

VDT User Manual Addendum

Table of Contents

Introduction.....	1
Changes in Version 2.0.....	1
Tool tree view and controls.....	2
Design menu horizontal toolbar.....	2
Tools context menu actions.....	3
Tool states.....	5
Automatic tool sequencing in VDT.....	6
Configuring a remote server to run external tools.....	6

Introduction

This Addendum to VDT User Manual (VDT-UserManual.pdf,VDT-UserManual.odt) is written to cover the new features and controls implemented since the original VDT was released and documented. Some of the functionality described in the original document was removed, but most is still applicable. Please see ExDT-TSL-Eng.pdf (ExDT-TSL-Eng.odt) for the description of the TSL – Tool Specification Language – this language is used to configure all the parameter menus and describe the external tool functionality.

Current release supports Xilinx® Vivado® as well as ISE tools, but it should be possible to integrate tools from other FPGA manufacturers to have the same IDE for Xilinx and Altera.

Changes in Version 2.0

VDT plugin for Eclipse IDE is updated to support wide range of development tools. This is the list of the features that were added in the current release:

- Control of the “tool sessions” where commands are sent to the open tool and response is read back from the tool output, extending the possibility of launching external tools through the command line parameters;
- Launching tools on a remote host, separate from the development environment, simplifying isolation of the proprietary tools and the Free Software programs;
- Supplementing Eclipse mechanism of handling problems (errors, warnings, info) reported by the external tools with the use of external text pre-processors/parsers, that receive the running tools stdout and stderr streams, filter/modify them and feed back to VDT for Eclipse;
- Processing of “SupressWarnings” keyword accompanied by “all” or list of tool names in a line preceding the source line causing a problem to reduce “noise” of known tool warnings;
- Recording, managing and playing back tool log files – this allows to change message filter settings and restore Eclipse problem markers without running the actual tools;
- Saving/restoring the tool states using external tool functionality or archiving the tool work area files, managing state timestamps
- Support of launching related tools, re-running ones that a “dirty” (source files, states or

parameter settings were modified since it ran) or the needed state is not current and was not saved;

- managing of tool dependency state - “pinning” tools so it will be considered current regardless of the input changes;

