SpyGlass® Constraints Submethodology (for GuideWare 2017.12)

Version N-2017.12-SP2, June 2018

SYNOPSYS®

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

Report an Error

The SpyGlass Technical Publications team welcomes your feedback and suggestions on this publication. Please provide specific feedback and, if possible, attach a snapshot. Send your feedback to *spyglass_support@synopsys.com*.

Contents

Preface **7**

About This Book **7**
Contents of This Book **8**
Typographical Conventions **9**

Constraints-Optimized Design **11**

Introduction **12**
Tool and Methodology Version 12
References 13
Terminology 13
The Constraints Problem **14**
Typical Problems within a Constraints File 14
Typical Problems with Constraints in a Design Flow 15
Optimizing and Cleaning the Design Constraints **18**
SpyGlass Constraints Overview 18
Goals for Block/IP 20
Goals for SoC RTL and Netlist 21
Constraints Validation using SpyGlass 22
Step-by-Step Solution **25**
Setup 25
Record Design Intent 25
Analyze the Flavor of SDC 26
Gather Design Data 27
Configure SpyGlass Design Constraint (SGDC) File 27
Run Sanity Checks on Inputs 29
Check the Coverage of the Constraints 30
Generate Constraints 31
SDC Generation in Batch Mode 45
Block/IP Methodology Flow 47
SpyGlass Constraints Block/IP Quick Start 48
Block/IP Detailed Procedure 48
SoC Methodology Flow 53
SpyGlass Constraints SoC Quick Start 54
SoC Detailed Procedure 54

SoC Methodology using Abstraction 58
Using the Methodology for SpyGlass Constraints Solution 58
Generating an Abstract View in SpyGlass Constraints 58
Validating Block Assumptions in SpyGlass Constraints 62
Using the Abstract View in SpyGlass Constraints 65
Analyzing Results 66
Debugging Reports 66
Waiving Messages 68
Conclusion 69

Appendix A: SpyGlass Constraints Design Data Checklist 71

Appendix B: Example Project File 73

Preface

About This Book

The SpyGlass® Constraints methodology guide describes the flow for using the Constraints methodology.

Contents of This Book

The SpyGlass Constraints methodology guide has the following sections.

Section	Description
Constraints-Optimized Design	The need for constraints-optimized design

Typographical Conventions

This document uses the following typographical conventions:

To indicate	Convention Used
Program code	`OUT <= IN;`
Object names	`OUT`
Variables representing objects names	*<sig-name>*
Message	Active low signal name '<sig-name>' must end with _X.
Message location	OUT <= IN;
Reworked example with message removed	OUT_X <= IN;
Important Information	**NOTE:** This rule...

The following table describes the syntax used in this document:

Syntax	Description
[ ] (Square brackets)	An optional entry
{ } (Curly braces)	An entry that can be specified once or multiple times
| (Vertical bar)	A list of choices out of which you can choose one
. . . (Horizontal ellipsis)	Other options that you can specify

Constraints-Optimized Design

Read the following sections to understand how to make your design constraints-optimized using the SpyGlass® Constraints solution:

- *Introduction*
- *The Constraints Problem*
- *Optimizing and Cleaning the Design Constraints*
- *Step-by-Step Solution*
- *Conclusion*

Introduction

Validating constraints throughout the design flow requires a methodology that guides designers through each step in the flow, specifying how to clean up and optimize the design constraints. This not only improves the QoR, but reduces expensive respins and iterations. This document introduces a methodology to make your design constraints-optimized using SpyGlass Constraints.

This section contains the following subsections:

- *Tool and Methodology Version*
- *References*
- *Terminology*

In the next section, *The Constraints Problem*, the designer is introduced to the concept of constraints, constraints-related problems typically faced in a design, and the overall impact.

The *Optimizing and Cleaning the Design Constraints* section describes in general how these problems can be fixed to avoid iterations and respins and achieve faster timing closure. This is followed by detailed instructions in the *Step-by-Step Solution* section prescribing a methodology.

This document is intended for use by both novices and advanced users of the SpyGlass Constraints solution. It is not a replacement for the SpyGlass Constraints rules reference guide or training materials. The reader is expected to be familiar with SpyGlass, the features, and data flow before using this document. Advanced users can go directly to the relevant sections, such as the *SpyGlass Constraints Block/IP Quick Start*, *SpyGlass Constraints SoC Quick Start*, and *Analyzing Results*.

While this methodology relates to specifically to timing constraints, GuideWare provides a start for design groups with SpyGlass goals readily usable at various phases of the IC design flow, such as Block/IP and SoC Integration. You can configure GuideWare to map to a specific design style and hand-off requirements.

Tool and Methodology Version

- SpyGlass Version: N-2017.12-SP2

- SDC Version: 2.0 or prior
- GuideWare Version: 2017.12

References

- SpyGlass Constraints Rules Reference Guide
- SpyGlass TXV Rules Reference Guide
- SpyGlass CDC Rules Reference Guide
- SpyGlass DFT Rules Reference Guide

Terminology

- **Design:** A design is a composed group of logic at any level. Therefore, the only level considered not to be a design, as it is used in the context of this document, is a primitive from a library. A design could be at RTL-level, Gate-level (netlist), or mixed.
- **SpyGlass Constraints:** Additional information about the design, which is not captured in the RTL description. Constraints in SpyGlass are typically captured in a <design>.sgdc file and include clock definitions, case or mode specifications, and signal dependencies.
- **Timing Constraints:** Additional information about the timing requirements for the design, such as clock definition, I/O delays, and timing exceptions, that are passed to synthesis, STA, or implementation tools. These are typically captured in an SDC and/or a Tcl file.
- **Parameters:** These SpyGlass options enable you to control behavior of rules during the analysis of constraints.

The Constraints Problem

This section describes typical problems within a constraints file and a design flow.

- *Typical Problems within a Constraints File*
- *Typical Problems with Constraints in a Design Flow*

Typical Problems within a Constraints File

A typical constraints file pertaining to a block may have many issues related to:

- **Clock Definitions**: Clock issues lead to excessive iterations among block synthesis, STA, and P&R. This includes inconsistencies in the specification of clocks, generated clocks, and all related clock data, such as latency, uncertainty, and buffering.
- **Input and Output Delays**: Inconsistencies in input and output delay specification can lead to incorrect or suboptimal synthesis results. Over-constraining may result in longer synthesis run times and extra buffering on tight paths. Under-constraining will result in not meeting chip-level timing goals.
- **Exceptions**: Exception validation is needed because:
  - Incorrect timing exceptions, especially the ones on timing critical paths, may lead to silicon failing to meet timing because the timing path violations are masked until silicon.
  - Too many exceptions overwhelm the implementation tools. Therefore, if there are exceptions on invalid paths, paths blocked by constant propagation, or functionally incorrect exceptions, it is better to identify them in advance and remove them from the constraints before implementation.
  - Verifying exceptions manually is a time-consuming and error-prone process.
  - Typical exception issues include false paths set on paths that are not structurally connected, false paths specified on true paths, or incorrect cycle counts specified for multicycle paths.

- The level of support on various commands/options for constraints varies from one tool to another. Therefore, maintaining a flow involving multiple vendors requires a tool that can intelligently indicate if the constraint is supported by a specific tool.
- Constraints issues are not limited to a single constraint file for a block, but can also occur in the hierarchical context involving multiple constraint files for several blocks and the chip-level design.

Here are some typical problems in a constraint file:

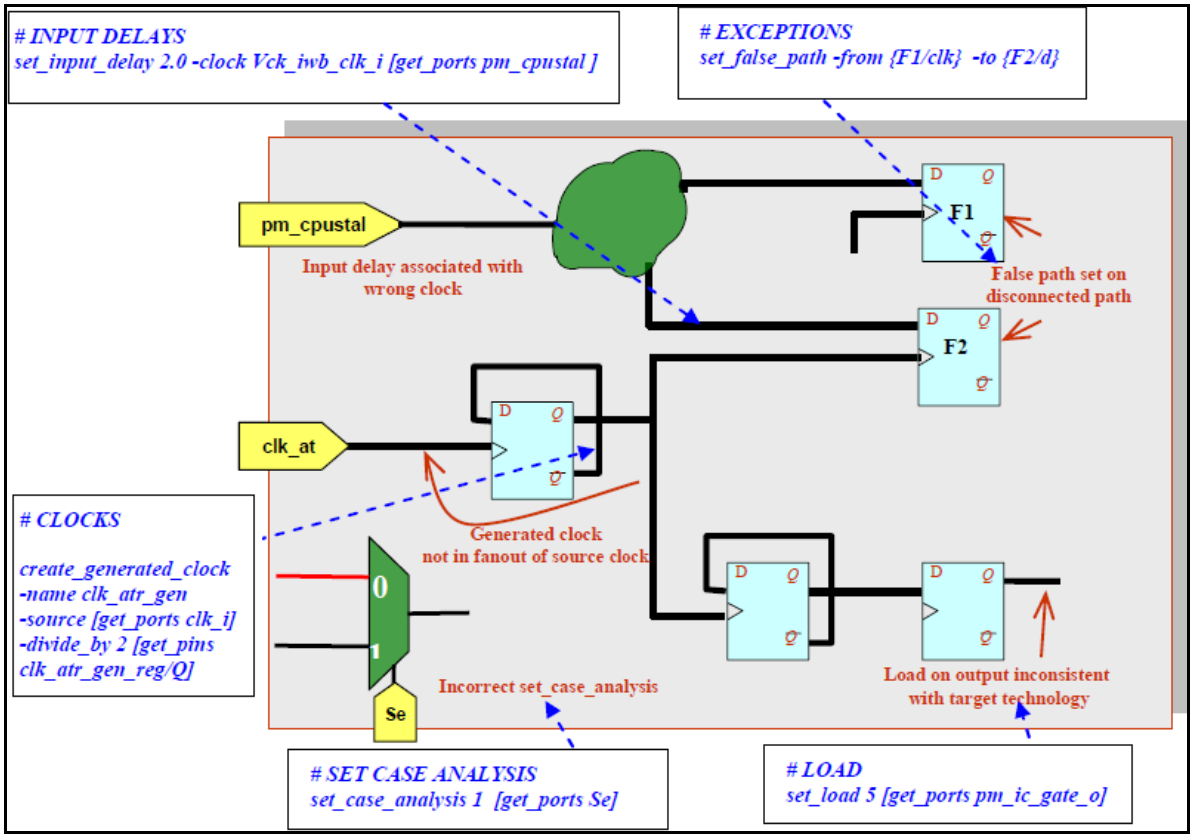


FIGURE 1. Typical constraints issues in a design

Typical Problems with Constraints in a Design Flow

Constraints definitions evolve during each stage of the implementation of a chip. Figure 2 illustrates the need for a constraints-checking solution in a typical design flow from RTL to layout.

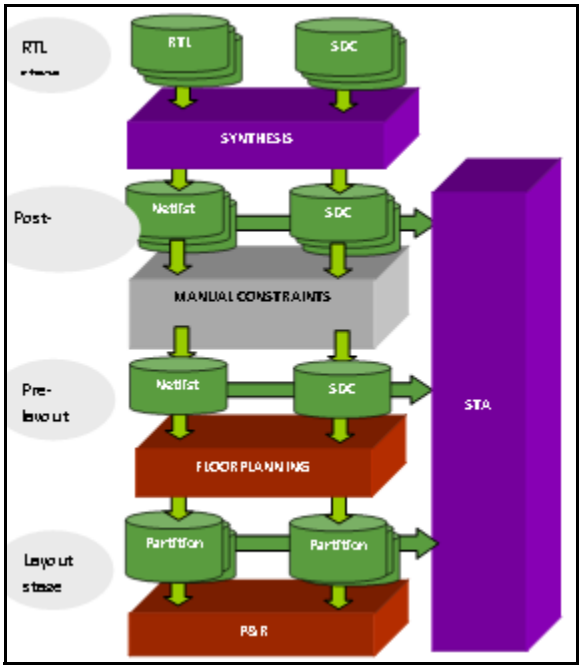


FIGURE 2. Constraints in a design flow

In each stage, the constraints-checking solution should address the following:

RTL Stage

- Incorrect, inconsistent, and incomplete constraints for an RTL design can cause longer implementation (synthesis, timing) cycles.
- First-pass constraints creation is typically a manual, error-prone, and drawn-out process. For large designs, creating clock definitions and input/output delays is often a tedious task.
- Incorrect exceptions lead to silicon failure or at a minimum cause suboptimal design performance. This is typically a time-consuming process.

Post-synthesis Stage

- Incorrect, inconsistent, and incomplete constraints for the netlist can cause longer implementation (synthesis, timing) cycles.
- Incorrect hook-up of test-logic and clock gating can introduce additional paths that tools unnecessarily attempt to optimize.
- Missing out correct exceptions, especially on paths that have failed timing prolongs timing closure. This is typically a manual, time-consuming, and iterative process.

Pre-layout Stage

- Chip-level timing closure takes longer if constraints are inconsistent across the hierarchy. Typically, block-level constraints are designed independent of chip-level constraints, and conflicts can occur between constraints at the two levels.
- Propagating block-level constraints to the chip level is a manual and error-prone task.
- Verify timing exceptions is an error-prone and time-consuming process.
- Missing out correct exceptions, especially on paths that have failed timing prolongs timing closure. This is typically a manual, time-consuming, and iterative process.

Layout Stage

- Incorrect, inconsistent and incomplete constraints for the netlist can cause longer implementation (synthesis, timing) cycles.
- Need to ensure that the physical partition budgets are correct.
- Missing out correct exceptions, especially on paths that have failed timing prolongs timing closure. This is typically a manual, time-consuming, and iterative process.
- Since different tools in the design flow require variations of SDC, managing multiple versions of constraints in a consistent and co-related way is required.

Optimizing and Cleaning the Design Constraints

This section contains the following sub-sections:

- *SpyGlass Constraints Overview*
- *Constraints Validation using SpyGlass*

SpyGlass Constraints Overview

The SpyGlass Constraints solution provides the following capabilities to the SpyGlass environment:

- **SpyGlass Constraints Creation**
  - ❒ Generates SDC template from RTL
- **SpyGlass Constraint Validation**
  - ❒ Pinpoint syntax, consistency, and methodology issues
  - ❒ Validates intrablock, interblock, blocks-versus-chip constraints

In relation to a typical design flow from RTL to layout, as shown in *Figure 2*, the SpyGlass Constraints solution performs the following:

RTL Stage

- Validates the RTL design constraints for correctness, consistency, and completeness to facilitate synthesis.
- Generates an SDC template.

Netlist Handoff

- Validates constraints for consistency and completeness to facilitate STA.
- Validates correct hook up of test-logic.
- Validates that clock gating constraints are set correctly.
- Validates hierarchical constraints consistency and reports the inconsistencies.
- Ensures consistent and complete constraints for layout and reports issues.

Layout Stage

- Validates that constraints are consistent and complete for P&R and reports issues.
- Reports hierarchical inconsistencies after physical partition budgets are created.

Timing Exception Verification and Exception generation for timing critical paths from STA reports is a capability of the SpyGlass TXV solution.

The GuideWare Reference Methodology describes two fields of use:

- **Block Development**: In this field of use, it is assumed that the RTL being developed is mostly new. No assumptions are made about existing behavior or stability. The key concerns are the feasibility and performance of the design. It is assumed that the design intent is mostly known to the engineers and they can specify it to SpyGlass. Checks and goals are organized to align with the evolution and maturity of the new RTL block.

  This field of use contains the following stages: Initial RTL Development, RTL Handoff, and Netlist Handoff.
- **SOC for RTL and Netlist**: The SoC integration phase includes stitching of the new RTL blocks or IPs. This field of use contains the following stages: SoC Integration (of RTL Blocks), SoC Netlist handoff, and SoC Layout handoff

The following sections show how the GuideWare fields of use correspond to the SpyGlass Constraints goals.

