation output. Due to necessary relocation of code segments (such as all declarations moved to the VHDL declaration section), the position of some comments may shift slightly.

Make VHDL keywords lowercase: This control specifies that X-HDL should make all VHDL keywords lowercase. The default is uppercase keywords.

Align port connections vertically: This control specifies that port connections within instantiations should be aligned vertically in the translated code.

Connect component instantiations by name: This control specifies that X-HDL should connect component instantiations by name regardless of how the component is connected within the source. This option requires that X-HDL be able to locate the translation information for the instantiated component. By default X-HDL will instantiate components using the same style as in the source code.

Map components to a package: This control specifies that component declarations should be mapped to a package rather than in the declaration section of the VHDL architecture. **Currently not implemented**.

Architecture name: The value entered in this control will be used for the architecture name. The default architecture name is 'trans'.

Indent size: This control specifies the indentation size for the translated code. The default size is 3.

Wrap at column: This control specifies that X-HDL should wrap long lines at the selected column or less. By default X-HDL does not wrap long lines (value of zero).

Identifier suffix: The value entered in this control is used by X-HDL as the suffix that is appended to identifers in the source code which are invalid in VHDL, thereby making the identifiers legal in VHDL. The default suffix is 'xhdl'.

Lint Options Page 1

The lint options page 1 provides controls for the user to select various HDL structural checks.

Figure 13 - Verilog-to-VHDL lint options page 1

Many of the lint controls are not currently implemented, as noted below. The lint options page 1 contains the following controls:

Assignment size check: This control enables assignment size checking. X-HDL will issue a warning if the size of the LHS of an assignment does not match the size of the RHS.

Case option check: This control enables evaluation of case options to ensure that the options are unique and complete. **Currently not implemented**.

Implied latch check: When this control is enabled, X-HDL will issue a warning if a latch structure is detected. **Currently not implemented**.

Combinational loop check: When this control is enabled, X-HDL will issue a warning if a combinational loop is detected. **Currently not implemented.**

Gated clock check: When this control is enabled, X-HDL will issue a warning if a gated clock structure is detected. **Currently not implemented.**

Non-reset FF check: When this control is enabled, X-HDL will issue a warning if it detects a flip-flop structure that does not contain a reset. **Currently not implemented**.

Unused signal check: When this control is enabled, X-HDL will issue a warning for all signals which are defined but not referenced.

Undriven signal check: When this control is enabled, X-HDL will issue a warning for all signals which are defined but undriven.

I/O port check: When this control is enabled, X-HDL will issue a warning if a module does not contain input and output ports.

X/Z state check: When this control is enabled, X-HDL will issue a warning whenever an X or Z constant is detected within the source code.

Nested conditions level: This control enables nested conditions level checking (if and case). If the number of nested condition blocks exceeds the value entered in the control, then X-HDL will issue a warning. A value of 0 for allows

infinite nesting (i.e. effectively disables nested condition checking). The default value for nested condition checking is 0.

Nested blocks level: This control enables nested blocks level checking. If the number of nested blocks exceeds the value entered in the control, then X-HDL will issue a warning. A value of 0 allows infinite nesting (i.e. effectively disables nested block checking). The default value for nested block checking is 0.

Lint Options Page 2

The lint options page 2 provides controls for the user to select various Verilog identifier format checks.

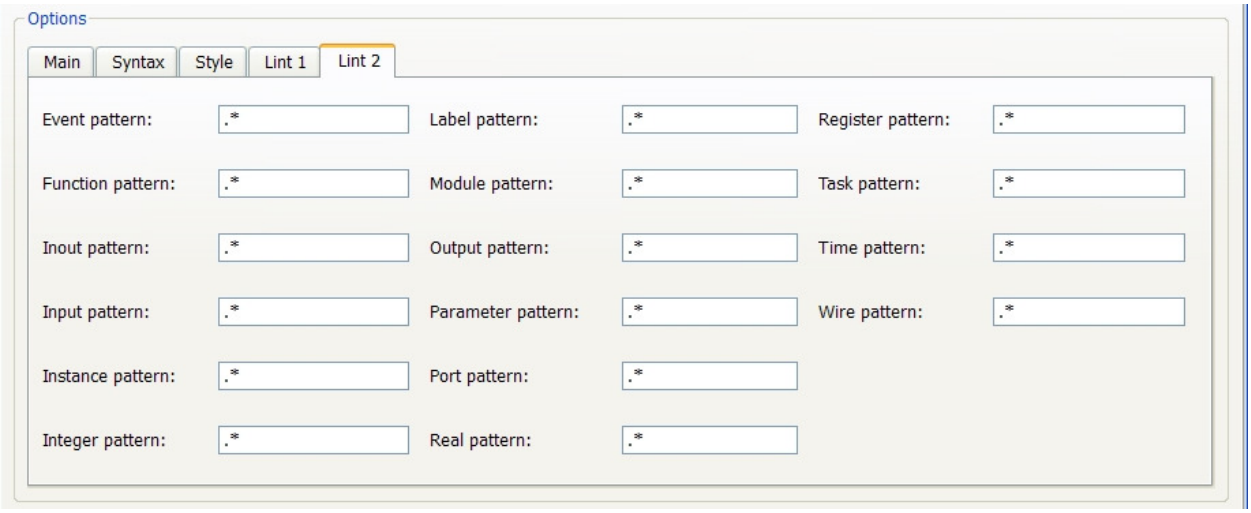

Figure 14 - Verilog-to-VHDL lint options page 2

All identifier formats are specified using regular expression patterns. Please see the Regular Expression Format section for more information on regular expressions used by X-HDL. The lint options page 2 contains the following controls:

Event pattern: Sets the event name check pattern. X-HDL will issue a warning if an event identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Function pattern: Sets the function name check pattern. X-HDL will issue a warning if a function identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Inout pattern: Sets the inout name check pattern. X-HDL will issue a warning if an inout identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Input pattern: Sets the input name check pattern. X-HDL will issue a warning if an input identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Instance pattern: Sets the instance name check pattern. X-HDL will issue a warning if an instance identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Integer pattern: Sets the integer name check pattern. X-HDL will issue a warning if an integer identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Label pattern: Sets the label name check pattern. X-HDL will issue a warning if a label identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Module pattern: Sets the module name check pattern. X-HDL will issue a warning if a module identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Output pattern: Sets the output name check pattern. X-HDL will issue a warning if an output identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Parameter pattern: Sets the parameter name check pattern. X-HDL will issue a warning if a parameter identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Port pattern: Sets the port name check pattern. X-HDL will issue a warning if a port (input, inout, output) identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Real pattern: Sets the real name check pattern. X-HDL will issue a warning if a real identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Register pattern: Sets the reg name check pattern. X-HDL will issue a warning if a reg identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Task pattern: Sets the task name check pattern. X-HDL will issue a warning if a task identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Time pattern: Sets the time name check pattern. X-HDL will issue a warning if a time identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Wire pattern: Sets the wire name check pattern. X-HDL will issue a warning if a wire identifier in the source code does not match the specified pattern. The default pattern is '.*'.

VHDL-to-Verilog Configuration

To begin a VHDL-to-Verilog translation, select the VHDL->Verilog tab in the main X-HDL window.

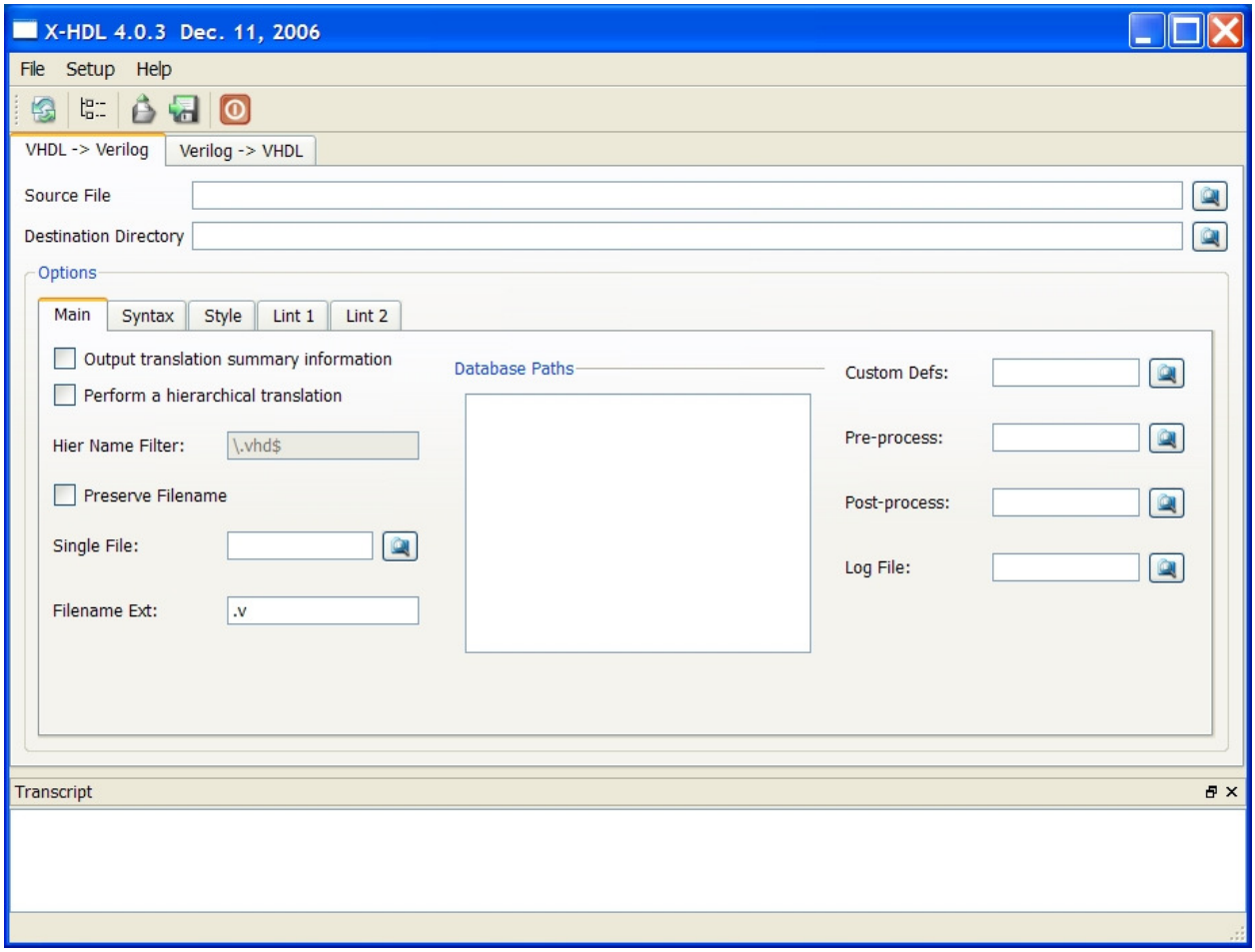

Figure 15 - VHDL-to-Verilog translation page

The next step is to select the source file to translate and the destination directory. These can be entered manually in the associated entry boxes in the translation page, or can be selected using the file-find button on the right side of the entry box.

