The Formal Verification of Design Constraints
by Ajay Daga, CEO, FishTail Design Automation Inc.

There are two approaches to the verification of design constraints: formal verification and structural analysis. Structural analysis refers to the type of analysis performed by a static timing tool where timing paths either exist or not based on constant settings and constant propagation. Formal verification, on the other hand, establishes the condition under which a timing path exists based on the propagation requirements for the path. These path propagation requirements are then used to prove or disprove constraint properties based on a formal analysis of the design. Structural analysis is fast because it is simple. Formal verification, however, is more complete and less noisy. Formal verification allows engineers to guarantee that their design is safe from silicon issues that result from an incorrect constraint specification. Structural analysis cannot make this claim because it cannot independently establish the correctness of a variety of design constraints.

This document illustrates aspects of constraint analysis that are better handled by formal verification. The issues belong to the following categories:

1) Clock propagation
2) Generated clocks
3) Logically exclusive clock groups
4) Physically exclusive clock groups
5) Case analysis
6) False paths
7) Multi-cycle paths

CLOCK PROPAGATION VERIFICATION

For the design shown in Figure 1, assume the following constraints are in place:

create_clock –period 5.0 CLKA
create_clock –period 3.0 TCLK
create_clock –period 10.0 CLKB

Structural constraint analysis simply establishes whether a clock propagates to a pin or not. Case analysis and disable timing commands are the only reasons for the propagation of a clock along a combinational path to stop. Formal constraint verification, on the other hand, establishes not only the pins that a clock propagates to but the condition under which a clock propagates to that pin.

For example, in Figure 1, formal constraint verification establishes that for the clock TCLK to propagate to U1/Y the requirement is that TEST must be 1 (note that there is no case analysis specified on TEST). As the requirement for TCLK to propagate to U3/Y is that TEST must be 0 it is impossible for TCLK to propagate through U1/Y and then also through U3/Y. As a result, the propagation of TCLK will halt at U3/A. Formal constraint verification flags this as an issue - the impact of which is that the flop FF2 will not be clocked by TCLK. This issue could have been the result of a design bug where instead of an or-gate driving the select line of the clock mux U3, a nor-gate was intended. Structural analysis will completely miss this issue and instead indicate that flop FF2 is clocked by TCLK. This provides a false assurance to a design engineer and misses an important bug in the clock-generation logic – a bug that could cause a silicon issue.

VERIFICATION OF GENERATED CLOCKS

Generated clock definitions specify a periodic waveform on a pin. The complexities associated with the different options used to create a generated clock coupled with design complexities result in the risk that an engineer specifies an incorrect waveform on a generated clock. Such a mistake is serious because it impacts the stati
Structural analysis tools check that an engineer has correctly captured the edges of the master clock at which a generated clock transitions. For the generated clock specified in Figure 2, structural analysis establishes that it is indeed correct that the generated clock has been specified to transition at rising edges of the master clock. Structural analysis does not, however, verify that the low-pulse, high-pulse and period of the generated clock is correctly specified and so does not issue any error message for the situation shown in Figure 2. There is, however, an issue with the specified generated clock. The engineer has specified a divide-by 3 waveform relative to the master clock when, in fact, the logic of the design results in a divide-by 4 generated clock. A formal logical analysis of the design flags this issue and provides a much more complete check regarding the correctness of the waveform specified on a generated clock.

**LOGICALLY EXCLUSIVE CLOCK GROUP VERIFICATION**

For the design shown in Figure 3, assume that the following constraints have been defined:

```plaintext
create_clock –period 5.0 clk
create_generated_clock –source clk –divide_by 2 genclk
set_false_path –from clk –to genclk
```

Structural analysis would flag the false path from `clk` to `genclk` as incorrect because it is specified between two clocks that have the same master clock. Formal verification, on the other hand, would provide a more interesting result. Formal verification would indicate that the path between FF3 and FF4 is incorrectly constrained because the propagation requirement for `clk` to reach the clock pins on these flops is “!sel” and the propagation requirement for `genclk` to reach the same clock pins is “sel”. As the propagation requirements are exclusive to each other, the two clocks are logically exclusive in the way they propagate to flops FF3 and FF4. Formal verification would indicate that the path between FF1 and FF2 is incorrectly constrained because there
is no propagation requirement for either clk or genclk to reach the clock pins on these flops. For this reason, clocks clk and genclk are not exclusive in the way they propagate to FF1 and FF2.

Formal verification provides a more nuanced assessment of clock crossing issues. Instead of complaining about a clock-to-clock exception simply based on the fact that two clocks share the same master clock, formal verification analyzes the design to establish if the clocks are logically exclusive to each other and only complains if they are not. As a result, a designer only needs to review the real clock crossing issues on a design.

