Blue Pearl Software

Visual Verification Suite Command Line Tcl Operation

Introduction

Blue Pearl Software's **Visual Verification Suite** (VVS) operates from either its own Graphical User Interface (GUI) or its Command Line Interface (CLI). The goals of this white paper are to introduce you to the CLI mode of interacting with the tool, and to point you in the right direction for further information if necessary.

The Blue Pearl GUI was designed and architected to provide only functionality that the CLI can do. Clearly, this does not include debug and visualization. However, **every** option, **every** setting and **all functionality** in analyzing the HDL design, is done via Tcl settings. If the GUI can do it, so can the CLI.

The ability to move back and forth between the CLI and GUI is seamless. It enables the automated nature of a CLI flow, with the ease of use, and advanced debugging capabilities of the GUI.

The basic interface to VVS is the Tool Command Language (Tcl). This paper is not designed to train you in Tcl. For further information on Tcl, please see the website http://www.tcl.tk. This paper will give you a solid understanding of the Command Line flow, as well as strong building blocks for running the tool from the command line.

The assumption for this paper, is that Blue Pearl has been installed into <installation>. Thus, starting the tool in command line mode with no other options:

```
$<installation>/BluePearlCLI
```

will leave the user at the BluePearlCLI prompt:

```
BluePearl-01%
```

Note, for those running in the Linux operating system, you are actually running a Shell Script that verifies the packages installed on the system, as well as setting up necessary environmental variables.

Throughout this document, example commands run from the operating system prompt will be prefixed with a $.

There are about 30 command line arguments to the BluePearlCLI executable. For detailed information on the options, you can enter:

```
$BluePearlCLI -help
```

At this point, it is useful to discuss two other command line arguments. The `–tcl` option specifies a file containing Tcl commands. You can also specify a single string to be executed by the system using the `–e` option. It is permissible to use multiple instances of both the `–tcl` and the `–e` options.

Of note, the Tcl command `exit` is not executed automatically. It is quite common to mix the two arguments as follows

```
    $BluePearlCLI -tcl projectsettings.tcl -e "BPS:run analyze" -e
exit
```

The above command will source the projectsettings.tcl file, then run the analysis on the project and exit. Note the required quotes around "BPS::run analyze" which are required due to the space in the command.

In addition to standard Tcl commands, there are also approximately 180 Blue Pearl Tcl commands and approximately the same number of Blue Pearl configuration variables. These are detailed in both the *Blue Pearl Tcl Reference Manual (<installation>/doc/BluePearlTclReference.pdf) and the Blue Pearl Analyze Reference Manual (<installation>/doc/BluePearlAnalyzeReference.pdf).*

This paper will cover a subset of these commands and configuration variables. Besides the Tcl Reference manual, you may refer to Section 3 of the *Blue Pearl Analyze Reference Manual.*

Getting Help from the BluePearlCLI prompt

All Blue Pearl Tcl commands and configuration variables have help available from the prompt. This includes the `help` command itself. All Blue Pearl commands are easily recognizable by their namespace prefix "BPS::" (for example BPS::run). The only exception to this is the command `help` itself. Additionally, all Blue Pearl commands provide the command argument "-help" providing detailed help on the command and its arguments.

For example, running the command `help -help` returns the following information:

```
BluePearl-01% help -help
'help' [-verbose='false']
[-type=<(All|Commands|Messages|Checks|Packages|Extras|Variables)>='Commands']
patterns=*
     Get help on commands
Arguments:
     -verbose - boolean     - Show hidden commands as well
     -type    - string      - Help Type
     patterns - string list - List of patterns to match
```

The default type of help that will be searched for is Commands, meaning help will return the information for any command that matches the pattern requested. You can also display help for commands, messages, checks, packages, extra information and the Blue Pearl configuration variables.

