run.tcl

```
################################################################################
## Template Script for RTL -> Gate-Level Synthesis
## Tool     : Cadence Genus
## Design   : SHA256
## Target   : 750 MHz (1.333 ns)
################################################################################


################################################################################
## Print system information (useful for debugging & runtime comparison)
################################################################################
if {[file exists /proc/cpuinfo]} {
  sh grep "model name" /proc/cpuinfo   ;# Print CPU model
  sh grep "cpu MHz"    /proc/cpuinfo   ;# Print CPU frequency
}

puts "Hostname : [info hostname]"       ;# Print machine hostname


################################################################################
## Global Variables
################################################################################
set DESIGN sha256                        ;# Top module name
set GEN_EFF medium                      ;# Generic synthesis effort
set MAP_OPT_EFF high                    ;# Mapping & optimization effort

# Create time-stamped directories to avoid overwriting old runs
set DATE [clock format [clock seconds] -format "%b%d-%T"]
set _OUTPUTS_PATH outputs_${DATE}
set _REPORTS_PATH reports_${DATE}
set _LOG_PATH logs_${DATE}


################################################################################
## Search Paths & Tool Configuration
################################################################################

# Library search path (.lib)
set_db / .init_lib_search_path {/home/TMSY/genus_work/sha256_fast_250MHz/lib_search_path}

# Script search path
set_db / .script_search_path {/home/TMSY/genus_work/sha256_fast_250MHz/script_search_path}

# RTL search path
set_db / .init_hdl_search_path {/home/TMSY/verilog}

# Limit number of CPUs used by Genus
set_db / .max_cpus_per_server 8

# Increase verbosity for better debug visibility
set_db / .information_level 7
```

```
################################################################################
## Library & Physical Setup
################################################################################

# Read timing library (FAST corner)
read_libs "fast.lib"

# Read LEF for physical cell information
read_physical -lef "gsclib045.fixed2.lef"

# RC estimation using capacitance table (pre-route)
set_db / .cap_table_file {/home/TMSY/logical_synthesis_sha256_fast/lib_search_path/captbl/best/capTable}


################################################################################
## Power Optimization Settings
################################################################################

# Allow Genus to insert integrated clock gating cells
set_db / .lp_insert_clock_gating true


################################################################################
## Read RTL & Elaborate Design
################################################################################

# Read all RTL source files
read_hdl "$DESIGN.v sha256_core.v sha256_k_constants.v sha256_stream.v sha256_w_mem.v"

# Elaborate the top design
elaborate $DESIGN

puts "Runtime & Memory after 'read_hdl'"

# Check for unresolved references or missing modules
check_design -unresolved


################################################################################
## Timing Constraints (SDC)
################################################################################

# Read timing constraints file
read_sdc "/home/TMSY/genus_work/sha256_fast_1.333ns_750MHz/sha256_fast.sdc"

# Verify clocks, IO delays, and timing intent
check_timing_intent


################################################################################
## Create Output Directories
```

```
################################################################################
if {![file exists ${_LOG_PATH}]} {
  file mkdir ${_LOG_PATH}
}

if {![file exists ${_OUTPUTS_PATH}]} {
  file mkdir ${_OUTPUTS_PATH}
}

if {![file exists ${_REPORTS_PATH}]} {
  file mkdir ${_REPORTS_PATH}
}


################################################################################
## Cost Group Definition (Timing Classification)
################################################################################

# Remove any existing cost groups
delete_obj [vfind /designs/* -cost_group *]

# Define cost groups only if registers exist
if {[llength [all_registers]] > 0} {

  define_cost_group -name I2C -design $DESIGN   ;# Input -> Register
  define_cost_group -name C2O -design $DESIGN   ;# Register -> Output
