Hierarchical Flow Support in Synopsys® Design Compiler Topographical Mode

Application Note

Version A-2007.12-SP2, March 2008

Comments?

Send comments on the documentation by going to http://solvnet.synopsys.com, then clicking "Enter a Call to the Support Center."

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

"This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____."

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, AMPS, Cadabra, CATS, CRITIC, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSIM, HSPICE, iN-Phase, in-Sync, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PrimeTime, Saber, SiVL, SNUG, SolvNet, System Compiler, TetraMAX, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, Columbia, Columbia-CE, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, Direct Silicon Access, Discovery, Encore, Galaxy, Gatran, HANEX, HDL Compiler, Hercules, Hierarchical Optimization Technology, HSIM^{plus}, HSPICE-Link, iN-Tandem, i-Virtual Stepper, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Raphael-NES, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, Taurus, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (^{SM})

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

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.
All other product or company names may be trademarks of their respective owners.

Contents

Preface

This preface includes the following sections:

- About This Application Note
- Customer Support

About This Application Note

The preferred synthesis flow in topographical mode is a top-down strategy. However, there was an increasing demand to support the bottom-up flow in order to address design and flow challenges. This application note describes various bottom-up flow configurations that are currently supported in Design Compiler topographical mode.

Audience

This application note is intended to enable current Design Compiler users to easily migrate their existing bottom-up synthesis scripts to topographical mode with minimal script changes and without needing to convert to a top-down methodology.

Related Publications

For additional information about Design Compiler topographical technology and hierarchical flow support, see

- Design Compiler User Guide, Chapter 8
- *Reference Methodology* for recommended scripts
- DFT Compiler User Guide: Scan and
  DFT MAX User Guide: Scan
- Documentation on the Web, which is available through SolvNet at
  http://solvnet.synopsys.com/

You might also want to refer to the documentation for the following related Synopsys products:

- Design Compiler
- DFT Compiler
- IC Compiler
- PrimeTime

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax.
`Courier italic`	Indicates a user-defined value in Synopsys syntax, such as *`object_name`*. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
`Courier bold`	Indicates user input—text you type verbatim— in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[ ]	Denotes optional parameters, such as *`pin1 [pin2 ... pinN]`*
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and "Enter a Call to the Support Center."

To access SolvNet,

1. Go to the SolvNet Web page at http://solvnet.synopsys.com.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)

If you need help using SolvNet, click HELP in the top-right menu bar or in the footer.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to http://solvnet.synopsys.com (Synopsys user name and password required), then clicking "Enter a Call to the Support Center."
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at http://www.synopsys.com/support/support_ctr.
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
- Find other local support center telephone numbers at http://www.synopsys.com/support/support_ct

Hierarchical Flows Supported in Synopsys Design Compiler Topographical Mode

This document describes the recommendations for a hierarchical or bottom-up flow in Design Compiler topographical mode.

In general, a top-down methodology produces the best results in Design Compiler and Design Compiler topographical mode. However, there is an increasing interest in a bottom-up flow in order to address design and flow challenges. For example, a hierarchical flow will enable you to

- Manage capacity and runtime for multi-million instance designs with smaller geometry processes
- Employ a divide-and-conquer design methodology through physical implementation
- Accommodate design tasks that are split across multiple development teams and sites
- Easily integrate designs involving third-party hard or soft IP blocks requiring customized synthesis techniques

Note that a hierarchical flow in Design Compiler topographical mode might impact quality of results (QoR) and correlation to the back-end, when compared to the top-down flow or when the front-end logical partition is different from the back-end physical partition.

This document describes various bottom-up flow methodologies that are currently supported in Design Compiler topographical mode. After reading this document you will be able to easily migrate your existing Design Compiler bottom-up synthesis scripts to topographical mode with minimal changes and without needing to convert to a top-down methodology.

This document includes the following sections:

- IC Compiler Reference Methodology (ICC-RM)
- Creating the Initial Netlist Using Design Compiler Topographical Mode
  - Top-Down Synthesis Option
  - Bottom-Up Synthesis Option
- Creating the Initial Floorplan Using IC Compiler
- Detailed Hierarchical Implementation Using Design Compiler Topographical Mode
- IC Compiler - Detailed Floorplan
- IC Compiler - Detailed Block and Top Level Design Closure

- Design Compiler Topographical Mode Hierarchical Flow
  - Block Level Design Implementation
  - Top-Level Design Integration with Design Compiler Topographical Mode .ddc Netlist Subblocks
  - Top-Level Design Integration with Design Compiler Topographical Mode ILM Subblocks
  - Top-Level Design Integration with IC Compiler ILM Subblocks
- Support for Design-For-Test
- Support for Power
- Conclusion

Overview

In a hierarchical flow, subblocks are independently synthesized as separate design units. Then the resultant mapped subblocks are included in subsequent synthesis runs of higher-level blocks; this process continues up to the top-level.

After a subblock is compiled, it can be saved as a full .ddc design netlist, or to further reduce design database size, as an abstract interface logic model (ILM). Alternatively, you can create an abstract ILM model in IC Compiler after physical implementation.

During top-level design integration and synthesis, a subblock is represented by either the abstract ILM model generated in Design Compiler topographical mode, the abstract ILM model generated in IC Compiler, or the mapped .ddc of the full netlist generated in Design Compiler topographical mode. The subblock is treated as a fixed physical block and is not touched during top-level synthesis. The subblock's logical structure and virtual placement (for the Design Compiler topographical mode generated abstract ILM model or .ddc netlist) or true placement (for the IC Compiler generated abstract ILM model) is propagated to the top level and preserved during top-level synthesis.

When synthesizing the top-level design, you can perform a full compile by using the **`compile_ultra`** command in order to map any top-level unmapped logic elements and integrate the physical subblocks. A full compile enables the Design Compiler topographical mode to effectively map and optimize the design, producing a better implementation netlist.

When you use the **`compile_ultra`** command at the top level, mapping and optimization are virtual placement-driven. Although the macro placer can place the physical subblocks just like any other macros during virtual placement, it is better if you provide fixed placement locations that match IC Compiler. This helps to maintain quality of results (QoR) and correlation to IC Compiler.

Design Compiler Topographical Mode to IC Compiler Hierarchical Flow

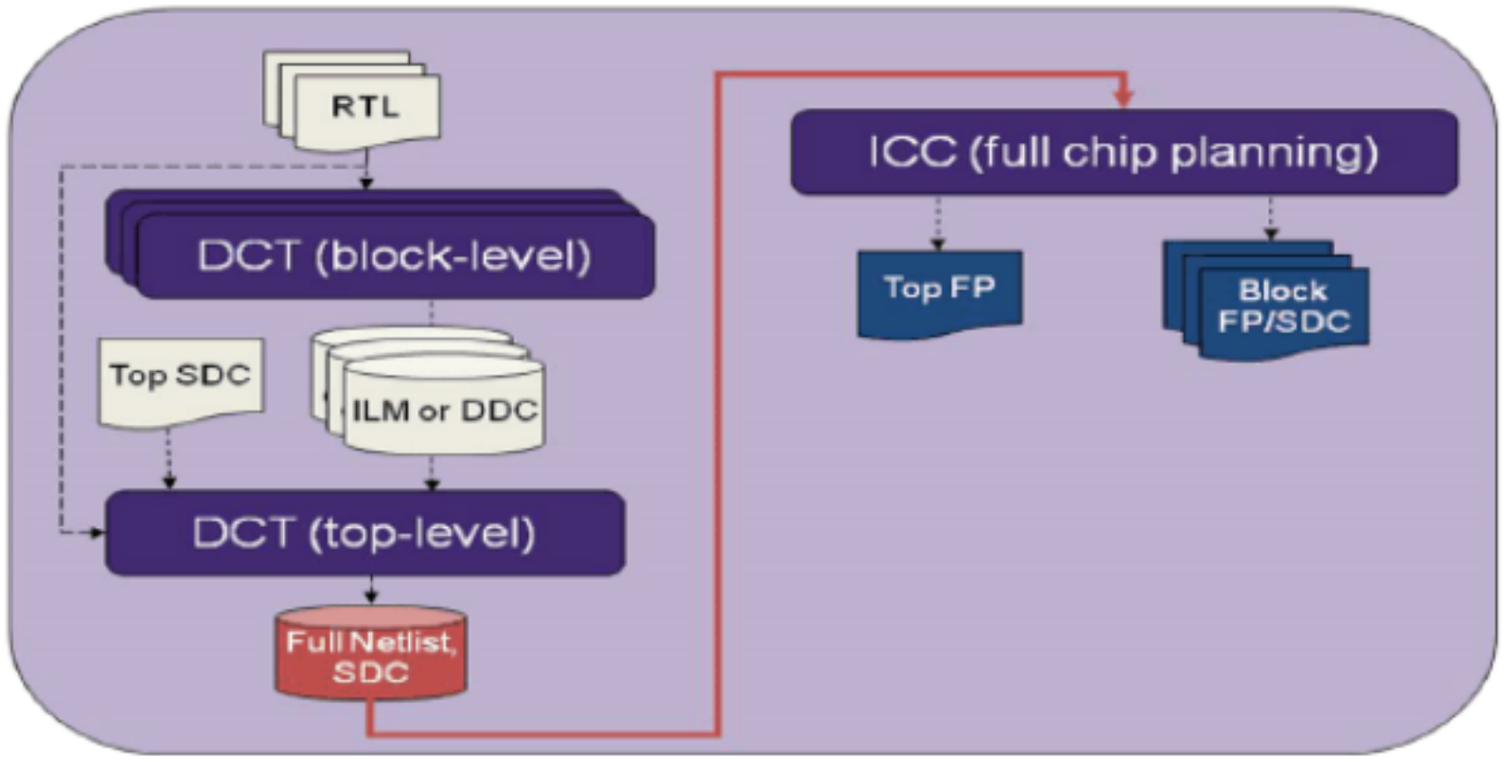
Design Compiler topographical mode and IC Compiler provide a complete hierarchical methodology support for you to address your design and flow challenges. Design Compiler topographical mode to IC Compiler hierarchical flow is a two stageflow.

Full chip planning is performed during the first stage of the hierarchical flow in order to generate the hierarchical physical constraints and budgeted block-level timing constraints. Detailed design implementation is performed during the second stage of the hierarchical flow.

Note that if the block-level timing constraints and hierarchical physical constraints are available, you can skip the first stage of the hierarchical flow and proceed directly to the detailed design implementation stage.

Figure 1 show the steps needed to be performed during the first stage of the Design Compiler topographical mode to IC Compiler hierarchical flow. You need to create the initial netlist in Design Compiler topographical mode for IC Compiler full chip planning. The outputs of this stage are the hierarchical (top-level and block-level) floorplans and the SDC timing constraints for subblocks.

Figure 1 First Stage of Design Compiler Topographical Mode to IC Compiler Hierarchical Flow Diagram