Tool tree view and controls

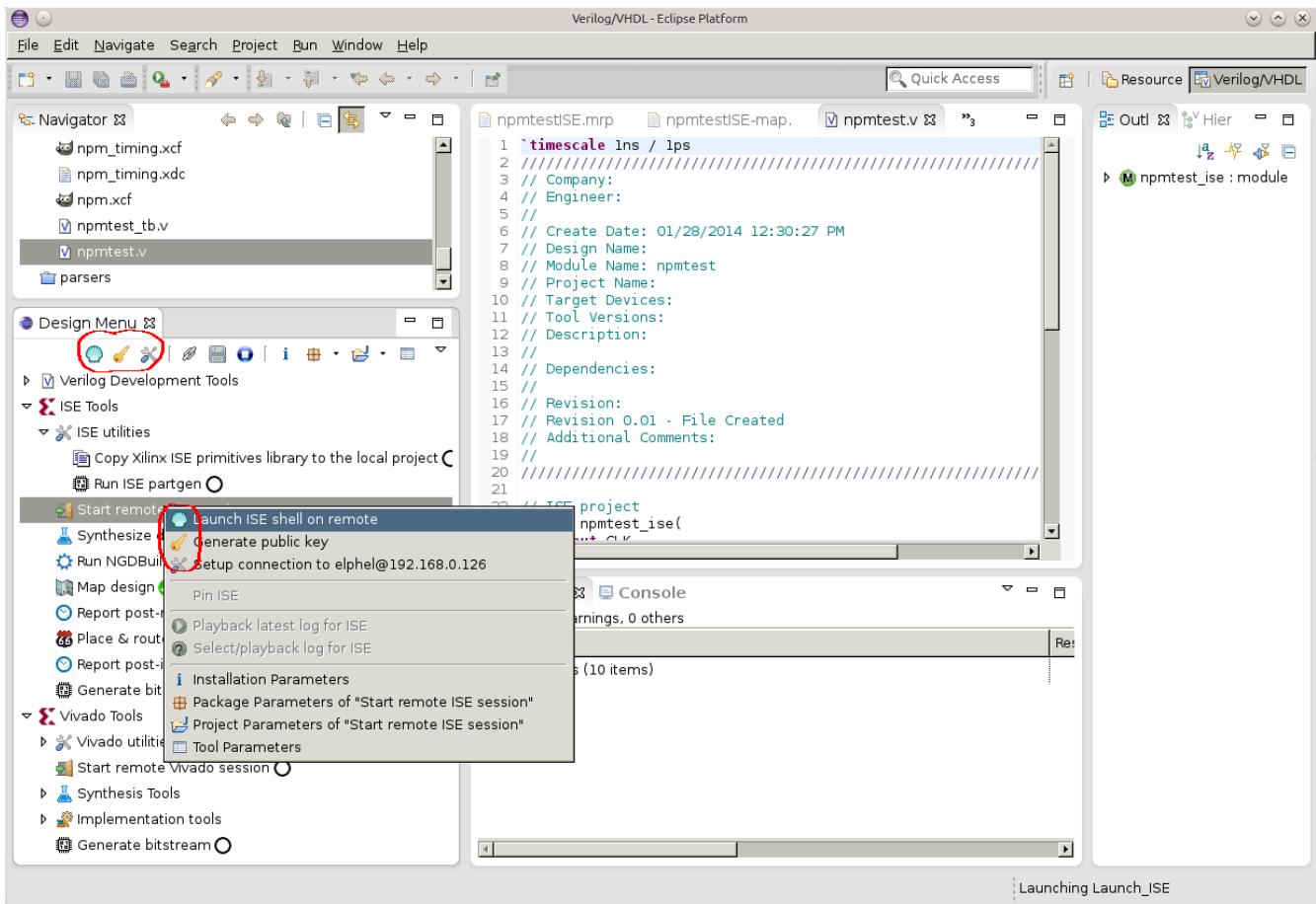




Fig. 1. Tool tree view with multi-action context menu

Tools are presented as a collapsible tree view, tool can be launched by double-click on the tree item, using the tool action button on the horizontal tool bar or using the top lines in the tool context menu. One tool item may have several related actions, on the Fig.1. there are 3 possible action for the same tool. When the tool is launched using double-click, the first action is always selected.

Design menu horizontal tool-bar

To the right of the action icons on the horizontal tool bar there are 3 toggle buttons, some of them may be disabled if the action is not possible for the current tool or tool state:

 - Tool link. If some tool is launched, this causes scanning of the preceding tools and launch them first if their state is not current. When the button is pressed, the link is broken and the tool launch does not trigger the other ones

 - Save state (only active if the current state is not saved). When this button is pressed, it will stay pressed until the current state is saved as a file (filename includes the time stamp) and the linked

resource is created to point to this state as “latest”

⊖- Stop. When this button is activated, the launch sequence will stop at the end of the currently running tool (before auto-save if it is enabled). If the shift key is held when stop is activated, the tool will be marked as failed and the tool launch lock (currently only one tool can run simultaneously) will be released. It is not a clean interruption of the external tool.

To the right of this button group there are controls for installation, package, project and tool properties settings as described in VDT-UserManual. These are additions to the next drop-down menu, as shown on Fig.2.:

