Compilation and Reporting Example Scripts

Compilation with a Non-Project Flow

The following is an example Tcl script that defines a Non-Project design flow.

The example script uses a custom command `reportCriticalPaths`. This is an illustration on how the Vivado Design Suite can be augmented with custom commands and procedures. The content of `reportCriticalPaths` is provided and explained in the section Defining Tcl Procedures.

```
# STEP#1: define the output directory area.
#
set outputDir ./Tutorial_Created_Data/cpu_output
file mkdir $outputDir
#
# STEP#2: setup design sources and constraints
#
read_vhdl -library bftLib [ glob ./Sources/hdl/bftLib/*.vhdl ]
read_vhdl ./Sources/hdl/bft.vhdl
read_verilog  [ glob ./Sources/hdl/*.v ]
read_verilog  [ glob ./Sources/hdl/mgt/*.v ]
read_verilog  [ glob ./Sources/hdl/or1200/*.v ]
read_verilog  [ glob ./Sources/hdl/usbf/*.v ]
read_verilog  [ glob ./Sources/hdl/wb_conmax/*.v ]
read_xdc ./Sources/top_full.xdc
#
# STEP#3: run synthesis, write design checkpoint, report timing,
# and utilization estimates
#
synth_design -top top -part xc7k70tfbg676-2
write_checkpoint -force $outputDir/post_synth.dcp
report_timing_summary -file $outputDir/post_synth_timing_summary.rpt
report_utilization -file $outputDir/post_synth_util.rpt
#
# Run custom script to report critical timing paths
reportCriticalPaths $outputDir/post_synth_critpath_report.csv
#
# STEP#4: run logic optimization, placement and physical logic optimization,
# write design checkpoint, report utilization and timing estimates
#
opt_design
reportCriticalPaths $outputDir/post_opt_critpath_report.csv
place_design
report_clock_utilization -file $outputDir/clock_util.rpt
#
# Optionally run optimization if there are timing violations after placement
if {[get_property SLACK [get_timing_paths -max_paths 1 -nworst 1 -setup]] < 0} {
    puts "Found setup timing violations => running physical optimization"
    phys_opt_design
}
write_checkpoint -force $outputDir/post_place.dcp
report_utilization -file $outputDir/post_place_util.rpt
report_timing_summary -file $outputDir/post_place_timing_summary.rpt
```

***see running synthesis with tcl PDF for more info on synth_design command**

```
#
# STEP#5: run the router, write the post-route design checkpoint, report the routing
# status, report timing, power, and DRC, and finally save the Verilog netlist.
#
route_design
write_checkpoint -force $outputDir/post_route.dcp
report_route_status -file $outputDir/post_route_status.rpt
report_timing_summary -file $outputDir/post_route_timing_summary.rpt
report_power -file $outputDir/post_route_power.rpt
report_drc -file $outputDir/post_imp_drc.rpt
write_verilog -force $outputDir/cpu_impl_netlist.v -mode timesim -sdf_anno true
#
# STEP#6: generate a bitstream
#
write_bitstream -force $outputDir/cpu.bit
```

Details of the Sample Script

The key steps of the preceding script can be broken down as follows:

- **Step 1:** Defines a variable, `$outputDir`, that points to an output directory and also physically creates the directory. The `$outputDir` variable is referenced as needed at other points in the script.
- **Step 2:** Reads the VHDL and Verilog files that contain the design description, and the XDC file that contains the physical and/or timing constraints for the design. You can also read synthesized netlists (EDIF or NGC) using the `read_edif` command.

  The Vivado Design Suite uses design constraints to define requirements for both the physical and timing characteristics of the design. The `read_xdc` command reads an XDC constraints file which will be used during synthesis and implementation.

IMPORTANT: *The Vivado Design Suite does not support the UCF format. For information on migrating UCF constraints to XDC commands refer to the ISE to Vivado Design Suite Migration Guide (UG911) [Ref 4] for more information.*

  The `read_*` Tcl commands are designed for use in Non-Project mode, as it allows a file on the disk to be read by the Vivado Design Suite to build an in-memory design database, without copying the file or creating a dependency on the file in any way, as it would in Project mode. All actions taken in the Non-Project mode are directed at the in-memory database within the Vivado tools. The advantages of this approach make the Non-Project mode extremely flexible with regard to the design. However, a limitation of the Non-Project mode is that you must monitor any changes to the source design files, and update the design as needed. For more information on running the Vivado Design Suite using either Project mode or Non-Project mode, refer to the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) [Ref 5].

- **Step 3:** Synthesizes the design on the specified target device.

This step compiles the HDL design files, applies the timing constraints located in the XDC file, and maps the logic onto Xilinx primitives to create a design database in memory. The in-memory design resides in the Vivado tools, whether running in batch mode, Tcl shell mode for interactive Tcl commands, or in the Vivado Integrated Design Environment (IDE) for interaction with the design data in a graphical form.