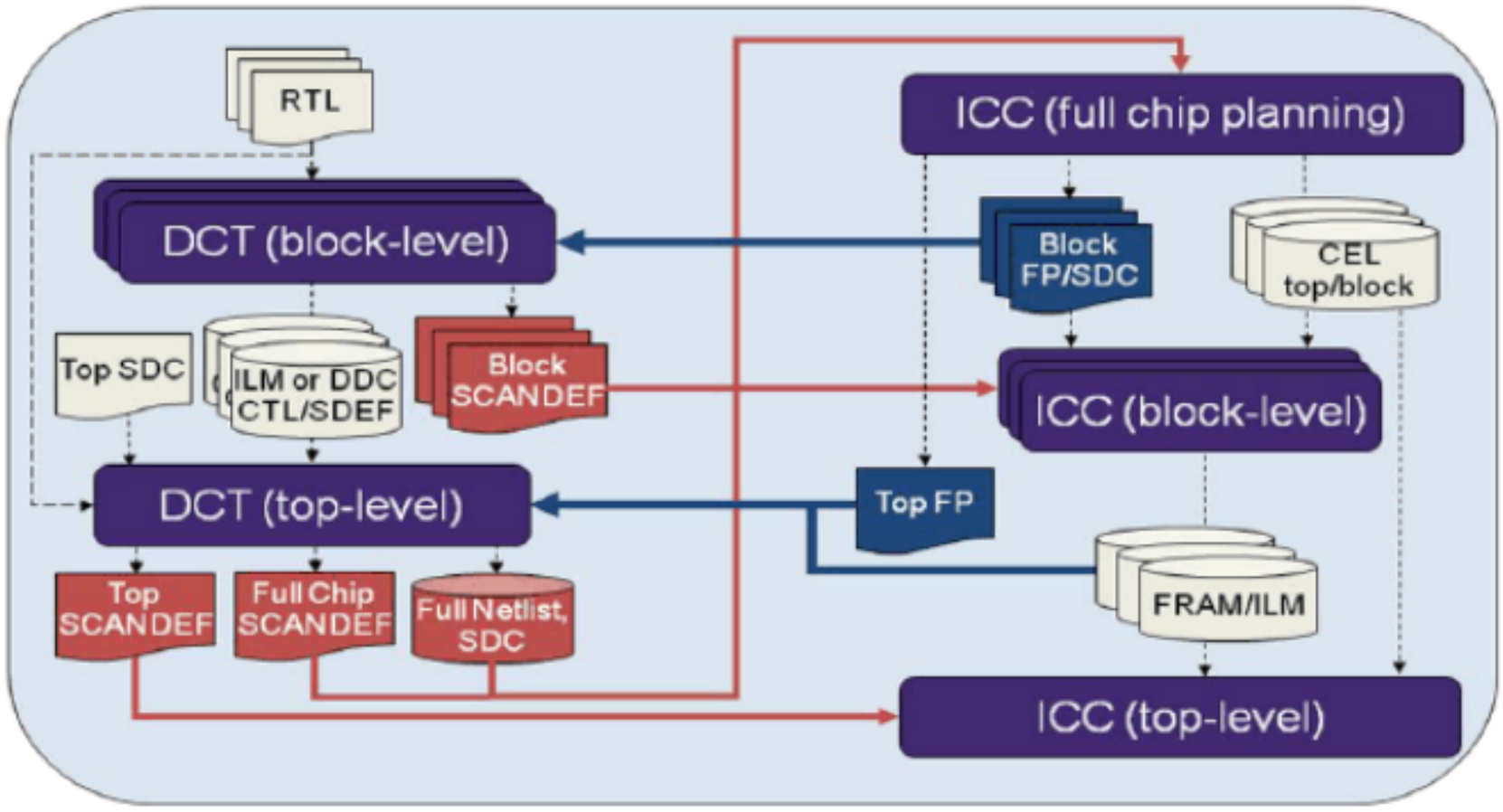
As shown in Figure 1, the first stage of the Design Compiler topographical mode to IC Compiler hierarchical flow requires the following steps:

1. First stage of the hierarchical flow:
    a. Create the initial full design netlist in topographical mode in order to generate a design hierarchical floorplan. See "Creating the Initial Netlist by Using Design Compiler Topographical Mode."
    b. Proceed to the full-chip design planning phase in IC Compiler in order to generate a hierarchical floorplan with physical constraints and SDC timing constraints. See "Creating Initial Floorplan Using IC Compiler."

With the hierarchical floorplans and block-level SDC timing constraints that you generate in the first stage of hierarchical flow, you can synthesize the design starting from HDL and using bottom-up flow in Design Compiler topographical mode. You can then proceed to refine your full chip planning step in IC Compiler before performing detailed block-level and top-level physical implementation in IC Compiler.

Figure 2 show the steps needed for detailed design implementation during the second stage of the Design Compiler topographical mode to IC Compiler hierarchical flow.

Figure 2 Second Stage of Design Compiler Topographical Mode to IC Compiler Hierarchical Flow Diagram



As shown in Figure 2, the second stage of Design Compiler topographical mode to IC Compiler hierarchical flow requires the following steps:

2. Second stage of the hierarchical flow:
    a. Complete a detailed design implementation in topographical mode using a bottom-up flow. See "Detailed Hierarchical Implementation Using Design Compiler Topographical Mode."
    b. Refine the design hierarchical floorplan in IC Compiler. See "IC Compiler - Detailed Floorplan."
    c. Perform detailed block-level and top-level design closure in IC Compiler. See "IC Compiler - Detailed Block and Top Level Design Closure."

Steps 1a and 2a, which involve initial design netlist creation and detailed design implementation in Design Compiler topographical mode, are discussed in the rest of this application note.

Steps 1b, 2b and 2c, which involve physical implementation in IC Compiler, are covered in the following IC Compiler Hierarchical Design Methodology application note: https://solvnet.synopsys.com/retrieve/023130.html

Design Compiler Reference Methodology (DC-RM)

The Design Compiler reference methodology (DC-RM) includes a complete set of scripts to demonstrate the recommended synthesis flow in Design Compiler and Design Compiler topographical mode.

You can download the DC-RM scripts from SolvNet using the following link:
https://solvnet.synopsys.com/retrieve/021023.html

Beginning in the A-2007.12-SP1 version of the DC-RM, reference scripts are also included for a hierarchical flow. This application note will refer to scripts from the DC-RM which can be consulted as complete examples. You may want to download and study the DC-RM scripts in conjunction with the information presented in this application note.

The following scripts are provided as part of DC-RM:

`common_setup.tcl` - Common design setup variables for Design Compiler and IC Compiler reference methodologies

`dc_setup.tcl` - Library setup for DC-RM

`dc_scripts/dc.tcl` - DC-RM script for block-level synthesis in a hierarchical flow

`dc_scripts/dc_top.tcl` - DC-RM top-level integration script for hierarchical flow

IC Compiler Reference Methodology (ICC-RM)

The IC Compiler reference methodology (ICC-RM) is also available from Synopsys. The IC Compiler-RM includes a complete set of scripts to demonstrate the recommended hierarchical physical implementation flow in IC Compiler.

You can download the ICC-RM scripts from SolvNet using the following link:
https://solvnet.synopsys.com/retrieve/021624.html

The IC Compiler reference methodology provides scripts for the following physical implementation steps:

- Hierarchical design planning phase
- Detailed block-level implementation phase
- Detailed top-level design closure phase

Creating the Initial Netlist Using Design Compiler Topographical Mode

The first step in a hierarchical synthesis methodology is the creation of the initial full design netlist in Design Compiler topographical mode. This initial netlist enables you to perform full chip planning in IC Compiler.

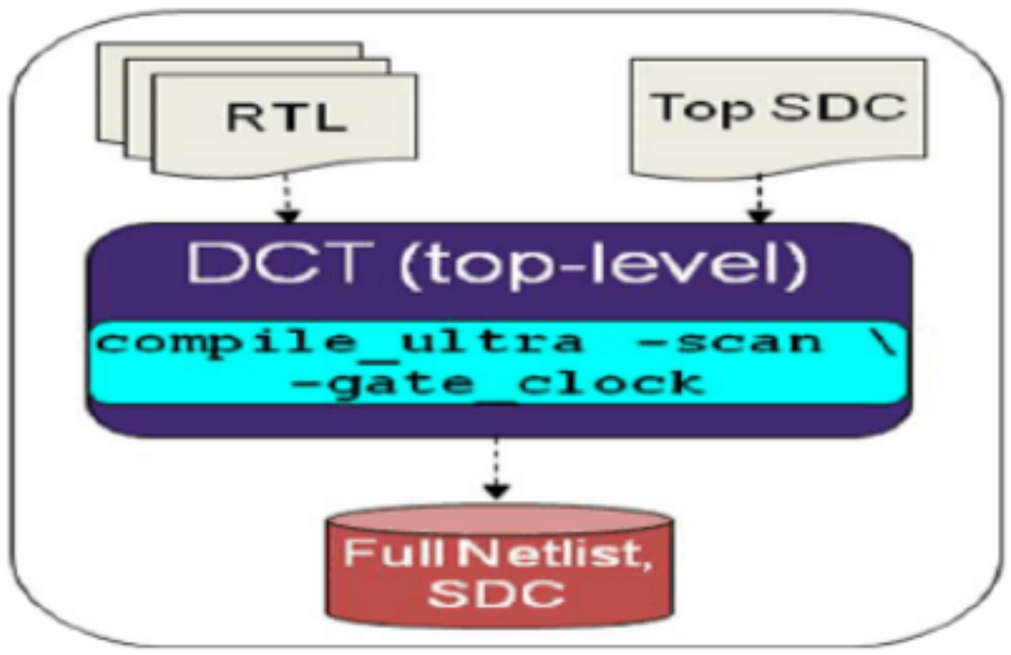
You have the option of using either a top-down synthesis strategy or a bottom-up synthesis strategy in creating the initial netlist to use for the chip planning in IC Compiler.

Top-Down Synthesis Option:

The simplest method for creating the initial netlist for a hierarchical flow is to perform a top-down flow, enabling you to use the full virtual placement aware capabilities of Design Compiler topographical mode to drive design mapping and optimization. Floorplan is not required for this step as Design Compiler derive a default floorplan including port location, macro placement and placement area. If floorplan information is available, it can be used to optimize the design.

Figure 3 show the steps to perform the top-down option in creating the initial design netlist for IC Compiler full chip planning.

Figure 3 Top-Down Synthesis Option Flow Diagram



Follow these steps to create the initial netlist:

a. Perform design-specific setup and specify the logic and physical libraries.
b. Read in the full design source files (Verilog, VHDL, netlist, or .ddc).
c. Set the current design to the top-level design and link the top-level design.
d. Apply timing constraints and power constraints.
e. (Optional) Provide any top-level physical constraints.
f. Perform a test-ready and clock-gating compile of the top-level design by using the **`compile_ultra -scan -gate_clock`** command.
g. Save the full design netlist in the .ddc format.

Notes:

1. Since the initial design netlist is created for hierarchical design floorplanning, the top level design may be optionally compiled with any applicable physical constraints.
2. Use the **-scan** option of the **compile_ultra** command to ensure that only scan registers are used in the initial design netlist, but do not perform scan stitching with **insert_dft** until after initial design planning in IC Compiler is complete. Inserting scan chains before the initial design hierarchical floorplanning will add extra module-level ports that will prevent you from starting block-level detailed implementation with RTL source.

3. Use **set_boundary_optimization** to disable boundary optimization across subblocks or logical hierarchical level that will eventually be partitioned to become a physical hierarchy during the planning step in IC Compiler.
4. Use **set_ungroup** to disable automatic ungrouping of lower-level blocks to ensure that the boundary interface matches the RTL. This will allow you to start the block-level detailed implementation directly from the RTL source.

Bottom-Up Synthesis Option:

If the size of the design precludes a top-down synthesis option, a hierarchical flow can be used in which subblocks are synthesized in parallel and then integrated at the top level with top-level glue logic.

For top-level integration, use either the **compile_ultra -scan -gate_clock** or **compile_ultra -scan -gate_clock -top** command, depending on the amount of unmapped glue logic at the top level.

If there is a large amount of unmapped glue logic at the top level, use the full optimization capabilities of the **compile_ultra** command to synthesize the top-level design in order to get the best initial netlist for design planning in IC Compiler.

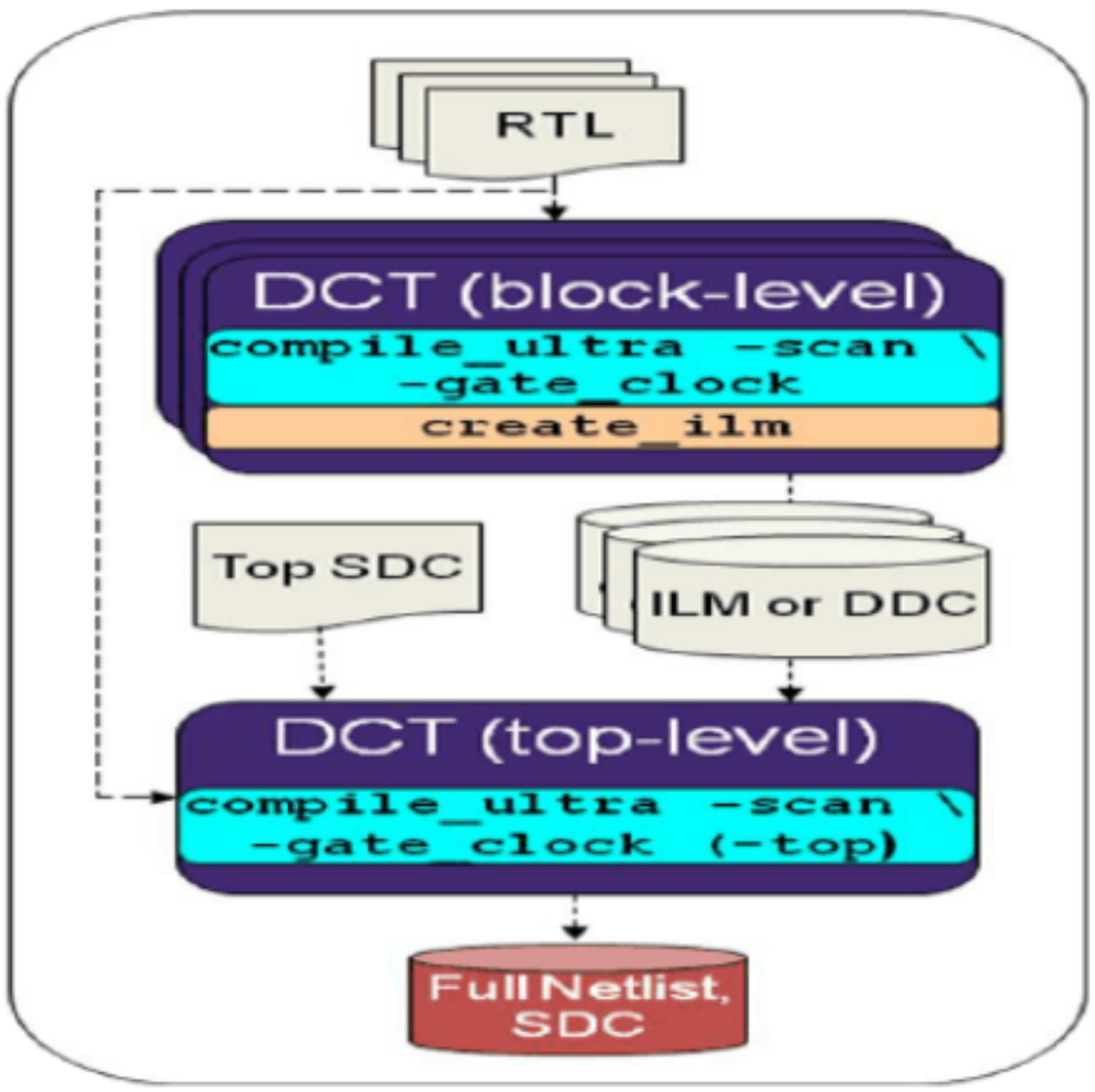
On the other hand, if the top-level consists primarily of mapped sub-blocks with minimal unmapped glue logic, use the **-top** option of the **compile_ultra** command to quickly integrate the mapped subblocks. The **-top** option supports the presence of unmapped, non-hierarchical glue logic at the top level, which is then mapped to appropriate technology library cells.

