

Using Tcl to Implement an Efficient Synthesis Environment

Tim L. Wilson

tim.l.wilson@intel.com

Rodney Pesavento

rodney.pesavento@intel.com

Intel Corporation

ABSTRACT

Since the 1999.05 release of Design Compiler, the Tcl language has been supported in `dc_shell`. We have used this language to implement a synthesis and static timing environment for Design Compiler (DC) and PrimeTime (PT). Using Tcl allows us to share setup scripts, constraint scripts and other scripts among the DC and PT tools for both pre-layout and post-layout compiles and timing analysis. We were able to convert our existing environment of C-shell scripts, DC shell scripts, and Perl code to use Tcl. We were also able to integrate new functionality for improved synthesis flow control and analysis. Basing our environment on Tcl gave us a high-powered, clean, extensible, and portable environment. The paper will show how we structured this environment, overcame some of the pitfalls in converting to Tcl, how it hooked into our gmake driven synthesis flow and give some Tcl coding tips.

1.0 Introduction

Why Tcl? The Tcl language has many benefits over the dcsh language. One benefit is that Tcl is a standard language supported by other EDA tools. There are good looping and control constructs in Tcl and the interpreter is very flexible. Tcl lends itself to modular programming techniques, which includes procedures and structures. It also has built-in features that make it easy to document procedures and variables. In addition there are many good reference books on Tcl.

As we began looking at moving our synthesis environment to Tcl, we had a variety of goals we wanted to achieve. One of the major goals was to reduce the complexity of the environment. Earlier environments used dcsh code, which included other dc_shell scripts, which called Perl scripts, which in turn called UNIX shell commands or C-shell scripts. Our hope was that most, if not all of these functions could be replaced by Tcl code. Another goal was to allow sharing of top-level constraints and setup since both PrimeTime (PT) and Design Compiler (DC) would be running Tcl. In the past we had to maintain two sets of scripts, one for DC and another for PT. Finally, we hoped to reduce the number of files and amount of setup needed to run synthesis jobs. This would simplify training and make it easier to furnish our environment to Synopsys in order to submit test cases that we may have for bug fixes. Our main goal was to develop a modular open-source framework for synthesis that could be used across multiple projects and sites. Developing this new framework which we call TOPS (Tcl Oriented Procedural Synthesis) would be a major rewrite of our current synthesis environment.

2.0 Major Features of TOPS

- **Library Flexibility** -The ability to change libraries at run time is one of the major features of TOPS. This is accomplished by assigning all the library dependent variables in a single file called proj_library.tcl. These variables include the target library, operating conditions, default drivers and loads, time scale units, etc. This assignment is done because some library dependencies must be setup BEFORE a design is read into memory (such as the target library) and others must be setup AFTER the design is in memory (such as operating conditions).
- **Language Independence**- The environment supports mixed language designs and uses the filename extension (*.v for Verilog and *.vhd for VHDL) to determine how to analyze and elaborate the file.
- **Project and Target Level Flexibility** - We introduced the concept of project level procedures and target level procedures. The project level provides a consistent setup for things like library setup, operating conditions, default compile strategies, clocking constraints, reports, format of netlists and common output files needed for layout support. The project level scripts and procedures can be replaced by target level scripts and procedures to change or modify the synthesis run, but are not necessarily needed for initial synthesis.

3.0 Tcl Script Files Overview

3.1 Project Level Script Files

The TOPS script files and project level Tcl files reside in a directory referenced by the UNIX environment variable \$PROJ_SYN. There is another key environment variable called \$PROJ_LIB which refers to the area that contains the target library DB files and Synopsys cache directory and files. The main Tcl files associated with this environment are:

- home.synopsys_dc.setup_tcl
- proj_dc_setup.tcl
- proj_library.tcl
- util_procs.tcl
- proj_main.tcl
- report_procs.tcl
- proj_procs.tcl
- proj_constraints.tcl
- proj_reports.tcl

Below is a brief description of the files in this area that are used in the environment.

- *home.synopsys_dc.setup_tcl* – A template file that is copied to a user's home directory as ".synopsys_dc.setup". This file is sourced by dc_shell and in turn sources proj_dc_setup.tcl.
- *proj_dc_setup.tcl* – Sets up the search_path, dc_shell variables, synthetic libraries, cache locations and other project variables.
- *proj_library.tcl* – Contains all library dependent information needed by Synopsys. It can handle switching libraries based on a variable \$G_LIBRARY_NAME. The file sets up the target, symbol and link libraries and reads them all into memory. It also assigns the wireload and operating condition variables that are used after the design is read into memory. The following global Tcl variables are set up in this file. Values for the LSI_10K library are shown.

```
G_NS 1
G_MAX_TRANSITION [expr 3.0 * $G_NS];
G_MAX_CAPACITANCE 5.0;
G_MAX_DELAY 5.0;
G_DEFAULT_DRIVING_CELL {FD1S};
G_DEFAULT_DRIVING_CELL_PIN {Q};
G_DEFAULT_LOAD_CELL {ND2};
G_DEFAULT_LOAD_CELL_PIN {A};
G_DEFAULT_NAND_CELL {ND2};
G_MAX_LIBRARY ${G_LIBRARY_NAME};
G_MIN_LIBRARY ${G_LIBRARY_NAME};
G_MAX_OP_CONDITION {WCCOM};
G_MIN_OP_CONDITION {BCCOM};
G_DEFAULT_WIRE_LOAD {90x90};
G_CWLM_LIBRARY [list ""];
```

G_PROJ_DONT_USE "\${G_LIBRARY_NAME}/FJK*";

- *util_procs.tcl* – Sourced from the *proj_dc_setup.tcl* file and contains many global utility procedures. It contains the following procedures:
 - ❑ **P_elapsed_time** <mark> - Uses a time mark to calculate elapsed time from <mark>. Can be used to time execution of procedures or other functions.
 - ❑ **P_timestamp** - Prints a timestamp line including user name and machine name.
 - ❑ **P_get_srcline** <filename> <subfub filenames> - Extracts the CVS \$ID strings from the source file(s) arguments.
 - ❑ **P_source_if_exists** <filename> <caller> - Used to determine the existence of a file. If the file is not found, the procedure returns a zero. A caller variable was added to print the invoking script's name.
 - ❑ **P_exec_if_exists** <filename> <caller> - Executes a procedure if it exists in memory. If the procedure does not exist, a zero is returned, otherwise a one is returned. A caller argument was added to print the invoking script's name.
 - ❑ **P_get_license** <license_name> - Checks out the specified license, waiting if not available. A timeout error will occur after a pre-determined wait time.
 - ❑ **P_check4lic** <license_name> - Used to check for a particular license in memory.
 - ❑ **P_read_vhdl** <filename.vhd> - Read the VHDL source code files using the analyze command.
 - ❑ **P_read_verilog** <filename.v> - Reads Verilog source code files using analyze -f.
 - ❑ **P_read_top** <top level filename> - Used to read in the top-level block using analyze and elaborate commands (language neutral).
 - ❑ **P_analyze_pkg** <filename.vhd> - Used to analyze only VHDL packages.
 - ❑ **P_read_subfubs** <\${SUBFUBS list}> - This procedure handles sub-block files (SUBFUBS) by analyzing them into memory rather than compiling them and then reading the DB files into memory. This is to promote top-down compiles with sub-blocks. Used in the *proj_main.tcl* script.
 - ❑ **P_compilelib** <libname> <libsourcedir> - Used to compile a library of <libname> into the ./dbs directory from the source pointed to by libsourcedir. Called from a Makefile directly (i.e. `dc_shell-t -x "P_compilelib libname libsrc"`).
 - ❑ **P_log_filter** <log_path> <target name> - Filters the synthesis log file for information of interest such as errors, warnings & elapsed times.
- *proj_main.tcl* - The main script that controls the synthesis flow. It is sourced through the *dc_shell-t* command line using the `-x` switch. It expects the setup of the following *dc_shell* variables (*G_PROJ_SYN*, *G_SCRIPTS_PATH* and *G_DESIGN_NAME*) and then sources the following files:

```
${G_SCRIPTS_PATH}/${G_DESIGN_NAME}_dc_setup.tcl
${G_PROJ_SYN}/proj_procs.tcl
${G_SCRIPTS_PATH}/${G_DESIGN_NAME}_procs.tcl
${G_PROJ_SYN}/report_procs.tcl
${G_PROJ_SYN}/proj_reports.tcl
${G_PROJ_SYN}/proj_library.tcl
```

After proj_main.tcl sources these files, it calls the other procedures needed to complete the synthesis run. These include analyzing the files, setting up the constraints and compiling the design. After compilation is complete, proj_main handles writing the outputs such as the dbs and netlist as well as any reports.

- *report_procs.tcl*- This file contains generic report procedures which write out report files to a specified directory using a combination of the design_name and report_name as the file name. (e.g. Given a design name of cnt, the procedure call "P_report_area \$G_REPORT_PATH \$current_design" generates a file called cnt_report_area.rpt in the reports directory pointed to by \$G_REPORTS_PATH) Some of these procedures are only single line Synopsys commands, others may involve multiple steps such as ungrouping components and generating reports. Below are the procedures declared in the report_procs.tcl file, where the arguments <rpath> is the report directory name and <dn> is the design name.
 - ❑ **P_report_area** <rpath> <dn> - Performs report_area command
 - ❑ **P_report_check_design** <rpath> <dn> - Performs check_design command
 - ❑ **P_report_check_design_ungroup** <rpath> <dn>- Performs ungroup then check_design
 - ❑ **P_report_check_timing** <rpath> <dn> - Performs check_timing command
 - ❑ **P_report_check_test** <rpath> <dn> - Performs check_test command
 - ❑ **P_report_clock** <rpath> <dn> - Generated report with various clock report comands
 - ❑ **P_report_compile_options** <rpath> <dn> - Performs report_compile_options command
 - ❑ **P_report_design** <rpath> <dn> - Performs report_design command
 - ❑ **P_report_dont_use_cells** <rpath> <dn> - Generates list of cells with don't use attribute
 - ❑ **P_report_ignored** <rpath> <dn> - Generated list of ignored timing requirements
 - ❑ **P_report_timing_requirements** <rpath> <dn> - Performs multiple report_timing_requirement commands
 - ❑ **P_report_endpoints** <rpath> <dn> - Generates end point only timing margin report
 - ❑ **P_report_pins_per_net** <rpath> <dn> - Performs report_net command
 - ❑ **P_report_port** <rpath> <dn> - Generate report with port attributes
 - ❑ **P_report_reference** <rpath> <dn> - Generate reference list report
 - ❑ **P_report_reference_ungroup** <rpath> <dn> - Performs ungroup of design and then generates the report reference list report
 - ❑ **P_report_test** <rpath> <dn> - Generate scan information report
 - ❑ **P_report_unmapped** <rpath> <dn> - Report any unmapped logic in design
 - ❑ **P_report_vio** <rpath> <dn> - Performs report_constraint -all_violators command
 - ❑ **P_report_vio_max_cap** <rpath> <dn> - Reports all max_cap violators
 - ❑ **P_report_vio_max_delay** <rpath> <dn> - Reports all max_delay violators
 - ❑ **P_report_vio_min_delay** <rpath> <dn> - Report all min_delay violators
 - ❑ **P_report_ins2flops** <rpath> <dn> - Report timing from inputs to first flops
 - ❑ **P_report_flops2outs** <rpath> <dn> - Report timing from last flops to outputs
 - ❑ **P_report_ins2outs** <rpath> <dn> - Report timing from inputs to outputs
 - ❑ **P_report_flop_latch** <rpath> <dn> - Reports number of flops and latches in design

- *proj_procs.tcl* – Contains most of the project specific procedures that are changed to support a specific project. These include procedures to constrain clocks and resets, control the compile strategy, generated outputs, and generate report files. A full listing of these procedures is given below.
 - ❑ **P_proj_annotate_clocks** <min_uncertainty>- Removes insertion delay, latency and constrains clocks for post-layout analysis. Used for PrimeTime analysis runs.
 - ❑ **P_proj_clocks** - Used to setup the project level clocks and resets.
 - ❑ **P_proj_constrain_reset** <reset_name>- Used to set the constraints on the "reset" ports
 - ❑ **P_proj_constrain_clock** <clk_name> <period> <uncertainty> <insertion> {hier""} {r1 ""} {f1 ""} {r2 ""} {f2 ""} – where hier,r1,f1,r2,f2 are optional arguments.
This procedure is used to define and create a clock named "clk_name" and assign various clock attributes such as period, uncertainty and latency to the clock port.
Example: P_proj_constrain_clock {pciclk 1000.0 150.0 0 "" 0 400 0 0}
 - ❑ **P_proj_inst_cnt** - Count the number of cell instances in a design.
 - ❑ **P_proj_change_names** - Defines the projects change names rules (removes hierarchy separators and special characters from the netlist) and is included prior to the netlist being written out.
 - ❑ **P_proj_compile** - Defines the default project compilation strategy.
 - ❑ **P_proj_compile_top** - Used to compile the top-level netlist or the top of any block that contains modules not needing to be compiled.
 - ❑ **P_proj_operating_conditions** - Sets the default operating conditions using min/max conditions from the proj_library global variables.
 - ❑ **P_proj_precompile_scan** - This procedure sets the scan configuration command & defines the scan ports for the design.
 - ❑ **P_proj_postcompile_scan** - Used to insert the scan chain after compilation of the design.
 - ❑ **P_proj_outputs** <db_path> <design_name> <outputs_path> <srcline> {out_string ""} {out_type ""}
Generates the default outputs including db, sdf, sdc and Verilog gate level netlist.
NOTE: If out_string is null, the procedure will only output a DB file and Verilog netlist. If out_type is "layout", all layout related information is written.
 - ❑ **P_proj_ungroup_dw** - Used to ungroup the DesignWare components. This is useful to remove unconnected ports that may exist.
 - ❑ **P_proj_wire_load** – Sets up the default wireload model for the design. Assumes \$G_DEFAULT_WIRE_LOAD variable set from proj_library script
- *proj_constraints.tcl* - Contains the default constraints for the project including input/output delays, default driving cells and loads as well as other default project design and timing constraints. This was made into a separate file due to the fact that it could contain a large number of commands. This could be made into a procedure in the *proj_proc.tcl* file if desired.

- *proj_reports.tcl* – Contains a set of project specific report procedures to generate various reports from dc_shell. Some are only single line Synopsys commands, others may be more complicated involving multiple steps like ungrouping components and generating a report. The main procedure is **P_proj_reports**, which executes other report procedures defined in *report_procs.tcl*. The *proj_reports.tcl* file may also define project related reports that are not available in the *report_proc.tcl* file. The **P_proj_reports** procedure is called from *proj_main*.

3.2 Target Level Scripts

In the *proj_main* script there are placeholders where target scripts and procedures can be included into the default *proj_main* flow. This enables any individual target to replace the global project scripts. These target scripts can include the global project scripts and procedures they may be replacing in order to add-to the function of the global scripts. These target specific scripts are always optional and reside in the *<unit>/syn/scripts* directory. These target scripts can consist of any of the following files.

- *<target>_dc_setup.tcl* - Unit specific setup file. Used for additional library or variable setup. This file is additive and is sourced from the *proj_dc_setup.tcl* file upon startup of dc_shell.
- *<target>_procs.tcl* - Allows target level procedures to be defined. These can be used to define procedures to replace many of the target Tcl files listed below. i.e a procedure named **P_<target>_compile** can be used in place of a *<target>_compile.tcl* script.
- *<target>_main.tcl* – Target level main script, redefines the *proj_main* flow for synthesis. Program flow must be terminated from inside this file.
- *<target>_<library>_dont_use.tcl* – Target specific cells that cannot be used during synthesis.
- *<target>_clocks.tcl* - Defines the target specific clocks and resets.
- *<target>_constraints.tcl* – Sets up a target specific set of constraints.
- *<target>_exceptions.tcl* – Sets up target level timing exceptions such as multi-cycle paths
- *<target>_wire_load.tcl* – Loads a custom target specific wireload model.
- *<target>_compile.tcl* – Target specific compile script.
- *<target>_outputs.tcl* -Target specific outputs.
- *<target>_reports.tcl* - Target specific reports for critical paths, etc.

Although the target scripts are optional, most targets will usually have at least two scripts or procedures defined. One is for clock constraints and the other is for timing constraints (since most constraints are target specific). Common timing constraints for internal busses may be put into the *proj_procs.tcl* file and called by the target scripts or procedures. The execution priority is determined in the *proj_main* script and may be modified by projects as required. The default priority is the target procedure, then the target level script, and by default if the others do not exist, the project level procedure or script.

NOTE: Projects may decide to use either a single target level procedure file (*<target>_procs.tcl*) or individual target level Tcl scripts or both depending on personal preference.

Below is a table that shows the priority of execution from proj_main:

Description	Target Level Procedure- Located in <target>_procs.tcl file	Target Level Script	Project Level script or procedure
clocks	P_<target>_clocks	<target>_clocks.tcl	P_proj_clocks
constraints	N/A	<target>_constraints.tcl	proj_constraints.tcl file
\$lib_dont_use	N/A	<target>_<lib>_dont_use.tcl	(included in proj_library.tcl)
exceptions	N/A	<target>_exceptions.tcl	N/A
wire_load	P_<target>_wire_load	<target>_wire_load.tcl	P_proj_wire_load
compile	P_<target>_compile	<target>_compile.tcl	P_proj_compile
outputs	P_<target>_outputs	<target>_outputs.tcl	P_proj_outputs
reports	P_<target>_reports	<target>_reports.tcl	P_proj_reports

Below is a sample from proj_main.tcl that shows how the above order is structured in Tcl:

```

echo " "
echo "#####"
echo "#           Synthesis Compilation Phase"
echo "#####"
echo " "

set G_TIME_MARK [clock seconds]; # save start time;
P_timestamp;

# If Target Compile exists, then source it, otherwise do Project Compile
if {[P_exec_if_exists P_${G_DESIGN_NAME}_compile "MAIN"]} {
} elseif {[P_source_if_exists [file join ${G_SCRIPTS_PATH}
${G_DESIGN_NAME}_compile.tcl ] "MAIN"]} {
} else {P_proj_compile}

echo "#MAIN: Elapsed time of Compile Phase was" [P_elapsed_time $G_TIME_MARK];

```

3.3 Overview of target level directory structure

Any directory structure can be used for synthesis, Below is the default directory structure that the scripts assume to exist. Of course, these paths can be changed on a project basis in the proj_dc_set.tcl file.

```

unit_name
├── src – contains source code for unit
├── syn – top level synthesis directory, also contains the Makefile for the targets
│   ├── scripts – directory for target level Tcl script files
│   ├── logs – directory for runtime log files
│   ├── outputs – directory for netlists and other output files
│   ├── reports – directory for report files
│   ├── dbs – directory for all DB files, mapped, unmapped , etc
│   └── worklib - directory for *.mra , *.syn and *.sim files after analyzing

```

4.0 Coding Conventions for Scripts and Environment

We have tried to establish a set of conventions to help people extend and add to this environment. Inside our company we are treating this environment as open-source. This means others can contribute procedures to the environment as long as they follow the following guidelines.