- *Goals for Block/IP*
- *Goals for SoC RTL and Netlist*

Goals for Block/IP

The following table lists goals you should run in each design stage for Block/IP. The table uses the following legend:

- M denotes mandatory goals for the design stage
- O denotes optional goals for the design stage
- NA denotes goals that are not applicable for the design stage

Goals	Design Stages		
	initial_ rtl	rtl_handoff	netlist_handoff
`sdc_gen` Creates SDC templates from RTL or netlist.	O	O	NA
`sdc_audit` Computes design coverage and reports uncovered design objects.	M	M	M
`sdc_check` Detects inconsistencies in specification of clocks, generated clocks, and perform basic checks.	M	M	M
`sdc_exception_struct` Checks that timing exceptions specified in a constraints file are on paths that are structurally connected.	NA	M	M
`sdc_redundancy` Remove any redundancy in the constraints and performs checks that might facilitate better retargeting.	NA	M	M
`sdc_abstract` Generates the abstract port for a design.	NA	O	O

Goals for SoC RTL and Netlist

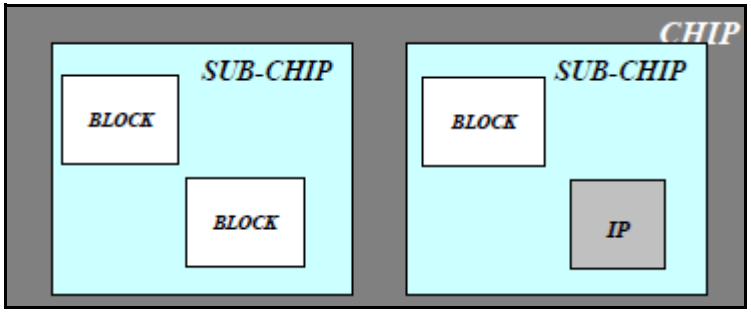
The following table the you should run in each design stage for Block/IP. The table uses the following legend: M denotes mandatory goals for the design stage, O denotes optional goals for the design stage, and NA denotes goals that are not applicable for the design stage.

Goal	Design Stages			
	SoC RTL		SoC Netlist	
	initial_ rtl	rtl_handoff	netlist_handoff	layout_handoff
`sdc_gen` Creates SDC templates from RTL or netlist.	O	O	NA	NA
`sdc_audit` Computes design coverage and reports uncovered design objects.	M	M	O	O
`sdc_abstract_validate` Validates the abstract port for a design.	M	M	M	M
`sdc_check` Detects inconsistencies in specification of clocks, generated clocks, and perform basic checks.	M	M	M	M
`sdc_exception_struct` Checks that timing exceptions specified in a constraints file are on paths that are structurally connected.	NA	M	M	M
`sdc_redundancy` Removes any redundancy in the constraints and performs checks that might facilitate better retargeting.	NA	M	M	M
`sdc_abstract` Generates the abstract port for a design.	NA	O	O	NA

Constraints Validation using SpyGlass

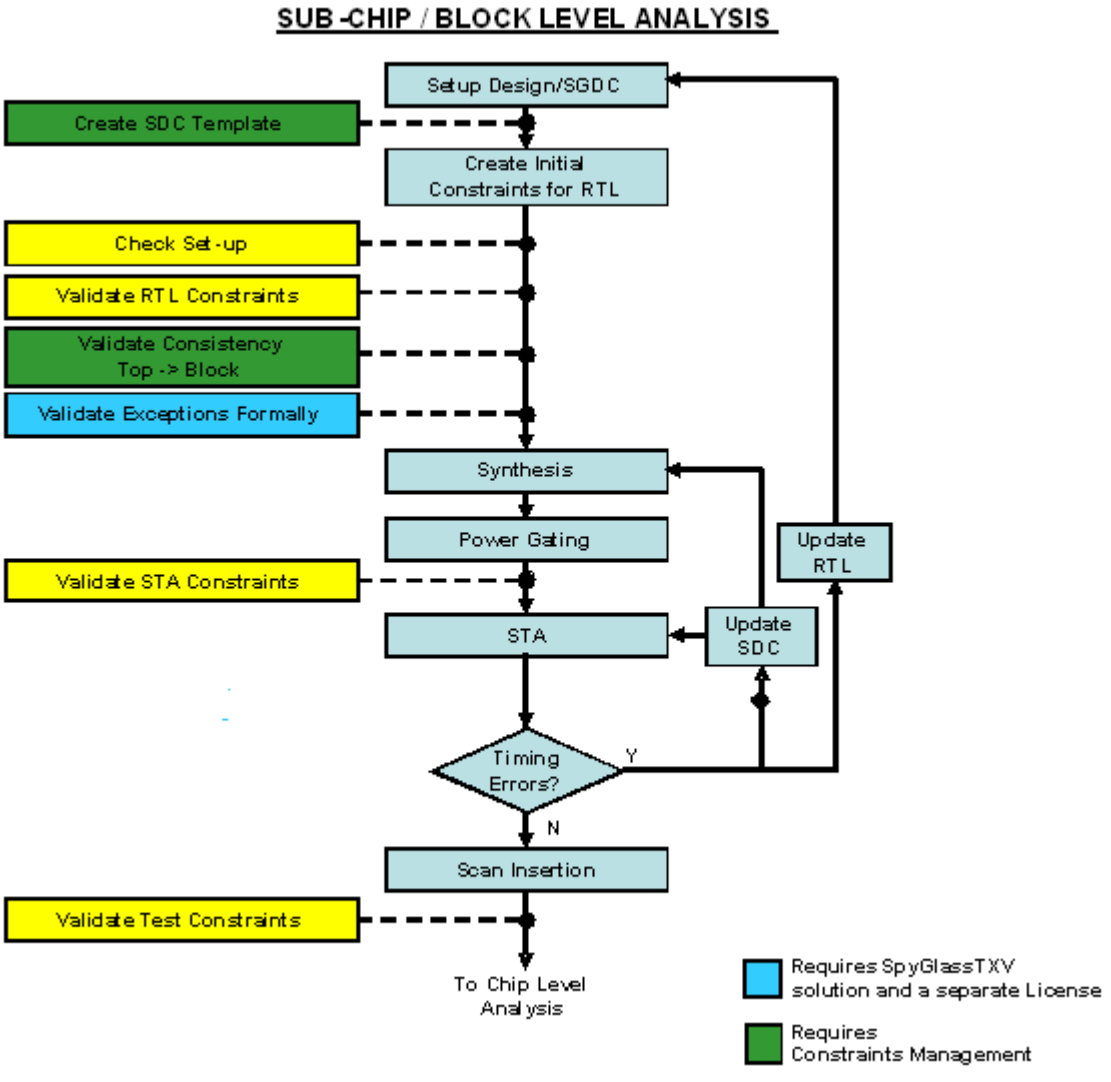
When analyzing the SDC of a design, the design can be categorized as:

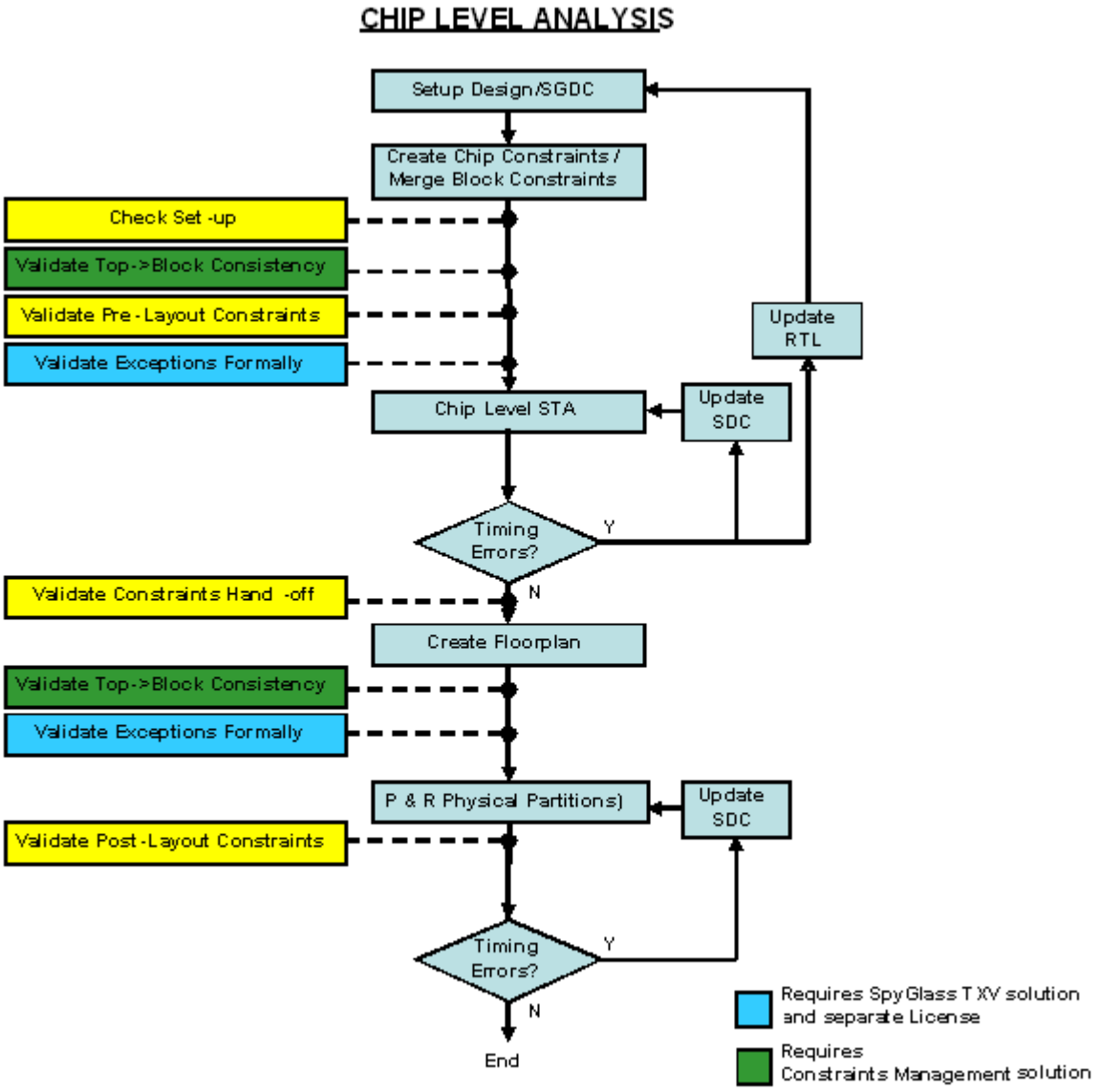
- **Block**: Lowest module-level in a design.
- **IP**: External IP or completed or legacy block for which a library model is available. Legacy blocks that have design information available can be treated as a **Block**.
- **Chip/Subchip**: The **Chip** corresponds to the top-level of a design. **Subchip** corresponds to a higher-level block, which has **Blocks** and/or **IPs** instantiated. SDCs for the lower-level (instantiated) blocks may be available, but this is not mandatory.



Please refer to *Goals for Block/IP* and *Goals for SoC RTL and Netlist* to see which goal, therefore the corresponding methodology step described in this document, applies to block, subchip or chip levels.

The following diagram illustrates the recommended steps for constraints validation using SpyGlass. The following sections of this document details each step of this flow.


SUB-CHIP / BLOCK LEVEL ANALYSIS
Setup Design/SGDC
Create SDC Template
Create Initial Constraints for RTL
Check Set-up
Validate RTL Constraints
Validate Consistency Top -> Block
Validate Exceptions Formally
Synthesis
Power Gating
Validate STA Constraints
STA
Update SDC
Update RTL
Timing Errors?
Y
N
Scan Insertion
Validate Test Constraints
To Chip Level Analysis
Requires SpyGlassTXV solution and a separate License
Requires Constraints Management


CHIP LEVEL ANALYSIS
Setup Design/SGDC
Create Chip Constraints / Merge Block Constraints
Check Set -up
Validate Top->Block Consistency
Validate Pre-Layout Constraints
Validate Exceptions Formally
Update RTL
Chip Level STA
Update SDC
Timing Errors?
Y
N
Validate Constraints Hand -off
Create Floorplan
Validate Top->Block Consistency
Validate Exceptions Formally
P & R Physical Partitions)
Update SDC
Validate Post-Layout Constraints
Timing Errors?
Y
N
End
Requires SpyGlass T XV solution and separate License
Requires Constraints Management solution

Step-by-Step Solution

This section contains the following subsections:

- *Setup*
- *SDC Generation in Batch Mode*
- *Block/IP Methodology Flow*
- *Block/IP Detailed Procedure*
- *SoC Methodology Flow*
- *SoC Methodology using Abstraction*
- *Analyzing Results*

Setup

The following list displays the steps for setup:

- *Record Design Intent*
- *Analyze the Flavor of SDC*
- *Gather Design Data*
- *Configure SpyGlass Design Constraint (SGDC) File*
- *Run Sanity Checks on Inputs*
- *Check the Coverage of the Constraints*
- *Generate Constraints*

Record Design Intent

This is a manual step to gather and record as much design intent information as possible.

We recommend you first run the solution at the **Block** level, and then move to **Chip/Subchip** level. Use a bottom-up approach and then as you make progress, start integrating the blocks sequentially and progressively move up the hierarchy until you reach the top level of the device.

Analyze the Flavor of SDC

There is a difference between an SDC that is read by DC or PT. Before analyzing the constraints, ascertain the flavor of SDC that is being supplied. There may not be information inside the SDC to indicate, if it is compliant to PT or DC or any other tool. In such cases, it is better to clarify this with the designer. The SpyGlass Constraints solution, by default, assumes the SDC to be PT compatible.

The SpyGlass Constraints can read in pure SDC format or may use the standard Tcl syntax. Here are the various components of a Tcl-based SDC file:

1. **Pure Tcl constructs**: These include native Tcl construct, such as `for each`, `while`, and `if-else`. These are sent to Tcl interpreter, and are handled completely by SpyGlass.
2. **Control type commands**: These include commands, which are extensions (in DC/PT shell) to Tcl. Since these are not part of standard Tcl, but are extensions to Tcl provided in the DC/PT shell, SpyGlass might not be able to read all of them. SpyGlass has support for the most commonly used items.
3. **Actual SDC commands**: These include commands, such as `create_clock` and `set_input_delay`, which are part of the SDC syntax as defined by Synopsys. SpyGlass currently supports SDC 2.0.
4. **Options to SDC commands**: For certain SDC commands, DC/PT shells might support additional options, while these options are not part of SDC.
5. **Non-SDC Commands**: DC/PT shells might support certain commands to apply constraints, but these commands are not part of SDC. SpyGlass has a limited support for these commands.

The workaround is often possible for non-SDC commands rejected by the SDC parser. If the constraints are specified using some commands of the shell (of another tool), read these commands in the native tool (for which these commands were written) and write out the SDC from that tool (for example, the `write_sdc` command for DC) and use that for analysis. Usually, most synthesis/STA tools provide a way to write out the commands in equivalent SDC format.

Alternatively, you can make the SDC parser ignore the commands by specifying them in a file and defining the filename through the `-tc_ignored_commands` parameter. You can do this in the Console GUI

or Tcl by using the following command:

```
set_parameter tc_ignored_commands '<file-name>'
```

After tool compatibility is ascertained, check the SDC to remove any reference to the .db file. If the SDC was generated for DC/PT, it may contain such references. If there are references to .db, obtain .lib and compile to .sglib through the SpyGlass Library Compiler.

For more details on creating .sglib using the SpyGlass Library Compiler, refer to the *SpyGlass Explorer User Guide* and the *LC Parser User Guide*.

Gather Design Data

This step ensures that the tool is provided the proper inputs. This step consists of gathering all required design files, library files and design constraints file, which may be combination of Tcl and SDC.