In topographical mode, the **compile_ultra -top** option automatically propagates block-level placement-aware back-annotated net delays and parasitics to the top level for logic mapping and optimization. However, block-level placements are not propagated to the top level and top-level implementation is not placement-driven.

Capacity at the top level can be improved by replacing the mapped sub-blocks with their DC-T ILMs.

Figure 4 illustrates the hierarchical flow to create the initial design netlist for IC Compiler full chip planning.

Figure 4 Bottom-Up Synthesis Option Flow Diagram



To create the initial netlist using the bottom-up synthesis option, follow these steps.

1. Follow these steps at the lower-level block:
   a. Perform design-specific setup and specify the logic and physical libraries.
   b. Read in the subblock (Verilog, VHDL, netlist, or .ddc).
   c. Set the current design to subblock and link the subblock.
   d. Apply block-level timing constraints (if available) and power constraints.
   e. Perform a test-ready and clock-gating compile of the subblock by using the **`compile_ultra -scan -gate_clock`** command.
   f. Use the **`uniquify_naming_style`** variable to specify unique naming convention to be used by the uniquify command and the **`uniquify -force`** command to uniquify the subblock in order to avoid naming conflicts during top-level design integration.
   g. (Optional) Use create_ilm command to create an abstract ILM model of the subblock in order to improve capacity at the top level.
   h. Save the mapped subblock in .ddc format.

2. Follow these steps at the top level:

a. Perform design-specific setup and specify the logic and physical libraries.
b. Read in the full-chip source files (Verilog, VHDL, netlist, or .ddc).
c. Remove the subblocks that have been mapped in Step 1 from memory.
d. Read in the .ddc netlist for mapped subblocks.
e. Set current design to the top-level design and link it.
f. Apply top-level timing constraints and power constraints.
g. (Optional) Provide any top-level physical constraints.
h. Perform a test-ready and clock-gating compile of the top-level by using either the **compile_ultra -scan -gate_clock** or **compile_ultra -scan -gate_clock -top** comman,d depending on the amount of unmapped glue logic at the top level.
i. If top-level contain abstract ILM models, remove them.
j. Save the full design netlist in the .ddc format.

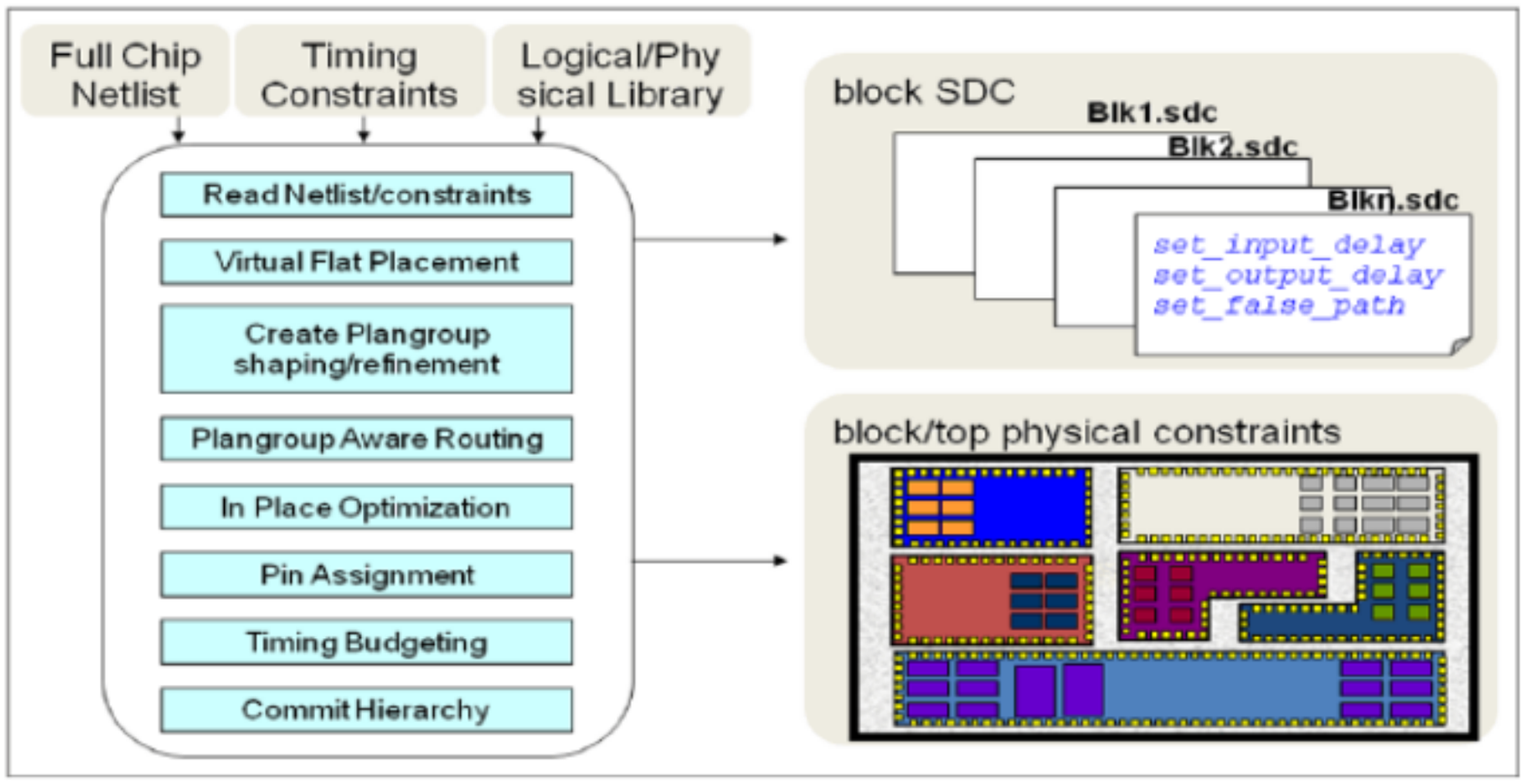
Notes:

1. Since the initial design netlist is created for hierarchical design floorplanning, you should compile subblocks without floorplan information (which is not available yet), while the top level may be optionally compiled with any applicable physical constraints.
2. Use the -**scan** option of the **compile_ultra** command to ensure that only scan registers are used in the initial design netlist, but do not perform scan stitching with **insert_dft** until after initial design planning in IC Compiler is complete. Inserting scan chains before the initial design hierarchical floorplanning will add extra module-level ports in the design's hierarchical floorplans and SDC timing constraints that are not in the RTL source. The extra module-level ports will hamper the flow when starting block-level detailed implementation with RTL source.
3. Use **set_boundary_optimization** to disable boundary optimization across subblocks or logical hierarchical level that will eventually be partitioned to become a physical hierarchy during the planning step in IC Compiler.
4. Use **set_ungroup** to disable automatic ungrouping of lower-level blocks to ensure that the boundary interface matches the RTL. This will allow you to start the block-level detailed implementation directly from the RTL source.
5. If you are using the **-top** option with the **compile_ultra** command, set the **dont_touch** attribute on the mapped subblocks to prevent top-level mapping and optimization from re-optimizing the subblock interface logic structures.
6. Remove all the abstract ILM subblocks before saving the top-level design netlist.

Creating the Initial Floorplan Using IC Compiler

Once the initial design netlist is generated, you can begin full chip design planning in IC Compiler. Figure 5 shows the steps required for full-chip design planning in order to partition the design and allocate top-level timing budgets to lower-level physical blocks.

Figure 5 Full Chip Design Planning Flow in IC Compiler



As shown in Figure 5, this phase produces block-level and top-level floorplans and SDC timing constraints that are used to perform detailed design implementation in Design Compiler topographical mode.

Note that detailed implementation is simplified significantly if all the physical partitions have the corresponding HDL logic boundaries during the physical partitioning steps. This will allow you to start the detailed implementation stage with HDL in Design Compiler topographical mode.

Once you have completed the design planning stage, it is highly recommend that you proceed to the detailed hierarchical implementation phase in Design Compiler topographical mode to re-implement the design starting from HDL using the block-level floorplans and timing constraints. This will ensure that an optimum implementation netlist is generated for each subblock and the top-level to meet your design goals.

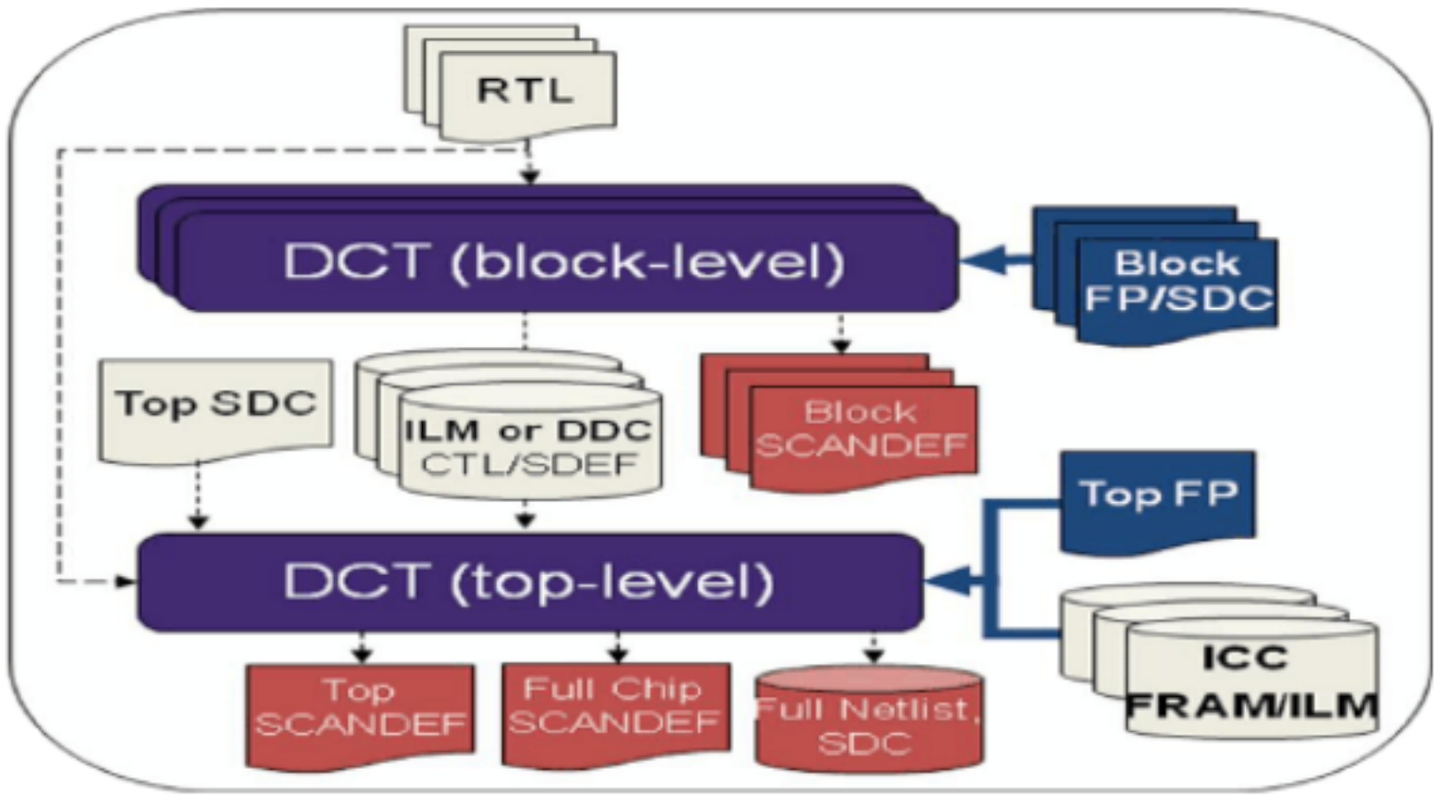
For detailed information on the various steps needed to generate design's hierarchical floorplan and budgeted timing constraints, refer to IC Compiler Hierarchical Design Methodology application note which you can download from SolvNet using the following link: https://solvnet.synopsys.com/retrieve/023130.html.

