Constraint Generation

Constraint generation refers to the creation of SDC constraints (clocks, generated clocks, case analysis, input/output delays, clock groups, clock senses, false and multi-cycle paths) given just the description of a design, and, optionally, any constraints already in place for the design. The expectation is that the constraints that are generated are correct, consistent and optimal with regard to their subsequent use in chip-implementation or static timing analysis (STA). Automated constraint generation significantly shortens the time taken to create the SDC constraints for a design. By generating a more complete set of constraints that prevent unnecessary optimization of the design, constraint generation can also reduce implementation runtime and provide modest improvements to the area/power/speed of the design.

Figure 1: Constraint generation flow.

The FishTail constraint generation flow, shown in Figure 1, takes as input the RTL or netlist description of a design along with any constraints that are already in place. Engineers can ask the tool to generate specific types of constraints. The generated constraints are immediately verified using Confirm and can be visualized/edited in Excel. FishTail constraint generation has the following applications:

1) Generation of Clock and IO Constraints
2) Generation of Modes and Case Analyses
3) Generation of Clock Groups
4) Generation of Clock Senses
5) Generation of Timing Exceptions
6) Generation of Liberty Models
Focus generates clock and input/output delay constraints for a design by first identifying the clock pins on the design and traversing back from these clock pins to identify the pins/ports on which clocks and generated clocks should be created. The key challenge is to distinguish between clock generation logic and clock gating/muxing logic so that clocks are only created where they should be and so that all registers on a design are clocked. Engineers need to specify the period for the clocks and the divide-by factor for the generated clocks extracted by the tool. They can also optionally modify the names of the clocks created by the tool. All of this is done by either editing the SDC or XLS file generated by Focus. Once the clock definitions are in place the tool creates input-output delays for ports based on the clocks that propagate to the interface registers that are driven by or drive these ports. Engineers can ask for input/output delays to be created relative to virtual clocks. The actual delay value budgeted for a port is specified by a user by editing either the XLS or SDC file generated by Focus.

For the design shown in Figure 2 Focus generates the following constraints:

```plaintext
create_clock clka -period 10 -name clka
create_clock clkb -period 10 -name clkb
create_generated_clock div1_clk -name clkb_1 -master clkb -divide_by 2
create_generated_clock div2_clk -name clka_1 -master clka -divide_by 2
create_generated_clock div2_clk -name clkb_1_1 -master clkb_1 -divide_by 2 -add
set_input_delay in 5 -clock clka_1
set_input_delay in 5 -add -clock clkb_1_1
```

Note that the tool establishes the generated clocks on the design, the master clocks associated with these generated clocks and avoids the creation of clock constraints on clock-gate/clock-mux enable logic.
On an IP that designers are unfamiliar with it is important for engineers to visualize the clock structure of a design, to understand the motivation for the clocks created by the tool. This is done by using a clock-browser that is part of the Focus GUI. Figure 3 illustrates the visualization provided by the clock-browser. It shows how a clock propagates to generated clocks, through clock muxes/clock gates and finally to the registers clocked by the clock.

**Figure 3: Clock browser.**

### Generation of Modes and Case Analyses

Given a design and its clock constraints, Focus can extract the modes on the design and the case analyses that put the design into the extracted modes. Focus does this by analyzing the clock-muxing logic and the requirements for clocks to propagate through this logic. For the design show in Figure 4 the tool extracts the following modes:

**Mode 1:** (TCLK propagates)

```
set_case_analysis 1 TEST
```

**Mode 2:** (CLKB propagates)

```
set_case_analysis 0 TEST
set_case_analysis 1 CLKB_EN
```

**Mode 3:** (CLKA propagates)

```
set_case_analysis 0 TEST
set_case_analysis 0 CLKB_EN
```
Generation of Clock Groups

Given a design and its clocks Focus can be used to generate missing clock groups. If two master clocks are specified as asynchronous, but the generated clocks derived from these master clocks have not been specified to be asynchronous, then Focus will generate asynchronous clock groups for the generated clocks. For example, consider the situation where the following constraints are provided as input to the tool:

create_clock clk -period 10
create_clock tclk -period 6.4
create_generated_clock -divide_by 2 -master clk -source clk {div2_clk}
set_clock_group -async -group {clk} -group {tclk}