Setting up the tool using bps_setup.tcl

Both the Command Line and GUI applications load initialization files from the user's system. This allows the user to override previously existing defaults. There are slight differences between Linux and Windows, however the general flow is as follows:

- Load the <installation>/bps_setup.tcl file. This sets the defaults as set by Blue Pearl.
- Load the bps_setup.tcl from the systems Program Data directory for Blue Pearl.
- For windows, Load the bps_setup.tcl from the Local Application data directory.
- Load the bps_setup.tcl from the user's documents directory.
- Load the bps_setup.tcl files found by looking in the BPS_STARTUP_SEARCH_PATH environmental path. This variable is a semi-colon (;) separated list of directories on windows, and a colon (:) separated list of directories on Linux.
- Load the bps_setup.tcl files found in the search path specified in the **Preferences** page of the GUI.
- For the CLI, from the current working directory. Note for the GUI, the current working directory is not checked.

The command `BPS::get_startup_search_path` will return the path of all directories search for the startup files. For further details, please reference the *Blue Pearl Analysis Reference Guide,* Section 2.1 Default Setup Control File Format.

Loading a design

The first step in loading a design into the Blue Pearl CLI tool is to specify the HDL source files for your design.

There are two methods of setting up the source files. The first is to use the Blue Pearl specific Tcl command `BPS::add_input_file`. This method gives you the most flexibility to setup your input files. It allows you to create custom Compilation File Units for System Verilog, overriding the file type by file or by list of files. For instance:

```
    %BPS::add_input_file -work svwork -type {System Verilog 2005}
{top.v mid.sv}
```

adds the two System Veriog files top.v and mid.sv to the end of the list of files to be loaded. It also sets them to be compiled into the svwork library, and read in using the System Verilog 2005 language ruleset.

Another method of specifying input files, is to use the command line directly. Following the Verilog-XL standard, BluePearlCLI supports the -f <file> argument. The file is expanded into command line arguments. Nested -f files are allowed, and all HDL source files with relative paths are searched for using the location of the -f file they were loaded from. There is no difference between using the -f methodology and simply placing all your HDL files on the command line directly. There are a several advantages to using the -f to call the CLI. Firstly, it allows the user to store the command line arguments for later use. This also has the side advantage of allowing 3rd party tools to use the same file for their execution. Another advantage is to prevent hitting the various OS limits on command line argument length.

Blue Pearl will ignore unknown command line argument switches, allowing you to use a -f file originally designed for a 3rd party tool with relative ease.

Unrecognized arguments that are not switches (starting with a – or +), will be assumed to represent a file name for an HDL source file. For this reason, a simple list of file names works equally well for Verilog, SystemVerilog, and VHDL files.

Along with setting your source files as arguments to the CLI, many language options are available. If you run the CLI's help, under the category *RTL Language Options*, the options `+define+`, `+incdir+, -y, -v` and so on are listed. This was done to ease the process of reading an existing -f file. It should be noted that a -f file intended for another tool often includes non-synthesizable files which cannot be read by the Blue Pearl VVS tool. Up front effort is required to delete the names of files such as test benches and simulation models from the -f file.

The Blue Pearl Tcl command used to actually run the load operation is `BPS::run load`. Note, if you wish to avoid being left at the BluePearlCLI prompt, the last Tcl command must be `exit`. Lastly, a semicolon can be used to delimit Tcl commands. So, a simple means of loading a design from the command line might be:

```
    BluePearlCLI –f hdllist.f –e "BPS::run load;exit"
```

If a design is self-contained, a complete specification of synthesizable source files should be sufficient to load the design. However, many designs, including those needing the FPGA libraries that are built into the Blue Pearl VVS tool, may need other Tcl commands in order to load. Information on setting FPGA libraries is available from the Tcl promp using `help set_altera`, `help set_xilinx`, or `help set_microsemi`.

Load vs. Analyze

The two phases involved in running the Blue Pearl VVS tool are known as *Load* and *Analyze*. The Tcl command for running Analyze is `BPS::run analyze`. Briefly, the *Load* phase parses the design and creates the underlying database, as well as performing linting and other basic checks. The *Analyze* phase performs more complex analysis, including any CDC and SDC checks.

