cadence™

RTLDesignFlow UserGuide

Release Version: RDF4.1

21st June 2017

Change History

Document Version	Date	Author	Remarks
0.1	9th August 2016	Mark Lewis	Initial version copied from http://sp/sites/WFO/ww_services/wwverificationservices/ip_factory/Shared%20Documents/Flow%20related%20docs/RTLDesignFlow%20UserGuides/RTLDesignFlow_UserGuide_r4.0.2_v1.docx?Web=1
0.2	13th April 2017	Mark Lewis	Added additional Spyglass guidance and updated AFL section.
0.3	20th April 2017	Vladimir Zivkovic	Tools and sections on synthesis, LEC, ATPG and simulation updated
0.4	25th April 2017	Vladimir Zivkovic	Minor updates with regard to setup and consistency
0.5	25th April 2017	Anna Gilbert	Minor updates with regard to formatting
0.6	21st June 2017	Patrick McKeever	Added section to support CFI transfer to RDF Updated synthesis section for CUI and genus Updated Innovus for CUI Amended to reference MMMC file
0.7	22nd of June 2017	Vladimir Zivkovic	Final minor updates

Table of Contents

Tables

Figures

Acronyms and Definitions

Acronym	Meaning
ATPG	Automatic Test Pattern Generation
CCD	Conformal Constraint Designer
HAL	HDL Lint Analysis
LEC	Logical Equivalence Checking
RAK	Rapid Adoption Kit
GA	General Availability
RDF	RTLDesignFlow
CDC	Clock Domain Crossing

1. RTLDesignFlow Overview

The RTL Design Flow (RDF) is a set of generic scripts that can be used during product development and sign off. These have been developed to primarily define checks that must be run during the RTL sign-off process. However, the tools used at sign-off are also useful during development and their use is encouraged early in the design flow. Reasons to use the RTLDesignFlow scripts:

- support development of new digital IP
- Catch RTL problems earlier in the design cycle
- assess the quality of an IP product
- form part of the sign-off process of a product prior to its general availability (GA)
- Obtain PPA metrics for Datasheet population or pre-sales information

As much as possible, the scripts have been written to minimize the amount of customization that needs to be performed to make them work with a target design and its associated environment.

Figure 2 Development/Sign-off Flow provides a high level overview of a standard RTL development flow ranging from basic lint checking to ATPG gate level simulations.

Table 1 Tools, purpose and recommendations provides a list of the various tools supported by the RDF and defines them as mandatory, optional or recommended for use on new (Greenfield) and legacy IP and whether they should be used during development and/or Sign-off.

Table 2 Tool versions, training and help details provides the tool versions that the scripts have been qualified with. Other tool versions may also work but the defined versions in this table are what the RDF was signed off with prior to release. This table all lists some sources of training if you are unfamiliar with a particular tool and expert aliases for various tools where support can be found.

Section 1.9, Support channels, provides further details on how to get support for the RDF scripts, report bugs in tools/RDF and some other places to obtain training on tools.

Section 1.6, Recommended Directory Structure, provides advice on how to set up a directory that will contain all files relevant to the RDF. It is strongly recommended that this structure be followed as it will provide customers with the same type of setup across IPs that they receive.

An RTLDesignFlow checklist exists that was previously used to sign off the IP but nowadays it is used for reference only to highlight the reports and logs that should be checked. This can be found here: example checklist. These checks have been implemented in a utility called Stork which will parse the required logfiles and provide a report on the status of the various checks with in the RDF. Stork should be run to verify all logs and reports on a regular basis and is mandatory for sign-off: RTLDesignFlow Checks in Stork

Any part of the flow that is not run and is deemed Mandatory must be signed off by IPG exec staff.

1.1. RDF setup

Figure 1 RDF Setup provides a flow of tasks for setting up and running the RDF.

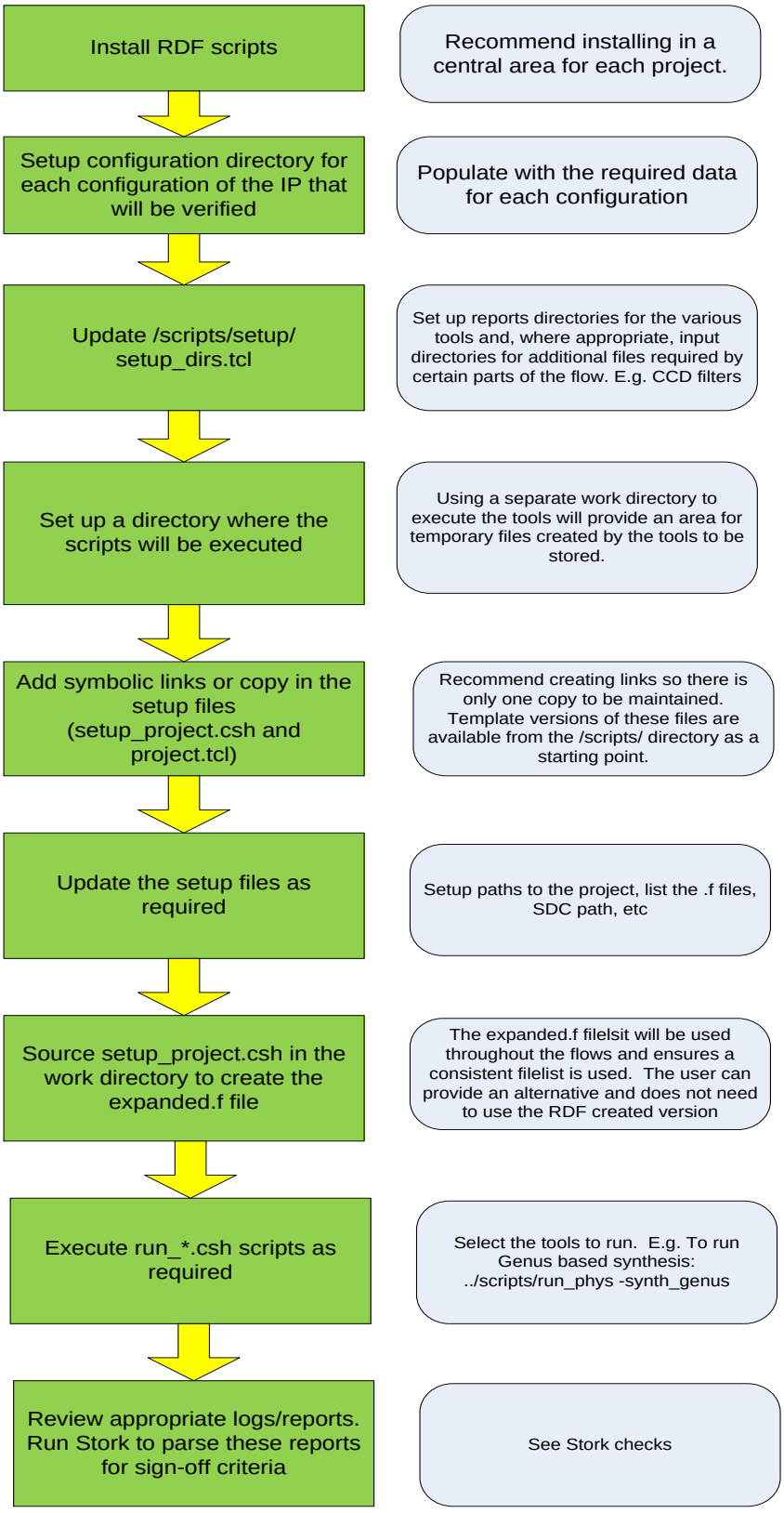


Figure 1 RDF Setup

1.2. High level overview of execution steps

Below is a list of recommended sequential steps when running the RDF. The user can refine as necessary.

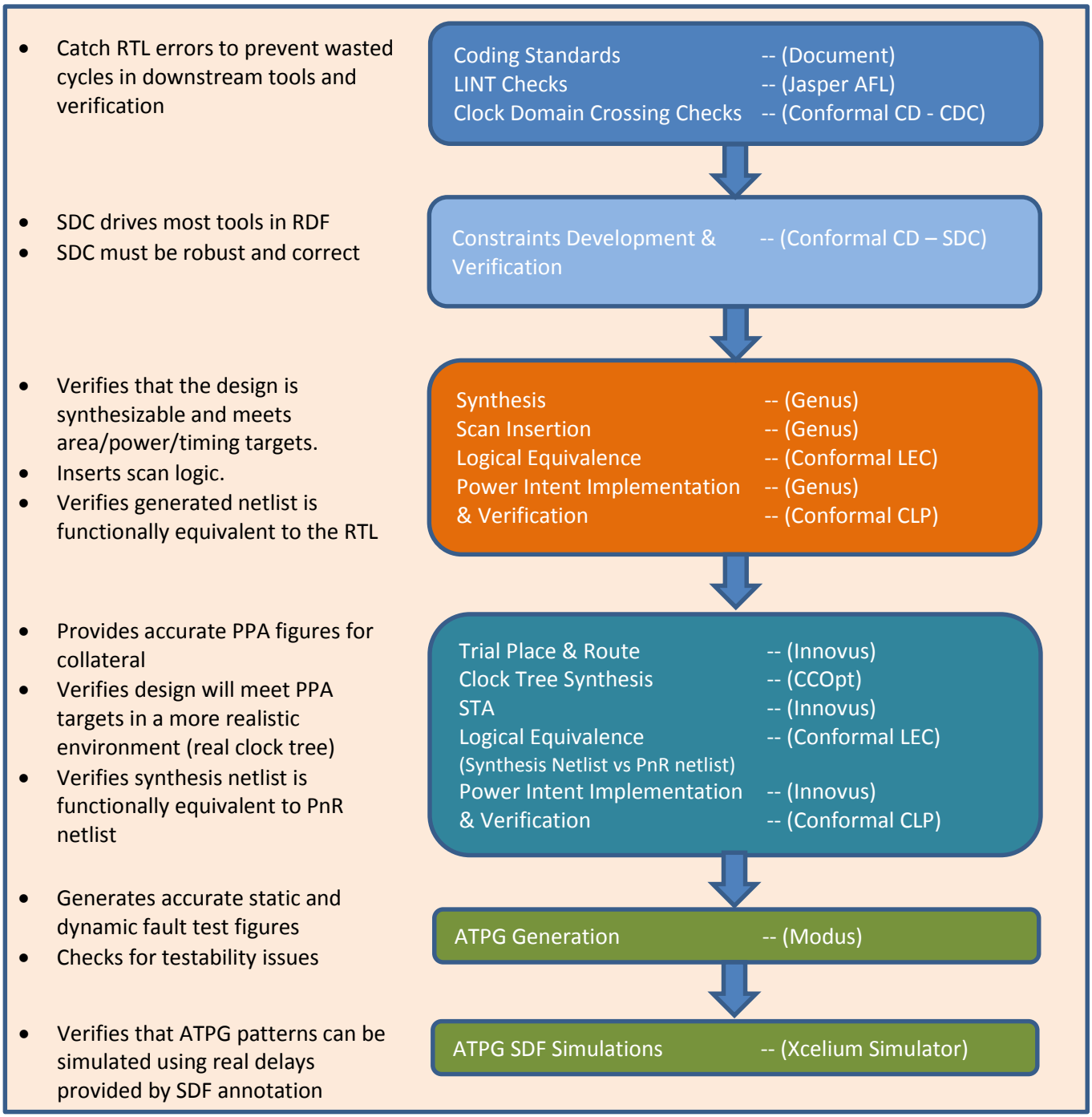


Figure 2 Development/Sign-off Flow (without Low Power Checks)

1.3. Defined Flow stages/tools

The table below defines the various tools that are supported by the RDF and provides guidance as to what tools should be run and during development and sign-off.

Tool	Sub Tool	Purpose	Legacy IP		Greenfie		Notes
			IPG recommendation		IPG recommendation		
			Development Stages	Sign off	Development Stages	Sign off	
Conformal	CCD Lint	RTL Lint checks	Optional	Optional	Optional	Optional	Useful Lint tool
	CCD Structural CDC	Checks for correct synchronisation logic	Recommended	Mandatory	Recommended	Mandatory	
	CDC Convergence	Post synchronizer re-convergence	Optional	Optional	Mandatory	Mandatory	Offset by randomized synchronizer block (if used)
	CCD SDC Checks	Lint and validation	Mandatory	Mandatory	Mandatory	Mandatory	
	Conformal LEC	Logical Equivalence Checks	Mandatory	Mandatory	Mandatory	Mandatory	
	Conformal Low Power	power intent verification	Mandatory	Mandatory	Mandatory	Mandatory	If CPF/UPF flow
JasperGold	AFL	Basic Lint (HAL) Advanced Lint (SPS)	Mandatory	Mandatory	Mandatory	Mandatory	Currently only basic lint is mandatory
Xcelium/ Incisive	N/A	ATPG gate level simulations	Optional	Optional	Optional	Optional	Mandatory for hard IP sign off
	Compilation Check	Verify single and multi step compilation	Optional	Mandatory	Optional	Mandatory	Not mandatory for hard IPs
Genus	N/A	Synthesis	Mandatory	Mandatory	Mandatory	Mandatory	
	N/A	Power Analysis (RTL based)	Recommended	Mandatory	Recommended	Mandatory	Activity files recommended (Joules may replace)
	N/A	Power Analysis (post Layout)	Recommended	Mandatory	Recommended	Mandatory	Activity files from GLS recommended (Joules may replace)
Modus	N/A	ATPG	Recommended	Mandatory	Mandatory	Mandatory	
Innovus	N/A	Placement, Clock tree optimisation	Recommended	Mandatory	Mandatory	Mandatory	Majority of the design avail. Soft IP only!
Joules	N/A	RTL Power Analysis	Recommended	Recommended	Recommended	Recommended	
Stork	N/A	Final quality sign-off	N/A	Mandatory	N/A	Mandatory	
Palladium	N/A	Palladium emulation compilation					Not implemented yet
Xilinx	Vivado	FPGA synthesis checks					Not implemented yet

Spyglass	N/A	3rd party sign off	N/A	Mandatory	N/A	Mandatory	

Definitions

Optional	Optional if an acceptable alternative is provided (At EPM discretion)
Recommended	Running these tools can provide valuable information to influence the design during development and/or highlight potential issues at signoff
Mandatory	Must be run
Development stages	Anytime during development for Cadence tools and at defined milestones within project for Spyglass (primarily to reduce license crunch)
Sign-off stage	Preparation for GA and/or customer release
Greenfield	New development from scratch

Table 1 Tools, purpose and recommendations

1.4. Tool versions

The following table provides a list of tool versions that were used to sign off this version of the RDF scripts. It also provides links to appropriate Rapid Adoption Kits (RAK) which are very useful learning packages if the user is unfamiliar with a particular tool.

Always install tools in the order Xcelium/Incisive first followed by Genus, then Modus or you will see a conflict. This is required as Modus comes with its own versions of Genus, NCVLOG and NCELAB (but not NCSIM).

Tool	Version	Purpose	RAK (if available)	Expert Tool Alias Email
Cadence Genus	genus/162/16.21	Synthesis	No RAK for common UI	ask_genus_expert
Cadence Conformal (LEC)	confrml/162/16.20.100 or later	Formal Equivalence Checking	Conformal LEC: LEC Jumpstart Kit	ask_conformal_expert
Cadence Innovus	innovus/162/16.21	Place and Route, Clock concurrent optimization (CCopt)	CCOpt RAK for Beginners with Innovus 16.20	ask_spc_expert, ask_ccopt_expert
Conformal Low Power	confrml/162/16.20.100 or later	Power Format Checking	Conformal Low Power (CLP) RAK for Beginners	ask_clp_expert
Joules	Jls/1510/15.10-s004_1	RTL Power Analysis	Joules: Joules RAK for Beginners	N/A
Modus	modus/162/16.20.000 For chambers with OpenLava use: modus/162/16.20_1660448EHF	ATPG	Modus-Test-ATPG-and-Analysis	ask_test_expert
Atrenta SPYGLASS	atrenta/spyglass/5.5.1 atrenta/ipkit/atrenta_5.5.1_v1 atrenta/ipkit/tsmc_3.0	lint/dft/cdc/sdc	N/A	http://www.atrenta.com/support_login.php
Cadence Conformal (CCD)	confrml/162/16.20.100 or later	lint/dft/cdc/sdc	Conformal Constraint Designer: SDC Constraint and CDC Verification Methodologies	ask_ccd_expert
Microsoft Excel	N/A	RTLDesignFlow Checklist (reference only)	N/A	N/A
Cadence Xcelium	xlm/201611/16.11.001	ATPG simulaitons	N/A	N/A
Jasper AFL	jasper/1703/17.03.000/1 incisive/152/15.20.024 (Earlier versions may not work)	LINT (Primary)	No RAK https://wiki.cadence.com/confluence/display/FAV/Automatic+Formal+Linting	fav_support

Table 2 Tool versions, training and help details

1.5. RDF Scripts Package

1.5.1. RDF Version

The current official release version of the Design Flow scripts is "Phase 4.1". This can be obtained by referencing the SVN tag **ipf_scripts_phase4.1**

Development of the scripts is ongoing, both in relation to bug fixes and new functionality. On the understanding that users want and need access to the latest script versions, the development team will periodically release interim TAGs, to reflect the most recent SVN revision number that is considered to provide a stable and usable set of Design Flow scripts. Users encountering issues should feedback via the JIRA system as described below, quoting the relevant TAG.

1.5.1.1. History of Released Versions

Tag Name	Release Date	Notes
ipf_scripts_phase3.2	6th February 2015	Main base version
ipf_scripts_phase3.2.1	25th February 2015	Bug fix release
ipf_scripts_phase4.0	6th November 2015	Major release
ipf_scripts_phase4.0.1	10th December 2015	Bug fix release
ipf_scripts_phase4.0.2	16th March 2016	Bug fix release
ipf_scripts_phase4.1	23rd June 2017	Major release

Table 3 History of Released Versions

1.5.2. Scripts Installation

To check out the Design Flow scripts using SVN externals:

NOTE you need a minimum of SVN 1.6 to check out the externals, the default on a lot of systems seems to be 1.5 so please check!

Using SVN externals provides a hard link between the user's project area and a specific version of the scripts. This has benefits in terms of revision control and traceability, but prevents any local customization of the scripts.

1. cd into the directory where you want to create the external, i.e. where you want the scripts directory to appear.
2. Edit the externals:
   a. *svn pe svn:externals* .
   b. In the editor window add:
      http://lvsvn:20000/svn/ip_factory/tags/ipf_scripts_phase4.1
      Save the file and close the editor.
3. Update to pick-up the external:
   svn update
4. Commit the change with an appropriate message:
   svn commit –m "Added external for RTLDesignFlow, pointing to tag ipf_scripts_phase4.1"

To checkout a copy of the Design Flow scripts using SVN:

If this option is followed it is the user's responsibility to ensure that the SVN revision number, and any local modifications that have been made to the scripts, are recorded in the relevant checklists during the IP approval process. Any modifications considered project-independent should be fed back to the development team for consideration, via the JIRA system as described below.

1. cd into the directory where you want the RTLDesignFlow directory to appear.
2. Check out the required version of the scripts:
    a. For the official tagged release:

svn export http://lvsvn:20000/svn/ip_factory/tags/ipf_scripts_phase4.1

1.6. Recommended Directory Structure (CFG)

Files used in the flow which are design dependent are now separated from the scripts directory. For example, files that should not have to be changed will remain in the delivered scripts directory and files that are design or configuration dependent will stay within the project specific configuration (cfg) directory as shown in Figure 3. An example of the contents of the cfg directory is contained in /scripts/example_cfg/ in the release tarball. The configuration directory should be set using the "CFG_DIR" variable in /setup/setup_dirs.tcl

The main benefit of this approach is that the design dependent data exists alongside the design and can be readily used with future versions of the scripts instead of the user having to populate the scripts directory with these files.

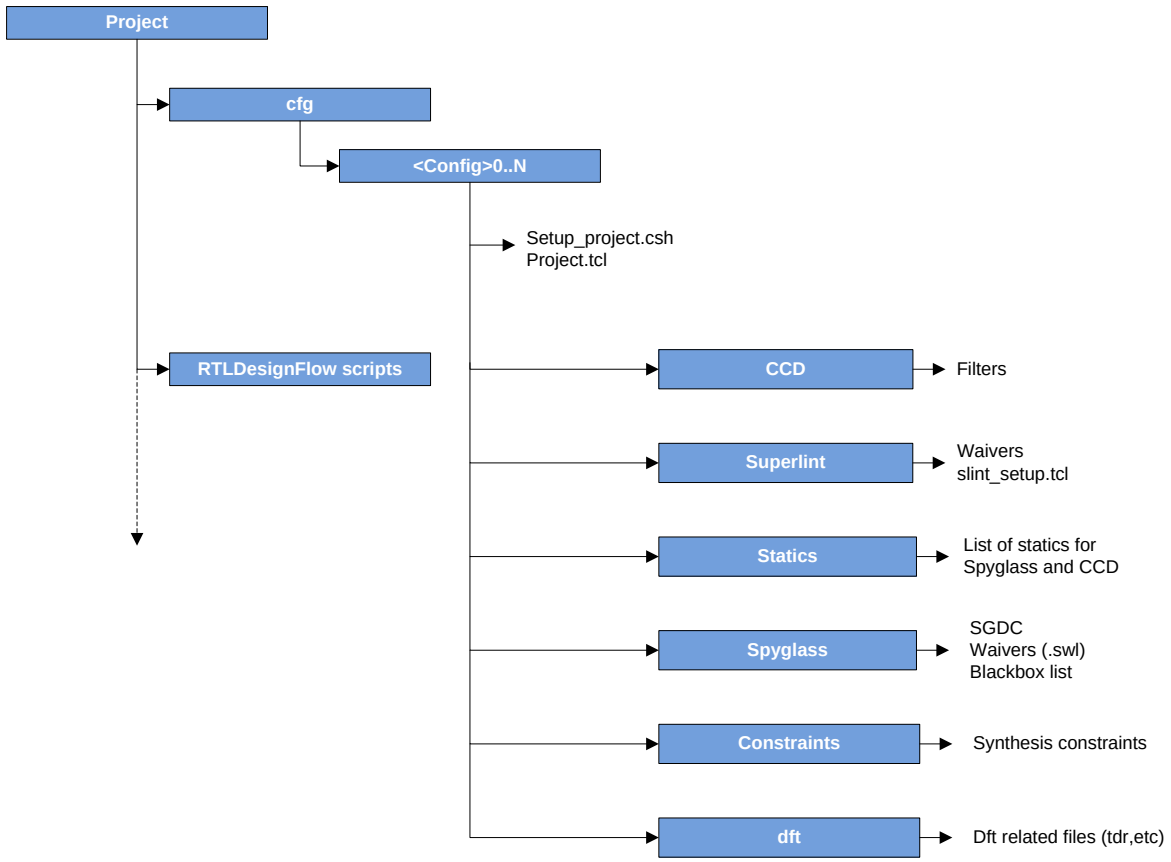


Figure 3 Recommended directory structure

1.7. Required inputs to RTLDesignFlow

The following items are required prior to executing the RTLDesignFlow scripts.