**PHYSICALLY EXCLUSIVE CLOCK GROUP VERIFICATION**

For the design shown in Figure 4, assume that the following constraints have been defined:

```verilog
create_clock -period 5.0 clk
create_generated_clock FF1/Q
    -source clk -divide_by 2 -name genclk1
create_generated_clock FF2/Q
    -source clk -divide_by 2 -name genclk2
set_clock_group -physically_exclusive
    -group genclk1 -group genclk2
```

Structural analysis would flag the physically exclusive clock group specification between genclk1 and genclk2 as incorrect because the two clocks are not defined on the same pin. Formal verification, on the other hand, would establish that the propagation requirement for the master clock clk to genclk2 is “!sel”. As a result, the condition under which generated clock genclk1 exists on the design is “sel” and the condition under which genclk2 exists in the design is “!sel”. As these conditions are exclusive the two generated clocks are physically exclusive – when one exists, the other does not.

Again, formal verification goes a step further than structural analysis and performs a more sophisticated analysis of the design. The benefit of this is less noise in the warnings reported by formal verification and this is key to maintaining the attention span of a seasoned designer.

**CASE ANALYSIS VERIFICATION**

For the design shown in Figure 5 assume that the following constraints are in place:

```verilog
set_case_analysis 0 U1/S
set_case_analysis 0 U2/S
```

A structural analysis of the design will flag no issue with these constraints because there is no conflict with the specified constant values when they are propagated forward in the design.

Formal constraint verification will, on the other hand, establish that the case analysis of 0 on U1/S requires flops FF1 and FF2 to both be high. The case analysis of 0 on U2/S requires flops FF1 and FF2 to both be low. These requirements conflict and so the specified case analysis is cumulatively illegal. On complex designs it is easy to specify case analysis values that are in conflict with each other. These values will be honored without complaint by a
static timing or implementation tool and could easily result in a silicon issue because timing signoff will be performed with the design placed in a bogus state – the actual analysis that is meant to take place will never be performed.

FALSE PATH VERIFICATION

For the design shown in Figure 6 assume that the following constraints are in place:

- create_clock –period 5.0 CLKA
- create_clock –period 10.0 CLKB
- set_false_path –from FF1/CP –to CLKB
- set_false_path –from FF2/CP –to CLKA

Structural analysis will not help establish whether the specified false paths are correct or incorrect. At best structural analysis will tell a designer that the specified paths do apply to real paths on a design. Whether they are meant to apply or not, is not something structural analysis will help with. Formal verification on the other hand will establish that the propagation requirement for the path from FF1 to FF3 is “!sel” and the propagation requirement for FF2 to FF3 is “sel”. Also, the propagation requirement for clock CLKA to FF3 is “!sel” and the propagation requirement for clock CLKB to FF3 is “sel”. As a result, since the propagation requirement of “!sel” from FF1 to FF3 conflicts with the propagation requirement of “sel” for CLKB to propagate to FF3 the false-path definition from FF1 to CLKB is correct. By a similar formal analysis the false-path definition from FF2 to CLKA is also correct.

The value of formally verifying false paths is to only leave for designer review the fpars that cannot formally be proven based on the design description, but require additional architectural considerations to be taken into account. To the extent this architectural information such as static registers or restrictions on the way configuration registers are programmed can be communicated using SVA assertions, formal verification is able to leverage these assertions by treating them as assumptions and prove more of the false path definitions in a constraint file. The inability to establish the correctness of any false-path definitions in a constraint file results in a gap between what structural analysis is able to do and what is required for constraint signoff.

MULTI-CYCLE PATH VERIFICATION

For the design shown in Figure 7, assume the following constraints are in place:

- create_clock –period 5.0 CLK
- set_multicycle_path –from FF1/CP –to FF2/D

Structural analysis is in no position to verify whether the specified multi-cycle path is correct or not. Formal verification, on the other hand establishes that for the FF1/Q pin to transition in the current clock cycle valid must be high in the previous clock cycle. Also, for the clock clk to propagate to FF2, valid must be high. The signal valid has a periodic relationship and toggles from high to low every clock cycle. For this reason, if valid is high in the current clock cycle, causing FF1/Q to transition in the next cycle, then valid is also low in the next cycle thereby disabling
the clock clk from propagating to FF2. This ensures multi-cycle behavior and the specified multi-cycle path from FF1 to FF2 is correct. MCP verification requires a sequential formal analysis of the design, working back several clock cycles until a multi-cycle property is proven. Again, the use of assertions to specify architectural restrictions (for example, the way configuration registers are programmed) is key to enabling formal verification when MCP behavior is not supported by just the synthesizable design description.

SUMMARY
Formal verification is key to obtaining a thorough and independent assessment of design constraints. Formal verification results play the role of devils advocate, questioning the rationale for constraints and driving an engineering review discussion. These results are credible because there is less noise and more nuanced issues are presented to designers. Structural analysis is far too limited in its scope leaving a whole range of constraint issues unverified. While fast, it relies on engineers to separate the wheat from the chaff. While the examples in this document are simple, the value of formal verification comes through on large complex designs where the correctness of constraints is not as obvious. Clean formal verification results eliminate the risk of silicon failure from bad constraints.