After the design category has been established, ascertain if the design is at RTL-level or at Gate-level. It is important to ensure that for an RTL design, you have provided a constraints file that references only the design object in the RTL. Typically designers use the terminology “constraints” for the RTL stage, since these are typically a combination of native Tcl and SDC constructs.

For the netlist stage, designers refer to this as “SDC”, because they are generated from within the tool. These are typically pure SDC files with no Tcl constructs.

NOTE: *This is a general norm and not a mandatory practice. These terms may get used interchangeably.*

Refer to *Appendix A: SpyGlass Constraints Design Data Checklist* for a list of data required for the SpyGlass Constraints run.

Configure SpyGlass Design Constraint (SGDC) File

After you have gathered all the required input files, specify information that is not available in the RTL (or Netlist). For example, the location of the SDC file. You can do this by a constraints file known as the SpyGlass Design Constraint (SGDC) file. The Clock/Reset information is needed only for the step when you are creating the SDC file for the first time. For the rest of the goals, clock information is already available from the SDC file.

Before performing the constraints analysis, make sure that the SGDC file is configured correctly. If the file is not configured correctly, you may see more warnings than expected, and a majority of these warnings could be false. Given below is an example of an SGDC file.

```
current_design top
sdc_data –file top_rtl.sdc
```

SGDC File Specification for Constraints Analysis

If the block under analysis includes multiple SDC files, set up the SGDC file as:

```
current_design top
sdc_data –file top_rtl1.sdc top_rtl2.sdc top_rtl3.sdc
```

This scheme is required only when the design requires separate SDC files. If the top-level SDC file sources other files as part of the Tcl script, you need to define the top-level SDC file. Then, SpyGlass extracts the other files/information as defined.

If checks require the SDC file for CHIP, SUBCHIP, and BLOCK level to be analyzed simultaneously, set up the SGDC file as:

```
current_design top
sdc_data -file <top-design-SDC-file-list> ...
block -name blockA sub-chipB

current_design blockA
sdc_data -file <blockA-SDC-file-list> ...

current_design sub-chipB
sdc_data -file <sub-chipB-SDC-file-list> ...
block -name blockC

current_design blockC
sdc_data -file <blockC-SDC-file-list> ...
```

The SpyGlass Constraints solution requires you to specify the clock nets in the design if you are planning to create constraints using the solution. This can be generated automatically using the SpyGlass CDC solution or created manually. Given below is an example of an SGDC file.

```
current_design top

clock -name "top.clka" -domain domain1

clock -name "top.clkb" -domain domain2
```

After setting up the constraint file, proceed with constraint analysis and/or creation.

Run Sanity Checks on Inputs

After you have collected design data and created the SGDC file needed to run the SpyGlass Constraints solution, load the design in SpyGlass to sanitize any errors or discrepancies in the design.

Before starting any analysis of the SDC file, ensure that RTL or netlist is “lint” clean using Connectivity, Structure, Synthesis, Simulation, and Clocks goals in the GuideWare installation. However, if you have customized these into different local goals, ensure that all goals taken together include these goals.

Errors in the SDC file relate mainly to syntactic correctness and compliance to the associated design. Syntax problems are typically fixed manually. Inconsistency between SDC and design may be because of many reasons:

- Either the design is incomplete. For example, objects referred in the SDC file are missing in the design.
- The library models do not have all the information.
- Many modules are black boxed.
- Design is not available because the module is an external IP.

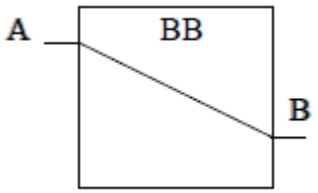
The module may be black boxed for several reasons:

- It may not have any kind of definition.
- It may only have a .lib file definition without any functionality and/or timing arcs specified in it.
- It may be not be synthesizable.

Since it is not possible to perform structural analysis on modules that are

not synthesizable, checks may fail to detect combinational loops and clock domain crossings. The relationship between inputs and outputs of a black box can be specified in the SGDC file, if they cannot be inferred from the .lib file. This is done using the `assume_path` command. You will have to provide more design information or get a more complete SDC to remove the inconsistencies.

```
assume_path –name BB –input A –output B
```



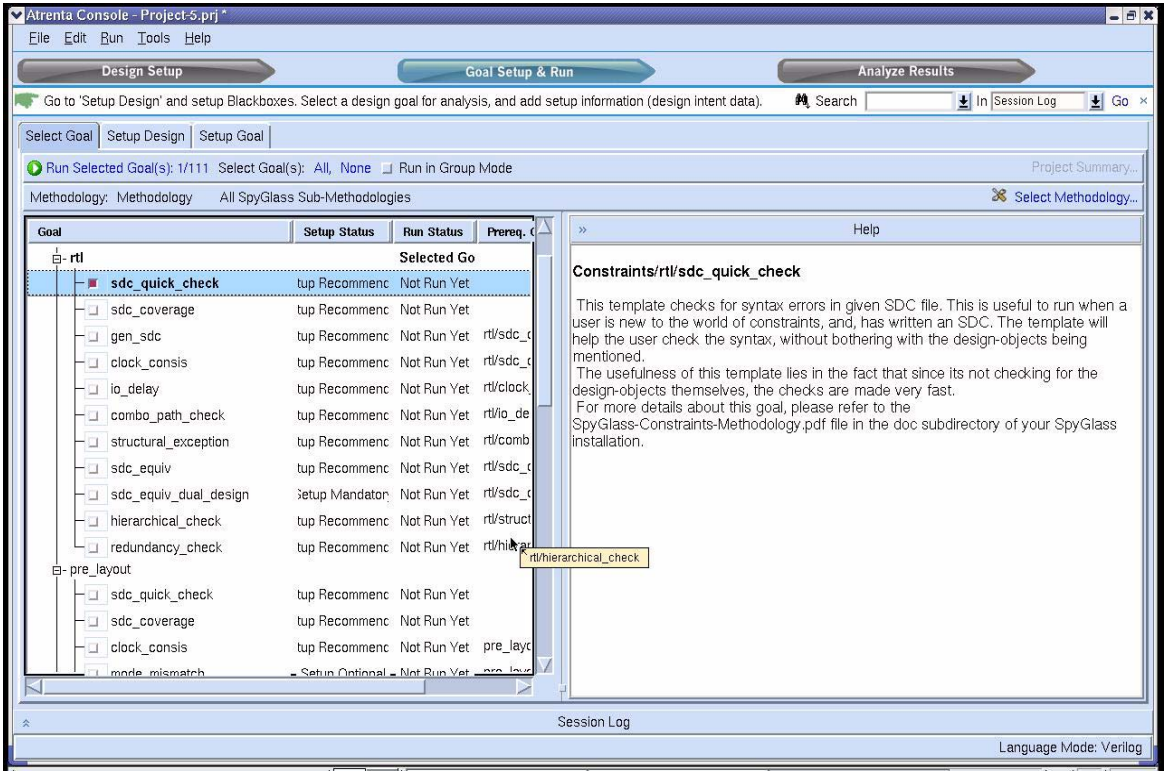
Refer to *SpyGlass Explorer User Guide* for detailed steps on how to read a design using SpyGlass. In some cases, where module is an external IP, you may have to waive these messages. Refer to section on *Waiving Messages*.

NOTE: *Use Save/Restore to prevent reread/resynthesis during multiple runs. This helps the run time especially if the design is large.*

Check the Coverage of the Constraints

After you have run the sanity check, it is a good idea to see the coverage the SDC file provides on your design. This is a good way to quantify the portions of the design that are not constrained. This generates a report of unconstrained ports and registers and the reason for detecting them as unconstrained. The report can help you to perform a more detailed analysis using the methodology steps detailed below.

To generate this report, run the `sdc_audit` goal.



Generate Constraints

This step is recommended only for RTL designs. Even though you can create a template for a netlist, it is not a recommended practice. It can result in extremely long run times. In addition, a netlist created with an incorrect or missing SDC at the RTL-level may be incorrect to begin with.

Objective

In the prior step where the SGDC file is populated to associate SDC files with the appropriate block. However, if there are blocks for which the constraints file is missing, this step is used to automatically generate an SDC template from the RTL. This template can be created incrementally and include the following SDC statements:

- Clocks with periods and waveforms as placeholders
- Generated clocks with an appropriate clock source
- Inputs/outputs tied to correct clocks or virtual clocks
- Clock constraints, such as latency, uncertainty, and transition
- Input transition and loads
- Minimum and Maximum delay for feed-through paths
- Set case analysis on the select pin reaching the multiplexer of a clock fan-in
- False paths to asynchronous clock domains

At each incremental step, you can specify a seed SDC file, which can contain constraints generated in an earlier step or a legacy file. From this seed file, you can inherit constraints. This approach gives you the flexibility to have a check and balance on constraints after each step. For example, you can first constrain all clocks and make sure the design needs are met. Then, you can incrementally add the I/O delays. After all ports are constrained, you can add additional constraints, such as exceptions.

- User controlled generation of the following constructs through a side file specified using the `gen_sdc_constraints_file` parameter. You can do this in the Console GUI or Tcl by using the following command:

  ```
  set_parameter gen_sdc_constraints_file '<file-name>'
  ```

  You can list the constraints that have to be generated in this file. The default file (gensdcConstraintsFile.txt) for this parameter is located at $SPYGLASS_HOME/policies/constraints. The following commands are currently supported:

  - create_clock
  - create_generated_clock
  - virtual_clock
  - set_case_analysis
  - set_input_delay
  - set_output_delay
  - set_clock_latency
  - set_propagated_clock

- ❒ set_clock_uncertainty
- ❒ set_clock_transition
- ❒ set_input_transition
- ❒ set_driving_cell
- ❒ set_drive
- ❒ set_load
- ❒ set_min_delay
- ❒ set_max_delay
- ❒ set_false_path (for clock domain crossings only)

Prerequisites

This requires the correct identification of all clocks in the SGDC file prior to running the rule. This can be done using the `cdc_setup` goal in the SpyGlass CDC solution. The SGDC file should look as follows

```
current_design <design_name>
clock -name "<design_name>.<clock1_name>" -domain domain1
clock -name "<design_name>.<clock2_name>" -domain domain2
```

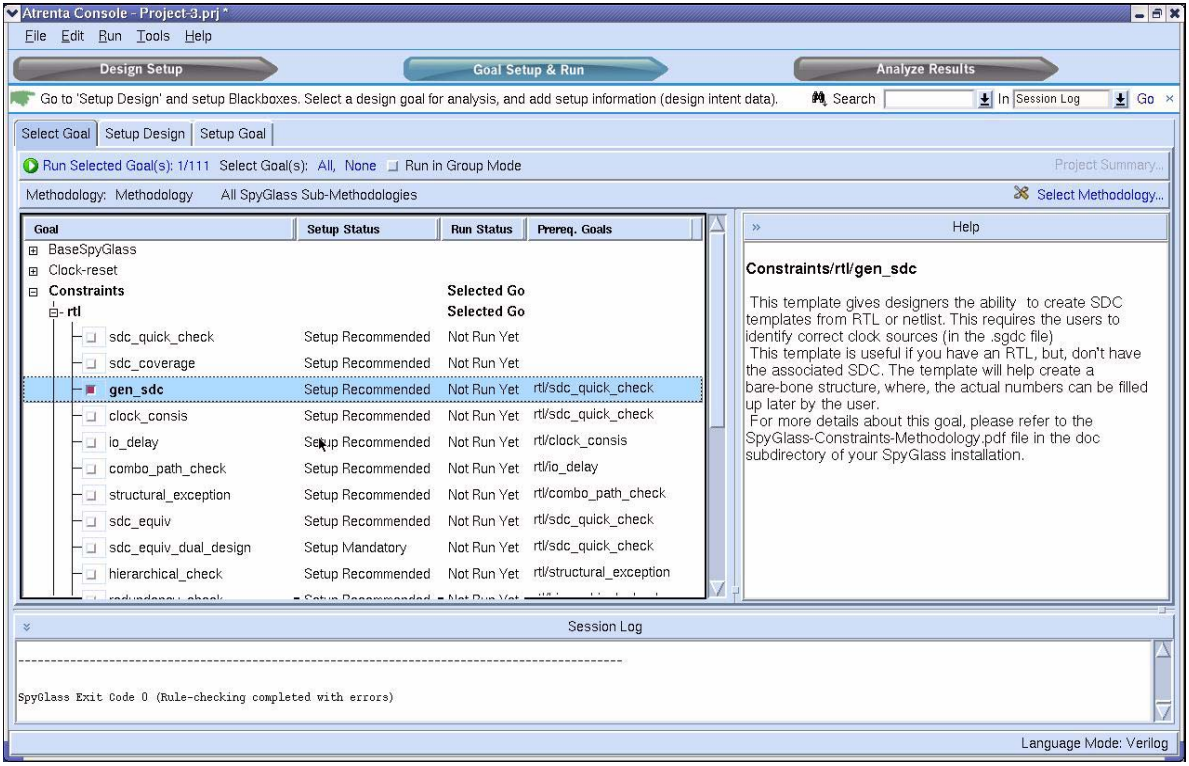
The clock period can be specified in the SGDC file and this is imported directly into the generated template.

```
current_design <design_name>
clock -name "<design_name>.<clk1_name>" -domain domain1 –
period 10
```

If the user specifies `set_case_analysis` constraints in the SGDC file, these constraints are used to define the `set_case_analysis` in the SDC template.