- Implementation Spec + User Guide.
- RTL
  - Including a .f file listing
- SDC constraints.
  - The primary SDC file should observe the naming convention ${SDC_PATH}/${DESIGN}_${CONFIG}.func.sdc, where func is the primary mode of the design.
  - SDC files representing other modes should observe the naming convention ${SDC_PATH}/${DESIGN}_${CONFIG}.${mode}.sdc

1.7.1. Setup files

The checkout will create a directory named "./ipf_scripts_phase4.1/scripts" containing all the scripts, including a project_template.tcl file, setup_project_template.csh and some example run_*.csh runscripts.

There are two main setup files:

- project_template.tcl
- setup_project_template.csh

These should be copied to the configuration directory (cfg/<config>) directory and renamed project.tcl / setup_project.csh. The setup files must be edited to set up variables appropriate for the target IP. Symbolic links can be made to these files from the directory where the RTLDesignFlow scripts will be executed. For example, from a temporary work directory:

ln –s ../../cfg/single_pixel/setup_project.csh

ln –s ../../cfg/single_pixel/project.tcl

The following project environment variables must be setup prior to running the Design Flow in setup_project.csh. These can be defined as part of the project environment, or within the run script that calls the tool scripts. Other variables need to be defined in project.tcl. See this file for descriptions.

Variable	Purpose
TECHNOLOGY	This variable is used by the tech_lib_setup.tcl script to determine the target technology to be used during synthesis.
DESIGN	This variable is used to identify the top level module name of the target design.
DESIGN_MODES	This variable is used to define the name of the design modes targeted during the design flow.
CONFIG (optional)	This variable is used to differentiate between different configurations of the same root design. (No longer defaults to generic – users can set this to "generic" to match previous versions of the flow) See Notes below.
LOG_PATH	This variable is used by the setup/setup_dirs.tcl script as the base directory for all output reports.
DUT_PATH	This variable is used to identify the root directory of the target design and/or configuration.

RTL_PATH	This variable is used to identify the root directory of the design RTL
CFG_DIR	Path to the configuration data directories where input files relevant to the flows should be placed.
TDR_PATH	Location of a local TDR file if required
RTL_F_FILELIST	A list of one or more .f files.
KEEP_PATHS	If set then the collated expanded.f file will retain the same format as in the provided .f file. (full pathnames won't be inserted)
HDL_SEARCH_PATH	Include directories should be added to the .f file. However this variable can has been retained for backwards compatibility with older scripts.
POWER_INTENT_FILE	Defines the CPF/1801 power intent file (optional)
SDC_PATH	This variable is used to define the path to the SDC file
RTL_TOGGLE_FILE_DIR	This variable is used to identify the root directory to all power state tcf directories from RTL simulations
NL_TOGGLE_FILE_DIR	This variable is used to identify the root directory to all power state tcf directories from GLS simulations
STAMP (optional)	This variable is used to assign a unique run ID to the path names of the logs and reports generated by the scripts. Defaults to "default".
IPF_DESIGN_FLOW_SCRIPTS	This variable is used to identify the root directory of the checked out Design Flow scripts. For example, it should be set to the path of the "ipf_scripts_phase4.1/scripts" directory.
ATPG_OTHER_MODULES	This variable is used to identify interface files required for ATPG
DFT_ABSTRACT_MODEL	This variable is used to describe pre-existing scan chains inside hard IP in the form of a scan_abstract file
DEFINEMACRO	This variable is used to set up any -defines required
BLACKBOXES	A list of modules to be blackboxed in CCD, AFL and Spyglass. NOTE: For Power and DFT analysis makes sure BLACK_BOX is set to '1 in project.tcl
MAX_CPUS_PER_SERVER	Limit the number of CPU's used by multithreading
MTHREAD_QUEUE	Define the queue for tool multithreading (NB not an interactive queue)
LSF	This variable is used to define local LSF commands
DELIVERY_TAG	This variable is used to define the delivery tag used for customer delivery prior to running directory checking scripts
STATICS_FILE	Contents of this file are used during clock domain analysis by Spyglass and CCD. Statics can be defined as: 1. Registers which are used for configuration where the main datapath is not enabled for a period after these registers are written. 2. Ports which are tied to a value prior to a datapath being enabled or a reset being applied to the design.
SPYGLASS_PACK_LOCATION	Directory where the Spyglass IPkit will be moved to after tar and zip (pack command)
PMA_NETLIST_RELEASE_PATH	New variable to facilitate the ATPG and ATPG simulation flow for hard IPs. This variable should also be set for the final ATPG simulation to point to the delivery area where the data need to be taken from.

Table 4 Environment variables to modify

1.7.2. Configurations

Multiple configurations can be handled by using the “CONFIG” variable in setup_project.csh. This can then be used to point to .f filelists, sdc constraints, etc. that are relevant to that configuration.

The CONFIG_FILE has been removed and associated defines should be included with the .f filelist. This will include any defines files that are `included in the Verilog files.

For example, for a design called mipi_csi2tx that has 2 modes or operation 1lane and 2lane:

Constraints files: mipi_csi2tx_1lane.func.sdc, mipi_csi2tx_2lane.func.sdc

Filelists: mipi_csi2tx_1lane.f, mipi_csi2tx_2lane.f

1.7.3. User Modifiable Files

Below is a list of all the files contained in the RTL Design flow that can be modified by the user:

scripts/setup_project_template.csh	copy to **setup_project.csh** and modify to set up environment variables
scripts/project_template.tcl	copy to **project.tcl** and modify to set up variables
scripts/tech_lib_setup.tcl	depending on library setup
scripts/ccd/cdc.do	In general this file should not require editing but if adding sync modules, skip instances etc. then edits can be made.
scripts/cpf/project.cpf	power intent file
scripts/cpf/tech.cpf	low power cell definitions
scripts/nc/annotate_sdf.tcl	for hard IPs with multiple sdfs
scripts/modus/write_vectors_template.txt	to define timing relations (padding) between scan signals

The following are provided in the release tarball only as example files that should be used in the appropriate configuration directory.

scripts/ccd/filters*	to setup various individual filters/waivers for CCD.
scripts/et/dummy.tdr	where necessary to extend parameters such as number of I/O, maximum clock frequency etc.
sdc/*	these are template constraints files only and need to be used as guidelines when developing project constraints.

1.7.4. File lists (.f listing)

To guarantee that every tool in the flow can accept the provided file listing a single filelist with the following characteristics is recommended:

- File lists should be in the Verilog command file format used by most tools. Genus will not read this directly but genus_flow.tcl will parse the +incdir+ and +defines+ from the .f file.
- The provided .f filelist should contain information related to that particular configuration. Previous versions of the scripts allowed the use of the CONFIG_FILE to provide this type of information but this has been deprecated.
- The contents of the .f filelist should be restricted to the following:
  - +incdir+<xxx>
  - +define+<xxx>
  - Filename
    - All paths must be relative to the run directory or absolute.

There are two scripts delivered that can be utilized to create a single .f file that can be used throughout the entire RDF:

- /scripts/setup/filelist_setup.csh was first delivered in RDF4.0. It can accommodate multiple filelists and expand them into a single .f file. It has some limitations in that it doesn't not expand incdir paths and does not handle nested .f files. This script will generate a file called rdf_4.0.1_expanded.f which may be used if the above limitations are not a problem.

- A second script, /scripts/setup/expand.pl now generates the default expanded.f file and has been proven on a handful of testcases. It handles a limited number of nested .f file and will expand the incdir lines to overcome the limitations of filelist_setup.csh. expand.pl is called from within filelist_setup.csh so by default both rdf_4.0.1_expanded.f and expanded.f will be generated.

  The user can select one of the generated .f files or use their own by setting the variable "RTL_F_FILE" in setup_project.csh.

  If the .f files presented only require to be collated the user can set "KEEP_PATHS 1" variable in setup_project.csh. The default is 0 which will replace contents with full paths.

RECOMMENDATION: source setup_project.csh and review the generated .f prior to running any tools. Some modification may be necessary.

1.7.5. Library Setup

To check out an example version of the tech_lib_setup.tcl in the current working directory:

svn export
http://lvsvn:20000/svn/ip_factory/tags/ipf_scripts_phase<phase>/library_setup/tech_lib_setup.tcl

This file will be different across sites depending on library installation directories

The physical flow will use the TCL variables in tech_lib_setup.tcl directly.

Spyglass, CCD and AFL will use a library list which is generated by parsing tech_lib_setup.tcl. This will allow the user to review the library information that is being read into the tools.

NOTE: The library is used by default in CCD and AFL but they can be omitted by using the appropriate switch (see help with the appropriate csh script).

Spyglass and CCD will use the generated file ./liblist.f and Jasper AFL will use ./liblist.v. These files will be generated if they do not exist when running run_ccd.csh, run_spyglass.csh or run_afl.csh.

1.7.6. Integrating RDF and CFI setup

The RDF depends on the tech_lib_setup for pointers to libraries, and technology specific variables. This same information is defined by the CFI flow in various files and directly referencing this information would ensure synthesis/CCD etc. would be run with the exact same library files and setup as CFI.

If the CFI MMMC file is CUI compliant it can be swapped with the MMMC file supplied with RDF scripts, **scripts/genus/mmmc.tcl.**

Remaining tech_lib_setup variables and CFI equivalent

RDF variable defined in tech_lib_setup	**Equivalent in CFI (file and variable)**
LIBLEF	BLK.globals -> $lef_files
PROCNODE	EDIFLOW.project.parameters-> designProcess
max_route_layer	EDIFLOW.project.parameters-> nanoroute_TopRoutingLayer
TIELIST	EDIFLOW.project.parameters-> addTie_tieLo_cellName, addTie_tieHi_cellName
DON'T_USE	CFI projects apply this in their SDC, if left as an empty variable RDF flow will run.
CLK_BUFFERS	EDIFLOW.project.parameters-> CCOPT_buffer_cells
CLK_INVERTERS	EDIFLOW.project.parameters-> CCOPT_inverter_cells

1.8. Pre-Requisites to QA Audit

Prior to submitting the design for audit the following must be completed.

- Completion of Stork run.
- The relevant reports & logs output from each tool
- Any waiver/filter documentation associated with warnings, errors, etc. for each tool.

1.9. Support channels

The following support channels are available:

1. For RDF issues please file a JIRA at http://jira.cadence.com:8080/browse/SISFLOW
   File the following types of issues with the above mechanism:
   a. flow issues
   b. enhancement requests
   c. Documentation (Wiki or UserGuide).
   d. Feedback relating to the Design Flow scripts and their usage.

2. Review the UserGuide (found in the <install>/docs directory).

3. Rapid Adoptions Kits are available for most tools to provide accelerated ramp on a tool. If you require training then inform your manager and/or look at http://lms/ for online or classroom based training.

4. Send an email to the appropriate expert alias.
   a. See available aliases at http://emailweb/cgi-bin/index.cgi .
   b. There is a dedicated expert email alias for any questions relating to the RTLDesignFlow ask_RTLDesignFlow_expert : ask_RTLDesignFlow_expert@cadence.com

      The idea, like other expert alias groups, is that this is a user forum for people to ask questions and share their knowledge. Anyone who subscribes can both ask and answer questions. The emails will also be searchable as they are archived online, so can be used as a first port of call in the case you have questions on the RTLDesignFlow.

5. For tools based issues please add a CCR against the appropriate tool: http://ccms/cqweb/

2. Constraints Methodology

2.1. Overview

Template SDC files can be found here <IPF_DESIGN_FLOW_SCRIPTS>/sdc
These can be used as a starting point to create new SDC constraints files. It is imperative that SDC files are verified prior to being used to drive the other tools within the flow. This will prevent wasted cycles in downstream tools.

Conformal Constraint Designer and Spyglass are used to perform SDC lint checks and verification. SDC checks are on by default in the CCD scripts but can be disabled in project.tcl.

For Spyglass related SDC goals see Atrenta Spyglass - Atrenta/TSMC Kits

Atrenta Spyglass - Atrenta/TSMC Kits

See section 0 for using Conformal for SDC checks

SDC constraints are used throughout the flow and a basic understanding of Static Timing Analysis as well as how to write SDC constraints is mandatory. Some online training can be found here: online SDC training

The PPA reporting methodology can be found here: Reporting Power, Performance and Area numbers

3. Synthesis

The RDF synthesis script reports the timing, power and area values of the RTL and corresponding SDC in the target library. Timing is optimised in the worst-case corner, while power is optimised at typical corner but reported across best case, typical and worst case corners.

It is essential to run this step to confirm the RTL and corresponding SDC will meet PPA targets in the target technology. If PPA targets cannot be met here, then they are unlikely to be met in the PD cycle.

This stage also reports any mismatches between RTL and SDC, and mismatches between RTL and any hard macros in the design.

3.1. Overview

Genus will be used to perform synthesis and scan insertion. From this release onwards, the synthesis scripts will be run in Genus native mode (CUI). Legacy mode is supported but the usage is discouraged.

3.2. Requirements

Genus requires a valid SDC constraint file to provide the tool with the correct clocks and their relationship with all inputs/outputs of the design.

- The primary SDC file should observe the naming convention ${SDC_PATH}/${DESIGN}_$CONFIG.func.sdc, where “func” is the primary mode of the design.
- SDC files representing other modes should observe the naming convention ${SDC_PATH}/${DESIGN}_$CONFIG.${mode}.sdc

For Low Power Designs a valid and verified CPF is required.

- The power intent file is defined in the setup_project.csh file. It should be verified pre/post synthesis using CLP.

Scan ports should **NOT** exist in the RTL with the exception of embedded hard macros with pre-existing scan chains. In this case, it is required to pull up the hard macro scan pins to the top-level RTL.

3.3. Flow

The synthesis flow should be run using Genus the synthesis tool from Cadence. If the user needs to use RC (previous Cadence synthesis tool) for legacy reasons then run genus in legacy mode.

Here are the various stages executed by genus_synth.tcl to perform RTL synthesis:

1. Update project.tcl with required locations and design information. This file is read by genus_ synth.tcl to set various project-specific TCL variables, which are then used by the tool and other point scripts.
2. Setup stage
    1. Define global attributes.
    2. If threading is selected in project.tcl, then threading servers will be defined here
        1. Note use of MAX_CPU.. env variable here
        2. **It is strongly recommended to use threading to improve runtime**
    3. Read library data
        1. Read .lib library files as determined by value of TECHNOLOGY variable.

        2. Create typical 85C corner for power analysis if it exists
        3. Optimise with WC timing and 25C TYP power
    4. Select VT library – this is HVT only by default
    5. Define "don't use" cells – these are set in the tech_lib_setup as default.
    6. Configure for PLE mode, ensures accurate correlation between Genus and EDI results.
3. Read in the RTL
4. If power intent file exists then read power intent file (if LP design)
5. Elaborate the design. Script supports extraction of source file list and parameters from .f file.
    1. Apply power intent (if LP design)
6. Preserve any pre-defined instances and pre-mapped netlists
7. Source dft_setup.tcl
    1. Define test signals (mode signals, shift enables, resets).
    2. Define test clocks.
    3. Define test signal for clock gating.
    4. Check that all flops pass DFT rule checks.
    5. Define clock constraint files
8. Recommit power intent and verify the power structure (if LP design)
9. Set up for power analysis, either using default probabilities and toggle rates, or data from a toggle file generated by simulation.
10. Synthesize to generic.
    1. Initial power report created
11. Synthesize to mapped.
    1. Mapped DB saved
12. Preserve any submodules or nets
13. Source dft_insert_scan.tcl
    1. Ensure mapped to scan flops.
    2. Preserve pre-existing shift registers.
    3. Define scan chains.
    4. Connect scan chains.
    5. Write out CTLs, scan abstracts and pin assignment files for Encounter Test.
14. Recommit power intent to capture design changes affecting low power (if LP design)
15. Incremental optimization, write resulting netlist.
16. Write reports.
17. Review reports.
    1. A summary of the reports is generated in html format and can be found in the run reports directory: /${DESIGN}_qor.html. This provides a clickable link to a lot of the report files and will help when filling in the check list.

3.3.1. Scripts

A set of synthesis scripts have been developed to help get started with synthesis and are located within the top level of the delivered scripts tarball.

- run_phys.csh --> Set up IP Factory environment variables, and run the Genus tool sourcing the genus/genus_synth.tcl script.
- project_template.tcl --> a template version of the project.tcl file, to be copied to project.tcl in the user's work directory and modified as appropriate to the target design.
- genus/genus_synth.tcl --> the main command script that is read into Genus.
- genus/dft_setup.tcl --> Sourced by genus_synth.tcl, configures Genus for full scan insertion.
- genus/dft_insert_scan.tcl -> Sourced by genus_synth.tcl, inserts scan chains into synthesized netlist and write DfT abstracts.
- cpf/project.cpf
- cpf/tech.cpf -> created by run_clp.csh from the technology library

Switches on run_phys.csh	Function
-h/help	Displays information on available switches
-synth_genus	Run synthesis stage using Genus

-synth_genus_legacy	Runs synth step using Genus in legacy mode
-synth_rc	Run synthesis stage using RC
-lec_1stage*	Run RTL v Final netlist checks
-lec_2stage	Run RTLvsMapped netlist and MappedvsFinal netlist checks
-lec_syn2pnr	Run Final synthesis netlist vs CCopt netlist from PnR
-pnr	Run Innovus flow
DEFAULT	Just synthesis is run using Genus

***** *Not supported anymore in Genus native mode*

Table 5 run_phys.csh switches

3.3.2. Known Issues

KP1

An attribute has been used in Genus during synthesis/dft insertion specifying the triggering edge of the first element of the actual scan chains to build in order to keep our scan chain interfaces uniform (best practice). If the triggering edge of the first element of an actual scan chain does not match the value specified for this attribute, the tool inserts a lockup flop at the beginning of the actual scan chain. This lockup flop will be driven by the same clock that drives the first element in the chain and will be triggered by the clock edge specified with this attribute. This causes TSV-095 Warnings in Modus (log_verify_test_structures) since the first 2 elements in these chains will always have the same state after the first +ve clock edge. These warnings should be reviewed and if related to this specific example, can be ignored.

KP2

The following errors may be seen in the genus.log file. During Beta testing this did not appear to cause issues with synthesis or LEC runs.

This appears to be a tool fix and should be resolved in the version 17:10 due for release at the end of June 2017. Associated fix in Conformal Conformal 16.20-d249. Rev 1.176.

Error : Found existing test signal with same name. [DFT-121] [define_dft lec_override_in]

: signal 'wdl_cons_0' already exists.

: You cannot redefine an existing test signal. If you need to redefine it, use the 'rm' command to first remove the existing test signal from the 'dft/test_signals' directory. Then redefine the test signal.

Error : Found existing test signal with same name. [DFT-121] [define_dft lec_override_in]

: signal 'wdl_cons_1' already exists.

3.3.3. Known Limitations

Currently the scripts do not have any built-in support for templating. The current usage model is that a project.tcl file will exist in the user's current working directory

The flow does not support embedded TAP controllers from a DfT perspective. For IP purposes, it is assumed that the TAP will be at SoC level and is therefore not a concern.

3.3.3.1. DfT Limitations

- For Hard Macros, the name of the .ctl or scan abstract files must conform to <module name>.<ctl/scan_abstract>
- In case of hard macros with analog/digital hierarchical crossings, the generated CTL or scan abstracts might not be plugged and played. The reason is that the analog macros are in .lib format and the Genus is not able to trace it through. The only workaround is to alter the scan abstracts module name to conform the analog macro where it has been contained and to check that the pins at the analog boundary are the same as in buried block.
- For Hard Macro Scan I/O names, the names must be the same with the only change being in and out e.g. if the scan input is pma_scan_in, the scan output must be pma_scan_out. This is due to the way the scripts derive the scan output names from the given scan input names

3.3.4. Notes

On initial runs, for DfT purposes, the variable in the project.tcl DEBUG_PHASE should be set to '1' this enables automatic identification of test clocks and signals. The results of this should be **CAREFULLY** reviewed and during the FINAL design phase, the test clocks and signals should be specified by the user in the relevant variables.

A new variable called ATPG_UNTESTABLE_INSTS has been introduced to facilitate memory wrapper handling. Genus will automatically add bypass logic around these instances

3.3.5. Checking results

The following reports will be generated and should be reviewed. See /scripts/setup/setup_dirs for report locations. Stork will also perform this analysis.