- Procedural Tcl files (i.e. *_procs.tcl) contain no executable code outside procedure declarations and non-procedure files (i.e. proj_dc_setup.tcl file) should not define procedures, but only executable commands.
- File names should follow the outlined project / target naming conventions.
- Recommend no new files be added to the environment. New procedures can be created for additional functionality utilizing existing files.
- File contents preserve the difference between utility, project level and target level scripts.
- All global variables begin with “G_”.
- End every line with a semicolon before adding comments after a command.
- Use the UNIX process ID (PID) for external temporary files that need to be created. (This insures uniqueness although the user should remove any files before exiting. Refer to the procedure **P_get_license** for an example of using external files.

The following are some procedure specific coding guidelines:

- All procedures begin with “P_” prefix.
- Procedures always return a value (1-success, 0-failure, or string data).
- Create internal Tcl parser routines using the “regexp” command when needed. Do not invoke Perl or C-shell scripts from inside the DC Tcl environment, although we do recommend “exec-ing” standard UNIX commands when required.
- Recommend that utility and project procedures use the built-in dc_shell-t Tcl command “define_proc_attributes” to supply help on context.
NOTE: Not recommended for target procedures or report procedures.
- When possible assign default values to procedure arguments so a subset can be passed, i.e..

```
P_proj_outputs {dbs_path design_name outputs_path srcline {out_string ""}  
               {out_type ""}}
```


In this procedure “out_string” and “out_type” are optional arguments
- Follow the default flow conventions, as defined by the proj_main flow. Targets can include project procedures in order to make the target function additive to the project defaults. **NOTE:** Targets should NEVER re-define project level procedures on the fly.

5.0 Tcl Conversion Tips and Techniques

Below are a few tricks we learned as we converted our dcsh scripts to Tcl.

5.1 Learn how scoping in procedures and global variables work

All built in dc_shell variables must be declared global inside a procedure in order to make any changes to the variable. This includes variables like “current_design”. This caused some problems since the message “Information: defining new variable (CMD-041)” was being suppressed by our defaults.

Therefore, when we set a `dc_shell` variable inside a procedure, it only effected the locally defined procedure variable, not the `dc_shell` variable we wanted to change.

5.2 There is a difference between lists and collections

Almost every variable in Tcl is treated as a list of strings. This leads to the use of the “`lappend`” command to do arithmetic. There is one exception to this rule. Some `dc_shell` commands return collections, instead of strings (such as the `get_pins` command). These require a different set of commands than strings use (i.e `foreach_in_collection` vs. `foreach`). There is a good SOLVIT article describing collections and how to use them. Look for “DC-Tcl Questions from SNUG '99” in SOLVIT.

5.3 Use “define_proc_attributes” command for built-in help of procedures

Procedures can also have a “define_proc_attributes” associated with it, but it must be executed after the procedure has been defined. This allows you to program help messages into the procedure and also make the procedure permanent in memory, i.e the procedure cannot be overwritten or re-defined by another “proc” command using the same procedure name.

5.4 Take advantage of the built-in Tcl commands for the TOPS environment

The following commands may help the user see what is defined inside DC.

- Print the global variables and values matching string “G_”.

```
dc_shell-t> printvar G_*  
G_DEFAULT_WIRE_LOAD = " 90x90"  
G_LIBRARY_NAME      = " lsi_10k"
```
- Print all project procedures matching the string “

```
dc_shell-t> help P_proj*  
P_proj_change_names # Procedure  
P_proj_compile      # Procedure  
P_proj_compile_top   # Procedure
```
- Display verbose help screen, showing available parameters for procedure `P_source_if_exists`

```
dc_shell-t> help -v P_source_if_exists  
P_source_if_exists  # PROST: Procedure to see if a file exists  
    filename          (filename)  
    caller             (Status flag)
```
- Print the body of the procedure (view source code)

```
dc_shell-t> info body P_read_vhdl  
P_get_license "VHDL-Compiler";  
echo "#INFO: (P_read_vhdl): reading $fn";  
analyze -f vhdl $fn;  
remove_license "VHDL-Compiler";
```

6.0 Integration with PrimeTime

One of the primary goals of this environment was to share scripts with PrimeTime. This task was accomplished for the most part, with some changes introduced to accommodate PT. When creating a joint environment using PT and DC, one must be aware that not all commands are compatible or even exist in both tools. Some of the commands that don’t exist in PT but exist in DC are “`set_attribute`”, “`set_dont_touch`” and “`set_dont_use`”. On the other hand, the command “`set_case_analysis`” exists in PT, but not in DC. Others commands may have a slightly different result or operation in each tool.

For example the “*link*” command performs an implicit “*check_design*” that happens under the hood in PT, but this does not occur in DC.

In order to work around these differences, the scripts check the `dc_shell` variable `$synopsys_program_name`. If `$synopsys_program_name = pt_shell`, the commands that do not exist in PT are not executed. This allowed us to use the same `proj_library.tcl` script file for both tools, although no “*dont_use*” commands are executed if PT is running.

In order to setup PrimeTime initially, there are two files, *proj_pt_set.tcl* and *home.synopsys_pt.setup_tcl* that are used for running PT. These files operate in a similar fashion to their DC counterparts.

7.0 Supporting Scripts

Not all the scripts for the environment could be written in Tcl, however the non-Tcl scripts are never called from within DC or PT. The supporting scripts are described in the following sections.

7.1 Synbatch

The synbatch Perl script provides the interface from gmake to `dc_shell`. It has some limited capability to support remote jobs by setting the `HOST` variable when invoking make. Synbatch also provides some job control capability, in order to allow ctrl-C to exit a `dc_shell` job cleanly. Synbatch is the utility program actually called by the Makefile, which in turn executes the `dc_shell` command using the `-x` switch. The `dc-shell` command can also be called directly from the Makefile if desired by changing the `syn_make.rules` file.

Below is an example of how gmake invokes synbatch, which invokes `dc_shell` on a remote node.

```
%>gmake HOST=edison
synbatch ../src/design_name.v SUBFUB="" LIB=DEFAULT HOST=edison
```

Synbatch executes the `dc_shell` command:

```
dc_shell-t -x "set G_LIBRARY_FILE DEFAULT; set G_DESIGN_FILE_NAME
../src/design.v; set G_SUBFUBS ""; source proj_main.tcl;"
```

This is after synbatch remotely logs onto the machine “

In order to change the target library gmake is invoked with the `LIB` variable set as shown:

```
%>gmake LIB=altera
synbatch ../src/design_name.v SUBFUB="" LIB=altera HOST= "
```

The actual `dc_shell` command:

```
dc_shell-t -x "set G_LIBRARY_FILE altera; set G_DESIGN_FILE_NAME
../src/design.v; set G_SUBFUBS ""; source proj_main.tcl;"
```

7.2 mkdepend (Currently under development)

Mkdepend is a Perl script that is currently be developed and will be used to create a dependency list for the target's local Makefile. The mkdepend script will utilize a Makefile template called "*syn_Makefile*" to create a new dependency list. The resulting Makefile also includes " " which defines how to make the synthesis dependencies for the target. These templates reside in the \$PROJ_SYN directory. The mkdepend script will have the following features:

- Create or modify the target's Makefile.
- Handle different compilation strategies such as top down, bottom up, or combinations.
- Mkdepend will utilize output from the HS2 script (unsupported Synopsys Hierarchy Surfer utility written by Erik Olson). HS2 is actually a good starting point to build the Makefile dependency lists manually if needed.

8.0 Driving dc_shell-t with synbatch and gmake

For code compilation environments, gmake is the tool of choice. The TOPS Tcl environment also utilizes gmake. The job of gmake is to invoke dc_shell-t to compile targets based on timestamp dependencies. For a design named cnt_unit that has two sub blocks named cnt4.v and cnt2.v, gmake calls synbatch which makes the basic call to dc_shell as:

```
dc_shell-t -x "set G_DESIGN_FILE_NAME cnt_unit.v; set G_SUBFUBS [list
../src/cnt4.v ../src/cnt2.v]; set G_LIBRARY_NAME DEFAULT; source
proj_main.tcl; "
```

The Makefile that TOPS uses contains several key variables-

- **RECURSEDIRS**: enables gmake execution to "recurse" into lower-level sub-modules (top-level chip synthesis). If no sub-modules exist outside the current unit source directory, this variable is not required.
- **SUBFUB_<target_name>**: This is a key variable that contains the list of files that the target depends on. This should include path to source files. SUBFUB files may be Verilog, VHDL or db format. If no lower-level modules exist, the variable is not required.
- **\$(DB)<target_name.db>**: This is the target for synthesis. Should include the SUBFUB_<target_name> and path to the source file for the target.
- **all.local**: Required variable that should be assigned to the top-level DB target that is being synthesized.

Below is a simple Makefile for a target with two sub-blocks using a top down compile:

```
SUBFUB_cnt_unit := ../src/cnt4_vlog.v ../src/cnt2_vlog.v
```

```
$(DB)cnt_unit.db : $(SUBFUB_cnt_unit) ../src/cnt_unit.vhd
all.local: $(DB)cnt_unit.db
```

Next is the same Makefile using a bottom up compile strategy.

```
$(DB)cnt2_vlog.db : ../src/cnt2_vlog.v
$(DB)cnt4_vlog.db : ../src/cnt4_vlog.v
SUBFUB_cnt_unit := ../dbs/cnt4_vlog.db ../dbs/cnt2_vlog.db

$(DB)cnt_unit.db : $(SUBFUB_cnt_unit) ../src/cnt_unit.vhd
all.local: $(DB)cnt_unit.db
```

NOTE: Bottom-up requires 2 extra targets and constraints to support accurate compilation. Also the SUBFUB variable lists compiled DB files instead of RTL code.

9.0 Recommended Improvements of Tcl in Synopsys Tools

After designing this environment, there are a few areas in the dc_shell Tcl implementation where improvements would have made the script development simpler. Here are a few suggested areas for improvements.

9.1 Suppress_message vs suppress_error

As we converted our scripts we found that the variable suppress_errors was being re-written by dc_shell after elaboration of a design. The local Synopsys support person informed us of another variable “*suppress_message*” that may work better. It works great but you must be aware that if you append the same message ID to the list three times, you must remove it from the list three times also. This is due to the fact that it keeps a simple list, so you must be careful when using the Tcl string commands to append to the list.

9.2 Allow report commands (such as report_area) to return results to a Tcl variable

One of the largest areas, which required Tcl workarounds, was the inability of report commands to assign their output to Tcl variables. This was accomplished by piping the output to a local file and concatenating the file into a Tcl variable where we regular expressions were used to parse the data. Sometimes other UNIX commands were executed on the file before it was read back into the DC Tcl environment.

9.3 Fix the get_license command to disable the error message when you already have the license checked out.

This is another area where we spent a lot of time working around DC in order to check out a license. Currently a zero is returned from get_license if the license has already been obtained. Technically, the get_license command fails to check out another copy of the same license, but it does not prevent running DC commands that require the license. Consequently, one must check for existing licenses before requesting one, in order to make sure you do not get an error on a license request for a license you already have.

9.4 Remove differences in operation between the same commands in DC and PT

The “link” command operation in DC and PT is a primary example of these differences.

10.0 Conclusion

The TOPS environment has proven to be very flexible and expandable, being used by multiple projects across multiple sites. By using TOPS we have been able to implement a structured Tcl-based synthesis environment, which allowed us to meet all of our original design goals.

Many people were involved in creating TOPS and integrating it across different sites and projects. We want to thank the following people for their contributions:

- Steve Brown for his help in setting up our earlier synthesis environment on which TOPS is based.
- Doug Hergatt of CX Design for all his help with the scripts and countless hours debugging our problems.
- Gregg Lahti for all his Makefile, synbatch work and also the many utility scripts he helped to write and debug.
- Jesse Mahoney for all his scan and Tcl support with developing the scripts.

11.0 Appendix

This appendix contains example scripts of the TOPS environment. The following files make up the TOPS environment.

- home.synopsys_dc.setup_tcl
- home.synopsys_pt.setup_tcl
- proj_constraints.tcl
- proj_dc_setup.tcl
- proj_interactive.tcl
- proj_library.tcl
- proj_main.tcl
- proj_procs.tcl
- proj_pt_setup.tcl
- proj_reports.tcl
- report_procs.tcl
- syn_Makefile*
- syn_make.rules
- synbatch*
- tops_setup*
- util_procs.tcl

```
#!/bin/csh -fx
#####
# Script name   : tops_setup
# Date created  : 12/12/99
# Author       : Gregg D. Lahti
# Project      : TOPS Example Code for SNUG
#####
# $Id$
#####
# Description   : Setup script for TOPS (source this file)
#####

#####
# Setup project default environment variables
#####
set proj = TOPS
setenv PROJ_ROOT "your root level directory here"
setenv PROJ_LIB $PROJ_ROOT/lib
setenv PROJ_SYN $PROJ_ROOT/tools/syn
setenv PROJ_NAME ${proj}
umask 002

#####
# Setup script info/error warning items
#####

echo ""
echo "***** Starting TOPS Setup *****"
```



```

#####
# Set up Project Default Tool Versions
#####
set syn_version = 2000.05

#####
# Set machine type (here so we can use it farther down the script)
#####
if (-f /etc/machinetype) then
    setenv MACHTYPE `cat /etc/machinetype`
endif

#####
# Set Up basic Unix tools and environment
#####

# Added tools/syn to path to include the TOPS synthesis scripts
set path = ($PROJ_ROOT/tools/syn $path)

#####
# Set up Synopsys
#####
# source your setup files for Synopsys tools HERE !!!!
echo "Setting up SYNOPSYS TOOLS"

# check the USER Synopsys DC setup file
diff ${PROJ_SYN}/home.synopsys_dc.setup_tcl ~/.synopsys_dc.setup > & /dev/null
# NOTE: YOU MUST IMMEDIATELY CHECK THE STATUS VAR TO SEE THE DIFF RESULTS!!
#####
# Check the status and modify ~/.synopsys_dc.setup as needed
#####
switch($status)
    # the project template and the USER setup compare
    case 0 :
        echo " USER Synopsys DC setup file verified"
        breaksw
    # the project template and the USER setup miscompare
    case 1 :
        set date_var = `date +%m%d%y`
        echo " USER Synopsys DC setup file does not compare to the template"
        echo " Moved USER Synopsys DC setup to ~/.synopsys_dc.setup_$date_var"
        mv ~/.synopsys_dc.setup ~/.synopsys_dc.setup_${date_var}
        cp ${PROJ_SYN}/home.synopsys_dc.setup_tcl ~/.synopsys_dc.setup
        breaksw
    # the USER setup file does not exist, copy template
    case 2 :
        echo " USER synopsys DC setup copied from template"
        cp ${PROJ_SYN}/home.synopsys_dc.setup_tcl ~/.synopsys_dc.setup
        breaksw
    # something is wrong
    default:
        echo "ERROR: Synopsys DC setup not complete"
        breaksw
endsw

#####
# check the USER Synopsys PT setup file
#####
diff ${PROJ_SYN}/home.synopsys_pt.setup_tcl ~/.synopsys_pt.setup > & /dev/null
# NOTE: YOU MUST IMMEDIATELY CHECK THE STATUS VAR TO SEE THE DIFF RESULTS!!
# Check the status and modify ~/.synopsys_pt.setup as needed

```

```

switch($status)
# the project template and the USER setup compare
case 0 :
    echo " USER Synopsys PT setup file verified"
    breaksw
# the project template and the USER setup miscompare
case 1 :
    set date_var = `date "+%m%d%y"`
    echo " USER Synopsys PT setup file does not compare to the template"
    echo " Moved USER Synopsys PT setup to ~/.synopsys_pt.setup_${date_var}"
    mv ~/.synopsys_pt.setup ~/.synopsys_pt.setup_${date_var}
    cp ${PROJ_SYN}/home.synopsys_pt.setup_tcl ~/.synopsys_pt.setup
    breaksw
# the USER setup file does not exist, copy template
case 2 :
    echo " USER synopsys setup copied from template"
    cp ${PROJ_SYN}/home.synopsys_pt.setup_tcl ~/.synopsys_pt.setup
    breaksw
# something is wrong
default:
    echo "ERROR: Synopsys PT setup not complete"
    breaksw
endsw

#####
# Done with script
#####

echo "***** End TOPS Setup *****"

#####
# $Log$
#####

#Tcl-s
#####
# File Name      : home.synopsys_dc.setup
# Date Created   : 2/24/00
# Author        : Tim Wilson
# Project       : TOPS Example Code for SNUG
#####
# $Id$
#####
# Description: Synopsys SYNTHESIS USER setup script
#####

echo "#SETUP: Sourcing USER Synopsys setup file ~/.synopsys_dc.setup";
echo "";

# Set for both unix sh cmds and tcl files
set sh_source_uses_search_path true;

# Check environment setup
if {[getenv {PROJ_SYN}] != ""} {
    set proj_startup_dir [getenv {PROJ_SYN}];

    # Sourcing the project setup file
    source ${proj_startup_dir}/proj_dc_setup.tcl;

```

```

    unset proj_startup_dir;
} else {
    echo "#SETUP: No project startup file included";
    echo "#SETUP: Set environment variable $PROJ_SYN for project-wide setup";
}

# Enable USER-defined aliases, if exists
if {[which ~/.synopsys_aliases.tcl] != ""} {
    source ~/.synopsys_aliases.tcl;
}

#####
# $Log$
#####

#Tcl-s
#####
# File Name      : home.synopsys_pt.setup
# Date Created   : 2/24/00
# Author        : Tim Wilson
# Project       : TOPS Example Code for SNUG
#####
# $Id$
#####
# Description: Synopsys PRIMETIME USER setup script
#####

echo "#SETUP: Sourcing USER Synopsys PRIMETIME setup file ~/.synopsys_pt.setup";
echo "";

# Set for both unix sh cmds and tcl files
set sh_source_uses_search_path true;

# Check environment setup
if {[getenv {PROJ_SYN}] != ""} {
    set proj_startup_dir [getenv {PROJ_SYN}];

    # Sourcing the project setup file
    source ${proj_startup_dir}/proj_pt_setup.tcl;
    unset proj_startup_dir;
} else {
    echo "#SETUP: No project startup file included";
    echo "#SETUP: Set environment variable $PROJ_SYN for project-wide setup";
}

# Enable USER-defined aliases, if exists
if {[which ~/.synopsys_aliases.tcl] != ""} {
    source ~/.synopsys_aliases.tcl;
}

#####
# $Log$
#####

#####
# File Name      : proj_dc_setup.tcl
# Date Created   : 03/09/2000

