Design Compiler and IC Compiler Physical Guidance Technology Application Note

Version G-2012.06, June 2012

SYNOPSYS®

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AEON, AMPS, ARC, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, CODE V, CoMET, Confirma, Design Compiler, DesignSphere, DesignWare, Eclypse, Formality, Galaxy Custom Designer, Global Synthesis, HAPS, HapsTrack, HDL Analyst, HSIM, HSPICE, Identify, Leda, LightTools, MAST, MaVeric, METeor, ModelTools, NanoSim, NOVeA, OpenVera, ORA, PathMill, Physical Compiler, PrimeTime, SCOPE, SiVL, SNUG, SolvNet, Sonic Focus, STAR Memory System, SVP Café, Syndicated, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, UMRBus, VCS, Vera, and YIELDExplorer are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, ASAP, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Cosmos, CosmosLE, CosmosScope, CRITIC, Custom WaveView, CustomExplorer, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Encore, EPIC, Galaxy, HANEX, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM^{plus}, i-Virtual Stepper, IC Compiler, IICE, in-Sync, iN-Tandem, Intelli, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Macro-PLUS, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, ORAengineering, Physical Analyst, Planet, Planet-PL, Platform Architect, Polaris, Power Compiler, Processor Designer, Raphael, RippledMixer, Saturn, Scirocco, Scirocco-i, SiWare, SPW, Star-RCXT, Star-SimXT, StarRC, Synphony Model System Compiler, System Compiler, System Designer, System Studio, Taurus, TotalRecall, TSUPREM-4, VCSi, VHDL Compiler, VMC, and Worksheet Buffer are trademarks of Synopsys, Inc.

Service Marks (^{SM})

MAP-in and TAP-in are service marks of Synopsys, Inc.

Third Party Trademark Acknowledgments

SystemC is a trademark of the Open SystemC Initiative and is used under license.
ARM and AMBA are registered trademarks of ARM Limited.
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.
Entrust is a registered trademark of Entrust, Inc. in the United States and in certain other countries. In Canada, Entrust is a trademark or registered trademark of Entrust Technologies Limited. Used by Entrust.net Inc. under license.
All other product or company names may be trademarks of their respective owners.

Synopsys, Inc.
700 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

Physical Guidance Technology in Design Compiler and IC Compiler

Design Compiler topographical synthesis performs placement-driven design mapping and optimization in order to achieve high quality of results (QoR), tight correlation between Design Compiler and IC Compiler, and improved routability. Design Compiler does not require a complete floorplan for synthesis; however, correlation between Design Compiler and IC Compiler can be improved when both tools use the same set of physical constraints.

For large designs at 45 nm and below with complex floorplan and design goals, achieving good QoR and correlation between Design Compiler and IC Compiler can be a challenge even with a good netlist generated by Design Compiler placement-aware synthesis as a starting point. The handoff from Design Compiler to IC Compiler is a netlist and floorplan that provides good correlation between the tools. With the physical guidance flow, the handoff to IC Compiler also includes the cell placement information and results in correlation within 5%.

In the physical guidance flow, Design Compiler performs further refined placement that is consistent with the IC Compiler `place_opt` command functionality and uses enhanced post-placement delay optimization in order to provide a better optimized starting point for physical implementation. The placement information is passed to the IC Compiler `place_opt` command and is used as seed placement to guide physical implementation of the design. Thus, placement is aligned between Design Compiler and IC Compiler, resulting in improved runtime, QoR, and correlation.

Physical guidance significantly speeds up IC Compiler runtime because IC Compiler reuses the physical guidance information from Design Compiler topographical as its starting point and therefore goes through fewer placement steps.

In addition, the physical guidance flow provides faster design closure for challenging designs. You should expect to see area and timing correlation within 5% between Design Compiler and IC Compiler. You should also see an average speedup in the IC Compiler `place_opt` runtime of 30% while the impact in Design Compiler during synthesis is about a 10% additional runtime. You should generally see improvement in the overall design turnaround time.

Flow Overview

The physical guidance flow is jointly supported in Design Compiler and IC Compiler in order to improve runtime and correlation. This section primarily focuses on the user interface and flow requirements for the physical guidance flow.

You enable the physical guidance flow by using the `-spg` option with the `compile_ultra` command in Design Compiler and by using the `-spg` option with the `place_opt` command in IC Compiler. The `-spg` option works seamlessly with the

existing `compile_ultra` options in Design Compiler and with the `place_opt` options in IC Compiler. Physical guidance performs the following functions:

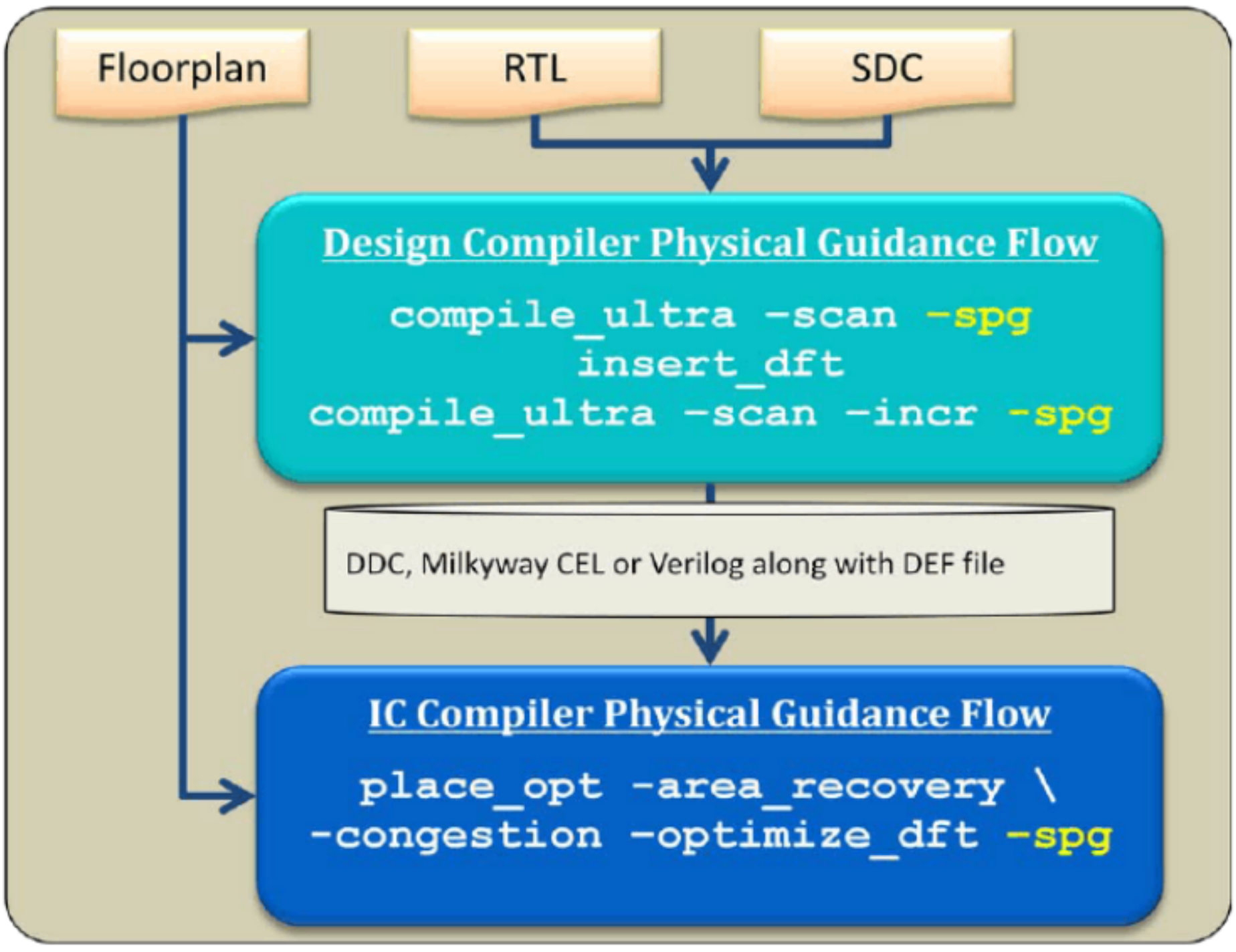
- In Design Compiler, the `-spg` option provides further refined placement that is consistent with the `place_opt` command in IC Compiler, performs enhanced post-placement delay optimization to provide a better optimized starting netlist for physical implementation, and optimizes the design for congestion to improve routability.

  You can save the placement information derived by Design Compiler in binary format in a .ddc file or a Milkyway CEL view, or you can save it in a .def (design exchange format) file. IC Compiler uses this placement information when you run the `place_opt -spg` command during physical implementation.

- In IC Compiler, the `-spg` option enables the `place_opt` command to use the Design Compiler placement as its seed placement. The `place_opt` command is enhanced to use topographical mode placement as the starting point in physical guidance mode.

When you use the physical guidance flow, there is minimal impact to your existing design implementation flow, as shown in the following flow example.

Figure 1: Synopsys Physical Guidance Flow Example



You should visit the Reference Methodology Retrieval System, RMgen, to configure and download a complete set of scripts that implement the recommended synthesis flow in Design Compiler and the physical implementation flow in IC Compiler. See the "Reference Methodology Scripts" section for more information.

Use the following checklist as a guideline for maintaining good QoR and correlation between Design Compiler and IC Compiler:

- Use consistent design and tool setup between Design Compiler topographical mode and IC Compiler. That is, you should use the same sets of logic and physical libraries to drive both tools and use consistent library settings, such as a dont_use list. Also, use consistent design goals specifications, such as timing constraints and design rule settings to drive logical and physical implementations.
- Apply a complete set of physical constraints by using a floorplan .def file or Tcl commands before the first `compile_ultra` command step in Design Compiler. Also, reapply the same sets of physical constraints in IC Compiler in order to preserve the placement consistency for physical guidance.
- Since the physical guidance information is restored when you run the IC Compiler `place_opt` command, you should ensure that the `place_opt` command is the first placement-related command that you run in IC Compiler so that Design Compiler placement is restored properly. Using other placement-related commands (such as `create_placement`, `remove_buffer_tree`, and so on) degrades the correlation and can lead to the loss of some physical guidance information.

Reference Methodology Scripts

The Design Compiler reference methodology and IC Compiler reference methodology include complete sets of scripts that implement the recommended synthesis flow in Design Compiler (including topographical mode) and the physical implementation flow in IC Compiler. Each reference methodology includes reference scripts for the hierarchical flow described in this application note. You can download and study the reference methodology scripts in conjunction with the information presented in this application note.