Check	Notes	Report
Verification		
Did the SDC's read in to the design without any errors/warnings?		${_REPORTS_PATH}/genus.log
Are there any errrors in the Logfile?		${_REPORTS_PATH}/genus.log
Are all existing blackboxes expected?		${_REPORTS_PATH}/<design_name>_check_design.unresolved.rpt
Are there any undriven signals ?		${_REPORTS_PATH}/<design_name>_check_design.undriven.rpt
Are there any multidriven signals ?		${_REPORTS_PATH}/<design_name>_check_design.multidriven.rpt
Clock Gating Status, ensure Cannot map to requested logic is 0%		${_REPORTS_PATH}/genus.log
Is >85% of the total flops clock gated?		$_REPORTS_PATH/${DESIGN}_clock_gating.rpt
Are all flip flops clocked? e.g ensure there are no Sequential clock pins without a clock waveform.		$_REPORTS_PATH/genus.log
Is there any unconnected logic?		$_REPORTS_PATH/$DESIGN_check_timing.rpt
Are there any generated clocks with multi-master clock?		${_REPORTS_PATH}/<design_name>_check_timing.rpt

Are there any paths constrained with different clocks?		$_REPORTS_PATH/$DESIGN_check_timing.rpt
Nets with multiple drivers?		$_REPORTS_PATH/$DESIGN_check_timing.rpt"
Pins/ports with conflicting case constants?		$_REPORTS_PATH/$DESIGN_check_timing.rpt
Are IO's constrained? (Inputs/Outputs without clocked external delays)		$_REPORTS_PATH/$DESIGN_check_timing.rpt"
What were the operating conditions used?		$_REPORTS_PATH/$DESIGN_qor.rpt
Is the area mode 'physical library' and interconnect mode 'global', i.e. PLE?	PLE is default in the scripts	$_REPORTS_PATH/$DESIGN_qor.rpt
Has the voltage threshold (SVT/HVT/LVT) % of library cells been reviewed and accepted?	There may be a specific reason to be HTV only - check with tech lead	$_REPORTS_PATH/$DESIGN_gates.rpt
Does DfT Pass Checks?		“$_REPORTS_PATH/$DESIGN_check_dft_rules.rpt” and “$_REPORTS_PATH/$DESIGN_check_dft_rules.dftInsertScan.rpt”
Are any flip flops marked don't scan?		“$_REPORTS_PATH/$DESIGN.dft_dontScanRegs”
How Manys Scanchains are created?		“$_REPORTS_PATH/$DESIGN.dft_chains”
Does the RTL vs CPF, CLP check pass?	This check compares the power format file against the RTL. All errors must be fixed	$_CLP_RPT_PATH/clp_presynth.rpt
Performance		
What is the max frequency of the design?		"$SDC_PATH/${DESIGN}_$CONFIG.$mode.sdc or $_REPORTS_PATH/report_clocks.${DESIGN}.rpt"
Are there any violating paths?	"If there are violating paths then design does meet max frequency, either adjust rtl or reduce sdc frequency"	$_REPORTS_PATH/$DESIGN_qor.rpt"
Power		
"Was a switching file (tcf/vcd) used? (If not results will be pessimistic)"	"A tcf file from rtl sims is recommended to accurately report power for a given mode"	$_REPORTS_PATH/genus.log/$DUT_PATH/cfg/${CONFIG}/project.tcl
What is the RTL Power Reported?		
Leakage	Use TYP corner power	" ${_REPORTS_PATH}/${DESIGN}_power.html or ${_REPORTS_PATH}/${DESIGN}_stochastic.${mode}.power.rpt"
Dynamic	Use TYP corner power	" ${_REPORTS_PATH}/${DESIGN}_power.html or

		${_REPORTS_PATH}/${DESIGN}_stochastic.${mode}.power.rpt"
Total	Use TYP corner power	" ${_REPORTS_PATH}/${DESIGN}_power.html or ${_REPORTS_PATH}/${DESIGN}_stochastic.${mode}.power.rpt"
Area		
What is the defined cell area?	"This area is at 100% utilisation Do not count the wire/net area "	$_REPORTS_PATH/${DESIGN}_area.rpt

Table 6 Synthesis reports

3.4. Interesting Reading

Trial Power Reporting - mPCIe

Overview of reporting power, performance and area can be found here: Appendix B:

3.5. Switches set in project.tcl used in synthesis

Switches	Explanation
LATCHES	Allow latches, default = 0
THREADING	Use multiple threading, default = 0
SCAN_SEQ_MODE	Allow mapping to scan flops when not inserting DFT, default = false
SET_RESET_LATCHES	Allow set/reset latches, default = 0
UNIQUIFY_MODULES	Uniquify modules to prevent module name clash when pulling multiple instantiated modules at top level, default = 1
PRESERVE_HIERARCHY	Prevent hierarchy flattening, default = 0
CLOCK_GATING	Insert clock gating, default = 1
PRINT_FLOPS, PRINT_IO_CLOCKS, PRINT_SYNC_FLOPS	Scripts to assist debug of design, default = 0
TYP_85C_CORNER	Use a ‘HOT’ typical corner for power estimation, default = 0
RC_USE_CONFORMAL	Use conformal during synthesis for LP verification, default = 0
CREATE_TCASE	Create a db test case for debugging, default = 0
REMOVE_BACKSLASH	Prevent ‘/’ in instance naming, default = 0
POWER_UNIT	Default mW
ATPG_UNTESTABLE_INSTS	Specify instances for which the ATPG tools cannot generate test patterns. Typically used for memory blocks.
DEBUG_PHASE	Set DEBUG_PHASE to 1 to enable automatic identification of test clocks and signals. The results of this should be reviewed and during the FINAL design phase, the test clocks and signals should be specified by the user in the below variables

INSERT_SCAN	Enable insertion of scan chains during synthesis
CONNECT_CHAINS	Enable stitching of scan chains during synthesis
RC_CLK_GATING	Enable control of pre-existing clock gating
HARD_MACRO	Identify any hard macro instance inside the IP
BLACK_BOX	Identify presence of analog instances or black boxes

Table 7 Synthesis switches

4. Conformal Low Power

4.1. Overview

Typically for low power design a file is created which specifies the power domains, shut off conditions and associated low power rules. This file will be in either CPF or 1801 format, both of which are supported by the flow. CLP will be used to verify this power file against the RTL, post synth netlist and post placed netlist.

It's important to run all 3 states to cover the varying low power requirements of each. For example synthesis will insert isolation cells but not power switches, which are inserted by Innovus. Any errors must be addressed.

CLP can also be used to create a tech.cpf from a library, see create_tech.cpf.

A Rapid Adoption Kit (RAK) is available for anyone who is not familiar with the tool see: Table 2 Tool versions, training and help details

4.2. Requirements

CLP requires the following files...

- Create_tech_cpf
  - Liblist.f
- Pre synth
  - Expanded.f
  - Liblist.f
- Post Synth
  - Post synth netlist
  - Liblist.f
- Post ccopt
  - Post ccopt netlist
  - Liblist.f

General Project Files as defined in setup_project.csh

- Power Format File (CPF/1801)

4.3. Flow

Here are the various stages executed by CLP run scripts to perform low power check.

1. Update project.tcl with required locations and design information.
2. Read the library using the liblist.f
3. Read the RTL/netlist
4. Elaborate
5. Read power intent
6. Commit power intent
7. Analyze
8. Report rule check

4.3.1. Scripts

- run_clp.csh --> Set up IP Factory environment variables, and run the CLP tool sourcing the clp/* scripts.

- project_template.tcl --> a template version of the project.tcl file, to be copied to project.tcl in the user's work directory and modified as appropriate to the target design.
- Main command scripts for CLP
    - clp/clp_pre_synth.do
    - clp/post_synth.do
    - clp/post_ccopt.do

4.3.2. Checking results

The following reports will be generated and should be reviewed. See /scripts/setup/setup_dirs for report locations.

	Notes	Report
SETUP		
Has the correct CPF been read in?		$_CLP_RPT_PATH/clp_postsynth.log
Post Synth VERIFICATION		
Does the post synth netlist vs CPF, CLP check pass?		$_CLP_RPT_PATH/clp_postsynth.rpt
Post EDI VERIFICATION		
Does the post EDI netlist vs CPF, CLP check pass?		$_CLP_RPT_PATH/clp_postedi.rpt

Table 8 CLP reports

5. Joules

5.1. Overview

Joules is a RTL power analysis product that provides a unified platform for RTL designers. Being a dedicated RTL power analysis tool it has some advantages over using Genus for power reporting, such as graphical output, ability to vary voltage levels, and ability to support multiple power states.

It can read RTL and perform a trial synthesis and clock tree insertion prior to reporting power. This is not as robust as Genus/Innovus and we prefer not to use this option.

See the presentation here...

http://sharepoint.cadence.com/sites/wfo/ww_services/DIP_PMO/CoE/LowPower/Project%20Documents/Joules-for-customer-2014_10_22.pptx

5.2. Requirements

Joules requires the following files...

- .f files as used in synthesis
- Power format file (CPF/1801)
- Activity files from RTL simulations (TCF/SAIF or VCD)

- Joules_rtl.tcl
  - Expanded.f
  - tech_lib_setup.tcl
  - Power format file (CPF/1801)
  - Activity files from RTL simulations (TCF/SAIF or VCD)
- Joules_post_synth.tcl
  - Post synth netlist
  - tech_lib_setup.tcl
  - Power format file (CPF/1801)
  - Activity files from RTL simulations (TCF/SAIF or VCD)
- Joules_gls.tcl
  - Post ccopt netlist
  - tech_lib_setup.tcl
  - Power format file (CPF/1801)
  - Activity files from GLS simulations (TCF/SAIF or VCD)

5.3. Flow

The scripts support Joules at 3 stages, pre synthesis (RTL), post synthesis and post GLS.

Here are the various stages executed by CLP run scripts to perform low power check.

- Source project.tcl to set project variables/paths
- Create output dirs.
- Read library info
- Read design and elaborate
- Read stimulus

- Synthesise (If needed)
- Build clock trees (if selected)
- Report power

5.3.1. Scripts

- run_joules.csh --> Set up IP Factory environment variables, and run the Joules tool sourcing the joules/* scripts.
- project_template.tcl --> a template version of the project.tcl file, to be copied to project.tcl in the user's work directory and modified as appropriate to the target design.
- Main command scripts for Joules
    - joules/joules_rtl.tcl --> RTL level power reporting using Joules synthesis engine and RTL simulation activity files
    - joules/joules_post_synth.tcl --> Post Genus synthesis power reporting using RTL simulation files
    - joules/joules_gls.tcl --> Post GLS power reporting

5.4. Interesting Reading

5.5. Switches

Switches	Explanation
JLS_CLOCK	Build a clock tree in Joules to assist power estimation, default = 0
TYP_85C_CORNER	Use a 'HOT' typical corner for power estimation, default = 0
POWER_UNIT	Default mW
VCD_FILE	File detailing switching activity over time
VCD_TIMESTAMPS	The time frames of VCD file we wish to analyse
FRAME_COUNT	The number of frames of equal length to divide the VCD file into
ACTIVITY_DUT	Top level of VCD activity file if different from design.
POWER_STATES	List of power states. Each power state should have its own directory which contains all associated average activity files for that state. E.g. ${NL_TOGGLE_FILE_DIR}/*${power_state}*/*.tcf

Table 9 Joules switches

6. Innovus Placement

6.1. Overview

Innovus will be used for placement. Please note RDF utilises Innovus in common_ui mode.

The placement step takes the post synthesis netlist into Innovus, creates a dummy floorplan, and reports a typical area required to place the design without congestion.

Placement stage will highlight timing and area problems in the design.

It is strongly recommended to submit a job with multiple processors and use the multithreading switch. The scripts will utilise the multiple CPU's for multithreading, hence improving runtime.

A Rapid Adoption Kit (RAK) is available for anyone who is not familiar with the tool see: Table 2 Tool versions, training and help details

6.2. Requirements

Innovus requires the following files...

Output of Synthesis

- Post synth netlist
- .def floorplan (if available)
    - If there are large or multiple macros in the design some initial floorplanning is needed to create this floorplan.def file prior to running the script.

General Project Files as defined in the project.tcl

- SDC file
- Library information if not a standard library
- Power Format file (CPF/1801) if LP design

Library Files as defined by TECHNOLOGY environmental variable and sourced in $IPF_DESIGN_FLOW_SCRIPTS/tech_lib_setup.tcl

- Library files (LEF/LIB/DB)
- DONT_USE cell list
- TIE cell list

6.3. Flow

Here are the various stages executed by place.tcl to perform place and route:

1. Update project.tcl with required locations and design information. This file is read by place.tcl to set various project-specific TCL variables, which are then used by the Innovus script and other point scripts.
2. Source genus /mmmc.tcl
    1. define analysis views
    2. define constraint modes
    3. Add process specific derating factor
3. Read Netlist
4. Initialize design defining analysis views for optimization
5. Load floorplan file (if available)

6. Global variables setup
    1. Max route layer defined (related to TECHNOLOGY variable)
    2. set process node (affecting RC extraction)
    3. Reorder scan during placement
7. Read in Scan Def (if available)
8. Load and commit power format file to ensure innovus specific cells are added (e.g. power switches)
9. Place hard macro on bottom of design (if hard macro exists)
10. Plan Design if separate power domains exist
11. Place and optimize design
12. Check Placement
13. Time Design
14. Add Tie HI/LO cells
15. Report final timing
16. Report power
17. Save Design

6.3.1. Scripts

- run_phys.csh --> Set up IP Factory environment variables, and run the Innovus tool sourcing the innovus/innovus_place.tcl script.
- project_template.tcl --> a template version of the project.tcl file, to be copied to project.tcl in the user's work directory and modified as appropriate to the target design.
- innovus/innovus_place.tcl --> the main command script that is read into EDI.
- mode and analysis view. This sets the MMMC definition

6.3.2. Known Limitations

Currently the scripts do not have any built-in support for templating. The current usage model is that a project.tcl file will exist in the user's current working directory

6.3.1. Checking results

The following reports will be generated and should be reviewed. See /scripts/setup/setup_dirs for report locations.

	Notes	Report
Setup		
Are dont_use cells defined?		tech_lib_setup.tcl or ${_PNR_RPTS_PATH}/encounter.place.cmd
What is the max route layer?		tech_lib_setup.tcl or ${_PNR_RPTS_PATH}/encounter.place.cmd
Multi VT Library Choice	There may be a requirement for design to be HVT only - check with tech lead for reqs	$DUT_PATH/cfg/${CONFIG}/project.tcl
Reports		
Are all cells placed?		${_PNR_RVW_PATH}/$DESIGN.postPlace_CheckPlace.rpt
What is the density?		${_PNR_RVW_PATH}/$DESIGN.postPlace_CheckPlace.rpt
Does the design meet timing?		${_PNR_RVW_PATH}/$DESIGN.post_PlaceOptTimingRpts/
Setup		$DESIGN.summary
"Hold		$DESIGN_hold.summary

(If holds are less than clock hold uncertainty as defined in SDC file the design can be considered ok for hold timing at this stage)"		
"Do all inputs have a drive assertion? (with exception of clock and scan ports)"		${_PNR_RVW_PATH}/TimingCheck_PrePlace.rpt
"Are there unconstrained endpoints? (with exception of clock and scan ports)"		${_PNR_RVW_PATH}/TimingCheck_PrePlace.rpt
What is the total area?		${_PNR_RVW_PATH}/$DESIGN.placed.gateCount
What is the leakage power as reported by EDI?		${_PNR_RVW_PATH}/$DESIGN.post_PlaceOpt_Pwr.WC_$mode.leakage.rpt
What is the VT distribution?		${_PNR_RVW_PATH}/$DESIGN.post_PlaceOpt_Pwr.WC_$mode.leakage.rpt

Table 10 Placement reports

6.4. Interesting Reading

6.5. Switches

Switches	Explanation
THREADING	Use multiple threading, default = 0
POWER_SWITCHES	Insert power switches, default = 0 (if LP)
FP_DEF	Floorplan def file
SWITCH_DOMAIN	Name of switchable power domain (if LP)
SWITCH_PITCH	Pitch of power switches (if LP)
SWITCH_CELL	Power Switch Cell (if LP)
ENABLE_NET_IN	Name of enable net to switchable domain (if LP)
ENABLE_NET_OUT	Name of enable net from switchable domain (if LP)

Table 11 Placement switches

7. Innovus CCopt

7.1. Overview

Inserting clocks trees and propagating the clocks is important to prove the SDC's and clocking structure of the design.

Poor clock tree insertion, due to clocking structure or SDC's will make it difficult to close timing and likely increase the area and power of the design, thus diminishing our IP competitiveness.

It is important to run this stage to verify clocks can be built correctly and post clock tree timing can be closed.

Innovus will be used to call CCopt.

A Rapid Adoption Kit (RAK) is available for anyone who is not familiar with the tool see: Table 2 Tool versions, training and help details

7.2. Requirements

CCopt requires the following files...

Output of Innovus Placement

- Post placed database

General Project Files as defined in the project.tcl

- SDC file
- Library information if not a standard library
- Power Format file (CPF/1801) if LP design

Library Files as defined by TECHNOLOGY environmental variable and sourced in $IPF_DESIGN_FLOW_SCRIPTS/tech_lib_setup.tcl

- Library files (LEF/LIB/DB)
- DONT_USE cell list
- Clock Buffer list

7.3. Flow

Here are the various stages executed by innovus_ccopt.tcl to perform clock tree insertion:

1. Restore the saved placed design from the end of place.tcl
2. Apply propagated .libs instead of ideal for hard macros
3. Set CCOPT variables
    1. setCCOptMode -cts_buffer_cells $CLK_BUFFER
    2. setCCOptMode -cts_opt_priority power
    3. setCCOptMode -route_top_bottom_preferred_layer 4
    4. setCCOptMode -route_top_top_preferred_layer 7
4. Set process node to ensure correct RC extraction methodology is used by tool
5. Remove any preexisting clock trees
6. Build clock trees - CCopt Design
7. Optimise Design for setup and hold

1. Only hold violations up to 50ps will be optimized. Anything over 50ps could indicate a design/SDC flaw and will need reviewed.

8. Report Gate count
9. Time Design (including hold report)
10. Report power
11. Write propagated clock SDF
12. Save Design

7.3.1. Scripts

- run_phys.csh --> Set up IP Factory environment variables, and run the Innovus tool sourcing the innovus/innovus_ccopt.tcl script.
- project_template.tcl --> a template version of the project.tcl file, to be copied to project.tcl in the user's work directory and modified as appropriate to the target design.
- innovus/innovus_ccopt.tcl --> the main command script that is read into Innovus.

7.3.2. Known Limitations

Currently the scripts do not have any built-in support for templating. The current usage model is that a project.tcl file will exist in the user's current working directory

7.3.3. Notes

While CCopt requires minimal input from the designer to build the clock trees, the designer should make themselves familiar with the commands and clock constraints generated by CCopt when building the clock trees.

These can be found in the directories az_ccopt, and az_import, which are created by the tool.

For example to set a specific pin as a leaf pin, the file az_import/leaf.tcl could be edited.

7.3.1. Checking results

The following reports will be generated and should be reviewed. See /scripts/setup/setup_dirs for report locations.

	Notes	Report
Setup		
What clock buffer cells were used?		tech_lib_setup.tcl or ${_PNR_RPTS_PATH}/encounter.ccopt.cmd
Reports		
"What is the max skew for each clock tree? Worst skew should be WC_func_dc:setup.late corner"		${_PNR_RVW_PATH}/$DESIGN.ccopt/ccopt_skewgroups.rpt
Does the design meet timing?		${_PNR_RVW_PATH}/PostCCOPTIncrTimingRpts
Setup		$DESIGN.summary
"Hold Hold fixing is not performed by default,	Any hold violations below hold uncertainty +50ps	$DESIGN_hold.summary

there may be minor hold violations needing review."	can be waived as easily fixable by tools	
What is the total area?		${_PNR_RVW_PATH}/$DESIGN.PostCCOPTIncr.gateCount
What is the leakage power as reported by EDI post CCopt?		${_PNR_RVW_PATH}/$DESIGN.post_PostCCOPTIncr_Pwr.WC_$mode.leakage.rpt
What is the VT distribution?		${_PNR_RVW_PATH}/$DESIGN.post_PostCCOPTIncr_Pwr.WC_$mode.leakage.rpt

Table 12 CCOPT reports

7.4. Interesting Reading

CCOPT web support

CCOPT commands

7.5. Switches

Switches	Explanation
THREADING	Use multiple threading, default = 0

Table 13 CCOPT switches

8. Logical Equivalence

8.1. Overview

Logical equivalence checking is used to ensure that a generated netlist is functionally equivalent to the RTL and that the transformation process has not added any errors. This check can highlight RTL that may prove difficult for LEC to analyse so Run LEC early and often!!!!!

The dofiles generated by RTLCompiler will be used to perform Logical Equivalence Checking. This is expected to be sufficient at the module level. When moving to system level designs the Genus/RC dofiles scripts can become problematic and tailored scripts may need to be used.

A Rapid Adoption Kit (RAK) is available for anyone who is not familiar with the tool see: Table 2 Tool versions, training and help details

To access the latest updates on features, FAQs, guides and dofile scripts use the Web Interface (See below).

8.2. Flow

Use the 2-stage (mandatory) flow:

```
./run_phys.csh -lec_2stage
```

Single stage flow was proven to be difficult for LEC to understand all the optimization steps that took place in synthesis and is now supported only with Genus legacy mode(but also discouraged there). If there are any issues then use the 2-stage flow.

```
./run_phys.csh -lec_1stage
```

Final synthesis netlist vs PnR netlist:

```
./run_phys.csh -lec_syn2pnr
```

8.2.1. Scripts

2 scripts are generated by Genus during synthesis.

- Two-Stage Flow
  - rtl2mapped.do
  - map2final.do

Single Stage Flow - rtl2final.do, that can be used only with Genus legacy mode

8.2.2. Synth to PNR netlist Checks

Use the lec/lec_gate2gate.do to check equivalence between the synthesis and post pnr netlists.

Check that the scan constraints are correct.

Review the reports for non-equivalence, aborts and not-mapped points.

8.2.3. Checking results

In the 2 stage flow the following reports can be used to during analysis of the results.