Finally, options can be set to control the translation, then selecting File->Translate will perform the translation.

The VHDL-to-Verilog translator contains a myriad of options and controls. These are detailed in the following sections.

Main Options Page

The main options page selects various file output, database search paths and customization options for the translation.

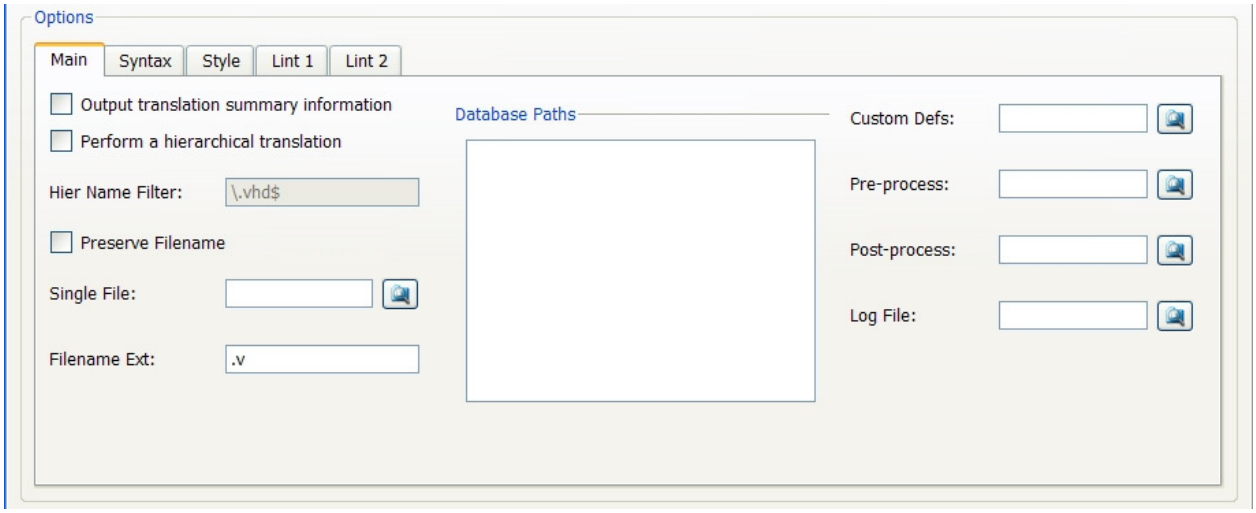

Figure 16 - VHDL-to-Verilog main options page

The main options page contains the following controls:

Output translation summary information: When this control is enabled, X-HDL will output translation summary information to the log window. Information provided in the summary includes:

- filename
- Number of errors
- Number of warnings
- Lines of code
- Number of input/output/bidir ports
- Number of parameters/regs/wires/events/integers/reals/times
- Number of concurrent statements
- Number of sequential statements
- Number of tasks/functions
- Number of instances
- Number of clock domains

Perform a hierarchical translation: When this control is enabled, X-HDL will automatically recurse through the directory specified in the Source entry and translate all Verilog source files matching the specified hierarchical filter (see Hier. Name Filter) in a botton-up order. All translated files will be placed in the location specified by the Destination directory.

Hier. Name Filter: When performing a hierarchical translation, the value entered in this control will be applied to all filenames in the searched directories. Files matching the filter will be translated. The default filter is '\.v$'. See the section on Regular Expression Formats for information on valid filter specifications.

Preserve Filename: By default, X-HDL creates the translated filename using the module name within the source code. This option causes X-HDL to use the source filename (with different extension) for the translated code.

Single File: Specifies that X-HDL should output all translated code to a single file. The filename will be the value entered into this control.

Filename ext: X-HDL will use the value in this control as the translated filename extension. The default extension is '.v'.

Database paths: When X-HDL instantiates a component, it requires information about the translated component name, port names, etc. X-HDL stores this information in a textual database during translation of the component. The entries in this control are used to direct X-HDL to the location of the translated component information. To enter a new path, right click on this control and select 'Add' from the popup menu. A database entry dialog will be invoked:

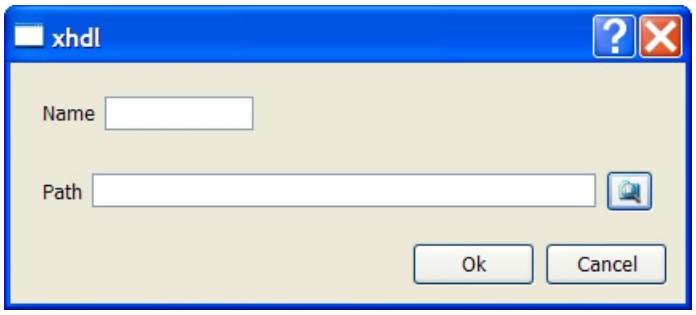

Figure 17 - VHDL-to-Verilog database path add dialog

In the Name control, enter the name of the 'library' of components. In the Path control, enter the path to the translated components.

Database paths can also be edited or deleted by selecting the path in the control and right clicking the mouse. On the popup menu, select 'Edit' or "Delete', respectively.

Custom defs: X-HDL provides the capability to define custom translation of identifier names, including functions and tasks, and operators. The definitions are specified in the file entered in this control. The file format is straightforward. Each line of the file specifies a from/to translation customization separated by '=>'. For example:

```
vhdl_id => vlog_id;
+ => add($1, $2);
func => newfunc($1);
```

would change all occurrences of 'vhdl_id' to 'vlog_id', + to a function call to add (note: $1 and $2 represent the original parameters from the + operation) and change the function call 'func' to 'newfunc' using only the first parameter from the original 'func' call.

Note: Future version of X-HDL will provide custom translation of functions and tasks based on the paramter types of the function/task calls.

Pre-process: Specifies that the source code should be processed through the executable program entered in this control before being translated. This provides the capability to massage the source code prior to translation if necessary. The executable program must accept stdin data and provide output on stdout.

Post-process: Specifies that the translated code should be processed through the executable program entered in this control before being written to the output file. This provides the capability to further customize the translated output if necessary. The executable program must accept stdin data and provide output on stdout.

Log file: Specifies that X-HDL should output messages to the file entered in this control.

Syntax Options Page

The syntax options page provides controls for the user to select various Verilog syntax options.

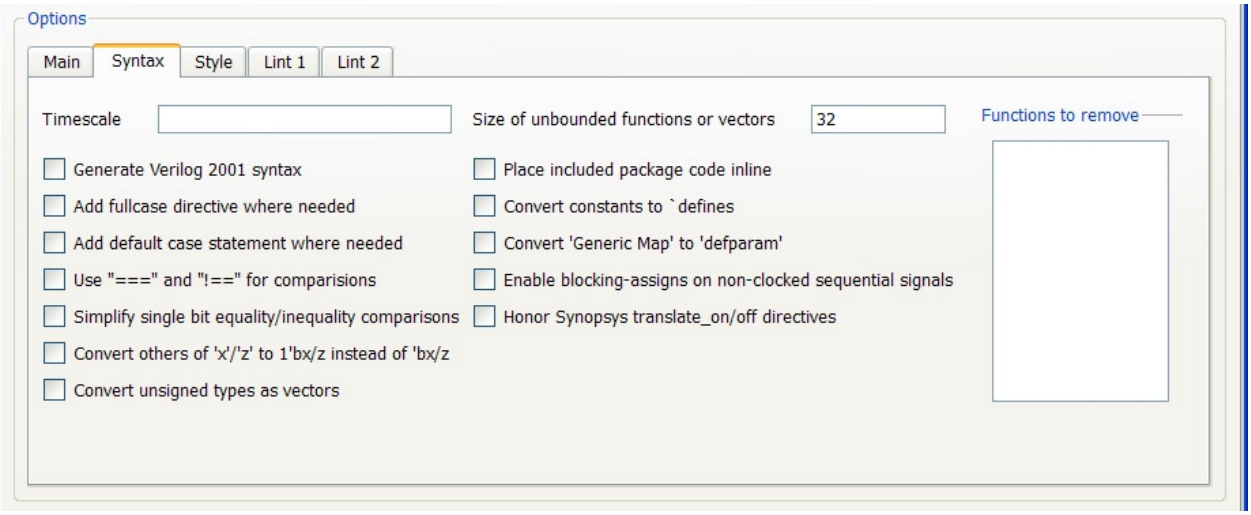

Figure 18 - VHDL-to-Verilog syntax options page

The syntax options page contains the following controls:

Timescale: The value entered into this control specifies the `timescale compiler directive that X-HDL should insert into the translated code. Note that X-HDL does not perform any validity checks on the supplied value.

Generate Verilog2001 syntax: This control specifies that X-HDL should generate Verilog 2001 syntax. Note that if X-HDL can only translate a VHDL construct as a Verilog-2001 construct (such as generates), it will automatically enable Verilog2001 syntax for that construct.

Add fullcase directive where needed: Specifies that X-HDL should add a 'fullcase' synthesis comment to any case statement that doesn't have one

Add default case statement where needed: Specifies that X-HDL should add a default case to any case statement that doesn't have a default.

Use "===" and "!==" for comparisons: This control specifies that X-HDL should use the Verilog strict compare operators for comparisons. The default is to use the non-strict compare operators '==' and '!='.

Simplify single bit equality/inequality comparisons: This control specifies that X-HDL should simplify single bit equality/inequality comparisons. For example, this option will cause the source: 'if (rst = '1')' to be translated to 'if (rst)' instead of 'if (rst == 1'b1)'.

Convert others of 'x'/'z' to 1'bx/z instead of 'bx/z: When enabled, this control specifies that X-HDL should convert an (others => 'x') or (others => 'z') to a single bit. The default is an unsized vector.

Convert unsigned types as vectors: This control specifies that X-HDL should convert unsigned types as vectors. By default, X-HDL converts unsigned types as unsigned integers.

Place included package code inline: This control apecifies that X-HDL should place VHDL package code inline. By default, X-HDL creates `include compiler directives for each included VHDL package.

***Convert constants to `defines:*** This control specifies that X-HDL should convert VHDL constants to Verilog `define compiler directives. By default, X-HDL converts VHDL constants to Verilog parameters.

Convert "Generic Map' to 'defparam': This control specifies that X-HDL should convert VHDL 'Generic Map' to Verilog 'defparam'. By default, X-HDL converts VHDL 'Generic Map' to Verilog parameter overrides.

Enable blocking-assigns on non-clocked sequential signals: This control specifies that blocking-assigns should be used on non-clocked sequential signals. By default X-HDL using non-blocking assigns.