Refer to IC Compiler Reference Methodology (ICC-RM) for a comprehensive script example.

Detailed Hierarchical Implementation Using Design Compiler Topographical Mode

After creating the hierarchical floorplans and budgeted block-level SDC timing constraints, you can proceed to the detailed hierarchical implementation phase.

Figure 6 Design Compiler Topographical Mode Detailed Implementation Flow Diagram



As can be seen in Figure 6, during top-level integration a mapped subblock can be represented in three ways:

- A .ddc netlist generated in Design Compiler topographical mode
- An interface logic model (ILM) generated in Design Compiler topographical mode
- An interface logic model (ILM) generated in IC Compiler

These three ways of representing the subblocks during top-level design integration imply three corresponding Design Compiler topographical mode top-level integration flows:

- Design Compiler topographical mode .ddc netlist top-level integration flow
- Design Compiler topographical mode ILM top-level integration flow
- IC Compiler ILM top-level integration flow.

Note that you can integrate a mixture of any of the subblock representations together at the top level in Design Compiler topographical mode.

	Improve Runtime	Improve Capacity	Support Flat Top-Level Placement
DCT DDC Netlist	Yes	No	Yes
DCT ILM	Yes	Yes	Yes
ICC ILM	Yes	Yes	N/A

Table 1 Comparison of Top-Level Integration Flows Supported in Design Compiler Topographical Mode

As shown in Table 1, all three types of Design Compiler topographical mode top-level integration flows provide runtime improvements when compared to a flat top-level flow as block-level synthesis can be independently run in parallel. Both the abstract model top-level integration flows provide further capacity improvements over the Design Compiler topographical mode .ddc netlist top-level integration flow as the abstract models are the reduced design size representation of the subblocks. Also, both the Design Compiler topographical mode generated ILM top-level integration and the Design Compiler topographical mode generated .ddc netlist top-level integration flow support flat top-level placement in IC Compiler. You can use whichever top-level integration that best suit your design and flow requirements.

One common factor among all three types of top-level integration flows supported in topographical mode is that top level mapping and optimization are virtual placement aware and block-level leaf objects placements and back-annotated parasitics are automatically propagated to the top level.

You can also specify a fixed location for a hierarchical physical subblock at the top level. The top-level hierarchical floorplan created in previous full chip design planning phase in IC Compiler should contain fixed location for the physical subblocks. You can then use the **`extract_physical_constraints`** command to apply these physical constraints in Design Compiler topographical mode. Otherwise, you can use the **`set_cell_location`** command in topographical mode to provide your user specified fixed location for the physical subblocks at the top level.

If any physical subblocks is missing a fixed location specification at the top level, the Design Compiler topographical mode virtual placer will end up placing the block while at the same time preserving the block-level logical structure and relative placements.

Note that you can encounter over-utilization issues if cell locations are not specified for physical subblocks.

For design-for-test (DFT) flow, you should use **`insert_dft`** command to connect block-level and top-level scan chains. Then use **`write_scan_def`** command to save the block-level, the top-level, and the full chip SCANDEF information in order to perform design placement, scan chain reordering and repartitioning in IC Compiler. The full chip SCANDEF information is needed during the detailed floorplanning stage in IC Compiler in order to prevent changes to the scan chain elements and order. You can see the "Support for Design-For-Test" section for more information.

For Power flow, you can set the appropriate power constraints in order to enable the respective power optimization during the block-level implementation and the top—level integration. You can see the "Supported for Power" section for more information.

To compile subdesigns independently, follow the required steps listed under "Block Level Design Implementation".

To perform top-level design integration with subblock models using abstract ILM models generated by IC Compiler, follow the required steps listed under "Top-Level Design Integration with IC Compiler ILM Subblocks".

To perform top-level design integration with subblock models using abstract ILM models generated by Design Compiler topographical mode, follow the required steps listed under "Top-Level Design Integration with Design Compiler Topographical Mode ILM Subblocks".

To perform top-level design integration using .ddc netlists generated by Design Compiler topographical mode, follow the required steps listed under "Top-Level Design Integration with Design Compiler Topographical Mode .ddc Netlist Subblocks".

IC Compiler - Detailed Floorplan

Once the detailed design netlist is generated in Design Compiler topographical mode, you can proceed to refine the design hierarchical floorplan in IC Compiler using the updated design netlist that now contain scan chains.

Note that this is a hierarchical floorplanning refinement steps needed to:

1. (Optional) Adjust the blocks sizes if necessary based on Design Compiler topographical mode optimization.
2. Place the additional pins resulting from the insertion of scan chains

Figure 7 IC Compiler Design Planning Flow

Figure 7 shows the steps required to refine the design hierarchical floorplan in IC Compiler. For more detailed information, refer to IC Compiler Hierarchical Design Methodology application note, which you can download from SolvNet using the following link: https://solvnet.synopsys.com/retrieve/023130.html.

Refer to IC Compiler Reference Methodology (ICC-RM) for a comprehensive script example.

Notes:

1. Load the full-chip SCANDEF information at the beginning of this design planning stage in order to prevent the scan chain elements from being re-optimized during the "In Place Optimization" step. Reading the full-chip SCANDEF information enables scan chain elements to have dont_touch or size_only attributes. As result, scan chain order is maintained through design planning stage.
2. During the physical partitioning steps, all the physical partitions should have the corresponding HDL logic boundaries so that you can start the detailed implementation stage with HDL in Design Compiler topographical mode.
3. Do not have feedthrough pins on scan net during pin assignment.
4. Keep the physical block hierarchies intact during hierarchical floorplanning refinement in order to maintain SCANDEF consistency with Design Compiler topographical mode generated top-level and block-level SCANDEFs.

IC Compiler - Detailed Block and Top Level Design Closure

After the design hierarchical floorplan has been refined, you can now perform detailed block-level and top-level implementation and design closure in IC Compiler.

Figure 8 IC Compiler Design Planning Flow

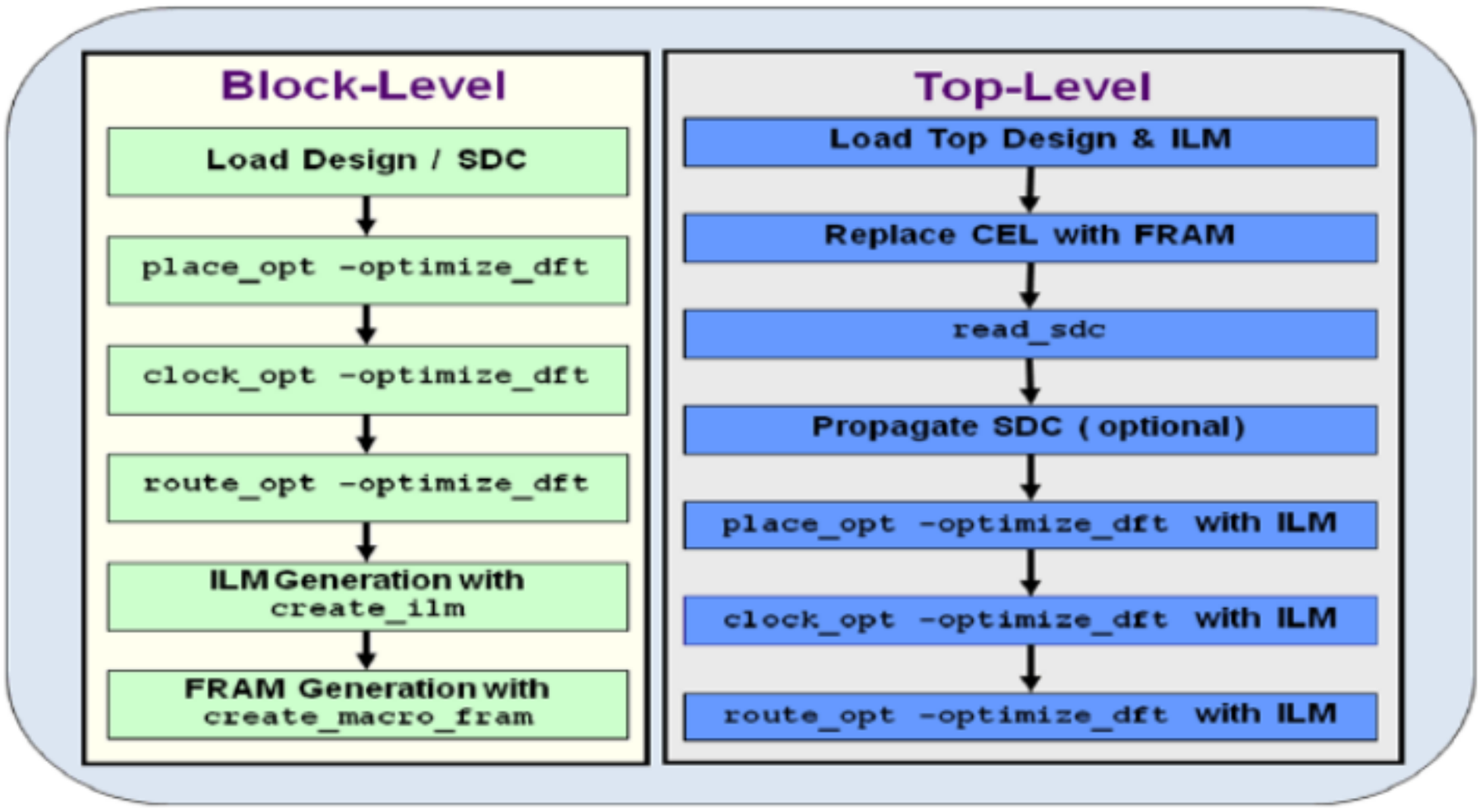


Figure 8 shows the various steps required for detailed block-level and top-level physical implementation in IC Compiler. For more detailed information, refer to IC Compiler Hierarchical Design Methodology application note which you can download from SolvNet using the following link: https://solvnet.synopsys.com/retrieve/023130.html.

Refer to IC Compiler Reference Methodology (ICC-RM) for a comprehensive script example.