	Notes	Report
Stage 1 - RTL vs Mapped		$_LEC_LOG_PATH/rtl2mapped.lec.log
a) Is the run Equivalent?		$_LEC_LOG_PATH/rtl2mapped_hier_comp.rpt
b) Are all blackboxes expected?	"HIER" are not an issue as they are created during the hierarchical run. Any other bboxes need to be reviewed, understood and expected.	$_LEC_LOG_PATH/rtl2mapped_bbox.rpt
c) Are there any "not-mapped" unmapped points?	This indicates that the design has not been fully mapped. If they exist in the RTL only they could have been removed during synthesis – was this expected?	$_LEC_LOG_PATH/rtl2mapped_notmapped.rpt
d) Are the extra and unreachable points expected?	Extra is usually due to scan, or logic added by synthesis. Unreachable key points are ones that do not cause a change in the functionality when '0' or '1' is applied. This could highlight deadcode.	$_LEC_LOG_PATH/rtl2mapped_unmapped.rpt
e) Are there any undriven signals?	Review to make sure they were expected and Tie off to appropriate levels in the RTL if possible.	$_LEC_LOG_PATH/rtl2mapped_floating.rpt
Stage 2 - Mapped to Final - is it Equivalent?		$_LEC_LOG_PATH/map2final.lec.log
a) Is the run Equivalent?		$_LEC_LOG_PATH/map2final_verification.rpt/ $_LEC_LOG_PATH/map2final_result.rpt
b) Are all blackboxes expected?	"HIER" are not an issue as they are created during the hierarchical run. Any other bboxes need to be reviewed, understood and expected.	$_LEC_LOG_PATH/map2final_bbox.rpt
c) Are there any "not-mapped" unmapped points?	This indicates that the design has not been fully mapped. If they exist in the RTL only they could have been	$_LEC_LOG_PATH/map2final_notmapped.rpt

	removed during synthesis – was this expected?	
d) Are the extra and unreachable points expected?	Extra is usually due to scan, or logic added by synthesis. Unreachable key points are ones that do not cause a change in the functionality when '0' or '1' is applied. This could highlight deadcode.	$_LEC_LOG_PATH/map2final_unmapped.rpt
e) Are there any undriven signals?	Review to make sure they were expected and Tie off to appropriate levels if possible.	$_LEC_LOG_PATH/map2final_floating.rpt
Stage 3 – gate2gate	Synth netlist vs PnR netlist	
Is the run Equivalent?		$_LEC_LOG_PATH /lec_g2g_reports/lec_g2g_results.rpt
All constraints expected	Check pin constraints, ignored outputs, etc	$_LEC_LOG_PATH /lec_g2g_reports/lec_g2g_environment.rpt
Are floating points expected	Undriven?	$_LEC_LOG_PATH /lec_g2g_reports/lec_g2g_floating.rpt
Are blackboxes expected?		$_LEC_LOG_PATH /lec_g2g_reports/lec_g2g_bbox.rpt
Any unmapped points?	Extra and unreachable	$_LEC_LOG_PATH /lec_g2g_reports/lec_g2g_unmapped.rpt
Any not-mapped points?	Design may not be fully verified as unmapped points are not checked.	$_LEC_LOG_PATH /lec_g2g_reports/lec_g2g_notmapped.rpt

Table 14 LEC reports

You may see modules that are Non-Equivalent in the logfile only to find later that the final result is Equivalent. This is likely due to dynamic flattening (see LEC UserGuide for more details).

8.2.4. Notes

- Recommendation is to always use the 2 stage flow.
- Debug using the GUI. On the command line use “set gui” when in vpxmode or “set_gui” when in tcl mode. Most command will work in both modes but in tclmode it is common to add a “_” where there is a space in vpxmode. e.g.

  “set system mode lec” would become:

  “set_system_mode lec”

- Alternate between tcl and standard(verplex) modes using “tclmode” and “vpxmode”

8.2.5. Known issue

There is a known issue inside the current tool version, captured in CCR #01644165 . While running LEC, users may experience the following error: -

// Error: Cannot create directory
/projects/rtldesignflow/work/anna/RDF_4_1_testing/Sierra_Beta/RDF_work/./projectsrtldesignflowworkannaRDF_4_1_testingSierra_BetaRDF_worksynthbeta_pwr_gnddata_sd0301m_t16ffc_11_vf162_2xa1xdh4xevhvh2r_mappedvg_projectsrtldesignflowworkannaRDF_4_1_testingSierra_BetaRDF_worksynthbeta_pwr_gnddata_sd0301m_t16ffc_11_vf162_2xa1xdh4xevhvh2rvgz_db: File name too long.

There are 2 possible workarounds.

1)The first one is to edit the map2final.lec.do file found in the synth/$STAMP/data_$CONFIG

set_verification_information
/projects/rtldesignflow/work/anna/RDF_4_1_testing/Sierra_Beta/RDF_work/./projectsrtldesignflowworkannaRDF_4_1_testingSierra_BetaRDF_worksynthbeta_pwr_gnddata_sd0301m_t16ffc_11_vf162_2xa1xdh4xevhvh2r_mappedvg_projectsrtldesignflowworkannaRDF_4_1_testingSierra_BetaRDF_worksynthbeta_pwr_gnddata_sd0301m_t16ffc_11_vf162_2xa1xdh4xevhvh2rvgz_db

to

set_verification_information ./temp_db

and re-run.

This may or may not work as has resulted in some cases as LEC INCOMPLETE.

The alternative workaround is

2) revert to the old flow, edit genus_synth.tcl to include the attribute **set_db wlec_write_lec_flow false**

Before you re-run synthesis, several other RDF scripts also will need to be modified inside scripts/genus directory: -

lec_pre_read.do

add vpxmode to end of file and change

// Generated by Cadence Encounter(R) RTL Compiler RC14.10 - v14.10-p008_1

to

Generated by Cadence Encounter(R) RTL Compiler RC14.10 - v14.10-p008_1

lec_pre_compare.do

add vpxmode to end of file

lec_pre_compare_rtl2map.do

add vpxmode to end of file

lec_pre_exit.do

add vpxmode to end of file

lec_pre_compare_map2final.do

add vpxmode to end of file

lec_pre_exit_m2f.do

add vpxmode to end of file

8.3. Web Interface

The Web Interface:

- Does not require a special license
- Enables access to information about a running batch job
- Offers more user-friendly viewing of logfiles
- Common web browsers supported (Internet Explorer, Firefox, etc …)
- Delivers a unique mechanism for information deployment

Use the following commands to invoke LEC and enable the web interface

```
lec -nogui
SETUP> set web_interface on
// Server URL is http://hostname:8090
```

A URL will be returned which you should paste into a browser to access the database.

8.4. RTL vs RTL checks

It is recommended that the user follows the latest methodology as prescribed by the Conformal team via documentation on obtained from the we_interface above.

Open the web interface and navigate to the Doc→Sample Dofiles→LEC Sample Dofiles → RTL versus RTL.

8.5. Interesting Reading

Please review Basic LEC Training

This will provide useful information on the terminology used with LEC.

9. Post Gate Level Simulation Power Reporting

9.1. Overview

The file rc_power.tcl will report the power figures based on post gate level simulation activity file and the netlist created post ccopt.

9.2. Requirements

Netlist

TCF activity files in the $NL_TOGGLE_FILE_DIR/$POWER_STATES directory. Each power state will have its own directory, where the corresponding TCF file must be placed.

If there is more than one TCF file for a state the script will average the files for the power reporting.

Power states defined in project.tcl

9.3. Flow

- rc -file rc/rc_power.tcl

9.3.1. Power Reports

The directory $PWR_RPTS_PATH will contain all the power reports in html format.

${_PWR_RPTS_PATH}/${DESIGN}_${power_state_clean}.nl.power.html

9.4. Interesting Reading

10. Trial ATPG

10.1. Overview

New tool, called Modus has been introduced to perform ATPG from this release onwards. Test sequences are generated that make use of the scan chains inserted during synthesis. The scan chains provide a mechanism for serially loading a design state prior to each test, while simultaneously unloading the resultant state of the previous test. The tests themselves attempt to detect static (stuck-at) and dynamic (at-speed) faults within the design. Note that the tests are intended to identify manufacturing (not design) faults.

We include this step in the flow so that we can generate fault coverage data that can be quoted in product literature. The reports can also be used to identify parts of the design that are resistant to scan-based testing. Acceptance criteria is defined for static and dynamic test coverage to ensure that our IP meets quality thresholds for testability.

A Rapid Adoption Kit (RAK) is available for anyone who is not familiar with the tool see: Table 2 Tool versions, training and help details

10.2. Requirements

Besides netlist(s), and models of the analog instances (for designs containing AMS blocks, only), ATPG requires a valid clock constraint file to provide the tool with valid clock-to-clock relationships and associated frequencies, as well as pin assignment files for stuck-at and at_speed tests. Clock constraints and pin assignment files are all generated automatically in the flow during synthesis. In addition, ATPG requires the so-called Tester Description Rule (TDR) file, to emulate the external automatic test equipment conditions. The default file is provided within Modus and can be changed within setup_project.csh (the variable $TDRPATH). The file name needs to be ‘dummy.tdr’ Also, at-speed test may require sdc.

10.3. Flow

Here are the various stages executed by modus_flow.tcl to perform ATPG:

1. Update setup_project.csh and project.tcl with required locations and design information. This files are used by modus_flow.tcl to set various project-specific TCL variables, which are then used by the Modus script and other point scripts.
2. Setup stage
    a) Check global attributes
    b) Read tech library files as determined by value of TECHNOLOGY variable.
    c) Check that <design_top>.<testmode>pinassign file contains mode signals, shift enables, resets, clocks, scan chains) and ensure resets have attribute +SC.
    d) Review the clock constraints file <design_top>/dft/constraints/clock_constraints.sdc as created by synthesis run, see 3.3
    e) If necessary change timing of scan in “padding file” (see next page for further explanation).
3. ATPG execution stage

***NOTE*:** There is a restriction in the dummy.tdr file called by default flow for clocks < 400Mhz. This can be overridden using a local dummy.tdr file by specifying the environment variable setenv TDRPATH (in setup_project.csh) and adjusting the parameters as required. The example is in the sequel.

The ATPG execution stage consists of the following

1) Build Model (reads in netlist from synthesis stage)
   a) Check that blackboxes are as expected. Set Boolean variable BLACK_BOX in project.tcl to '1' to indicate the existence of analog macros. The analog macros must refer to their HLM representation (High-Level Models), not the MLM (Medium-Level Models) ones.
2) Build Testmode
   a)
3) Verify Test Structures
   a) Check that all scan chains are observable and controllable.
4) Report Test Structures
5) Build Fault Model
6) Run ATPG
   a) Creates scan chain logic and delay test
   b) Creates static logic tests
   c) Reads in clock constraints
   d) Creates dynamic (at-speed) logic delay tests
   e) Commits the tests
7) Write Vectors.
   a) ATPG tool uses default waveforms for scan signals, which can be changed if necessary through ./modus/write_vectors_template.txt, see below for explanations.
8) Review logfiles.

***NOTE*:** For this release, the decision has been taken to separate stuck-at from at-speed test mode explicitly, in order to support IPS requirements. In a number of case, the IPS require separate approach for ATPG and rely on two independent test modes, captured within pin assignment files. Per default, pin assignment files are created by Genus for both stuck-at mode (FULLSCAN) and at-speed (DELAY), with an equal content. Typically, they will be placed under:

${_OUTPUTS_PATH}/$testmode/$DESIGN.$testmode.pinassign

When necessary, the pin assignment file for DELAY can be changed to allow for multiple test modes option. This can be done through AT_SPEED_PINASSIGNMENT_FILE variable in project.tcl, to point to a new file. This should be done if and only if the settings between two test modes are different. Also mind that the mode names (FULLSCAN and DELAY) are pre-defined within the flow and cannot be changed.

10.3.1. Scripts

From this release onwards, ATPG run scripts are now changed to allow for multiple options, similar to other tools. The scripts are listed below.

- run_atpg.csh --> Set up IP Factory environment variables, and run the Modus tool sourcing the modus/modus_flow.tcl script.
- project_template.tcl --> a template version of the project.tcl file, to be copied to project.tcl in the user's work directory and modified as appropriate to the target design.
- modus/modus_flow.tcl --> the main command script
- modus/modus_reports.tcl -> coverage report script for Modus flow
- modus/modus_clock_checks -> script to automatically check ATPG logfiles for clock constraints consistency
- et/et_flow.tcl --> the legacy mode command script.
- et/et_reports.tcl -> legacy mode for ET coverage
- modus/padding_setup.tcl -> Padding setup script for write_vectors command

Modus ATPG run supports therefore the following arguments with run_atpg.csh script.

Switch on run_atpg.csh	Function
-h/help	Displays information on available switches
-atpg	Runs ATPG and reports coverage using Modus.
-atpg_legacy	Runs ATPG and reports coverage using ET
-padding_parallel <filename>	Option to control timing relationship between test control signals for parallel testbench generation, specified in file <filename>. Can be used only with Modus!
-padding_serial <filename>	Option to control timing relationship between test control signals for serial testbench generation, specified in file <filename>. Can be used only with Modus!
-default	In case no options specified, the default run is Modus ATPG

Table 15 run_sims_atpg.csh switches

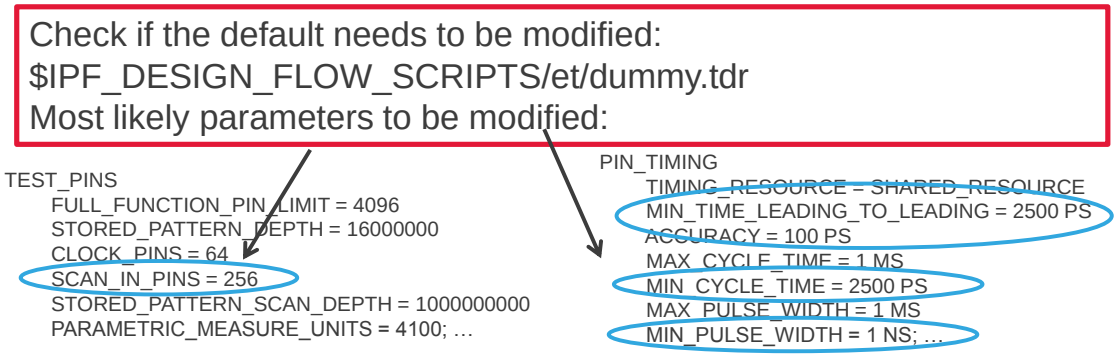
10.3.2. Updating dummy.tdr

For information on the individual parameters in the dummy.tdr file, refer to Encounter® Test: Guide 2: Testmodes manual (Pin Timing Section): -

http://support.cadence.com/wps/PA_DocumentViewer/pubs/et_ug_testmodes/et_ug_testmodes14.1.102/et_ug_testmodes.pdf

If the default flow falls over, it should state in * log_create_logic_delay_tests_FULLSCAN_logic_delay_pi_trans* which parameters are exceeded.

From the DfT training, we highlighted the most likely parameters that would potentially require modification for certain designs.



Note that the table above is only an indication and that other parameters from the dummy.tdr file may also have to be altered. Most notably, for designs with using high clock frequencies, the MIN_SCAN_PULSE_WIDTH parameter will need to be decreased to reflect the frequency requirement.

Users are encouraged to contact DfT CoE for guidance if unsure when having to use a local TDR file.

10.3.3. Updating dummy.tdr

In some cases, the ATPG simulations would fail because of the inappropriate conditions of scan control and data signals during shift and capture cycle. This is usually a consequence of timing in the design. In practice, this is not a problem, as test engineering team can introduce the necessary conditioning between scan signals. The padding template file (write_vectors_tempate.txt) emulates exactly this situation and allow user to define timing relations between various scan signals, as well as pulse width, test clock periods etc. This file should be copied to another location and follow the

instructions for setup provided within the template file itself. It is recommended to have separate files for parallel and serial simulations, though not mandatory. Obviously, if nothing specified, the flow will assume the defaults value from Modus.

Mind: this option should be used only in case when ATPG simulations fail and if the root cause points to inappropriate relation between scan signals.

#*Example 1*: Parallel simulations with scan to define padding relations between test signals

```
-explicitshifts 2
-scanbidioffset 2
-scanperiod 10
-scanpioffset=0
-scanpioffsetlist=cdb_pclk=4,cmn_psmclk_in=4,cmn_scanclk_fullrt=4
-scanpulsewidth 2
-scanstrobeoffset 3
-scantimeunits ns
-testbidioffset 0
-testperiod 20
-testpioffset 0
-testpioffsetlist
-testpulsewidth 1
-teststrobeoffset 19
-testtimeunits ns
-waitcycleafterdynamic 2
-waitcyclebeforepulse 1
-waitcyclebeforescan 2
-waitcyclebeforescanprecond 1
-waitcyclebeforescanexit 1
```

The parameters specified above will be used by write_vectors command of Modus. Please check Modus manual for the list of all supported arguments.

This file can also facilitate the simulation debug, as given in the second example:

#*Example 2*: Simulation debug example

```
-testrange=2.1.1.2.1.5
#This will force write vectors to generate only the testbench containing the timing window for the given pattern
-testrange=1.2.1.2.1:1.2.1.3.15
#This will cause write vectors to generate only the testbench with the odometer range above.
```

10.3.4. Notes

Clock constraint file for at-speed ATPG is generated during synthesis, based on the valid paths between different clock domains that are described in sdc. The matter is that the clock-to-clock relationships will be generated even in case when there is only one single path between two domains, which may unnecessarily increase the run-time as the Encounter Test would try to generate patterns for all paths, including those that are not valid. The suggestion is to always try to capture valid and invalid clock-to-clock relationship in the dedicated at-speed sdc and use this file during synthesis. In addition, the clock constraint file should always be reviewed, in case when the removal of certain clock domains inter-relations may actually significantly increase run time with insignificant loss of fault coverage.

10.3.5. Checking results

Check	Notes	Actions	Report
SETUP			
Has the correct NETLIST been read in?	Manually check the correct netlist has been used as the flow will pick up either post layout or post synth. If available, this should always be post layout		$_ATPGWORK_PATH/testresults/logs/log_build_model
Define any macros present in the design and how they are handled (e.g. .lib)	The .lib files ised in synthesis should have been represented with their (.v) models for ATPG. In addition, $DEFINEMACRO variable should contain any macro (ifdef) as requested for ATPG		$_ATPGWORK_PATH/testresults/logs/log_build_model
Have the following clocks being properly applied?	The clocks should be defined as -ES		${_OUTPUTS_PATH}/$testmode/$DESIGN.$testmode.pinassign
Have the following at-speed frequencies been applied?	This relates to at-speed frequencies		$ATPG_CLOCK_CONSTRAINTS_FILE
Have resets been correctly defined as +SC			${_OUTPUTS_PATH}/$testmode/$DESIGN.$testmode.pinassign
VERIFICATION			
Are there any errrors in the Logfiles			$_ATPGWORK_PATH/testresults/logs/*log*
Have severe warnings been reviewed?			$_ATPGWORK_PATH/testresults/logs/*log*
Are all existing blackboxes expected?			$_ATPGWORK_PATH/testresults/logs/log_build_model

Are all scanchains controllable and observable? ((TTM-357))			$_ATPGWORK_PATH/testresults/logs/log_build_testmode_<testmode>
Are all scanchains controllable and observable? ((TSV-381))			$_ATPGWORK_PATH/testresults/logs/log_verify_test_structures_<testmode>
Is static Global %ATCov >99%			$_ATPGWORK_PATH/testresults/logs/log_create_logic_delay_tests_DELAY_logic_delay_pi_trans
Is dynamic Testmode %ATCov >**90%**	This figure assumes I/Os are dynamically controllable for ATPG using the allowedpitransitions keyword		$_ATPGWORK_PATH/testresults/logs/log_create_logic_delay_tests_DELAY_logic_delay_pi_trans
Messages Requiring User Action	**User Response**	**Possible Causes**	
(TSV-093) / (TSV-193): These messages indicate possible three-state contention.	Correct the testmode definition or rerun ATPG using the contentionprevent and/or contentionreport keywords.	The following are possible causes: ❍ A design error (multiple drivers) ❍ Missing interface files so Modus does not interpret pin direction correctly. ❍ Incomplete modelling.	$_ATPGWORK_PATH/testresults/logs/log_build_model
(TSV-034) / (TSV-035): These messages indicate a keeper device that is exposed to glitches	Modify the design so that the glitch source is prevented from gating the keeper in this test mode	The following are possible causes: ❍ A design error	$_ATPGWORK_PATH/testresults/logs/log_build_model

(TFW-060): This message indicates a problem reading the globalDataFile	Check error reason supplied within the message. ❍ Does the file exist? ❍ Do you have the correct permissions for the file?	The following are possible causes: ❍ Failure of previous step to complete without error	$_ATPGWORK_PATH/testresults/logs/*log*
(TSV-384) / (TSV-385): These messages indicate that the design contains scan chains that are broken	Select the specific message from the Specific Message List. The schematic display is updated to show the specified Scan pin and the deepest register of the controllableonly scan chain. The circuit is set to the Scan state.	The following are possible causes: ❍ Incorrect stability values on SE, clock or reset PIs ❍ For mux scan designs, the scan path data input of the mux is not selected by the scan state.	$_ATPGWORK_PATH/testresults/logs/log_verify_test_structures_<testmode>
(TSV-095): This message indicates a potential clock race	Investigate the instances reported	The following are possible causes: ❍ insertion of lockups at the head of the scanchains, in this instance this is fine and the message can be ignored	$_ATPGWORK_PATH/testresults/logs/log_verify_test_structures_<testmode>

Table 16 Modus reports

10.4. ATPG Simulations Flow

From this release onwards, ATPG simulations will use Xcelium simulator, where certain increase of performance level against Incivie simulator has been observed, at least to some degree. The scripts have been modified accordingly and need to be run at the following stages:

1. Run Simulation (zero delay)
2. Review simulation logfiles
3. Run Simulations (sdf)
4. Review simulation logfiles

10.4.1. Scripts

- run_sims_atpg.csh --> Set up IP Factory environment variables, and run the Xcelium simulator according to either default settings or settings defined within the argument switches in the table below.
- project_template.tcl --> a template version of the project.tcl file, to be copied to project.tcl in the user's work directory and modified as appropriate to the target design.
- nc/run_zero_sim.tcl --> the main command script to run zero delay DfT simulations.
- nc/run_sdf_sim.tcl --> The main command script to run back annotated SDF DfT simulations

- nc/create_snapshot.tcl --> The script to compile and elaborate the design for simulation.
- nc/annotate_sdf.tcl -> The script for sdf annotation. Used only within run_sdf_sim.tcl
- nc/support_files/template_sdfcmdfile.do --> Template script used to specify design-specific annotations.
- nc/force.inp --> input file for the simulation that contains force signals

The ATPG simulations have to be carried out with and without the timing information applied on the top-level of the design. Ultimately, both will have to pass when the timing constraints are met during the design.