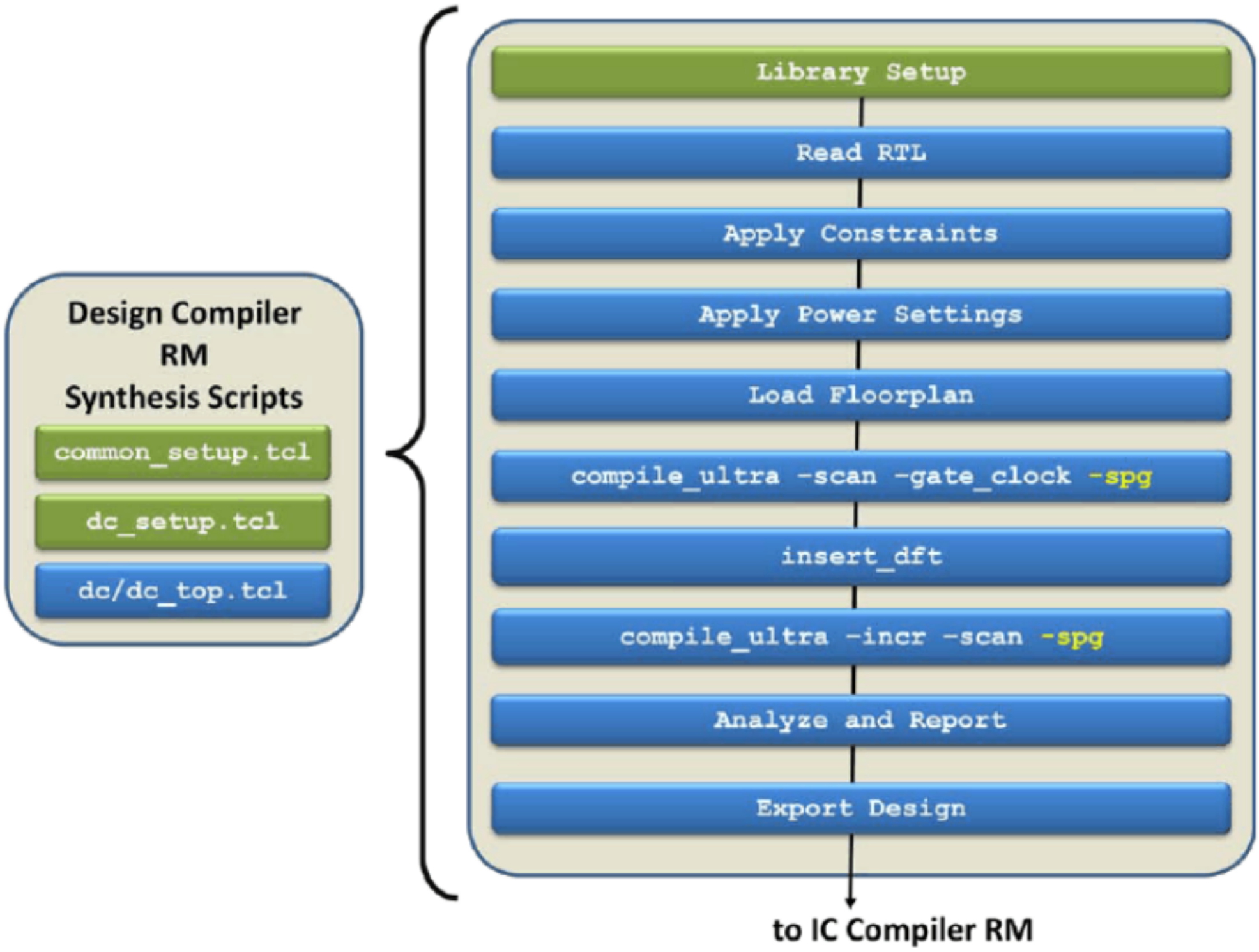
The Reference Methodology Retrieval System, RMgen, provides an easy way to configure and download feature-specific methodology scripts for a given product and release. You select the product name, release number, and feature option settings from a set of menus, and then download the customized scripts based on your selections. You can configure and download the scripts from SolvNet using the following link:

https://solvnet.synopsys.com/rmgen/

Physical Guidance Synthesis Flow

The physical guidance flow can easily be integrated into your existing design flow and environment. The following figure shows how the physical guidance flow can be integrated into a complete implementation flow using the Design Compiler reference methodology.

Figure 2: Design Compiler Reference Methodology Synthesis Flow With Physical Guidance



The Design Compiler reference methodology provides reference scripts for a complete top-down and bottom-up synthesis flow that consists of the following primary steps:

- Library and design setup
- Design constraints and power settings
- Design implementation or compilation
- Design-for-test
- Analysis and exporting the design

The following sections describe how the physical guidance flow impacts these steps.

Library and Design Setup for Synthesis

The first design implementation step in Design Compiler includes setting up the logic and physical libraries to be used for synthesis, providing the TLUplus files used for net parasitics, and loading the design source files.

The physical guidance flow does not impact any of the commands used to load logic and physical libraries or TLUplus files; however, you need to ensure that the setup steps, especially the files used for the setup, are consistent between Design Compiler and IC Compiler. That is, you should load the same sets of the following files in both tools:

- Logic library (.db)
- Milkyway reference library
- Technology file (.tf)
- Mapping file (which maps the technology file layer names to TLUplus layer names)
- Min and max TLUplus files for net parasitics

You should make sure that each standard cell and macro in the logic libraries has a corresponding abstract physical view in the Milkyway reference library. Otherwise, the cells are treated as dont_use cells for mapping and optimization.

In addition, you should use consistent library-specific attribute settings with IC Compiler. For instance, your library cell dont_use list, dont_touch settings, and design rule settings (max_fanout and transition time) should be aligned with the settings used in IC Compiler.

Design Input Source

In general, there is no change in the design input file format for the physical guidance synthesis flow or how the input files are sourced in Design Compiler.

The following table shows the design input formats that are supported.

Table 1: Design Input Source Supported in Design Compiler Topographical

Design Source	Characteristics	Recommended Compile
RTL or GTECH	Unmapped	compile_ultra -spg
Design Compiler wireload .ddc	Mapped	compile_ultra -spg
Design Compiler topographical (non-physical	Mapped with Design Compiler topographical non-physical-guidance-based	compile_ultra -spg

guidance) .ddc	virtual layout info	
Design Compiler topographical (with physical guidance) .ddc	**Mapped with Design Compiler topographical physical-guidance-based virtual layout info**	**compile_ultra -incr -spg**
ASCII netlist	**Mapped**	**compile_ultra -spg, compile_ultra -incr -spg**

The preferred input is RTL because it takes full advantage of Design Compiler optimization, datapath, structuring and mapping, and sequential mapping.

For designs starting with a mapped input source, either a .ddc file or an ASCII gate-level netlist, design implementation is still driven by placement-based optimization; however, datapath, structuring mapping, and sequential mapping are either restricted or limited.

In addition, a .ddc file generated by Design Compiler in topographical mode contains placement information that is used during incremental compilation. Because the incremental compilation is based on the placement and QoR results from the initial `compile_ultra` command run, the subsequent incremental `compile_ultra` step in the physical guidance flow requires that the input .ddc file or in-memory design be generated in physical guidance mode. That is, the design must have been synthesized with the `compile_ultra -spg` command in order for you to run additional `compile_ultra -incremental -spg` command steps.

In short, your choice of design input source impacts QoR. You should use RTL in the physical guidance synthesis flow in order to provide the best starting implementation netlist for IC Compiler. However, a .ddc file is the recommended input for the `compile_ultra -incremental -spg` command in Design Compiler topographical mode.

Design Constraints and Power Settings for Synthesis

Before performing design implementation, you need to first specify the design QoR objectives for synthesis. You need to apply the design constraints and power settings that will be used to perform mapping and placement-driven optimization in Design Compiler topographical mode.

In the physical guidance synthesis flow, it is especially important that you fully specify your design goals before you run the first `compile_ultra -spg` command step in order to achieve good QoR results that correlate to IC Compiler.

Keeping in mind that Design Compiler physical guidance information is used as seed placement for the `place_opt -spg` command in IC Compiler, the following design objectives should be consistent so the placement is aligned and the timing context is the same between both tools:

- Design-specific settings
- Timing constraints

- Design physical constraints
- Power optimization settings

The following sections describe how physical guidance impacts the various design constraint and power optimization steps.

Design-Specific Settings

Align your design settings in Design Compiler with the settings used in IC Compiler. For example, if your back-end flow disables the use of certain metal layers for routing or uses any commands that impact net parasitic (resistance and capacitance) estimation, you need to specify the same design settings in Design Compiler.

Here are examples of the type of design settings that should be consistent between Design Compiler and IC Compiler:

- Use the `set_ignored_layers` command to disable the routing resources on certain metal layers. Ignoring metal layers impacts Design Compiler placement-driven net capacitance and resistance computation. The ignored metal layer is not considered part of the routing resource; its impact on the resistance and capacitance is not considered during optimization and congestion estimation in topographical mode.
- Use the `set_delay_estimation_options` command to specify scaling factors for derived resistance (R) and capacitance (C) from the library or override the per-unit R and C values of any metal layers specified in the physical library. A scaling factor within 5% to 10% is usually sufficient.
- Use the `set_ahfs_options` command to fine tune automatic high-fanout synthesis options. By default, Design Compiler performs placement-based automatic high-fanout synthesis for high-fanout nets (with fanout above 100) such as reset and scan-enable, with the exception of dont_touch nets, ideal nets, and DRC-disabled clock and constants nets. Placement of fanout influences buffering. As a result, you need to allow Design Compiler to perform the default automatic high-fanout synthesis and only configure automatic high-fanout synthesis options to the same options as IC Compiler for consistent buffering behavior. IC Compiler automatically performs incremental automatic high-fanout synthesis as needed when you run the `place_opt` command.
- Use the `dont_touch` and `size_only` attributes to control cell mapping and optimization. The usage of these attributes limits optimization flexibility and impacts QoR and correlation. Thus, you should use the same sets of `dont_touch` and `size_only` attributes in both tools.
- Use the same set of `set_ideal_network` constraints that are defined during `place_opt` in IC Compiler to ensure correlation.

You should only use these commands when they are being used in your back-end flows because usage impacts placement, parasitic estimation, mapping and placement-based optimization in Design Compiler in topographical mode the same way as in IC Compiler. Using the commands in both tools results in improved correlation for physical implementation.

Timing Constraints

Design Compiler topographical performs placement-based optimization that helps to achieve your design goals. One of the critical design goal components is specifying the timing constraints used to drive synthesis. Timing constraints consist of the following:

- Clock constraints (period, latency, transition, uncertainty, and so on)
- Primary input and output delays
- Timing exceptions – multicycle and false paths
- Path groups, timing weight, and critical range

In the physical guidance synthesis flow, you need to eliminate any over-constrained margins in Design Compiler topographical mode and use the same design timing constraints in IC Compiler.