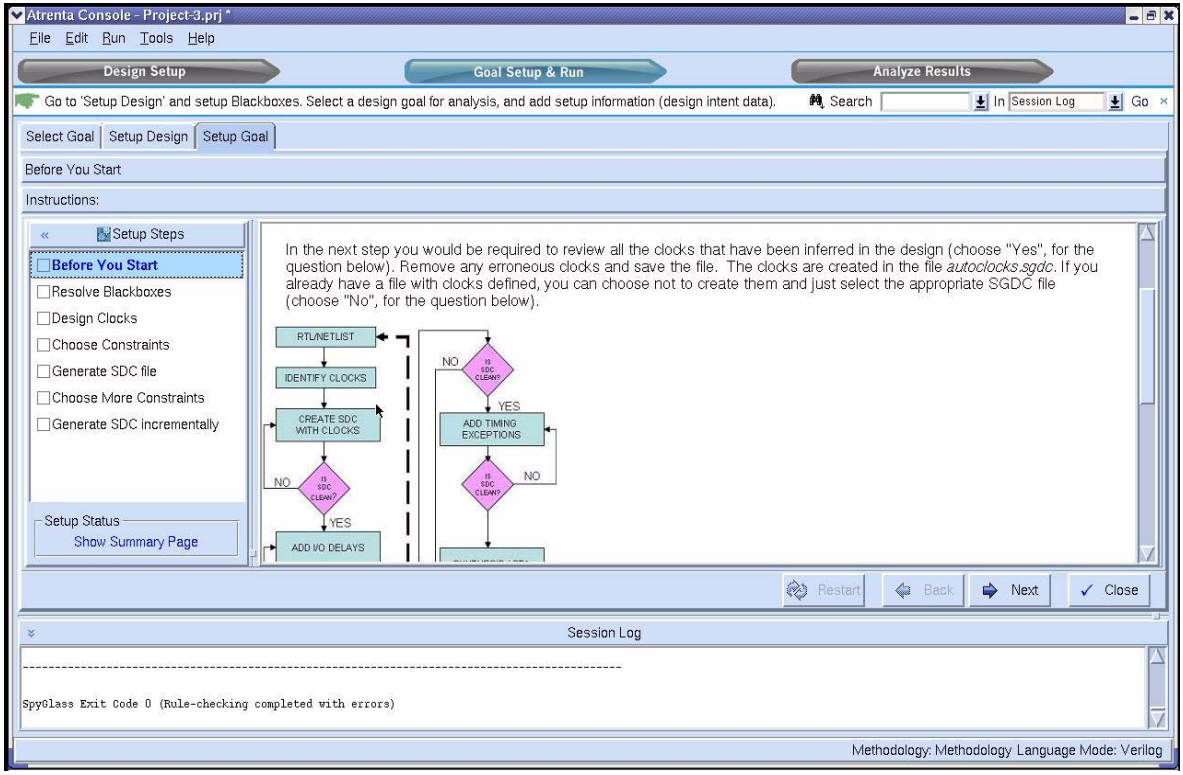
SDC Generation in the Console GUI

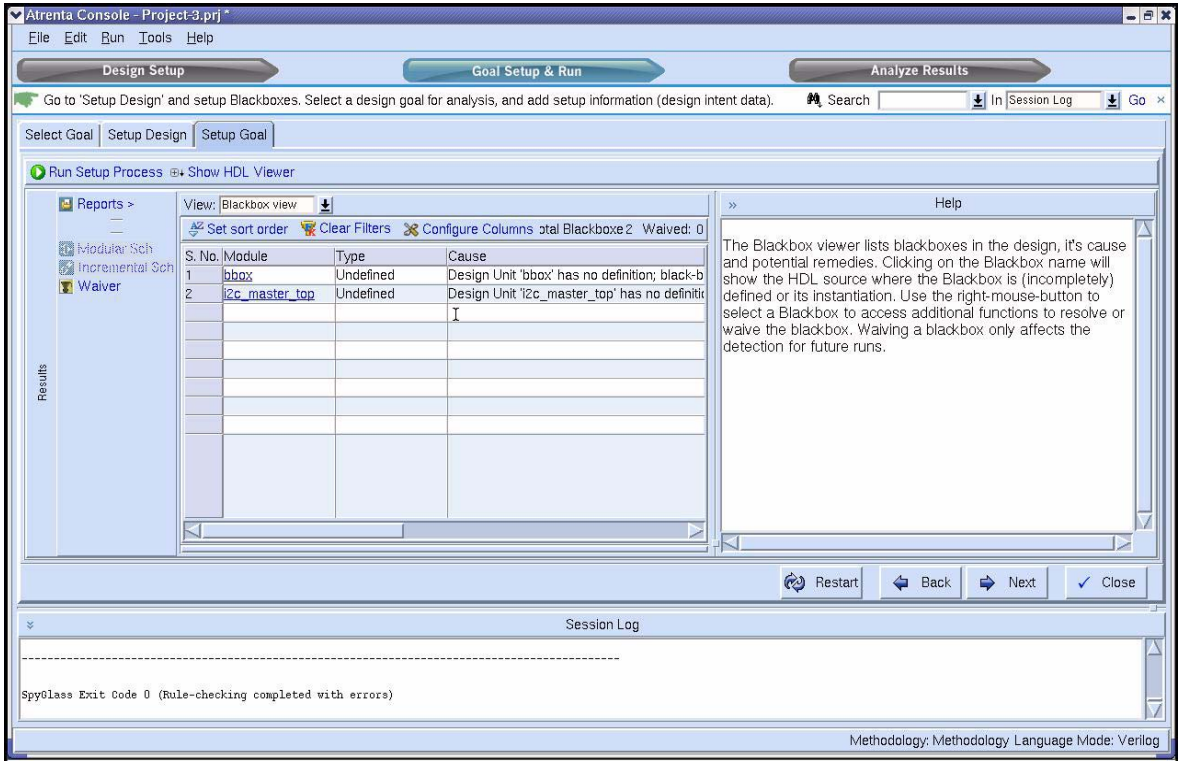
1. Choose the `sdc_gen` goal in the Goal Selection window and perform the steps in the Goal Setup.



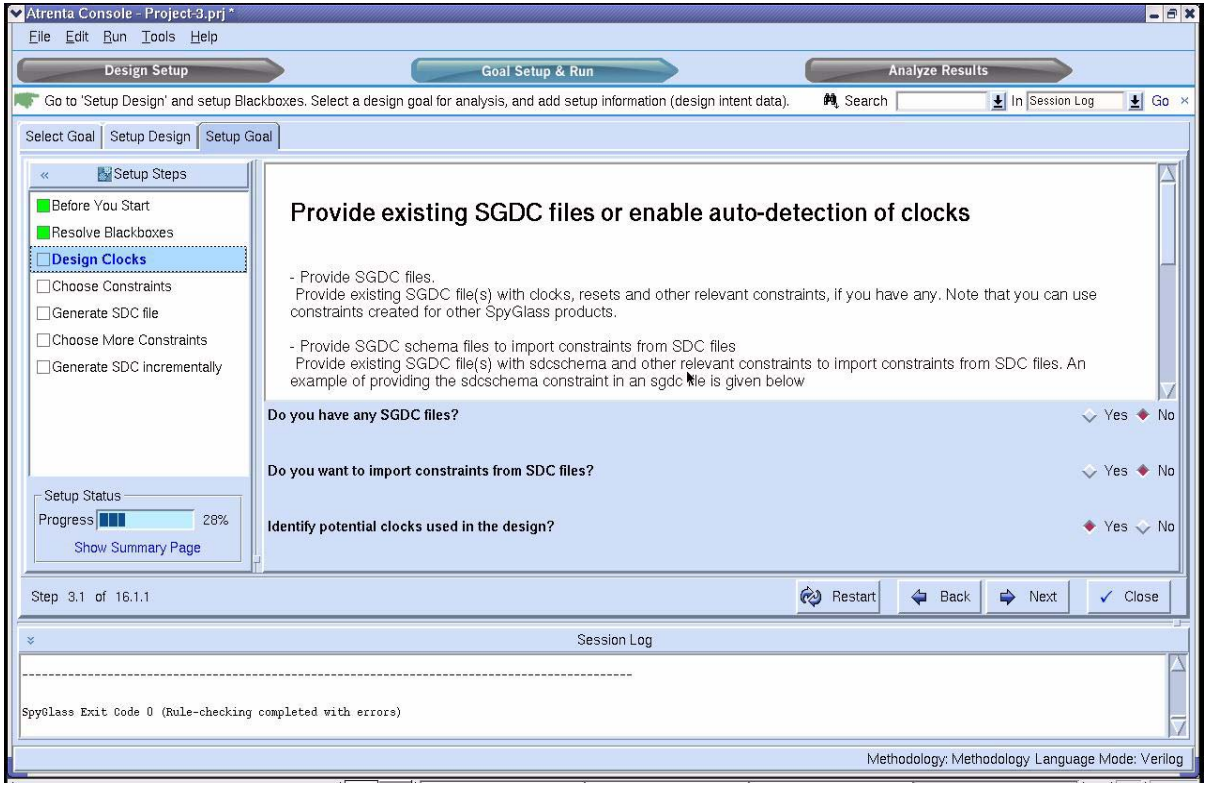
2. The setup guides you through all the steps. After the introduction, the first step is to resolve all the black boxes.

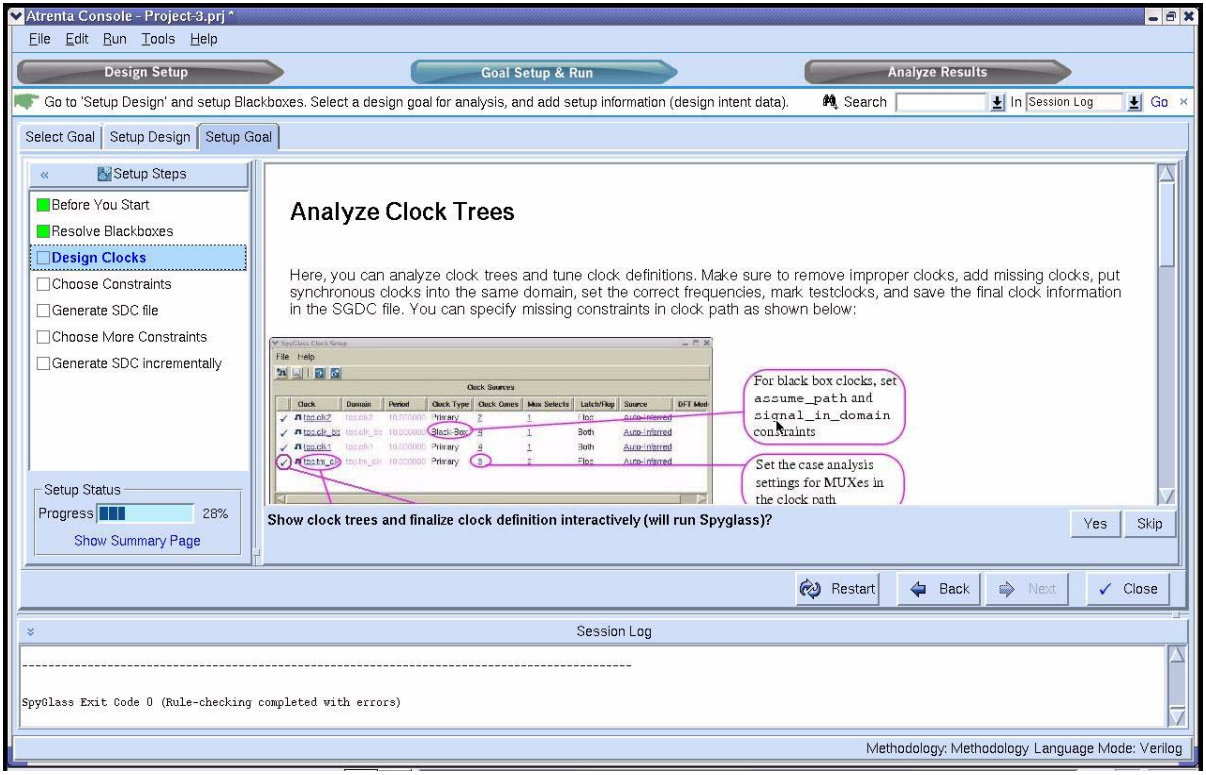
Step-by-Step Solution

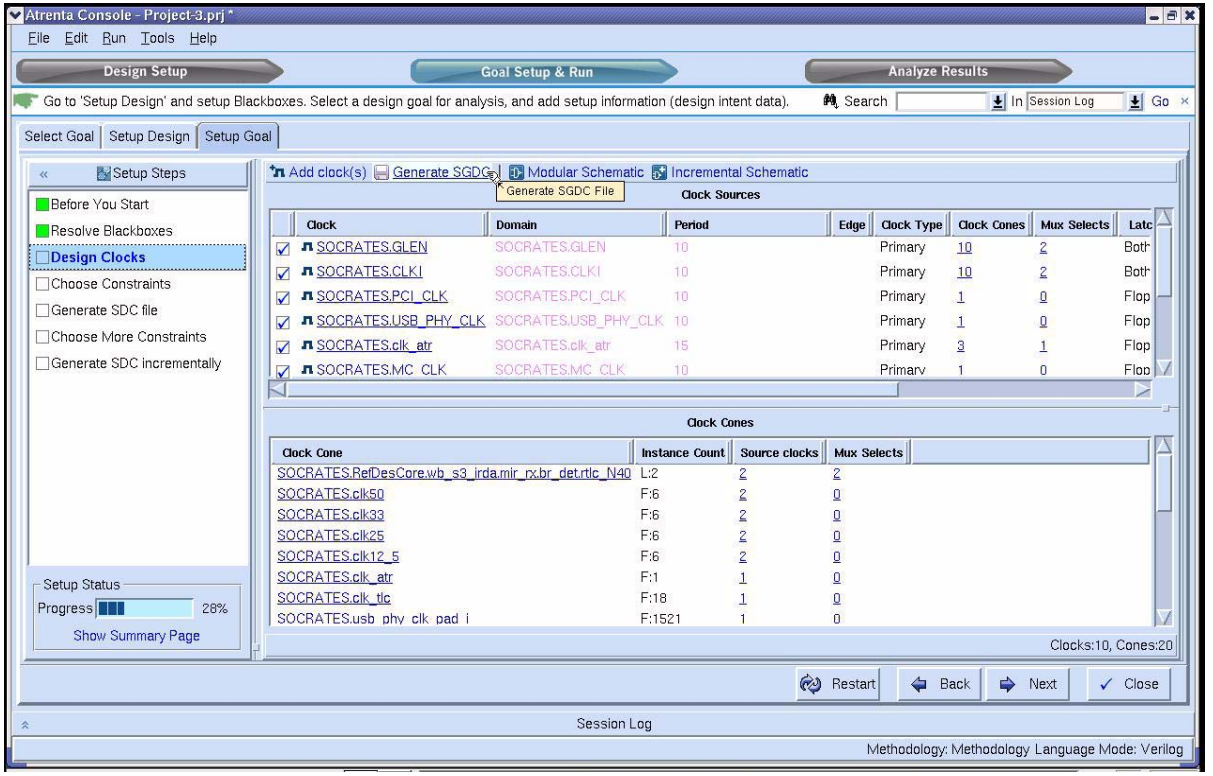




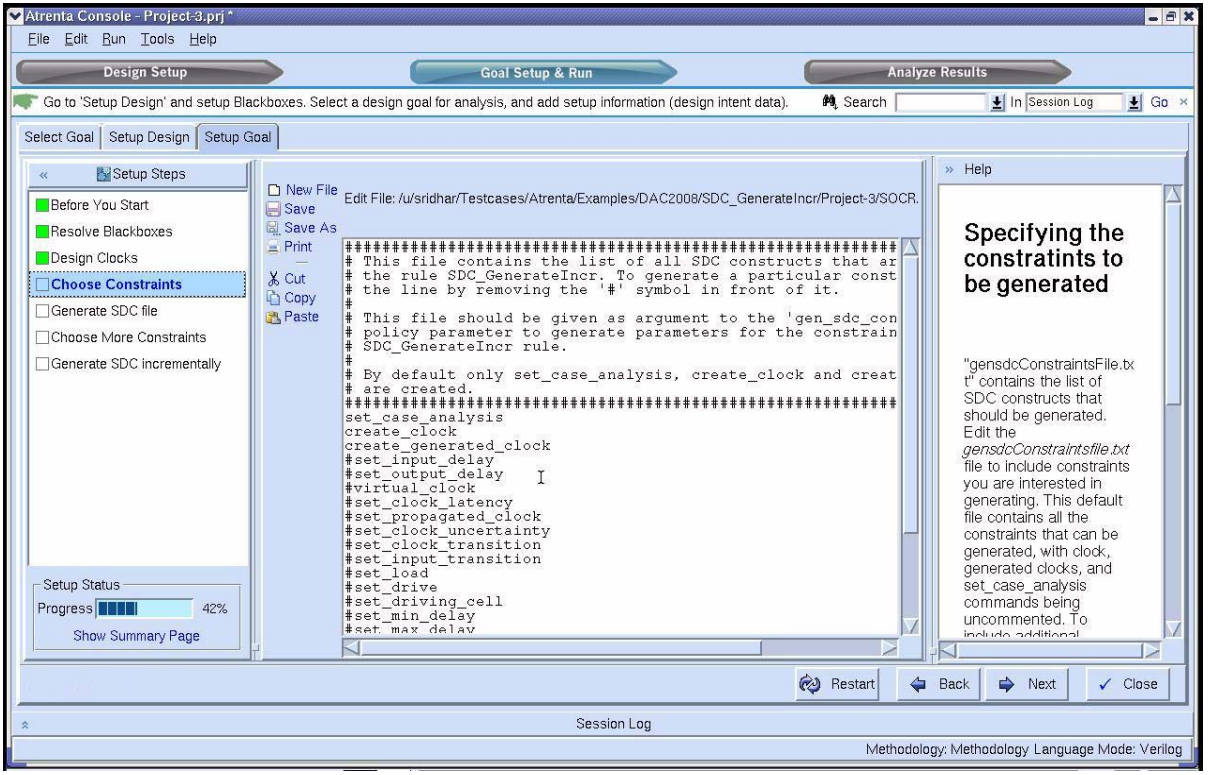
3. After the black boxes have been resolved the next step is to identify all the clocks in the design. You can choose whether you want to infer the clocks automatically or want to specify an SGDC file with clocks defined. When you run this set up process, SGDC files are created based on clocks traced back from clock pins. Remove any erroneous clocks due to top-level gates in the generated SGDC file and save it.