Switch on run_sims_atpg.csh	Function
-h/help	Displays information on available switches
-gui_p[arallel]	Interactive run with gui for parallel simulations
-waves_p[arallel]	Waveform input file will be created with all nets/pins included for parallel simulations.
-gui_s[erial]	Interactive run with gui for serial simulations
-waves_s[erial]	Waveform input file will be created with all nets/pins included for serial simulations
-gui_p[arallel]0	Interactive run with gui for parallel simulations with zero delay
-gui_s[erial]0	Interactive run with gui for serial simulations with zero delay
-f[orces]	Force specific values to be applied on user specified nets/ports during all simulations. The input file is located at ./nc/support_files/force.inp and requires editing prior to simulation run.
-zero_only	Only the zero delay simulation will be performed, in combination with other arguments, if given
-sdf_only	Only the sdf simulation will be performed, in combination with other arguments, if given
-other_args <filename>	An option to give additional arguments to simulation, captured in <filename> file.
-default	In case no options specified, the default simulation will run as in previous releases.

Table 17 run_sims_atpg.csh switches

It is possible to combine the switches, as long as that make sense. In general, the switches are introduced to facilitate the debugging process and should be used only when necessary. For example, the –f(orce) switch enables forcing of the internal signals to a predefined state during simulation. This might be required in AMS design when analog power supplies are required to be active (and defined in digital simulation) when analog block harbors embedded digital block. The forces will have to be defined in the nc/force.inp file, the syntax template is also given there. The waves group of switches write all ports and nets of the design in the database whose waveforms can be viewed after the simulation with the '*simvision*' tool.

The latest RDF patch now also support manifest file (*.f) for the ATPG simulations. The manifest file needs to be defined in project.tcl file by setting the following variable:

set SIM_OTHER_MODULES_NETLISTS <manifest.f>

Another enhancement is the addition of the variable to deal with the typical situations in the PMAs, where the top level netlist is placed at a location that does not conform the RTLDesignFlow structure. The variable that needs to be set is

setenv PMA_NETLIST_RELEASE_PATH

This variable is also useful for the final check of delivery area consistency, when the final ATPG simulation have to point to the release area where the data need to be taken from.

Designs containing AMS blocks should preferably use MLM (medium level models) for simulation. and not HLM (High-Level Models) that are used for ATPG. The user can select that through the MLM_ATPG_SIM variable in project_tempalte.tcl.

10.4.2. Known Limitations

The sdf simulations in RDF 4.1 are multi-mode simulations taking corresponding sdf for scan shift, scan capture and at-speed simulations. In case there is only functional simulation mode, all three mentioned simulations will be performed based on functional sdc.

The SDFNAME variable in project.tcl file is set by default to:

${_PNR_DATA_PATH}/$DESIGN.postccoptincr

This variable should not be changed for soft IPs. In case of hard IPs, this variable might need to be changed to point to the path where sdf data live. The sdf basename (without mode and corner data) also needs to be included. If the sdf includes $corner (typical case for the signoff of hard cores), the sdf file name must comply with the following notation:

${DESIGN}.${design_mode}.${corner}

Or

${DESIGN}.${design_mode}_${corner}

Currently, the script uses the hardcoded corner names (wc and bc). If the actual corner names are different, they will have to be specified under ‘corners’ variable in the project.tcl.

Also refer to other issues here

10.4.3. Notes

For step by step guidance on how to setup and run simulations with multiple modes and or sdf files, please see here

10.4.4. Checking results

In the 2 stage flow the following reports can be used to during analysis of the results.

Check	Notes	Report
Setup		
Have both wc and bc corners been run		$DUT_PATH/cfg/${CONFIG}/project.tcl $corner wc/bc logfiles (${_SIMWORK_PATH}/$corner/ncsim_${testname}_${corner}*.log)
Is the correct SDF file being read		$_SIMWORK_PATH/$corner/ncsdfc.log

Verification		
Are there any errrors in the Logfiles	Mind the difference between zero delay and sdf simulations	$_SIMWORK_PATH/*.log (for zero only) $_SIMWORK_PATH/$corner/*.log (for sdf only)
Did the SDF's annotate to the design without any errors/warnings?	Look for the section Annotating SDF timing data: in the logfile. Review any warnings	$_SIMWORK_PATH/$corner/ncelab_*_${corner}_*.log
Did the SDF's annotate to the design fully	Look for the section SDF statistics: check that Annotated = 100.00%	$_SIMWORK_PATH/$corner/ncelab_*_${corner}_*.log
Do zero delay scan shift tests pass?	INFO (TVE-203): The total number of miscomparing vectors is 0	$_SIMWORK_PATH/ncsim_FULLSCAN_data_scan_ex*_ts*_<serial/parallel>.log
Do zero delay stuckat patterns pass?	INFO (TVE-203): The total number of miscomparing vectors is 0	$_SIMWORK_PATH/ncsim_FULLSCAN_data_logic_ex2_ts*_<serial/parallel>.log
Do zero delay at-speed patterns pass?	INFO (TVE-203): The total number of miscomparing vectors is 0	$_SIMWORK_PATH/ncsim_DELAY_data_logic_ex2_ts*_<serial/parallel>.log
Do wc/bc scan shift tests pass?	INFO (TVE-203): The total number of miscomparing vectors is 0	$_SIMWORK_PATH/$corner/ncsim_FULLSCAN_data_scan_ex*_ts*_${corner}_<serial/parallel>.log
Do wc/bc stuckat patterns pass?	INFO (TVE-203): The total number of miscomparing vectors is 0	$_SIMWORK_PATH/$corner /ncsim_FULLSCAN_data_logic_ex2_ts*_${corner}_<serial/parallel>.log
Do wc/bc at-speed patterns pass?	INFO (TVE-203): The total number of miscomparing vectors is 0	$_SIMWORK_PATH/$corner/ncsim_DELAY_data_logic_ex2_ts*_${corner}_<serial/parallel>.log
Have all testbenches been simulated	cross check the number of testbenches agains the ncsim logfiles. Also mind the difference for zero delay and sdf simulations	Testbenches: - ${_ATPGWORK_PATH}/parallel/VER.$(testmode}.data.scan.ex*.ts*.verilog ${_ATPGWORK_PATH}/parallel/VER.${testmode}.data.logic.ex*.ts*.verilog ${_ATPGWORK_PATH}/serial/VER.${testmode}.data.scan.ex*.ts*.verilog ${_ATPGWORK_PATH}/serial/VER.${testmode}.data.logic.ex*.ts*.verilog logfiles:- $_SIMWORK_PATH/ncsim_${testname}_*_serial.log $_SIMWORK_PATH/ncsim_${testname}_*_parallel.log $_SIMWORK_PATH/$cornerncsim_${testname}_*_serial.log $_SIMWORK_PATH/$corner/ncsim_${testname}_*_parallel.log

Table 18 ATPG Simulation reports

10.5. ATPG for DDR

As a part of an integral effort of RTLDesignFlow adoption within DDR business unit, new scripts were made to support the ATPG of DDR High-speed PHY specific configurations. Typically, their HS PHY configurations make use of OPCG (On-Product Clock Generation) scheme that previous RTLDesignFlow releases did not support. For that sake, new ATPG script, as well as the new setup file have been made, that facilitate the transfer from DDR specific synthesis and P&R flow (make_bets) to RTLDesignFlow in terms of ATPG. Mind that ATPG for DDR still requires ET.

10.5.1. New scripts

Two new scripts have been added to the RTLDesignFlow standard suite

- run_atpg_ddr.csh → ATPG setup script and call to ATPG tool.
- et/et_flow_ddr.tc → Actual ATPG script.

The ATPG setup script has the following variables that need to be configured, as given in table below.

Variable	Function
DDR_NETLIST	Top-level netlist. It can be post-synthesis or post-layout
DDR_USER_SETUP	Technology specific file, as generated by DDR flow (make_bets) and requested for DDR specific ATPG
SCANCLOCKNAME	Name of the external clock (typically only one)
SCANCLOCKFREQUENCY	External scan clock frequency

Table 19 ATPG DDR variables

The actual ATPG script should not be modified. Of course, setup_project.csh still requires appropriate configuration (as explained above in the document), whereas project.tcl typically requires only two variables to be set

Variable	Function
ATPG_SDC_FILE	Path (not the actual file) to ATPG constraints
SIM_OTHER_MODULES_NETLISTS	Behavioural models of pads and any other mixed-signal modules

Table 20 ATPG DDR variables

Mind that the DDR flow does not have unified technology file, hence the tech_lib_setup file should contain only the following two variables:

Variable	Function
ATPGLIB	Standard cells that are used in the design
VFILE	Verilog models of the standard cells

Table 21 ATPG tech variables

The two files can actually point to the same content if Verilog libraries are also used for ATPGLIB, which is a typical case for 16ff and 10nm technology.

Mind that the new ATPG flow for DDR does not hamper nor interfere with any other part of RTLDesignFlow.

10.5.2. DDR ATPG simulations

The ATPG simulation scripts were also devised for DDR, with the scripts tuned for Incisive. There are three simulation scripts for three different modes to match current implementation within ATPG scripts. These are:

1. FULLSCAN – invoked with 'run_sims_atpg_DDR.csh'
2. FULLDYNAMIC – invoked with 'run_sims_atpg_DDR_FULLDYNAMIC.csh'
3. FULLDYNAMIC_slow – invoked with 'run_sims_atpg_DDR_FULLDYNAMIC_slow.csh'

The switches given in table 6 are also valid for each of the three simulation modes.

Note that FULLDYNAMIC and/or FULLDYNAMIC_slow can also be used in the conventional part of the RTLDeisgnFlow, if there are modes other than 'FULLSCAN' being used in ATPG script. The requirement is that in such case, the other modes need to match the mode names as used in DDR, i.e. 'FULLDYNAMIC' and/or 'FULLDYNAMIC_slow'. Mind that the ATPG does not fully support different modes in this version of the RTLDesignFlow for the flows other than DDR.

10.6. Interesting Reading

Training material for the DFT aspects of the RTLDesignFlow can be found here: RTLDesignFlow DFT Training

DFT Centre of Excellence

How to Analyze Low ATPG Coverage

A number of hard IPs will also need to be checked for the DfT consistency using the third party tools (Synopsys Tetramax). A user guide on how to setup the flow for Tetramax can be found here:

http://sp/sites/WFO/ww_services/wwverificationservices/ip_factory/Shared%20Documents/Flow%20related%20docs/RTLDesignFlow%20UserGuides/Supplement/Tetramax%20flow%20user%20guide.docx

10.7. Switches set in project.tcl used in ATPG and ATPG simulations

Switches	Explanation
ATPG_CLOCK_CONSTRAINTS_FILE	Modus clock constraint file for at-speed ATPG. It is auto-generated by flow, but it should be reviewed and potentially changed for some designs.
ATPG_SDC_FILE	Path to constraints file for at-speed ATPG, when required
HARD_MACRO	Switch to identify hard macro instances inside the IP
BLACK_BOX	Switch to identify black boxes (analog macros) inside the IP
MAXVECTORS	Specify maximum number of vectors written to test datafile for ATPG simulations
MLM_ATPG_SIM	Switch to indicate HLM or MLM model usage for ATPG simulations

Table 22 ATPG and Simulation switches

11. Conformal CCD

Conformal Constraint Designer is used to verify SDC (single and multi-mode), structural clock domain crossing verification (CDC) and RTL LINT.

11.1. Scripts

All files associated with executing a CCD run:

- run_ccd.csh → Setup script location, design name and call to CCD tool.
- ccd/cdc.do → The main dofile that is read into CCD. Includes LINT, Modelling and Clock Domain Checking.
- ccd/ccd_lint.setup → Includes call to refined HDL rule set and any filters related to hdl_checks
- ccd/report_cdc_check.tcl → script to provide a formatted summary report suitable for distribution.
- ccd/example_filters/filters_*.do → Example filters.
- ccd/ccd_lint_refined_rules_<datestamp>.txt → Refined rule set for CCD lint checks.

Notes

a) All required filters and initialization files should be created under the "_CCD_INPUT_PATH" directory.
b) If no filters exist, example filters will be copied to the appropriate directory which should then be updated accordingly.
c) You can enter GUI mode from the command line by typing "set gui" from vpxmode.
d) To toggle between tclmode and vpxmode simply type "tclmode" or "vpxmode".
e) Most tclmode commands need a "_" where a space exists. E.g. “set gui” → “set_gui”
f) Use the Expert Alias to get help with CCD - Email "ask_ccd_expert"

11.1.1. Blackboxes

Blackboxes can be specified in setup_project.csh. Blackboxes should be used for RAMs, analog blocks or other types of macros that don’t need to be analyzed.

/scripts/ccd/cdc.do can also be modified to add blackboxes (see Conformal documentation).

CCD will automatically blackbox .lib files when read in.

11.1.2. Switches

The following variables have been **removed** from project.tcl and instead can be provided on the run_ccd.csh command line (See Table 24 run_ccd.csh switches). LINT checks will always be run.

Variable	Function
CCD_CDC_STRUCT	Enable structural Clock Domain Checking
USE_REFINED_CCD_LINT_RULES	Use IPF refined rules for LINT
USE_REFINED_CCD_SDC_RULES	Use IPF refined rules for SDC
CCD_SDC_CHECKS	Enable SDC checking
CCD_FUNCTIONAL_CHECKS	Enable Functional Clock Domain Checking
CCD_MODELLING_CHECKS	Enable RTL Modelling Checks
CCD_SETRESET_CHECKS	Enable set/reset Checking

Table 23 CCD project.tcl variables (no longer applicable)

Example scripts to run scripts can be found within the top level of the delivered scripts tarball.

- run_ccd.csh - script used to execute ccd dofile

Switch on run_ccd.csh	Function
-h/help	Displays information on available switches
-exit	By default CCD will NOT exit on completion of the analysis. Use this switch to exit the run automatically.
-cpt <filename.cpt>	Restart CCD with the checkpoint file (used to allow store results for analysis at a later date)
-cdc_struct	Run structural CDC checks
-sdc	Run single SDC lint and Policy checks
-sdc_multi_mode	Run multi SDC checks (not other checks except lint will execute).
-model	Run RTL modelling checks
-sr	Run Structural set/reset synchronisation checks
-no_sdc_rules	Don't include refined SDC ruleset
-no_lint_rules	Don't include refined LINT ruleset
-default	Runs cdc_struct, single sdc, set/reset, set/reset sync checks and lint checks

Table 24 run_ccd.csh switches

- ccd/ccd_lint.setup - Add lint rules, disable checks, selects the refined rule set to use. Check that the latest available date stamped rule set is selected.
- ccd/example_filters/filters*.do – NOTE: This are just example files and any filters/waivers should be located in "$DUT_PATH/cfg/<config>/ccd/filter_*.do" This directory and files should be created after the first invocation of the script if it does not already exist. It can also be manually created.

  Notes
  1. Update project.tcl with required locations and design information.
  2. Exclude any design files that are don't need to be checked using "set rule handling -exclude" (See ccd_lint.do for examples)
  3. When adding a filter, the message that reported the issue is used and this contains the line number of the occurrence. If the code is updated so that the error moves to a different line then the filter will become redundant. Therefore, make sure you are filtering on the final code base prior to release or update the existing filters.

11.1.3. Severity Levels

There are 4 severity levels defined:

Error: These must be fixed before carrying on with further coding as simulation mismatches may cause re-work

Warning: These must be addressed prior to final release and should be checked/waived as appropriate

Note/Info: These must be reviewed but no waivers are required.

Ignore: These are of no consequence and can be discarded

11.1.4. Interesting Reading

The following document contains constraints checking advice when using CCD.

CCD SDC recommendations

A Rapid Adoption Kit (RAK) is available for anyone who is not familiar with the tool see: Table 2 Tool versions, training and help details

11.2. SDC Checks

CCD requires a functional SDC constraint file to setup the tool correctly for CDC analysis

11.2.1. Flow – Single mode

Following are the various stages required to run CCD to check SDC constraints:

1. Ensure the correct SDC file location is setup in setup_project.csh and that the filename follows the RTLDesignFlow convention.
2. Use the appropriate command line switch to enable SDC checks (no longer in project.tcl)
3. Perform Lint Checks
    a. Results are easier to review in the GUI. On the command line type "set gui"
    b. Open the SDC Lint Manager and debug any Errors. (Tools -> SDC Lint Manager)
4. Add filters if required and/or fix lint errors/warning before moving on to check policy rules.
    a. Add filters to a file "filters_sdc_lint.do" which should exist in the directory pointed to by the $_CCD_INPUT_PATH variable in project.tcl
5. Verify clock policy rules before any other rules
    a. Open the Rule Manager and debug Clock Policy Errors. (Tools -> Rule Manager -> sdc_clock_checks -> sdc_clock_checks)
6. Add filters if required and/or fix lint errors/warning before moving on to check policy rules.
    a. Add filters to a file called "filters_sdc_policy.do" which should exist in the directory pointed to by the $_CCD_INPUT_PATH variable in project.tcl.
7. Verify all other rules (includes clock rules above)
    a. Open the Rule Manager and debug Clock Policy Errors. (Tools -> Rule Manager -> sdc_clock_checks -> sdc_iodelay_checks/sdc_exception_checks/sdc_design_checks)
8. Add filters if required and/or fix lint errors/warnings
    a. Add filters to a file called "filters_sdc_policy.do" which should exist in the directory pointed to by the $_CCD_INPUT_PATH variable in project.tcl.
9. Repeat stages 3-9 if required.

11.2.2. Flow – Multi mode

Following are the various stages required to run Multi-Mode SDC checks. These types of checks will highlight missing constraints between the SDCs as well as lint and policy SDC issues.

1. Ensure the correct SDC file location is setup in setup_project.csh and that the filename follows the RTLDesignFlow convention. It is expected that all SDC files exist in the same directory.
2. Update the environment variable “DESIGN_MODES” within setup_project.csh with the modes to be verified.
3. Execute checks with “run_ccd.csh –sdc_multi_mode”
    a. NOTE: When running multi-mode checks no other checks except lint checks will be run as SDC checks only support multi-mode.
    b. SDC Lint checks will have individual reports generated for each mode.
    c. Policy checks will have a single report of merged results (SDC file is mentioned in results to know which mode is affected.)

11.2.3. Checking results

The following reports will be generated and should be reviewed. See /scripts/setup/setup_dirs for report locations. Stork will also perform this analysis.

Check	Notes	Report
VERIFICATION		
Are there any errors in the Logfile	Review	$_CCD_RPT_PATH/general/$DESIGN.log
Are all clocks defined?	Review	$_CCD_RPT_PATH/general/report.clocks
Have clocks been correctly grouped?	Review	$_CCD_RPT_PATH/general/report.clocks
LINT CHECKS		
a)Are there any LINT Errors (not filtered	Fix them - Any constraint that fails LINT will not be considered for Policy checks	$_CCD_RPT_PATH/sdc/sdc_lint_checks.nofiltered.error
b) Are there any Lint Warnings (not filtered)?	Fix or Filter them	$_CCD_RPT_PATH/sdc/sdc_lint_checks.nofiltered.warning
c) Are there any Lint Notes (not filtered)?	Review	$_CCD_RPT_PATH/sdc/sdc_lint_checks.nofiltered.notes
POLICY CHECKS		
a) sdc_clock_checks	Fix or Filter Errors/Warnings. Review Notes	$_CCD_RPT_PATH/sdc/sdc_clk_checks.nofiltered.error/warning/note
b) sdc_iodelay_checks	Fix or Filter Errors/Warnings. Review Notes	$_CCD_RPT_PATH/sdc/sdc_policy_checks.nofiltered.error/warning/note
c) sdc_exception_checks	Fix or Filter Errors/Warnings. Review Notes	$_CCD_RPT_PATH/sdc/sdc_policy_checks.nofiltered/warning/note
d) sdc_design_checks	Fix or Filter Errors/Warnings. Review Notes	$_CCD_RPT_PATH/sdc/sdc_policy_checks.nofiltered.error/warning/note
FILTERS		
Have all filters been reviewed		$_CCD_INPUT_PATH/ccd_filters.do

Table 25 CCD-SDC reports

11.3. Clock Domain Checks (CCD)

11.3.1. Overview

Conformal Constraint Designer will be used to perform structural clock domain checks including:

- Verify that the correct structures are in place at each clock domain crossing
- Synchronizer checks (E.g. Dual FF, Mux synchronizer, FIFO)
- Convergence checks (Individually synchronized signals converging at the same point in the destination domain)

- Set/Reset checks (Asynchronous set / Synchronous reset)

Here are some useful slides on terminology used within CCD to help understand errors and aid debug CCD Terminology.pptx

Guidance on multi-mode SDC checks is available here: **Error! Reference source not found.**

Guidance on how to handle failing FIFOs: Write FIFO Instance Usage

11.3.2. Requirements

CCD requires a valid SDC constraint file to provide the tool with the correct clocks and their relationship with all inputs/outputs of the design.

11.3.3. Flow

There are the various stages required to run CCD to check asynchronous clock domain crossing:

1) Declare static registers and static ports. CCD uses the cdc_config_object attribute for this.
   a) The RDF flow provides the ability to store statics in a file that can be used with both Spyglass and CCD. However, the same format won't be handled for both Spyglass and CCD. The recommendation is to write static that will work in CCD and make any required updates for other tools in that tools environment (e.g. SGDC in Spyglass).
   Place a statics file named "statics_<CONFIG>" in the directory "$DUT_PATH/cfg/<config>/statics/".
   Put a list of static inputs and registers in here WITHOUT the top level design name in the path!
   e.g. "SDI_BANK0/i_csi2rx_regs/static_conf_static_dphy_sel*"

2) Handle blocks where there is not RTL available.
   a) Add .lib file if available
      i) Additional .lib files can be appended to the SLOWLIB variable in project.tcl.
   b) If no .lib is available provide an empty module with IO definitions.

3) Declare blocks where the RTL is present but analysis is only required up to the interface of that block. CCD uses the "skip_instance" attribute. Currently, the /ccd/cdc.do file must be updated to add skip modules. See Section "Skip Instances" cdc.do.
4) Read in the design and check that it elaborates

5) Read in the SDC constraints and verify that the constraints are applied correctly. All clock related, input and output constraints must be setup correctly.
6) Check the clock groups that have been defined in the constraints prior to committing the groups. Check that synchronous clocks appear in the same clock group.