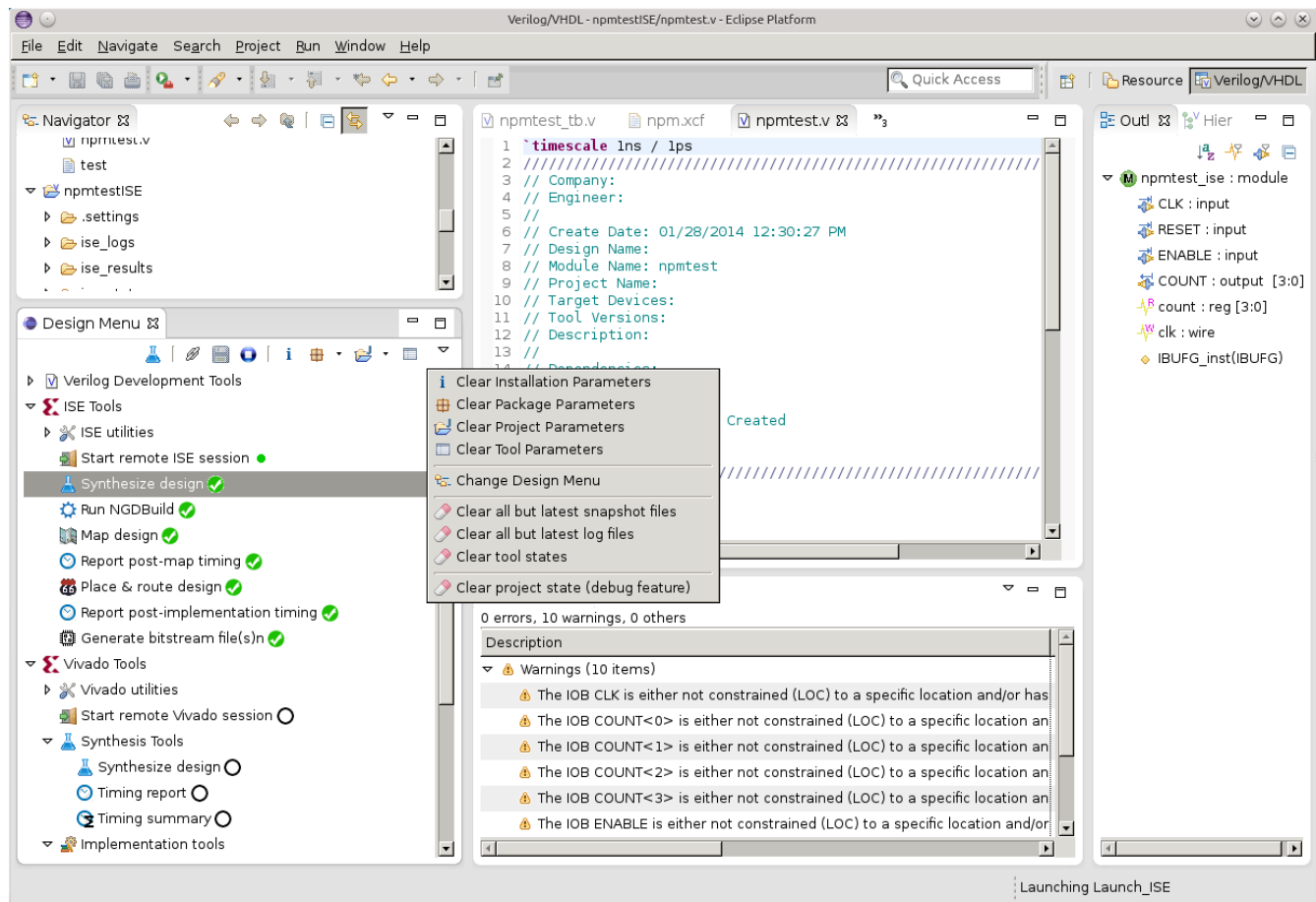


Fig.2. Drop-down Clear menu

Options to clear parameter settings for different contexts and an option to change design menu (see VDT-UserManual) There are four new items:

- 🧼 *Clear all but the latest snapshot files* – mark all the tools as if they never ran;
- 🧼 *Clear all but the latest log files* – preserve only the latest log for each tool, erase all the older ones;
- 🧼 *Clear tool states* – preserve only the latest available state file for each tool ran, erase all others;
- 🧼 *Clear project state (debug feature)* – erase all the persistent properties attached to the current project;

Tools context menu actions

In addition to the tool actions (circled red on the Fig.1.) and the parameter setup menus (identical to the corresponding horizontal tool-bar buttons) there are several new actions that might be enabled in the

menu as shown on Fig.3.

- Pin <tool-name> - this is a toggle button available for the tools that were completed successfully before. When checked the tool state icon (shown to the right of the tool name) changes to a pin and this tool will not automatically rerun when a dependent tool is launched and the files/states this one depends on are modified.