If your design has already been loaded, there is no reason to load it again from the original HDL. Blue Pearl will load the design from the database stored in the results directory when you execute `BPS::run analyze.` Before the design is loaded from the database, the load settings will be checked to make sure the design has not changed. This includes the design files themselves, as well as changing the top module/entity, or many of the configuration options the effect the interpretation of the design.

If the design was not previously loaded, you can skip load phase and simply run `BPS::run analyze`, or if you wish to force a load during the analyze phase run `BPS::run analyze -forceload`. This will tell the tool to ignore any existing databases and load the HDL design from the source.

If you plan on debugging and analyzing the results in the GUI, it is recommended you run `BPS::run` with the `–generate_module_database` argument, which creates the additional databases used for Visual Verification.

This brings up another CLI argument, `-output <directory>`. This option allows you to direct the tool where to place all results. This includes text report files, our internal databases as well as our support files. For the CLI, it is expected that the user will want to direct the output to their needs, the default directory is <pwd>/bluepearl.results. The GUI uses “Results” with a capital R as the default output.

Setting up packages, checks and configuration options

In Blue Pearl’s terminology, *checks* determine what analysis is performed on the design, and *messages* are the method of reporting the results to the user. *Packages* are user definable, and simply group the checks. They are quite often set in the bps_setup.tcl setup file(s). You can create new packages using the command `BPS::add_package`. You can also set Blue Pearl to report on a given package’s results.

Checks enable various phases of analysis in the system. Some checks are associated with the Load phase and some with the Analyze phase. Most checks cover one and only one message, but there are checks that output multiple messages. There are numerous messages, mostly having to do with serious problems or the running of the tool itself, that are not controlled by any check. There are also a few messages that may be covered by multiple checks. Checks often have configuration options. For example, the ITE_DEPTH check analyzes if/then/else statements deeper than a threshold set by the configuration variable `ite_depth`, the default value is 3 for this option, but can be set by the user. The Tcl commands to set this check and its associated configuration variable are, respectively:

```
BPS::set_check_enabled ITE_DEPTH -enabled true
```

and

```
set ite_depth 5
```

The help file *<installation>/doc/BluePearlAnalyzeReference.pdf* is the best resource for all the available checks, messages, Tcl commands, and configuration variables.

You can also access the built in help for each command using `help -check <checkname>` or `help -message <messagename>`

Message Waivers

All messages generated by running Load and Analyze are written into an SQL database. When a check is enabled, all messages will be reported to the log and stored in the SQL database. But what if only a few carefully considered instances of a certain message must be excluded from the results? This is the reason for waivers.

Waivers are a very powerful mechanism for adjusting your analysis results. You should think of them as a "big stick" not to be used frivolously.

A waiver adds a notation to one or more specific instances of a particular message in the database. There are actually two possible notations, *Won't-fix* and *Must-fix*. (Yes, we counter-intuitively call it a *Must-fix waiver*. The labeling mechanism is the same.)

Waivers are stored in xml format in a file given the default name of *waivers.xml*. The Tcl command used to create a waiver is `BPS::add_msg_waiver` and it requires at least four parameters, two of which are regular expressions. It is assumed that the reader is familiar with regular expressions; for more information on regular expressions please see http://www.regular-expressions.info. The following arguments are required to add a message waiver:

- The message ID
- Either the filename or the module name associated with the message(s)
- You must declare that the waiver is either Must-fix or Won't-fix
- And lastly, you must declare a reason for the waiver.

This string specifying "why" is recorded in the xml file along with "who" created or edited the waiver and "when" it was done. Optionally, you may wish to specify, also as regular expressions, an object name and/or the text of the message.

```
    BPS::add_msg_waiver -msgid BPS-0001 -module ip -wontfix -reason
"Cannot fix ip blocks"
```

This command will create a new wont-fix waiver, that waives all BPS-0001 messages inside module ip, with the reason "Cannot fix ip blocks".