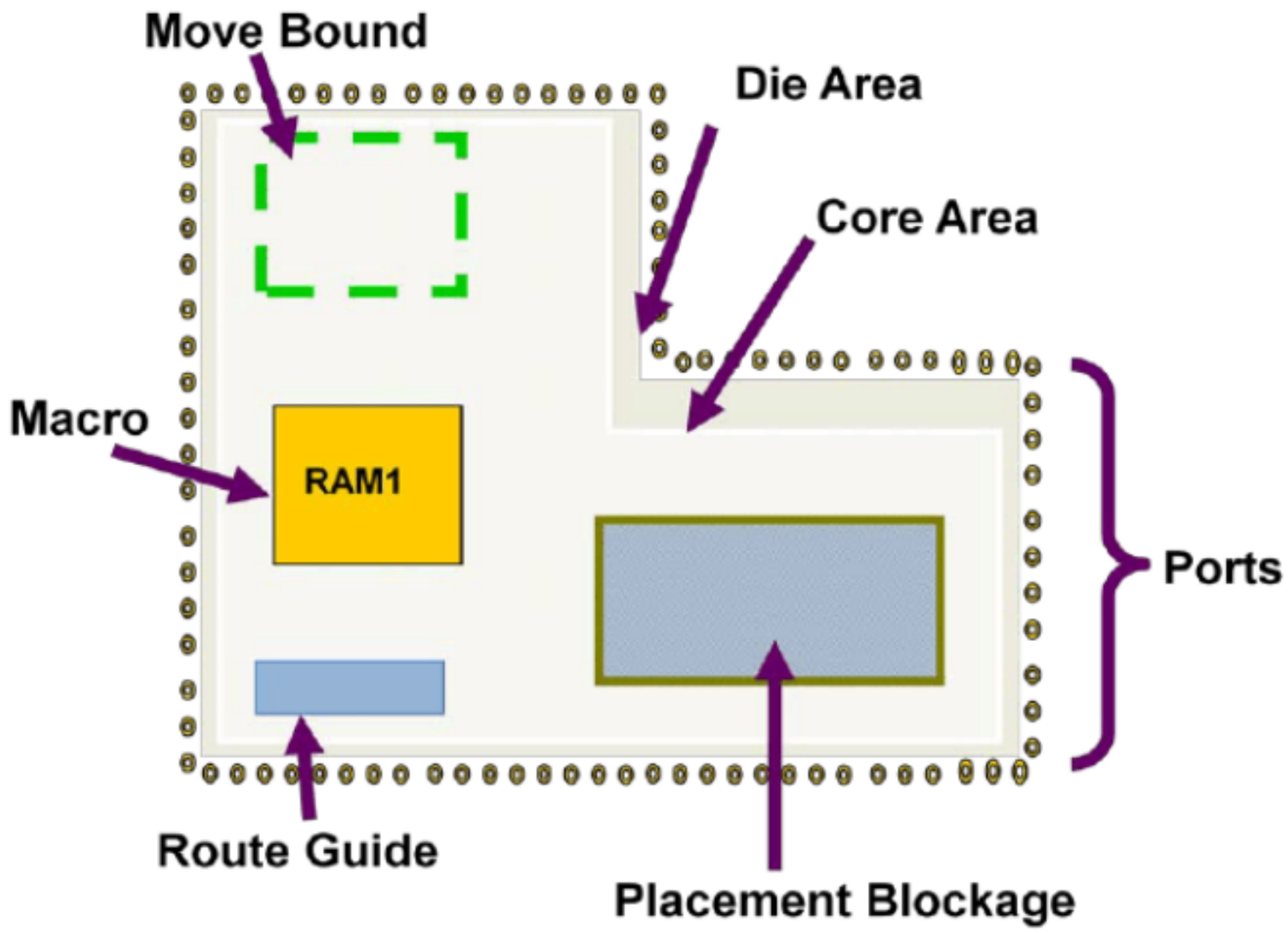
Design Physical Constraints

In the physical guidance flow, you should drive placement-based optimization in synthesis with the same set of physical constraints used in IC Compiler in order to preserve placement consistency and improve correlation to IC Compiler.

Design Compiler supports a number of physical constraints. The physical constraints are either extracted from your design floorplan DEF file with the `extract_physical_constraints` command or directly sourced from the Tcl output of the IC Compiler `write_floorplan` command in Design Compiler topographical mode.

The following figure shows some of the physical constraints supported in topographical mode.

Figure 3: Supported Physical Constraints in Design Compiler Topographical Mode



In addition to the physical constraints shown in the figure, Design Compiler also supports constraints for specifying the following:

- Net shapes or preroutes
- Relative placement
- Voltage area
- Tracks

You should consider the following physical constraint specification and usability guidelines for physical guidance flows and comply with the minimum set of floorplan requirements before you proceed with the `compile_ultra` implementation phase:

- Use the same set of physical constraints and floorplan in Design Compiler as in IC Compiler (recommended).
- Although the preference is for you to drive synthesis with the same floorplan used for physical implementation in IC Compiler, you should comply with the first sets of physical constraints as a minimum before proceeding with mapping and placement-based optimization using the `compile_ultra -spg` command.

The following list provides the physical constraint requirements and recommendations for placement consistency and improved correlation between the tools:

- Die area (required)
- IC Compiler block abstraction and ILM location and Design Compiler physical block location (required)
- site arrays (recommended)
- Macro locations (recommended)
- Port locations for all primary I/Os (recommended)

In order to ensure that you have provided the required floorplan information, explicit physical guidance usability checks are performed at the onset of the `compile_ultra -spg` command run. You can also use the `compile_ultra -check_only -spg` command to get information about any missing physical constraints.

- For test ports, the preferred methodology is that you include the test ports in the RTL code so that everything downstream from the RTL has the same interface. In this case, you should specify the fixed port locations before you first run the `compile_ultra` command in the physical guidance flow. Alternatively, you can create the test ports after the first `compile_ultra` run and then specify the fixed port locations to be used in subsequent incremental compile steps.
- Do not modify the floorplan information between the initial `compile_ultra` step and subsequent incremental compilation steps with the exception of creating new test ports and specifying locations for the new test ports. The following errors are issued if you try to change the physical constraints in between subsequent `compile_ultra` steps, and the new physical constraints are discarded:
  - `Error: The design was optimized using compile_ultra -spg. Cannot update the physical constraints using the %s command. (SPG-008)`
  - `Error: The design was optimized using compile_ultra -spg.  Cannot update the physical constraints. (SPG-009)`

If you need to modify the floorplan, you can use the floorplan exploration capability available with Design Compiler Graphical. For more information about the Design Compiler Graphical floorplan exploration capability, see the *Design Compiler Graphical Floorplan Exploration Application Note* on SolvNet, located at the following link:

https://solvnet.synopsys.com/retrieve/030613.html

Power Optimization Settings

Power optimization and prediction are supported in the physical guidance synthesis flow in the same way as the non-physical guidance synthesis flow. This capability is enabled when you run either the `compile_ultra -spg` command or the `compile_ultra -incremental -spg` command. It estimates clock tree power and therefore provides correlated power post-synthesis when compared to place-and-route numbers.

You can enable leakage and dynamic power optimization by using the following commands:

- `set_leakage_optimization true`
- `set_dynamic_optimization true`

When leakage power optimization is enabled, you should provide multithreshold libraries as target libraries for the best power results.

In addition, you can use the `-gate_clock` option with the `compile_ultra` command to enable clock gating optimization. A clock gating cell is automatically inserted, modified, or removed unless you have marked the cell or its parent hierarchical cell as dont_touch with the `set_dont_touch` command.

If you enable power correlation by using the `set_power_prediction` command, clock tree estimation is run within the `compile_ultra` or `compile_ultra -incremental` commands as the last phase. Any subsequent incremental compilations cause clock tree estimation to run again.

You should use the `-cts_references` option with the `set_power_prediction` command to provide the list of library cells to be used for clock tree estimation. Using the same clock buffers that are used in IC Compiler clock tree synthesis helps ensure that the predicted power is further correlated with IC Compiler.

With the power prediction mode enabled, you can use the `report_power` command to report the correlated power after the design has been mapped to technology-specific cells. Otherwise, the `report_power` command reports only the total (static and dynamic) power used by the design without accounting for power usage attributed to clock-tree estimates.

compile_ultra Flow

The following sections describe the physical guidance flow impact on the `compile_ultra` command and other implementation commands in topographical mode.

compile_ultra -spg Command

As stated previously, you use the `-spg` option with `compile_ultra` command to enable the physical guidance flow in Design Compiler. In Design Compiler physical guidance mode, mapping and placement-based optimization is further enhanced to produce highly optimized placements and a better starting netlist for IC Compiler.

The `compile_ultra -spg` command impacts Design Compiler mapping and placement-driven optimization as described in the following sections.

- The `-spg` option works seamlessly with most `compile_ultra` command-line options, as shown in the following table.

Table 2: Usage of –spg Option with Other compile_ultra Command Options in Design Compiler Topographical Mode

compile_ultra Options	Use with -spg Option
-no_seq_output_inversion	Yes
-no_autoungroup	Yes
-exact_map	Yes
-no_boundary_optimization	Yes
-timing_high_effort_script	Yes
-retime	Yes
-incremental	Yes
-no_design_rule	Yes
-only_design_rule	Yes
-top	No
-check_only	Yes
-self_gating	Yes
-scan	Yes
-gate_clock	Yes

- In the physical guidance flow, you are required to provide the die area. In addition, the locations of IC Compiler block abstractions and ILMs and Design Compiler physical blocks must be fixed. It is also recommended that you define the port locations and macro placements.
- Physical guidance usability checks are automatically performed the first time you run the `compile_ultra -spg` command in topographical mode to make sure you have provided the required floorplan information. You can also use the `compile_ultra -check_only -spg` command to perform these checks. The following errors and warnings are issued for any missing floorplan information:
    - `Error: The design does not have a user-specified core area. (SPG-001)`
    - `Warning: The macro %s does not have a user-specified location. (SPG-003)`
    - `Warning: Port '%s' is missing location, compile_ultra will automatically assign a location for this port. (SPG-011)`

    - `Warning: '%d' ports inserted by test are missing location, compile_ultra will automatically assign a location for these ports. For more details run compile_ultra -check_only -spg. (SPG-012)`
    - `Warning: '%d' ports are missing location, compile_ultra will automatically assign a location for these ports. For more details run compile_ultra -check_only -spg. (SPG-013)`

- In the physical guidance flow, RTL congestion optimization is enabled as well as congestion-aware placement to consider cell density and provide a more accurate net model for optimization in topographical mode. This helps to ensure that timing is consistent between the end point of Design Compiler and the starting point of IC Compiler with the `place_opt` command.

  At the same time, placement and placement-based optimization is improved in physical guidance mode by enhancing the delay optimization effort in Design Compiler to be consistent with the behavior in IC Compiler.

- At the end of the `compile_ultra -spg` run, Design Compiler creates signatures of the physical constraints that were used. The signatures are saved in binary format, in either the .ddc file or the Milkyway CEL, to be later matched in IC Compiler against the physical constraints used for `place_opt -spg`. This allows you to ensure that the physical constraints used in Design Compiler match the physical constraints used in IC Compiler.

- At the end of the `compile_ultra -spg` command run, the mapped top-level design is marked with a read-only `dct_spg_flow_done` attribute to signify that the design was implemented with physical guidance. You can query the attribute to help determine if your design .ddc file or Milkyway CEL was generated using physical guidance.