7) Check FIFOs are handled correctly:
   a) Valid FIFOs are automatically discovered using "add fifo instance -default".
   b) Failing FIFOs will not be considered for structural checks therefore any failing CDC paths crossing these FIFOs will not be reported so it's important that failing FIFOs are understood
   c) There are various ways to handle failing FIFOs:
      i) Fix the FIFO in the RTL
      ii) If the FIFO RTL is good then the FIFO can be manually added by providing FIFO parameters (See Write FIFO Instance Usage) and follow this guidance:
         (1) After automatic FIFO detection has been run - write out a file with the attributes for all FIFOs:
         'write fifo instance * -file <filename>–replace'

(2) Edit this file to update specific attributes that you believe are invalid (e.g. in CTC designs there may be more than one read pointer) or give the correct instances for the FIFO structure and pointers.
(3) Read in the new FIFO file on subsequent runs using dofile<filename> where you would normally do the automatic FIFO detection. There is no need to run the automatic FIFO detection again unless the design changes. You may need to add '-replace' on each 'add fifo instance' line in this file if you have already run automatic FIFO detection.
(4) Commit the FIFOs using 'commit fifo'
(5) Finally run a FIFO report using 'report fifo instance -all –verb'

iii) File a CCR if you believe the FIFO RTL is good and should be automatically detected by CCD.
iv) If the FIFO is "good by design" then just leave it as failing and as it should not cause structural fails.)

8) Execute the checks. The script allows the following checks to be performed:
   a) Lint checks
   b) Structural clock domain crossing checks
   c) Clock domain convergence checks
   d) set/reset synchronizer checks
   e) RTL modelling checks

9) Review reports
   a) Convergence:
      i) Convergence failure is more likely to be an issue when points converge at a FSM which could cause transfer to an invalid state and potential lock-up condition.
      ii) Convergence in the write data a memory interface is unlikely to cause a functional issue depending on the use of the data when read from the memory.
      iii) You can reduce the reports down to more manageable sizes for reviewing by changing some of the attributes available in Conformal. This involves manually changing the /scripts/ccd/cdc.do.
         (1) The types of convergence checks can be reduced to only analyze 'invector' paths. These are paths that converge from the different bits of the same bus. For example, gray coded buses.
            (a) set_attribute [find -ruleinst cdc_def_rs/conv_checks/] check_vector_conv invector*
            (b) Similarly, 'outvector'paths which are paths that converge from different vectors can be checked on their own by setting:
            (c) set_attribute [find -ruleinst cdc_def_rs/conv_checks/] check_vector_conv outvector*
            (d) Examples of the above and where to use them are available in the cdc.do script.
            (e) NOTE: Make sure the final analysis is run on both invector and outvector which is the default setting.

10) Add Filters – See Filters

11.3.3.1. Notes

By default, any unclocked IO is not considered for clock domain checks. However, later versions of the tool allow these to be checked by setting the following environment variable:

```
set attribute [find -conformal] cdc_validate_unclocked_to_clocked 1
```

11.3.4. Checking results

The following reports will be generated and should be reviewed. See /scripts/setup/setup_dirs for report locations. Stork will also perform this analysis.

Check	Notes	Report
SETUP		
Are all clocks defined?		$_CCD_RPT_PATH /general/report.clocks
Are the any errors SDC reading in the SDC		$_CCD_RPT_PATH /sdc/sdc_report.summary
Have related clocks been grouped correctly?		$_CCD_RPT_PATH /general/report.clocks
FIFOs	CDC issues around failing FIFOs will not be reported anywhere else and the reasons for failing must be understood	
a) Do all FIFOs pass auto detection?	Report fifo tells you what have passed and failed auto detection. The tool will report on all 2-d arrays	$_CCD_RPT_PATH /general/report.fifo
b) Are any failing FIFOs 'good by design'	If yes, document why	$_CCD_RPT_PATH /general/report.fifo
c) Have any FIFOs been passed by manually guiding the tool?	Use of "add fifo instance"	$_CCD_RPT_PATH /general/report.fifo
Statics		
a) Have all static inputs/register been reviewed		$_CCD_RPT_PATH /general/report.statics
Exceptions		
a) Are all existing blackboxes expected?		$_CCD_RPT_PATH /general/report.bbox
b) Are any modules checked up to the boundary only	skip_instance used	$_CCD_RPT_PATH /general/report.skip_instances
VERIFICATION		
Are there any undriven nets/pins in the design		$_CCD_RPT_PATH /general/report.floating_signals
Review design summary		$_CCD_RPT_PATH /general/report.design_summary
Have all rule filters been reviewed		$_CCD_RPT_PATH /cdc/report.filters
Check the following reports for Fails (after filtering)		

a) Are there any Fails in the summary report		$_CCD_RPT_PATH /cdc/report_cdc_summary.final
b) Are there any Structural CDC Failures		$_CCD_RPT_PATH /cdc/(report_cdc_summary.final /report.nofiltered.cdc)
c) Are there any Convergence Failures		$_CCD_RPT_PATH /cdc/(report_cdc_summary.final /report.nofiltered.convergence)
d) Are there any Set/Rst Failures		$_CCD_RPT_PATH /cdc/(report_cdc_summary.final/report.nofiltered.set_reset)
e) Are there any Set/Rst Sync issues		$_CCD_RPT_PATH /cdc/(report_cdc_summary.final/report.nofiltered.sr_sync_cross)
f) Are there any modelling issues		$_CCD_RPT_PATH /modelling/report.nofiltered.modeling
Have all filters been clearly documented		$_CCD_INPUT_PATH/ccd_filters.do
Has a correct initialization file been used to get the design into a functional mode of operation for functional checking?		$_CCD_INPUT_PATH/init.seq

Table 26 CCD-CDC reports

11.3.5. Interesting Reading

cummings_FIFO_Design.pdf

Application note on handling of set/reset checks in CCD: Handling Asynchronous Resets in CCD

11.4. Lint Checks

11.4.1. Overview

Conformal Constraint Designer (CCD) is used during development and sign off to reduce the risk of downstream tools misinterpreting the HDL.

It is far easier to find lint style errors with HAL/CCD as opposed to resolving in simulation. It is common for Lint checks to be left until RTL freeze but this can result in re-work and difficulty in resolving issues if other RTL has been designed around an existing lint issue. It is therefore recommended that CCD rule checks be run on a regular basis throughout development. This should lead to less issues during the release phase.

11.4.2. Refined Rules

Lint checks are notorious for the 'noise' generated in reports which can make it difficult to zone in on real issues. A refined ruleset is used in CCD and AFL checks. Local amendments to these rules are discouraged although refinements may be necessary due to coding styles. Any updates to the rules file should be declared at sign off.

The current refined rule set is located within the top level of the delivered scripts tarball under the ccd directory e.g. ccd_lint_refined_rules_<date>.txt and is set in ccd/ccd_lint.setup.

11.4.3. Notes

It is recommended that required filters, initialisation files, etc. should be located under the cfg/ccd directory.

You can enter GUI mode from the command line by typing "*set gui*" from vpxmode.

To toggle between tclmode and vpxmode simply type "*tclmode*" or "*vpxmode*".

Most tclmode commands need a "_" where a space exists.

11.4.4. Checking results

The following reports will be generated and should be reviewed. See /scripts/setup/setup_dirs for report locations. Stork will also perform this analysis.

Check	Notes	Report
FLOW		
Is a refined ruleset being used?		$_CCD_RPT_PATH /general/$DESIGN.log
a) If yes, which version?		
Are all existing blackboxes expected?		$_CCD_RPT_PATH /general/report.bbox
VERIFICATION		
Are there any errrors in the Logfile		$_CCD_RPT_PATH /general/$DESIGN.log
Are all clocks defined?		$_CCD_RPT_PATH /general/report.clocks
Have clocks been correctly grouped?		$_CCD_RPT_PATH /general/report.clocks
Are there any LINT Errors/Warnings (not filtered)?	Fix Errors / Fix or Filter Warnings	$_CCD_RPT_PATH /lint/(report.lint.all / report_lint.nofiltered.fail)
Have all LINT rule filters been reviewed and are clearly documented?		$_CCD_INPUT_PATH/ccd_filters.do

Table 27 CCD-Lint reports

11.5. Filters

Filters can be added for any failing check via the GUI or using a file. In previous versions of the RDF scripts there were individual filter files for lint, sdc and cdc checks. From RDF4.0, these filter files can

still be used but a collated file will be generated. This collated file, ccd_filters.do, is created automatically at the end of the cdc.do script. Once this collated file has been created then only this filter file is required and the individual filter files, “filters_*.do” files must be deleted. If new filters have been added during an interactive run then use “write_filters” on the command line to write the filters to the existing filter file which is defined by “$_CCD_INPUT_PATH/ccd_filters.do”

11.5.1. CDC Filters

From CCD version 15.1 there is a new method to add and automatically create pre-filters from post-filters. Pre-filters are much more efficient for CCD.

There are two types of filters: pre and post.

a) Post Filtering: See “add rule filter” command in the CCD reference manual.
   i) These can be added via the GUI by right clicking on the failing occurrence and selecting “Filter Occurrence”-> “With Description”
   Add a relevant description with sufficient detail
   By using the “write_filters” tcl proc which executes the CCD command “write_rule_filter” this will generate the collated filters file, ccd_filters.do, which will have pre-filters that have been constructed from the post filters, add rule filter, command. The generated pre-filters are what CCD will apply in future runs and the post filters are retained for documentation purposes only since they contain the filter description.

b) Pre-filtering: See "filter_paths" attribute in the CCD reference manual. There are examples in the documentation and in the supplied in scripts/ccd/filters_cdc.do.
   i) Modules can be excluded from specific rule_instances using the attribute "exclude_module"
   ii) Atomic checks can be excluded by using the attribute "exclude_atomic_checks"

Below is an example of using filter_paths for convergence check filtering.

```
# Gray code synchronizers converge to generate FIFO write status/control is ok as only one bit will change at a time
set_attribute -add_one [find -ruleinst cdc_def_rs/conv_checks/convergence_in_sys_clk] filter_paths \
[list from [find -instance DEPACKER_FAST_MODE_RESYNC_ASYNC_FIFO/rdptr_grey_resync_gen*.rdptr_grey_resync/signal_in_q_reg] \
to [find -instance {DEPACKER_FAST_MODE_RESYNC_ASYNC_FIFO/full_reg \
SDI_CORE0/DEPACKER_FAST_MODE_RESYNC/DEPACKER_FAST_MODE_RESYNC_ASYNC_FIFO/wr_usedw_reg\[4\]}]]
```

NOTES:

- Always try to provide "from" and "to" points in the filter to constrain the filter as much as possible.
- Use the "-add_one" to allow simpler "filter_paths" definitions as the attribute is not additive unless this switch is used.
- The "to" point is "convergence end point" in the CDC report
- Always provide a sufficient explanation as customers may be provided the filters file.

11.5.2. LINT/SDC Filters

The application and re-use of filters is notoriously tricky within CCD. Here are a few tips that may help make the application of filters and the ability to reuse them easier.

1. Any module that has parameters will have its module name described with these parameters. The parameters part can be wild carded to make the filter portable for other configurations. The risk associated with such application must be considered. For example:
   -message “In line 554, file '../rtl/depacker_core.v' (depacker_core_PIXEL_OUT_NB2_PACKED_OUT_NB2_RAM_SIZE1408_CMD_FIFO_DEPTH2)
   could be reduced to:
   -message “In line 554, file '../rtl/depacker_core.v' (depacker_core_*)
2. The line number may change with updates to the code so it is sometimes valid to wildcard the line number depending on the type of error. Care must be taken not to make the filter global if there is a danger that it will remove other checks elsewhere in that file.
3. Always add a description to the filter for future reference.
4. Filters are generally applied in tclmode which means that special characters must be escaped with the “/” character.
5. Use wildcards where possible.

12. Jasper AFL

Jasper AFL implements both HAL and Structural Property Synthesis (SPS) checks. These check for common design errors and LINT issues in RTL.

12.1. Refined Rules

A refined IPG rule set which has been developed from an initial set obtained from HAL R&D and can be obtained by checking out the last tagged svn release. The latest date stamped rule set should be used. For example, "hal_important_rules_**<latest datastamp>**.def". DO NOT MODIFY THIS FILE.

12.2. Updating Refined Rules Set

Please provide feedback or requests for rule refinement through JIRA.

Requests for rule changes are captured in this document along with outcomes of that request: HAL rules change request document

12.3. Using other Rule sets

Other rule files, such as customer rules, can be run using the “-ruleset” and “-check” switches on the run_afl.csh command line.

12.4. Training slides

HAL Training slides

Jasper AFL Training

12.5. Jasper AFL

12.5.1. Running Jasper AFL with supplied scripts

There are 3 stages to run when using run_afl.csh

1) run_afl.csh -clean
   a) This will clear all local files and directories used by run_afl.csh including liblist.v.
   b) The expanded.f will NOT be cleaned out. NOTE: manually remove this file if a new expanded.f is to be generated at the setup stage.

2) run_afl.csh –setup
   a) Local file, expanded.f will be generated if it doesn’t already exist. This is not relevant if you don’t use expanded.f as your primary filelist for the RDF.
   b) A template afl_setup.tcl file will be copied to the “_AFL_INPUT_PATH as specified in setup_dirs.tcl which the user should modify to include the design clocks and resets. This file be used to drive AFL.
      NOTE: Clocks and resets only need to be defined for advanced lint. Basic lint does not require the clocks and resets to be set up.

   c) Handling Blackboxes. There are various methods that can be used to handle blackboxing in AFL. Following are some methods that have been tested in this release:

i) Case when there is a missing module description. For example, the module only has a liberty (.lib) representation.
   (1) Use –bblist on the run_afl.csh command line and specify an empty file
ii) Case when all module descriptions are available and particular modules should not be analysed
   (1) Use –bb_list <file> on the run_afl.csh command line, where file is a list of module names only OR
   (2) Use –bblist <file> on the run_afl.csh command line, where the file contains a list of hierarchical paths and instance names to be blackboxed. Mind that the module of this instance need also to be captured in a separate file as an empty module (with ports descriptions, only).

NOTE: If the "BLACKBOXES" variable is set in setup_project.csh then a file, afl_bbox_list, will be generated in the path specified in setup_dirs.tcl by the "_AFL_INPUT_PATH" variable. This will contain a list of modules that are specified in setup_project.csh. This can then be modified as per the above description with modules or instances as it will only reflect the contents specified in setup_project.csh.

12.5.2. Running Jasper Testmode DFT checks

DFT checks are split into 2 sets:

1. DFT_FUNCTIONAL
2. DFT_TESTMODE → Only required when there is existing test related logic in the RTL. Additional information is required to run these checks and this can be provided using the "design_info" option on the run_afl.csh command line.
   See the AFL documentation for design_info contents and syntax. Example syntax can be found in : scripts/afl/example_design_info.txt

3) run_afl.csh – run -gui
   a) This will execute the lint checks in GUI mode.

Switch on run_afl.csh	Function
-h/help	Displays information on available switches
-clean	Remove all temp files, logs and compiled libraries
-run	Run LINT analysis
-run_adv	Run SPS checks
-setup_adv	Sets the environment for the advanced lint checks but does not run any checks. The checks are manually selected when the GUI is brought up
-compile_lib	Compile the Verilog gate library if any preserved cells are used in the design. Use with –run option
-gui	Open GUI to check run results
-rulefile <def file>	Use supplied rules (def) file
-check	Select category to run within rulefile
<any other AFL/IRUN commands>	Passed directory to irun command line. e.g. –design_info design_info.txt

Table 28 run_afl.csh switches

It should be noted that running the advanced lint checks "-run_adv" switch takes a considerable amount of time due to the various checks that the tool has to go through. Creating the switch

“setup_adv” helps in reducing the run time as the user can manually select the checks needed for a particular purpose. For example, deadlock and livelock in FMS checks only, could be executed. The run time of the check will vary depending on the rules selected.

If the “-setup_adv” switch is selected, the following steps should be used in order to run any checks once the GUI is brought up.

Step 1: Click on “configure AFL checks” as highlighted below. This brings up another window as shown in figure 5

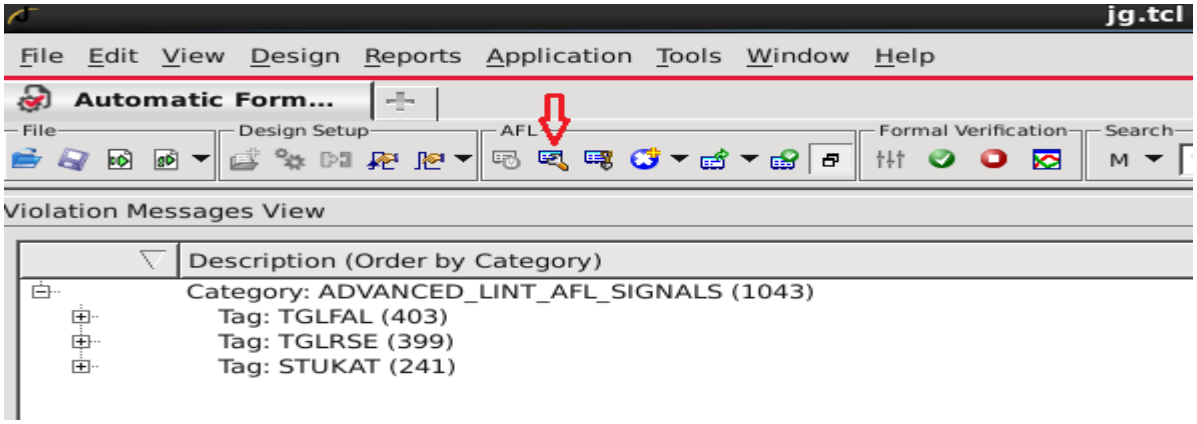


Figure 4 Configure AFL Checks

Step2: Manually select the various rules to run under the three categories as shown.

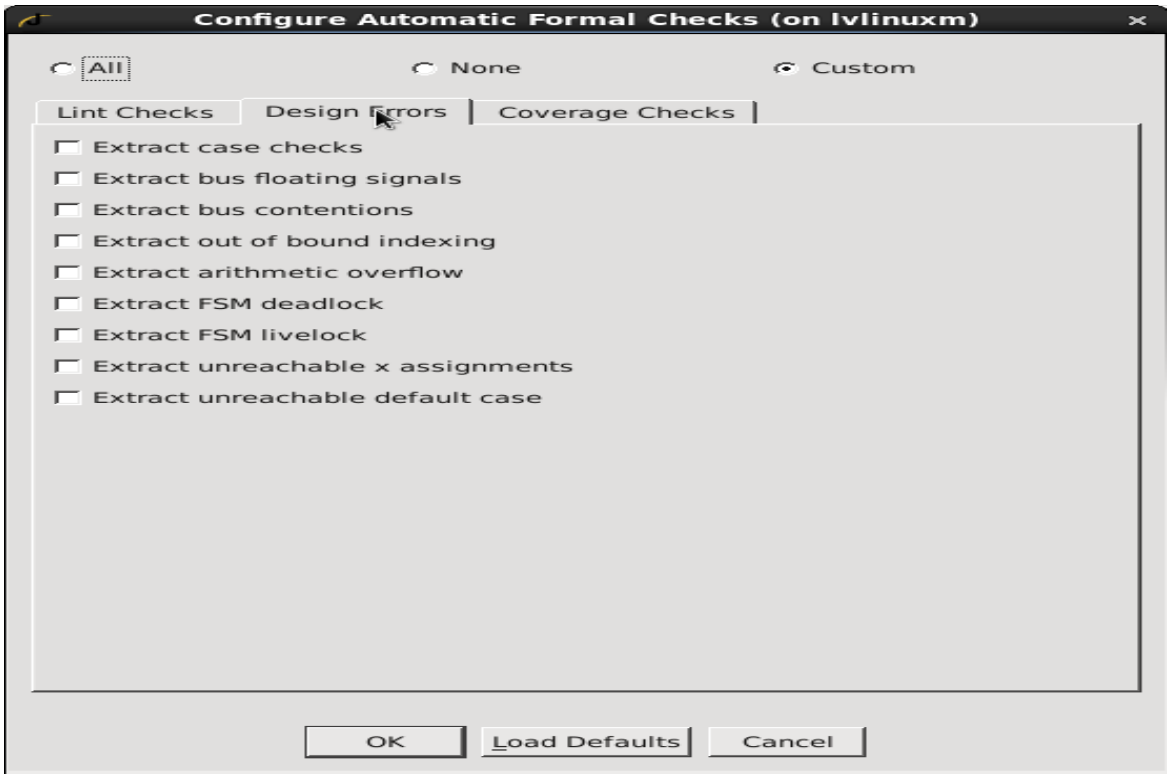


Figure 5 configure Automatic Formal checks

Step 3: Extract AFL checks by clicking on the button indicated below. It is recommended to check in the console window if the AFL checks have actually been extracted.