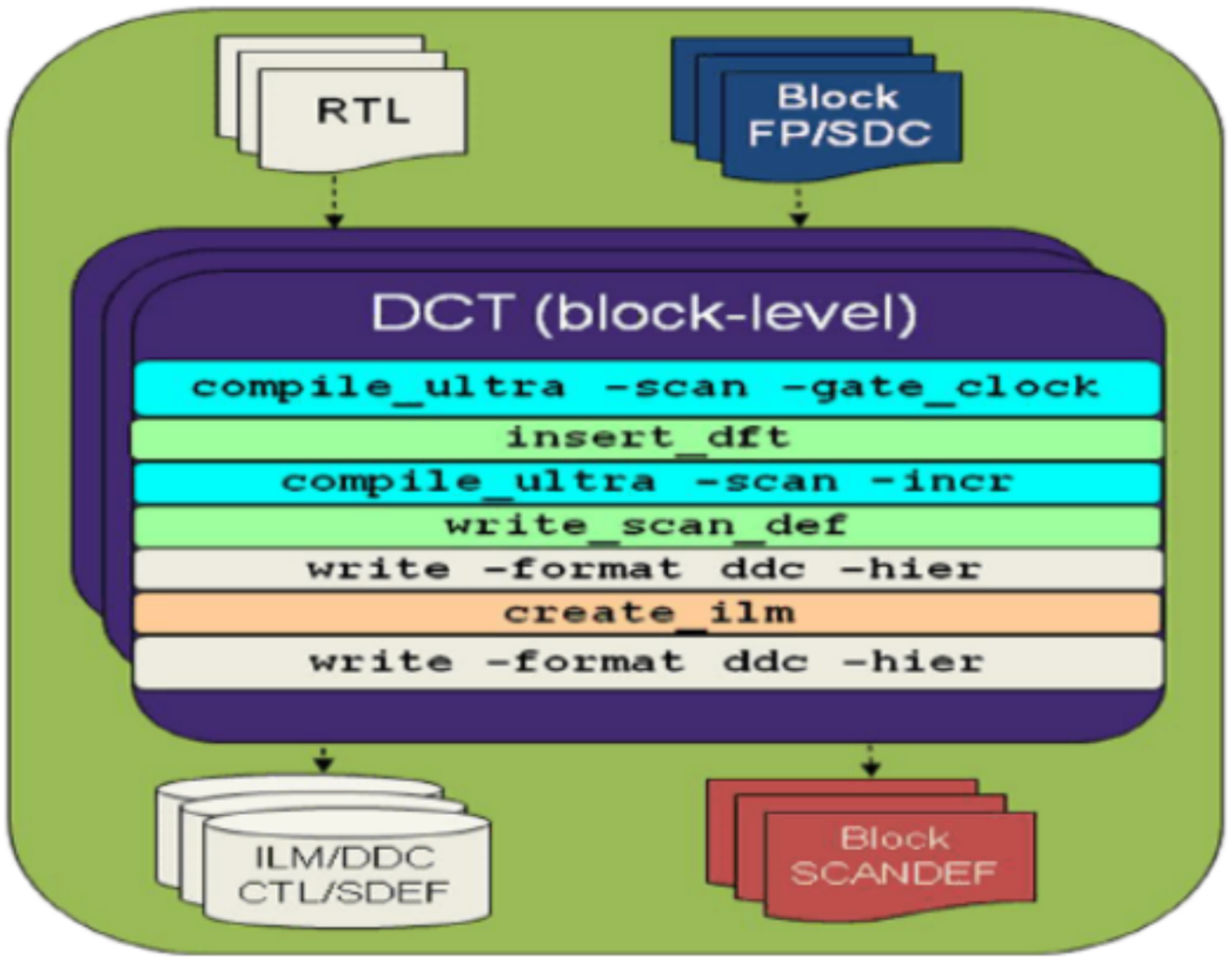
Design Compiler Topographical Mode Hierarchical Flow

This section describes the different bottom-up flow methodologies supported in Design Compiler topographical mode.

Block-Level Design Implementation

Figure 9 show the steps to perform the detailed block-level design implementation in order to generate either the Design Compiler topographical mode .ddc netlist or the Design Compiler topographical mode abstract ILM model.

Figure 9 Detailed Block-Level Design Implementation Flow Diagram



Compiling the subblock requires the following steps (see Design Compiler Reference Methodology (DC-RM) dc_scripts/dc.tcl):

1. Perform design-specific setup and specify the logic and physical libraries.
2. Read in the subblock (Verilog, VHDL, netlist, or .ddc) files
3. Set the current design to the subblock and link it.
4. Apply block-level timing constraints and power constraints.
5. (Optional but highly recommended) Apply block-level physical constraints or floorplan.
6. Perform test-ready and clock-gating compile of the subblock by using the **`compile_ultra -scan -gate_clock`** command.
7. Specify the design-for-test configuration and perform scan stitching with the **`insert_dft`** command so that the block-level scan chain can be incorporated in the top-level scan chain later.
8. Run the **`compile_ultra -scan -incremental`** command.

9. Use **uniquify_naming_style** to specify unique naming convention to be used by uniquify command and **uniquify -force** to uniquify the subblock in order to avoid naming conflicts during top-level design integration.
10. Use the **write_scan_def** command to save SCANDEF information and use the **write_test_model** command to save test model information about the block-level scan chain.
11. Save the mapped subblock in .ddc format.

Note:

The order of the **write_scan_def** command and **write -format ddc** command is important. You should use the **write_scan_def** command to first save the SCANDEF information of the subblock before using the **write -format ddc** command to generate the .ddc netlist in topographical mode because the binary SCANDEF information is also saved in the .ddc netlist.

If the subblock is presented as a .ddc netlist generated in topographical mode during top-level design integration, proceed on to "Top-Level Design Integration with Design Compiler Topographical Mode .ddc Netlist Subblocks" with the .ddc netlist created in step 10 above.

If the subblock is presented as an abstract ILM model generated in topographical mode during top-level design integration, follow these steps to create the interface logic model:

a. Run the **create_ilm** command to generate the abstract ILM model.
b. Use the **write -format ddc** command to save the abstract ILM model in .ddc format.

Note:

An abstract interface logic model (ILM) generated by Design Compiler topographical mode does not have sufficient physical information to be used in IC Compiler. Thus, a .ddc ILM netlist generated in topographical mode should not be used in IC Compiler.

Refer to Design Compiler Reference Methodology (DC-RM) dc_scripts/dc.tcl file for a comprehensive compile reference script example.

Sample Block-Level Design Compiler Topographical Mode Script:

```
%dc_shell-topo
. . . .
<Milkyway Design Library and TLU+ Setup>
. . . .
# Reading in Source Files
read_verilog [list M2B.v M2C.v M2D.v]
```

```
current_design M2B
link
read_sdc M2B.sdc
<Power Optimization Settings>
compile_ultra -scan -gate_clock
# DFT Insertion
<DFT Configurations>
hookup_testports -verbose
create_test_protocol
dft_drc
preview_dft -show all
insert_dft
compile_ultra -incremental -scan
set uniquify_naming_style "M2B_%s_%d"
uniquify -force
change_names -rules verilog -hierarchy
write_scan_def -output M2B.scandef
write_test_model -format ctl -output M2B.scan.ctl
write -format ddc -hierarchy -output M2B.ddc
. . . .
create_ilm
write -format ddc -hierarchy -out M2B.ILM.ddc
. . . .
```

With an abstract model generated in topographical mode, proceed on to "Top-Level Design Integration with Design Compiler Topographical Mode ILM Subblocks".

Alternatively, if the subblock is going to be presented as an abstract ILM model generated by IC Compiler during top-level design integration, refer IC Compiler Reference Methodology (ICC-RM) for the script and steps needed to create the interface logic model in IC Compiler.

With the abstract model generated by IC Compiler, proceed to "Top Level Design Integration with IC Compiler ILM Subblocks".

Top-Level Design Integration with Design Compiler Topographical Mode .ddc Netlist Subblocks

Figure 10 shows the design block diagram of a Design Compiler topographical mode .ddc netlist hierarchical flow where the subblock is presented as a topographical mode generated .ddc netlist. The numbered coding denotes the sequence of compile steps that you might follow to synthesize your design.

Figure 10 Design Block Diagram of a Design Compiler Topographical Mode .ddc Netlist Hierarchical Flow

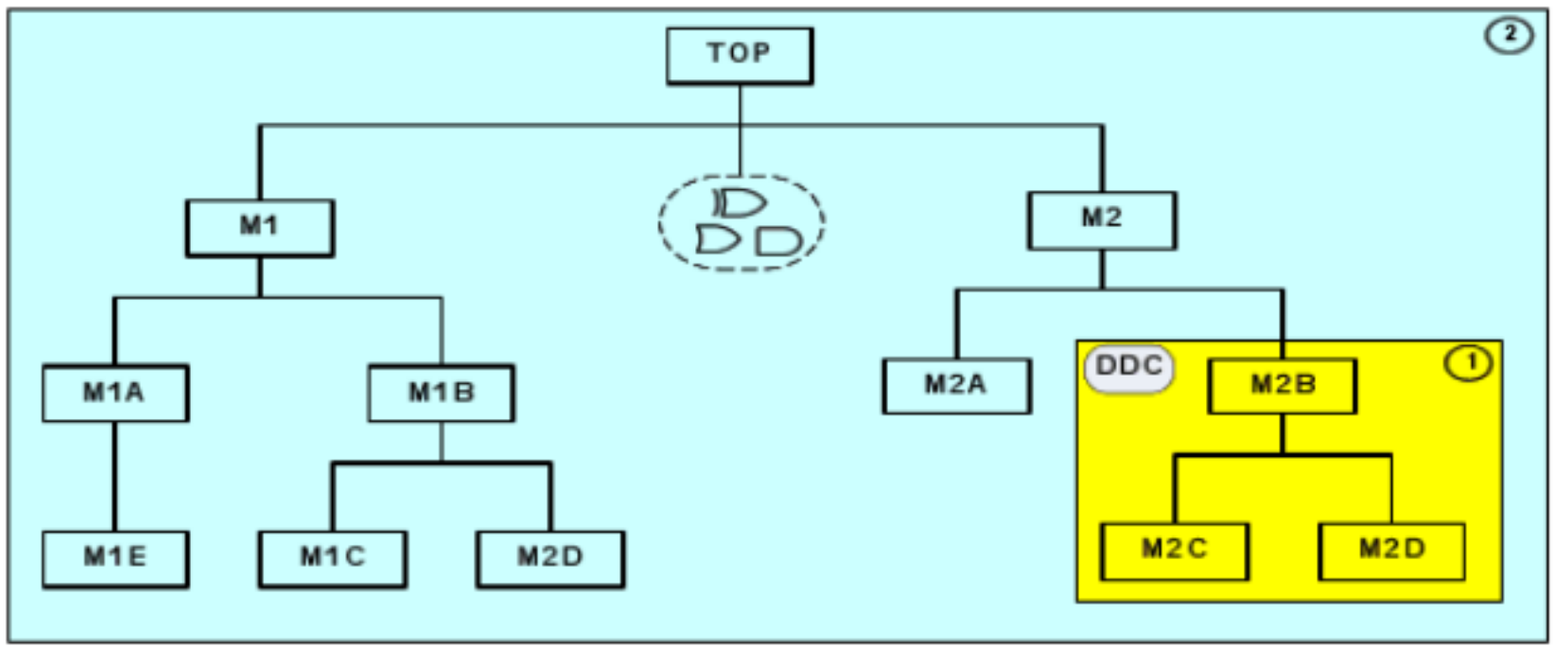
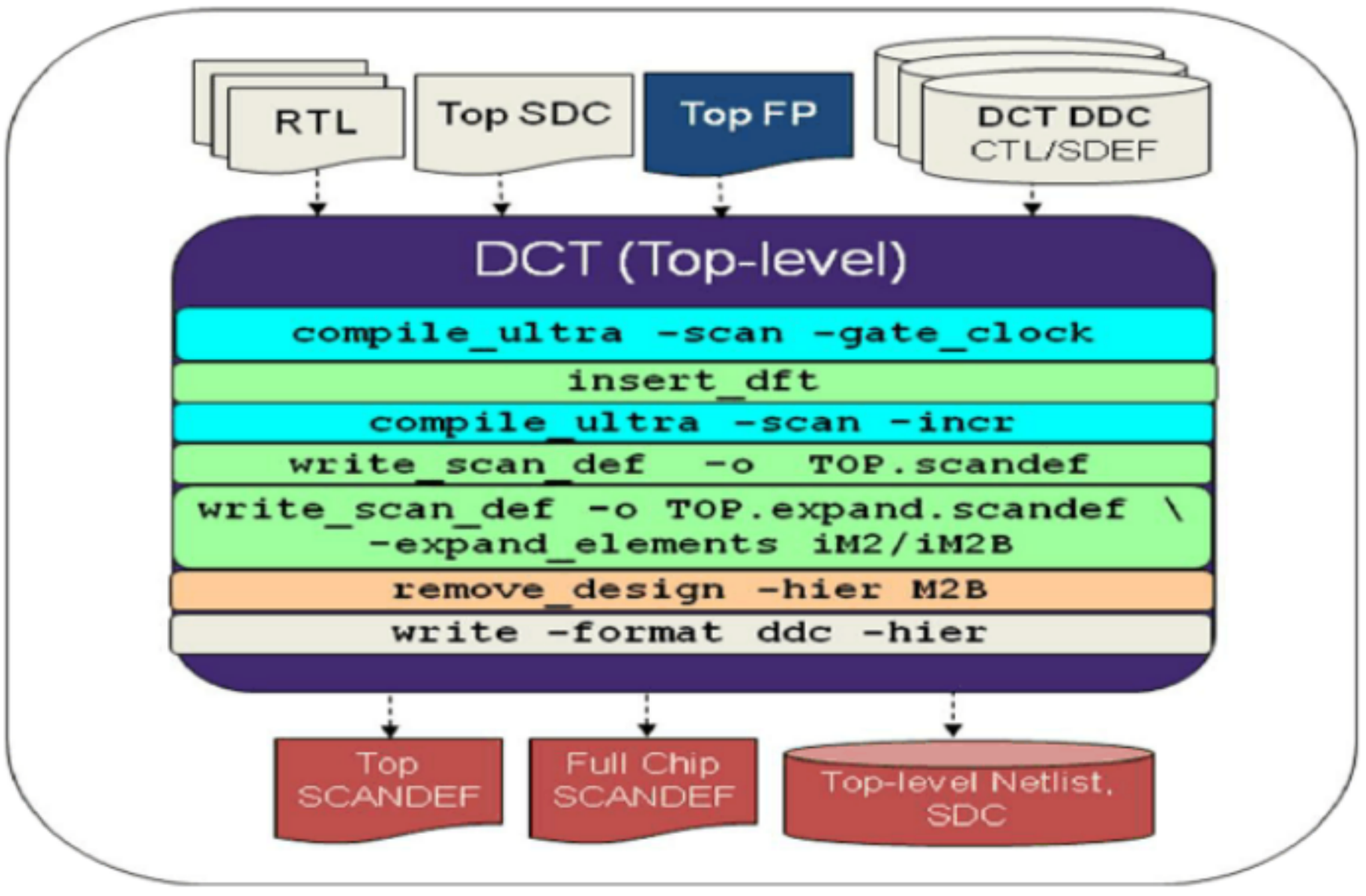


Figure 11 shows the tool usage for performing the top-level integration with Design Compiler topographical mode .ddc netlist subblocks.