Blue Pearl uses PERL style regular expression syntax; for more information on PERL regular expressions, please visit http://www.perl.com. It should be understood why we call waivers a "big stick". If any of your regular expressions are too broad and encompassing, it is possible to waive many more messages than you originally intended to. Therefore, when using waivers it is recommended that you run the waivers analysis, as well as turn on the must-fix and wont-fix waiver summary.

```
BPS::set_log_options -mustfixsummary -wontfixsummary
BPS::analyze_msg_waivers -report -wontfix -mustfix -overlap -superfluous
```

The first command turns on the generation of both the must-fix and wont-fix summary report at the end of a call to `BPS::run`. The second command runs a waivers report showing the effectiveness of the waivers, including the unused waivers as well as waivers that overlap.

Enabling this functionality will allow you to confirm the message count being waived, as well as telling you the number of remaining must-fix waivers.

It is also possible, using Tcl commands, to edit, disable, or delete a waiver. Disabled waivers can be enabled at a later time. These operations change the xml file and cannot be undone. In order to do any of these operations, you must first use the command `BPS::get_msg_waivers` to obtain an ID number for the waiver in question. Once you have the ID number, you can use `BPS::delete_msg_waiver` or `BPS::disable_msg_waiver` or `BPS::enable_msg_waiver` or `BPS::edit_msg_waiver` to delete, disable, enable, or edit the waiver, respectively. In the case of disabling, enabling or editing, you are required to supply a reason for the changes made.

CDC Waivers

The Blue Pearl VVS tool also includes a similar set of commands to waive CDCs. These waivers are stored in the same xml file as the message waivers, but allow you to waive a Clock Domain Crossing, and all messages associated with that crossing. They also waive the CDCs from the *Clock Domain Crossing* window in the GUI.

```
    BPS::add_cdc_waiver -src_obj dff1 -reason “Actually a False
Path”
```

This command will create a cdc waiver, that will waive all CDC issues with the source object named dff1, with a reason of “Actually a False Path”.

Embedded disabling of Messages in Verilog or VHDL source files

While waiving messages is very powerful, it is not uncommon that you want to simply disable messages from being reported for small isolated blocks of the design. You **cannot** enable a message if the check is not enabled for the system. Also, you **cannot** disable error messages, syntax errors, or any message escalated to the severity of error.

You can disable/enable specific messages in the source file for portions of the HDL code by specifying:

```
    // bluepearl disable <message_id_list>|all
```

to disable the check with message ID code message_id or:

```
    // bluepearl enable <message_id_list>|all
```

to enable the check with message ID code message_id. If “all” is specified instead of a list of message IDs, all checks are disabled.

For VHDL code, the equivalent pragmas are:

```
-- bluepearl disable <message_id_list>|all

-- bluepearl enable <message_id_list>|all
```

Example:

To disable message ID BPS-0262 for a portion of the Verilog RTL code, enclose the code in comment directives as follows:

```
// bluepearl disable BPS-0262
```

<Verilog RTL code for which checking is to be disabled>

```
// bluepearl enable BPS-0262
```

The "all" message_id can be used to enable/disable all checks, and is the default.

There is one key difference between locally disabling a message and waiving it. When a message is waived, it is still stored in the database for later viewing, and will be reported and viewed if the waiver is deleted or disabled. However, if a message is disabled in the HDL source code, it is as if the check is locally disabled for that particular area of code. The message is never stored, and cannot be viewed later.

From CLI to GUI and back again

Along with the CLI, The Blue Pearl Visual Verification System includes the GUI, also known as the **Visual Verification Environment** or **VVE**. Broadly speaking, the GUI has two functions. The primary role is to display the VVS analysis results in an accessible, intuitive, and aesthetically pleasing manner. The secondary role is to act as a script generator for running the CLI executable. The Visual Verification System generates -f and Tcl script files, which it then uses to call the CLI tool. These files are available to be used directly by users running in command line mode, and the Tcl script file can be read back into the GUI so that results generated using command line mode can be examined using the GUI.