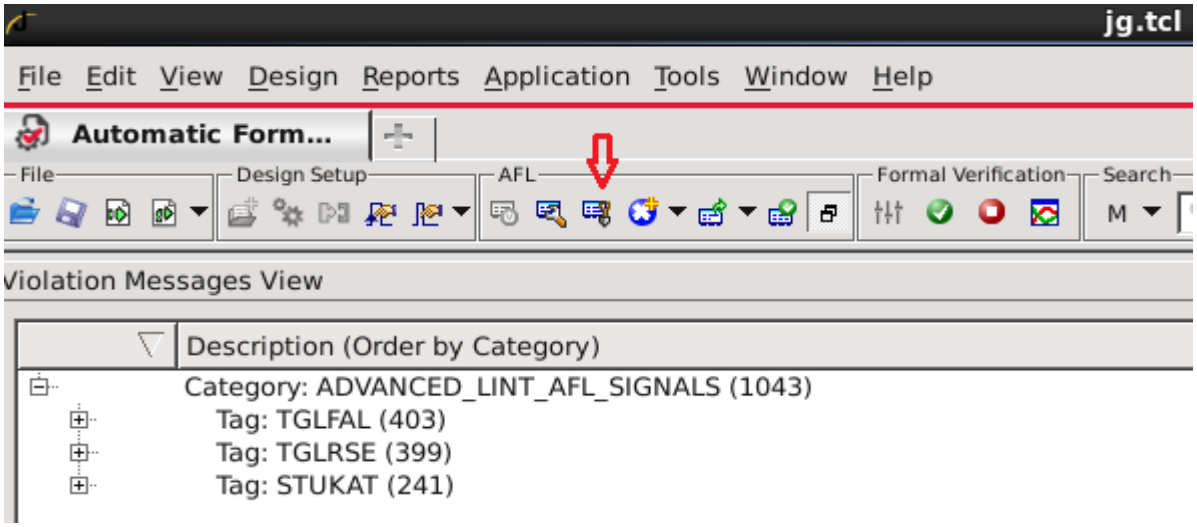


Figure 6 Extract AFL checks

Step 4: Export to task. This is done by clicking on the drop down menu shown below and select "Export to task" which brings up another window as shown in figure 8

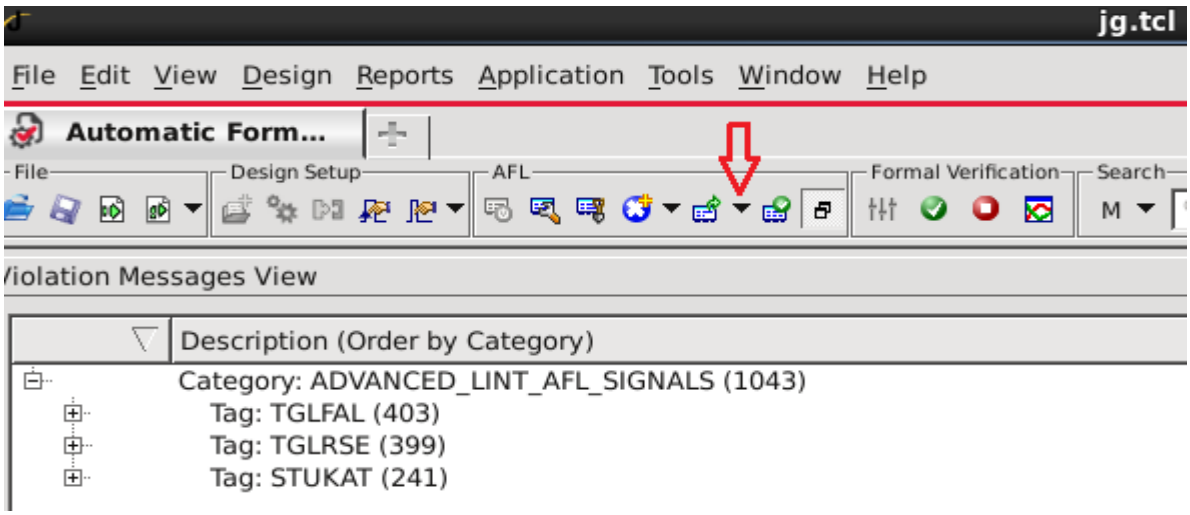


Figure 7 Export to Task 1

Step 5: Click on "OK"

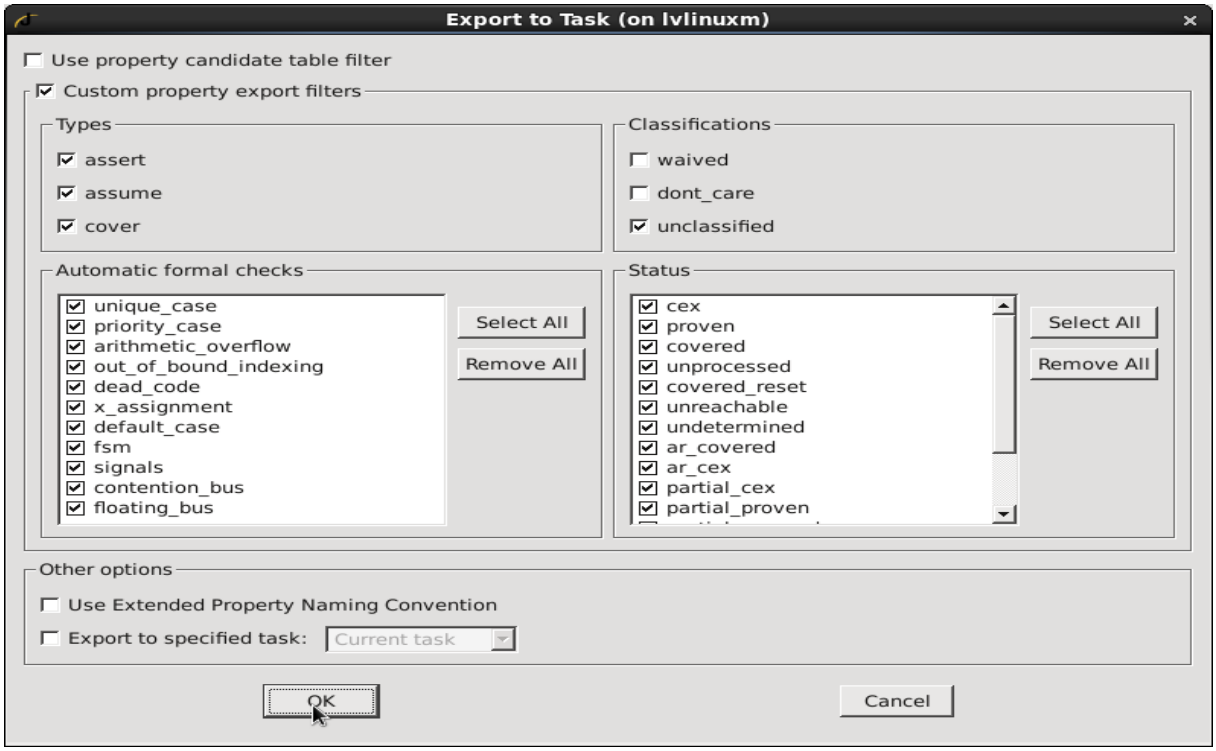


Figure 8 Export to Task 2

Step 6: AFL approved exported; this sets the test to run with the various rules selected.

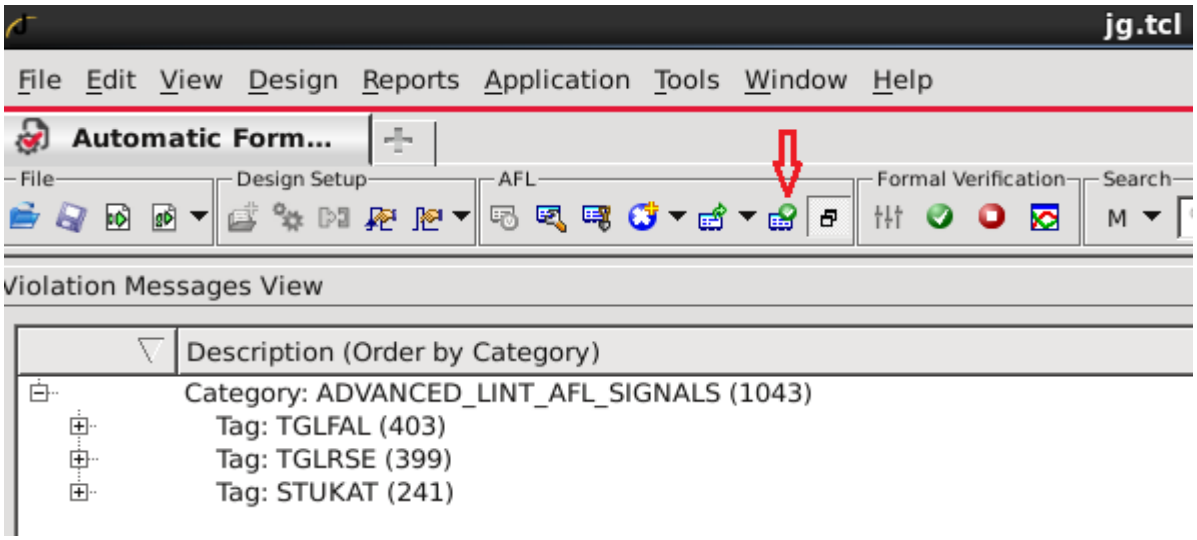


Figure 9 AFL approved exported

Step7: Review the results and apply waivers where applicable.

NOTE: Any unrecognized commands will be passed directly to the IRUN command line.

A HAL design information file can be used to provide additional information for HAL execution if required by passing using: -design_info <hal_design_info file>

See HAL documentation on its use.

12.5.3. Waivers

All errors must be fixed and warnings must be fixed/filtered and notes reviewed. Filters can be applied by a right-click on the issue and selecting “Waiver”.

You should add a description of why the warning is ok in these reports. Use the following syntax to make is traceable:

WAIVER DESCRIPTION: Warning is not an issue due to.....

Example:
ULCMPE
// FILTER DESCRIPTION: <detailed explanation of why this can be filtered>
*W, (/projects/kessel/work/mlewis/mipi_csi2rx_isp/rtl/csi2rx_protocol.v,614): Unequal length operands in equality operator encountered (padding produces incorrect result) in module/design-unit CSI2RX_PROTOCOL. LHS operand is 15 bits, RHS operand is 14 bits.

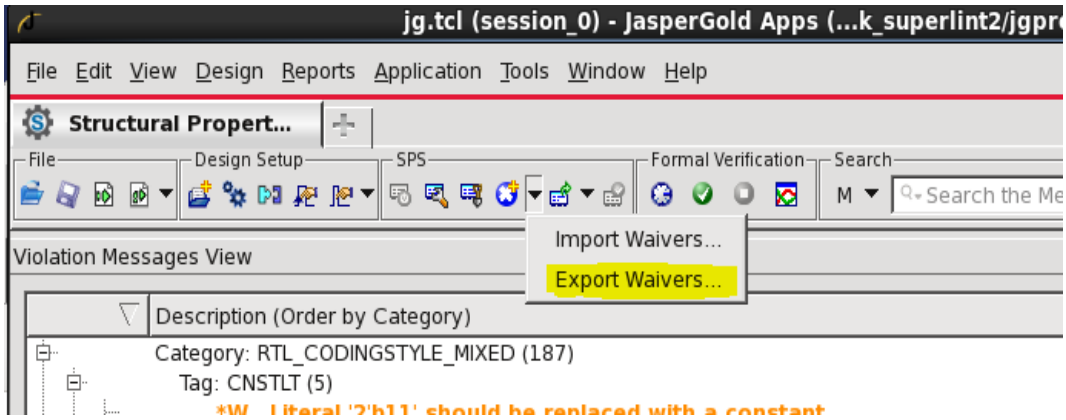
Waivers can be saved for future use by “export” tab on the GUI or by using the command line:

check_afl_waiver –export –file_name <file>

Similarly, existing waiver files can be imported using the GUI tab or the command line:

check_afl_waiver –import –file_name <file>

A tcl routine is also available in afl_setup.tcl which will write out a waiver file. Use “write_waivers” to write the waivers out to a file.



Waivers files stored in the following file DUT_PATH/cfg/<CONFIG>/afl/jlint_waivers.txt will be read automatically via the afl_setup.tcl file.

12.5.4. Reports

Various reports are generated using commands in afl_setup.tcl including html, text and xml. These will be stored in the directory specified in setup_dirs.tcl by the "_AFL_PATH" variable.

A tcl routine has been added to afl_setup.tcl which will write out the required reports. Use "write_reports" to write out these files.

12.5.1. Checking results

The following reports will be generated and should be reviewed. See /scripts/setup/setup_dirs for report locations. Stork will also perform this analysis.

Check	Notes	Report
3 reports will be generated		
Check Errors/Warnings	HTML report	$ _AFL_PATH/jlint_violations.htm
Check Errors/Warnings	Text report	$ _AFL_PATH/jlint_violations.txt
Check Errors/Warnings	XML report	$ _AFL_PATH/jlint_violations.xml
	Waiver file	${_AFL_INPUT_PATH}/jlint_waivers.txt

Table 29 Jasper AFL reports

13. Atrenta Spyglass - Atrenta/TSMC Kits

13.1. Overview

Atrenta Spyglass IPKit provides RTL quality sign-off criteria that must be used at sign-off and at certain project milestones.

On occasion, interactive debug will be required and this will consume a license so this effort must be kept to a minimum but is necessary particularly for CDC analysis. Please try to make use of the offline debug capability where possible. This is reasonably intuitive when using the html dashboard reports.

A useful overview of the IPKit can be found in the installation directory: /grid/scp/hw/tools/atrenta/atrenta_ip_kit_*/docs/Atrenta_IP_Kit_Quick_Start_Guide.pdf

All IPs should be verified with the Atrenta IPKit and also with the TSMC Kit if the design is to be registered with the TSMC 9000 Soft IP program. These 2 kits are very similar so once clean in the Atrenta IPKit the TSMC sign off will be trivial.

13.2. Tool Setup

Use "*module avail atrenta*" to view the available versions of Spyglass and IP Kits

To access the tool use "*module load atrenta/spyglass/<version>*"

To access the IPKit use "*module load atrenta/ipkit/<version>* "

TSMC Kit can be found here if it is not already installed on your network: TSMC 3.0 Kit

You can find tarballs of Spyglass and IPKits here: **/it/vlsjdata/bhavin/downloads/Atrenta (located in San Jose)** or contact bhavin@cadence.com or mlewis@cadence.com who can download later versions of the tool.

When new installations become available they will be installed here: "/grid/scp/hw/tools/atrenta" which gets mirrored to various sites including Noida, Livingston and Poland.

New installations will be required for VCAD chambers (Cary and Columbia).

When installing a new version of the tool please refer to this helpdesk ticket for some gotchas (# 03367277). The following permissions need updated:

chmod 755 <Spyglass install directory>/SPYGLASS_HOME/auxi/tkgui-new/new_db_package/

13.3. Licenses

Currently, there are 8 licenses (1 dedicated to Tensilica) available within Cadence so the tools usage guidelines must be adhered to in order to ensure the most efficient and effective use of the available licenses.

As there are only 7 licenses available it's likely that they may all be in use. You can check license status using this command:

lmstat -c 2710@sjtplic -a

13.4. Scripts

A set of Spyglass template scripts have been developed to help get started and are located here within the top level of the delivered scripts tarball.

A single script is available to control Spyglass, run_spyglass.csh. Use ./run_spyglass.csh -help for instructions and available switches

13.4.1. Switches

There are a number of switches available for the user:

Switch on run_spyglass.csh	Function
-h/help	Displays information on available switches
-waiver <filename.swl>	Include a spyglass waiver file (swl)
-activity_file <filename.xx>	Simulation activity file (e.g. VCD)
-sgdc <filename.sgdc>	Include an existing Spyglass constraints file (see notes below)
-power_file <filename>	Power Intent file (CPF/UPF)
-setup	Clean setup. Regenerates sg_setup directory
-read	Performs read of design and SDC and runs basic setup goals for dft, sdc and cdc.
-nosv	By default SystemVerilog support is enabled (use with -read). Use this switch with a new setup.
-showgoals	Lists the available goals within the installed IPKit
-run_goal <selection>	Pass selection directly to aipk_run –goal <selection.
-outdir <directory>	Select a directory for output files.
-phy_target <library name>	If not defined then the first library from ./liblist.f will be used.
-run_flow_signoff	Atrenta IPKit – RTLDesignFlow specified sign-off goals
-run_flow_initial	Atrenta IPKit – RTLDesignFlow recommended setup goals (run and clean prior to any other goals)
-gui	Open GUI for interactive debug.
-pack	A tarball will be created in the run directory. This should be stored in a suitable location such as /cfg/<config>/spyglass/ for future reference. This will also record waivers and SGDC files for future runs.

Table 30 run_spyglass.csh switches

NOTE: “-SGDC usage”

- This switch should only be used to include an existing SGDC that has been **proven** to work.
- If an existing sg_setup with SGDC exists
- It can be run at 2 different stages:
    - It can be used when an existing SGDC exists within the sg_setup directory
        - In this case the only change that will take place is that and “include <file>,sgdc will be added to the existing sgdc file.
    - It can be used when there is no SGDC existing i.e. new setup

        - In this case the only change that will take place is that and “include <file>,sgdc will be added to the existing sgdc file and no SDC2SGDC translation will take place.
    - If you are taking on constraints that have not been proven then this could cause the existing SGDC within sg_setup to be corrupted. A back up will be saved (.sgdc.bak) within the sg_setup directory which can be used to recover. It is recommended to test constraints in sg_shell mode – See Spyglass documentation. Note sg_shell is a tcl interface.
    - The SGDC file supplied with the –sgdc switch should NOT be placed in existing sg_setup directory as the provided sgdc file will be used to generated the new sgdc file within sg_setup.

The –waiver switch can be used during the first ‘read’ process.

13.4.2. Blackboxes

Blackboxes can be added through the “BLACKBOXES” variable in setup_project.csh. Blackboxes can also be added to the Spyglass runs by specifying the “set_option stop <module>” command in the <top>.prj file.

The user can also prevent analysis of parts of the design by using the ip_block constraint (See Spyglass documentation for more details). One drawback is that reconvergence of output ports from the level specified with the ip_block constraint will not be checked.

13.5. Atrenta IPKit

The Atrenta IPKit provides a comprehensive set of goals that can be run. However, it only makes sense to run a subset of goals at various stages of RTL maturity.

13.5.1. Predefined groups of goals

Some groups of goals have been added to the run_spyglass.csh script. Use the "*run_spyglass -help*” to see the available groups supported with that particular release of the script.

The user can select any goal or make their own groups with the *"-run_goal*" option.

It is recommended that the user runs the "*-run_flow_initial*" goals before running any other goals. These goals must be clean before running further goals.

13.5.2. Flow stages:

There are 4 stages to running Spyglass. All available commands to the wrapper script can be see using this command:

./run_spyglass.csh –help

Stage 1) Generate the sg_setup directory.

./run_spyglass.csh -setup

Stage 2) Read in the design, constraints and generate SGDC file:

./run_spyglass.csh –read

Review the generated SGDC and refine as necessary. Do not run goals until SGDC has been reviewed and refined as necessary.

Stage 3) Run the required goals using:

a) *./run_spyglass.csh -run_<goal group>* **OR**
b) *./run_spyglass.csh -run_goal* <comma separated list of goals with no whitespace>

Stage 4) Package the reports

a) ./run_spyglass.csh –pack

Interactive debug is run using: ./run_spglass.csh -gui

13.6. TMSC Kit

Quick Start Guide: TSMC IP Handoff Kit Quick Start should be reviewed prior to running the TSMC Kit.

The TSMC and Atrenta IPKit are now closely tied. TSMC 3.0 kit now support the setup stage.

13.7. Hints and Tips

13.7.1. GUI vs batch Mode

Note that logfiles are stored in different directories depending on run mode. Suggest running always in batch mode and only using the gui for analysis.

13.7.2. Setup Stage

After reading in the design with. /run_spyglass -read please review the following:

- Check the <project>_input_abstract.sgdc which will highlight any unconstrained ports.
- Ensure that clock groups are correctly defined in the SGDC. Clock groups are not automatically translated by spyglass into the sgdc file and therefor need to be added manually into the file. Example syntax: -

clock_group -name group0 -clock { CLK1 CLK1_GENERATED_CLK }
clock_group -name group1 -clock { CLK2 }
clock_group -name group2 -clock { CLK3 }
clock_group -name group3 -clock { IO_ASYNC }

Where the name used in the -clock statement is the -tag name from the sgdc file for each clock

13.7.3. CDC violation reduction

The user can modify sg_setup/<design>/<design>_goals_setup.tcl and add “set_parameter fa_msgmode fail” to the cdc_verify section (This prevents Partially Proven assertions being reported – Ensure formal depth is sufficient in cdc_verify verification report).

13.7.4. Blackboxing

The user can prevent analysis on particular blocks

13.7.5. Constraints

Prior to running any goals always verify/update the contents of the spyglass constraints file (sg_setup/<project>/<project>.sgdc.

There are 2 main methods of constraining the design for clock domain analysis:

1. Constant or static registers or ports --> Use "quasi_static" in the sgdc
2. Signals that cross from very slow domains to fast clock --> Potential use of "cdc_false_path" in the sgdc.
3. Signals that are true false paths (no signal will cross it functionally) --> Use "cdc_false_path" in the sgdc.
   IMPORTANT: In all cases the constraint must be documented which sufficient information and a review of the Spyglass documentation is required to ensure the constraint being used is applicable to that situation.

It is important to apply Spyglass constraints instead of waivers where possible for CDC violations. See Spyglass documentation for available constraints.

Applying Spyglass constraints instead of waivers can prevent issues being hidden if the design changes. By using constraints the user has more control over the scope of what is affected. It is advisable to use scoped constraints where possible to:

1. Allow the constraint to be re-usable in other designs as scoped constraints work at the module level.
2. Reduce the text in the constraints file
3. Advantages when working at the IPS level where PHY and/or controller SGDC can be imported into the run to allow complete verification without the need to apply blackboxes. This will significantly reduce re-verification with other team's designs.
   NOTE: it is still valid to blackbox pre-verified components.

13.7.5.1. Scoped constraints examples:

The following quasi-static constraint can be reduced from this:

quasi_static -name "cdns_csi2rx_ips.i_cdns_csi2rx_phy.i_cdns_csi2rx_soc.i_cdns_csi2rx_top.i_cdns_csi2rx.i_csi2rx_ctrl_reg. stream_fcc_ctrl_fcc_en_r*""

to this:

quasi_static -name "csi2rx_ctrl_reg::stream_fcc_ctrl_fcc_en_r*"

Another example is coherency related constraints (convergence). Here is an example of constraining gray coded signals. This will help to reduce convergence violations.