Honor Synopsys translate_on/translate_off directives: This control directs X-HDL to honor Synopsys translate_on and translate_off directives. Code within this these directives is not translated nor contained in the output code. **Currently not implemented.**

Size of unbounded functions or vectors: The value entered in this control is used by X-HDL to set the number of bits for all unbounded VHDL functions or vectors. The default unbounded size is 32.

Style Options Page

The style options page provides controls for the user to select various Verilog code style options.

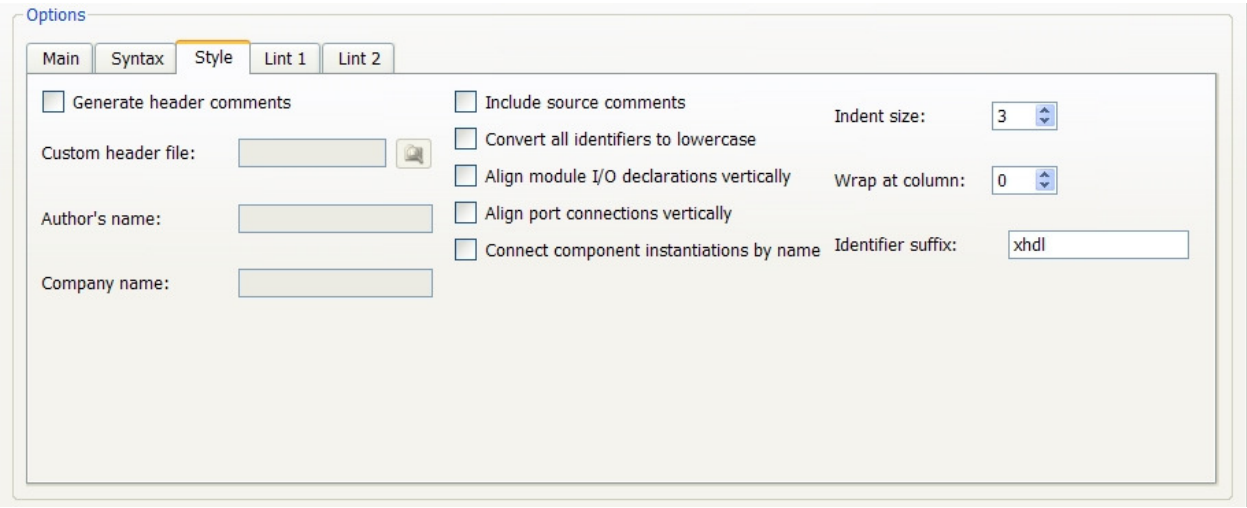

Figure 19 - VHDL-to-Verilog style options page

The syntax options page contains the following controls:

Generate header comments: Enables the generation of header comments. If the Custom header file control is not enabled (see below), a default header will be used.

Custom header file: When header comments are included in the translated output X-HDL will use the contents of file specified in this control for the header comments. The header comments file is free form, however, the following specifiers may be used to direct X-HDL to auto-insert system information:

%a : author's name (see Author's name, below)
%c : company name (see Company name, below)
%d : current date
%f : source filename
%n : module name
%t : current time
%v : X-HDL version
%V : X-HDL version date

Author's name: When the header comments are enabled, the author's name is set to value entered in this control. The default author's name is ''.

Company name: When the header comments are enabled, the company name is set to <name>. The default company name is ''.

Include source comments: When enabled, X-HDL will include source comments in the translation output. Due to necessary relocation of code segments (such as all declarations moved to the VHDL declaration section), the position of some comments may shift slightly.

Convert all identifiers to lowercase: This control specifies that all VHDL identifiers should be translated to lowercase in the translated code. By default, X-HDL uses the case defined by the first occurrence of an identifier.

Align module I/O declarations vertically: This control specifies that X-HDL should align the module I/O declarations vertically.

Align port connections vertically: This control specifies that port connections within instantiations should be aligned vertically in the translated code.

Connect component instantiations by name: This control specifies that X-HDL should connect component instantiations by name regardless of how the component is connected within the source. This option requires that X-HDL be able to locate the translation information for the instantiated component. By default X-HDL will instantiate components using the same style as in the source code.

Indent size: This control specifies the indentation size for the translated code. The default size is 3.

Wrap at column: This control specifies that X-HDL should wrap long lines at the selected column or less. By default X-HDL does not wrap long lines (value of zero).

Identifier suffix: The value entered in this control is used by X-HDL as the suffix that is appended to identifers in the source code which are invalid in VHDL, thereby making the identifiers legal in VHDL. The default suffix is 'xhdl'.

Lint Options Page 1

The lint options page 1 provides controls for the user to select various HDL structural checks.

Figure 20 - VHDL-to-Verilog lint options page 1

Many of the lint controls are not currently implemented, as noted below. The lint options page 1 contains the following controls:

Assignment size check: This control enables assignment size checking. X-HDL will issue a warning if the size of the LHS of an assignment does not match the size of the RHS.

Case option check: This control enables evaluation of case options to ensure that the options are unique and complete. **Currently not implemented**.

Implied latch check: When this control is enabled, X-HDL will issue a warning if a latch structure is detected. **Currently not implemented**.

Combinational loop check: When this control is enabled, X-HDL will issue a warning if a combinational loop is detected. **Currently not implemented.**

Gated clock check: When this control is enabled, X-HDL will issue a warning if a gated clock structure is detected. **Currently not implemented.**

Non-reset FF check: When this control is enabled, X-HDL will issue a warning if it detects a flip-flop structure that does not contain a reset. **Currently not implemented**.

Unused signal check: When this control is enabled, X-HDL will issue a warning for all signals which are defined but not referenced.

Undriven signal check: When this control is enabled, X-HDL will issue a warning for all signals which are defined but undriven.

I/O port check: When this control is enabled, X-HDL will issue a warning if a module does not contain input and output ports.

X/Z state check: When this control is enabled, X-HDL will issue a warning whenever an X or Z constant is detected within the source code.

Nested conditions level: This control enables nested conditions level checking (if and case). If the number of nested condition blocks exceeds the value entered in the control, then X-HDL will issue a warning. A value of 0 for allows infinite nesting (i.e. effectively disables nested condition checking). The default value for nested condition checking is 0.

Nested blocks level: This control enables nested blocks level checking. If the number of nested blocks exceeds the value entered in the control, then X-HDL will issue a warning. A value of 0 allows infinite nesting (i.e. effectively disables nested block checking). The default value for nested block checking is 0.

Lint Options Page 2

The lint options page 2 provides controls for the user to select various VHDL identifier format checks.

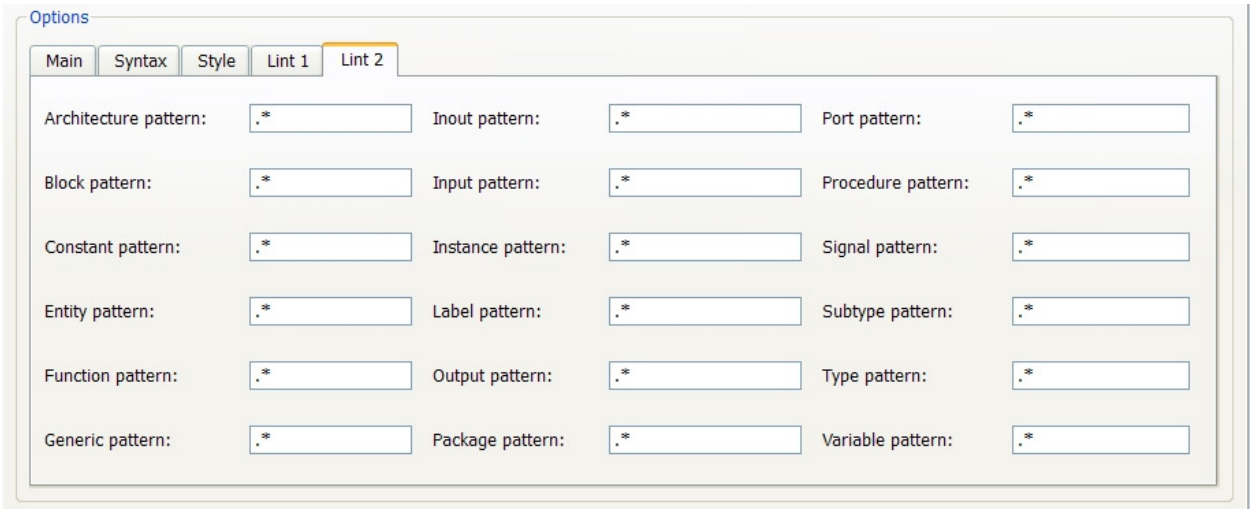

Figure 21 - VHDL-to-Verilog lint options page 2

All identifier formats are specified using regular expression patterns. Please see the Regular Expression Format section for more information on regular expressions used by X-HDL. The lint options page 2 contains the following controls:

Architecture pattern: Sets the architecture name check pattern. X-HDL will issue a warning if an architecture identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Block pattern: Sets the block name check pattern. X-HDL will issue a warning if a block identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Constant pattern: Sets the constant name check pattern. X-HDL will issue a warning if a constant identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Entity pattern: Sets the entity name check pattern. X-HDL will issue a warning if an entity identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Function pattern: Sets the function name check pattern. X-HDL will issue a warning if a function identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Generic pattern: Sets the generic name check pattern. X-HDL will issue a warning if a generic identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Inout pattern: Sets the inout name check pattern. X-HDL will issue a warning if an inout identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Input pattern: Sets the input name check pattern. X-HDL will issue a warning if an input identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Instance pattern: Sets the instance name check pattern. X-HDL will issue a warning if an instance identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Label pattern: Sets the label name check pattern. X-HDL will issue a warning if a label identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Output pattern: Sets the output name check pattern. X-HDL will issue a warning if an output identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Package pattern: Sets the package name check pattern. X-HDL will issue a warning if a package identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Port pattern: Sets the port name check pattern. X-HDL will issue a warning if a port (input, inout, output) identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Procedure pattern: Sets the procedure name check pattern. X-HDL will issue a warning if a procedure identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Signal pattern: Sets the signal name check pattern. X-HDL will issue a warning if a signal identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Subtype pattern: Sets the subtype name check pattern. X-HDL will issue a warning if a subtype identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Type pattern: Sets the type name check pattern. X-HDL will issue a warning if a type identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Variable pattern: Sets the variable name check pattern. X-HDL will issue a warning if a variable identifier in the source code does not match the specified pattern. The default pattern is '.*'.

Command-line Mode

X-HDL provides a command-line interface for use by "power-users" or to run automated tests in a batch mode. In the Unix/Linux environment, X-HDL is invoked in command-line mode by running the xhdl executable and specifying the translator to run, any command-line options and the files to be translated.

% xhdl vhdl|vlog [<options>] filename [filename….]