- The derived placement for standard cells and the design floorplan information used to drive synthesis are retained on the in-memory implemented design. This physical guidance information is later saved in binary format in the .ddc file and the Milkyway CEL by the `write_file` and `write_milkyway` commands, respectively. In the ASCII flow, the physical guidance information can be saved in DEF format in floorplan exploration.

compile_ultra -incremental -spg Command

Design Compiler also enables physical guidance with an incremental flow. The information provided in the "compile_ultra –spg Command" section also applies to the `compile_ultra -incremental -spg` command, in addition to the following:

- When you run the `compile_ultra -incremental -spg` command on an input .ddc file, an explicit check is performed looking for the presence of the read-only `dct_spg_flow_done` attribute to ensure that the design was initially implemented in physical guidance mode.

  The following error message is issued when you run the `compile_ultra -incremental -spg` command on a .ddc file that was generated in non-physical guidance mode:

    - `Error: Cannot run compile_ultra incremental in spg mode.`

      `Design was not compiled with DC-Topographical spg optimizations (DCT-054)`

- Once Design Compiler detects that your incoming .ddc file was generated in physical guidance mode, any subsequent placement-based optimization using the `compile_ultra -incremental` command uses physical guidance regardless of whether you include the `-spg` option. That is, the `compile_ultra -incremental` and `compile_ultra -incremental -spg` commands have the same effect and the following warning is issued:

    - `Warning: Run compile_ultra incremental in spg mode.`

      `Design was compiled with DC-Topographical spg optimizations. (DCT-055)`

- Although it is recommended that you run the `compile_ultra -incremental -spg` command on a .ddc file generated in topographical mode, the physical guidance flow supports the entire input source format spectrum listed in the "Design Input Source" section.

  The expected behavior for the `compile_ultra -incremental -spg` command on various input source types is as follows:

    - **.ddc file generated with -spg option**: The `compile_ultra -incremental -spg` command proceeds with placement-based optimization in physical guidance mode.
    - **Unmapped netlist**: The `compile_ultra -incremental -spg` command runs a full `compile_ultra` synthesis by first mapping the design and then performing placement-based optimization in physical guidance mode.
    - **ASCII mapped netlist**:
        - If you provide standard cell placement information by using the `extract_physical_constraints -standard_cell spg` command, the `compile_ultra -incremental -spg` command proceeds with placement-based optimization in physical guidance mode. The tool does not support a netlist that contains both mapped and unmapped cells in this flow.
        - If you provide standard cell placement information by using the `extract_physical_constraints -standard_cell topo` command, the `compile_ultra -incremental -spg` command issues a DCT-054 error message because the design was not compiled with Design Compiler topographical physical guidance optimizations.
        - If you do not provide standard cell placement information, the `compile_ultra -incremental -spg` command runs limited mapping optimizations and then performs placement-based optimization in physical guidance mode.

- **.ddc file generated without -spg option**: The `compile_ultra -incremental -spg` command issues a DCT-054 error message. You must first use `compile_ultra -spg` to map the design.

Other Mapping Commands (Such as optimize_registers and insert_dft)

If you use other commands, such as `optimize_registers` or netlist editing commands that perform various degrees of design modifications, it is recommended to run an explicit incremental compilation flow afterward with the `-spg` option. However, if you do not use the `compile_ultra -incremental -spg` command, the `place_opt -spg` command can still handle the cells that do not have physical guidance defined.

The following list of commands can be followed by a `compile_ultra -incremental -spg` command step after you run them:

- `optimize_registers`
- `insert_dft`
- `insert_buffer`
- `remove_buffer`
- `balance_buffer`
- `change_link`

Design-For-Test (DFT) in Design Compiler

Design Compiler supports design-for-test (DFT) insertion features with the `insert_dft` command. DFT Compiler run in topographical mode behaves as follows:

- No timing optimization or DRC fixing is performed with the `insert_dft` command. You must run an incremental compile after DFT insertion using the `compile_ultra -incremental -scan -spg` command.
- Scan chain ordering and partitioning is done based on the placement information from the initial compilation.

The following DFT features are supported in Design Compiler topographical mode:

- Scan
- AutoFix
- Scan compression (adaptive scan)
- PLL flows (user-defined clock controller stitching)
- Core wrapping
- Boundary scan
- Clock-gating connection

- Specification of compressor and decompressor location
- Top-down compression flow (using the `define_dft_partition` command)

For scan chain reordering in IC Compiler, see the "Design-For-Test (DFT) in IC Compiler" section.

Exporting the Design

You can save standard cell placement in Design Compiler in binary or ASCII format, as described in the following sections.

Saving in Binary Format

You can save standard cell placement in .ddc format by using the `write_file -format ddc` command, or save it in Milkyway CEL format by using the `write_milkyway` command.

In regular topographical mode, the standard cell placement can only be used by Design Compiler for further optimization. IC Compiler does not use it. However, in the physical guidance flow, the standard cell placement information is propagated to IC Compiler through the .ddc file or the Milkyway CEL.

The physical constraints and floorplan information that you used for synthesis are not passed from Design Compiler to IC Compiler. However, the physical constraint signatures that are created from the physical constraints at the end of the `compile_ultra -spg` and the `compile_ultra -incremental -spg` command steps in the physical guidance flow are saved in the .ddc and Milkyway CEL and are read by IC Compiler. The signatures are matched in IC Compiler when the physical guidance information is restored in order to ensure floorplan consistency between the floorplan used in synthesis and the floorplan used for physical implementation.

All binary netlists that are generated in the physical guidance synthesis flow, whether a .ddc file or a Milkyway CEL, also contain a read-only `dct_spg_flow_done` attribute to signify that the design was implemented with the physical guidance flow. You can query the attribute to determine if your design .ddc file or Milkyway CEL was generated using the physical guidance flow.

Saving in ASCII Format

When you save the design in ASCII format, either in a Verilog or a VHDL netlist, the file that is created does not contain standard cell placement information or the signatures of the physical constraints.

You can save the standard cell placement information in a DEF file by using the `write_def` command in Design Compiler Graphical floorplan exploration. You can then read the DEF file, along with the ASCII netlist, back into Design Compiler to be optimized with the `compile_ultra -incremental -spg` command, or you can read it into IC Compiler for the `place_opt -spg` step.

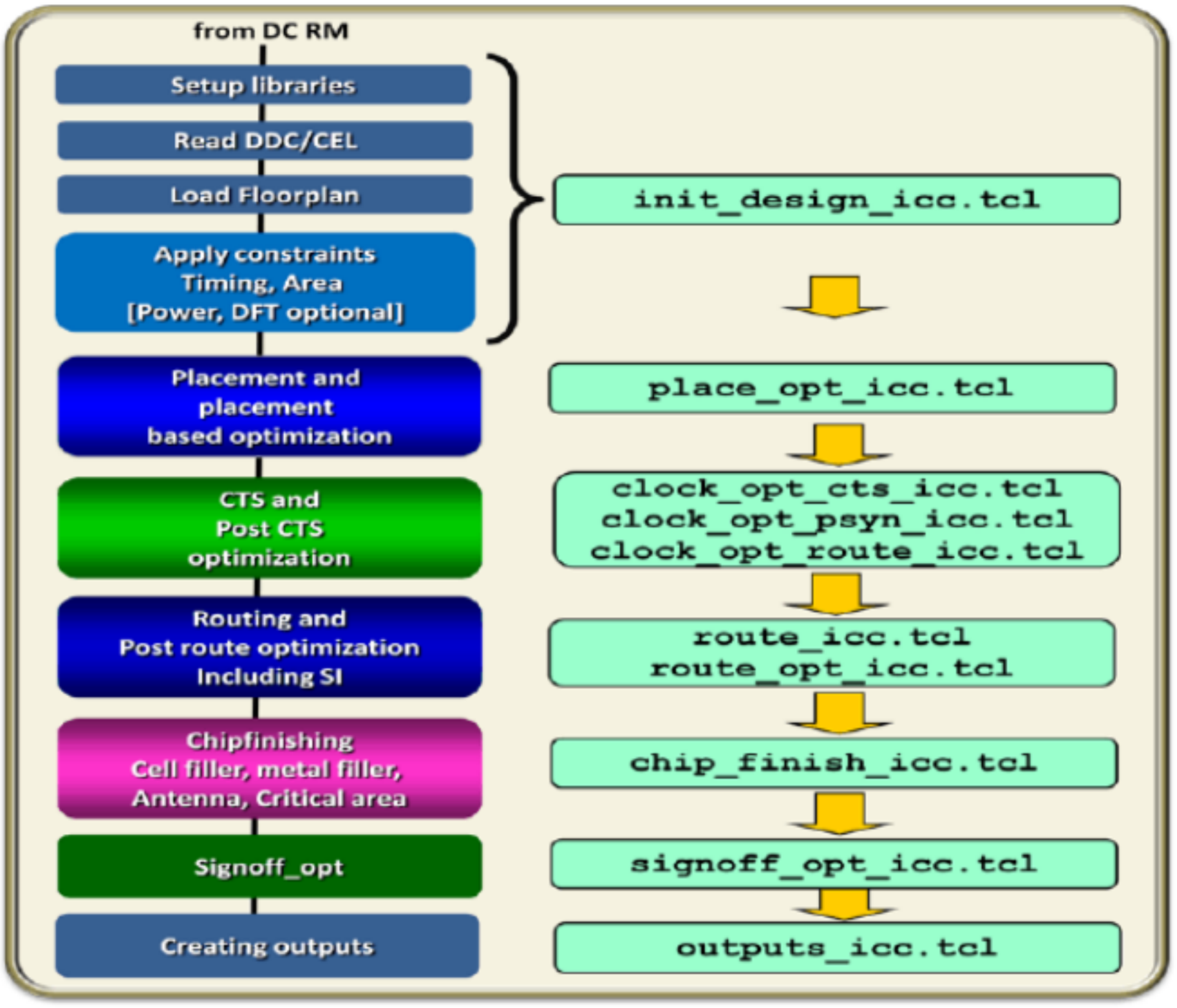
Physical Guidance Implementation Flow

You enable the physical guidance flow for IC Compiler physical implementation by using the `-spg` option with the `place_opt` command. The physical guidance flow can be easily integrated into your existing design flow and environment.

The IC Compiler reference methodology provides a set of reference scripts that you can use as a recommended guideline for a complete physical implementation flow. The following figure shows the scripts that you can use to perform design loading, placement, clock tree synthesis, and routing using the data files produced by the topographical mode physical guidance synthesis flow:

The following figure shows the IC Compiler reference methodology flow.

Figure 4: IC Compiler Reference Methodology Flow Diagram



The IC Compiler reference methodology includes the following files:

- common_setup.tcl/icc_setup.tcl – Includes design setup variables for reference methodologies
- init_design_icc.tcl – Loads design netlist, constraints, and floorplan DEF file
- place_opt_icc.tcl – Runs placement and placement-based optimization
- clock_opt_cts_icc.tcl – Runs clock tree synthesis and optimization
- clock_opt_psyn_icc.tcl – Runs post-clock tree synthesis optimization
- clock_opt_route_icc.tcl – Runs clock tree routing
- route_icc.tcl – Runs routing
- route_opt_icc.tcl – Runs postroute optimization
- chipfinish_icc.tcl – Performs chip finishing
- signoff_opt_icc.tcl – Runs sign-off-driven optimization using StarRC and PrimeTime
- outputs_icc.tcl – Exports completed design

The physical guidance flow impacts only the placement and placement-based optimization stages in the IC Compiler reference methodology flow, while the remaining stages are not affected.