  define_cost_group -name C2C -design $DESIGN   ;# Register -> Register

  path_group -from [all_registers] -to [all_registers] -group C2C -name C2C
  path_group -from [all_registers] -to [all_outputs]   -group C2O -name C2O
  path_group -from [all_inputs]    -to [all_registers] -group I2C -name I2C
}

# Input -> Output paths
define_cost_group -name I2O -design $DESIGN
path_group -from [all_inputs] -to [all_outputs] -group I2O -name I2O

# Pre-synthesis timing reports
foreach cg [vfind / -cost_group *] {
  report_timing -group [list $cg] >> $_REPORTS_PATH/${DESIGN}_pretim.rpt
}


################################################################################
## Generic Synthesis (RTL → Generic Logic)
################################################################################

set_db / .syn_generic_effort $GEN_EFF

# Perform generic synthesis
syn_generic

puts "Runtime & Memory after 'syn_generic'"
time_info GENERIC
```

```

# Datapath inference report (adders, muxes, shifters)
report_dp > $_REPORTS_PATH/generic/${DESIGN}_datapath.rpt

# Save design snapshot
write_snapshot -outdir $_REPORTS_PATH -tag generic

# QoR summary
report_summary -directory $_REPORTS_PATH


################################################################################
## Technology Mapping (Generic → Standard Cells)
################################################################################

set_db / .syn_map_effort $MAP_OPT_EFF

# Map design to standard cells
syn_map

puts "Runtime & Memory after 'syn_map'"

# Save mapped snapshot
write_snapshot -outdir $_REPORTS_PATH -tag map

# QoR summary
report_summary -directory $_REPORTS_PATH

# Datapath report after mapping
report_dp > $_REPORTS_PATH/map/${DESIGN}_datapath.rpt

# Post-map timing per cost group
foreach cg [vfind / -cost_group *] {
  report_timing -group [list $cg] > $_REPORTS_PATH/${DESIGN}_[vbasename $cg]_post_map.rpt
}

# Generate RTL → mapped LEC script
write_do_lec \
  -revised_design fv_map \
  -logfile ${_LOG_PATH}/rtl2intermediate.lec.log \
  > ${_OUTPUTS_PATH}/rtl2intermediate.lec.do


################################################################################
## Incremental Optimization
################################################################################

# Remove continuous assigns
set_db / .remove_assigns true

# Replace constants using unique tie-hi / tie-lo cells
set_db / .use_tiehilo_for_const unique

# Incremental optimization (faster & safer)
```

```
syn_opt -incremental

# Save optimized snapshot
write_snapshot -outdir $_REPORTS_PATH -tag syn_opt_incr

# QoR summary
report_summary -directory $_REPORTS_PATH

puts "Runtime & Memory after 'syn_opt'"
time_info OPT

# Post-opt timing
foreach cg [vfind / -cost_group *] {
  report_timing -group [list $cg] > $_REPORTS_PATH/${DESIGN}_[vbasename $cg]_post_opt.rpt
}


################################################################################
## Final Reports & Netlist Generation
################################################################################

# Final datapath report
report_dp > $_REPORTS_PATH/${DESIGN}_datapath_incr.rpt

# Tool messages
report_messages > $_REPORTS_PATH/${DESIGN}_messages.rpt

# Final snapshot
write_snapshot -outdir $_REPORTS_PATH -tag final
report_summary -directory $_REPORTS_PATH

# Write gate-level netlist
write_hdl > ${_OUTPUTS_PATH}/${DESIGN}_m.v

# Save Genus script
write_script > ${_OUTPUTS_PATH}/${DESIGN}_m.script

# Write post-synthesis SDC
write_sdc > ${_OUTPUTS_PATH}/${DESIGN}_m.sdc


################################################################################
## Logical Equivalence Checking (LEC)
################################################################################

# Mapped → Final
write_do_lec \
  -golden_design fv_map \
  -revised_design ${_OUTPUTS_PATH}/${DESIGN}_m.v \
  -logfile ${_LOG_PATH}/intermediate2final.lec.log \
  > ${_OUTPUTS_PATH}/intermediate2final.lec.do

# RTL → Final
write_do_lec \
  -revised_design ${_OUTPUTS_PATH}/${DESIGN}_m.v \
  -logfile ${_LOG_PATH}/rtl2final.lec.log \
  > ${_OUTPUTS_PATH}/rtl2final.lec.do


################################################################################
## Finalization
################################################################################

puts "============================"
puts "Synthesis Finished ........."
puts "============================"

# Archive stdout log
file copy [get_db / .stdout_log] ${_LOG_PATH}/.

# Save Genus database
write_db -to_file synthesized.db

## quit   ;# Uncomment if running in batch mode

```