Atrenta Console - Project-3.prj *
File Edit Run Tools Help
Design Setup
Goal Setup & Run
Analyze Results
Go to 'Setup Design' and setup Blackboxes. Select a design goal for analysis, and add setup information (design intent data).
Search
In Session Log
Go
Select Goal
Setup Design
Setup Goal
Setup Steps
Before You Start
Resolve Blackboxes
Design Clocks
Choose Constraints
Generate SDC file
Choose More Constraints
Generate SDC incrementally
Setup Status
Progress
28%
Show Summary Page
Provide existing SGDC files or enable auto-detection of clocks
- Provide SGDC files.
Provide existing SGDC file(s) with clocks, resets and other relevant constraints, if you have any. Note that you can use constraints created for other SpyGlass products.
- Provide SGDC schema files to import constraints from SDC files
Provide existing SGDC file(s) with sdcschema and other relevant constraints to import constraints from SDC files. An example of providing the sdcschema constraint in an sgdc file is given below
Do you have any SGDC files?
Yes
No
Do you want to import constraints from SDC files?
Yes
No
Identify potential clocks used in the design?
Yes
No
Step 3.1 of 16.1.1
Restart
Back
Next
Close
Session Log
SpyGlass Exit Code 0 (Rule-checking completed with errors)
Methodology: Methodology Language Mode: Verilog


Atrenta Console - Project-3.prj *
File Edit Run Tools Help
Design Setup
Goal Setup & Run
Analyze Results
Go to 'Setup Design' and setup Blackboxes. Select a design goal for analysis, and add setup information (design intent data).
Search
In
Session Log
Go
Select Goal
Setup Design
Setup Goal
Setup Steps
Before You Start
Resolve Blackboxes
Design Clocks
Choose Constraints
Generate SDC file
Choose More Constraints
Generate SDC incrementally
Setup Status
Progress
28%
Show Summary Page
Analyze Clock Trees
Here, you can analyze clock trees and tune clock definitions. Make sure to remove improper clocks, add missing clocks, put synchronous clocks into the same domain, set the correct frequencies, mark testclocks, and save the final clock information in the SGDC file. You can specify missing constraints in clock path as shown below:
For black box clocks, set assume_path and signal_in_domain constraints
Set the case analysis settings for MUXes in the clock path
Show clock trees and finalize clock definition interactively (will run Spyglass)?
Yes
Skip
Restart
Back
Next
Close
Session Log
SpyGlass Exit Code 0 (Rule-checking completed with errors)
Methodology: Methodology Language Mode: Verilog



4. Specify the constraint that you want to generate. The setup will provide you with the default file, gensdcConstraintsFile.txt, for specifying the constraints. Edit the gensdcConstraintsFile.txt file to specify the constraints that you want to generate in the first run.



5. The setup then prompts you for creating a parameterized SDC. Depending on your response, a different kind of SDC would be created. If you click "Yes", all placeholders are replaced by Tcl parameters and you only need to set the values to the parameters. For example, the SDC file will look like the following:

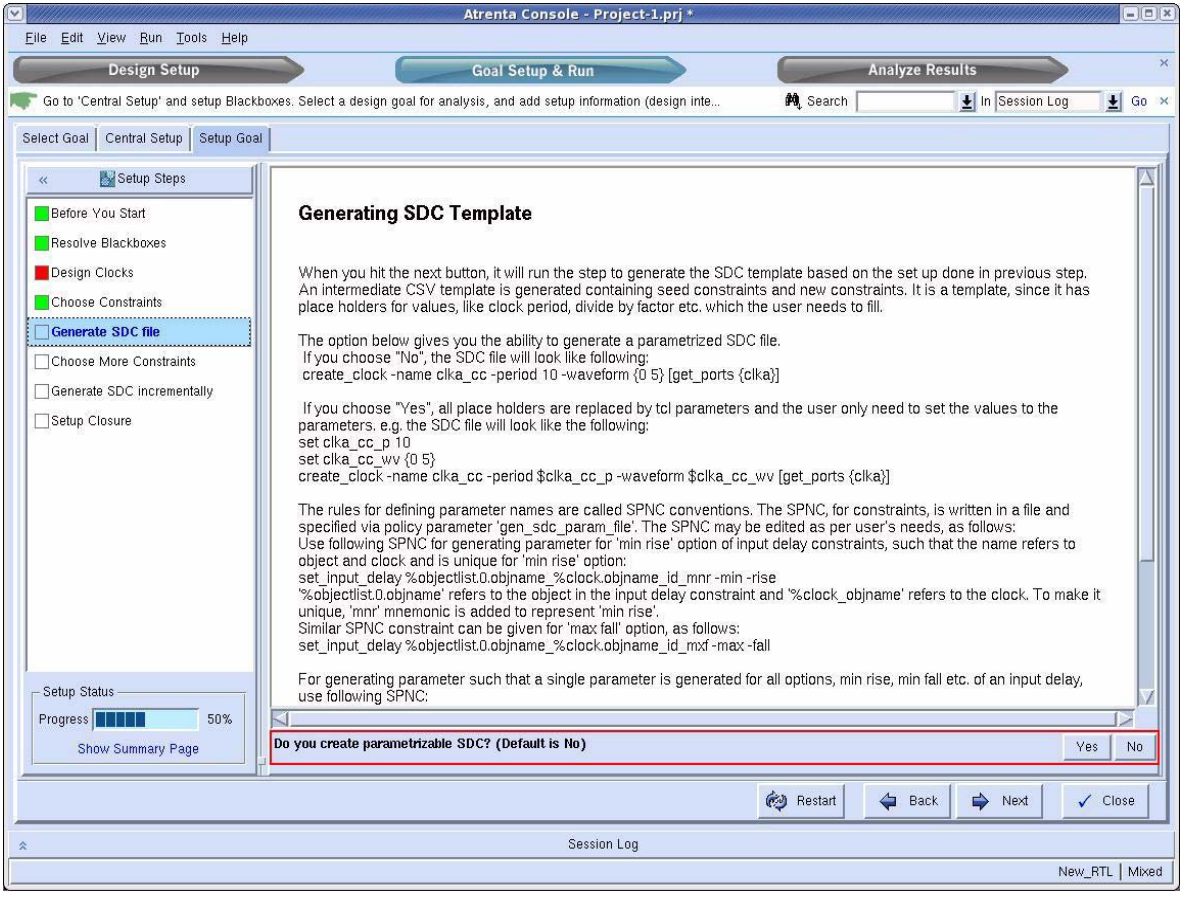
```
set_clka_cc_p 10

set clka_cc_wv {0 5}

create_clock -name clka_cc -period $clka_cc_p -waveform
$clka_cc_wv [get_ports {clka}]
```

If you choose "No", the SDC file will look like:

```
create_clock -name clka_cc -period 10 -waveform {0 5}
[get_ports {clka}]
```

This step will complete the setup for SDC Generation. When you click

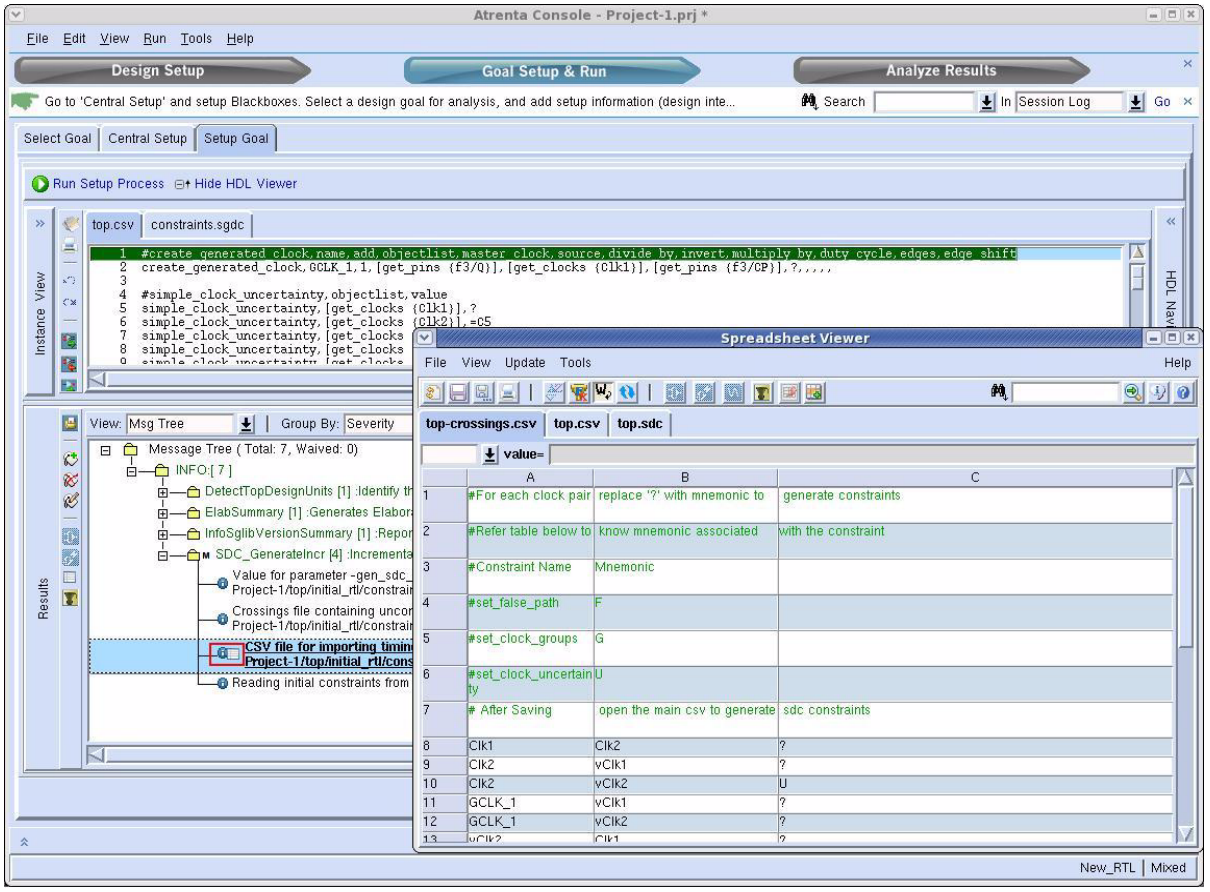
the **Next** button, the constraints are generated.



After the generation is complete, look for the `SDC_GenerateIncr` rule in the INFO category. This is the rule that was run to create the SDC template. Double-click the message with a spreadsheet icon. The Spreadsheet Editor appears. You can use this editor to modify the clock crossing file, if generated, and the placeholders in the generated template.

The clock crossing file is generated if there are unclassified interacting clock pairs in the design. Before you can go ahead with the population of the template, review all clock crossing information to indicate the relationships between clock pairs. This information is used to generate false paths or interclock uncertainty or clock groups between clock pairs.

You cannot edit the template SDC file before classifying the clock crossing file.



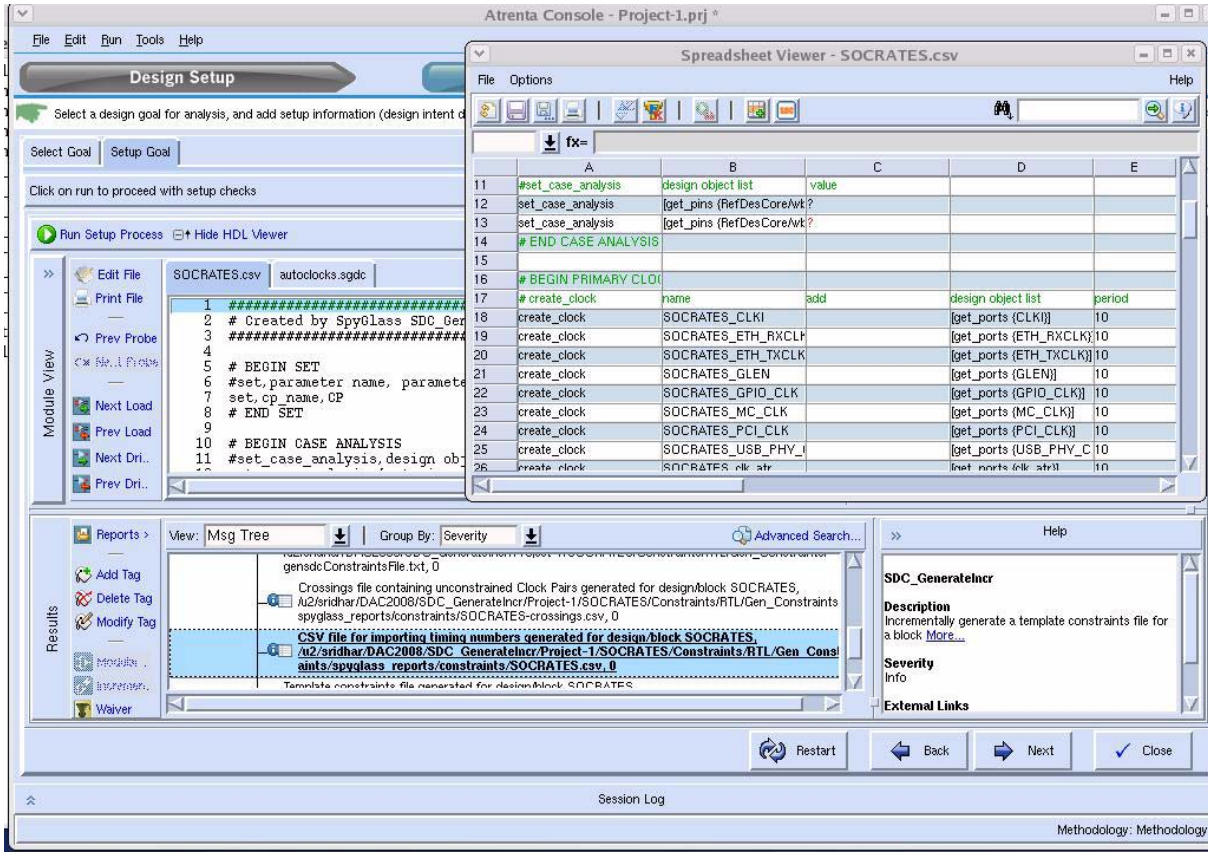
As you can see, the clock crossing file contains **?** (question mark) character. This character signifies that the clock pair is unclassified. To classify the clock pairs, replace the **?** character for each pair as described in the following table.