```

```

# Author      : Tim L Wilson, Gregg D. Lahti & Rodney Pesavento
# Project     : TOPS Example Code for SNUG
#####
# $Id$
#####
# Description: This file is sourced from the USER's .synopsys_dc.setup
#####

#####
# Get environment variables from project environment setup script
#####
set G_PROJ_SYN [getenv PROJ_SYN]; # global synthesis dir
set G_PROJ_LIB [getenv PROJ_LIB]; # global library dir
set G_PROJ_DBS [file join ${G_PROJ_LIB} dbs]; # global library dbs dir, contains all
gate/io libs

echo "#SETUP: Sourcing Project Synopsys setup file ($G_PROJ_SYN/proj_dc_setup.tcl)";

#####
# Set Company & Designer (author)
#####
set company "TOPS";
set designer [getenv USER];

#####
# DC/BC Message output variables
#####
set verbose_messages false;

#####
# Create variables from unix environment variables &
# Define subdirectories of the unit's synthesis area
#####
# The following var G_SYN_PATH can be hardcoded to "."
set G_SYN_PATH "."; # PROJECT Edit, unit synthesis dir
set G_SRC_PATH [file join .. src];
set G_LOG_PATH [file join ${G_SYN_PATH} log];
set G_OUTPUTS_PATH [file join ${G_SYN_PATH} outputs];
set G_REPORTS_PATH [file join ${G_SYN_PATH} reports];
set G_SCRIPTS_PATH [file join ${G_SYN_PATH} scripts];
set G_WORKLIB_PATH [file join ${G_SYN_PATH} worklib];
set G_DBS_PATH [file join ${G_SYN_PATH} dbs];

#####
# Enable source command to use search path:
#####
set sh_source_uses_search_path true;
set sh_source_logging true;
set sh_returns_process_status true;

#####
# Setup Synopsys search path
# G_DBS_PATH is not included since synopsys will automatically search path
# for db's to link which is not always the desired case.
#####
# Set the default search path from $SYNOPSYS admin setup file
set searchline1 [file join ${synopsys_root} libraries syn]
set searchline2 [file join ${synopsys_root} dw sim_ver]
set search_path [list . $searchline1 $searchline2];
set search_path "$search_path $G_PROJ_DBS $G_PROJ_SYN $G_SRC_PATH $G_SCRIPTS_PATH";

```

```

define_design_lib work -path $G_WORKLIB_PATH;

#####
# Setup Synthetic Libraries
#####
set synthetic_library [list dw01.sldb dw02.sldb dw03.sldb dw04.sldb dw05.sldb dw07.sldb];

#####
# $TOPS is a global array use to pass a set of vars to a proc
# Note: each elem need not be set yet, this is project/machine specific
#####
global G_TOPS;
set G_TOPS(NULL) {/dev/null};          # add elem NULL to global array
set G_TOPS(TMP) {/tmp/TOPS};           # add elem TMP to global array

#####
# Project log file names and paths
#####
set command_log_file [file join ${G_LOG_PATH} command.log];
set exit_delete_command_log_file false;
set view_command_log_file [file join ${G_LOG_PATH} view_command.log];
set filename_log_file [file join ${G_LOG_PATH} filenames.log];
set exit_delete_filename_log_file false;

#####
# Setup the project Cache Root Directory
#####
# Assumes the synopsys_cache has been setup in the ${G_PROJ_LIB}
set myline1 [file join ${synopsys_root} libraries syn]
set myline2 [file join ${G_PROJ_LIB} synopsys_cache]
set cache_read "$myline1 $myline2";
set cache_write [file join ${G_PROJ_LIB} synopsys_cache];
set cache_read_info true;
set cache_write_info true;
set cache_file_chmod_octal 777;
set cache_dir_chmod_octal 777;

#####
# If desired, USER may Suppress some warning messages:
#####
# use suppress_message instead of set suppress_errors
# since elaborate changes the suppress_errors var
suppress_message {UI-31 UID-401 NMA-16 SYNOPT-8 SYNOPT-9 SYNOPT-18 \
                  CMD-030 CMD-041 EQN-10 VHDL-2097 SYNH-3 DDB-74};

#####
# Warn about dotfiles that may cause path problems, etc.
# Recommend NOT having local synopsys setup files, allows too much variability.
#####
if {[file exists [file join . .synopsys_dc.setup]]} {
    echo "#SETUP: WARNING .synopsys_dc.setup found in working directory";
    echo "#SETUP:          This could cause problems...";
}

#####
# Source TOPS global procedures file
#####
source [file join ${G_PROJ_SYN} util_procs.tcl];

#####
# DC variable settings needed project-wide

```

```
#####
# Put limits on special licenses
set synlib_disable_limited_licenses true;
set hdl_keep_licenses false;
set hdlin_enable_vpp true;
set hdlin_ff_always_async_set_reset true
set hdlin_translate_off_skip_text true;
set hdlout_internal_busses true;

#####
# Set bus naming variables
#####
set bus_inference_style {%s[%d]};
set bus_naming_style {%s[%d]};
set bus_range_separator_style {:};

#####
# Incorrect verilog written if change_names_dont_change_bus_members = false
#####
set change_names_dont_change_bus_members true;

#####
# Changes a sub designs name based on parameters passed to it.
#####
set template_naming_style "%s_%p";
set template_separator_style "_";
set template_parameter_style "%d";

#####
# Set VERILOGOUT Variables
#####
set verilogout_no_tri true;
set verilogout_single_bit false;
set verilogout_levelize false;
set verilogout_higher_designs_first true;
set verilogout_show_unconnected_pins true;

#####
# Set the history command to "keep" 100 instead of default 20
#####
history keep 100;

#####
# Define any project-specific aliases here. These are provided for
# interactive mode ONLY! Never recommended for scripts.
#####
alias h "history";
alias so "source";
alias cdn "current_design";

echo "#SETUP: Completed sourcing Project Synopsys setup";
echo "";

#####
# $Log$
#####

#####
# File Name      : proj_pt_setup.tcl
```

```

# Date Created : 03/09/2000
# Author      : Tim L Wilson
# Project     : TOPS Example Code for SNUG
#####
# $Id$
#####
# Description: This file is sourced from the USER's .synopsys_dc.setup
#####

#####
# Suppress some warnings:
#####
# use suppress_message instead of
# set suppress_errors (elaborate chgs this var)
suppress_message {CMD-041 CMD-030}

#####
# Get environment variables from project environment setup script
#####
set G_PROJ_SYN [getenv PROJ_SYN]; # global synthesis dir
set G_PROJ_LIB [getenv PROJ_LIB]; # global library dir
set G_PROJ_DBS [file join ${G_PROJ_LIB} dbs]; # global library dbs dir, contains all
gate/io libs

echo "#SETUP: Sourcing Project Synopsys PT setup file (${G_PROJ_SYN}/proj_pt_setup.tcl)";

#####
# Set Company & Designer (author)
#####
set company "TOPS";
set designer [getenv USER];

#####
# Create variables from unix environment variables &
# Define subdirectories of the unit's synthesis area
#####
# The following var G_SYN_PATH can be hardcoded to "."
set G_SYN_PATH "."; # PROJECT Edit, unit synthesis dir
set G_LOG_PATH [file join ${G_SYN_PATH} log];
set G_OUTPUTS_PATH [file join ${G_SYN_PATH} outputs];
set G_REPORTS_PATH [file join ${G_SYN_PATH} reports];
set G_SCRIPTS_PATH [file join ${G_SYN_PATH} scripts];
set G_DBS_PATH [file join ${G_SYN_PATH} dbs];

#####
# Enable source command to use search path:
#####
set sh_source_uses_search_path true;

#####
# Setup Synopsys search path
# G_DBS_PATH is not included since synopsys will automatically search path
# for db's to link which is not always the desired case.
#####
set search_path [list . [file join ${synopsys_root} libraries syn] \
[file join ${synopsys_root} dw sim_ver]];
set search_path ". $search_path $G_PROJ_DBS $G_PROJ_SYN $G_SCRIPTS_PATH";

#####
# $TOPS is a global array use to pass a set of vars to a proc
# Note: each elem need not be set yet

```

```
#####
global G_TOPS;
set G_TOPS(NULL) {/dev/null};          # add elem NULL to global array
set G_TOPS(TMP) {/tmp/TOPS};           # add elem TMP to global array

#####
# Project log file names and paths
#####
set command_log_file [file join ${G_LOG_PATH} command.log];
set exit_delete_command_log_file false;
set view_command_log_file [file join ${G_LOG_PATH} view_command.log];
set filename_log_file [file join ${G_LOG_PATH} filenames.log];
set exit_delete_filename_log_file false;

#####
# Setup Synthetic Libraries
#####
# Set because link_library will require this being set (proj_library.tcl)
set synthetic_library "";

#####
# Warn about dotfiles that may cause path problems, etc.
# Recommend NOT having local synopsys setup files, allows too much variability.
#####
if {[file exists [file join . .synopsys_pt.setup]]} {
    echo "#SETUP: WARNING .synopsys_pt.setup found in working directory";
    echo "#SETUP:          This could cause problems...";
}

#####
# Source TOPS global procedures file
#####
source [file join ${G_PROJ_SYN} util_procs.tcl];

#####
# Set the history command to "keep" 100 instead of default 20
#####
history keep 100;

#####
# Define any project-specific aliases here
#####
alias h "history";
alias so "source";

echo "#SETUP: Completed sourcing Project Synopsys setup";
echo "";

#####
# $Log$
#####

#####
# File Name      : proj_main.tcl
# Date Created   : 12/01/99
# Author         : Tim L. Wilson, Gregg D. Lahti & Rodney Pesavento
# Project        : TOPS Example Code for SNUG
#####
# $Id$
```



```

#####
# Description:
# This script defines the project synthesis flow. It is intended to
# provide a generic synthesis capability for most synthesis targets.
# For targets that require more specific synthesis flows, the
# target_main.tcl may be sourced from this script.
#####

#####
# Check for required variables
#####
set dc_shell_status [catch {list $G_DESIGN_FILE_NAME}];
if {$dc_shell_status != 0} {
    echo "#MAIN: Error-Variable G_DESIGN_FILE_NAME not set";
    quit; # Fatal Error, exit the tool
}
set dc_shell_status [catch {list $G_SUBFUBS}];
if {$dc_shell_status != 0} {
    echo "#MAIN: Info G_SUBFUBS not set, setting to null";
    set G_SUBFUBS "";
}

# Set G_DESIGN_NAME (remove path & extension)
set G_DESIGN_NAME [file tail [file rootname $G_DESIGN_FILE_NAME]];

# Save start time
set G_MAIN_START_TIME [clock seconds];
P_timestamp;

#####
# Source the Project, Local & Target specific procedures
#####
P_source_if_exists [file join ${G_SCRIPTS_PATH} ${G_DESIGN_NAME}_dc_setup.tcl] "MAIN";
P_source_if_exists [file join ${G_PROJ_SYN} proj_procs.tcl] "MAIN";
P_source_if_exists [file join ${G_SCRIPTS_PATH} ${G_DESIGN_NAME}_procs.tcl] "MAIN";
P_source_if_exists [file join ${G_PROJ_SYN} report_procs.tcl] "MAIN";
P_source_if_exists [file join ${G_PROJ_SYN} proj_reports.tcl] "MAIN";

#####
# Load project library setup script
#####
P_source_if_exists [file join ${G_PROJ_SYN} proj_library.tcl] "MAIN";

#####
# Source the Target Main script which overrides TOPS Main. This allows
# project to define special flows. The flow control should exit in that
# path.
#####
P_source_if_exists [file join ${G_SCRIPTS_PATH} ${G_DESIGN_NAME}_main.tcl] "MAIN";

echo ""
echo "#####"
echo "#           Design Analyze & Elaborate Phase"
echo "#####"
echo ""

# Set timestamp & Read the SUBFUB's if they exist
set G_TIME_MARK [clock seconds];
P_timestamp;

#####

```

```

# Analyze ONLY the SUBFUB list
#####
P_read_subfubs $G_SUBFUBS;

#####
# Analyze & elaborate top level design
#####
P_read_top $G_DESIGN_FILE_NAME;

#####
# Get the Id CVS string from the source files
#####
set srcline [P_get_srcline $G_DESIGN_FILE_NAME $G_SUBFUBS];

# Link design
link;

echo "#MAIN: Writing out unmapped db for $G_DESIGN_NAME";
write -f db -hier -o [file join ${G_DBS_PATH} ${G_DESIGN_NAME}_unmapped.db]
$G_DESIGN_NAME;

echo "#MAIN: Elapsed time of Analyze & Elaborate Phase was" [P_elapsed_time
$G_TIME_MARK];

echo ""
echo "#####
echo "#          Apply Synthesis Design Constraints Phase"
echo "#####
echo ""

set G_TIME_MARK [clock seconds];
P_timestamp;

#####
# Set the operating conditions
#####
P_proj_operating_conditions;

#####
# Set wire_load. Design-specific script overrides project default
#####
# Source Target-level, else set the default project WLM
if {[P_exec_if_exists P_${G_DESIGN_NAME}_wire_load "MAIN"]} {
} elseif {[P_source_if_exists [file join ${G_SCRIPTS_PATH}
${G_DESIGN_NAME}_wire_load.tcl] "MAIN"]} {
} else {P_proj_wire_load}

#####
# Project & Target-level constraint loop.
# Include clocks/resets & constraints.
# Turn off auto-link to speed up constraint reading
#####
set auto_link_disable true;

# Set clocks for design
if {[P_exec_if_exists P_${G_DESIGN_NAME}_clocks "MAIN"]} {
} elseif {[P_source_if_exists ${G_DESIGN_NAME}_clocks.tcl "MAIN"]} {
} else {P_proj_clocks}

# Set Constraints for design. This list could be extended if needed
set includes {constraints};

```

```

foreach file $includes {
    # Target-level files REPLACE project-level files
    if {[P_source_if_exists [file join ${G_SCRIPTS_PATH} ${G_DESIGN_NAME}_${file}.tcl]
"MAIN"]} {
        } else {P_source_if_exists proj_${file}.tcl "MAIN"}
    }
}

#####
# Target-level only constraint loop for dont_use & exceptions.
# Design-specific scripts are added to project defaults
#####
# Create a variable with for the library without the corner
set lib [lindex [split $G_LIBRARY_NAME "_"] 0];
# This list could be extended if needed
set includes "${lib}_dont_use {exceptions}";
foreach file $includes {
    # Source target constraints if they exist
    P_source_if_exists [file join ${G_SCRIPTS_PATH} ${G_DESIGN_NAME}_${file}.tcl] "MAIN"
}

set auto_link_disable false;

echo "#MAIN: Elapsed time of Constraints Phase was" [P_elapsed_time $G_TIME_MARK];

#####
# Check Timing Constraints
#####
check_timing;

echo ""
echo "#####"
echo "#           Synthesis Compilation Phase"
echo "#####"
echo ""

set G_TIME_MARK [clock seconds];
P_timestamp;

# If Target Compile exists, then source it, otherwise do Project Compile
if {[P_exec_if_exists P_${G_DESIGN_NAME}_compile "MAIN"]} {
} elseif {[P_source_if_exists [file join ${G_SCRIPTS_PATH} ${G_DESIGN_NAME}_compile.tcl]
"MAIN"]} {
} else {P_proj_compile}

echo "#MAIN: Elapsed time of Compile Phase was" [P_elapsed_time $G_TIME_MARK];

set G_TIME_MARK [clock seconds];
P_timestamp;

#####
# Change the names before writing out the ouptuts
#####
P_proj_change_names $G_LOG_PATH $G_DESIGN_NAME;

echo "#MAIN: Elapsed time of Change Names was" [P_elapsed_time $G_TIME_MARK];

echo ""
echo "#####"
echo "#           Write Design Outputs Phase (db, netlist, SDF, SDC)"
echo "#####"
echo ""

```

```

set G_TIME_MARK [clock seconds];
P_timestamp;

# If Target Outputs exists, then exec it, otherwise do Project Outputs
if {[P_exec_if_exists P_${G_DESIGN_NAME}_outputs "MAIN"]} {
} elseif {[P_source_if_exists [file join ${G_SCRIPTS_PATH} ${G_DESIGN_NAME}_outputs.tcl]
"MAIN"]} {
} else {P_proj_outputs $G_DBS_PATH $G_DESIGN_NAME $G_OUTPUTS_PATH $srcline}

echo "#MAIN: Elapsed time of Output Generation Phase was" [P_elapsed_time $G_TIME_MARK];

echo ""
echo "#####
echo "#           Write Design Reports Phase"
echo "#####
echo ""

set G_TIME_MARK [clock seconds];
P_timestamp;

# If Target Reports exists, then exec it, otherwise do Project Reports
if {[P_exec_if_exists P_${G_DESIGN_NAME}_reports "MAIN"]} {
} elseif {[P_source_if_exists [file join ${G_SCRIPTS_PATH} ${G_DESIGN_NAME}_reports.tcl]
"MAIN"]} {
} else {P_proj_reports $G_REPORTS_PATH $G_DESIGN_NAME}

echo "#MAIN: Elapsed time of Report Generation Phase was" [P_elapsed_time $G_TIME_MARK];

echo "#MAIN: Elapsed time of proj_main script was " [P_elapsed_time $G_MAIN_START_TIME];
P_timestamp;

#####
# Filter the synthesis compile log for Errors, Warnings, & Elapsed time
#####
P_log_filter $G_LOG_PATH $G_DESIGN_NAME;

exit 0;

#####
# $Log$
#####

#####
# File Name      : proj_library.tcl
# Date Created   : 3/09/2000
# Author        : Tim L. Wilson, Gregg D. Lahti & Rodney Pesavento
# Project       : TOPS Example Code for SNUG
#####
# $Id$
#####
# Description: Set up library path/variable for Intel 859hd library
#####

# Trap to check if G_LIBRARY_NAME is not set (USER invocation)
if {[info exists G_LIBRARY_NAME] == 0} {
    # Set to null & allow default loop below to set
    set G_LIBRARY_NAME "DEFAULT";
}