In the MS Windows environment, a separate executable is provided for command-line mode named xhdlc.exe.

Verilog-to-VHDL Command-line Options

To obtain a list of valid Verilog-to-VHDL translation options, invoke the Verilog translator with the –h command-line option:

% xhdl vlog –h

The valid Verilog command-line options are:

-arch

Format: -arch <arch>

Use <arch> for the architecture name. The default architecture name is 'trans'.

-assign

Format: -assign

Lint function. Enables assignment size checking. X-HDL will issue a warning if the size of the LHS of an assignment does not match the size of the RHS.

-author

Format: -author <name>

When the header comments are enabled (see –header and –custom_header), the author's name is set to <name>. The default author's name is ''.

-blocks

Format: -blocks <#>

Lint function. Enables nested blocks level checking. If the number of nested blocks exceeds <#> then X-HDL will issue a warning. A value of 0 for <#> allows infinite nesting (i.e. effectively disables nested block checking). The default value for nested block checking is 0.

-byname

Format: -byname

Specifies that X-HDL should connect component instantiations by name regardless of how the component is connected within the source. This option requires that X-HDL be able to locate the translation

information for the instantiated component. By default X-HDL will instantiate components using the same style as in the source code.

-case

Format: -case

Lint function. X-HDL will evaluate the case options to ensure that options are unique and complete. **Currently not implemented**.

-comments

Format: -comments

X-HDL will include source comments in the translation output. Due to necessary relocation of code segments (such as all declarations moved to the VHDL declaration section), the position of some comments may shift slightly.

-comp2pkg

Format: -comp2pkg

Component declarations will be mapped to a package rather than in the declaration section of the VHDL architecture. **Currently not implemented**.

-company

Format: -company <name>

When the header comments are enabled (see –header and –custom_header), the company name is set to <name>. The default company name is ‘’.

-conditions

Format: -conditions <#>

Lint function. Enables nested conditions level checking (if and case). If the number of nested condition blocks exceeds <#> then X-HDL will issue a warning. A value of 0 for <#> allows infinite nesting (i.e. effectively disables nested condition checking). The default value for nested condition checking is 0.

-custom_header

Format: -custom_header <file>

When header comments are included in the translated output (see –header) X-HDL will use the contents of <file> for the header comments. The header comments file is free form, however, the following specifiers may be used to direct X-HDL to auto-insert system information:

%a	: author’s name (see –author)
%c	: company name (see –company)
%d	: current date
%f	: source filename
%n	: module name
%t	: current time
%v	: X-HDL version
%V	: X-HDL version date

-custom_trans

Format: -custom_trans <file>

X-HDL provides the capability to define custom translation of identifier names, including functions and tasks, and operators. The definitions are specified in the file named <file>. The file format is straightforward. Each line of the file specifies a from/to translation customization separated by '=>'. For example:

```
vlog_id => vhdl_id;
+ => add($1, $2);
func => newfunc($1);
```

would change all occurrences of 'vlog_id' to 'vhdl_id', + to a function call to add (note: $1 and $2 represent the original parameters from the + operation) and change the function call 'func' to 'newfunc' using only the first parameter from the original 'func' call.

Note: Future version of X-HDL will provide custom translation of functions and tasks based on the paramter types of the function/task calls.

-db

Format: -db <db>

When X-HDL instantiates a component, it requires information about the translated component name, port names, etc. X-HDL stores this information in a textual database during translation of the component. The -db option is used to direct X-HDL to the location of the translated component information. The format of the <db> specification is: name=path, where 'name' is the name of the 'library' (which has little to no meaning in Verilog) and 'path' is the path to the translated componet(s). This option may be specified multiple times.

-defaultcase

Format: -defaultcase

Directs X-HDL to add an 'others' case to any case statement that doesn't have a default

-define

Format: -define <sym>

Define a new symbol <sym>. This is equivalent to having `define <name> <value> in the Verilog source code. The format for <sym> is: name=value | name. This option may be specified multiple times.

-delay

Format: -delay <type>

Specifies which delay value in a Verilog triplet delay specification to be used in the translation.
The valid <type> values are:

- min
- typ
- max

The default delay is 'typ'

-delay0

Format: -delay0

Specifies that X-HDL should add a 'wait for 0 ns' after each blocking ('=') assignment to emulate the operation of the blocking assignment in VHDL. The default is to convert blocking assignments to VHDL non-blocking assignments.

-dest

Format: -dest <dir>

Specify the destination directory for the translated code. The default <dir> is the current working directory.

-event

Format: -event <re>

Lint function. Set the event name check pattern to <re>. X-HDL will issue a warning if an event identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-ext

Format: -ext <ext>

Use <ext> as the translated filename extension. The default extension is '.vhd'.

-fix_multiple

Fomat: -fix_multiple

Fix signals which are assigned in multiple procedural blocks, which causes multiple driver conflicts in VHDL. **Currently not implemented**.

-fullcase

Format: -fullcase

Add a 'fullcase' synthesis comment to any case statement that doesn't have one

-function

Format: -function <re>

Lint function. Set the function name check pattern to <re>. X-HDL will issue a warning if a function identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-gated

Format: -gated

Lint function. Enable gated clock checking. X-HDL will issue a warning if a gated clock structure is detected. **Currently not implemented.**

-header

Format: -header

Generate header comments. If –custom_header is not also specified, a default header will be used.

-hier

Format: -hier

Perform a hierarchical translation. In this case, rather than filenames specified on the command-line, directory names are specified. X-HDL will automatically recurse through the directory structure starting at the specified directory and translate all Verilog source files matching the specified hierarchical filter (see –hier_filter) in a botton-up order. All translated files will be placed in the location specified by –dest.

-hier_filter

Format: -hier_filter <filter>

When performing a hierarchical translation (see –hier), <filter> will be applied to all filenames in the searched directories. Files matching the filter will be translated. The default filter is '\.v$'. See the section on Regular Expression Formats for information on valid <filter> specifications.

-include

Format: -include <file>

Include additional options from the file <file>.

-incpath

Format: -incpath <path>

Specifies the path to search for `include files. This option may be specified multiple times.

-indent

Format: -indent <#>

Specify the indentation size for the translated code. The default size is 3.

-inout

Format: -inout <re>

Lint function. Set the inout name check pattern to <re>. X-HDL will issue a warning if an inout identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-input

Format: -input <re>

Lint function. Set the input name check pattern to <re>. X-HDL will issue a warning if an input identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-instance

Format: -instance <re>

Lint function. Set the instance name check pattern to <re>. X-HDL will issue a warning if an instance identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-integer

Format: -integer <re>

Lint function. Set the integer name check pattern to <re>. X-HDL will issue a warning if an integer identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-iocheck

Format: -iocheck

Lint function. When enabled, X-HDL will issue a warning if a module does not contain input and output ports.

-label

Format: -label <re>

Lint function. Set the event name label pattern to <re>. X-HDL will issue a warning if a label identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-latch

Format: -latch

Lint function. When enabled, X-HDL will issue a warning if a latch structure is detected. **Currently not implemented**.

-log

Format: -log <file>

Specifies that X-HDL should output messages to the file <file>.

-loop

Format: -loop

Lint function. When enabled, X-HDL will issue a warning if a combinational loop is detected. **Currently not implemented.**

-lowercase

Format: -lowercase

Specifies that X-HDL should make all VHDL keywords lowercase. The default is uppercase keywords.

-module

Format: -module <re>

Lint function. Set the module name check pattern to <re>. X-HDL will issue a warning if a module identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-no_auto_pkgs

Format: -no_auto_pkgs

By default, X-HDL automatically adds standard VHDL package/library clauses to the translated output. This option specifies that these clauses should not be automatically added (see –pkg option).

-output

Format: -output <re>

Lint function. Set the output name check pattern to <re>. X-HDL will issue a warning if an output identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-package

Format: -package

Some Verilog files which are `include-d into other Verilog source files are good candidates to be translated to VHDL packages. This control enables translating a partial Verilog source file into a VHDL package. The partial Verilog file must contain only parameter declarations and subprograms. Other Verilog constructs are not appropriate for VHDL packages. If this control is not enabled, the `include-d code will be placed in-line at the point of the `include statement and translated.

-param2int

Format: -param2int

By default, X-HDL translated unsized parameters to VHDL integers and sized parameters to VHDL vectors. This control directs X-HDL to translate all parameters to VHDL integers regardless whether they are sized or not.

-parameter

Format: -parameter <re>

Lint function. Set the parameter name check pattern to <re>. X-HDL will issue a warning if a parameter identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-pkg

Format: -pkg <pkg>

Specifies that the package <pkg> should be added to the VHDL output code (see –no_auto_pkgs). This option may be specified multiple times.

-port

Format: -port <re>

Lint function. Set the port name check pattern to <re>. X-HDL will issue a warning if a port identifier (input, output, or inout) in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-post

Format: -post <name>

Specifies that the translated code should be processed through the executable program <name> before being written to the output file. This provides the capability to further customize the translated output if necessary. The executable program must accept stdin data and provide output on stdout.

-pre

Format: -pre <name>

Specifies that the source code should be processed through the executable program <name> before being translated. This provides the capability to massage the source code prior to translation if necessary. The executable program must accept stdin data and provide output on stdout.

-preserve

Format: -preserve

By default, X-HDL creates the translated filename using the module name within the source code. This option causes X-HDL to use the source filename (with different extension) for the translated code.

-queue

Format: -queue

If a license is not available –queue will direct X-HDL to wait until the license becomes available. The default operation is to run in demo mode.

-real

Format: -real <re>

Lint function. Set the real name check pattern to <re>. X-HDL will issue a warning if a real identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-register

Format: -register <re>

Lint function. Set the event reg check pattern to <re>. X-HDL will issue a warning if a reg identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-reset

Format: -reset

Lint function. Enable non-reset FF checking. X-HDL will issue a warning if it detects a flip-flop structure that does not contain a reset. **Currently not implemented**.

-sep_files

Format: -sepfiles

Specifies that X-HDL should generate separate entity/architecture files. By default, X-HDL places both the entity and architecture into the same file.

-single

Format: -single <file>

Specifies that X-HDL should output all translated code to a single file named <file>.

-srcpath

Format: -srcpath <path>

Specifies the path of the source files to be translated. This option is helpful if multiple source files are being translated from a directory other than current working directory.

-std2comb

Format: -std2comb

Indicates that standard Verilog gate instances (and, or, nor, etc.) should be translated to combinational logic rather than a gate instance.

-suffix

Format: -suffix <text>

Specifies that X-HDL should append <suffix> to identifers in the source code which are invalid in VHDL, thereby making the identifiers legal in VHDL. The default suffix is 'xhdl'.