Replace ? Character with...	To Generate...	Relationship
F	set_false_path	Not applicable
U	set_clock_uncertainty	Not applicable

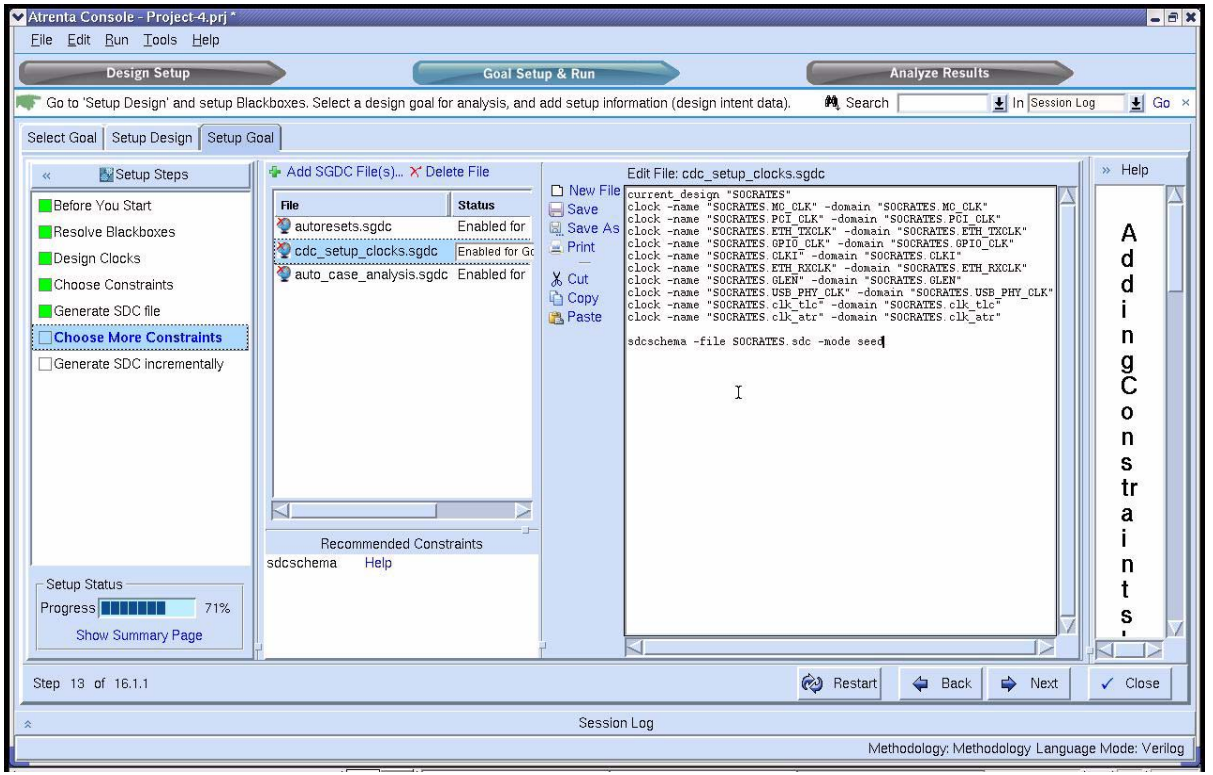
GA	set_clock_groups	asynchronous
GL	set_clock_groups	logically_exclusive
GP	set_clock_groups	physically_exclusive

After making changes for all clock pairs, save the changes. The Console GUI displays the status of the crossing pairs classification. The CSV file contains an updated list of `set_false_path`, `set_clock_uncertainty`, and/or `set_clock_groups` constraints specified.

After you have completed the clock crossing file classification, click the **Update** button to save the file. You cannot update the crossing file after you have saved it. Now, enter all the timing values in the template SDC file and generate the output SDC by clicking the **Generate SDC** option in the **File** menu of the Spreadsheet Editor.



6. The SDC generated in the previous step will act as the seed SDC for incremental addition of more SDC constraints. Modify the SGDC file to include this SDC as the seed file.



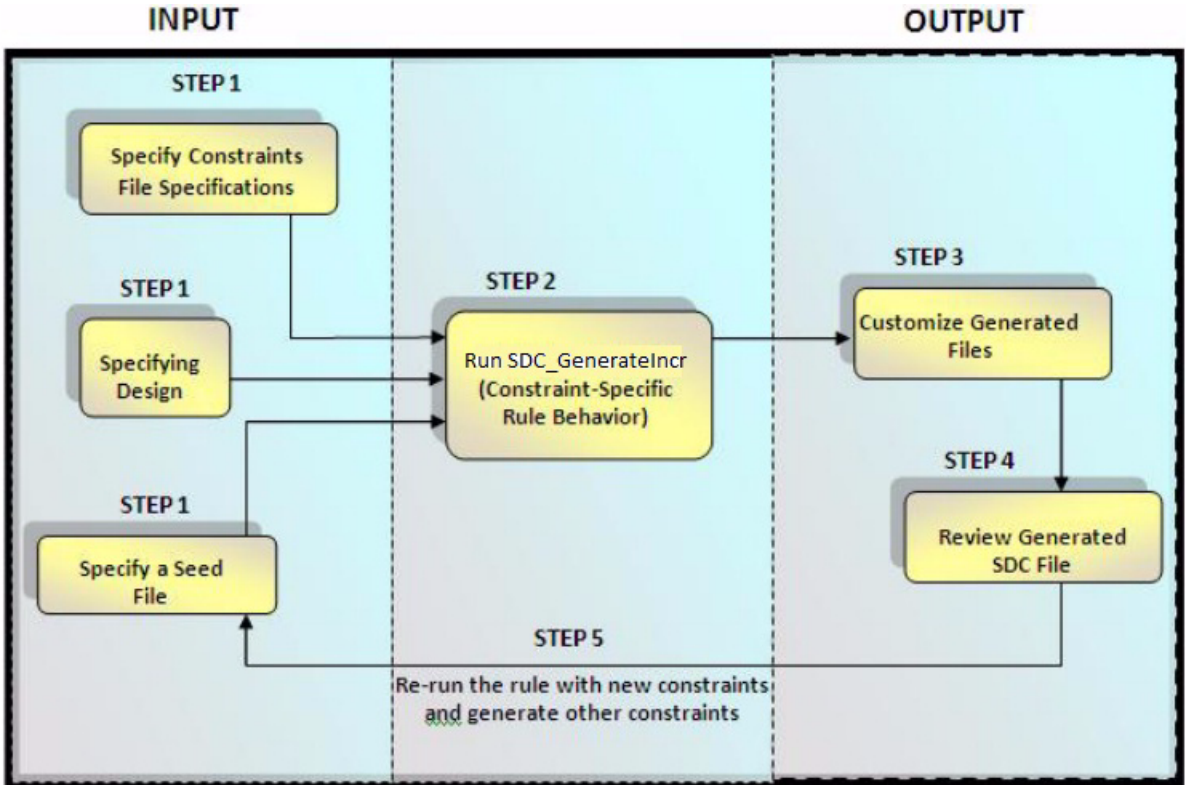
7. The next step is exactly same as step 4. You can choose additional constraints add incrementally to the seed SDC file. After the constraints have been added, populate the template using the Spreadsheet Editor.

Exit Criteria

A clean run with a generated SDC template is the exit criteria for this step.

SDC Generation in Batch Mode

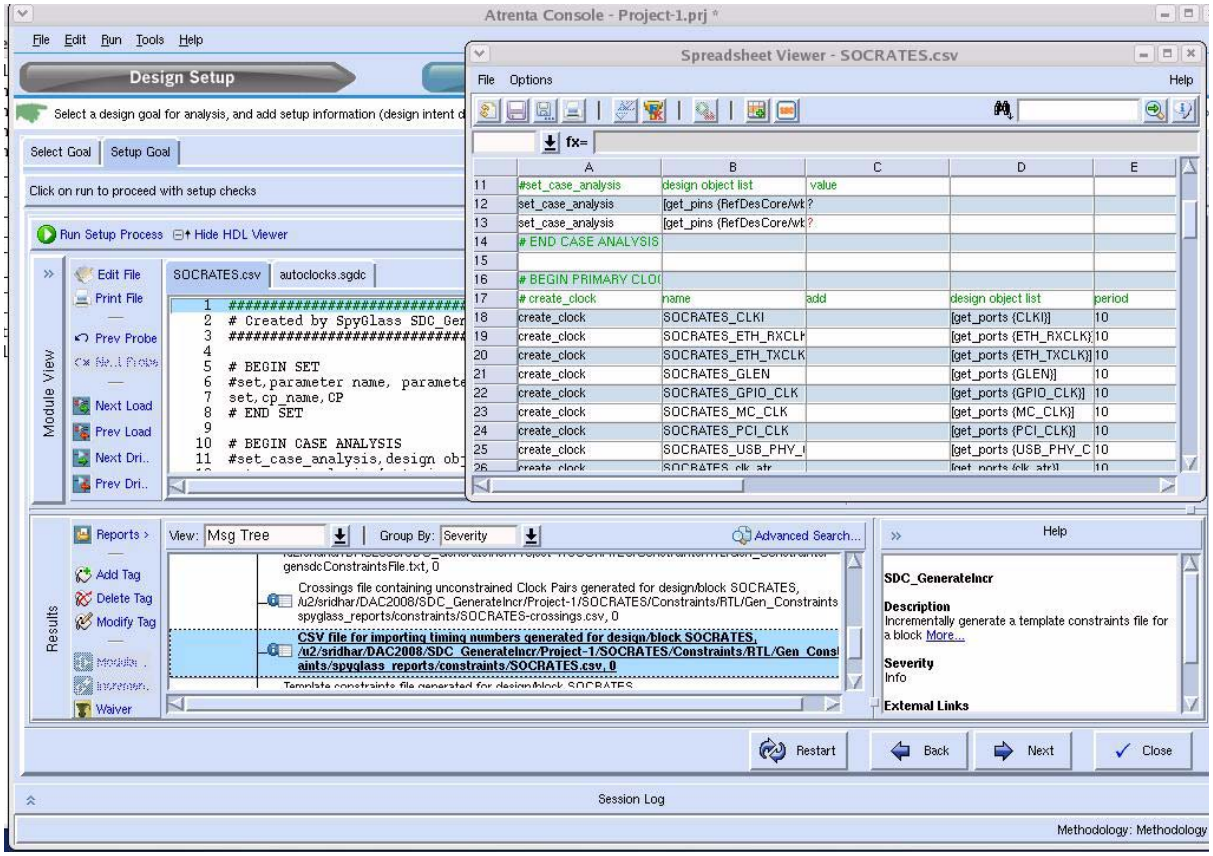
The following figure illustrates the steps for running `sdc_gen` rule in batch mode.



For the first run, initial constraints are provided through a seed SDC file or when you do not have a seed SDC file, you can create clock specification in SGDC and then run the `sdc_gen` goal

```
spyglass -project design.prj -goal initial_rtl/constraints/
sdc_gen -batch
```

After running the goal, the project can be loaded in the Console GUI, and the SDC file can be generated through the rule interface. The generated file can be used as the seed file to generate more constraints.



Block/IP Methodology Flow

This stage involves the development of a new RTL. The process of the development of a new RTL goes through progressive RTL refinement starting with simpler goals that meet the basic timing requirements, such as clean clocks and I/O delays. As the RTL code and design constraints mature, the design goals evolve to include exceptions, constraints redundancy and constraints equivalence to ensure that design intent is preserved.

Refer to the *Goals for Block/IP* section for details of all the goals for the Initial RTL Development, RTL Handoff, and Netlist Handoff goals.

SpyGlass Constraints Block/IP Quick Start

No.	Step	Prerequisite	Goal	More Information
1	Check Design Coverage and Extract Domain Information	Since domains are extracted based on set_false_path, set_clock_group, and set_clock_uncertainty constraints, violation messages from previous steps for these constraints need to be cleaned up before executing the step.	`sdc_audit`	*Step 1 – Check Design Coverage and Extract Domain Information*
2	Basic Consistency, Clean Clock Definition, and Delay Constraints	All design files must have passed the tests performed in the setup phase.	`sdc_check`	*Step 2 – Check Consistency, Clean Clock Definition, and Delay Constraints*
3	Check Timing Exceptions Structurally	It is recommended that timing constraints are cleaned up before you clean up exceptions.	`sdc_exception_struct`	*Step 3 – Check Timing Exceptions Structurally*
4	Remove Redundancy and Achieve Better Retargeting	All mandatory steps are completed and cleaned.	`sdc_redundancy`	*Step 4 – Remove Redundancy and Achieve Better Retargeting*

Block/IP Detailed Procedure

Perform the following steps for this flow:

- *Step 1 – Check Design Coverage and Extract Domain Information*
- *Step 2 – Check Consistency, Clean Clock Definition, and Delay Constraints*
- *Step 3 – Check Timing Exceptions Structurally*
- *Step 4 – Remove Redundancy and Achieve Better Retargeting*
- *Step 5 - Generate Abstract View of the Block*

Step 1 – Check Design Coverage and Extract Domain Information

Objective

This step enables you to compute the design coverage and identify uncovered design objects, such as ports and registers that have not been constrained. In addition, domain information is extracted from the SDC commands. The extracted information is checked for consistency with the SGDC domain information. You can identify conflicting clock domain classifications in the SDC file.

An SGDC file is generated containing the `clock_group` information inferred from the SDC constraints, which can be used for further analysis. The generated clocks corresponding to source clocks are reported in a tabular format.

Prerequisites

Since domains are extracted based on `set_false_path`, `set_clock_group`, and `set_clock_uncertainty` constraints, violation messages from previous steps for these constraints need to be cleaned up before executing the step.

Executing the Step

Run the `sdc_audit` goal in the Console GUI by entering the following command:

```
spyglass -project design.prj -goal rtl_handoff/constraints/
sdc_audit
```

Step 2 – Check Consistency, Clean Clock Definition, and Delay Constraints

Objective

The objective of this step is to detect the inconsistencies in the specification of clocks and generated clocks and to perform basic checks on overwritten and conflicting constraints. Without clean clock definition, other constraint validation and exception verification is ineffective. Overwritten and conflicting constraints may capture the design intent incorrectly.

This step also detects the inconsistencies in specification of input/output delays, clock latency, and clock uncertainty. Such inconsistencies not only produce incorrect synthesis or static timing analysis results, they can potentially allow these tools to assume a greater slack than available. This

translates to insufficient or incomplete optimization by synthesis, which directly affects the QoR.

Finally, this step checks that all combinational paths are constrained correctly. If a combination path is unconstrained or incorrectly constrained, implementation tools do not perform timing checks on these paths. As a result, the operation of a device at any specified speed cannot be guaranteed.

Pre-requisites

At this stage, all the design files must have passed the tests performed in the setup phase. For details, refer to the *Run Sanity Checks on Inputs* section.

Executing the Step

Run the `sdc_check` goal in the Console GUI by entering the following command:

```
spyglass -project design.prj -goal rtl_handoff/constraints/
sdc_check
```

Exit Criteria

1. All conflicting constraints should be cleaned up.
2. All overwritten constraints should be reconciled or justified.
3. All clock definition issues should be cleaned up. There should be no undefined clocks or clocks with incorrect sources or clocks with inappropriate characteristics.
4. All delay values must be positive numbers.
5. All input and output delays must be associated with the correct clocks.
6. Input/Output delays must meet the slack requirement.
7. All combinational paths must be constrained.
8. The delay numbers must be valid and meet the slack requirements.
9. After cleaning up reported issues, the step should be rerun to ensure that all rules in this step exit with zero violations.

Step 3 – Check Timing Exceptions Structurally

Objective

The objective of this step is to check that timing exceptions specified in a constraints file as are on paths which are structurally connected. This step

is a prerequisite before the paths can be verified formally to be correct. Exceptions set on paths that are not structurally connected are redundant and increase the run time of implementation tools.

Prerequisites

Even though this step can be run right after the sanity checks, it is recommended that timing constraints are cleaned up before you clean up exceptions.

Executing the Step

Run the `sdc_exception_struct` goal in the Console GUI by entering the following command:

```
spyglass -project design.prj -goal design.prj -goal
rtl_handoff/constraints/sdc_exception_struct
```

Exit Criteria

1. All exceptions must be on physically connected paths.
2. Exceptions should not overlap.
3. The step must exit with zero violations for the selected rules.

Step 4 – Remove Redundancy and Achieve Better Retargeting

Objective

The objective of this step is to remove any redundancy in the constraints and perform checks that might facilitate better retargeting.

Prerequisites

This step requires that all mandatory steps are completed and cleaned.

Executing the Step

Run the `sdc_redundancy` goal in the Console GUI by entering the following command:

```
spyglass -project design.prj -goal rtl_handoff/constraints/
sdc_redundancy
```

Step 5 - Generate Abstract View of the Block

Objective

The objective of this optional step is to generate an abstract view of the

block. An abstract view is a representative model of a block that contains relevant block information required during SoC-level verification.

For example, it contains block information, such as combinational path details, boundary registers and related clock/reset information, domain information, and boundary constraints used to analyze the block.

An abstract view contains such information in the form of SGDC constraints. SpyGlass provides a way to generate and consumes these abstract view. For more details, please refer to section on *SoC Methodology using Abstraction*.