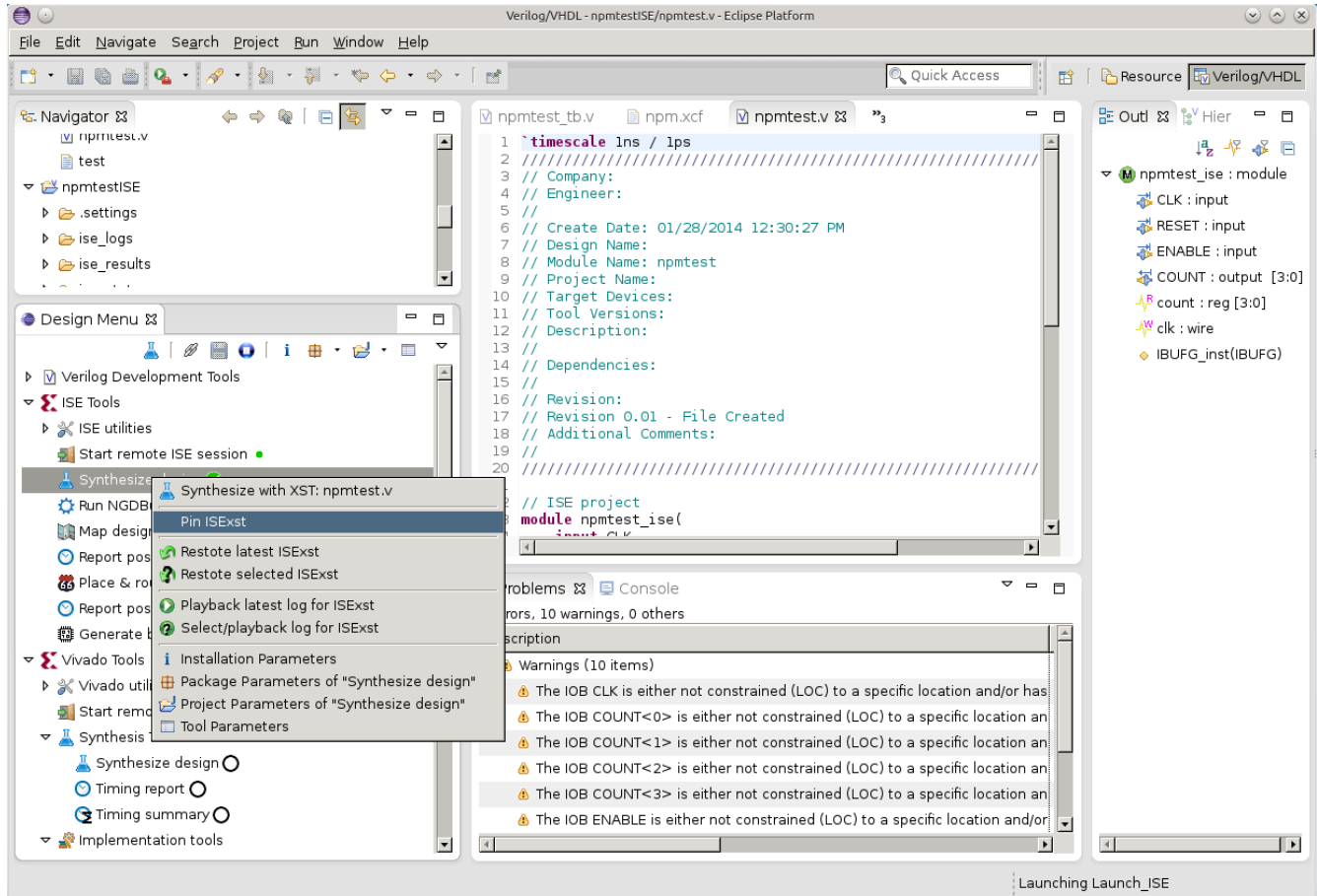






Fig.3. Tool context menu options


-  *Restore latest <tool-name>* - restore the latest saved state of the tool. This tool will be “pinned” (and marked with a pin icon) to prevent this tool to re-run even if it is older than the files/states it depends on;
-  *Restore selected <tool-name>* - open file selection dialog (appropriately filtered) to select one of the saved state files for the tool;
-  *Playback latest log for <tool-name>* - read the latest saved log file for the tool and run it through the same parsers/filters as specified for the tool, obeying current (possibly modified) parameter settings. This allows to restore Eclipse problem (error/warning/info) markers and to fine-tune the parser settings;
-  *Select/playback log for <tool-name>* - open file selection dialog and play back selected log file for the tool;

Tool states


There are two distinct types of the tools used in VDT:


- regular tools that involve a sequence of actions that terminate in in “success”, “failure” or “undefined” state and the tool sequencer waits for it to finish and
- console sessions – tools that do not terminate after the particular command is over and stay open, ready for the next commands. Many of the regular tools include sending some commands to one of the console sessions, capturing the output as log files and watching for the output keywords to determine success/failure and “finished” state. Current software version uses a “@@FINISH@@” pattern to be output (using “echo” commands for the shell scripts and “puts” for TCL).

Console sessions may be in a “new”, “failed” or “heartbeat” states, other states are applicable to the regular tools only. Tool states are indicated by an icon in the tree view, shown to the right of the tool label.