-summary

Format: -summary

Output translation summary information to the display and the log file, if enabled (see –log). Information contained in the summary includes:

- filename
- Number of errors
- Number of warnings
- Lines of code
- Number of input/output/bidir ports
- Number of parameters/regs/wires/events/integers/reals/times
- Number of concurrent statements
- Number of sequential statements
- Number of tasks/functions
- Number of instances
- Number of clock domains

-task

Format: -task <re>

Lint function. Set the task name check pattern to <re>. X-HDL will issue a warning if a task identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-time

Format: -time <re>

Lint function. Set the time name check pattern to <re>. X-HDL will issue a warning if a time identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-trans_onoff

Format: -trans_onoff

Specifies that X-HDL should honor Synopsys translate_on and translate_off directives. Code within this these directives is not translated nor contained in the output code. **Currently not implemented.**

-type

Format: -type <type>

Specifies that the VHDL <type> should be used for the translation of all Verilog regs and wires. The valid values for <type> are:

bit
std_logic
std_ulogic

The default for <type> is ‘std_logic’

-undriven

Format: -undriven

Lint function. Enable undriven signal checking. X-HDL will issue a warning for all signals which are defined but undriven.

-unused

Format: -unused

Lint function. Enable unused signal checking. X-HDL will issue a warning for all signals which are defined but not referenced.

-verbose

Format: -verbose <#>

Specifies the detail of messages output during the translation process. The valid values for <#> are:

0 = no messages.
1 = basic informational messages
2 = verbose messages
3 = very verbose messages

-vert

Format: -vert

Specifies that port connections within instantiations should be aligned vertically in the translated code.

-vhdl93

Format: -vhdl93

Specifies that X-HDL should generate VHDL'93 syntax. By default, X-HDL uses VHDL'87 syntax.

-wire

Format: -wire <re>

Lint function. Set the wire name check pattern to <re>. X-HDL will issue a warning if a wire identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-wrap

Format: -wrap <#>

Specifies that X-HDL should wrap long lines at column <#> or less. By default X-HDL does not wrap long lines.

-xzcheck

Format: -xzcheck

Lint function. Enable X/Z state checking. X-HDL will issue a warning whenever an X or Z constant is detected within the source code.

VHDL-to-Verilog Command-line Options

-architecture

Format: -architecture <re>

Lint function. Set the architecture name check pattern to <re>. X-HDL will issue a warning if an architecture identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-assign

Format: -assign

Lint function. Enables assignment size checking. X-HDL will issue a warning if the size of the LHS of an assignment does not match the size of the RHS.

-author

Format: -author <name>

When the header comments are enabled (see –header and –custom_header), the author's name is set to <name>. The default author's name is ''.

-block

Format: -block <re>

Lint function. Set the block name check pattern to <re>. X-HDL will issue a warning if a block identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-blocks

Format: -blocks <#>

Lint function. Enables nested blocks level checking. If the number of nested blocks exceeds <#> then X-HDL will issue a warning. A value of 0 for <#> allows infinite nesting (i.e. effectively disables nested block checking). The default value for nested block checking is 0.

-blockseq

Format: -blockseq

Specifies that blocking-assigns should be used on non-clocked sequential signals. By default X-HDL using non-blocking assigns.

-byname

Format: -byname

Specifies that X-HDL should connect component instantiations by name regardless of how the component is connected within the source. This option requires that X-HDL be able to locate the translation information for the instantiated component. By default X-HDL will instantiate components using the same style as in the source code.

-case

Format: -case

Lint function. X-HDL will evaluate the case options to ensure that options are unique and complete. **Currently not implemented**.

-comments

Format: -comments

X-HDL will include source comments in translation output. Due to necessary relocation of code segments (such as local functions or procedures), the position of some comments may shift slightly.

-company

Format: -company <name>

When the header comments are enabled (see –header and –custom_header), the company name is set to <name>. The default company name is ‘’.

-conditions

Format: -conditions <#>

Lint function. Enables nested conditions level checking (if and case). If the number of nested condition blocks exceeds <#> then X-HDL will issue a warning. A value of 0 for <#> allows infinite nesting (i.e. effectively disables nested condition checking). The default value for nested condition checking is 0.

-constant

Format: -constant <re>

Lint function. Set the constant name check pattern to <re>. X-HDL will issue a warning if a constant identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-custom_header

Format: -custom_header <file>

When header comments are included in the translated output (see –header) X-HDL will use the contents of <file> for the header comments. The header comments file is free form, however, the following specifiers may be used to direct X-HDL to auto-insert system information:

%a	: author’s name (see –author)
%c	: company name (see –company)
%d	: current date
%f	: source filename
%n	: module name
%t	: current time
%v	: X-HDL version
%V	: X-HDL version date

-custom_trans

Format: -custom_trans <file>

X-HDL provides the capability to define custom translation of identifier names, including functions and tasks, and operators. The definitions are specified in the file named <file>. The file format is straightforward. Each line of the file specifies a from/to translation customization separated by ‘=>’. For example:

```
vhdl_id => vlog_id;
+ => add($1, $2);
func => newfunc($1);
```

would change all occurrences of ‘vhdl_id’ to ‘vlog_id’, + to a function call to add (note: $1 and $2 represent the original parameters from the + operation) and change the function call ‘func’ to ‘newfunc’ using only the first parameter from the original ‘func’ call.

Note: Future version of X-HDL will provide custom translation of functions and tasks based on the paramter types of the function/task calls.

-db

Format: -db <db>

When X-HDL instantiates a component, it requires information about the translated component name, port names, etc. X-HDL stores this information in a textual database during translation of the component. The -db option is used to direct X-HDL to the location of the translated component information. The format of the <db> specification is: name=path, where ‘name’ is the name of the ‘library’ and ‘path’ is the path to the translated componet(s). This option may be specified multiple times.

-defaultcase

Format: -defaultcase

Directs X-HDL to add an 'others' case to any case statement that doesn't have a default

-defines

Format: -defines

Specifies that X-HDL should convert VHDL constants to Verilog `define compiler directives. By default, X-HDL converts VHDL constants to Verilog parameters.

-defparam

Format: -defparam

Specifies that X-HDL should convert VHDL 'Generic Map' to Verilog 'defparam'. By default, X-HDL converts VHDL ‘Generic Map’ to Verilog parameter overrides.

-dest

Format: -dest <dir>

Specify the destination directory for the translated code. The default <dir> is the current working directory.

-entity

Format: -entity <re>

Lint function. Set the entity name check pattern to <re>. X-HDL will issue a warning if an entity identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-ext

Format: -ext <ext>

Use <ext> as the translated filename extension. The default extension is '.v'.

-fullcase

Format: -fullcase

Add a 'fullcase' synthesis comment to any case statement that doesn't have one

-function

Format: -function <re>

Lint function. Set the function name check pattern to <re>. X-HDL will issue a warning if a function identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-gated

Format: -gated

Lint function. Enable gated clock checking. X-HDL will issue a warning if a gated clock structure is detected. **Currently not implemented.**

-generic

Format: -generic <re>

Lint function. Set the generic name check pattern to <re>. X-HDL will issue a warning if a generic identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-header

Format: -header

Generate header comments. If –custom_header is not also specified, a default header will be used.

-hier

Format: -hier

Perform a hierarchical translation. In this case, rather than filenames specified on the command-line, directory names are specified. X-HDL will automatically recurse through the directory structure starting at the specified directory and translate all Verilog source files matching the specified hierarchical filter (see –hier_filter) in a botton-up order. All translated files will be placed in the location specified by –dest.

-hier_filter

Format: -hier_filter <filter>

When performing a hierarchical translation (see –hier), <filter> will be applied to all filenames in the searched directories. Files matching the filter will be translated. The default filter is '\.v$'. See the section on Regular Expression Formats for information on valid <filter> specifications.

-include

Format: -include <file>

Include additional options from the file <file>.

-indent

Format: -indent <#>

Specify the indentation size for the translated code. The default size is 3.

-inline_pkg

Format: -inline_pkg

Specifies that X-HDL should place VHDL package code inline. By default, X-HDL creates `include compiler directives for each included VHDL package.

-inout

Format: -inout <re>

Lint function. Set the inout name check pattern to <re>. X-HDL will issue a warning if an inout identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-input

Format: -input <re>

Lint function. Set the input name check pattern to <re>. X-HDL will issue a warning if an input identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-instance

Format: -instance <re>

Lint function. Set the instance name check pattern to <re>. X-HDL will issue a warning if an instance identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-iocheck

Format: -iocheck

Lint function. When enabled, X-HDL will issue a warning if a module does not contain input and output ports.

-label

Format: -label <re>

Lint function. Set the label name check pattern to <re>. X-HDL will issue a warning if a label identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-latch

Format: -latch

Lint function. When enabled, X-HDL will issue a warning if a latch structure is detected. **Currently not implemented**.

-log

Format: -log <file>

Specifies that X-HDL should output messages to the file <file>.

-loop

Format: -loop

Lint function. When enabled, X-HDL will issue a warning if a combinational loop is detected. **Currently not implemented.**

-lowercase

Format: -lowercase

Specifies that all VHDL identifiers should be translated to lowercase in the translated code. By default, X-HDL uses the case defined by the first occurrence of an identifier.

-output

Format: -output <re>

Lint function. Set the output name check pattern to <re>. X-HDL will issue a warning if an output identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-package

Format: -package <re>

Lint function. Set the package name check pattern to <re>. X-HDL will issue a warning if a package identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-port

Format: -port <re>

Lint function. Set the port name check pattern to <re>. X-HDL will issue a warning if a port (input, output, inout) identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-post

Format: -post <name>

Specifies that the translated code should be processed through the executable program <name> before being written to the output file. This provides the capability to further customize the translated output if necessary. The executable program must accept stdin data and provide output on stdout.

-pre

Format: -pre <name>

Specifies that the source code should be processed through the executable program <name> before being translated. This provides the capability to massage the source code prior to translation if necessary. The executable program must accept stdin data and provide output on stdout.

-preserve

Format: -preserve

By default, X-HDL creates the translated filename using the module name within the source code. This option causes X-HDL to use the source filename (with different extension) for the translated code.

-procedure

Format: -procedure <re>

Lint function. Set the procedure name check pattern to <re>. X-HDL will issue a warning if a procedure identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-queue

Format: -queue

If a license is not available –queue will direct X-HDL to wait until the license becomes available. The default operation is to run in demo mode.

-remove

Format: -remove <function>

Specifies that the function <function> should be removed from the translated output. This option is typically used to remove unneeded type-cast functions.

-reset

Format: -reset

Lint function. Enable non-reset FF checking. X-HDL will issue a warning if it detects a flip-flop structure that does not contain a reset. **Currently not implemented**.

-signal

Format: -signal <re>

Lint function. Set the signal name check pattern to <re>. X-HDL will issue a warning if a signal identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-simplify_compare

Format: -simplify_compare

Specifies that X-HDL should simplify single bit equality/inequality comparisons. For example, this option will cause the source: ‘if (rst = ‘1’)’ to be translated to ‘if (rst)’ instead of ‘if (rst == 1’b1)’.