Figure 11 Tool Flow Diagram of a Design Compiler Topographical Mode .ddc Netlist Hierarchical Flow



To compile the top-level design requires the following steps (see Design Compiler Reference Methodology (DC-RM) dc_scripts/dc_top.tcl):

1. Perform design-specific setup and specify the logic and physical libraries.
2. Read in the full-chip source files (Verilog, VHDL, netlist, or .ddc).
3. Remove the subblocks that will be treated as physical subblocks from memory.
4. Use the **`read_ddc`** command to read in the subblock .ddc netlist that you mapped in the Block Level Design Implementation step.
5. Set the current design to the top-level design and link it.
6. Use the **`set_physical_hierarchy`** command to specify that the subblock should be treated as a physical block. This will treat the subblock as a logical as well as a physical dont_touch block in order to preserve the block-level logical configuration and relative placement at the top level. The block-level virtual placements and back-annotated parasitics will be automatically propagated to the top level.

   Note:

   > If you do not specify that the subblock should be treated as a physical block, then the block-level relative placements will not be propagated to the top level and will not be used during top-level virtual-placement driven mapping and optimization. As a result, top-level virtual placement will be performed flat.

7. Apply top-level timing constraints and power constraints.
8. (Optional but highly recommended) Provide any top-level physical constraints.
9. Perform a test-ready and clock-gating compile of the top-level design by using the **`compile_ultra -scan -gate_clock`** command.
10. Specify the design-for-test configuration and perform scan stitching with the **`insert_dft`** command. The block-level scan chain will be incorporated in the top-level scan chain at the physical block interface test pins.
11. Run the **`compile_ultra -scan -incremental`** command.
12. (Optional) Uniquify the design.
13. Use the **`write_scan_def`** command to save the top-level SCANDEF information and the **`write_scan_def -expand_elements`** command to save the top-level SCANDEF information that includes expanded block-level scan leaf objects.
14. Remove all the physical block designs from memory.
15. Save the mapped top-level design netlist in .ddc format.

Refer to Design Compiler Reference Methodology (DC-RM) dc_scripts/dc_top.tcl file for a comprehensive compile reference script example.

Sample Top-Level Design Compiler Topographical Mode Script:

```
%dc_shell-topo
. . .
```

```
<Milkyway Design Library and TLU+ Setup>
. . . .
# Top-Level Mapping
read_verilog {TOP.v M1.v M1A.v M1B.v M1C.v \
                 M2.v M2A.v M2B.v M2C M2D}
remove_design -hierarchy [get_designs M2B]
read_ddc M2B.ddc
current_design TOP
link
set_physical_hierarchy [get_cells iM2/iM2B]
read_sdc TOP.sdc
extract_physical_constraints TOP.def (including location for sub-
block or with set_cell_location as shown below)
set_cell_location -coordinates {x1 y1} -fixed [get_cells iM2/iM2B]
<Power Optimization Settings>
compile_ultra -scan -gate_clock
# DFT Insertion
<DFT Configurations>
hookup_testports -verbose
create_test_protocol
dft_drc
preview_dft -show all
insert_dft
compile_ultra -scan -incremental
change_names -rules verilog -hierarchy
write_scan_def -output TOP.scandef
write_scan_def -output TOP.expand.scandef \
               -expand_elements [get_cells iM2/iM2B]
remove_design -hierarchy [get_designs M2B]
write -format ddc -hierarchy -output TOP.ddc
. . . .
```

Top-Level Design Integration with Design Compiler Topographical Mode ILM Subblocks

Figure 12 shows the design block diagram of a Design Compiler topographical mode ILM hierarchical flow where the subblock is presented as an abstract interface logic model generated by the topographical mode. The numbered coding denotes the sequence of compile steps that you might follow to synthesize your design.

Figure 12 Design Block Diagram of a Design Compiler topographical mode ILM Hierarchical Flow

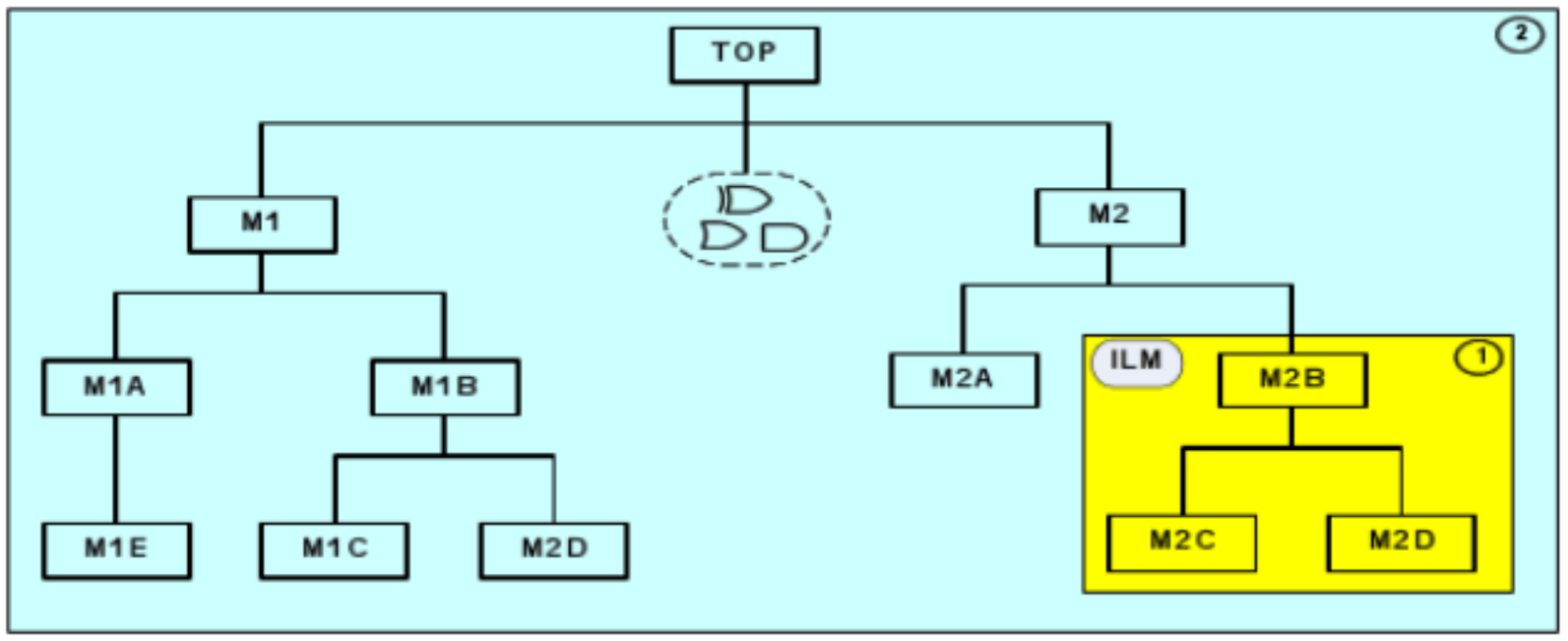
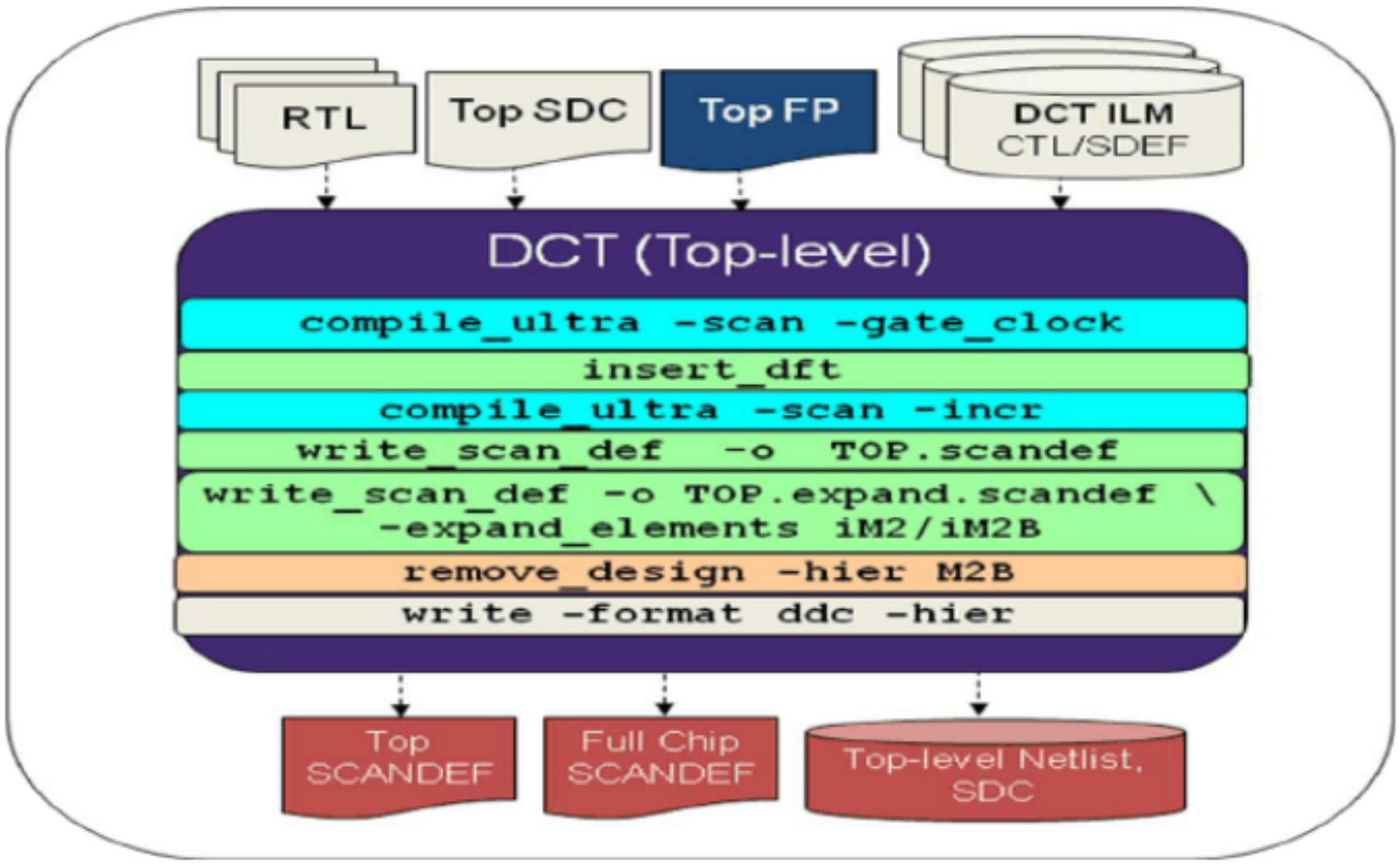


Figure 13 shows the tool usage for performing the top-level integration with Design Compiler topographical mode ILM subblocks.