 *New* – tool that never ran before (or the tool history was erased with “clear tool state” command). Console session tools are always marked as new when the program session starts (regular tools preserve their state between sessions through the project persistent properties) or the related console window is manually closed.

- *Heartbeat* – this animated icon indicates that the console session program is up and running, ready to receive commands from the regular tools.


 *Running* – this animated icon is shown when the tool is running at the moment.


 *Waiting* - this animated icon is used for the tool that was selected for launch, but can not run yet as some other tools must run first to satisfy the dependencies of the selected one.


 *Success* – tool successfully finished execution.

 *Success (“dirty”)* - tool successfully ran before, but now some of the parameters values it depends on are changed or the files/states it depends on were modified after it ran.

Note: relevant parameter modification is determined by calculating hash codes of the command strings excluding parser parameters, so a) modification of the parser parameters does not make a tool “dirty” and b) restoring parameter values to the same as when the tool ran removes the “dirty” status from that tool.

 *Failure* – tool failed. Launch sequence is immediately terminated without waiting for the “@@FINISHED@@” signature, so actual tools may still be running. You may verify the tool finished by watching the appropriate console.

 *Undefined* – both success and failure signature sequences are provided in tool command line description, but none of them was detected in the tool output before the “@@FINISH@@” marker.

 *Pinned* – the tool finished with success and is in “pinned” state, so tool sequence will not try to re-run it even if the source files, states or parameters it depends on have changed since. This state can be individually set from the tool context menu, globally from the horizontal tool-bar. This state is automatically assigned when the tool state is manually restored from the context menu.

There is also blinking “Success” variant of the state – it indicates that the current state is in the process of being restored from a previously created snapshot or the tool has finished execution with success and now is creating and saving a snapshot of this state.

Automatic tool sequencing in VDT

When one of the tools is launched it may trigger execution of multiple related related tools (if tool linking is not disabled in the horizontal tool-bar):

- First VDT analyses the states and files this tool depends, and if some dependencies are not satisfied it goes upstream to find the tool that is needed to run first to satisfy the dependencies
- Then before launching the found tool VDT verifies that required console session(s) is (are) open, and if not – starts it (them)
- If VDT finds that the current session state does not match the required one to run next tool, but the snapshot exists, it runs “restore” tool (if available) instead of running the tool itself
- When some tool finishes successfully (in the case of a failure the tool launch sequence is terminated immediately), VDT looks if it has a “save” tool and auto-save is enabled (or manual save button on the horizontal tool-bar is activated). If it is the case, the save tool runs.
- After a tool finishes successfully, and there is no “save” tool left (not available or already ran) VDT looks if there are any enabled “report” tools – tools that do not change the state of the console session – such tools can run in any order without the need to create/save snapshots. If multiple such tools are found VDT determines the sequence of running the tools granting specified priority.
- When there are no save and report tools left and the originally specified tool is still in the wait state, VDT repeats the process of determining if any other tools need to run first. If none are left, VDT changes the tool state from wait to running, runs it and then, if the state has changed considers save and report tools to launch. That concludes the tool launch sequence.