Prerequisites

Since the abstract view is a representation of the block, the block must be fully clean before this view is generated, otherwise it can result in false positive and false negative violation messages during the SoC analysis.

Executing the Step

Run `sdc_abstract` goal in the Console GUI by entering the following command:

```
spyglass -project design.prj -goal rtl_handoff/constraints/
sdc_abstract
```

SoC Methodology Flow

During the SoC or subsystem integration, the design architect needs to stitch the block IPs. These block IPs may have been developed internally or selected from a third-party vendor. Depending on the extent of reuse of these IPs, some of them may not have gone through the process of SDC cleaning. This is typically seen in legacy IPs that have existed in prior versions of the design. This creates new challenges during the integration phase.

SoC RTL

This stage involves the verification of an SoC design or a subset of design (subsystem) that has been integrated by using various blocks. This field of use involves checks related to interblock/inter-IP issues and consistency across blocks. In addition, it ensures that block constraints are consistent with SoC constraints.

SoC Netlist

There are two stages in this category

Netlist Handoff

The Netlist Handoff goals check whether a design is ready for floor-planning, layout, and back-end implementation.

Layout Handoff

The Layout Handoff goals check whether a design has gone through specific activities, such as floor-planning and layout, in preparation for tape out.

Refer to *Goals for SoC RTL and Netlist* for the application of the goals in this field of use.

SpyGlass Constraints SoC Quick Start

No.	Step	Prerequisite	Goal	More Information
1	Check Design Coverage and Extract Domain Information	Since domains are extracted based on set_false_path, set_clock_group, and set_clock_uncertainty constraints, violation messages from previous steps for these constraints need to be cleaned up before executing the step.	`sdc_audit`	*Step 1 – Check Design Coverage and Extract Domain Information*
2	Validate Abstract Views	Block-level abstract view has been created.	`sdc_abstract_validate`	*Step 2 – Validate Abstract Views*
3	Check Consistency, Clean Clock Definition, and Delay Constraints	All design files must have passed the tests performed in the setup phase.	`sdc_check`	*Step 3 – Check Consistency, Clean Clock Definition, and Delay Constraints*
4	Check Timing Exceptions Structurally	It is recommended that timing constraints are cleaned up before you clean up exceptions.	`sdc_exception_struct`	*Step 4 – Check Timing Exceptions Structurally*
5	Remove redundancy and Achieve Better Retargeting (Optional)	All mandatory steps are completed and cleaned.	`sdc_redundancy`	*Step 5 – Remove Redundancy and Achieve Better Retargeting (Optional)*

SoC Detailed Procedure

Perform the following steps for this flow:

- *Step 1 – Check Design Coverage and Extract Domain Information*
- *Step 2 – Validate Abstract Views*
- *Step 3 – Check Consistency, Clean Clock Definition, and Delay Constraints*
- *Step 4 – Check Timing Exceptions Structurally*
- *Step 5 – Remove Redundancy and Achieve Better Retargeting (Optional)*

Step 1 – Check Design Coverage and Extract Domain Information

Objective

This step enables you to compute the design coverage and identify uncovered design objects, such as ports and registers that have not been constrained. In addition, domain information is extracted from the SDC commands. The extracted information is checked for consistency with the SGDC domain information. You can identify conflicting clock domain classifications in the SDC file.

An SGDC file is generated containing the `clock_group` information inferred from the SDC constraints, which can be used for further analysis. The generated clocks corresponding to source clocks are reported in a tabular format.

Prerequisites

Since domains are extracted based on `set_false_path`, `set_clock_group`, and `set_clock_uncertainty` constraints, violation messages from previous steps for these constraints need to be cleaned up before executing the step.

Executing the Step

Run the `sdc_audit` goal in the Console GUI by entering the following command:

```
spyglass -project design.prj -goal rtl_handoff/constraints/
sdc_audit
```

Step 2 – Validate Abstract Views

Objective

During the SoC integration the integrator may choose to use the complete block definition or use the abstract view generated during block-level runs. If an abstract view of the block is used, this step must be run to ensure that the block-level assumption for constraints used during abstract creation match the SoC-level constraints. For example, for the clock frequency, the case analysis assumed during the block-level run must match the chip-level requirements.

Prerequisites

Block-level abstract view has been created.

Executing the Step

Run the `sdc_abstract_validate` goal in the Console GUI by entering the following command:

```
spyglass -project design.prj -goal rtl_handoff/constraints/
sdc_abstract_validate
```

Exit Criteria

1. There should be no inconsistency between top-level and block-level constraints.
2. If inconsistencies are identified, the block may have to be rerun to create a new abstract. Alternatively, the abstract view may have to discarded and the flow must be run with the RTL view of the block.

Step 3 – Check Consistency, Clean Clock Definition, and Delay Constraints

Objective

The objective of this step is to detect the inconsistencies in the specification of clocks and generated clocks and to perform basic checks on overwritten and conflicting constraints. Without clean clock definition, other constraint validation and exception verification is ineffective. Overwritten and conflicting constraints may capture the design intent incorrectly.

This step also detects the inconsistencies in specification of input/output delays, clock latency, and clock uncertainty. Such inconsistencies not only produce incorrect synthesis or static timing analysis results, they can potentially allow these tools to assume a greater slack than available. This translates to insufficient or incomplete optimization by synthesis, which directly affects the QoR.

Finally, this step checks that all combinational paths are constrained correctly. If a combination path is unconstrained or incorrectly constrained, implementation tools do not perform timing checks on these paths. As a result, the operation of a device at any specified speed cannot be guaranteed.

Prerequisites

At this stage, all the design files must have passed the tests performed in the setup phase. For details, refer to the *Run Sanity Checks on Inputs* section.

Executing the Step

Run the `sdc_check` goal in the Console GUI by entering the following command:

```
spyglass -project design.prj -goal rtl_handoff/constraints/
sdc_check
```

Exit Criteria

1. All conflicting constraints should be cleaned up.
2. All overwritten constraints should be reconciled or justified.
3. All clock definition issues should be cleaned up. There should be no undefined clocks or clocks with incorrect sources or clocks with inappropriate characteristics.
4. All delay values must be positive numbers.
5. All input and output delays must be associated with the correct clocks.
6. Input/Output delays must meet the slack requirement.
7. All combinational paths must be constrained.
8. The delay numbers must be valid and meet the slack requirements.
9. After cleaning up reported issues, the step should be rerun to ensure that all rules in this step exit with zero violations.

Step 4 – Check Timing Exceptions Structurally

This step is same as *Step 3 – Check Timing Exceptions Structurally* in the RTL stage.

Executing the Step

Run the `sdc_exception_struct` goal in the Console GUI by entering the following command:

```
spyglass -project design.prj -goal rtl_handoff/constraints/
sdc_exception_struct
```

Step 5 – Remove Redundancy and Achieve Better Retargeting (Optional)

This step is same as *Step 4 – Remove Redundancy and Achieve Better Retargeting* in the RTL stage.

Executing the Step

Use the `sdc_redundancy` goal in the Console GUI by entering the following command:

```
spyglass -project design.prj -goal rtl_handoff/constraints/
sdc_redundancy
```

SoC Methodology using Abstraction

Using the Methodology for SpyGlass Constraints Solution

In SoC, you are concerned whether the top-level SoC constraints are matching with the block-level assumptions and SDC constraints. Constraints catering to the internals of a block do not matter in an SoC. These constraints do not impact the SoC-level timing. However, constraints at the block boundary do have an impact at the SoC level.

At the SoC level, the top-level SDC constraints should match the boundary conditions of the block. For example, suppose there is a clock applied with a period of 10 at the SoC level. This clock is connected to a block port, Port A. At the SoC level, it is important to ensure that there is a similar clock with a period of 10 defined in the block level SDC.

This section describes generating and validating an abstract view while using SpyGlass Constraints Solution.

NOTE: *This methodology is valid only for structural SpyGlass Constraints checks, such as SDC checks, equivalence, and mode coverage. However, it is not valid for false path and MCP verification.*

Generating an Abstract View in SpyGlass Constraints

To generate an abstract view while using SpyGlass Constraints solution, run the `sdc_abstract` goal.

The following figure shows the process of generating an abstract view in SpyGlass Constraints solution:

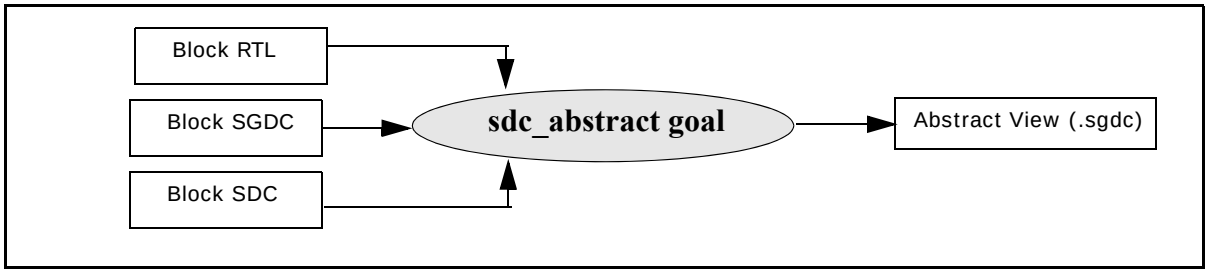


FIGURE 3.

Example - Generating Abstract View in SpyGlass Constraints

Consider the following design, SGDC file, and SDC file as the input for generating an abstract view:

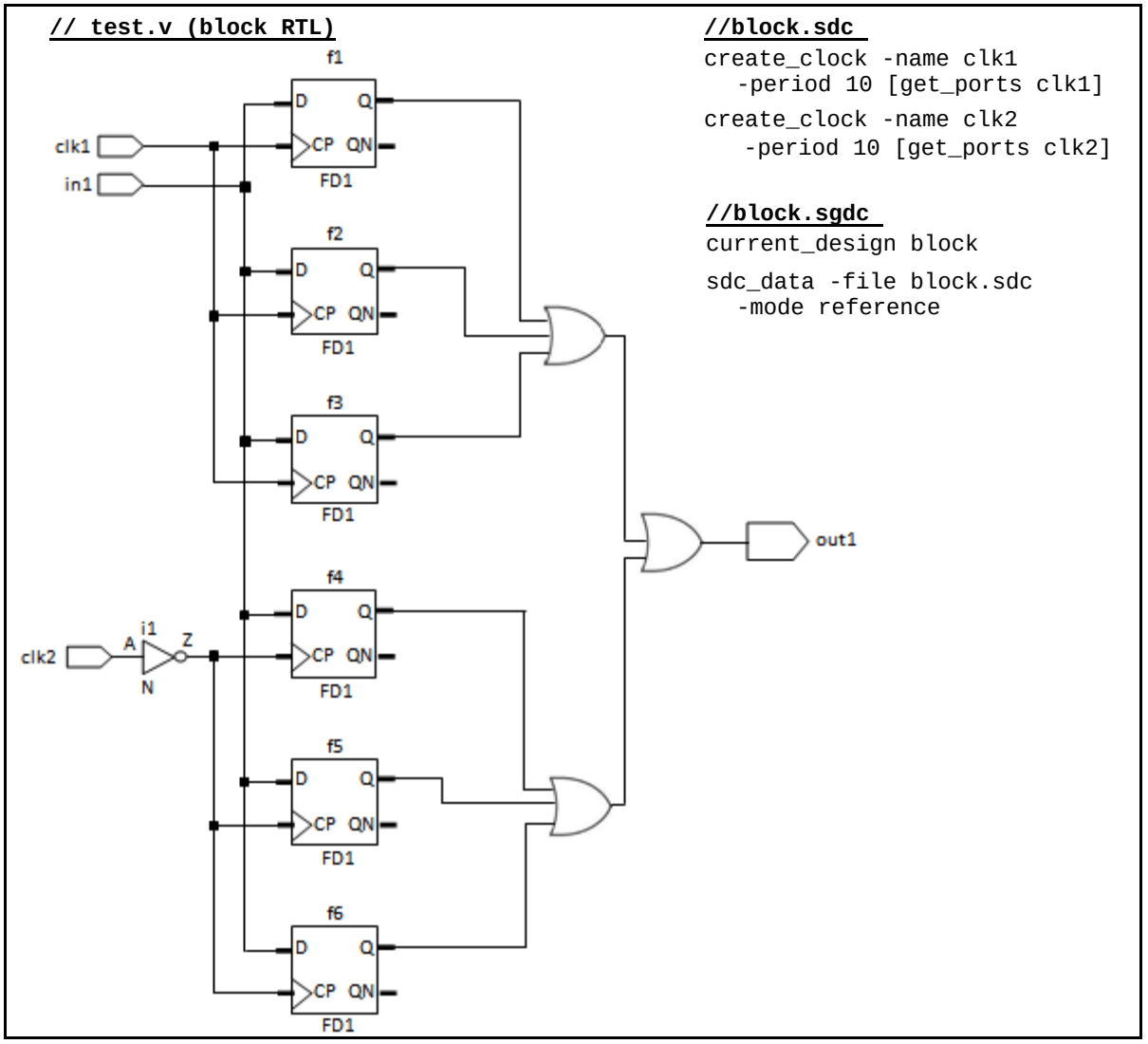


FIGURE 4.

After specifying the above inputs to SpyGlass, run the `sdc_abstract` goal.

When the goal run is complete, SpyGlass generates the following SGDC file that represents the abstract view of the block:

```
abstract_port -ports in -connected_inst "\top.f1 " -inst_pin
D -inst_master RTL_FD -path_logic buf -scope const -mode
reference

abstract_port -ports in -connected_inst "\top.f4 " -inst_pin
D -inst_master RTL_FD -path_logic buf -scope const -mode
reference

abstract_port -ports out1 -connected_inst "\top.f1 "
-inst_pin Q -inst_master RTL_FD -path_logic buf -scope const
-mode reference

abstract_port -ports out1 -connected_inst "\top.f4 "
-inst_pin Q -inst_master RTL_FD -path_logic buf -scope const
-mode reference

abstract_port -ports clk1 -connected_inst "\top.f1 "
-inst_pin CP -inst_master RTL_FD -path_logic buf -scope const
-mode reference

abstract_port -ports clk2 -connected_inst "\top.f4 "
-inst_pin CP -inst_master RTL_FD -path_logic inv -scope const
-mode reference
```

In the abstract view above, the `sdc_abstract` goal generates two `abstract_port` constraints for each input port (`in`) and output port (out1) because the design is serving the following two types of flip-flops:

- Flip-flops driven by the `clk1` clock. These flip-flops are `f1`, `f2`, and `f3`.
- Flip-flops driven by the `clk2` clock. These flip-flops are `f4`, `f5`, and `f6`.

In addition, `sdc_abstract` goal generates two `abstract_port` constraints for the `clk1` and `clk2` port. The `abstract_port` for `clk1` is generated with the `-path_logic` option as `buf`, represents buffer, type because there is no path inversion between the source object and destination clock pin, f1/CP. RTL_FD is the name of master module inside instance `f1` and `-inst_pin` is representing the pin of the master module, `RTL_FD`, which is connected with block ports, such as `in` and

clk1.

Validating Block Assumptions in SpyGlass Constraints

To validate assumptions on an abstract view while using the SpyGlass Constraints solution, run the `sdc_abstract_validate` goal.