-single

Format: -single <file>

Specifies that X-HDL should output all translated code to a single file named <file>.

-srcpath

Format: -srcpath <path>

Specifies the path of the source files to be translated. This option is helpful if multiple source files are being translated from a directory other than current working directory.

-strict_compare

Format: -strict_compare

Specifies that X-HDL should use the Verilog strict compare operators ('===' and '!==') for comparisons. The default is to use the non-strict compare operators '==' and '!='.

-subtype

Format: -subtype <re>

Lint function. Set the subtype name check pattern to <re>. X-HDL will issue a warning if a subtype identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-suffix

Format: -suffix <text>

Specifies that X-HDL should append <suffix> to identifers in the source code which are invalid in Verilog, thereby making the identifiers legal in Verilog. The default suffix is 'xhdl'.

-summary

Format: -summary

Output translation summary information to the display and the log file, if enabled (see –log). Information contained in the summary includes:

- filename
- Number of errors
- Number of warnings
- Lines of code
- Number of input/output/bidir ports
- Number of generics/constant/signals/variables
- Number of concurrent statements
- Number of sequential statements
- Number of procedures/functions
- Number of instances
- Number of clock domains

-timescale

Format: -timescale <ts>

Specifies that X-HDL should set the `timescale compiler directive to <ts> in the translated code. Note that X-HDL does not perform any validity checks on the supplied value of <ts>.

-trans_onoff

Format: -trans_onoff

Specifies that X-HDL should honor Synopsys translate_on and translate_off directives. Code within this these directives is not translated nor contained in the output code. **Currently not implemented.**

-type

Format: -type <re>

Lint function. Set the type name check pattern to <re>. X-HDL will issue a warning if a type identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-unbound_size

Format: -unbound_size <size>

Specifies that X-HDL should use <size> number of bits for all unbounded VHDL functions or vectors. The default unbounded size is 32.

-undriven

Format: -undriven

Lint function. Enable undriven signal checking. X-HDL will issue a warning for all signals which are defined but undriven.

-unsigned_vec

Format: -unsigned_vec

Specifies that X-HDL should convert unsigned types as vectors. By default, X-HDL converts unsigned types as unsigned integers.

-unused

Format: -unused

Lint function. Enable unused signal checking. X-HDL will issue a warning for all signals which are defined but not referenced.

-variable

Format: -variable <re>

Lint function. Set the variable name check pattern to <re>. X-HDL will issue a warning if a variable identifier in the source code does not match the specified pattern. The default pattern is '.*'. See the section on Regular Expression Formats for information on valid <re> specifications.

-vert

Format: -vert

Specifies that port connections within instantiations should be aligned vertically in the translated code.

-vertio

Format: -vertio

Specifies that X-HDL should align module I/O declarations vertically.

-vlog2001

Format: -vlog2001

Specifies that X-HDL should generate Verilog 2001 syntax. Note that if X-HDL can only translate a VHDL construct as a Verilog-2001 construct (such as generates), it will automatically enable –vlog2001 for that construct.

-wrap

Format: -wrap <#>

Specifies that X-HDL should wrap long lines at column <#> or less. By default X-HDL does not wrap long lines.

-xzcheck

Format: -xzcheck

Lint function. Enable X/Z state checking. X-HDL will issue a warning whenever an X or Z constant is detected within the source code.

-xzsize

Format: -xzsize

Specifies that X-HDL should convert others of 'x'/'z' to 1'bx/z instead of 'bx/z

Translation Limitations and Considerations

Though the goal of the X-HDL translator is to translate HDL source code 100% cleanly, incompatible differences between Verilog and VHDL make this goal difficult to achieve, and in some cases near impossible. The following sections detail some of the limitations of language translation for both Verilog and VHDL. Some language constructs to consider not using when developing translatable source code are also discussed.

Note that this section is still in work and will be expanded in future X-HDL releases.

Verilog limitations/considerations

**`include Files:** The `include directive is a pre-processor directive. Therefore, any code that is `include-d into your main source code body will be inserted in-line by the X-HDL pre-processor prior to translation. The impact of this in-lining of the code is that if your `include-d file is used by multiple modules, the code will be multiply copied in the VHDL translations. So you lose the ability to make a change, say a variable value, and have it applied to all files that use it.

Other pre-processor directives: The other pre-processor directives, `define, `undef, `ifdef, `ifndef, `else and `endif are evaluated prior to translation. The functionality of these directives is cannot be converted to VHDL and is, therefore, lost during the translation process. For best translation results it is best to avoid the use of these directives.

Fork-Join Blocks: Fork-Join blocks are inherently non-translatable. Avoid at all costs. X-HDL will produce a warning and essentially ignore the block.

For-Loops: For best results, the for-loop iterator variable should be declared an integer rather than a reg. A reg translates to a VHDL vector and a vector is not easily used as a VHDL for-loop iterator. If the Verilog iterator is a reg, X-HDL will translate the Verilog for-loop construct into a VHDL while-loop construct.

deassign/force/release/disable: These Verilog operations do not have a similar counterpart in VHDL and are ignored by X-HDL.

Multiple Drivers: In Verilog it is perfectly acceptable to assign a reg from multiple *always* blocks. However, when translated, this will result in signals having multiple drivers in VHDL which causes unknown values on those signals. It is best to avoid assigning a reg from more than one *always* block.

Tasks: Tasks are translated to VHDL procedures. If the task drives regs declared outside of the task, then the task will be translated as a VHDL local procedure, otherwise as a VHDL global procedure. A local procedure will be defined within a VHDL process that references that procedure. If the Verilog task is referenced in multiple *always* blocks, then the VHDL procedure will be defined in multiple processes in VHDL. This situation can cause a multiple-driver condition in VHDL (see the Multiple Drivers paragraph).

Functions: Functions are translated to VHDL functions. If the function drives regs declared outside of the function, then the function will be translated as a VHDL local function, otherwise as a VHDL global function. A local function will be defined within a VHDL process that references that function. If the Verilog function is referenced in multiple *always* blocks, then the VHDL function will be defined in multiple processes in VHDL. This situation can cause a multiple-driver condition in VHDL (see the Multiple Drivers paragraph).

System Tasks and Functions: Most system tasks and functions are ignored by X-HDL. The notable exception is the $time function which is translated to the VHDL *now* function.

Parameters: Parameters which are sized (i.e. parameter [7:0] param;) are translated to VHDL vector constants. Unsized parameters (i.e. parameter param) are translated to VHDL integer constants. This behavior can be overridden using the –param2int command-line option which translates all parameter to integers regardless of size.

Parameters that are declared prior to a port declaration are translated to VHDL generics to retain the Verilog parameter override capability. If your code `include-s another file which contains parameters, note that those parameters will be translated to generics if the `include statement occurs before a port declaration.

Though parameters of type 'string' are supported, their use is typically problematic because registers can hold logical as well as string data. Consider:

```
parameter text = "hello";
reg [8*5:0] message;

always
  message <= text;
```

Register 'message' will be translated to VHDL type std_logic_vector because it is a sized register. However, the subsequent assignment to 'message' will produce an error in VHDL because the target needs to be of type string.

VHDL limitations/considerations

Subprograms: If the parameters of subprograms are unbounded, when translated, X-HDL will add a size parameter to the parameter list for each unbounded parameter. It will also set the size on the unbounded parameter to the user-specified unbound size (see –unbound command-line option), which defaults to 32. Subsequent subprogram calls will be modified to include the size of the actual parameter being passed to the subprogram. Example:

VHDL:
```
function example (param : in std_logic_vector) return std_logic is…..
```

VHDL function call:
```
signal data : std_logic_vector(7 downto 0);
target <= example(data);
```

Verilog:
```
function example (param, _param);
  input [31:0] param;
  input _param;
  integer _param;
```

Verilog function call:
```
reg [7:0] data;
target <= example(data, 8);
```

Functions: Function return types are limited to logic types (bit, std_logic, std_ulogic, etc.), logic vector types (bit_vector, std_logic_vector, std_ulogic_vector, etc.) and integers. Other return types, such as records, arrays, etc., will be ignored by X-HDL.

Physical types: Physical types are not supported by X-HDL.

Packages: Packages must be translated prior to translation of the entity/architecture that uses the package. If a package contains only type, component and subprogram declarations, the resultant translation file will be empty. However, the declaration information is stored in an X-HDL database which is used during subsequent entity/architecture translations. The VHDL *USE* clause is translated to a Verilog `include pre-processor directive.

X-Tek License Manager

X-HDL utilizes a custom license manager for application control, called X-Tek License Manager. The license manager is a suite of applications which operate together to provide license handling functions. These applications are detailed in the following sections.

xlmhostid

xlmhostid is a small application that is used to report the host ID of the machine that it is executed on. xlmhostid does not have any command-line options and is simply run from the command line like so:

% xlmhostid

The application will report the host ID(s) of the machine, which can then be used for licensing information.

xlmserver

xlmserver is a license server application that is used to provide licenses to multiple hosts. xlmserver can also operate in a triple-redundant configuration, providing license services from three redundant server machines. The triple-redundant configuration is controlled by the license file and does not require any special setup of xlmserver.

xlmserver is invoked from the command line using the following format:

% xlmserver <options> &

The valid xlmserver options are:

-log <file> : log all activity to the file <log>
-quiet : do not echo output to the display
-license <file> : read license information from file <file>
-verbose : output extra information for all activities

Upon invocation, xlmserver will also read license information from:

a) The local license directory: <X-HDL install_dir>/licenses
b) The license file pointed to by the environment variable XLMLICENSE_FILE, if it exists.

The xlmserver application uses TCP port 12345 for communication with client applications. It is possible that this default port will be in use by another program. In that situation, the environment variable XLMPORT can be used to specify a different TCP port for xlmserver and client applications to use. XLMPORT can be set to any unused TCP port desired.

Additionally, xlmserver requires the time of client machines to be within 10 minutes of the xlmserver machine. If, for some reason, the time skew between machines is greater than 10 minutes and cannot be changed, the environment variable XLMOFFSET can be used to specify a time offset for xlmserver to bring it into time-alignment with the client machine(s). XLMOFFSET can also be specified on client machines to skew their time relative to the xlmserver machine.

xlmcontrol

xlmcontrol is an application to communicate with and control the xlmserver application. xlmcontrol sends commands to xlmserver and receives responses which is then echoes to the display.

The xlmcontrol command formats are detailed below. All commands accept a single command-line option, -server <servername>. The –server command-line option must be used if xlmserver is running on a different machine than xlmcontrol.