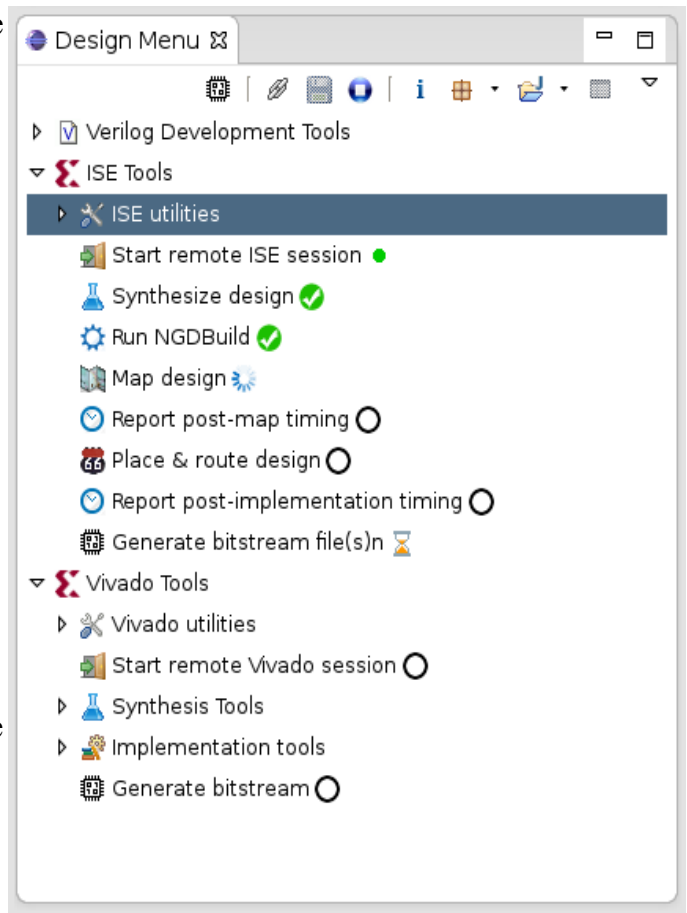


Fig.4. Running a tool sequence

Using playback feature to analyze tool output

VDT provides flexible means to parse tool output and integrating the results with the embedded editor so the detected problems can be shown in the source code context. These features are designed to be suitable for different tools, so there is nothing tool-specific in the VDT plugin core code. Parsing of the tool output is not a trivial task, even the same tool displays similar information in different formats. So the overall parsing is split into two parts:

- Final parsing is performed using VDT plugin code that relies on Eclipse problem reporting and navigation with additional tool-agnostic regex-based filtering to access editor object database

- external pre-processor that can be easily modified (or replaced) by the user.

Current (and rather simple) version of such parser is written in Python for synthesis tools of the two Xilinx products: Vivado and ISE. In the simple cases tool includes source file name and line number, so converting it to a link to the source location can be achieved with just regular expressions, but significant part of the tools output does not provide such information – just a reference to instances, modules and ports of the design, that also do not have direct match to the source because of the design optimization. Additionally it is common to see multiple similar lines of the output notifying of say automatic removal of the individual bits of some bus – each on its own line, making it difficult to read - some removal is OK (like when you use just some range of the wider address bus), but removal of others indicate a real problem. Python script while not having access to the internal object database of the editor can easily handle consolidation of the messages related to the same bus and formatting the object reference in a consistent manner to be parsed by VDT (there is still possible to customize object format recognized by VDT through several regex parameters).

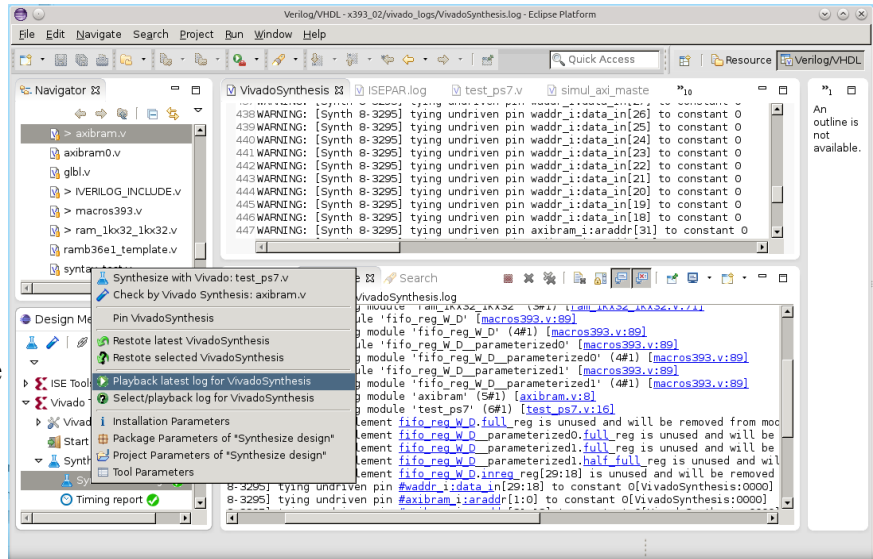


Fig. 5. Processing output of Xilinx Vivado synthesis tool

Two illustrations: Fig.5 and Fig 6 show the output of the same design synthesized with the different tools, both have the raw tool output shown in the editor panel instead of the source code. Multiple messages about the different bits are compacted in the bottom-right (console) panels.