Once synthesis completes, a checkpoint is saved for reference. At this point the design is an unplaced synthesized netlist with timing and physical constraints. Various reports like timing and utilization can provide a useful resource to better understand the challenges of the design.

This sample script uses a custom command, `reportCriticalPaths`, to report the TNS/WNS/Violators into a CSV file. This provides the ability for you to quickly identify which paths are critical.

Any additional XDC file read in after synthesis by the `read_xdc` or `source` commands is used during the implementation steps only. They will be stored in any subsequent design checkpoints, along with the netlist.

- **Step 4:** Performs pre-placement logic optimization, in preparation for placement and routing. The objective of optimization is to simplify the logic design before committing to physical resources on the target part. Optimization is followed by timing-driven placement with the Vivado placer.

  After each of those steps, the `reportCriticalPaths` command is run to generate a new CSV file. Having multiple CSV files from different stages of the design lets you create a custom timing summary spreadsheet that can help visualizing how timing improves during each implementation step.

  Once the placement is done, the script uses the `get_timing_paths` command to examine the SLACK property of the worst timing path in the placed design. While the `report_timing` command returns a detailed text report of the timing path with the worst slack, the `get_timing_paths` command returns the same timing path as a Tcl object with properties that correspond to the main timing characteristics of the path. The SLACK property returns the slack of the specified timing path, or worst path in this case. If the slack is negative then the script runs physical optimization to resolve the placement timing violations whenever possible.

  At the end of Step 4, another checkpoint is saved and the device utilization is reported along with a timing summary of the design. This will let you compare pre-routed and post-routed timing to assess the impact that routing has on the design timing.

- **Step 5:** The Vivado router performs timing-driven routing, and a checkpoint is saved for reference. Now that the in-memory design is routed, additional reports provide critical information regarding power consumption, design rule violations, and final timing. You can output reports to files, for later review, or you can direct the reports to the Vivado IDE for more interactive examination. A Verilog netlist is exported, for use in timing simulation.

- **Step 6:** Writes a bitstream to test and program the design onto the Xilinx FPGA.

Compilation with a Project Flow

The following script illustrates a Project flow that synthesizes the design and performs a complete implementation, including bitstream generation. It is based on the CPU example design provided in the Vivado installation tree.

```
#
# STEP#1: define the output directory area.
#
set outputDir ./Tutorial_Created_Data/cpu_project
file mkdir $outputDir
create_project project_cpu_project ./Tutorial_Created_Data/cpu_project \
    -part xc7k70tfbg676-2 -force
#
# STEP#2: setup design sources and constraints
#
add_files -fileset sim_1 ./Sources/hdl/cpu_tb.v
add_files [ glob ./Sources/hdl/bftLib/*.vhdl ]
add_files ./Sources/hdl/bft.vhdl
add_files [ glob ./Sources/hdl/*.v ]
add_files [ glob ./Sources/hdl/mgt/*.v ]
add_files [ glob ./Sources/hdl/or1200/*.v ]
add_files [ glob ./Sources/hdl/usbf/*.v ]
add_files [ glob ./Sources/hdl/wb_conmax/*.v ]
add_files -fileset constrs_1 ./Sources/top_full.xdc
set_property library bftLib [ get_files [ glob ./Sources/hdl/bftLib/*.vhdl ]]
#
# Physically import the files under project_cpu.srcs/sources_1/imports directory
import_files -force -norecurse
#
# Physically import bft_full.xdc under project_cpu.srcs/constrs_1/imports directory
import_files -fileset constrs_1 -force -norecurse ./Sources/top_full.xdc
# Update compile order for the fileset 'sources_1'
set_property top top [current_fileset]
update_compile_order -fileset sources_1
update_compile_order -fileset sim_1
#
# STEP#3: run synthesis and the default utilization report.
#
launch_runs synth_1
wait_on_run synth_1
#
# STEP#4: run logic optimization, placement, physical logic optimization, route and
#         bitstream generation. Generates design checkpoints, utilization and timing
#         reports, plus custom reports.
set_property STEPS.PHYS_OPT_DESIGN.IS_ENABLED true [get_runs impl_1]
set_property STEPS.OPT_DESIGN.TCL.PRE [pwd]/pre_opt_design.tcl [get_runs impl_1]
set_property STEPS.OPT_DESIGN.TCL.POST [pwd]/post_opt_design.tcl [get_runs impl_1]
set_property STEPS.PLACE_DESIGN.TCL.POST [pwd]/post_place_design.tcl [get_runs impl_1]
set_property STEPS.PHYS_OPT_DESIGN.TCL.POST [pwd]/post_phys_opt_design.tcl [get_runs impl_1]
set_property STEPS.ROUTE_DESIGN.TCL.POST [pwd]/post_route_design.tcl [get_runs impl_1]
launch_runs impl_1 -to_step write_bitstream
wait_on_run impl_1
puts "Implementation done!"
```