```

```

# If not set by synbatch or syn_make.rules, set to project default
if {$G_LIBRARY_NAME == "DEFAULT"} {
    # Set the default, this is the only place lib vars need to be set
    set G_LIBRARY_NAME lsi_10k; # PROJECT Specific Edit
}

# Source the files required for the synthesis target library
switch -exact -- $G_LIBRARY_NAME {
    altera {
        echo "#LIBRARY: ERROR: altera not setup yet!\n";
    }
}

lsi_10k {
    echo "#LIBRARY: Setting up $G_LIBRARY_NAME library files & variables";
    #####
    # Load project library setup script
    #####
    # Set multiplier, library is in picoseconds
    set G_NS 1;

    # These variables CAN modify default library values
    #set G_MAX_TRANSITION [expr 3.0 * $G_NS];
    #set G_MAX_CAPACITANCE 5.0;
    #set G_MAX_DELAY 5.0;

    # Default Scan Flop used for drive
    set G_DEFAULT_DRIVING_CELL {FD1S};
    set G_DEFAULT_DRIVING_CELL_PIN {Q};

    # Default nand used for load
    set G_DEFAULT_LOAD_CELL {ND2};
    set G_DEFAULT_LOAD_CELL_PIN {A};

    # This variable is used to normalize to "NAND equivalent gates"
    set G_DEFAULT_NAND_CELL {ND2};

    # Set Operating conditions & min/max libraries for synthesis
    set G_MAX_LIBRARY {$G_LIBRARY_NAME};
    set G_MIN_LIBRARY {$G_LIBRARY_NAME};
    set G_MAX_OP_CONDITION {WCCOM};
    set G_MIN_OP_CONDITION {BCCOM};

    # Setup wireload variables
    set G_DEFAULT_WIRE_LOAD {90x90};
    set G_CWLM_LIBRARY [list ""];
    set auto_wire_load_selection false;

    #####
    # Set library name and search path
    #####
    ## NOTE: Assumes dbs path already set in search_path from proj_dc_setup.tcl

    #####
    # Define Synthesis Libraries
    #####
    # Set the target library, include RAM and other special cell libs
    set target_library [list {$G_MAX_LIBRARY}.db];
    set symbol_library [list lsi_10k.sdb];

    # Synthetic_library is assigned in proj_dc_setup.scr, include * for designs in memory

```

```

set link_library "*" $target_library $synthetic_library";

# Read all the libraries into memory
foreach lib [concat $target_library ${G_MIN_LIBRARY}.db] {
    read_db $lib;
}; # end of foreach

#####
# Create default don't use list, if not PT shell session.
# Recommend using the G_PROJ_DONT_USE variable to set the attribute.
# This allows the USER to remove the attribute:
# dc_shell-t> remove_attribute $G_PROJ_DONT_USE dont_use
#####
if {[set synopsys_program_name] != "pt_shell"} {
    # Not using JK flops
    set G_PROJ_DONT_USE "${G_LIBRARY_NAME}/FJK*";

    # Not using latch components
    set G_PROJ_DONT_USE [concat $G_PROJ_DONT_USE "${G_LIBRARY_NAME}/LD*"];
    set G_PROJ_DONT_USE [concat $G_PROJ_DONT_USE "${G_LIBRARY_NAME}/LS*"];

    # Set dont_use attribute on cells in list
    set_dont_use $G_PROJ_DONT_USE;
}; # end if
}

default {
    echo "#LIBRARY: Error, G_LIBRARY_NAME variable set incorrectly";
    echo "#LIBRARY: G_LIBRARY_NAME = $G_LIBRARY_NAME\n";
}

}; # end switch

# Set dont_use attribute on the slow design_ware components (speeds compilation)
if {[set synopsys_program_name] != "pt_shell"} {
    read_db standard.sldb;
    set_dont_use {standard.sldb/DW02_mult/csa \
        standard.sldb/DW01_add/rpl \
        standard.sldb/DW01_add/cia \
        standard.sldb/DW01_sub/rpl \
        standard.sldb/DW01_sub/cia \
        standard.sldb/DW01_addsub/rpl \
        standard.sldb/DW01_addsub/cia \
        standard.sldb/DW01_inc/rpl \
        standard.sldb/DW01_inc/cia \
        standard.sldb/DW01_dec/rpl \
        standard.sldb/DW01_dec/cia \
        standard.sldb/DW01_incdec/rpl \
        standard.sldb/DW01_incdec/cia \
    };
}; # end if

#####
# $Log$
#####

#####
# File Name      : proj_procs.tcl
# Date Created   : 3/06/2000

```

```

# Author      : Tim L Wilson, Gregg D. Lahti & Rodney Pesavento
# Project     : TOPS Example Code for SNUG
#####
# $Id$
#####
# Description: Project-Specific utility procedures for synthesis
#####

#####
# Usage: P_proj_annotate_clocks {min_uncertainty}
#
# This procedure removes insertion delay & prepares clocks for postlayout analysis
#####
proc P_proj_annotate_clocks {min_uncertainty} {
    # Adjust the clock uncertainty to min
    echo "Annotating all defined clocks for postlayout analysis";
    set_clock_uncertainty 10 [all_clocks];
    set_clock_latency 0.0 -source [all_clocks];

    # Do not want this clock to be ideal, set to propagated
    set_propagated_clock [all_clocks];
}; # end P_annotate_clocks

# This is the help attribute definition
define_proc_attributes P_proj_annotate_clocks \
    -info "PROJ_PROCS: Removes insertion delay & prepares clocks for post-layout analysis"
\
    -define_args {
        {min_uncertainty "Minimum Clock uncertainty" min_uncertainty float required}
    }

#####
# Usage: P_proj_clocks
#
# This procedure is used to setup the projects clocks.
#####
proc P_proj_clocks {} {
    global DEFAULT_CLOCK;

    # Define all clocks with the following invocation format:
    #
    # P_proj_constrain_clock clk_name period uncertainty insertion
    #
    # REQUIRED arguments:
    #   clk_name:      clock name
    #   period:        clock period (time units of lib ps)
    #   uncertainty:   clock uncertainty (time units of lib ps)
    #   insertion:     clock insertion delay (time units of lib ps)
    #
    # OPTIONAL arguments (the following should only be necessary when manipulating
    #                     duty cycle and complex waveform creation):
    #   hierarchy:     hierarchical instance path to clock
    #   risel:         first rise time (time units of lib ps)
    #   fall1:         first fall time (time units of lib ps)
    #   rise2:         next rise (time units of lib ps)
    #   fall2:         next fall (time units of lib ps)
    #
    # Example:
    #   P_proj_constrain_clock "clk_pxb" 5.024 .175 0.250 2.762;

    # Project clocks

```

```

P_proj_constrain_clock "cpuclk" 40000 150 50;
P_proj_constrain_clock "apb_pclk" 38000 150 50;
P_proj_constrain_clock "clk32k" 1000000 150 50;
P_proj_constrain_clock "iclk" 12345.6 150 50;
P_proj_constrain_clock "MCLK" 12345.6 150 50;
P_proj_constrain_clock "mclk" 12345.6 150 50;
P_proj_constrain_clock "tsclk" 12345.6 150 50;
P_proj_constrain_clock "clk_18" 55555 150 50;
P_proj_constrain_clock "clk" 3000 100 50;

# Setup default in case there is only one clock
set G_DEFAULT_CLOCK apb_pclk;

#####
# Create a clock for multicycle path timing exceptions
#####
#create_clock -period 38000 -name apb_pclk_mc;

# Display current clock info
report_clock -nosplit;
report_clock -skew -nosplit;

#####
# Constrain reset ports for design
#####
P_proj_constrain_reset "biERSTqf";
P_proj_constrain_reset "totalreset";
P_proj_constrain_reset "syncreset";
P_proj_constrain_reset "sysrst";
P_proj_constrain_reset "apb_presetn";
}; # end P_proj_clocks

define_proc_attributes P_proj_clocks \
    -info "PROJ_PROCS: Procedure to setup all the project clocks"

#####
# Usage: P_proj_constrain_reset {reset_name}
#
# This procedure is used to set the constraints on the "reset" ports
#####
proc P_proj_constrain_reset {reset_name} {
    global G_TOPS;
    global synopsys_program_name;

    # If port does not exist, send warning results to bit bucket
    redirect $G_TOPS(NULL) \
        {set test4rst [get_ports -filter {@port_direction == in} $reset_name]};

    # If port exists, go ahead and constrain
    if {$test4rst != {}} {
        echo "#RESET-Info: Setting reset attributes on port - $reset_name";

        remove_driving_cell $reset_name;
        set_ideal_net $reset_name;
        set_drive 0 $reset_name;
        set_false_path -from $reset_name;

        # This attribute gets used in the constraints script
        if {[set synopsys_program_name] != "pt_shell"} {
            set_attribute $reset_name is_reset true -type boolean;
        }
    }
}

```



```

    }; # end if
    unset test4rst;
}; # end P_proj_constrain_reset

define_proc_attributes P_proj_constrain_reset \
    -info "PROJ_PROCS: Procedure to set the constraint on RESET port" \
    -define_args {
        {reset_name "Name of the port on which constraints are to be set" reset_name string
        required}
    }

#####
# Usage: P_proj_constrain_clock {clk_name period uncertainty insertion \
#                                     hier {r1 ""} {f1 ""} {r2 ""} {f2 ""}}
# Example: P_proj_constrain_clock {pciclk 1000.0 150.0 0 "" 0 400 0 0}
#
# This procedure is used to define & create clock "clk_name" with attributes.
# NOTE: User must specify "hier"/ if setting waveform vars
#####
proc P_proj_constrain_clock {clk_name period uncertainty insertion {hier ""} {r1 ""} {f1
""} {r2 ""} {f2 ""}} {
    global G_CLOCK_ARRAY;
    global G_TOPS;
    global G_CWLM_LIBRARY;
    global synopsys_program_name;

    # Test for the clock to exist. use hierarchy if supplied
    if {$hier != ""} {
        set clk_name $hier$clk_name;
        # test4clk is a collection, if the collection exists the clock exists
        redirect $G_TOPS(NULL) \
            {set test4clk [get_ports $clk_name]};
    } else {
        # test4clk is a collection, if the collection exists the clock exists
        redirect $G_TOPS(NULL) \
            {set test4clk [get_ports -filter {@port_direction == in} $clk_name]};
    }

    # Constrain clocks if they exist
    if {$test4clk != {}} {
        echo "#CLOCKS-Info: Setting clock attributes on port - $clk_name";

        # Create a global array to contain all clock info
        set G_CLOCK_ARRAY($clk_name,name) $clk_name;
        set G_CLOCK_ARRAY($clk_name,period) $period;
        set G_CLOCK_ARRAY($clk_name,uncertainty) $uncertainty;
        set G_CLOCK_ARRAY($clk_name,insertion) $insertion;

        # Create clock
        if {$r1 != "" || $f1 != "" || $r2 != "" || $f2 != ""} {
            create_clock -period $period -waveform [list $r1 $f1 $r2 $f2] $clk_name;
        } else {
            create_clock -period $period $clk_name;
        }

        # Only if not hierarchical
        if {$hier == ""} {
            remove_driving_cell $clk_name;

            # Set input port drive to infinite drive capability (no delay)
            set_drive 0 $clk_name;
        }
    }
}

```

```

    }

    # Sets the dont_touch attribute on pin or clock
    set_dont_touch_network $clk_name;

    # Set the uncertainty, models clock skew
    set_clock_uncertainty $uncertainty $clk_name;

    # Sets insertion delay of clock tree
    set_clock_latency $insertion -source $clk_name;

    # This attribute gets used in the constraints script
    if {[set synopsys_program_name] != "pt_shell"} {
        set_attribute $clk_name is_clock true -type boolean;
    }

    # Only use cwlm if it exists
    if {$G_CWLM_LIBRARY != "{}" && $hier == ""} {
        # Get cwlm rootname & set_wire_load
        set cwlm_lib_name [file rootname $G_CWLM_LIBRARY];
        set_wire_load_model -name CLKNET -lib $cwlm_lib_name $clk_name;
    } else {
        set_resistance 0.0 $clk_name;
        set_load 0.0 $clk_name;
    }; # end if
}; # end if
unset test4clk;
return;
}; # end P_proj_constrain_clock

# This is the help attribute definition
define_proc_attributes P_proj_constrain_clock \
    -info "PROJ_PROCS: Define & create clock for clk_name with attributes" \
    -define_args {
        {clock_name "Clock Name" clock_name string required}
        {period "Clock Period in library time units" period float required}
        {uncertainty "Clock Uncertainty in library time units" uncertainty float required}
        {insertion "Clock Insertion Delay in library time units" insertion float required}
        {hier "Hierarchy to the clock source" hier string}
        {r1 "Waveform Rise1" r1 float}
        {f1 "Waveform Fall1" f1 float}
        {r2 "Waveform Rise2" r2 float}
        {f2 "Waveform Fall2" f2 float}
    }
}

#####
# Usage: P_proj_inst_cnt
#
# This procedure is used to count the number of instances
#####
proc P_proj_inst_cnt {} {
    set cell_count [sizeof_collection [get_cells -hier * ]]

    set leaf_cells [get_cells -hier * -filter "@is_hierarchical == false"]

    set hier_cell_count [sizeof_collection [get_cells -hier * -filter "@is_hierarchical == true"]]
    set leaf_cell_count [sizeof_collection [get_cells -hier * -filter "@is_hierarchical == false"]]

    set seq_leaf_count [sizeof_collection [filter $leaf_cells "@is_sequential == true"] ]

```

```

    set combo_leaf_count [sizeof_collection [filter $leaf_cells "@is_combinational ==
true"]]

    echo "*****"
    echo "*"
    echo "*" Design          : " [get_object_name [current_design]]
    echo "*" Number of cells : " $cell_count
    echo "*" Hierarchical    : " $hier_cell_count
    echo "*" Leaf            : " $leaf_cell_count
    echo "*" Sequential      : " $seq_leaf_count
    echo "*" Combinational   : " $combo_leaf_count
    echo "*"
    echo "*****"
    return 0;
}; # end P_proj_cell_cnt

define_proc_attributes P_proj_inst_cnt \
    -info "PROJ_PROCS : Procedure to count the number of instances"

#####
# Usage: P_proj_change_names
#
# This procedure defines the projects change names rules (removes hierarchy separators
# and special characters from netlists) and is included prior to the outputs
# being written out.
#
# Magic variables - for retaining busses
# Note: these are only a few of the variables used to write out
# HDL and EDIF. The others have their default values in this example.
#####
proc P_proj_change_names {log_path_lcl design_name_lcl} {
    # Set global for scoping
    global bus_inference_style;
    global bus_naming_style;
    global change_names_dont_change_bus_members;
    global hdlout_internal_busses

    echo "#PROJ: Changing project names";

    #####
    # Name rule to convert "/" (hierarchical separator) to "_"
    # Note: Since bus_naming_style is changed you will see NMA-16 warnings.
    # These are OK; the busses will still be preserved.
    # Also, a separate rule for the slash makes the output more readable.
    # the simple_??? rule fixes / but not as cleanly as a separate rule.
    # slashes occur in designs when they are ungrouped.
    #####

    define_name_rules slash -restricted {/} -replace {_};

    #####
    # Name rule to make legal VHDL, Verilog names. Note: This example
    # only covers one-dimensional arrays. For two-dimensional arrays the
    # mapping should have four pairs instead of two.
    #####
    # There are different name rules for ports and nets vs cells.
    # removing the bus naming style causes mismatches between ports and nets.
    # NOTE: it is dangerous to change the port names, mismatches between
    # connected instances can occur.
    #####
    # Top level Ports and Nets connected to Ports must already follow the naming

```

```

# convention, however Synopsys created nets and ports (think DW) may not.
#####

define_name_rules simple_port \
    -allowed {A-Za-z0-9[_]} -first {0-9_} -last {_} \
    -equal_ports_nets -type port;

define_name_rules simple_net \
    -allowed {A-Za-z0-9[_]*} -first {0-9_} -last {_} \
    -type net;

define_name_rules simple_cell \
    -allowed {A-Za-z0-9_} -first {0-9_} -last {_} \
    -map {{ {"[","_"} {""],"",""} {"[","_"} {""],"",""} }} -remove -type cell;

#####
# Exec desired names rules
#####
redirect [file join ${log_path_lcl} ${design_name_lcl}_change_names_slash.log] \
    {change_names -hier -rules slash -verb};

redirect [file join ${log_path_lcl} ${design_name_lcl}_change_names_simple.log] \
    {report_name_rules simple_port};
redirect -append [file join ${log_path_lcl} ${design_name_lcl}_change_names_simple.log] \
    {change_names -hier -rules simple_port -verb};

redirect -append [file join ${log_path_lcl} ${design_name_lcl}_change_names_simple.log] \
    {report_name_rules simple_net};
redirect -append [file join ${log_path_lcl} ${design_name_lcl}_change_names_simple.log] \
    {change_names -hier -rules simple_net -verb};

redirect -append [file join ${log_path_lcl} ${design_name_lcl}_change_names_simple.log] \
    {report_name_rules simple_cell};
redirect -append [file join ${log_path_lcl} ${design_name_lcl}_change_names_simple.log] \
    {change_names -hier -rules simple_cell -verb};
}; # end P_proj_change_names

define_proc_attributes P_proj_change_names \
    -info "PROJ_PROCS: Procedure to remove hierarchy separators and special characters \
netlists and SDF's, included prior to netlist being written out." \
    -define_args {
        {log_path_lcl "Path to the log files" log_path_lcl string required}
        {design_name_lcl "design name" design_name_lcl string required}
    }

#####
# Usage: P_proj_compile
#
# This procedure is used to create the project compilation strategy
# May include procedures for pre & post scan insertion
#####
proc P_proj_compile {} {
    set_flatten false;
    set_structure true;

    #####

```

```

# May want to try new boolean optimization for 1997.01 & up
#####
# set compile_new_boolean_structure true
# set_structure true -boolean true -boolean_effort medium

# Fix multiply instantiated designs
uniquify;

# This is required for test-ready compile (compile -scan)
P_get_license "Test-Compiler";

echo "#PROJ: Performing project compile";
compile -map medium -scan;

# Ungroup the designware components
P_proj_ungroup_dw;

# Perform incremental compile to clean up after the DW ungrouping
compile -map_effort low -incremental;
}; # end P_proj_compile

define_proc_attributes P_proj_compile \
    -info "PROJ_PROCS: compiles the project"

#####
# Usage: P_proj_compile_top
#
# This procedure is used to compile the top-level netlist or the top
# of any target (unit, super-unit) that contains modules that do not
# need to be compiled.
#####
proc P_proj_compile_top {} {
    set_flatten false;
    set_structure true;

    echo "#PROJ: Performing project top-level compile";
    compile -top;
}; # end P_proj_compile_top

define_proc_attributes P_proj_compile_top \
    -info "PROJ_PROCS: compiles the project top-level"

#####
# Usage: P_proj_insert_buffers lib_cell_name inst_string fix_hold_mode
#
# This procedure is used to fix HOLD violations by inserting strategic
# buffers in the scan, data or both paths of the design. It is intended
# to be invoked pre-layout to reduce the number of violations.
#####
proc P_proj_insert_buffers {lib_cell_name inst_string fix_hold_mode} {

    # Generate list of violators & filter on VIOLATED
    report_constraint -min_delay -all > tmp;
    set violator_list [exec sed -n -e "/VIOLATED/p" tmp];
    sh rm tmp;

    # Check for no violations
    if {$violator_list == ""} {
        echo "#INFO: No MIN violations in current design";
        return;
    }; # end if