The following sections describe how the physical guidance flow impacts the loading of the design netlist, constraints, and floorplan information, in addition to placement and placement-based optimization.

Library Setup and Design Constraints for Physical Implementation

You need to specify the same logic and physical library settings for the physical implementation flow that were specified in the design implementation stage in Design Compiler before you load the design source files that are produced in physical guidance mode. For more information, see the "Library and Design Setup for Synthesis" section.

Before you start the design physical implementation, you must also apply the design constraints and power settings to be used to perform placement and placement-based optimization in IC Compiler. It is important that you fully specify the same set of design objectives used in topographical mode for the physical guidance implementation flow in order to achieve good QoR results that correlate with Design Compiler.

Keeping in mind that Design Compiler physical guidance information is used as seed placement for IC Compiler, you should align your design-specific settings with those in topographical mode. For instance, any design settings that affect placement and extraction should be consistent so that placement is aligned and the timing context is the same between Design Compiler and IC Compiler. For more information, see the "Design-Specific Settings" section.

You also need to align your power flow to the power optimization settings used in Design Compiler to achieve tight power correlation. To effectively enable power

optimizations in the IC Compiler physical guidance flow, you need to use the `set_power_options` command to specify your power optimization options in conjunction with the `-power` option with the `place_opt` command.

Design Physical Constraints

In the physical guidance implementation flow, you must drive placement-based optimization in IC Compiler with the same set of physical constraints, or floorplan information, used in Design Compiler in order to preserve placement consistency and improve correlation between both tools.

Do not modify any of the physical constraints used to drive Design Compiler in topographical mode. When the .ddc file or Milkyway CEL from Design Compiler is read in IC Compiler during the `place_opt -spg` step, IC Compiler matches its floorplan constraints to the signatures created by Design Compiler. Thus, you should not change the physical constraints, such as the die area, any fixed macro placements, port locations, move bounds, and so on.

However, you can provide additional floorplan information that is not currently supported by Design Compiler that is needed for physical implementation. Warning and error messages are generated when a change in the physical constraints between Design Compiler and IC Compiler are detected.

Design-For-Test (DFT) in IC Compiler

Physical scan chain reordering is performed in IC Compiler with the `place_opt -optimize_dft -spg` command. The information needed to perform scan chain reordering is generated by DFT Compiler after DFT insertion. This information is commonly known as the SCANDEF file.

The SCANDEF information is embedded in the in-memory design when you use the `write_scan_def` command in Design Compiler. The SCANDEF information is then directly transferred from Design Compiler to IC Compiler in the .ddc file that is written out in topographical mode.

If you are using a Milkyway CEL, you need to write out the SCANDEF information to a disk in ASCII format using the `write_scan_def -output` command in DFT Compiler, and read it into IC Compiler using the `read_def` command.

The `optimize_dft` command is also available in IC Compiler as an atomic command if you want to perform scan chain reordering after the `place_opt` command. However, the recommended command is `place_opt -optimize_dft -spg`. When run as a core command, `place_opt -optimize_dft -spg` ignores the existing scan connections for placement purposes so that scan nets do not influence placement results.

If DFT logic is present, functional timing and power should correlate between Design Compiler and IC Compiler. That is, functional timing before DFT insertion in Design Compiler should correlate with functional timing after scan reordering in IC Compiler.

Design Physical Implementation (Placement)

The following sections describe the impact of the physical guidance flow on placement and placement-based optimization.

- The `-spg` option works seamlessly with other `place_opt` command-line options, as shown in the following table.

Table 3: Usage of –spg Option with Other place_opt Command Options in IC Compiler

place_opt Options	Use with –spg Option
-effort medium | high	yes
-area_recovery	yes
-optimize_dft	yes
-congestion	yes
-cts	yes
-power	yes

In addition, the physical guidance flow supports both a medium- and high-effort level for the `place_opt` command. It does not support a low-effort level and issues the following error message:

```
o  Error: Cannot use '-spg' option with the '-effort low'
   option. (UIO-9)
```

- An explicit check is performed to ensure that the read-only `dct_spg_flow_done` attribute is present in the design netlist before placement can proceed in IC Compiler. This check only applies when you use a binary flow (as opposed to an ASCII flow). The following table shows various flows and the corresponding warning and error messages that are issued when you do not use the recommended physical guidance flow. As the table shows, the recommended flow uses the `compile_ultra -spg` and `place_opt -spg` commands.

Table 4: Physical Guidance Flow Recommendation in Design Compiler Topographical Mode and IC Compiler

Design Compiler Flow	IC Compiler Flow	Recommended Flow (Error/Warning)
compile_ultra (Design Compiler wireload)	place_opt –spg	No Error: RTL was not optimized using "compile_ultra -spg" in Design Compiler Graphical. (SPG-100)
compile_ultra	place_opt –spg	No Error: Design does not have Physical Guidance Information. (SPG-101)

compile_ultra –spg	place_opt	No Warning: Design RTL was optimized using "compile_ultra -spg", please use -spg in place_opt. (SPG-103)
compile_ultra –spg	place_opt –spg	Yes (None)

- The physical guidance information is restored at the beginning of the `place_opt` command when it is run in IC Compiler. The following message is printed in the log file once the physical guidance information is restored successfully:

```
icc_shell> place_opt -spg . . . .
Restoring DCT Placement
```

However, if you use the `place_opt -spg` command on a design netlist lacking the required physical guidance information and the read only `dct_spg_flow_done` attribute, you get the following error:

```
icc_shell> place_opt -spg

Restoring DCT Placement
Failed to restore DC Graphical Physical guidance
Information, please remove -spg option. (SPG-102)
```

- The `place_opt -spg` command has a usability check to verify that the floorplan constraints used in Design Compiler match the floorplan constraints defined in IC Compiler. This usability check only works with a binary handoff (.ddc or Milkyway CEL) from Design Compiler to IC Compiler. When the usability check detects any inconsistency in the floorplan constraints, the following set of warnings are generated.

Table 5: Mismatched Scenarios Between Physical Guidance Information and User-Specified Floorplan Information in IC Compiler

Mismatched Scenarios	Warnings
Mismatch between floorplan used in Design Compiler topographical and IC Compiler (IC Compiler constraints prevail)	Warning: Number of %s provided in DC Graphical %d does not match ICC %d. (SPG-120) Warning: %s information does not match. (SPG-121)
Found cells with both physical guidance info and fixed location in IC Compiler (physical guidance info is ignored)	Warning: Cell '%s' has both DC Graphical physical guidance and fixed location. (SPG-105)
Found cells with both physical guidance and placed location in IC Compiler (physical guidance is preserved)	Warning: Cell '%s' has both DC Graphical physical guidance and placed location. (SPG-106)

Other Flows Supported in the Physical Guidance Flow

The following sections describe other flows that are supported in the physical guidance flow.

Restoring Ports and Standard Cell Placement Before place_opt –spg

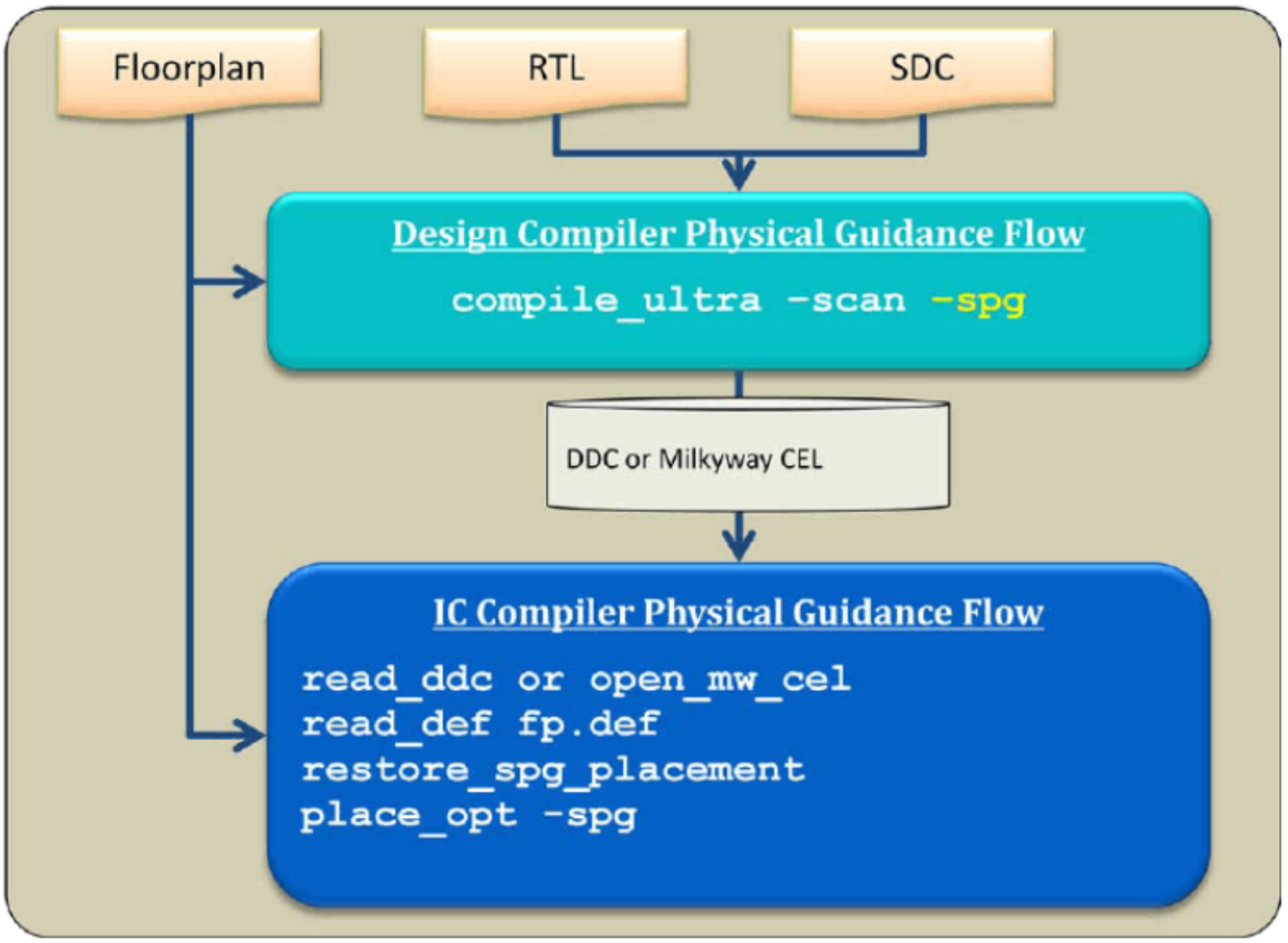
In IC Compiler, you can use the `restore_spg_placement` command to restore

- The standard cell placement done by the Design Compiler `compile_ultra -spg` command.
- The port location created by the Design Compiler `compile_ultra -spg` command for ports that were not constrained by the `set_port_location` command or defined in the DEF file.