Figure 11 Tool Flow Diagram of a Design Compiler Topographical Mode ILM Hierarchical Flow



To compile the top-level design requires the following steps (see Design Compiler Reference Methodology (DC-RM) dc_scripts/dc_top.tcl):

1. Perform design-specific setup and specify the logic and physical libraries.
2. Read in the full-chip source files (Verilog, VHDL, netlist, or .ddc).
3. Remove the subblocks that will be treated as physical subblocks from memory.
4. Use the **`read_ddc`** command to read in the subblock abstract interface logic model (.ddc netlist format) that you mapped in the "Block-Level Design Implementation" step.

   The subblock being presented as an abstract ILM model at the top level is automatically treated as a physical block. That is, the subblock will be treated as a logical as well as a physical dont_touch block in order to preserve the logical configuration and relative placement of the interface logic at the top level. Thus, block-level virtual placements and back-annotated parasitics will be automatically propagated to the top level by default.

5. Set the current design to the top-level design and link it.
6. Apply top-level timing constraints and power constraints.
7. (Optional but highly recommended) Provide any top-level physical constraints.
8. Perform a test-ready and clock-gating compile of the top-level design by using the **`compile_ultra -scan -gate_clock`** command.
9. Specify the design-for-test configuration and perform scan stitching with the **`insert_dft`** command. The block-level scan chain will be incorporated in the top-level scan chain at the physical block interface test pins.
10. Run the **`compile_ultra -scan -incremental`** command.
11. (Optional) Uniquify the design.
12. Use the **`write_scan_def`** command to save the top-level SCANDEF information and the **`write_scan_def -expand_elements`** command to save the top-level SCANDEF information that includes expanded block-level scan leaf objects.
13. Remove all the physical block designs from memory.
14. Save the mapped top-level design netlist in .ddc format.

Refer to Design Compiler Reference Methodology (DC-RM) "dc_scripts/dc_top.tcl" file for a comprehensive compile reference script example.

Sample Top-Level Design Compiler Topographical Mode Script:

```
%dc_shell-topo
. . . .
<Milkyway Design Library and TLU+ Setup>
. . . .
# Top-Level Mapping
read_verilog {TOP.v M1.v M1A.v M1B.v M1C.v \
                    M2.v M2A.v M2B.v M2C M2D}
```

```
remove_design -hierarchy [get_designs M2B]
read_ddc M2B.ILM.ddc
current_design TOP
link
read_sdc TOP.sdc
extract_physical_constraints TOP.def (including location for sub-
block or with set_cell_location as shown below)
set_cell_location -coordinates {x1 y1} -fixed [get_cells iM2/iM2B]
<Power Optimization Settings>
compile_ultra -scan -gate_clock
# DFT Insertion
<DFT Configurations>
hookup_testports -verbose
create_test_protocol
dft_drc
preview_dft -show all
insert_dft
compile_ultra -scan -incremental
change_names -rules verilog -hierarchy
write_scan_def -output TOP.scandef
write_scan_def -output TOP.expand.scandef \
               -expand_elements [get_cells iM2/iM2B]
remove_design -hierarchy [get_designs M2B]
write -format ddc -hierarchy -output TOP.ddc
. . . .
```

Top-Level Design Integration with IC Compiler ILM Subblocks

Figure 14 shows the design block diagram of an IC Compiler ILM hierarchical flow where the subblock is presented as an abstract interface logic model generated in IC Compiler. The numbered coding denotes the sequence of compile steps that you might follow to synthesize your design.

Figure 14 Design Block Diagram of an IC Compiler ILM Hierarchical Flow

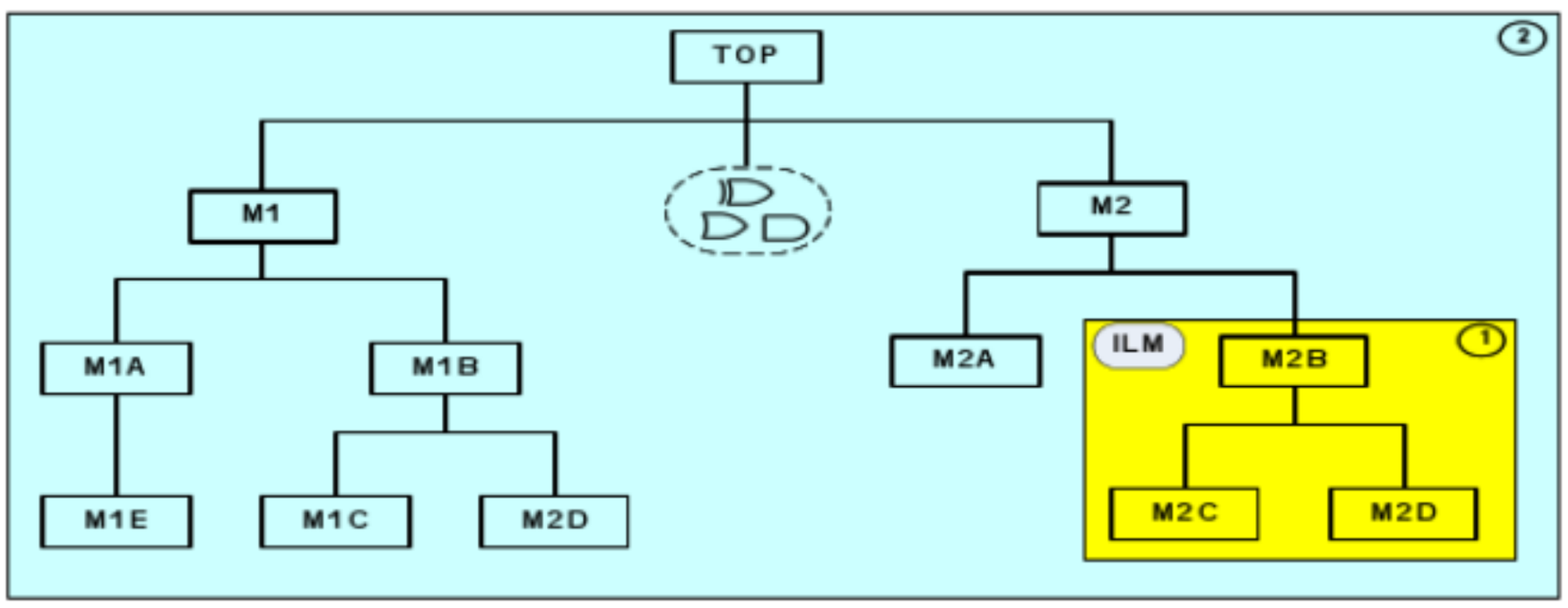
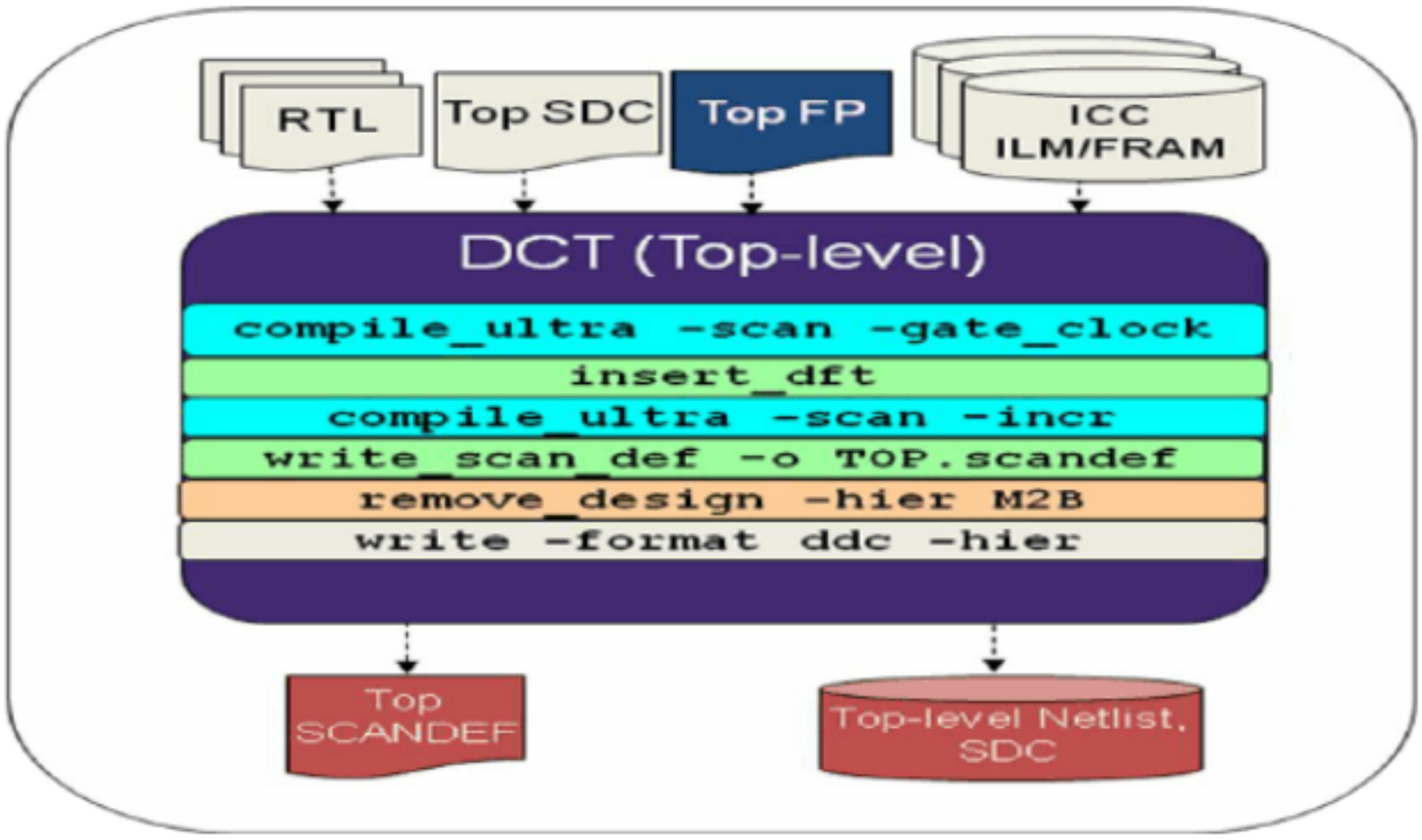


Figure 14 shows the tool usage for performing the top-level integration with IC Compiler ILM subblocks.

Figure 14 Tool Flow Diagram of a IC Compiler ILM Hierarchical Flow



To compile the top-level design requires the following steps (see Design Compiler Reference Methodology (DC-RM) dc_scripts/dc_top.tcl):

1. Perform the design-specific setup and specify the logic and physical libraries.
2. At the top-level design integration, integrating IC Compiler generated Milkyway ILM and FRAM views that you created in "Block Level Design Implementation" requires these two steps:

   a. Add the subblock Milkyway ILM view to the link library variable specification. For example, **`set link_library "$link_library M2B.ILM"`**

   b. Add the IC Compiler Milkyway subblock design library to the mw_reference_library variable specification. For example, **`set mw_reference_library "$mw_reference_library M2B.mw"`**.

The two steps mentioned above are automatically performed in dc_setup.tcl in Design Compiler Reference Methodology (DC-RM).
The subblock being presented as an abstract ILM model at the top level is automatically treated as a physical block. That is, the subblock will be treated as a logical as well as a physical dont_touch block in order to preserve the logical configuration and relative placement of the interface logic at the top level. Thus,

the block-level virtual placements and back-annotated parasitics will be automatically propagated to the top level by default.

3. Read in the full-chip source files (Verilog, VHDL, netlist, or .ddc).
4. Remove the subblocks that will be treated as physical subblocks from memory.
5. Set the current design to the top-level design and link top-level design.

   Note:

   When you link the top-level design, the link command will automatically link to the subblock abstract ILM model or ILM view under the specified subblock Milkyway design library.

6. Use **`read_test_model`** to load the test models for physical subblocks.
7. Apply the top-level timing constraints and power constraints.
8. (Optional but highly recommended) Provide any top-level physical constraints.
9. Perform a test-ready and clock-gating compile of the top-level design by using the **`compile_ultra -scan -gate_clock`** command.
10. Specify the design-for-test configuration and perform scan stitching with the **`insert_dft`** command. The block-level scan chain will be incorporated in the top-level scan chain at the physical block interface test pins.
11. Run the **`compile_ultra -scan -incremental`** command.
12. (Optional) Uniquify the design.
13. Use the **`write_scan_def`** command to save the top-level SCANDEF information. Since an abstract ILM model generated by IC Compiler is a hard macro, you do not have to use **`-expand_elements`** with the **`write_scan_def`** command in this case.
14. Remove all the physical block designs from memory.
15. Save the mapped top-level design netlist in .ddc format.

Refer to Design Compiler Reference Methodology (DC-RM) "dc_scripts/dc_top.tcl" file for a comprehensive compile reference script example.