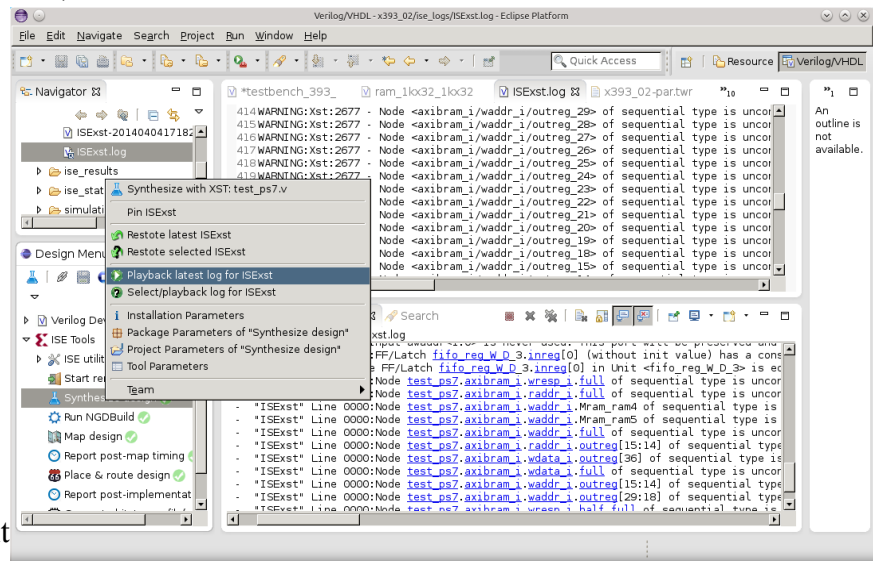


Fig. 9. Processing output of Xilinx ISE (XST) synthesis tool

VDT recognizes module, instance and port references according to the provided regular expressions and splits hierarchical names into segments (where possible). Then it tries to locate the objects in the editor database, starting with literal match, then allowing provided regex for suffixes that may be automatically inserted by the tools. And finally broadening matches by allowing any suffixes. The matched portions of the hierarchical names have links attached, so a signal may be traced through the hierarchy, the problem marker is attached to the last segment definition in the source code.

When the actual tool is running, the raw output is normally recorded as a log file and can be later fed to the same external parser. This is rather convenient as the same output console is reused by multiple

tools and the contents is regularly overwritten when tools are running in a sequence. It is also useful when working on a custom parser modification - external script code or fine-tuning regular expressions for filtering.

Configuring a remote server to run external tools

Current configuration of the tools made for the GNU/Linux distributions uses ssh and rsync programs to communicate and launch programs on the remote host. This is not a part of the VDT plugin itself, so it should be possible to use other programs when host and/or client computers run different operating systems. Initial release only supports GNU/Linux on both computers.

It is possible to run tools on the same physical computer as the development environment, in that case the remote IP will be 127.0.0.1 (localhost) and the default user name is the current OS user name.

Configuration of the remote access requires two steps to set up password-less ssh using key pairs:

- Generating keys on a client computer (should have empty passphrase) and
- Providing the generated public key to the remote host

First step can be automatically performed by the VDT, you just need to setup package parameters as shown on Fig.7 – host IP and the user name (defaults are 127.0.0.1 and the current system user)

Parameter	Value
Remote Host IP	192.168.0.126
Remote user name	elphel
Vivado release	2013.4
Vivado root	/opt/Xilinx/vivado
ISE release	14.7
ISE root	/opt/Xilinx

Fig.7. Package parameters for remote server setup

With the username and host IP configured you may use “Generate public key” sub-action in the context menu of the “Start remote ISE session” or “Start remote Vivado Session” as shown as an example on Fig.1.

Next step - “Setup connection to [<username@host_ip>](#)” requires entering your user password, and there is no pre-configured way to do it in the VDT plugin on most distributions. It requires a program *ssh-askpass* that is not universally installed on the systems. In *ubuntu GNU/Linux you may run in the terminal:

```
sudo apt-get install ssh-askpass
```

and enter your password when prompted.

And then run “Setup connection to [<username@host_ip>](#)”. You may try that action first – if *ssh-askpass* is not available VDT will output the recommendations in the console view. If the program is found then a separate window asking for a password to the remote host will open.

Alternative way is to run the following command

```
ssh-copy-id <username@host\_ip>
```

from a system terminal manually (replacing [<username@host_ip>](#) with the actual username and IP). It will ask for the password, and if successful you (and VDT) will be able to connect to the host server from the current client automatically without entering a password – that would not be practical with VDT that regularly connects multiple times when running each tool.