This allows you to

- Access the physical guidance information before using the `place_opt -spg` command.
- View the physical guidance in the IC Compiler graphical user interface.
- Modify the restored standard cell physical guidance placement before `place_opt -spg`, if required.
- Run `place_opt -spg` even if some port locations are missing in the DEF file by using the placement created during the `compile_ultra -spg` command run.

Figure 5: restore_spg_placement Example



The order that you run the `read_def` and `restore_spg_placement` commands is important for standard cell guidance.

When the DEF file contains placed standard cells, the `place_opt -spg` command determines standard cell placement based on the following rules:

- If you use the `restore_spg_placement` command after the `read_def` command, the `place_opt -spg` command uses the standard cell placement from the `restore_spg_placement` command.
- If you use the `read_def` command after the `restore_spg_placement` command, the `place_opt -spg` command uses the standard cell placement from the `read_def` command.

The `place_opt -spg` command always uses the port location defined in the DEF file in place of the port location restored from the `restore_spg_placement` command.

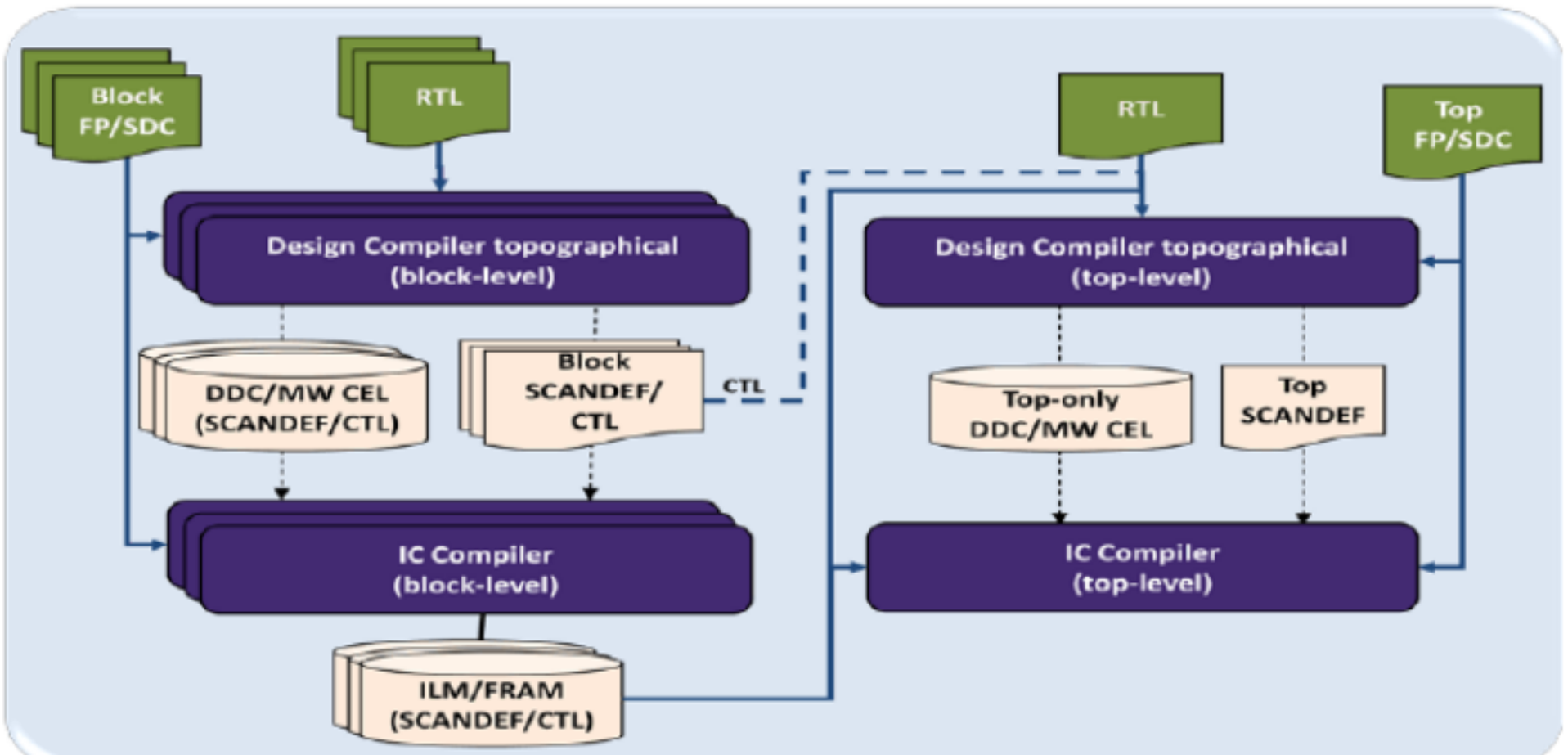
Hierarchical Flow

You can use ILMs or block abstractions that were generated by IC Compiler for subblocks in the Design Compiler and IC Compiler hierarchical top-level integration flow. This aligns the physical block usage and maintains flow consistency in both tools. The integration of the same ILMs or block abstractions at the top level in both Design Compiler and IC Compiler ensures that the block-level timing and physical information are consistent between the tools and helps ensure tight correlation. Note that you cannot combine ILMs and block abstractions in the same flow.

The `compile_ultra -spg` command also supports the use of Design Compiler generated ILMs, physical hierarchies, or block abstractions. However, these are not supported by the `place_opt -spg` command.

Beginning with the HDL source code and using the hierarchical floorplan DEF file and budgeted SDC timing constraints, as shown in the following figure, you can perform block-level design synthesis and physical implementation in Design Compiler and IC Compiler, respectively, with the physical guidance flow. At the end of block-level physical implementation, you create an abstract model and use the ILM for top-level design integration in Design Compiler and IC Compiler.

Figure 6: Example of Hierarchical Physical Guidance Flow Using IC Compiler ILMs



The Design Compiler and IC Compiler hierarchical flow using IC Compiler ILMs consists of the following steps:

1. To implement the subblock, perform the following steps (see the Design Compiler and IC Compiler reference methodologies scripts):
   a. In Design Compiler, load the subblock files (Verilog, VHDL, netlist, or .ddc).
   b. Apply block-level timing constraints, power constraints, and physical constraints.

c. Perform a test-ready and clock-gating compile of the subblock by using the `compile_ultra -scan -gate_clock -spg` command.

d. Perform block-level DFT insertion and subsequent incremental mapping with the `compile_ultra` **`-incremental`** `-scan -gate_clock -spg` command.

e. Use the `uniquify_naming_style` command to specify the unique naming convention to be used by the `uniquify` or `uniquify -force` command to uniquify the subblock. This prevents naming conflicts during top-level design integration.

f. Run the `change_names -rules verilog -hierarchy` command to apply the Verilog naming rules to all the design objects prior to writing out the design data files.

g. Use the `write_scan_def` command to save the SCANDEF information and use the `write_test_model` command to save test model information about the block-level scan chain.

h. Save the mapped subblock in .ddc format and/or in a Milkyway CEL.

i. In IC Compiler, load the mapped subblock files (.ddc or Milkyway CEL).

j. Apply the same block-level physical constraints used in topographical mode as well as SCANDEF information for specifying the scan chain.

k. Perform placement, clock tree synthesis, and routing in IC Compiler.

l. Generate the block abstraction, ILM view, or the FRAM view for physical subblocks in IC Compiler to be later used for the top-level integration run in Design Compiler and IC Compiler.

2. To compile the top-level design, perform the following steps (see the Design Compiler reference methodology script dc_scripts/dc_top.tcl):

   a. Perform the following steps at the top-level design integration stage in Design Compiler to integrate the Milkyway ILM and FRAM views that you created using IC Compiler:

      i. Add the subblock Milkyway ILM view to the link library variable specification, as shown in the following example:

```
set link_library "$link_library $block.ILM"
```

      ii. Add the IC Compiler Milkyway subblock design library to the `mw_reference_library` variable specification, as shown in the following example: `set mw_reference_library "$mw_reference_library $block.mw"`

   b. Read in the top-level source files (Verilog, VHDL, netlist, or .ddc) and link the top-level design. When you link the top-level design, the `link` command automatically links to the subblock abstract ILM model or ILM view under the specified subblock Milkyway design library.

   c. Use the `read_test_model` command to load the test models for physical subblocks.

d. Apply the top-level timing constraints, power constraints, and physical constraints. All subblocks must have a fixed location.

e. Perform a test-ready and clock-gating compilation of the top-level design by using the `compile_ultra -scan -gate_clock -spg` command.

f. Specify the design-for-test configuration and perform scan stitching with the `insert_dft` command. The block-level scan chain is incorporated in the top-level scan chain at the physical block interface test pins.

g. Run the `compile_ultra -scan -incremental -spg` command.

h. Use the `write_scan_def` command to save the top-level SCANDEF information and save the mapped top-level design netlist in .ddc format or the Milkyway CEL view.

i. In the IC Compiler top-level design, you can integrate the Milkyway ILM and FRAM views that you created for the subblock by adding the IC Compiler Milkyway subblock design library to the `mw_reference_library` variable specification, as shown in the following example: `set mw_reference_library "$mw_reference_library $block.mw"`

j. Load the mapped top-level files (.ddc or Milkyway CEL) and apply the same top-level physical constraints used in topographical mode as well as the SCANDEF information for specifying the scan chain.

k. Perform placement, clock tree synthesis, and routing in IC Compiler.