The following can be reduced from this:

cdc_attribute -exclusive

cdns_csi2rx_ips.i_cdns_csi2rx_phy.i_cdns_csi2rx_soc.i_cdns_csi2rx_top.i_cdns_csi2rx.i_stream0.i_stream_fifo.u_pushptr_synch.meta[6] \

cdns_csi2rx_ips.i_cdns_csi2rx_phy.i_cdns_csi2rx_soc.i_cdns_csi2rx_top.i_cdns_csi2rx.i_stream0.i_stream_fifo.u_pushptr_synch.meta[5] \

cdns_csi2rx_ips.i_cdns_csi2rx_phy.i_cdns_csi2rx_soc.i_cdns_csi2rx_top.i_cdns_csi2rx.i_stream0.i_stream_fifo.u_pushptr_synch.meta[4] \

cdns_csi2rx_ips.i_cdns_csi2rx_phy.i_cdns_csi2rx_soc.i_cdns_csi2rx_top.i_cdns_csi2rx.i_stream0.i_stream_fifo.u_pushptr_synch.meta[3] \

cdns_csi2rx_ips.i_cdns_csi2rx_phy.i_cdns_csi2rx_soc.i_cdns_csi2rx_top.i_cdns_csi2rx.i_stream0.i_stream_fifo.u_pushptr_synch.meta[2] \

cdns_csi2rx_ips.i_cdns_csi2rx_phy.i_cdns_csi2rx_soc.i_cdns_csi2rx_top.i_cdns_csi2rx.i_stream0.i_stream_fifo.u_pushptr_synch.meta[1] \

cdns_csi2rx_ips.i_cdns_csi2rx_phy.i_cdns_csi2rx_soc.i_cdns_csi2rx_top.i_cdns_csi2rx.i_stream0.i_stream_fifo.u_pushptr_synch.meta[0]

to this:

cdc_attribute -exclusive

csi2rx_stream::i_stream_fifo.u_pushptr_synch.meta[6] \

csi2rx_stream::i_stream_fifo.u_pushptr_synch.meta[5] \

csi2rx_stream::i_stream_fifo.u_pushptr_synch.meta[4] \

csi2rx_stream::i_stream_fifo.u_pushptr_synch.meta[3] \

csi2rx_stream::i_stream_fifo.u_pushptr_synch.meta[2] \

csi2rx_stream::i_stream_fifo.u_pushptr_synch.meta[1] \

csi2rx_stream::i_stream_fifo.u_pushptr_synch.meta[0]

13.7.6. Run Stage

Select goals carefully and do NOT run the mandatory atrenta rule set by default.

13.7.7. Clock Domain Checks

- Please see here for guidance on mulit-mode SDC checks: **Error! Reference source not found.**
- run cdc_setup_check and work on the messages to ensure that the SGDC constraints are complete and consistent with the RTL
- Once this is done, the user then runs cdc_verify_struct, which adds the structural CDC checks to the analysis
- Once the structural CDC is clean, the user then runs cdc_verify which adds in the functional CDC checks. MAKE SURE ALL STRUCTURAL CHECKS ARE CLEAN BEFORE RUNNING THIS GOAL.
- CDC violations should NOT be waived and instead should be resolved through:

1. fixing SDC constraints
2. Updating sgdc file to include static registers
3. fixing the RTL

- cdc_verify will cause cdc_struct and cdc_setup to be re-run so there is no need to include these as individual goals
- If using the run_spyglass script you can use a separate file to define static registers and static inputs. These should be stored in the /scripts/statics directory and contain a list of static inputs and static registers. The static registers should NOT contain the top level module name. The statics filename should be matched to the particular configuration e.g. statics_<CONFIG>.

- You may see this violation "Setup_req01 ERROR N.A. 0 10 2 prescribed setup requirements violated "
  You can change the default numbers to a more reasonable number by adding the following to the sgdc file:

#The cdc_matrix_attributes constraint is used to set a limit for

#SpyGlass-CDC attributes during the SpyGlass-CDC setup stage.

See Setup_req01 rule

cdc_matrix_attributes -crossing_per_clock_pair_limit 1500

cdc_matrix_attributes -src_per_dest_limit 200

Depending on which limit is being exceeded. To check which limit and values, you can refer to cdc/cdc_setup_check/spyglass_reports/clock-reset/cdc_matrix.rpt or via the gui > Reports> More CDC Reports>cdc_matrix.

The run_spyglass.csh script will automatically add some commands into the sg_setup files to generate the cdc_matrix_report (summary) and a more comprehensive report to see the actual crossings to aid debug. These can be found here:

<reports_dir>/<design>_sg_reports/<design>_cdc_cdc_setup_check/cdc_matrix.rpt

<reports_dir>/<design>_sg_reports/<design>_cdc_cdc_setup_check/Ac_sync_group_detail.rpt

13.7.7.1. Functional CDC (Formal Checks)

Following is some user feedback that can improve runtime and debugging when running the formal based functional clock domain checks.

Enable the Multicore option by doing the following 2 things:

a) First make sure you submit the spyglass job to the farm with multi core option:

   $BSUB_CMD -n 2 -R"span[hosts=1]"

   Where 2 is the number of CPU's you want to reserve. 2 is normally sufficient.

b) Uncomment the following line in ./sg_setup/<design>"goals_setup.tcl":

#set_parameter fa_multicore

NOTE: Although this doesn't consume more than one license it can help runtime by running the analysis in parallel. **DO NOT** do this by default as only the formal runs can make use of the additional CPU.

Other useful advice:

1. If you are viewing the waveforms, the spyglass waveform viewer can be problematic. You can load the VCD's generated by spyglass direct into simvision.
2. If you are debugging the waves for formal checks and the signal you need isn't shown then rerun with the following parameter set… "*set_parameter fa_vcdfulltrace allnets*" in ./sg_setup/<design>"goals_setup.tcl"
3. If you continue to get Partially Proven, then you have a few options:

    a) Increase fa_atime until "depth" ./sg_setup/<design>"goals_setup.tcl") as reported in adv_cdc.rpt is deep enough that you are comfortable that a failure can't occur in that many edges.
    b) Since fa_atime impacts ALL assertions and you are most concerned with Ac_datahold01a, then disable the other formal rules until you get through with these so the runtime will be more manageable.

13.7.7.2. Lint

- Adv_lint will run deadcode analysis and should be used sporadically to long runtimes. Information from the initial run can be used for offline debug.

13.7.7.3. DFT related

Atspeed_06

These occur when the design has synchronizers in the RTL to ensure safe clock domain crossings in functional mode. These synchronizers do not do anything to ensure safe crossings during test mode so you see these reported as clock domain crossings during test mode e.g.

[WARNING] <path_count> domain crossings from clock domain <domain_name1> to all other clock domains are not blocked under capture atspeed mode

The recommendation here is the following: -

Apply a waiver to Atspeed_06 for Spyglass.

Include notes in the Design integration guide for the user as follows: -

- Option 1. The User could insert blocking logic to block the domain crossing during test
- Option 2. The User should timing close ALL clock paths during test mode therefore considers only one test clock domain.
- Option 3: Recommended approach, the User takes a test coverage hit and defines the clock domain crossings as false_paths in the constraints

The delivery should include full details of these paths

13.7.7.4. DfT Transition Test Coverage Targets

Why the targets are different for Spyglass Vs Internal targets from ATPG?

Firstly, Spyglass for dynamic coverage expects >95%. Remember this figure is looking at potentially achievable coverage on the RTL and not actual coverage. On a simple IP, this should be 100%. The ONLY reason, IPs should have less than this is typically due to synchronizers in the IP, where clock domain crossings are therefore not testable without additional consideration which should be handled at SoC level or on the customer side at any rate. (See Atspeed_06 above)

The RDF target is >90% “testmode” dynamic coverage on a scan inserted netlist. Due to test mode constraints, 100% will never be reachable on a scan inserted netlist.

13.7.7.5. Reducing run time

cdc_verify and adv_lint are the most time consuming goals typically adding hours to the runtime.

13.7.7.6. Offline Debug

You can use the designread and designrun dashboards for offline debug. e.g.

firefox <project>l/<project>_sg_reports/html_reports/dashboard.html

13.7.8. Waivers

Using wildcards in waivers will make them portable when running checks in different work areas. For example, when waiving checks the full pathname to the Spyglass reports file is usually used in the waiver. Applying a wildcard down to the area where the reports file is stored in a project can work well. See example below where the wildcard in this instance replaces “/projects/kessel/work/<user/tsmc_checks/temp/”

waive -rule "High_Fan03b" -msg "File '*csi2tx_topleveldual_pixel_150115_dual_pixel/csi2tx_toplevel/csi2tx_toplevel/constraints/sdc_redundancy/spyglass_reports/constraints/highFan03b_reportedNets' containing high fanout nets

13.7.9. Known Spyglass issues

Duplicate SGDC. You may see the warning "**checkCMD_duplicate03**". This is a known tool bug and can be resolved by setting the following environment variable: **setenv SGS_REGR_AVOID_DUPL_READ yes**

13.8. Deliverables

The following deliverables are defined as optional and are packaged up with the aipk_pack command.

- Datasheet Reports
- Dashboard Reports
- Waivers
- Spyglass constraints file

13.9. Interesting Reading

Training slides can be found on many aspects of the tool here: Atrenta Training Slides

14. Compilation Check

14.1.1. Overview

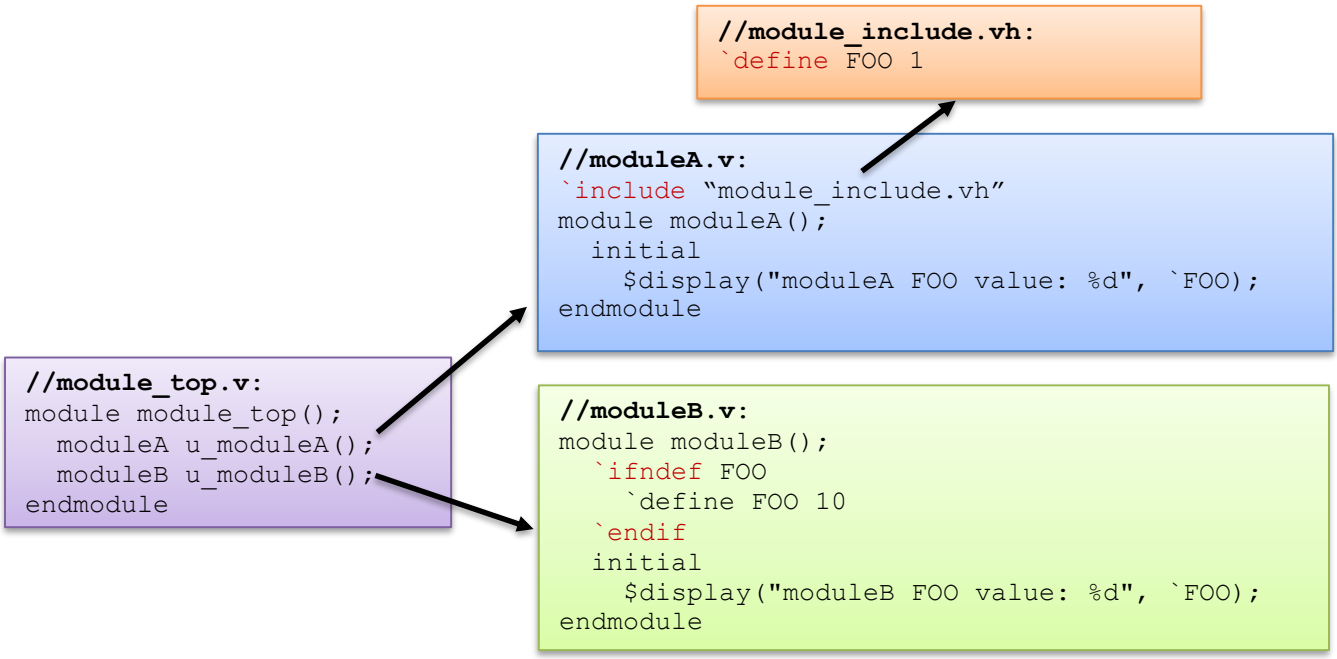
This check is aimed at verifying that the RTL sources are compiled and elaborated the same way using file-by-file and single step based approach. Different EDA tools (also 3rd party) may interpret badly written code differently depending on the compilation strategy. For sake of this document we define:

- single-step compilation – all sources are compiled together using a shared compilation process – used by e.g. Cadence irun/Spyglass/RTL Compiler
- file-based compilation – each source file is compiled in a separate compilation process – used by e.g. Synopsys DC

The `define (as any other compiler directives) are only valid (seen) for the current compilation process. In other words, macro defines are shared between source files in the single-step compilation. If the a source file does not include all definition of macros that are used within the file, the single-step may still generate snapshot due to the fact that the missing macro is defined in other source file that was compiled in the same process. This may lead to:

- errors during file-based compilation (for instance with DC)
- or mismatches between the elaborated design depending on the compilation strategy (EDA tools)

This is a very serious portability issue. The example below illustrate the latter problem.



In this example, the module_include.vh header file is included only in the moduleA but the FOO macro is used in both modules.

Using single-step compilation in irun

```
%> irun rtl/module_top.v \
    rtl/moduleA.v \
```

```
rtl/moduleB.v \
-incdir rtl
```

results in:

```
ncsim> run
moduleA FOO value:    1
moduleB FOO value:    1
```

Applying the file-based compilation (also chaining the compilation order with irun) results in

```
ncsim> run
moduleA FOO value:    1
moduleB FOO value:    10
```

In order to avoid mismatches due to compilation method, all macro defines have to specified within the scope of all files.

14.1.2. Requirements

The following requirements need to be met in order to successfully run the compilation_check.csh script:

- an appropriate incisive(e) module needs to be loaded
- the setup_project.csh should be visible in the directory where the compilation_check.csh is invoked
- the $RTL_F_FILE variable should point to an appropriate .f file that contains a complete list of source files and a complete list of include directories for the $DESIGN to be compiled

Note: Currently, the compilation_check.csh script supports the HDL_SEARCH_PATH csh variable set in setup_project.csh. However, it is recommended to place all necessary +incdir+ directives in the $RTL_F_FILE file.

Note: The .f file should not contain any +define directives. However, the compilation_check.csh supports command-line arguments which are passed directly to ncvlog/irun via +define+ command.

Example: If the user need to define GENERIC_TECH to instantiate generic RTL code for technology-specific instances in the design, the user specify the macro from the command line (without +define+ keyword)

UNIX> compilation_check.csh GENERIC_TECH

which results in the INFO message in the log file.

INFO: User-defined command-line params : +define+GENERIC_TECH

14.1.3. Flow

Here are the stages executed by the compilation_check.csh

1. Clean up the current working directory
2. Compile all the source files in a file-by-file manner with ncvlog
3. Decompile the $DESIGN snapshot into ncdc_3step directory with ncdc
4. Compile all source files in a single-step manner with irun
5. Decompile the $DESIGN snapshot into ncdc_irun directory with ncdc

6. Compare the ncdc_3step and ncdc_irun directories and issue an error message if differences are detected.

Error and any stage of the compilation_check.csh script breaks the flow and an appropriate ERROR message is printed – e.g.:

ERROR: ncvlog compilation failed
Check .../ncvlog.log

If differences between the decompiled directories are detected the following error messages is issued

ERROR: Files differ
Total number of different files: 20
Check .../compilation_compare.log for complete list

14.1.4. Debugging

The ncdc tool decompiles the snapshot into set of textual files that fallows the source file naming (without directory structure). Therefore, it is relatively easy to debug possible different lines using tools like meld or compare on entire

- ncdc_3step and
- ncdc_irun directories.

What is more, each file contains a pointer to the original source in its header.

15. IPS (Integrated Protocol Stack) Related Information

THIS SECTION IS WORK IN PROGRESS

15.1. Jasper AFL

Jasper AFL cannot read in Liberty files so any parts of the design that are represented by a .lib will have to be blackboxed using the -bblist or -bb_list IRUN switches as appropriate.

15.2. Conformal CD (CCD)

Conformal will read in Liberty and will make use of the timing arcs defined within to perform CDC checks.

15.3. Implementation (Synthesis, PnR, DfT)

15.4. Spyglass

There are 2 ways to handle parts of the design represented by a .lib or a part of the design that has been previously analyzed and where repeat analysis is not required (e.g. PHY level)

For CDC checks, the ip_block constraint can be used to only verify up to the interface of the PHY. This check will not prevent analysis for other Spyglass goals.

To prevent other goals from analyzing the PHY the user can blackbox using the “set_option stop <module>” attribute. If using this method and running CDC checks then additional constraints will be required to associate the clock and I/O relationships.

Appendix A: Multi-Mode CDC

A1 Overview

The set up process for clock domain checks for Spyglass and Conformal is based on the clock periods set in the chosen SDC file.

If a design has multiple modes whereby the clock relationships can change from mode to mode then the setup from a single SDC becomes insufficient to determine if the design will operate correctly with no clock domain violations.

It is critical that the design is analysed in each mode individually as currently Spyglass and Conformal will not accept multi-mode SDC constraints for clock domain analysis. This means that a separate SDC file is used in a clean run for both Conformal and Spyglass.

Following is an example of the issue:

SDC: ACLK=100MHz, BCLK=200MHz
A single cycle pulse from ACLK to BCLK is OK with these frequencies, but if BCLK has a mode where it can actually run at 50MHz, then this is a problem which won't be caught without additional clock domain checks for that mode.

Appendix B: Reporting Power, Performance and Area numbers

We want to have consistent reporting of product data to our internal and external customers. Please consider the guidance here when compiling PPA numbers.

The design flow has been constructed to replicate the design challenges our customers go through when getting to silicon; it considers the 'worst case'. When reporting our PPA numbers we should show our working, stating assumptions, library and corner used. It's a positive story to tell, and will convince customers our data is solid and based on realistic scenarios. In future we may produce a flow to show products under the best possible circumstances, this would be in addition to the current flow and bring us into line with how others report.

B1 How to generate Datasheet numbers

- In project.tcl of the RDF, set the switch DATASHEET_PPA to 1, this will override the following switches
    - set INSERT_SCAN 0 (default 1)
    - set CONNECT_CHAINS 0 (default 1)
    - set LEAKAGE_LIB LOW (default LOW)
    - set TYP_85C_CORNER 1 (default 0)
- It will also set the synthesis to use the typical library for optimisation

B2 Recommended Target Library and VT mix

- TSMC28HPC_9T and TSMC16FFPLUS are the current recommended target libraries
- The PPA reporting default will only use HVT cells

B3 Accounting for RAM in soft IP

- It is currently recommended that we consider any RAM as external to the IP.

B4 When to report power:

- A power report based on a post gate level simulation activity file with matching post PNR database is the preferred method for reporting power.
- The script rc_power.tcl performs this task.
- It can be time consuming to get gate level simulations, as such the second preferred option is to report power post synthesis based on an RTL simulation activity file
- If no activity file at all then please report the stochastic power numbers from synthesis
- The dynamic power from this report is generally very pessimistic so only rely on leakage numbers

B5 Reporting from Synthesis

B5.1 Power

Power reporting should be based on a switching file generated by simulation of the design. Either TCF or SAIF is acceptable.

- The power report from synthesis reports the power in 4 corners (WC/BC/TYP/TYP_85C)
- If the design has an RTL based activity file
    - Define RTL_TOGGLE_FILE setting in project.tcl of RDF

- The dynamic power report after an RTL/tcf based synth is within 10% of the final power report using a gate level TCF at EDI. (<7% correlation for 28nm, <8% correlation for 40nm)

Refer to these reports from synthesis for power numbers

- ${_REPORTS_PATH}/${DESIGN}_${activity}_TYP.power.rpt)
- ${_REPORTS_PATH}/${DESIGN}_power.html (All 3 corners reported)

If the design does not have an RTL based activity file:

- The dynamic power reporting based on tool defaults is generally pessimistic and is only good for a comparative study. It is not recommended to use default activity numbers in the datasheet.

Refer to these reports from synthesis for power numbers

${_REPORTS_PATH}/${DESIGN}_rtl.power.rpt (initial rtl power report)

${_REPORTS_PATH}/${DESIGN}_stochastic.power.rpt (post synth)

B5.2 Performance (Timing)

- It is not necessary to over constrain clock frequencies to provide a margin. The flow uses a qrcTechFile for parasitic information which provides a close correlation to final sign off timing.
- Clock uncertainty for hold and setup should be applied in the SDC file to account for jitter/eventual clock skew. (see SDC template)

Refer to this report from synthesis to verify timing

$_REPORTS_PATH/${DESIGN}_qor.rpt

B5.3 Area

Cell Area

- It is important to report the *cell* area only, not the total area. The cell area is the footprint of the cells at 100% utilization, the total area includes the estimated routing area and is not the footprint area.

Refer to this report from synthesis for cell area

$_REPORTS_PATH/${DESIGN}_area.rpt

NAND2 equivalent gate count

- It is recommended to use the ND2D1BWP35 cell when calculating the NAND2 equivalent gate count.
- This is pessimistic in relation to using a ND2XD1BWP35, but using the smallest NAND2 cell provides for easier comparison between libraries.

B5.4 NAND2 Cell sizes:

	ND2D1	ND2D2
16FF	0.207um^2	0.311 um^2
28 HPC	0.378um^2	0.630 um^2

B1 Reporting from EDI (place or ccopt)

B1.1 Power

Power reporting should be based on a switching file either TCF or SAIF format provided by simulation of the design.

- Do not report the incidental dynamic power numbers from EDI (post place/post ccopt). They are generally very pessimistic and have little relation to a TCF based analysis.
- Perform gate level sims to generate a tcf file at the end of the flow and report dynamic power numbers again for all 3 corners (WC, BC, TYP)
- Use the rc_power.tcl script for this
- WC reports the most optimistic power.
- BC reports the most pessimistic power
- Recommend using the TYP corner power reports

B1.2 Performance (Timing)

- See above for notes on QRC tech files/over constraining.
- After CCOPT the clock uncertainty should be reduced to reflect only clock jitter, clock skew is now in the design and accounted for.

Refer to these reports from EDI to verify timing

Post place

${_PNR_RVW_PATH}/$DESIGN.post_PlaceOptTimingRpts

Post ccopt

${_PNR_RVW_PATH}/PostCCOPTOptTimingRpts

B1.3 Area

Placement Area

- Post placement or CCOPT area as reported by EDI represents an estimated area required for routing with approximately 75% utilization. If reporting the area at this stage please note the utilization.

Refer to these reports from EDI to show area

Post place

${_PNR_RPTS_PATH}/$DESIGN.placed.gateCount

Post ccopt

${_PNR_RPTS_PATH}/$DESIGN.PostCCOPTOpt.gateCount