We recommend, but do not require, that users start a new design using the VVE. Once this is done, the results directory will include (among many others) three important files. They are:

1. bluepearl.runme.tcl
2. <*design_name*>.settings.tcl
3. <*design_name*>.bluepearlgenerated.f

The VVE executes the command :

```
BluePearlCLI –f <design_name>.bluepearlgenerated.f.
```

The -f files calls the two .tcl files using the `–tcl` option. The user is free to execute the same command or variations thereof.

Quite often, you might receive a project from a co-worker who packaged up the design without realizing the relative paths had changed. Every Blue Pearl run prints out the following information: where the tool was run from (PWD), the actual command line arguments, and the effective command line arguments (with all the -f files fully expanded). This allows you to reproduce the results of your co-worker, or to rebuild the relative paths as necessary.

```
--BPS-0730: Status: Running from: D:\testarea\Demo
--BPS-0730: Status: Command Line Expanding -f file:
'D:/testarea/Demo/Results/OR1200_Xil_FPGA.bluepearlgenerated.f'.
--BPS-0730: Status:
D:/testarea/Demo/Results/OR1200_Xil_FPGA.bluepearlgenerated.f
Expanding -f file: 'D:/testarea/Demo/bluepearl.f'.
--BPS-0730: Status: Unprocessed CLI Arguments: -f
Results/OR1200_Xil_FPGA.bluepearlgenerated.f
--BPS-0730: Status: Effective CLI Arguments:
   ./source/verilog/reconv_with_sync.v
   ./source/verilog/slow_to_fast_mcp.v ./source/verilog/top.v
   ./source/verilog/vfp.v ./source/verilog/mux_synch.v
   ./source/verilog/OR1200.v
   +incdir+./source/include
   -top OR1200
   -output D:/testarea/Demo/Results
   -tcl D:/testarea/Demo/Results/OR1200_Xil_FPGA.settings.tcl
   -tcl D:/testarea/Demo/Results/bluepearl.runme.tcl
```

All Tcl script files generated by either the Blue Pearl CLI or GUI will contain two important variables for allowing you to migrate the scripts from one location to another. The first is `BPS::project_rel_to_dir` and the other is `BPS::project_results_dir`. All files in the generated script file will be relative to the first variable. If you are running from a different relative location, you **must** update this variable. For instance, if the project's HDL has been moved into the sub-directory "ipblock1", you simply change the command to:

```
set BPS::project_rel_to_dir ipblock1
```

and all input files will be loaded correctly.

The other variable allows the GUI to load the Tcl script file and determine where the results have been saved.

Another means of creating a comprehensive Tcl script file is by running the Tcl command:

```
BPS::save_project filename.tcl
```

The resulting file is similar to the *design_name*.settings.tcl file created by the VVE. Either of these files can be read back into the VVE using the following steps:

1. Start the GUI
2. Execute the command *Setup Design* → *Open Design…*
3. In the resulting *Open Design Project File* dialog, change the file type from *.bps* to *.tcl*.
4. Click *Open*

One of the parameters that must be known in order to run the tool is the name of the output directory. This is set in command line mode by the `-output` command line option, and the default value is *bluepearl.results*. There is no corresponding Tcl command or configuration variable, since this value must be known before the Tcl shell is initialized. You can, however, access this value to use in your Tcl script using the command `BPS::get_outputdir`.

The state of all settings in the VVE is recorded in an xml-format file which is called "*design_name*.bps". As in command line mode, one of the parameters set in the BPS file is the name of the output directory. However, in the VVE the default value is *Results* (always with a capital *R*). As with other values, this can be changed by the user.

There are three requirements for using the VVE to access results generated using the CLI.

1. No other CLI or GUI session may be active.
2. The name of the results directory generated by the CLI must match that in the BPS file.
3. The BPS file location must be the same as the location of the results directory (not within the results directory, but at the same level and location as the results directory).