```

```

set si_violator_list "";
set d_violator_list "";

# Now filter on "/SI" & create list
foreach violator $violator_list {
    if [string match */SI $violator] {
#        echo $violator;
        set si_violator_list [concat $si_violator_list $violator];
    }; # end if
    if [string match */D $violator] {
#        echo $violator;
        set d_violator_list [concat $d_violator_list $violator];
    }; # end if
}; # end foreach

# Build the list to fix
if {$fix_hold_mode == "scan"} {
    set all_violator_list $si_violator_list;
} elseif {$fix_hold_mode == "data"} {
    set all_violator_list $d_violator_list;
} elseif {$fix_hold_mode == "both"} {
    set all_violator_list [concat $d_violator_list $si_violator_list];
} else {
    echo "ERROR: Incorrect Fix Hold Mode. Please specify scan, data or both";
}; # end if

# Reset cell counter (used to create a unique instance name
set cell_cnt 0;

echo "#INFO: Fixing Hold violations on the violator paths.";

# Fix all the violations for the violator list
foreach reg_cell $all_violator_list {

    # Get the instance name of the violator
    regexp {(.*).*.} $reg_cell var1 var2
    set inst_name [file tail $var2];

    # Get the hierarchy path to the violating register
    regexp {(.*).*.} $var2 var1 hier_path

    # Increment the cell counter
    set cell_cnt [expr $cell_cnt + 1];

    # Find everything connected to the pin of this register
    set orig_net [all_connected [get_pins $reg_cell]];
    # Create new instance name (HOLD delay cell)
    set new_cell ${inst_name}${inst_string}${cell_cnt};
    # Create new net name (net inserted between new HOLD cell & register pin)
    set new_net ${new_cell}_net;

    # Disconnect the orig net & create the new cell & net
    disconnect_net $orig_net $reg_cell;
    create_cell -instance $hier_path $new_cell $lib_cell_name;
    create_net -instance $hier_path $new_net;

    # Add the hierarchy path to the new net & cell vars
    set new_net [file join $hier_path $new_net];
    set new_cell_in [file join $hier_path ${new_cell}/A];
    set new_cell_out [file join $hier_path ${new_cell}/O];

```

```

        # Connect the nets
        connect_net $new_net $reg_cell;
        connect_net $new_net $new_cell_out;
        connect_net $orig_net $new_cell_in;
    }; # end foreach

    echo "#INFO: Added $cell_cnt HOLD cells to the design.";
}; # end proc

define_proc_attributes P_proj_insert_buffers \
    -info "PROJ: Procedure to fix HOLD violations inserting strategic buffers" \
    -define_args {
        {lib_cell_name "Library Cell Name to insert" lib_cell_name string required}
        {inst_string "Instance String to insert in the new cell" inst_string string required}
        {fix_hold_mode "Fix Hold Mode: scan, data, or both" fix_hold_mode string required}
    }
}

#####
# Usage: P_proj_operating_conditions
#
# This procedure Sets default operating conditions using min/max conditions
# from library.
# NOTE: Synopsys does not maintain this information when saving a design.
#       This proc needs to be run again when loading a design into memory.
#####
proc P_proj_operating_conditions {} {
    global G_MAX_OP_CONDITION;
    global G_MIN_OP_CONDITION;
    global G_MAX_LIBRARY;
    global G_MIN_LIBRARY;

    # Set min_library only if G_MAX_LIBRARY not same as G_MIN_LIBRARY
    if {[string match ${G_MAX_LIBRARY} ${G_MIN_LIBRARY}]} {
        # Assume that min library already in memory (proj_library.tcl)
        set_min_library ${G_MAX_LIBRARY}.db -min_version ${G_MIN_LIBRARY}.db;
    }; # end if

    # Set operating conditions
    set_operating_conditions \
        -max $G_MAX_OP_CONDITION \
        -max_library $G_MAX_LIBRARY \
        -min $G_MIN_OP_CONDITION \
        -min_library $G_MIN_LIBRARY;

}; # end of P_proj_operating_conditions

define_proc_attributes P_proj_operating_conditions \
    -info "PROJ_PROCS: Sets the default operating conditions using the min/max conditions
from the library"

#####
# Usage: P_proj_precompile_scan
#
# This procedure is used to set the scan config & define the scan ports
# for the design.
#####
proc P_proj_precompile_scan {} {
    echo "#PROJ: Setting scan configuration";

```

```

# Provided the full spectrum of scan options, choose wisely
set_scan_configuration \
  -add_lockup false \
  -area_critical false \
  -bidi_mode input \
  -chain_count default \
  -clock_gating entire_design \
  -clock_mixing no_mix \
  -dedicated_scan_ports true \
  -disable false \
  -existing_scan false \
  -hierarchical_isolation false \
  -methodology full_scan \
  -multibit_segments true \
  -rebalance false \
  -replace true \
  -route true \
  -route_signals "serial_scan_enables" \
  -style multiplexed_flip_flop \
  -internal_clocks true \
  -create_test_clocks_by_system_clock_domain false \
  -physical false

# Define naming style for the design after inserting scan
set insert_test_design_naming_style "%s_test_%d";

# Set the scan test mode pins
#set_test_hold 1 [get_port scantestmode];

# Method 1: Allow Synopsys to determine scan ports
# Here are some variables to rename the ports used for scan chains
set test_scan_enable_port_naming_style "test_se%s";
set test_scan_in_port_naming_style "test_si%s%s";
set test_scan_out_port_naming_style "test_so%s%s";

# Method 2: Specify the scan ports for Synopsys
#set_scan_path chain1
#set_scan_signal test_scan_in -port my_scan_in -chain chain1
#set_scan_signal test_scan_out -port my_scan_out -chain chain1

}; # end P_proj_precompile_scan

define_proc_attributes P_proj_precompile_scan \
  -info "PROJ_PROCS: Procedure to set scan configuration"

#####
# Usage: P_proj_postcompile_scan {report_path design_name}
#
# This procedure is used to insert scan
#####
proc P_proj_postcompile_scan {report_path design_name} {
  echo "#PROJ: Performing check_test";
  P_report_check_test $report_path $design_name;

  echo "#PROJ: Inserting scan chains";

  # Keeps insert_scan from placing inverters in scan path
  set test_disable_find_best_scan_out true;
  set test_dont_fix_constraint_violations true;

  # Insert scan chains based on variables set

```



```

#insert_scan;
insert_scan -ignore_compile_design_rules;

}; # end P_proj_postcompile_scan

define_proc_attributes P_proj_postcompile_scan \
    -info "PROJ_PROCS : Procedure to insert scan"

#####
# Usage: P_proj_outputs {dbs_path design_name outputs_path srcline \
#                      {out_string ""} {out_type ""}}
#
# This procedure generates the default outputs. This includes db, sdf, sdc &
# verilog gate netlist.
# srcline is a list.
# NOTE: If out_string is null, the proc will only output db & verilog netlist.
#       If out_type is "layout", all layout related info is written.
#####
proc P_proj_outputs {dbs_path design_name outputs_path srcline {out_string ""} {out_type
""}} {
    global bus_naming_style;
    global hdlout_internal_busses;
    global set_fix_multiple_port_nets;
    global verilout_single_bit;

    #####
    # Write a mapped DB file
    #####
    # By default, out_string = ". If USER requires other name, set $out_string
    set output_file [file join ${dbs_path} ${design_name}${out_string}.db];

    echo "#OUTPUTS: Writing out mapped db to ${output_file}";
    write -f db -hier -o $output_file;

    #####
    # Set naming style comments (VHDL or Verilog)
    #####
    set timestamp [exec date];

    #####
    # Comments for verilog and sdf
    #####
    set vlog_comment "//";
    set vlog_info "$vlog_comment This netlist was created with the following source
files:";
    set vlog_timestamp "$vlog_comment $timestamp";
    set vlog_cvslines "$vlog_comment \${Id}\$";

    #####
    # Verilog Gates
    #####
    if {${out_string} == ""} {
        set out_string "_gates"
    }

    set output_file [file join ${outputs_path} ${design_name}${out_string}.v];
    echo "#OUTPUTS: Writing verilog netlist to ${output_file}";
    P_get_license "HDL-Compiler";

    write -f verilog -hier -o $output_file;

```

```

redirect -append $output_file {echo $vlog_info};
redirect -append $output_file {echo $vlog_timestamp};

# Dump the CVS ID's for all elements in the srcline list
foreach srcline_elem $srcline {
    echo $srcline_elem;
    set vlog_srcline "$vlog_comment netlist CVS srcid: $srcline_elem";
    redirect -append $output_file {echo $vlog_srcline};
}; # end foreach
redirect -append $output_file {echo $vlog_cvsline};

remove_license "HDL-Compiler";

#####
# Write SDF file
#####
set output_file [file join ${outputs_path} ${design_name}${out_string}.max.sdf];
echo "#OUTPUTS: Writing sdf timing data to ${output_file}";
write_sdf -version 2.1 $output_file;

redirect -append $output_file {echo $vlog_info};
redirect -append $output_file {echo $vlog_timestamp};

# Dump the CVS ID's for all elements in the srcline list
foreach srcline_elem $srcline {
    echo $srcline_elem;
    set vlog_srcline "$vlog_comment netlist CVS srcid: $srcline_elem";
    redirect -append $output_file {echo $vlog_srcline};
}; # end foreach
redirect -append $output_file {echo $vlog_cvsline};

# By default, don't write the following files
if {$out_type == "layout"} {

    #####
    # Writes Synopsys Design Constraints (SDC) V1.1 format
    #####
    set output_file [file join ${outputs_path} ${design_name}${out_string}.sdc];
    echo "#OUTPUTS: Writing sdc synopsys constraints to ${output_file}";
    write_sdc $output_file;

}; # end if

#####
# Finished writing outputs
#####
echo "#OUTPUTS: Finished executing P_proj_outputs";

}; # end of proc P_proj_outputs

define_proc_attributes P_proj_outputs \
    -info "PROJ_PROCS: Procedure to generate default outputs This includes the db, sdf, \
sdc & verilog gate netlist" \
    -define_args {
        {dbs_path "Path to the dbs file" dbs_path string required}
        {design_name "Name of the design" design_name string required}
        {outputs_path "Path to the output files" outputs_path string required}
        {srcline "Source code revision information" srcline string required}
        {out_string "Output filename string" out_string string}
        {out_type "Output types string" out_type string}
    }

```

```

#####
# Usage: P_proj_ungroup_dw
#
# This procedure is used to ungroup the DesignWare components. This is
# useful to remove unconnected ports (caused layout issues)
#####
proc P_proj_ungroup_dw {} {
    global G_TOPS;
    global current_design;

    echo "#UNGROUP_DW: ungrouping designware parts"

    # Save current location
    set this_design $current_design;

    # Need to traverse through hierarchy & flatten DW components
    redirect $G_TOPS(NULL) {set hier_designs [filter [find design *] {@is_hierarchical ==
true}]];
    if {$hier_designs != {}} {
        foreach_in_collection tmp_design $hier_designs {
            redirect $G_TOPS(NULL) {set current_design $tmp_design};
            redirect $G_TOPS(NULL) {set cell_list [filter [find cell *] {@DesignWare ==
true}]];
            if {$cell_list != {}} {
                foreach_in_collection tmp_cell $cell_list {
                    echo "#UNGROUP_DW: ungrouping" [get_object_name $tmp_cell] \
                        "in design" [get_object_name $tmp_design];
                    ungroup $tmp_cell -flatten -prefix DW_ -simple_names;
                }; # end foreach_in_collection
            }; # end if
            unset cell_list;
        }; # end foreach_in_collection
        unset tmp_design;
    }; # end if

    # Reset current back to where we started from */
    redirect $G_TOPS(NULL) {set current_design $this_design}

    # Clean up namespace
    unset this_design
    unset hier_designs
}; # end P_proj_ungroup_dw

define_proc_attributes P_proj_ungroup_dw \
    -info "PROJ_PROCS: Used to ungroup the Design Ware components for the design"

#####
# Usage: P_proj_wire_load
#
# This procedure is used to set the default wireload for the target.
# Assumes G_DEFAULT_WIRE_LOAD set from proj_library script
#####
proc P_proj_wire_load {} {
    global G_DEFAULT_WIRE_LOAD;

    echo "#PROJ: Setting wireload";

    set_wire_load_model -name $G_DEFAULT_WIRE_LOAD;
    set_wire_load_mode top;
    set auto_wire_load_selection false;

```

```

}; # end P_proj_wire_load

define_proc_attributes P_proj_wire_load \
    -info "PROJ_PROCS: Sets the default wire load model"

#####
# $Log$
#####

#####
# File Name      : proj_constraints.tcl
# Date Created   : 12/06/99
# Author        : Tim L Wilson, Gregg D. Lahti & Rodney Pesavento
# Project       : TOPS Example Code for SNUG
#####
# $Id$
#####
# Description: Set up project specific constraints
#####

#####
# Customize the defaults here
#####
# Default unit-to-unit I/O delays, and which clock they reference
set G_DEFAULT_INPUT_DELAY [expr .750 * $G_NS];
set G_DEFAULT_OUTPUT_DELAY [expr .750 * $G_NS];

# Default cell loading output ports
set G_DEFAULT_LOAD [load_of [file join $G_LIBRARY_NAME $G_DEFAULT_LOAD_CELL
$G_DEFAULT_LOAD_CELL_PIN]];
set G_DEFAULT_LOAD [expr 10 * $G_DEFAULT_LOAD];

#####
# The following variables are included as possible constraints that
# may be set by the project. Uncomment the commands below.
#####
# Set max area, Synopsys recommends setting close to expected area (NOT 0)
#set G_DEFAULT_MAXAREA 0;

# Set max fanout (allow library to constrain)
#set G_DEFAULT_MAX_FANOUT 16;

#####
# Set critical range so that more than just the critical path gets
# optimized. The compile command only works on the path with the
# greatest negative slack, so this command makes compile work on
# more paths.
#####
#set G_DEFAULT_CRITICAL_RANGE 0.0;

#####
# You shouldn't have to edit the rest of this file if you set the
# variables above and the simplistic case is all you need
#####
echo "";
echo "#CONSTRAINTS: DEFAULT CONSTRAINTS";
echo "#CONSTRAINTS: -----";
echo "#CONSTRAINTS: DRIVING_CELLPIN -> "
$G_DEFAULT_DRIVING_CELL/$G_DEFAULT_DRIVING_CELL_PIN;

```

```

echo "#CONSTRAINTS: LOAD          -> " $G_DEFAULT_LOAD;
#echo "#CONSTRAINTS: AREA          -> " $G_DEFAULT_MAXAREA;
#echo "#CONSTRAINTS: CRITICAL_RANGE -> " $G_DEFAULT_CRITICAL_RANGE;
#echo "#CONSTRAINTS: FANOUT        -> " $G_DEFAULT_MAX_FANOUT;
echo "";

#####
# Set max_area, critical range, fanout, fix joined ports
#####
#set_max_area $G_DEFAULT_MAXAREA;
#set_critical_range $G_DEFAULT_CRITICAL_RANGE $G_DESIGN_NAME;
#set_max_fanout $G_DEFAULT_MAX_FANOUT $G_DESIGN_NAME;

# Variable used to cleanup verilog assign statements
set_fix_multiple_port_nets -all;

#####
# Create a list of all inputs that are not clock pins
#####
set clk_ports [get_ports -filter {@is_clock == true} "*"];
set rst_ports [get_ports -filter {@is_reset == true} "*"];

set in_list [remove_from_collection \
             [get_ports -filter {@port_direction == in} "*"] \
             [concat $clk_ports $rst_ports]];

set out_list [all_outputs];

#####
# Set default input and output delays
#####
if {[info exists G_DEFAULT_CLOCK]} {
    echo "#CONSTRAINTS: setting default delay with respect to $G_DEFAULT_CLOCK";
    set_output_delay $G_DEFAULT_OUTPUT_DELAY -clock $G_DEFAULT_CLOCK $out_list;
    set_input_delay $G_DEFAULT_INPUT_DELAY -clock $G_DEFAULT_CLOCK $in_list;
}

#####
# Set default driving cell for input pins
#####
set_driving_cell \
    -cell $G_DEFAULT_DRIVING_CELL \
    -pin $G_DEFAULT_DRIVING_CELL_PIN \
    -library $G_LIBRARY_NAME $in_list;

#####
# Set default load on all outputs
#####
set_load $G_DEFAULT_LOAD [all_outputs];

#####
# Clean up local variables
#####
unset in_list;
unset out_list;
unset G_DEFAULT_INPUT_DELAY;
unset G_DEFAULT_OUTPUT_DELAY;
unset G_DEFAULT_LOAD;

#####
# $Log$

```

```

#####

#####
# File Name      : proj_reports.tcl
# Date Created   : 12/01/99
# Author        : Tim L. Wilson, Gregg D. Lahti & Rodney Pesavento
# Project       : TOPS Example Code for SNUG
#####
# $Id$
#####
# Description:   Project specific reports procs. Add others as needed.
#####

#####
# Procedure to run all proj reports and TOPS scripts
#####
proc P_proj_reports {rpath dn} {
    global G_TOPS; # reference global TOPS variable array

    # This is a list of the TOPS report procs. Comment & use as needed
    P_report_area $rpath $dn;
    P_report_check_design $rpath $dn;
    P_report_check_design_ungroup $rpath $dn;
    P_report_check_timing $rpath $dn;
    P_report_clock $rpath $dn;
    P_report_compile_options $rpath $dn;
    P_report_design $rpath $dn;
    P_report_dont_use_cells $rpath $dn;
    P_report_ignored $rpath $dn;
    P_report_port $rpath $dn;
    P_report_reference $rpath $dn;
    P_report_reference_ungroup $rpath $dn;
    P_report_test $rpath $dn;
    P_report_unmapped $rpath $dn;
    P_report_vio $rpath $dn;
    P_report_vio_max_cap $rpath $dn;
    P_report_vio_max_delay $rpath $dn;
    P_report_vio_min_delay $rpath $dn;
    P_report_ins2flops $rpath $dn;
    P_report_flops2outs $rpath $dn;
    P_report_ins2outs $rpath $dn;
    P_report_flop_latch $rpath $dn;
    P_report_timing_requirements $rpath $dn;

    redirect $G_TOPS(NULL) {exec grm -rf $G_TOPS(TMP)}

}; # end P_proj_reports

define_proc_attributes P_proj_reports \
    -info "PROJ_REPORTS : Procedure to run all project reports and TOPS scripts" \
    -define_args {
        {rpath "path to the report file" rpath string required}
        {dn "design name" dn string required}
    }

#####
# $Log$
#####