If Focus finds a path between a register clocked by div2_clk and a register clocked by tclk it will generate the following clock group:

set_clock_group -async -group {div2_clk} -group {tclk}

Focus also generates missing logically and physically exclusive clock groups based on an analysis of the clock propagation conditions on the design. For example, for the design in Figure 4, Focus will generate the following clock group:

set_clock_group -logically-exclusive -group {CLKA CLKB} -group {TCLK}

Focus generates physically exclusive clock groups when multiple clocks are defined on the same pin, or when generated clocks defined on different pins of a design cannot simultaneously exist. For example, for the design in Figure 5, Focus will generate the following clock group:

set_clock_group -physically-exclusive -group genclk1 -group genclk2

Figure 5: Physically exclusive clock group generation.
The propagation requirement for the master clock $clk$ to $genclk_1$ is “$sel$”. Similarly, the propagation requirement for the master clock $clk$ to $genclk_2$ is “$!sel$”. As a result, the condition under which generated clock $genclk_1$ exists on the design is “$sel$” and the condition under which $genclk_2$ 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.

**Generation of Clock Senses**

When multiple clocks propagate to the registers on a design, then the complexity of STA increases because timing needs to be checked relative to all the clocks. Focus analyzes the timing requirement that needs to be satisfied at each register on the design after accounting for all the clocks, clock groups and timing exceptions that apply to the register. Based on this analysis Focus establishes the critical subset of clocks that need to propagate to the register. These critical clocks are the ones that play a role in establishing the timing slack at the register. The non-critical clocks are frivolous, play no role in STA, and only increase STA complexity. Focus generates `set_clock_sense - stop_propagation` constraints to block the propagation of non-critical clocks. The impact of these constraints is a reduction in the clocks/register on the design. This, in turn, improves STA and implementation runtime.

![Reduction of clocks/register.](image)

For the design in Figure 6, registers $f_1$ and $f_2$ receive 4 clocks, registers $f_3$ and $f_4$ receive 2 clocks, and registers $f_5$ and $f_6$ receive 3 clocks. Focus generates the following clock senses for this design:

```
set_clock_sense -stop_propagation m1/B -clock 10mclk
set_clock_sense -stop_propagation m2/B -clock 50mclk
set_clock_sense -stop_propagation i1/A -clock 30mclk
```

$30mclk$ establishes a tighter requirement on the downstream flops when compared to the $10mclk$, and so the propagation of $10mclk$ can be stopped at $m1$. Similarly, $100mclk$
establishes a tighter timing requirement when compared to 50mclk and so the propagation of 50mclk can be stopped at m2. Finally, the propagation of 30mclk can be stopped at i1. With the clock senses generated by Focus for the design in Figure 6, each register receives one clock with no change in the STA results.

Application of this flow on a customer design resulted in the average clocks/register dropping from 12 to 2.8. This resulted in a 3.3x improvement in P&R runtime which dropped from 154 hours to 47 hours with no change to the QoR for the design.

**Generation of Timing Exceptions**

Given a design and its clock constraints Focus generates false and multi-cycle paths for the design. The justification for Focus generated timing exceptions is solely based on the logic of the design.

For the design shown in Figure 7 Focus will generate the following exceptions:

```plaintext
set_false_path -from FF1/CP -to [get_clocks CLKB]
set_false_path -from FF2/CP -to [get_clocks CLKA]
```

Focus establishes 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”. Since the propagation requirement of “!sel” from FF1 to FF3 conflicts with the propagation requirement of “sel” for CLKB to propagate to FF3 the path from FF1 to CLKB is false. Similarly, there is a false path from FF2 to CLKA.

For the design shown in Figure 8 Focus will generate the following MCPs:

```plaintext
set_multicycle_path -from FF1/CP -to FF2/D -setup 2
set_multicycle_path -from FF1/CP -to FF2/D -hold 1
```
Focus establishes that for the FF1/Q pin to transition valid must be high in the previous clock cycle. Also, for 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 preventing clk from propagating to FF2. This ensures multi-cycle behavior.

Focus generated exceptions are verified by Confirm. Engineers can also ask Confirm to generate assertions for Focus generated exceptions and use their functional simulation environment to corroborate the correctness of Focus generated exceptions.

**Generation of Liberty Models**

Given the RTL and SDC for a design, Focus can generate a Liberty model that accurately captures the I/O timing arcs on the design (setup/hold, sequential delay, combinational delay). The timing senses for the arcs are accurately captured in the Liberty model. The delay values for timing arcs are based on the clock periods and the input/output delays in the SDC file. The generated Liberty model can be used while implementation of the block progresses and accurately reflects the I/O timing that the block implementation is attempting to meet.