Commands:

shutdown : instructs the license server to terminate

```
% xlmcontrol shutdown
```

status [<license>] : report the status of all licenses, or just the specified license.

```
% xlmcontrol status
```

load <filename> : load the licenses from the file <filename>. If the file doesn't exist, the command is ignored.

```
% xlmcontrol load /usr/local/share/licenses.txt
```

clear : clear all licenses being managed by the server

```
% xlmcontrol clear
```

remove <license> : remove a specific license from the server. If the license doesn't exist, the command is ignored.

```
% xlmcontrol remove 472D39503F8971A7B0A1
```

free <license id> : If the specified license is checked out, force it to be checked in. If the license doesn't exist, the command is ignored.

```
% xlmcontrol free 472D39503F8971A7B0A1-1
```

checkout <feature> <version> : Checkout a license for the specified application feature and version. If the checkout is successful, a license identifier is returned. If the checkout is not successful, a empty line is returned.

```
% xlmcontrol checkout XHDL 2006.1031
```

checkin <license id> : checkin a license. If the license is not checked out, the command is ignored.

```
% xlmcontrol checkin 472D39503F8971A7B0A1-1
```

refresh <license id> : refresh a checked out license. All checked out licenses must be refreshed at least every 10 seconds. If the license is not refreshed in the allotted time, it is automatically checked in. Normally, applications which checkout a license perform this function automatically.

```
% xlmcontrol refresh 472D39503F8971A7B0A1-1
```

info <license id> : get information regarding the supplied license. The information includes the feature, version, end data, start date, count and servers for the licenses. If the license doesn't exist, the command is ignored.

```
% xlmcontrol info 472D39503F8971A7B0A1-1
```

term <license id> : get the number of days before the license terminates. If the license does not exist, the command is ignored. If the license is permanent, a very large number is returned.

Supplied VHDL Packages

In nearly all VHDL designs, packages need to be included in the design. These packages usually contain type definitions and conversion functions. The translated VHDL code is no different - it too must have package definitions. However, many VHDL systems are different and require different packages to be included. Therefore, X-HDL does not automatically include the necessary package information to the translated code. Rather, the user can specify the packages to be included and X-HDL will add those to the code. In GUI mode, the packages are added in the 'Package' box in the Verilog translation options page. In batch mode, the packages are defined using one or more -pkg command line options.

In both the GUI and batch mode cases, the package definition is the same - a fully qualified package name such as: IEEE.std_logic_1164.all

X-HDL will automatically separate out the library name (IEEE, in this case) and add a library clause to the code. If more than one package is used from any library, the library clause is only generated once.

During Verilog-to-VHDL translation, some conversion functions may need to be added to the code to account for Verilog's inherent type-casting abilities. X-HDL includes several packages of conversion functions separated by VHDL type. These packages are:

* XHDL_bit.vhdl
* XHDL_std_logic.vhdl
* XHDL_std_ulogic.vhdl

X-HDL also contains a package of conversion functions for reals, integers and such. This package is named:

* XHDL_misc.vhdl

The X-HDL packages can be found in the following directories:

<install dir>/packages/synthesizable

and

<install dir>/packages/non_synthesizable

If you want any of these package definitions included in your source code, you must specify it just like any other package.

Regular Expression Formats

Regexps are built up from expressions, quantifiers, and assertions. The simplest form of expression is simply a character, e.g. x or 5. An expression can also be a set of characters. For example, [ABCD], will match an A or a B or a C or a D. As a shorthand we could write this as [A-D]. If we want to match any of the captital letters in the English alphabet we can write [A-Z]. A quantifier tells the regexp engine how many occurrences of the expression we want, e.g. x{1,1} means match an x which occurs at least once and at most once. We'll look at assertions and more complex expressions later.

We'll start by writing a regexp to match integers in the range 0 to 99. We will require at least one digit so we will start with [0-9]{1,1} which means match a digit exactly once. This regexp alone will match integers in the range 0 to 9. To match one or two digits we can increase the maximum number of occurrences so the regexp becomes [0-9]{1,2} meaning match a digit at least once and at most twice. However, this regexp as it stands will not match correctly. This regexp will match one or two digits within a string. To ensure that we match against the whole string we must use the anchor assertions. We need ^ (caret) which when it is the first character in the regexp means that the regexp must match from the beginning of the string. And we also need $ (dollar) which when it is the last character in the regexp means that the regexp must match until the end of the string. So now our regexp is ^[0-9]{1,2}$. Note that assertions, such as ^ and $, do not match any characters.

If you've seen regexps elsewhere, they may have looked different from the ones above. This is because some sets of characters and some quantifiers are so common that they have special symbols to represent them. [0-9] can be replaced with the symbol \d. The quantifier to match exactly one occurrence, {1,1}, can be replaced with the expression itself. This means that x{1,1} is exactly the same as x alone. So our 0 to 99 matcher could be written ^\d{1,2}$. Another way of writing it would be ^\d\d{0,1}$, i.e. from the start of the string match a digit followed by zero or one digits. In practice most people would write it ^\d\d?$. The ? is a shorthand for the quantifier {0,1}, i.e. a minimum of no occurrences a maximum of one occurrence. This is used to make an expression optional. The regexp ^\d\d?$ means "from the beginning of the string match one digit followed by zero or one digits and then the end of the string".

Our second example is matching the words 'mail', 'letter' or 'correspondence' but without matching 'email', 'mailman', 'mailer', 'letterbox' etc. We'll start by just matching 'mail'. In full the regexp is, m{1,1}a{1,1}i{1,1}l{1,1}, but since each expression itself is automatically quantified by {1,1} we can simply write this as mail; an 'm' followed by an 'a' followed by an 'i' followed by an 'l'. The symbol '|' (bar) is used for alternation, so our regexp now becomes mail|letter|correspondence which means match 'mail' or 'letter' or 'correspondence'. Whilst this regexp will find the words we want it will also find words we don't want such as 'email'. We will start by putting our regexp in parentheses, (mail|letter|correspondence). Our regexp still matches any of the three words but now they are grouped together as a unit. This is useful for building up more complex regexps. It is also useful because it allows us to examine which of the words actually matched. We need to use another assertion, this time \b "word boundary": \b(mail|letter|correspondence)\b. This regexp means "match a word boundary followed by the expression in parentheses followed by another word boundary". The \b assertion matches at a position in the regexp not a character in the regexp. A word boundary is any non-word character such as a space a newline or the beginning or end of the string.

Regexps provide a rich language that can be used in a variety of ways. For example suppose we want to count all the occurrences of 'Eric' and 'Eirik' in a string. Two valid regexps to match these are \b(Eric|Eirik)\b and \bEi?ri[ck]\b. We need the word boundary '\b' so we don't get 'Ericsson' etc. The second regexp actually matches more than we want, 'Eric', 'Erik', 'Eiric' and 'Eirik'.

Characters and Abbreviations for Sets of Characters

Element Meaning

c Any character represents itself unless it has a special regexp meaning. Thus c matches the character c.

\c A character that follows a backslash matches the character itself except where mentioned below. For example if you wished to match a literal caret at the beginning of a string you would write ^.
\a This matches the ASCII bell character (BEL, 0x07).
\f This matches the ASCII form feed character (FF, 0x0C).
\n This matches the ASCII line feed character (LF, 0x0A, Unix newline).
\r This matches the ASCII carriage return character (CR, 0x0D).
\t This matches the ASCII horizontal tab character (HT, 0x09).
\v This matches the ASCII vertical tab character (VT, 0x0B).
\xhhhh This matches the Unicode character corresponding to the hexadecimal number hhhh (between 0x0000 and 0xFFFF).
\0ooo (i.e., \zero ooo) matches the ASCII/Latin1 character corresponding to the octal number ooo (between 0 and 0377).
. (dot) This matches any character (including newline).
\d This matches a digit (QChar::isDigit()).
\D This matches a non-digit.
\s This matches a whitespace (QChar::isSpace()).
\S This matches a non-whitespace.
\w This matches a word character (QChar::isLetterOrNumber() or '_').
\W This matches a non-word character.
\n The n-th backreference, e.g. \1, \2, etc.

Sets of Characters

Square brackets are used to match any character in the set of characters contained within the square brackets. All the character set abbreviations described above can be used within square brackets. Apart from the character set abbreviations and the following two exceptions no characters have special meanings in square brackets.

^ The caret negates the character set if it occurs as the first character, i.e. immediately after the opening square bracket. For example, [abc] matches 'a' or 'b' or 'c', but [^abc] matches anything except 'a' or 'b' or 'c'.
- The dash is used to indicate a range of characters, for example [W-Z] matches 'W' or 'X' or 'Y' or 'Z'.

Using the predefined character set abbreviations is more portable than using character ranges across platforms and languages. For example, [0-9] matches a digit in Western alphabets but \d matches a digit in any alphabet.

Note that in most regexp literature sets of characters are called "character classes".

Quantifiers

By default an expression is automatically quantified by {1,1}, i.e. it should occur exactly once. In the following list E stands for any expression. An expression is a character or an abbreviation for a set of characters or a set of characters in square brackets or any parenthesised expression.

E? Matches zero or one occurrence of E. This quantifier means "the previous expression is optional" since it will match whether or not the expression occurs in the string. It is the same as E{0,1}. For example dents? will match 'dent' and 'dents'.
E+ Matches one or more occurrences of E. This is the same as E{1,}. For example, 0+ will match '0', '00', '000', etc.
E* Matches zero or more occurrences of E. This is the same as E{0,}. The * quantifier is often used by a mistake. Since it matches zero or more occurrences it will match no occurrences at all. For example if we want to match strings that end in whitespace and use the regexp \s*$ we would get a match on every string. This is because we have said find zero or more whitespace followed by the end of string, so even strings that don't end in whitespace will match. The regexp we want in this case is \s+$ to match strings that have at least one whitespace at the end.

E{n} Matches exactly n occurrences of the expression. This is the same as repeating the expression n times. For example, x{5} is the same as xxxxx. It is also the same as E{n,n}, e.g. x{5,5}.
E{n,} Matches at least n occurrences of the expression.
E{,m} Matches at most m occurrences of the expression. This is the same as E{0,m}.
E{n,m} Matches at least n occurrences of the expression and at most m occurrences of the expression.

If we wish to apply a quantifier to more than just the preceding character we can use parentheses to group characters together in an expression. For example, tag+ matches a 't' followed by an 'a' followed by at least one 'g', whereas (tag)+ matches at least one occurrence of 'tag'.

Note that quantifiers are "greedy". They will match as much text as they can. For example, 0+ will match as many zeros as it can from the first zero it finds, e.g. '2.0005'.

Assertions

Assertions make some statement about the text at the point where they occur in the regexp but they do not match any characters. In the following list E stands for any expression.