For more information about Design Compiler and IC Compiler hierarchical flow support, see the *Hierarchical Flow Support in Design Compiler Topographical Mode and IC Compiler Application Note* on SolvNet:

https://solvnet.synopsys.com/retrieve/026926.html

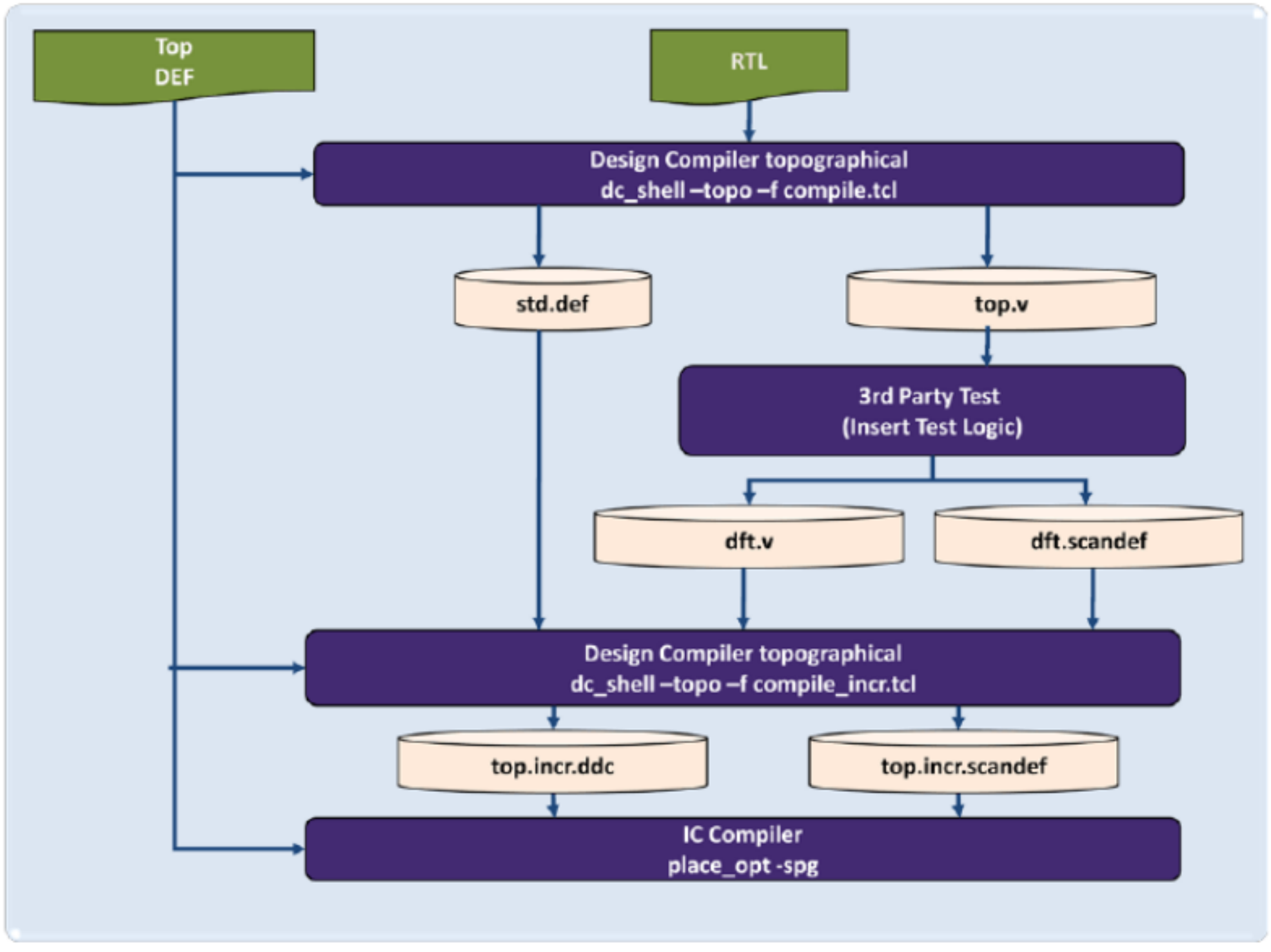
Incremental ASCII Flow With a Third-Party DFT Support Example

Physical guidance information is automatically saved with your design when you use the .ddc or Milkyway file format. However, some flows require you to use a Verilog or VHDL (ASCII) netlist output format. Verilog and VHDL formats are not suitable for physical guidance storage. Therefore, you need to save the physical guidance information in a separate DEF file along with the gate-level Verilog or VHDL netlist. This section describes the incremental ASCII flow using a third-party DFT flow as an example.

Saving the physical guidance information in DEF format allows you to retain the placement of the standard cells after the initial `compile_ultra -spg` command run. The standard cell placement is restored for the netlist coming back from test insertion and is used in the incremental compilation with the `compile_ultra -incremental -spg` command. During the incremental step, Design Compiler uses the placement information that was saved in the DEF file as the seed placement, allowing better correlation between the `compile_ultra -spg` and `compile_ultra -incremental -spg` steps. This flow only works with a fully mapped netlist.

The following figure shows the physical guidance flow for third-party DFT flows.

Figure 7: Physical Guidance Third-Party Test Flow Support



compile.tcl Script

The following compile.tcl script shows a typical initial `compile_ultra` flow. However, it also uses Design Compiler Graphical floorplan exploration after the `compile_ultra` step in order to generate the complete floorplan constraints that contain the standard cell placement. The floorplan exploration step is run with the `start_icc_dp -f icc_dp.tcl` command.

```
# Analyze/elaborate step
analyze -f verilog top_design.v
elaborate top_design

# UPF constraints
load_upf upf_constraints.upf

# Timing constraints
source timing_constraints.tcl

extract_physical_constraints -allow_physical top.def

compile_ultra -spg -scan -gate_clock

change_names -rule verilog -hierarchy
set write_sdc_output_net_resistance false
set write_sdc_output_lumped_net_capacitance false
write_sdc top.sdc
save_upf top.upf
write_file -f verilog -hierarchy -output top.v
start_icc_dp -f icc_dp.tcl
```

icc_dp.tcl Script

The icc_dp.tcl script generates the standard cell placement so you can reuse it during the `compile_ultra -incremental -spg` step.

The icc_dp.tcl script consists of the following commands:

```
write_def -version 5.7 -component -verbose -output std.def
exit
```

compile_incr.tcl Script

The compile_incr.tcl script reads in the Verilog netlist after DFT insertion and reads the placement from the first `compile_ultra -spg` run so that the incremental step can use the standard cell placement as seed placement.

```
# Reading the DFT netlist
read_verilog top.dft.v

# UPF
load_upf top.upf

# Timing constraints
read_sdc top.sdc

# Restoring floorplan constraints and standard cells placement from
# compile_ultra -spg step
extract_physical_constraints -allow_physical top.def
extract_physical_constraints -allow_physical -standard_cell spg std.def

# Reading the SCANDEF file from DFT insertion step
set dct_placement_ignore_scan true
read_scan_def dft.scandef

compile_ultra -spg -incremental

write_scan_def -output top.incr.scandef
write_file -f ddc -hierarchy -output top.incr.ddc
```

The `-standard_cell spg` option of the `extract_physical_constraints` command directs Design Compiler to read the standard cell placement in addition to the floorplan constraints. The `compile_ultra -incremental -spg` command will use that information as the seed placement during the placement optimization.

If you do not use the `-standard_cell spg` option, Design Compiler is forced to redo a full placement. This can hurt runtime and correlation because the incremental step cannot use the standard cell placement from the original `compile_ultra -spg` run.

After you run the `compile_ultra -incremental -spg` command step, you can save the design in .ddc format, along with the SCANDEF file, and proceed with physical implementation in IC Compiler with the `place_opt -spg -optimize_dft` command.

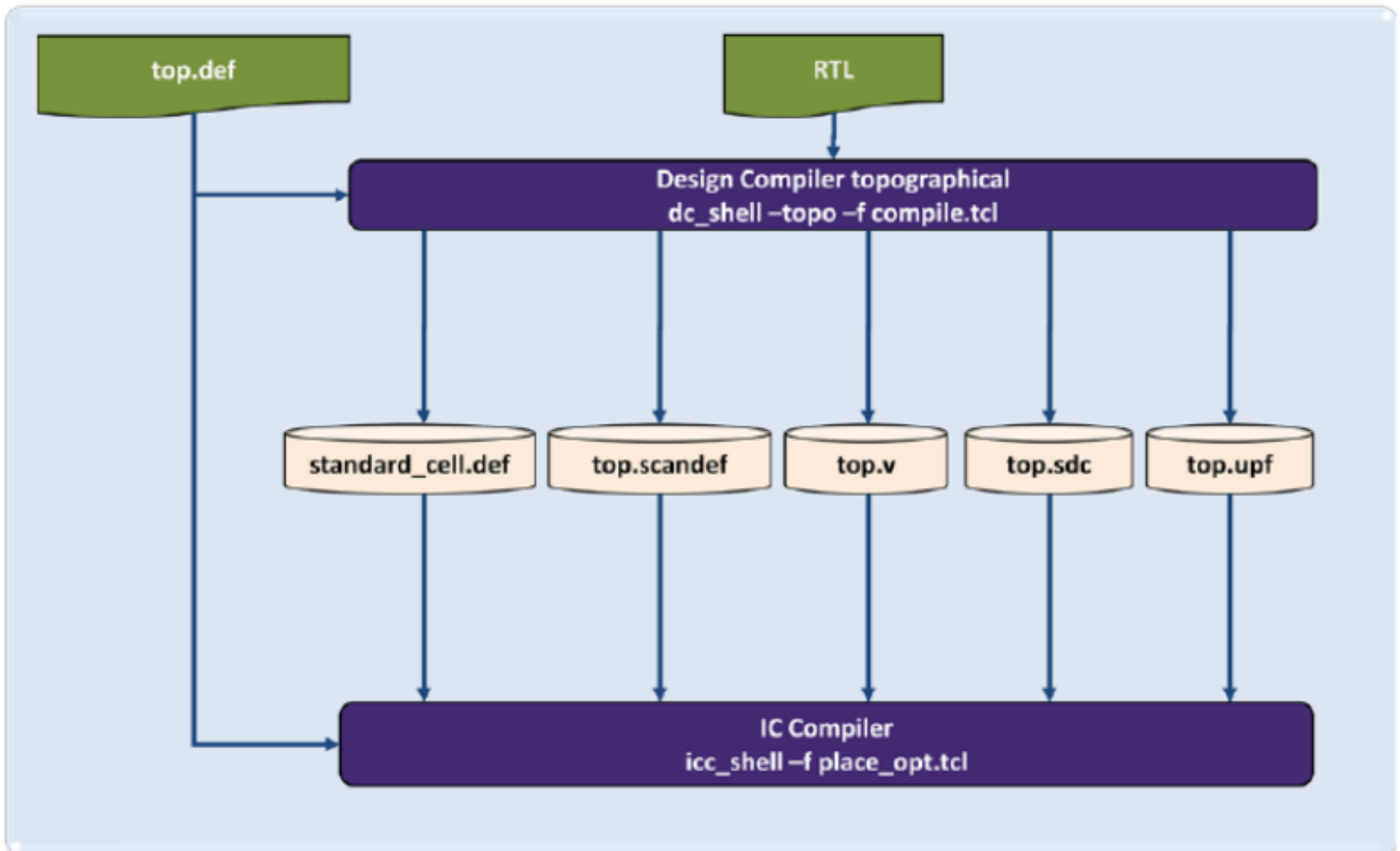
Note: The flow presented here uses the `-spg` option for `compile_ultra` and `compile_ultra -incremental`. However, writing out the placement of standard cells in order to reuse the information when reading back an ASCII netlist is not limited to the physical guidance flow. You can use the incremental ASCII flow without physical guidance enabled. If you are not using a physical guidance flow, simply omit the `-spg` option when you run the `compile_ultra` and `compile_ultra -incremental` commands and use the `-standard_cell topo` option with the `extract_physical_constraint` command instead of the `-standard_cell spg` option.

Physical Guidance ASCII Flow Handoff from Design Compiler to IC Compiler

Using a DEF file that contains the standard cell placement from Design Compiler allows flows that require an ASCII handoff from Design Compiler to IC Compiler to benefit from physical guidance.