The following figure shows the process of validation in SpyGlass Constraints solution:

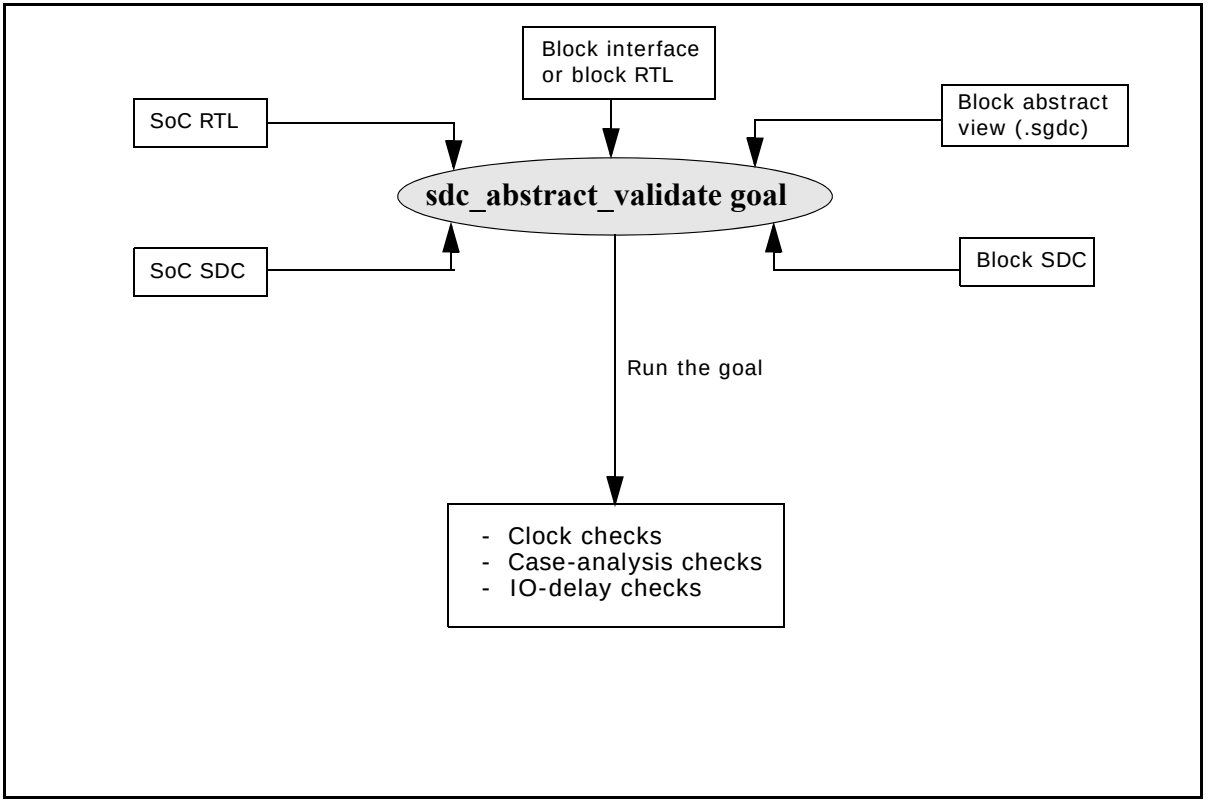


FIGURE 5.

During the goal run, constraints-specific validation is performed pertaining to clocks, `set_case_analysis`, and IO delay. These checks are

described below:

- **Clock Checks**
  - Clock defined at the top-level reaching to the block boundary and no clock is present in block.sdc. Refer to the Example below for more information on this check.
  - Clock is present in both top.sdc as well as block.sdc but clock characteristics are not same.
- **Case Analysis Checks**
  - set_case_analysis is given at top level reaching to the block boundary and no set_case_analysis is mentioned in block.sdc
  - set_case_analysis is present in both top.sdc and block.sdc but they are not same.
- **IO Delay Checks**
  - set_input_delay/set_output_delay is mentioned at a point in block.sdc but no delay value is reaching from top level or vice versa.
  - set_input_delay at block level is less than set_input_delay reaching from top level.
  - set_output_delay at block level is less than set_output_delay reaching from top level.

Example - Validating Block Assumptions in SpyGlass Constraints

The following example illustrates a clock check by validating the imported SGDC file, block1_abstract.sgdc, with the specified top SDC file, top.sdc.

```
//test.v
module top( in1, clk1, clk2, out1, out2 );
input in1;
input clk1, clk2;
output out1, out2;
wire w;
block1 b(.in(in1), .clk(clk1), .out(out1));
endmodule


module block1( in, clk, out);
input in, clk;
```

```
output out;
FD1 f1( .D(in), .CP(clk), .Q(out) );
endmodule
```

```
//test.sgdc
current_design top
sdc_data -file top.sdc -mode reference
sgdc -import block1 block1_abstract.sgdc
block -name block1
```

```
current_design block1
sdc_data -file block.sdc -mode reference
```

Here, the `sgdc -import` constraint for the current design `top` specifies to import the abstract view of `block1` in the current design `top`.

Suppose the `abstract_port` constraints specified in the block1_abstract.sgdc file are as follows:

```
// block1_abstract.sgdc
```

```
current_design block1
```

```
abstract_port -ports in -connected_inst "\block1.f1 "
-inst_pin D -inst_master RTL_FD -path_logic buf -scope const
-mode reference
```

```
abstract_port -ports out -connected_inst "\block1.f1 "
-inst_pin Q -inst_master RTL_FD -path_logic buf -scope const
-mode reference
```

```
abstract_port -ports clk -connected_inst "\block1.f1 "
-inst_pin CP -inst_master RTL_FD -path_logic buf -scope const
-mode reference
```

Clock is specified for "top" only as:

```
//top.sdc
create_clock -name clk1 -period 10 [get_ports clk1]
```

To validate, run the `sdc_abstract_validate` goal.

After the `sdc_abstract_validate` goal is run, a Warning message is reported because the `clock` constraint is defined for top only. However, you should also define this constraint for the `clk` port of the block.

Using the Abstract View in SpyGlass Constraints

The following figure shows the process of using an abstract view during SoC-level verification:

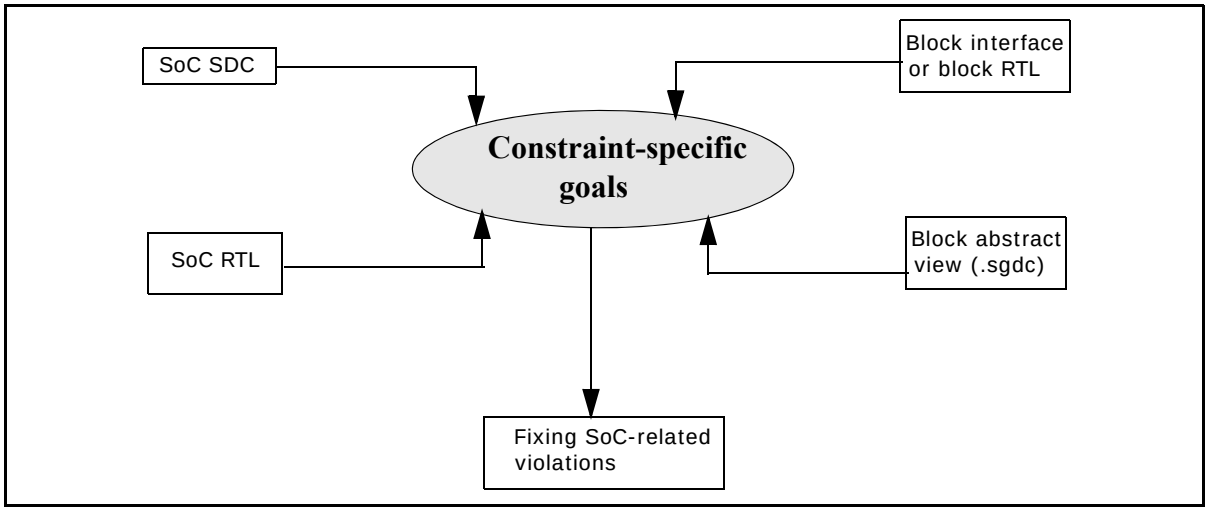


FIGURE 6.

For details on the steps during this stage, refer to the Using the *Abstract View during SoC-Level Verification* section in the *SoC User Guide*.

Example - Using the Abstract View in SpyGlass Constraints

The abstract view for the block is specified in the SGDC file through the `sgdc -import` command, as shown below.

```
//test.sgdc
current_design top
sdc_data -file top.sdc -mode reference
```

```
sgdc -import block1 block1_abstract.sgdc
block -name block1

current_design block1
sdc_data -file block.sdc -mode reference
```

Analyzing Results

This section contains the following sub-sections:

- *Debugging Reports*
- *Waiving Messages*

Debugging Reports

-tcdecompile

Setting the `-tcdecompile` parameter generates a file called TCdecompiledInfo that contains the expanded interpretation of the constraints as applied by the SpyGlass Constraints solution. This is very useful in debugging situations when you need to determine how constraints are being expanded. You can set this parameter in the Console GUI or Tcl by using the following command:

```
set_parameter tcdecompile 'yes'
```

The generated file is also useful to understand where the constraints file had a problem and if the SDC file is not read in successfully. The TCdecompiledInfo file shows exactly how each of the Tcl variables was defined or how they were interpreted. You can view this file from the following location:

spyglass_reports/constraints/TCdecompiledInfo

An excerpt of the TcdecompiledInfo file is as follows:

```
#ideal.sdc@@28@@
sg_set_ideal_network port_pin_list {A1/in1}
#ideal.sdc@@29@@
```

```
sg_set_ideal_network port_pin_list {A1/rtlc_I2/Z A1/rtlc_I2/
in1 A1/rtlc_I2/in2}
```

-tc_ignored_commands

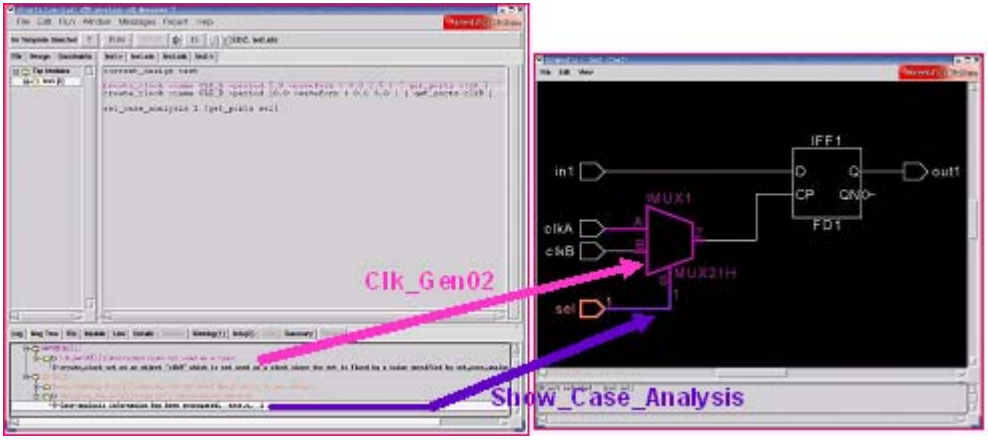
The `-tc_ignored_commands` parameter specifies the file containing the list of SDC commands to be ignored by the `SDCPARSE` rule. The file contains one command per line, which should be ignored. You should only specify the SDC commands that are currently not supported by SpyGlass so that there are no invalid command errors while parsing the SDC files. You can set this parameter in the Console GUI or Tcl by using the following command:

```
set_parameter tc_ignored_commands '<file-name>'
```

During the parsing, all other constraints that are ignored are written to a file called tc_unparsed_command in the $CWD/ <vdb-name>_reports/constraints directory. The corresponding report file named tc_unparsed_commands.rpt is created in the current working directory and can also be accessed from the **Report** Menu of the Console GUI.

Show_Case_Analysis

The `Show_Case_Analysis` rule should always be run the the Console GUI. This rule shows, as a schematic, how `set_case_analysis` propagates in the design. This is very useful when several case analysis settings result in blocking certain timing paths.



NOTE: *This rule has no value in batch mode as it shows only schematic.*

Waiving Messages

Waivers provide a means of reducing the number of violations being reported. Use them:

- If you are aware of an intentional violation of a specific check in the design/constraints.
- A specific module/file is already known to be clean and you do not want to look into anything inside it.
- A specific module/file is used for the sake of completeness of the design, for more accurate analysis, but you are not interested in analyzing the results for that module/file.

Use waivers in two ways, either during preprocessing or during post processing:

- **Applying waivers during preprocessing**: When you do not want to view the constraints issues reported in a block that you do not want to analyze, apply a waiver on the block before analysis.
- **Applying waivers during post-processing**: As you analyze the reported violation and you perform proper analysis on it, apply a waiver on it.

You can ‘waive’ rules/messages during analysis:

- In a file or design unit, by using waivers

```
waive –file src/top.v –rule Clk_Gen01
```

- Waive an instance of a message by using waivers

```
waive –file src/top.v -msg “Clock ‘clk’ doesn’t have a clock constraint”
```

- Waiving a group of messages through regular expression (‘regexp’)

```
waive -du "top" -regexp -msg ".* is not driven by a register"
```

Waivers are useful, when there are methodology-based violations in the SDC that may not be applicable. As a practice, we do not recommend the use of waivers.

Conclusion

As chip complexity increases, the issues related to constraints become critical to success because the issues can increase the risk associated with silicon respin risk and poorer chip quality in terms of area, power, and timing. The iterations in the implementation are already bad enough with the design issues, throwing an additional curve ball with constraints issues makes iterations worse. Moreover, resolving the constraints issues is a time-consuming task.

The SpyGlass Constraints solution is a part of the SpyGlass family that works at the RTL and netlist stage that:

- Checks the constraints for consistency and completeness against design, at the block level, chip level, and in the hierarchical context
- Provides debugging environment to quickly pinpoint root cause of the issues
- Creates constraint templates
- Verifies timing exceptions
- Creates timing critical exceptions for quick timing closure from STA report

Having a solution is the first step, without a proper methodology that suits the customer design flow, it is not effective. You do not know which rules to apply at what stage. Too many rules applied to a stage leads to too many violation messages; only a few of which are really critical. This creates a barrier in adoption.

In this document we have laid out a recommended step-by-step methodology that applies to generic design flow. We have created goals for each of the steps.

Appendix A: SpyGlass Constraints Design Data Checklist

TABLE 1 Library

	Information	Required/ Optional	Reason for Data	Customer Contact Profile
1.1	.lib for standard cells, I/O pads, IP's and memories	Must have	To identify valid paths in a cell, so checks like case analysis propagation and associated clocks can be performed	One of the following: • Library Group • Whoever runs DC or PT • BE designer • CAD (in some cases)

TABLE 2 Design

	Information	Required / Optional	Reason for Data	Customer Contact Profile
2.1	RTL or Netlist (Verilog or VHDL or mixed)	Must have	Read Design information. For RTL, it is required for internal synthesis	For RTL: RTL designer Whoever runs DC For Netlist: Whoever runs PT Physical designers

TABLE 3 SpyGlass Constraints

	Information	Required / Optional	Reason for Data	Customer Contact Profile
3.1	SDC with Clock definitions, Input and Output constraints, set_case_analysis, Timing Exceptions	Must have	For constraint validation	Front-end designer or whoever runs DC
3.2	SGDC Clocks list and clock domains list (can be created by SpyGlass CDC solution)	Must have if you want to create constraint	Needed for generation of SDC template.	If customer has SpyGlass, an AE can generate. Front-end Designer's help would be needed to clean up/sanitize the list of clocks generated by SpyGlass
3.3	Additional information *Multi-mode SDC, block or top level SDC, RTL or Prelayout or Layout SDC*	Optional but recommended	For selecting the right set of rules and to show value of Validation.	Design Manager/ Architect

Appendix B: Example Project File

The following code is an example of a project file. Read the comments in the code to understand the code.

```
##Data Import Section

read_file -type verilog top.v
read_file -type sglib lib.sglib

##Common Options Section

set_option language_mode mixed
set_option projectwdir .
set_option projectcwd .
set_option active_methodology $SPYGLASS_HOME/Methodology
set_option top top

##Goal Setup Section

current_methodology $SPYGLASS_HOME/Methodology

current_goal Constraints/rtl/hierarchical_check -top top
read_file -type sgdc Project-1/top/Constraints/rtl/
hierarchical_check/constraints.sgdc
```