^ The caret signifies the beginning of the string. If you wish to match a literal ^ you must escape it by writing \^. For example, ^#include will only match strings which begin with the characters '#include'. (When the caret is the first character of a character set it has a special meaning, see Sets of Characters.)
$ The dollar signifies the end of the string. For example \d\s*$ will match strings which end with a digit optionally followed by whitespace. If you wish to match a literal $ you must escape it by writing \$.
\b A word boundary. For example the regexp \bOK\b means match immediately after a word boundary (e.g. start of string or whitespace) the letter 'O' then the letter 'K' immediately before another word boundary (e.g. end of string or whitespace). But note that the assertion does not actually match any whitespace so if we write (\bOK\b) and we have a match it will only contain 'OK' even if the string is "It's OK now".
\B A non-word boundary. This assertion is true wherever \b is false. For example if we searched for \Bon\B in "Left on" the match would fail (space and end of string aren't non-word boundaries), but it would match in "tonne".
(?=E) Positive lookahead. This assertion is true if the expression matches at this point in the regexp. For example, const(?=\s+char) matches 'const' whenever it is followed by 'char', as in 'static const char *'. (Compare with const\s+char, which matches 'static const char *'.)
(?!E) Negative lookahead. This assertion is true if the expression does not match at this point in the regexp. For example, const(?!\s+char) matches 'const' except when it is followed by 'char'.

Frequently Asked Questions

What are the restrictions of Demo Mode?
X-HDL contains a demonstration mode that allows you to run the tool, with some restrictions, when either you are trying out the tool to see how well it will meet your needs, or when a product license is not currently available. In demo mode, the translator is fully functional except that the file size is restricted to 1024 bytes.

What information from my computer do I need to obtain an evaluation/permanent license?
To generate a license for your copy of X-HDL, X-Tek needs to know the host ID of your computer. The easiest way to find the host ID of your machine is to run X-HDL, and select Help->Host ID. This will invoke a dialog box showing the host ID of the machine. Alternatively. you can run the xlmhostid program included with X-HDL.

We are currently using X-HDL version ***. Is a more recent version is available?
The most recent version of X-HDL is always available on the X-Tek web-site at:
www.x-tekcorp.com

Why is X-HDL inserting strange functions into my resultant VHDL code?
When X-HDL translates Verilog to VHDL, it often needs to convert one type to another. To do so, it uses X-Tek created functions. Also, bit-reduction operators in Verilog are translated to equivalent functions in VHDL. These functions can be found in the packages located in the $XTEK_HOME/packages directory.

Why don't the source VHDL package code appear in the resultant Verilog?
VHDL package information is included in the resultant code using the `include compiler directive. If your translated code does not contain a `include for the desired package check the following:

1) Make sure the package has been translated
2) Make sure you have specified a database mapping for the package

Why doesn't the source Verilog `include file code appear in the resultant VHDL?
This usually means that X-HDL could not locate the file specified in the `include compiler directive.

Why are my Verilog parameters being translated into VHDL generics?
Any parameters defined immediately after the port-list of a module definition are subject to being overridden by the parent module in Verilog using the parameter override #(..) syntax. To retain this capability in the resultant VHDL code, those parameters are translated to VHDL generics. To prevent a parameter from being translated as a generic make sure at least one input/output/inout declaration preceeds it in the code.

Why don't my Verilog `define directives get translated to VHDL constants?
Verilog `define directives are actually text replacement macros, not constant definitions. Therefore, prior to the translation process, X-HDL replaces the defines with the specified text.

If my VHDL is contained in separate entity/architecture files the resultant translation file is missing the entity information.
When translating separate entity/architecture files, be sure to select the 'Preserve Filename' (-preserve) option. This will place both the translated entity and architecture code into the desired file.

Software License Agreement

WARNING: INSTALLATION AND USE OF THIS SOFTWARE IS SUBJECT TO THE FOLLOWING SOFTWARE AGREEMENT AND LIMITED WARRANTY. BY INSTALLING THE SOFTWARE BUYER ACKNOWLEDGES READING, UNDERSTANDING AND AGREEING TO THIS AGREEMENT.

1. GRANT OF LICENSE:
X-Tek Corporation hereby grants User/Buyer a non-exclusive, non-transferable, perpetual, royalty-free and worldwide license to use the Licensed Software. Buyer's right to use the Licensed Software terminates upon the violation of any material provision of this License. If any provision of this License shall be held to be unenforceable, such holding shall not affect the enforceability of any other provisions hereof. The "Licensed Software" includes the X-HDL computer program and associated audio-visual material and other documentation.

2. NUMBER OF LICENSED USERS:
The number of simultaneous users of this software is specified and controlled by the license key(s) provided by X-Tek to the Buyer. X-Tek reserves the right to charge additional fees for re-issuing of license keys.

3. PROPRIETARY RIGHTS:
Buyer agrees to take all reasonably necessary steps to ensure that the provisions of this License are not violated by Buyer or by any person under Buyer's control or in Buyer's service, and Buyer shall not, nor shall Buyer cause or permit any such person to disassemble, reverse compile, decrypt, or tamper with the Licensed Software. All copyright, patent, trade secret, trademark and other intellectual and proprietary rights in the Licensed Software are and shall remain the valuable property of X-Tek.

4. REPRODUCTION AND COPYRIGHTS:
Portions of the Licensed Software are entitled to protection under Copyright laws. Buyer may not copy, allow anyone else to copy, or otherwise reproduce any part of the Licensed Software without the prior written consent of X-Tek, except for backup purposes and to install the Licensed Software on Buyer's system. Buyer may not remove or omit any copyright or other proprietary notices from the Licensed Software. Ownership of all copies is retained by X-Tek. Buyer may not copy the printed documentation accompanying the Licensed Software.

4a. LIMITED WARRANTY AND DISCLAIMER:
X-Tek warrants the media on which the Licensed Software is provided to be free from defects in materials and workmanship for one (1) year after delivery. Defective media may be returned for replacement without charge during the one (1) year warranty period unless the media have been damaged by accident or misuse. X-Tek warrants, for one (1) year after purchase, that the unaltered Licensed Software substantially conforms to the documentation and is without material defects. During this Warranty period, X-Tek shall promptly provide all Releases, Updates, and Upgrades at no additional cost to Buyer. X-Tek warrants that X-Tek or X-Tek's vendors have legal title and rights of ownership of the Software and supplemental documentation, and that X-Tek has all necessary rights, title, and interest to grant the rights set forth herein to Buyer. X-Tek warrants that the Software does not infringe any intellectual property right of any third party. X-Tek warrants that the Software contains no disabling code and is free from any viruses at the time of delivery to Buyer. X-Tek also warrants that the Software (i) will function without error or interruption related to Date Data from more than one century; (ii) the Software requires all Date Data (whether received from users, systems, applications or other sources) include an indication of century in each instance; and (iii) all date output and results, in any form, shall include an indication of century in each instance. As used herein, "Date Data" means any data or input which includes an indication of or reference to date. Notwithstanding anything to the contrary contained in this Agreement, X-Tek represents and warrants that there will be no disruption in the delivery of Software or Services under this Agreement as a result of or due to the date change from and between December, 1999, and January, 2000, nor due to the year 2000 being a leap year.

Any implied warranties are limited to the duration of the express warranties above. X-TEK's entire liability and Buyer's exclusive remedy shall be, at the option of Buyer, either (a) return of the price paid or (b) repair or replacement of the Licensed Software that does not meet the foregoing warranty and is returned to X-TEK. Any

replacement software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

THE FOREGOING EXPRESS LIMITED WARRANTIES ARE IN LIEU OF, AND X-TEK DISCLAIMS ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

5. LIMITATION OF LIABILITY:
EXCEPT AS PROVIDED FOR UNDER SECTION 6, BELOW, IN NO EVENT WILL X-TEK OR ITS DISTRIBUTORS OR DEALERS BE LIABLE TO BUYER FOR INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOSS OF INCOME, USE OR INFORMATION; NOR SHALL THE TOTAL LIABILITY OF X-TEK AND ITS DISTRIBUTORS AND DEALERS EXCEED THE AMOUNT PAID FOR THE LICENSED SOFTWARE.

6. INTELLECTUAL PROPERTY INDEMNIFICATION:
X-Tek shall defend, indemnify and hold Buyer harmless from any costs, expenses (including reasonable attorneys' fees), losses, damages or liability incurred because of actual or alleged infringement of any patent, copyright, trade secret, trademark, mask work, or other proprietary right arising out of Buyer's use of the licensed Software or Services provided by X-Tek. Buyer shall notify X-Tek of such claim or demand. With respect to Software, X-Tek's indemnification under this subsection applies only to: (i) unmodified Software licensed to Buyer; (ii) Software modified by X-Tek for Buyer; and (iii) Software modified with X-Tek's express permission.

If a third party's claim endangers or disrupts Buyer's use of the Software, X-Tek shall, at Buyer's option and at no charge to Buyer, (a) obtain a license so Buyer may continue use of the Software; (b) modify the Software to avoid the infringement; (c) replace the Software with a compatible, functionally equivalent and non-infringing product; or if these options are commercially unreasonable (d) refund to Buyer the amount paid for the Software.

7. CONSUMER RIGHTS:
For personal, family, or household use of the Licensed Software some states and provinces do not allow the exclusion of limitation of incidental or consequential damages or limitations on how long an implied warranty lasts. So, the above limitations or exclusions may not apply to Buyer. These warranties give Buyer specific legal rights and remedies; Buyer may also have other rights and remedies which arise from operation of law and vary from state to state or province to province.

8. U.S. GOVERNMENT:
The Licensed Software and documentation is acquired with Restricted Rights. Use, duplication or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data Clause as DFARS 252.227-7013. Contractor/Manufacturer: X-TEK CORPORATION, 6304 Chickaloon Drive, McHenry, IL 60050.

9. TERM AND TERMINATION:
Licensee may terminate this agreement by giving X-Tek written notice of termination. X-Tek may terminate this agreement by giving Licensee written notice of termination if Licensee commits a material breach hereof. Upon any termination due to material breach, Licensee shall cease all use of the Licensed Software associated with such termination, and destroy all copies associated with such termination then in its possession.

10. EFFECT OF AGREEMENT:
This agreement and any agreement between the parties governing access to Licensed Software-compatible data embody the entire understanding between the parties with respect to, and supersede any prior understanding or agreement, oral or written, relating to the Licensed Software and documentation.

11. GOVERNING LAW:
This agreement shall be governed by and construed under the laws of the State of Illinois, U.S.A. jurisdiction. All parties hereto agree to the jurisdiction and venue of the courts of McHenry County, Illinois, and hereby waive any other venue or jurisdictional rights.

12. OTHER PROVISIONS:
Neither this agreement nor any part or portion hereof shall be assigned, sublicensed or otherwise transferred by Buyer. Should any provision of this agreement be held to be void, invalid, unenforceable, or illegal by a court, the validity and enforceability of the other provisions shall not be affected thereby. Failure of a party to enforce any provision of this agreement shall not constitute or be construed as a waiver of such provision or of the right to enforce such provision.