Similar to the incremental ASCII flow described in the previous section, Design Compiler Graphical floorplan exploration is used in the following flow to generate the DEF file containing the standard cell placement information. IC Compiler then reads that information when you run the `place_opt -spg` command.

Figure 8: Physical Guidance ASCII Flow Handoff Flow



compile.tcl Script

The following compile.tcl script generates the files that are used in IC Compiler to constrain the design. Because this is an ASCII flow, you need to generate these files along with the Verilog top.v netlist:

- SDC constraints: top.sdc
- UPF constraints: top.upf
- SCANDEF file: top.scandef

The icc_dp.tcl script generates the standard_cell.def file.

```
# Analyze/elaborate step
analyze -f verilog top_design.v
elaborate top_design

# UPF constraints
load_upf upf_constraints.upf

# Timing constraints
source timing_constraints.tcl

extract_physical_constraints -allow_physical top.def

compile_ultra -spg -scan -gate_clock

# DFT constraints
source dft_constraint.tcl

create_test_protocol
insert_dft

compile_ultra -incremental -scan -spg

change_names -rule verilog -hierarchy
set write_sdc_output_net_resistance false
set write_sdc_output_lumped_net_capacitance false
write_sdc top.sdc
save_upf top.upf
write_scan_def -output top.scandef
write_file -f verilog -hierarchy -output top.v
start_icc_dp -f icc_dp.tcl
```

icc_dp.tcl Script

The following icc_dp.tcl script generates the placement for the standard cells.

```
write_def -version 5.7 -component -output standard_cell.def
exit
```

The script generates only the component section in the DEF file that contains the standard cell placement information. All other physical constraints used for IC Compiler come from the golden DEF file used in Design Compiler (top.def).

place_opt.tcl Script

In this script, the Verilog netlist (top.v) is read in IC Compiler along with all other files generated by the compile.tcl script.

The script uses the `spg_enable_ascii_flow` variable set to `true` to enable the ASCII physical guidance flow in IC Compiler. Without this setting, the `place_opt -spg` command would quit with an error because the design would not be considered ready for physical guidance.

```
# Reading the netlist
read_verilog top.v

# UPF constraints
load_upf top.upf

# Timing constraints
read_sdc top.sdc

# Enable the SPG ASCII Flow in IC COmpiler
set spg_enable_ascii_flow true

# Reading the golden floorpaln
read_def top.def

# Reading the standard cells placement from Design Compiler
read_def standard_cell.def

# Reading the SCANDEF file from Design Compiler
read_def top.scandef

place_opt -spg -congestion -area_recovery
```

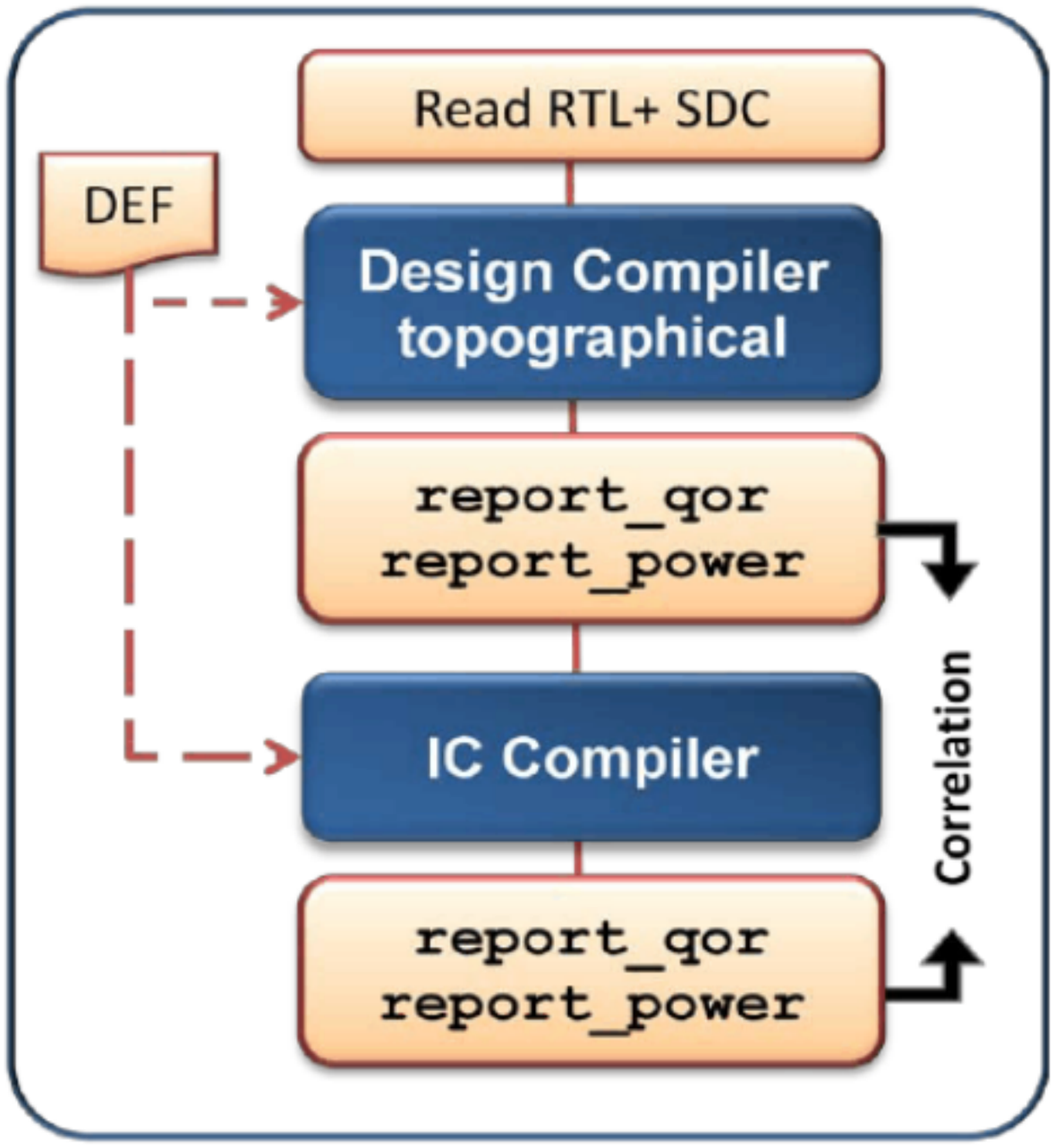
In this script, the `place_opt -spg` command considers the standard cell placement defined in the standard_cell.def file as physical guidance to be used as seed placement.

Reporting Options

For correlation-result comparison between the non-physical guidance and physical guidance flows, you should compare post-synthesis results against respective post-placement results for each flow. However, you should measure and compare post-placement results instead of post-synthesis results when looking at overall quality of results comparison between the non-physical guidance and physical guidance flows.

The following figure shows how correlation results are measured between Design Compiler and IC Compiler.

Figure 9: Correlation Measurement Between Design Compiler and IC Compiler



You can use the following set of equations to measure and compare correlation results for the worst negative slack (WNS) for each path group between Design Compiler and IC Compiler for non-physical guidance and physical guidance flows.

Table 6: Slack Correlation Measurement Between Design Compiler Topographical and IC Compiler

Path group	Period	Design Compiler		IC Compiler		WNS % Correlation
WNS	Clk_perio	WNS_{DCT}	TNS_{DCT}	WNS_{ICC}	TNS_{ICC}	$100 \times \left(\frac{\min(WNS_{ICC}, 0) - \min(WNS_{DCT}, 0)}{Clk\_period} \right)$

For area correlation measurements between Design Compiler and IC Compiler, you can use the set of equations listed in the following table for non-physical guidance and physical guidance flows.

Table 7: Area Correlation Measurement Between Design Compiler Topographical and IC Compiler

	Design Compiler	IC Compiler	% Correlation
Area	$Area_{DCT}$	$Area_{ICC}$	$100 \times \left(\frac{Area_{DCT}}{Area_{ICC}} - 1 \right)$

For performance and capacity, you can use the set of equations listed in the following table to assess the overall runtime improvement and memory utilization for non-physical guidance and physical guidance flows.

Table 8: Runtime and Memory Measurements Equations

Flow	Design Compiler	IC Compiler	Total
Runtime (CPU)	$Runtime_{DCT}$	$Runtime_{ICC}$	$Runtime_{DCT} + Runtime_{ICC}$
Memory	Mem_{DCT}	Mem_{ICC}	$\max(Mem_{DCT}, Mem_{ICC})$
Flow	**Non-Physical Guidance Flow**	**Physical Guidance Flow**	**% Improvement**
Total Runtime / Memory	$Total_{Non-SPG}$	$Total_{SPG}$	$100 \times \left(1 - \frac{Total_{SPG}}{Total_{Non-SPG}} \right)$

For power measurement, use the `report_power` command to report the total power usage of the design (leakage and dynamic) in Design Compiler and IC Compiler. With the power prediction mode enabled, the total power usage reported by the `report_power` command includes the power usage attributed to clock-tree estimates.

For congestion reporting, use the `report_congestion` command to report routing congestion estimation in Design Compiler topographical and IC Compiler. In addition to the textual congestion reporting, you should also look at the congestion map in order to fully assess the severity of design congestion status.

Known Limitations

The following limitations are known physical guidance flow limitations:

- In Design Compiler, the `compile_ultra` command does not support the `-top` option in the physical guidance flow.
- In IC Compiler, the `place_opt_feasibility` command does not support the physical guidance `-spg` option.
- IC Compiler, the `place_opt` command does not support low-effort level placement in the physical guidance flow. As a result, you cannot use the `-effort low` setting in conjunction with the `place_opt -spg` command.
- You cannot combine ILMs and block abstractions in the same flow.
- IC Compiler does not support ILMs and block abstractions that were created by Design Compiler. Therefore, in the hierarchical flow between Design Compiler and IC Compiler, you must use ILMs or block abstractions that were created by IC Compiler.

Conclusion

The physical guidance flow is jointly supported in Design Compiler and IC Compiler in order to improve QoR, runtime, correlation, and routability. The physical guidance flow is enabled with the `-spg` option with the `compile_ultra` command in Design Compiler and with the `-spg` option with the `place_opt` command in IC Compiler. The `-spg` option works seamlessly with existing `compile_ultra` options in Design Compiler and with the `place_opt` options in IC Compiler.

In the physical guidance flow, Design Compiler performs further refined placement that is consistent with the IC Compiler `place_opt` command functionality and uses enhanced post-placement delay optimization in order to provide a better optimized starting point for physical implementation. The placement information is passed to the IC Compiler `place_opt` command and is used as seed placement for physical implementation of the design. Thus, placement is aligned between Design Compiler and IC Compiler, resulting in improved runtime, QoR, and correlation.

Using the physical guidance flow, the Design Compiler to IC Compiler handoff also includes cell placement information and results in correlation within 5%.

In addition, physical guidance provides significant speedup of the IC Compiler runtime because IC Compiler reuses the physical guidance information from Design Compiler topographical as its starting point and runs through fewer placement steps. As a result, you generally see improvement in the overall design turnaround time.