Sample Top-Level Design Compiler Topographical Mode Script:

```
%dc_shell-topo
. . . .
set link_library "$tink_library M2B.ILM"
set mw_reference_library "$mw_reference_library M2B.mw"
. . . .
<Milkyway Design Library and TLU+ Setup>
. . . .
# Top-Level Mapping
read_verilog {TOP.v M1.v M1A.v M1B.v M1C.v \
                    M2.v M2A.v M2B.v M2C.v M2D.v}
remove_design -hierarchy [get_designs M2B]
current_design TOP
```

```
link
read_test_model -format ctl -design M2B M2B.scan.ctl
read_sdc TOP.sdc
extract_physical_constraints TOP.def (including location for sub-
block or with set_cell_location as shown below)
set_cell_location -coordinates {x1 y1} -fixed [get_cells iM2/iM2B]
<Power Optimization Settings>
compile_ultra -scan -gate_clock
# DFT Insertion
<DFT Configurations>
hookup_testports -verbose
create_test_protocol
dft_drc
preview_dft -show all
insert_dft
compile_ultra -scan -incremental
change_names -rules verilog -hierarchy
write_scan_def -output TOP.scandef
remove_design -hierarchy [get_designs M2B]
write -format ddc -hierarchy -output TOP.ddc
. . . .
```

Support for Design-For-Test

Design Compiler topographical mode supports design-for-test (DFT) features that Design Compiler does. However, the **`insert_dft`** command performs scan chain ordering and stitching based on virtual placement. Note that it uses stitch-only mode and does not perform any timing or DRC optimizations.

Some of the DFT features supported in Design Compiler topographical mode are listed below:

- Basic Scan
- Adaptive Scan
- AutoFix
- Internal pins
- Multi-mode scan
- Memories with test-models
- SCANDEF generation
- Core Wrapping
- Hierarchical Flows
- Multivoltage Flow

The following DFT features are not supported in Design Compiler topographical mode hierarchical flow:

- Bounday-scan design
- On-Chip Clocking support

For design-for-test (DFT) support in Design Compiler topographical mode hierarchical methodology, you should perform DFT insertion (scan-register replacement and scan-chain stitching) bottom-up. Block-level scan chains must be connected first during block-level implementation. Top-level scan chain can then connect to block-level chains at the subblock boundary test pins during top-level integration.

For block-level DFT insertion in a bottom-up flow, a test model describing the block-level scan chain structure is automatically attached to the block after the **`insert_dft`** command is executed. You must write out a .ddc netlist and a SCANDEF file in order to perform design placement, scan chain reordering, and repartitioning in IC Compiler.

For top-level integration, you can integrate a mixture of adaptive scan and pure scan blocks.

In hierarchical adaptive scan synthesis (HASS), adaptive scan logic is placed at the block level, and all cores with adaptive scan logic are integrated at the chip level. This approach helps reduce the routing congestion prevalent in multimillion-gate designs.

The hierarchical adaptive scan synthesis-hybrid flow provides additional support for adaptive scan insertion for user-defined logic (UDL). A normal hierarchical adaptive scan synthesis flow supports insertion of basic scan chains for UDL, and the hierarchical adaptive scan synthesis-hybrid flow supports insertion of adaptive scan for UDL.

You cannot integrate blocks that have boundary scan or On Chip Clocking controllers inserted.

During top-level DFT insertion, the **`insert_dft`** command connects the block-level scan chain of all blocks with attached test models to the top-level scan chain at the interface test pins. As a result, all blocks represented by test models will be described using the BITS construct in the top-level SCANDEF file unless you use **`the -expand_elements`** option of the **`write_scan_def`** command to have top-level SCANDEF directly reference block-level scan cells.

The subblocks described with the BITS construct in the top-level SCANDEF must be treated as abstract Interface Logic Model (ILM) blocks during top-level placement in IC Compiler; otherwise, IC Compiler errors out with this SCANDEF and design netlist that contains the subblock functionality. For the subblocks in question, IC Compiler treats the subblock scan segments as black-box segments and does not reorder block-level scan cells or segments during top-level design closure phase.

You must run the **compile_ultra -scan -incremental** command after the **insert_dft** command in topographical mode so that virtual placement information can be derived for any cells introduced by the **insert_dft** command.

Steps in the DFT Flow

1. Follow these steps for lower-level blocks:
    a. Map the block by using the **compile_ultra -scan -gate_clock** command.
    b. Specify the DFT configurations.
    c. Perform block-level DFT insertion by using the **insert_dft** command.
    d. Perform **compile_ultra -scan -incremental** mapping and optimization.
    e. Write out the .ddc netlist and SCANDEF file for IC Compiler.

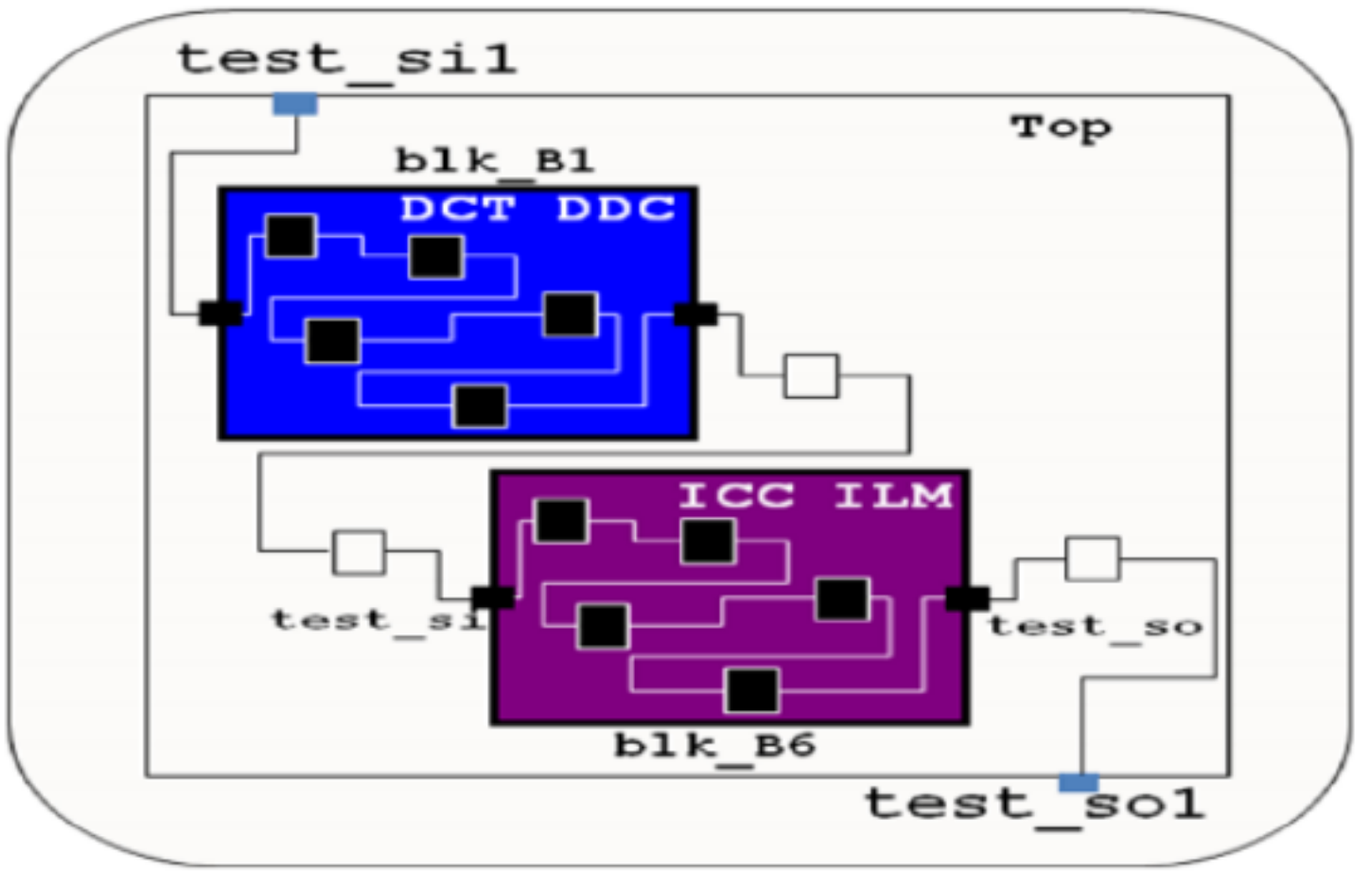
2. Follow these steps for the top-level design:
    a. Read in the block-level mapped .ddc netlist and other top-level source files.
    b. Map the top level by using the **compile_ultra -scan -gate_clock** command.
    c. Specify the DFT configurations.
    d. Perform top-level DFT insertion by using the **insert_dft** command.
        Top-level DFT insertion connects the block-level scan chain of a scan-stitched physical subblock with attached test models to the top-level scan chain at the interface test pins, along with any top-level scan-ready registers.
    e. Use the **compile_ultra -scan -incremental** command to perform mapping and optimization.
    f. Write out the .ddc netlist and SCANDEF file for IC Compiler.

The top-level SCANDEF file contains the BITS construct for all subblocks represented by test models. Thus, this physical partition must be maintained in IC Compiler as well.

You must use the **-expand_elements** option with **write_scan_def** command to write a flat top-level design SCANDEF that directly references block-level scan leaf objects.

Figure 15 shows a design block diagram with two physical subblocks. The **-expand_elements** option of **write_scan_def** command can be used to control whether top-level SCANDEF contains reference to block-level scan chain objects or "BITS" construct for this design example.

Figure 15 SCANDEF Top Design Block Diagram



With "**write_scan_def -expand_elements** blk_B1" command, the following top-level SCANDEF information is generated:

```
DESIGN Top ;
SCANCHAINS 1 ;
- 1
+ START test_si1
+ FLOATING
blk_B1/cell1 ( IN TI ) ( OUT SO )
blk_B1/cell2 ( IN TI ) ( OUT SO )
blk_B1/cell3 ( IN TI ) ( OUT SO )
blk_B1/cell4 ( IN TI ) ( OUT SO )
blk_B1/cell5 ( IN TI ) ( OUT SO )
glue_logic_cell1 ( IN TI ) ( OUT SO )
glue_logic_cell2 ( IN TI ) ( OUT SO )
blk_B6
  ( IN test_si ) ( OUT test_so ) (BITS 5)
 glue_logic_cell3 ( IN TI ) ( OUT SO )
+ PARTITION Clk_45_45
+ STOP test_so1 ;
```

Support for Power

In Design Compiler topographical mode, power optimization and prediction support includes Clock Tree Estimation (CTE). CTE is used in the **compile_ultra** or **compile_ultra -incremental** command to provide accurate power estimation as well as correlated power post-synthesis when compared to place-and-route numbers because the tool considers the clock tree power utilization of the design during synthesis.

In topographical mode, clock tree estimation is run within the **compile_ultra** or **compile_ultra -incremental** commands as the last phase if power correlation is enabled by using the **set_power_prediction** command or any one of the following gate-level power optimization commands:

- **set_max_leakage_power**
- **set_max_dynamic_power**
- **set_max_total_power**
- **set_power_gating_style**

Any subsequent incremental compiles would cause CTE to be run again. A .ddc file generated in topographical mode would always have the CTE power number from the last CTE run, unless CTE was never run in any of the previous compiles.

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 with the **compile_ultra (-incremental)** command. Otherwise, the **report_power** command will only report the total (static and dynamic) power used by the design without accounting for power usage attributed to clock-tree estimates.

The recommended power flow in a bottom-up strategy in topographical mode is to enable power correlation and run CTE during block-level block compiles. Doing so provides accurate power for the lower-level modules.

If you treat the subblock as an abstract interface logic model generated in either topographical model or IC Compiler during top-level design integration, the accuracy of power correlation is reduced because the estimated power number for this lower-level abstract block is not accounted for at the top level.

The clock tree estimation feature is automatically disabled during top-level design integration if any of the physical blocks are being modeled as abstract ILM models.

Conclusion

This application note described three hierarchical flows supported in Design Compiler topographical mode: the topographical mode based .ddc netlist hierarchical flow, the topographical mode generated ILM based hierarchical flow, and the IC Compiler generated physical ILM based hierarchical flow. The preferred synthesis flow in Design Compiler topographical mode is a top-down strategy. The bottom-up methodologies described in this application note can be used to address runtime and capacity design challenges or to employ a divide-and-conquer synthesis approaches. It is recommended that you use consistent timing and physical constraints throughout the design flow. This will ensure better quality of results (QoR) and will result in tighter correlation between Design Compiler topographical mode and IC compiler.