```

```

#####
# File Name      : util_reports.tcl
# Date Created   : 12/01/99
# Author        : Tim L. Wilson, Gregg D. Lahti & Rodney Pesavento
# Project       : TOPS Example Code for SNUG
#####
# $Id$
#####
# Description:   Contains the TOPS general reporting procedures.
#####

#####
# Report area
#####
proc P_report_area {rpath dn} {
    set output_file [file join $rpath ${dn}_area.rpt];
    echo "#REPORTS: report_area: in $output_file";
    redirect $output_file {report_area -nosplit};
    unset output_file;
}; # end P_report_area

#####
# check_design
#####
proc P_report_check_design {rpath dn} {
    set output_file [file join $rpath ${dn}_check_design.rpt];
    echo "#REPORTS: check_design: in $output_file";
    redirect $output_file {[check_design]};
    unset output_file;
}; # end P_report_check_design

#####
# Do a check_design, *but* ungroup -all first
#####
proc P_report_check_design_ungroup {rpath dn} {
    global G_TOPS;
    set output_file [file join $rpath ${dn}_design_ungroup.rpt];
    echo "#REPORTS: check_design ungroup: in $output_file";

    ##set tmpfile [file join $G_TOPS(TMP) ${dn}${timestamp}];
    set tmpfile ${dn}[eval pid][clock seconds].tmp

    echo "#REPORTS: check_design ungroup: saving db before ungroup";
    write -f db -hier -o $tmpfile;
    echo "#REPORTS: check_design ungroup: ungrouping design";
    ungroup -all;
    redirect $output_file {check_design};

    remove_design -designs;
    echo "#REPORTS: check_design ungroup: re-reading original design";
    read_file -f db $tmpfile;

    current_design $dn;

    redirect $G_TOPS(NULL) {exec grm -f $tmpfile};
    unset output_file;
}; # end P_report_check_design_ungroup

#####

```

```

# Do a check_timing
#####
proc P_report_check_timing {rpath dn} {
    set output_file [file join $rpath ${dn}_check_timing.rpt];
    echo "#REPORTS: check_timing: in $output_file";
    redirect $output_file {check_timing};
    unset output_file;
}; # end P_report_check_timing

#####
# Do a check_test
#####
proc P_report_check_test {rpath dn} {
    set output_file [file join $rpath ${dn}_check_test.rpt];
    echo "#REPORTS: check_test: in $output_file";
    redirect $output_file {check_test};
    unset output_file;
}; # end P_report_check_test

#####
# Report clocks
#####
proc P_report_clock {rpath dn} {
    set output_file [file join $rpath ${dn}_report_clock.rpt];
    echo "#REPORTS: report_clock & -skew: $output_file";
    redirect $output_file {report_clock -nosplit};
    redirect -append $output_file {report_clock -skew -nosplit};
    redirect -append $output_file {report_timing_requirements -to [all_clocks]};
    redirect -append $output_file {report_clock -attributes -nosplit};
    unset output_file;
}; # end P_report_clock

#####
# Report compile options
#####
proc P_report_compile_options {rpath dn} {
    set output_file [file join $rpath ${dn}_compile_options.rpt];
    echo "#REPORTS: compile_options: in $output_file";
    redirect $output_file {report_compile_options -nosplit};
    unset output_file;
}; # end P_report_compile_options

#####
# Report design
#####
proc P_report_design {rpath dn} {
    set output_file [file join $rpath ${dn}_design.rpt];
    echo "#REPORTS: report design: in $output_file";
    redirect $output_file {report_design -nosplit};
    unset output_file;
}; # end P_report_design

#####
# Report library cells that have dont_use attribute
#####
proc P_report_dont_use_cells {rpath dn} {
    global G_LIBRARY_NAME;
    set output_file [file join $rpath ${dn}_${G_LIBRARY_NAME}_dont_use_cell.rpt];
    echo "#REPORTS: Generating file of cells with dont_use attribute set";

    # Create list of cells

```



```

set cell_list [filter [find cell $G_LIBRARY_NAME/*] "@dont_use == true"];
redirect $output_file {echo "Library cells with dont_use attribute set";}
foreach_in_collection cell_elem $cell_list {
    # Echo cells to file
    redirect -append $output_file {echo [get_object_name $cell_elem]};
}
unset output_file;
}; # end P_report_dont_use_cells

#####
# Report ignored timing constraints
#####
proc P_report_ignored {rpath dn} {
    set output_file [file join $rpath ${dn}_ignored.rpt];
    echo "#REPORTS: report ignored: in $output_file";
    redirect $output_file {report_timing_requirements -ignored -nosplit};
    unset output_file;
}; # end P_report_ignored

#####
# Report timing constraints
#####
proc P_report_timing_requirements {rpath dn} {
    set output_file [file join $rpath ${dn}_timing_req.rpt];
    echo "#REPORTS: report timing requirements: in $output_file";
    redirect $output_file {report_timing_requirements -nosplit};
    unset output_file;
}; # end P_report_timing_requirements

#####
# Report all endpoint margins
#####
proc P_report_endpoints {rpath dn} {
    set output_file [file join $rpath ${dn}_endpoints.rpt];
    echo "#REPORTS: report endpoints: in $output_file";
    redirect $output_file {report_timing -path end -nosplit};
    unset output_file;
}; # end P_report_endpoints

#####
# Total nets / pins per net report
#####
proc P_report_pins_per_net {rpath dn} {
    set output_file [file join $rpath ${dn}_pins_per_net.rpt];
    echo "#REPORTS: pins_per_net: in $output_file";
    redirect $output_file {report_net -nosplit};
    unset output_file;
}; # end P_report_pins_per_net

#####
# Report ports
#####
proc P_report_port {rpath dn} {
    set output_file [file join $rpath ${dn}_port.rpt];
    echo "#REPORTS: report_port: in $output_file";
    redirect $output_file {report_port -verbose -nosplit};
    unset output_file;
}; # end P_report_port

#####
# Report resources

```

```
#####
proc P_report_reference {rpath dn} {
    global G_TOPS;
    set output_file [file join $rpath ${dn}_reference.rpt];
    echo "#REPORTS: report_reference: $output_file";
    redirect $output_file {report_reference -nosplit};
    unset output_file;
}; # end P_report_reference

#####
# Report resources, but ungroup -all first
#####
proc P_report_reference_ungroup {rpath dn} {
    global G_TOPS;

    set output_file [file join $rpath ${dn}_reference_ungroup.rpt];
    set tmpfile ${dn}[eval pid][clock seconds].tmp

    echo "#REPORTS: report_reference ungroup: in $output_file";
    write -f db -hier -o $tmpfile;
    echo "#REPORTS: report_reference ungroup: ungrouping design";
    ungroup -flatten -all;
    echo "#REPORTS: report_reference ungroup: doing report_reference";
    redirect $output_file {report_reference -nosplit};

    remove_design -designs;
    echo "#REPORTS: report_reference ungroup: re-reading original design";
    read_file -f db $tmpfile;
    current_design $dn;

    redirect $G_TOPS(NULL) {exec grm -f $tmpfile};

    unset output_file;
    unset tmpfile;
}; # end P_report_reference_ungroup

#####
# Report test
#####
proc P_report_test {rpath dn} {
    set output_file [file join $rpath ${dn}_report_test.rpt];
    echo "#REPORTS: report scan insertion: in $output_file";
    redirect $output_file {report_test -configuration -port -scan_path -nosplit};
    unset output_file;
}; # end P_report_test

#####
# Script to detect unmapped parts
#####
proc P_report_unmapped {rpath dn} {
    global G_TOPS;

    set output_file [file join $rpath ${dn}_unmapped.rpt];
    echo "#REPORTS: report unmapped: in $output_file";
    echo "#REPORTS: report unmapped: checking for unmapped components";
    redirect $G_TOPS(NULL) \
        {set gtech_cells [get_cells * -hier -filter "@is_unmapped==true"]};

    if { $gtech_cells != {} } {
        echo "#REPORTS: report unmapped: design has unmapped cells (ERROR)";
        redirect $output_file {report_cell -nosplit $gtech_cells};
    }
}

```

```

    } else {
        echo "#REPORTS: report unmapped: there are no unmapped cells in the design";
        redirect $output_file \
            {echo "#REPORTS: unmapped: there are no unmapped cells in the design"};
    }
    unset gtech_cells;
    unset output_file;
}; # end P_report_unmapped

#####
# Report constraints
#####
proc P_report_vio {rpath dn} {
    set output_file [file join $rpath ${dn}_all_violators.rpt];
    echo "#REPORTS: all_violators: in $output_file";
    redirect $output_file {report_constraint -all_violators -nosplit};
    unset output_file;
}; # end P_report_vio

#####
# Report all max cap violations
#####
proc P_report_vio_max_cap {rpath dn} {
    set output_file [file join $rpath ${dn}_vio_max_cap.rpt];
    echo "#REPORTS: all_violators max_cap: in $output_file";
    redirect $output_file {report_constraint -all_violators -max_cap};
    unset output_file;
}; # end P_report_vio_max_cap

#####
# Report all max delay violations
#####
proc P_report_vio_max_delay {rpath dn} {
    set output_file [file join $rpath ${dn}_vio_max_delay.rpt];
    echo "#REPORTS: all_violators max_delay: in $output_file";
    redirect $output_file {report_constraint -all_violators -max_delay -verbose};
    unset output_file;
}; # end P_report_vio_max_delay

#####
# Report all min delay violations
#####
proc P_report_vio_min_delay {rpath dn} {
    set output_file [file join $rpath ${dn}_vio_min_delay.rpt];
    echo "#REPORTS: all_violators min_delay: in $output_file";
    redirect $output_file {report_constraint -all_violators -min_delay -verbose};
    unset output_file;
}; # end P_report_vio_min_delay

#####
# Inputs to Registers
#####
proc P_report_ins2flops {rpath dn} {
    set output_file [file join $rpath ${dn}_ins2flops.rpt];
    echo "#PROJ_REPORTS: report_ins2flops: in $output_file";

    set in_list [remove_from_collection \
        [get_ports -filter {@port_direction == in} "*"] \
        [get_ports -filter {@is_clock == true} "*"]];

    redirect $output_file {echo ""};
}

```

```

foreach inpin $in_list {
    redirect -append $output_file {report_timing -path short -from $inpin \
        -to [all_registers] -nets -trans -max_paths 5 -nosplit};
}

unset inpin;
unset in_list;
unset output_file;
}; # end P_report_ins2flops

#####
# Registers to Outputs
#####
proc P_report_flops2outs {rpath dn} {
    set output_file [file join $rpath ${dn}_flops2outs.rpt];
    echo "#PROJ_REPORTS: report_flops2outs: in $output_file";

    redirect $output_file {echo ""};

    foreach outpin [all_outputs] {
        redirect -append $output_file {report_timing -path short \
            -from [all_registers -clock_pins] -to $outpin -nets -trans -nosplit};
    }

    unset outpin;
    unset output_file;
}; # end P_report_flops2outs

#####
# Inputs to Outputs
#####
proc P_report_ins2outs {rpath dn} {
    set output_file [file join $rpath ${dn}_ins2outs.rpt];
    echo "#PROJ_REPORTS: report_ins2outs: in $output_file";

    set in_list [remove_from_collection \
        [get_ports -filter {@port_direction == in} "*"] \
        [get_ports -filter {@is_clock == true} "*"]];

    redirect $output_file {report_timing -path short -from $in_list \
        -to [all_outputs] -nets -trans -max_paths 5 -nosplit};

    unset in_list;
    unset output_file;
}; # end P_report_ins2outs

#####
# Report number of Registers & Latches
#####
proc P_report_flop_latch {rpath dn} {
    set output_file [file join $rpath ${dn}_flop_latch.rpt];
    echo "#PROJ_REPORTS: report_flop_latch: in $output_file";

    set flop_cnt [sizeof_collection [all_registers -edge_triggered]];
    set latch_cnt [sizeof_collection [all_registers -level_sensitive]];

    redirect $output_file {echo "Total number of flops in $dn = $flop_cnt"};
    redirect -append $output_file \
        {echo "Total number of latches in $dn = $latch_cnt"};
}

```

```

    unset output_file;
}; # end P_report_flop_latch

#####
# Report timing requirements
#####
proc P_report_timing_requirements {rpath dn} {
    set output_file [file join $rpath ${dn}_timing_requirements.rpt];
    echo "#PROJ_REPORTS: report_timing_requirements: in $output_file";

    redirect $output_file {report_timing_requirements -to [all_clocks]};

    unset output_file;
}; # end P_report_timing_requirements

#####
# $Log$
#####

#!/usr/intel/bin/perl
#####
# File Name      : synbatch
# Date Created   : 6/16/00
# Author        : Doug Hergatt, Gregg D. Lahti & Tim L. Wilson
# Project       :
#####
# $Id$
#####
# Description:
# This script is used to drive the synthesis compilation of a target.
# The G_DESIGN_FILE_NAME, G_SUBFUBS, & G_LIBRARY_NAME are passed in
# from gmake (or syn_make.rules).
#####

# Check for project specific environment variables
die "Environment variable PROJ_NAME is not defined!\n" if (!defined($ENV{PROJ_NAME}));
die "Environment variable PROJ_SYN is not defined!\n" if (!defined($ENV{PROJ_SYN}));

$proj_name = $ENV{PROJ_NAME};
$proj_syn = $ENV{PROJ_SYN};

# This is the project setup script
$proj_script = $ENV{PROJ_SYN} . "/" . "tops_setup";

# Setup interrupt handler to die on SIGINT (ctrl-c)
$SIG{INT} = 'die_on_error';

# Set the hostname
$machine = `hostname`;

# Get design name from make (stripped of path & suffix from gmake)
$design_file_name = $ARGV[0];

# Strip off path & extension
$design = $design_file_name;
$design =~ s/(\S+)\..*/$1/;
$design =~ s%.*?([^/]+)%$1%;

# Create log file

```

```

$logfile = "log/${design}.log";
# Retain last copy of log file
if (-e $logfile) {
    system("mv $logfile $logfile.1");
}

# Parse the command line for arguments
shift @ARGV;
while (@ARGV) {
    if ($ARGV[0] =~ /LIB=(.+)/) {
        $lib = $1;
    } elsif ($ARGV[0] =~ /SUBFUB=(.+)/) {
        $subfub_list = $1;
    } elsif ($ARGV[0] =~ /HOST=(.+)/) {
        $rsh_host = $1;
    }
    shift @ARGV;
}

# Print info on current compilation process
print "\nExecuting with arguments:\n";
print "G_LIBRARY_FILE: $lib\n";
print "G_DESIGN_FILE_NAME: $design_file_name\n";
print "G_SUBFUBS: $subfub_list\n";
print "REMOTE_HOST: $rsh_host\n\n";

# Set default, TOPS will assign actual default lib
if ($lib eq "") {
    $lib = "DEFAULT";
}

# This builds the command line that will be executed below
$line = "dc_shell-t -x \"set G_DESIGN_FILE_NAME $design_file_name; set G_SUBFUBS \[list
$subfub_list\]; set G_LIBRARY_NAME $lib; source ${proj_syn}/proj_main.tcl\" ";

# if running on remote machine, add rsh, and extra commands to front of line to execute

$CMD_FILE = $rsh_host . ".csh";
open (CMD_FILE_HANDLE,">$CMD_FILE") || die "Error: can't open $CMD_FILE file";

if ($rsh_host ne "") {
    print CMD_FILE_HANDLE "#!/bin/tcsh\n";
    print CMD_FILE_HANDLE "# auto generated command file for running TOPS remotely\n\n";

    $pwd = `pwd`;
    chomp($pwd);
    print CMD_FILE_HANDLE "cd $pwd;\n";

    print CMD_FILE_HANDLE "source $proj_script;\n\n";
    print CMD_FILE_HANDLE "echo \"Starting dc_shell session\"\n\n";

    print CMD_FILE_HANDLE "dc_shell-t -x \"set G_DESIGN_FILE_NAME $design_file_name; set
G_SUBFUBS [list $subfub_list]; set G_LIBRARY_NAME $lib; source
${proj_syn}/proj_main.tcl\";\n\n";
    print CMD_FILE_HANDLE "echo \"Finished remote shell script\"\n\n";
    close (CMD_FILE_HANDLE);
    system("chmod +x $CMD_FILE");
    $line = "rsh $rsh_host \"'$pwd/$CMD_FILE'\"";
    $machine = $rsh_host;
}

```

```

# Fork our process into parent/child processes
FORKU: {
  if ($pid = fork) {
    # print "\nparent pid is $$\n";
    # this is the parent process
    #waitpid $pid,0;
    wait;
  } elseif (defined $pid) {
    # this is the child process
    # execute actual command, pipe off verbage crap from rsh to bit-bucket
    print "Compiling $design on $machine\n";
    # Execute the command file (remote) or line generated above
    exec("$line 2>&1 > $logfile");
    $childstatus = $?;
    # print "\nchild exited with: $childstatus, $childstatus2\n";

    # check for resultant db, if it isn't die with a fail exit status
    if (-e "$pwd/dbs/$design.db") {
      $errorstatus = 1;
    }
  } elseif ($! =~ /No more process/) {
    # Barf! A supposedly recoverable fork process
    sleep 5;
    redo FORKU;
  } else {
    # weird fork error
    die "Can't fork: $!\n";
  }
}

# We're done, cleanup and exit nicely
if ($childstatus != 0 or $errorstatus != 0) {
  exit(1);
} else {
  system("rm $CMD_FILE");
  exit(0);
}

# trap & cleanup subroutines
sub die_on_error {
  local ($sig) = @_;
  # print STDOUT "trapping signal $sig!\n";
  kill 9, $pid++;
  kill 9, $pid++;
  kill 9, $pid++;
  kill 9, $pid;
  exit(1);
}

#####
# $Log$
#####

#####
# File Name      : util_procs.tcl
# Date Created   : 12/06/99
# Author         : Tim L Wilson, Gregg D. Lahti & Rodney Pesavento
# Project        : TOPS Example Code for SNUG
#####

```

```

# $Id$
#####
# Description: TOPS utility procedures for synthesis
#####

# Display the TOPS environment release
set G_TOPS_VER 1.5;

echo "#SETUP: Sourcing TOPS procedure file ($G_PROJ_SYN/util_procs.tcl)";
echo "#SETUP: TOPS Environment Version: $G_TOPS_VER\n";

#####
# Usage: P_elapsed_time mark
# This procedure sets up a global var to allow procs to mark time of scripts
#####
proc P_elapsed_time {mark} {
    set now [clock seconds];
    clock format [expr $now - $mark -(17*3600)] -format %H:%M:%S;
}; # end elapsed_time;

define_proc_attributes P_elapsed_time \
    -info "TOPS: Set up a global variable to allow procedures to mark time of scripts" \
    -define_args {
        {mark "mark is the time mark" mark float required}
    }

#####
# Usage: P_timestamp
# This procedure prints the timestamp into the log file
#####
proc P_timestamp {} {
    set c [clock format [clock seconds]];
    set h [sh hostname];
    set u [sh whoami];
    echo "#TIMESTAMP: " $c " " $h " " $u;
}; # end P_timestamp;

define_proc_attributes P_timestamp \
    -info "TOPS: Puts the unix time stamp into the log files"

#####
# Usage: P_get_srcline filename
# This procedure to get the Id CVS string from the source file.
# NOTE: filename: a fully qualified path and filename or a path and
#       filename relative to the unit's syn dir.
#####
proc P_get_srcline {fn sfm} {
    echo "#INFO: (P_get_srcline) : Extracting revision data from file(s) " $fn $sfm;

    # Initialize file list & srcline list variables
    set file_list [concat $fn $sfm];
    set srcline_list "";

    foreach src_file $file_list {
        set srcline [exec sed -n -e "/Id: /p" $src_file];

        # Check if srcline set (doesn't get set when file doesn't have CVS header
        if {$srcline == ""} {