If these requirements are met, opening the BPS file using the VVE will display the results from the CLI session.

If you plan on going back and forth between the GUI and CLI on a regular basis, it is recommended you run the CLI in the same manner the GUI does.

Specifying variables and parameters

It is a common practice for designers to use parameters and generics to control their designs. It makes the HDL code more flexible. These must be set in order to successfully load your design into the Blue Pearl VVS. There are two ways to specify parameters. It can be done from the command line, or it can be done using a Tcl command. **Please note that if you use both methods, the Tcl command will be predominate**.

The command line option is `-g` *`param_name=value`*.

The Tcl command is `BPS::add_parameters {`*`param_name value`*`}`.

Message Analysis

Both the GUI and CLI provide easy methods to analyze the results of the run. The command `BPS::result_run_compare` allows you to compare two runs. The command `BPS::get_result_run_summary` reports the summary of the messages for the requested run.

Note, many Blue Pearl Tcl commands return information (for instance, the message analysis commands listed above). When running from the Tcl environment, it is quite often necessary to programmatically analyze these results. For this reason, the default behavior for most Blue Pearl commands is to return a Tcl object, not to return a string that the user would then have to parse.

If you wish to have the return value presented in human readable format, these commands will have a -report option.

For instance:

```
%BPS::get_result_run_summary
{{} 0 {{Total 6} {Errors 0} {Warnings 0} {Informationals 0} {Comments 6}}
{{BPS-0730 {} 6 comment {Status: %s}}}}
```

by default returned a tcl list of data. However the same command with the -report argument set returns a human readable table:

```
%BPS::get_result_run_summary -report
Message Summary:
=================================
    Message ID:  BPS-0001: 73 warnings - Missing Else-Block for IF statement.
    Message ID:  BPS-0002: 1 warning - No default assignment for case statement.
    Message ID: VERI-1927: 21 warnings - port %s remains unconnected for this instance
=================================
** Maximum message limit was reached for some messages.
** You can increase the message limit on the 'Design Settings -> Log Options' dialog or by
setting 'default_message_limit' in TCL.
=================================
Summary:
    Total Number of Messages: 1808
    Number of Errors:         0
    Number of Warnings:       1238
    Number of Informationals: 305
    Number of Comments:       265
```

Reporting

Quite often, when running Blue Pearl as a validation tool against a known standard, such as DO-254 you want to confirm "Zero Issues." Typically, only messages that show an issue are counted, leaving you to assume all is OK if no messages were counted.

However, you can report on a package, using the command `BPS::get_result_run_messages -package`, which will report on all messages covered by checks contained in that package. The command:

```
BPS::get_result_run_messages -package {DO-254}
```

will report on how your design compared to the DO-254 package provided by Blue Pearl VVS.

Blue Pearl VVS also has numerous reports that are generated during the Load and Analyze phase of the run. These can be enabled via the BPS::set_report command. For instance:

```
BPS::set_report {Ports} -enabled
```

will turn on the generation of the Ports report.

If you have the Management Dashboard license, you can also get text based histogram reports of the history of your design using the commands `BPS::messages_histogram`, `BPS::cdcs_histogram` and `BPS::wiavers_histogram`.

Another powerful aspect of the Management Dashboard license is the Design Signoff report. This command reports on the status of the configuration and results of your design using the command `BPS::report_design_signoff_checklist`

Conclusion

The Blue Pearl Visual Verification System contains a comprehensive set of Tcl commands to load, configure and analyze your HDL designs. The Tcl commands are also available to perform post HDL analysis message reporting. The ability to seamlessly go between the CLI and GUI allows automated server runs as well the debugging of the results previously run via the CLI.

Further Reading

- Blue Pearl Tcl Reference Guide: <installation>/doc/BluePearlTclReference.pdf
- Blue Pearl Analyze Reference Manual: <installation>/doc/BluePearlAnalyzeReference.pdf
- Tcl online reference: http://www.tcl.tk
- PERL online reference: http://www.perl.com