```



```

        echo "#WARNING: No extracted revision data from file (please append header
template)";
        set srcline "No CVS Id for $src_file (please append header template)"
    }
    regexp {[\\$](.+)\\$} $srcline srcline srcline;
    lappend srcline_list $srcline;
}; # end foreach

return $srcline_list;
}; # end P_get_srcline

define_proc_attributes P_get_srcline \
    -info "TOPS: Procedure to get the Id CVS string from the source file" \
    -define_args {
        {fn "a fully qualified path and filename or a path and filename relative to the
unit's syn dir" fn string required}
        {sfn "list of fully qualified subfub files" sfn string required}
    }

#####
# Usage: P_source_if_exists filename caller
# This procedure is used to see if a file exists.  If it doesn't, return a zero.
# caller added to print invoking script's name
#####
proc P_source_if_exists {filename caller} {
    if {[which $filename] != ""} {
        set LOCAL_TIME_MARK [clock seconds];
        set full_name [which $filename];
        echo "$caller: Sourcing $full_name";
        # Source the file in the top-level context
        uplevel source $filename;
        # echo "$caller: Elapsed Time for Sourcing $full_name: [P_elapsed_time
$LOCAL_TIME_MARK]";
        return 1;
    } else {
        # File was not found
        return 0;
    }
}; # end P_source_if_exists

define_proc_attributes P_source_if_exists \
    -info "TOPS: Procedure to source file if it exists" \
    -define_args {
        {filename "File name" filename string required} \
        {caller "Invoking script name" caller string required}
    }

#####
# Usage: P_exec_if_exists filename caller
# Procedure to exec procedure if it exists.  If it doesn't, return
# caller added to print Invoking script's name.
#####
proc P_exec_if_exists {procname caller} {

    # Test to see if Procedure exists
    if {[info proc ${procname}] == ${procname}} {
        # Procedure exists, execute
        echo "$caller: Executing ${procname}";
        # Evaluate the proc in the top-level context
        uplevel $procname;
        return 1;
    }
}

```

```

    } else {
        # Procedure does not exist, return
        return 0;
    }
}; # end P_exec_if_exists

define_proc_attributes P_exec_if_exists \
    -info "TOPS: Procedure to execute procedure if it exists" \
    -define_args {
        {procname "Procedure name" procname string required} \
        {caller "Invoking script name" caller string required}
    }

#####
# Usage: P_get_license HDL-Compiler
# This procedure grabs the specified license. Note that DC gives error status
# if we already have the license. Hence, must determine if we
# do have the license first before we actually get it.
# Also re-check for the license in 60 second intervals for
# 1 hour before we exit with an error if we can't get a license.
#####
proc P_get_license {ln} {

    set fetch_license 1;
    set sleep_val 60;          # sleep in seconds we wait for a license
    set max_timeout 60;        # number of 60-second waits until we die

    # First must determine which licenses we have. To do this
    # we get the output of list_licenses into a TCL variable.
    #
    # Hack! DC is broke & can't set list_lic output to a variable,
    # so must stuff it to a tmp file & read it back in. Use process
    # id (pid) in filename as a safer-alternative, touch & rm -rf to
    # first to be extra safe
    exec touch [eval pid].tmp;
    exec rm -rf [eval pid].tmp;
    redirect [eval pid].tmp {list_licenses};
    set licenses [exec cat [eval pid].tmp];
    exec rm -rf [eval pid].tmp;

    # now that we have a TCL variable with the list_license output,
    # strip out the crap that DC uselessly puts in and parse the list
    regsub {Licenses in use:} $licenses {} licenses;
    foreach checkedout_license $licenses {
        if [string match $checkedout_license $ln] {
            set fetch_license 0;
        }
    }
}

if {$fetch_license == 1} {
    set sleep_time 0;
    set dc_status [get_license $ln];
    while { ($dc_status == 0) && ($sleep_time < $max_timeout) } {
        if {$sleep_time == 0} {
            set current_time [exec date];
            echo "#INFO: waiting for license $ln @ $current_time";
        }
        sh sleep $sleep_val;
        set sleep_time [expr $sleep_time + 1];
        set dc_status [get_license $ln];
    }
    if {$dc_status == 0} {

```

```

        echo "Error: Cannot get license $ln after $sleep_time seconds, dying!";
        return 0;
    } else {
        #echo "#INFO: checked out license $ln";
        return 1;
    }
} else {
    #echo "#INFO: already have license, continuing along";
    return 1;
}
}; # end P_get_license

define_proc_attributes P_get_license \
    -info "TOPS: Procedure to get a license" \
    -define_args {
        {ln "license name" ln string required}
    }
}

#####
# Usage: P_check4lic MOTIVE-PrimeTime
# This procedure is used to check for a particular license.
# Returns a 1 if the license is checked out.
# Returns a 0 if not checked out.
# Can be used to determine if PT or DC is running.
#####
proc P_check4lic {ln} {
    set license_exists 0;

    # First must determine which licenses we have. To do this
    # we get the output of list_licenses into a TCL variable.
    #
    # Hack! DC is broke & can't set list_lic output to a variable,
    # so must stuff it to a tmp file & read it back in. Use process
    # id (pid) in filename as a safer-alternative, touch & rm -rf to
    # first to be extra safe
    exec touch [eval pid].tmp;
    exec rm -rf [eval pid].tmp;
    redirect [eval pid].tmp {list_licenses};
    set licenses [exec cat [eval pid].tmp];
    exec rm -rf [eval pid].tmp;

    # Now that we have a TCL variable with the list_license output,
    # strip out the crap that DC uselessly puts in and parse the list
    regsub {Licenses in use:} $licenses {} licenses;
    foreach checkedout_license $licenses {
        if [string match $checkedout_license $ln] {
            set license_exists 1;
        }
    }
    return $license_exists;
}; # end P_check4lic

define_proc_attributes P_check4lic \
    -info "TOPS: Procedure to check for a particular license" \
    -define_args {
        {ln "license name" ln string required}
    }
}

#####
# Usage: P_read_vhdl filename.vhd
# This procedure is used to read the VHDL source code file.

```

```

# Library may be optionally specified.
#####
proc P_read_vhdl {filename {libname default}} {
    P_get_license "VHDL-Compiler";
    if {$libname == "default"} {
        analyze -f vhdl $filename;
    } else {
        analyze -library $libname -f vhdl $filename;
    }
    remove_lic "VHDL-Compiler";
}; # end P_read_vhdl;

define_proc_attributes P_read_vhdl \
    -info "TOPS: Procedure to read a vhdl file" \
    -define_args {
        {filename "VHDL File Name" filename string required}
        {libname "Library Name" libname string}
    }
}

#####
# Usage: P_read_verilog filename.v
# This procedure is used to read the Verilog source code file.
# Library may be optionally specified.
#####
proc P_read_verilog {filename {libname default}} {
    P_get_lic "HDL-Compiler";
    if {$libname == "default"} {
        analyze -f verilog $filename;
    } else {
        analyze -library $libname -f verilog $filename;
    }
    remove_lic "HDL-Compiler";
}; # end P_read_verilog;

define_proc_attributes P_read_verilog \
    -info "TOPS: Procedure to read a verilog file" \
    -define_args {
        {filename "Verilog File Name" filename string required}
        {libname "Library Name" libname string}
    }
}

#####
# Usage: P_read_top top_unit
# This procedure is used to read in top unit (language neutral).
# Other file extensions may be added as required
#####
proc P_read_top {filename {libname default}} {
    set design_name [file tail [file rootname $filename]];
    echo "#INFO: P_read_top: reading $filename";
    switch -exact -- [file extension $filename] {
        .vhd {
            if {$libname == "default"} {
                P_read_vhdl $filename;
                elaborate $design_name;
            } else {
                P_read_vhdl $filename $libname;
                elaborate -library $libname $design_name;
            }
            remove_license VHDL-Compiler;
        }
        .vhdl {

```

```

        if {$libname == "default"} {
            P_read_vhdl $filename;
            elaborate $design_name;
        } else {
            P_read_vhdl $filename $libname;
            elaborate -library $libname $design_name;
        }
        remove_license VHDL-Compiler;
    }
    .v {
        if {$libname == "default"} {
            P_read_verilog $filename;
            elaborate $design_name;
        } else {
            P_read_verilog $filename $libname;
            elaborate -library $libname $design_name;
        }
        remove_license HDL-Compiler;
    }
    .db {
        echo "#INFO: Reading db $filename";
        read_db $filename;
        elaborate $design_name;
    }
    default {
        echo "#INFO: Reading db $filename";
        read_db $filename;
        elaborate $design_name;
    }
}
}; # end P_read_top;

define_proc_attributes P_read_top \
    -info "TOPS: Procedure to read in the target" \
    -define_args {
        {filename "File Name of the target" filename string required}
        {libname "Library Name for the target" libname string}
    }

#####
# Usage: P_analyze_pkg filename.vhd
# This procedure is used to do analyze only packages for TOPS.
# This is useful for working with VHDL packages.
#####
proc P_analyze_pkg {fn} {
    global G_PROJ_SYN;
    global synthetic_library;
    global link_library;
    global synopsys_program_name;

    # Load the library setup script
    set myfile [file join ${G_PROJ_SYN} proj_Library.tcl]
    P_source_if_exists "$myfile" "SETUP";

    P_get_license "VHDL-Compiler";
    analyze -f vhd1 $fn;
    remove_license "VHDL-Compiler";

    # Done, exit code
    exit;
}; # end P_analyze_pkg;

```

```

define_proc_attributes P_analyze_pkg \
  -info "TOPS: Procedure to analyze VHDL packages" \
  -define_args {
    {filename "vhdl filename" filename string required}
  }

#####
# Usage: P_read_subfubs $SUBFUBS_list
# This procedure is used to deal with sub-units (subfubs) by
# reading them in rather than compiling them and then reading them
# in. This is to promote top-down compiles with sub-units. Used
# in the proj_main.tcl script.
# Project may add extensions as required.
#####
proc P_read_subfubs {subfubs {libname default}} {
  global G_SRC_PATH;

  # Check if any subfubs to read. If none, exit else process list;
  if {[string match $subfubs ""]} {
    set subfub_list [join $subfubs];      # must join it into list first;
    echo "SUBFUBS: $subfub_list";

    # Now parse out each file;
    foreach {subfub} $subfub_list {
      set dirpath [file dirname $subfub];      # get directory path first;

      # No directory, use $G_SRC_PATH
      if {[string match $dirpath ""] == 1} {
        set subfub [file join $G_SRC_PATH $subfub];
      }

      switch -exact -- [file extension $subfub] {
        .vhd {
          if {$libname == "default"} {
            P_read_vhdl $subfub;
          } else {
            P_read_vhdl $subfub $libname;
          }
        }
        .vhdl {
          if {$libname == "default"} {
            P_read_vhdl $subfub;
          } else {
            P_read_vhdl $subfub $libname;
          }
        }
        .v {
          if {$libname == "default"} {
            P_read_verilog $subfub;
          } else {
            P_read_verilog $subfub $libname;
          }
        }
        .db {
          echo "#INFO: Reading db $subfub";
          read_db $subfub;
        }
        default {
          echo "#INFO: Reading db $subfub";
          read_db $subfub;
        }
      }
    }
  }
}

```

```

    }
    }; # end switch
  }; # end foreach
}; # end if
}; # end P_read_subfubs;

define_proc_attributes P_read_subfubs \
  -info "TOPS: Procedure to read the subfubs before compiling" \
  -define_args {
    {subfubs "list of the subfubs" subfubs string required}
    {libname "Library Name for the subfubs" libname string}
  }

#####
# Usage: P_compilelib libname libsrcdir
# This procedure is used to compile a library of [libname] into the ./dbs
# directory from the source pointed to by libsrcdir. Called from
# Makefile directly (dc_shell-tcl -x P_compilelib [libname libsrc])
#####
proc P_compilelib {libname libsrcdir} {

  set mark_main [clock seconds]; # save start time
  P_timestamp;
  if { $libname == {} } {
    echo {ERROR: variable libname is not set}
    # exit 1
  }
  if { $libsrcdir == {} } {
    echo {ERROR: variable libsrcdir is not set}
    # exit 1
  }
  P_get_license "Design-Compiler"

  set myfile [[file join ${libsrcdir} ${libname}.lib];
  echo "#P_compilelib: reading $myfile\n";
  read_lib "$myfile";

  set myfile [[file join dbs ${libname}.db];
  echo "#P_compilelib: writing out $myfile";
  write_lib $libname -f db -o "$myfile"

  set myfile [file join log ${libname}.hst];
  redirect "$myfile" {history -h};

  # done!
  echo "#P_compilelib: Elapsed time of execution was " [P_elapsed_time $mark_main];
  P_timestamp;
}; # end P_compilelib;

define_proc_attributes P_compilelib\
  -info "TOPS: Procedure to compile a library into the ./dbs from the source pointed to
by libsrcdir" \
  -define_args {
    {libname "Name of the library to be compiled" libname string required}
    {libsrcdir "Directory that points to the library" libsrcdir string required}
  }

#####
# Usage: P_log_filter {log_path unitname}
# This procedure is used to filter the synthesis log file for the information
# of interest. Errors, warnings & elapsed times.

```

```
#####
proc P_log_filter {log_path design} {
    set log_file [file join $log_path ${design}.log];
    set filter_log_file [file join $log_path ${design}.log.summary];
    set error_cnt 0; #number of error messages
    set warning_cnt 0; #number of warning messages
    set line_cnt 0; #number of line in log file
    set elapsed_cnt 0; #number of elapsed messages

    set LOG_HANDLE [open $log_file r];
    set FILT_LOG_HANDLE [open $filter_log_file w];

    # Search for lines with errors and warnings
    while {[gets $LOG_HANDLE line] >= 0} {
        incr line_cnt 1;
        if {[regexp {^.*Error} $line]} {
            regsub Error $line "" new_line;
            set errors($error_cnt) "line no ${line_cnt}${new_line}";
            incr error_cnt 1;
        } elseif {[regexp {^Warning} $line]} {
            regsub Warning $line "" new_line;
            set warnings($warning_cnt) "line no ${line_cnt}${new_line}";
            incr warning_cnt 1;
        } elseif {[regexp {Elapsed} $line]} {
            set elapsed($elapsed_cnt) $line;
            incr elapsed_cnt 1;
        }
    }

    # Print Errors and Lines in a filtered log
    puts $FILT_LOG_HANDLE "There are $error_cnt errors, $warning_cnt warnings";

    # Print the Elapsed messages if any
    if {$elapsed_cnt > 0} {
        puts $FILT_LOG_HANDLE "\nElapsed :";
        for {set i 0} {$i < $elapsed_cnt} {incr i} {
            puts $FILT_LOG_HANDLE $elapsed($i)
        }
    }

    # Print the errors if any
    if {$error_cnt > 0} {
        puts $FILT_LOG_HANDLE "\nErrors :";
        for {set i 0} {$i < $error_cnt} {incr i} {
            puts $FILT_LOG_HANDLE $errors($i);
        }
    }

    # Print the warnings if any
    if {$warning_cnt > 0} {
        puts $FILT_LOG_HANDLE "\nWarnings :";
        for {set i 0} {$i < $warning_cnt} {incr i} {
            puts $FILT_LOG_HANDLE $warnings($i);
        }
    }

    close $FILT_LOG_HANDLE;
    close $LOG_HANDLE;
}; # end proc P_log_filter

define_proc_attributes P_log_filter \

```



```

    -info "TOPS: Procedure to filter the logfile and print only the errors, warnings and
elapsed messages" \
    -define_args {
        {log_path "the directory in which the log_file is to be created" log_path string
required}
        {design "Name of the unit" design string required}
    }

```

```

#####
# $Log$
#####

```

```

#Tcl-s
#####
# File Name      : proj_interactive.tcl
# Date Created   : 03/19/00
# Author        : Gregg D. Lahti
# Project       : TOPS Example Code for SNUG
#####
# $Id$
#####
# Description    : Main entry point into the synthesis, reads procedures
#####

```

```

echo "\n#INTERACTIVE: This is the Interactive Project setup
($G_PROJ_SYN/proj_interactive.tcl)";
set sh_source_logging false;

```

```

#####
# Source the Project specific procedures
#####
P_source_if_exists [file join ${G_PROJ_SYN} proj_procs.tcl] "INTERACTIVE";
P_source_if_exists [file join ${G_SCRIPTS_PATH} local_procs.tcl] "INTERACTIVE";
P_source_if_exists [file join ${G_PROJ_SYN} reports_procs.tcl] "INTERACTIVE";
P_source_if_exists [file join ${G_PROJ_SYN} proj_reports.tcl] "INTERACTIVE";

```

```

#####
# Load project library setup script
#####
# Setting default library taken care of in proj_library script
P_source_if_exists [file join ${G_PROJ_SYN} proj_library.tcl] "INTERACTIVE";

```

```

# Return control to USER now

```

```

#####
# $Log$
#####

```

```

#####
# File Name      : syn_Makefile
# Date Created   : 12/01/99
# Author        : Gregg D. Lahti
# Project       : TOPS Example Code for SNUG
#####
# $Id$
#####
# Description:

```

```

#
# The main Makefile slurps the make.rules file in when gmake is called.
# The make.rules uses $(DB) to expand into a ./dbs/ variable which is
# pre-appended to the target being compiled. The ./dbs/[file].db is used as
# a "timestamp" of the VHDL file being compiled (as well as the final object).
#
# User edits:
#
# Each Makefile should have a main target called "all.local" which
# the syn_make.rules file uses as it's target. Set the RECURSEDIRS variable to
# a list of sub-directories that make should traverse first as a dependency.
#
#####
# INCLUDE YOUR RECURSEDIRS LIST HERE!
#####

# RECURSEDIRS: Is a variable that invokes Sub-unit Makefiles for lower-level units.
#
# - Directories are referenced from current directory.
# - This is how to compile lower-level units that this unit depends on
# - Leave empty if no subdirs.
#
# example: RECURSEDIRS = ../../foo/syn ../../bar/syn
#
RECURSEDIRS =

#####
# Include the Synthesis Make Rules (How to build)
# Note: This needs to be defined *after* the RECURSEDIRS variable (gmake weirdness)
#
include $(PROJ_SYN)/syn_make.rules
#####
# Include your dependencies here!
#####

# SUBFUB_unit_name: is a special variable that passes the Unit's Sub-functional
# units or modules to the synthesis scripts.
# - These modules may be vhdl, verilog or db format.
# - The path to the subfubs modules MUST be included.
# - Implements top-down synthesis flow, these modules will be compiled under the Unit
#
# Example: SUBFUB_foo_unit := ../src/bar.vhd ../src/bletch.v ./dbs/comps_kg.db
#
# Create the Unit or top-level make dependencies
# Example: $(DB)foo_unit.db : $(SUBFUB_foo_unit) $(SRC)foo_unit.vhd
#
# Use the mkdepend script to help generate unit dependencies. Paste or
# replace mkdepend results below.
SUBFUB_replace_unit_name :=

$(DB)replace_unit_name.db : $(SUBFUB_replace_unit_name) $(SRC)replace_unit_name.v

# Required target
all.local: $(DB)replace_unit_name.db

#####
# $Log$
#####

```

```
#####
# File Name      : syn_make.rules
# Initial Creation : 5/05/97
# Date for project : 6/16/00
# Author        : Gregg D. Lahti
#####
# $Id$
#####
# Description : This is gmake rules file for synthesis-based Make.
#####
#
# Usage:
#
# gmake          - (calls gmake all by default)
# gmake all      - default, recursive make, doesn't use synbatch
# gmake clean    - recursive clean (removes timestamps)
# gmake cleandist - recursive cleandist (removes timestamps and libs)
# gmake all.local - local make
# gmake clean.local - local clean
# gmake cleandist.local - local cleandist
# gmake J=[N]    - default=1, recursive make with N jobs
#
# Overview:
#
# Each Makefile should have a main target called "all.local" which
# this makefile uses. To execute make locally, use "gmake all.local". This
# will not compile any leaf cells below the hierarchy of this module. For a
# full compile of all sub-unit leaf cells, use "gmake" or "gmake all". You
# can substitute "clean" and "cleandist" for "all" (or "clean.local" and
# (cleandist.local") to get clean and cleandist functions in either of the
# two gmake commands above.
#
# The all, clean, and cleandist use the $(TARGET) variable to recursively
# traverse the $(RECURSEDIRS) directories defined in the user-edited
# Makefile. Make will actually execute "all.local" in each subdir
# (rather than all).
#
# A routine called "check" is done first to verify if the J variable is
# passed on the command line. If it isn't, J gets set to 1 (J=1) so that
# gmake won't execute a boatload of sub-gmakes when J is NULL.
#
#####
# Makefile variables
#####
SHELL      = /bin/sh
SRC        = ../src/
MC_SRC     = ../mc/
RTL        = ../rtl/
DB         = ../dbs/
OUTPUTS    = ../outputs/
LOG        = ../log/
WORK       = ../worklib/
REPORTS    = ../reports/
LAYOUT     = ../layout/
DC         = dc_shell-t
BC         = bc_shell
DV         = design_vision -tcl_mode
PT         = pt_shell
RM         = /bin/rm -rf
COMPILE    = $(PROJ_SYN)/synbatch
LIB        = "DEFAULT"
```

```

HOST                = " "

# don't want recursive variable set recursively
unexport $(RECURSEDIRS)

# export the makefile command-line flags to sub-makes
export $(MAKEFLAGS)
export $(J)

# prevent make from removing db files if make is dorked
# before reports & timing are finished
.PRECIOUS: %.db $(DB)%.db

#####
# Main target rules
#
# There's some history behind why this is setup as it is. The first thing
# I wanted users to do was do a "gmake" with no args and be recursive by
# default. To do this, the .DEFAULT flag is set to all. The "all" rule
# does a "make check" and then a make with the $(TARGET) variable being set
# to all.local (which coincidentally is the main target in the Make.rules
# file). When this happens, the recursive target rule gets used and gmake
# happily starts the recursive make process by executing "gmake recursive
# all.local". The recursive rule is defined farther down the Makefile.
#
# The same recursive procedure applies to clean and cleandist when the user
# issues a "gmake clean" or "gmake cleandist".
#####
# recursive target rule
$(TARGET): $(RECURSEDIRS)

.DEFAULT: all
all:
    @ $(MAKE) check > /dev/null
    @ $(MAKE) TARGET=all.local -j $(J) LIB=$(LIB) MAKEFLAGS=$(MAKEFLAGS)
clean:
    @ $(MAKE) TARGET=clean.local LIB=$(LIB) MAKEFLAGS=$(MAKEFLAGS)
cleandist:
    @ $(MAKE) TARGET=cleandist.local LIB=$(LIB) MAKEFLAGS=$(MAKEFLAGS)

#####
# The following rules utilize a gmake facility to strip out the target name
# from the rest of the path info. We use this to create the correct variable
# name (i.e. SUBFUB_unit) and then read in the contents of the variable to be
# used within the proj_main.tcl script.
# For example, if the variable was set in the gmake file and we were compiling
# the counter_unit.vhd file:
#
#     SUBFUB_counter_unit = foo.vhd bar.vhd
# the rules will strip out the path info from the target ../src/counter_unit.vhd
# to result in a variable called SUBFUB_counter_unit which we then reference
# as a variable ${SUBFUB_counter_unit} to get the sub-units to be read in first.
#####

#####
# VHDL->DB synthesis rule. We snarf the basename without the directory
# prefix and pass this as the design name to the $(COMPILE) shell script.
# Uses files in SRC directory ($SRC is set in variables section).
#####
$(DB)%.db: $(SRC)%.vhd
    @ $(COMPILE) $< SUBFUB="${SUBFUB_$(notdir $(basename $<))}" LIB=$(LIB)
HOST=$(HOST)

```

```
#####
# VHDL->DB synthesis rule. Just like above, except with the wrong-suffix
# name convention (here for legacy usage within Intel)
# Uses files in SRC directory ($SRC is set in variables section).
#####
$(DB)%.db: $(SRC)%.vhd1
    @ $(COMPILE) $< SUBFUB="$${SUBFUB}_$(notdir $(basename $<))}" LIB=$(LIB)
HOST=$(HOST)

#####
# Verilog->DB synthesis rule. We snarf the basename without the directory
# prefix and pass this as the design name to the $(COMPILE) shell script.
# Uses files in SRC directory ($SRC is set in variables section).
#####
$(DB)%.db: $(SRC)%.v
    @ $(COMPILE) $< SUBFUB="$${SUBFUB}_$(notdir $(basename $<))}" LIB=$(LIB)
HOST=$(HOST)

#####
# LIB->DB synthesis rule. Pass in the basename plus src directory path to
# dc_shell-tcl directly with the P_compilelib procedure. This is a workaround
# to get libraries compiled without a lot of extra scripting.
#####
$(DB)%.db: %.lib
    @ gtouch -f log/$(notdir $(basename $<)).log
    @ rm -rf log/$(notdir $(basename $<)).log
    @ $(DC) -x "P_compilelib $(notdir $(basename $<)) $(dir $<):quit;" | tee
log/$(notdir $(basename $<)).log

#####
# VHDL package->analyzed synopsys data rule. Pass in the VHDL package name
# and run the P_analyze_pkg procedure. We used the log file instead of %.syn
# because the WORK directory may not be consistent between projects
#####
$(LOG)%.log::
    @ gtouch -f ./log/$(notdir $(basename $<)).log
    @ rm -rf ./log/$(notdir $(basename $<)).log
    @ echo "\nExecuting with arguments:\nAnalyzing Package: $<\n"
    @ $(DC) -x "P_analyze_pkg $<" > ./log/$(notdir $(basename $<)).log

#####
# Basic targets to just get the tool up interactively.
# Issue them as "gmake dc" or "gmake bc" to get tool command line up with the
# basic setup environment
# Default library_name is set to "", TOPS sets default lib
#####
# Invoke interactive DC shell session
dc :
    @ $(DC) -x "set G_LIBRARY_NAME $(LIB); source $(PROJ_SYN)/proj_interactive.tcl"

# Invoke interactive Design Vision (replaces Design Analyzer) session
dv :
    @ $(DV) -x "set G_LIBRARY_NAME $(LIB); source $(PROJ_SYN)/proj_interactive.tcl"

# Invoke interactive PT shell session
pt :
    @ $(PT) -x "set G_LIBRARY_NAME $(LIB); source $(PROJ_SYN)/proj_interactive.tcl"

#####
# Generic targets of clean and cleandist
```

```
#####
clean.local:
    @ $(RM) $(DB)*.db
    @ $(RM) $(REPORTS)*.*
    @ $(RM) $(LOG)*.*
    @ $(RM) $(WORK)*.*

cleandist.local:
    @ $(RM) $(DB)*.db
    @ $(RM) $(REPORTS)*.*
    @ $(RM) $(LOG)*.*
    @ $(RM) $(WORK)*.*
    @ $(RM) $(OUTPUTS)*.*

#####
# check for J=X prescence (number of parallel jobs), give J a value of 1 if NULL
#####
check:
ifndef J
    J=1
endif

#####
# Recursiveness.
#
# If $(TARGET) is set, we actually do a "gmake recursive [object]" type of
# make to all the sub-directories specified by the $(RECURSEDIRS) variable.
# Since $(TARGET) is set, that gets passed to the sub-make along with
# any make flags $(MAKEFLAGS) (such as -n). The FORCE rule is needed
# as a blank placeholder to do a "cd" into each subdirectory and do
# the actual make command by the "foreach" being done by the $(RECURSEDIRS)
# variable rule. The value of $(RECURSEDIRS) is checked so that make doesn't
# infinitely loop on an NULL value.
#
# this is commented out here (in the Makefile) but kept here for historical
# reasons. Make can't expand the variable correctly when recursing, so this
# needs to be in the Makefile instead.
#####
$(RECURSEDIRS): FORCE
    @ cd $@; $(MAKE) --print-directory $(TARGET) MAKEFLAGS=$(MAKEFLAGS) NB=$(NB)
LIB=$(LIB)

# intentionally left blank!
FORCE:

#####
# Help! (The beatles!) Embedded into makefile for portability issues
#####
help:
    @echo " Make Help - Gregg D. Lahti 06/16/00"
    @echo " "
    @echo " -----"
    @echo " Make targets:"
    @echo " -----"
    @echo " "
    @echo " gmake                - (calls gmake all by default)"
    @echo " gmake help           - gets this help screen"
    @echo " gmake all            - default, recursive make"
    @echo " gmake clean          - recursive clean (removes timestamps)"
    @echo " gmake cleandist      - recursive cleandist (removes timestamp & "
    @echo "                      $MTI_VER/[lib] directories)"

```

```

@echo " gmake all.local          - local make"
@echo " gmake clean.local        - local clean"
@echo " gmake cleandist.local     - local cleandist"
@echo " gmake J=[N]               - default N=1, recursive make with N parallel
jobs"
@echo "                               (example gmake J=3 will get 3 jobs)"
@echo " gmake LIB=[libname]        - makefile using non-default synopsys library"
@echo " "
@echo " For a complete overview on gmake, surf the GNU Make web page:"
@echo " http://www.gnu.org/manual/make-3.77/make.html"
@echo " "
@echo " "
@echo " -----"
@echo " Makefile Overview:"
@echo " -----"
@echo " "
@echo " Each Makefile should have a main target called \"all.local\" which"
@echo " the makefile uses as the \"entry point\" to building. For a full compile"
@echo " of all sub-unit leaf cells (providing the RECURSEDIRS variable is set), "
@echo " use \"gmake\" or \"gmake all\". You can substitute \"clean\" and
\"cleandist\" "
@echo " for \"all\" to remove the compiled libraries, binaries, and logfiles. "
@echo " Cleandist nukes all auto-created items."
@echo " "
@echo " The all, clean, and cleandist use the \044(TARGET) variable to
recursively"
@echo " traverse the \044(RECURSEDIRS) directories defined in the user-edited"
@echo " Makefile. Gmake will actually execute \"all.local\" in each subdir"
@echo " (rather than all).\"
@echo " "
@echo " To compile without recursion into sub-dirs, use \"gmake all.local\".
This"
@echo " will ignore the RECURSEDIRS settings and not compile any leaf cells
specified"
@echo " in hierarchy. It will only compile the current working directory in which
"
@echo " gmake was executed in. You can also issue \"clean.local\" and"
\"cleandist.local\" "
@echo " to get clean and cleandist functions local without sub-dir recursion."
@echo " "
@echo " The Makefile includes the make.rules file in when gmake is called."
@echo " The make.rules file uses \044(DB) to expand into a ./dbs/ variable which
is "
@echo " pre-appended to the target being compiled. The ./dbs/[file].db is used as
"
@echo " a \"timestamp\" of the VHDL file being compiled (as well as the final
object).\"
@echo " The ../src/[file].[v,vhd,lib] is the source of the target ./dbs/[file].db
being"
@echo " compiled by Synopsys."
@echo " "
@echo " When gmake executes the rules in the syn_make.rules file, a routine called
"
@echo " \"check\" is done first to verify if the J variable is passed on the
command"
@echo " line. If it isn't, J gets set to 1 (J=1) so that gmake won't execute a"
@echo " plethora of gmakes when J is NULL."
@echo " "
@echo " "
@echo " -----"
@echo " Makefile User edits:"

```

```

@echo " -----"
@echo " "
@echo " Each Makefile should have a main target called \"all.local\" which"
@echo " the main make.rules uses as it's target. Set the RECURSEDIRS variable to"
@echo " a list of sub-directories that make should traverse first as a
dependency."
@echo " Do not edit the include line, and be sure to keep the RECURSEDIRS
variable"
@echo " set *ABOVE* the include of the syn_make.rules line. Failure to do so
will"
@echo " result in gmake unpredictability in execution. Finally, add in the
appropriate"
@echo " dependencies for all.local and any other file/compile dependencies you"
@echo " wish to make. "
@echo " "
@echo " "
@echo " -----"
@echo " Basic Make syntax:"
@echo " -----"
@echo " "
@echo " Putting a '#' on the first line denotes a commented line which gmake
ignores."
@echo " All rules use a \"target: src\" identifier, where the first item is the"
@echo " \"target\" in which gmake will build and the second is the source files in
"
@echo " which gmake will use. You may also assign variables using the \"var :=
[list]\" "
@echo " syntax. The variable referenced within gmake should be written as
\044(var)."
@echo " "
@echo " Important note! For every dependency or list assigned, the next line "
@echo " must begin with a tab character! If the rule isn't finished (ie you've"
@echo " got more than 1 line of rules to describe), you must put a \ (backslash)"
@echo " to show that the rule is continued to the next line. "
@echo " "
@echo " Here's a few examples:"
@echo " "
@echo " "
@echo " -----"
@echo " Makefile Example 1 (bottom-up compile approach):"
@echo " -----"
@echo " "
@echo "     RECURSEDIRS = "
@echo " "
@echo "     FOFILES := \044(DB)bar.db \\"
@echo "                \044(DB)hack.db "
@echo " "
@echo "     \044(DB)foo.db: \044(FOFILES)
@echo " "
@echo "     \044(DB)top.db: \044(DB)foo.db \\"
@echo "                   \044(DB)bletch.db \\"
@echo "                   \044(DB)spew.db"
@echo " "
@echo "     all.local: \044(DB)top.db"
@echo " "
@echo " "
@echo " In this example, there are no subdirectories to recurse through (hence
the"
@echo " RECURSEDIRS is empty). The entry point (or starting point) for gmake is"
@echo " all.local. We assign a rule to all.local, which happens to be a target"
@echo " called \044(DB)top.db. Remember that the \044(DB) gets expanded to the"

```



```

        @echo " ./db directory (set in the syn_make.rules file). The top.db file has"
        @echo " dependencies on the sub-units foo.db, bletch.db, and spew.db. The foo.db"
        @echo " file has a dependency on a variable called \044(FOOFILES). The
\044(FOOFILES)"
        @echo " variable is set to two other files, bar.db and hack.db."
        @echo " "
        @echo " Don't forget that the 2 lines after the \"\044(DB)top.db : \044(DB)foo.db
\" line"
        @echo " must have a tab as the beginning character in the line!"
        @echo " "
        @echo " In effect, the bottom-up compile strategy defined above will result in
the"
        @echo " source files compiled in this order (assuming that the source files are
verilog"
        @echo " files): "
        @echo " "
        @echo "         bar.v "
        @echo "         hack.v "
        @echo "         foo.v "
        @echo "         bletch.v "
        @echo "         spew.v "
        @echo "         top.v "
        @echo " "
        @echo " Remember that the defined list is the targets. The gmake rules
understands"
        @echo " how to compile the source to .db and has specific rules for verilog and
VHDL"
        @echo " source files."
        @echo " "
        @echo " "
        @echo " -----"
        @echo " Makefile Example 2 (top-down compile approach:"
        @echo " -----"
        @echo " "
        @echo "     RECURSEDIRS = ../../newunit/syn"
        @echo " "
        @echo "     SUBFUB_counter_unit := \044(DB)seq.vhd \044(SRC)gray.vhd
\044(DB)john.v\"
        @echo " "
        @echo "     \044(DB)counter_unit.db: \044(SRC)seq.vhd \"
        @echo "                             \044(SRC)gray.vhd \"
        @echo "                             \044(DB)john.v"
        @echo " "
        @echo "     all.local: \044(DB)counter_unit.db"
        @echo " "
        @echo " "
        @echo " In this example, there is one subdirectory to recurse through. The entry
point"
        @echo " for gmake is all.local, and it points to the counter_unit.db file. There
is"
        @echo " a dependency list for the counter_unit.db: the source files, or subfubs
of"
        @echo " the hierarchy. To get a top-down compile, a special variable prefix
called"
        @echo " SUBFUB_ is used to prefix the design name (in this case counter_unit) as
the"
        @echo " variable name. The variable is set to the list of files that must be
analyzed,"
        @echo " rather than compiled, when the counter_unit is read in. This enables a"
        @echo " top-down compile method. In this example, the subufubs seq.vhd, gray.vhd,
and"

```

```

        @echo " john.v are analyzed (not compiled) before the counter_unit.vhd is read in"
        @echo " and the compile strategy then executes as a full compile of all units."
        @echo " "
        @echo " Note that there is still a dependency for the sub-units of the
counter_unit.db"
        @echo " which points to the source files. This dependency enforces a recompile
of"
        @echo " counter_unit if any of the subfubs have changed. Don't forget that the 2
lines"
        @echo " after \"\\044(DB)counter_unit.db : \\044(DB)seq.vhd \" line must have a tab
as the"
        @echo " beginning character in the line!"
        @echo " "
        @echo " "
        @echo " -----"
        @echo " Helpful debugging tips:"
        @echo " -----"
        @echo " "
        @echo " The -n flag will "run" gmake but not actually execute gmake. Gmake will"
        @echo " show the specific commands that will be executed as if it was really going
to"
        @echo " run."
        @echo " "
        @echo " The -d flag is a debugging flag. It's verbose, and runs through the
entire"
        @echo " rules list (internally to gmake first) and then through the Makefile
being"
        @echo " used."
        @echo " "

#####
# $Log$
#####

```