# STA by examples - using SDC

- Mitu Raj, chip@chipmunklogic.com, Dec-2024

## Prologue

Along with the RTL, I have always made it a habit to write the timing constraints for the designs whenever I compile and synthesise the RTL on FPGAs. It enables timing analysis (STA) of the design and helps to understand how good the design is timing-performance-wise.

This document compiles a set of design examples with commonly encountered clocking schemes, and discusses how to write the SDC timing constraints for the clocks. Timing exceptions are also discussed with examples. Tool specific options are also discussed in some examples. The document covers only the fundamental aspects of writing timing constraints using SDC. It doesn't go deep into the syntax, intricacies, and variations of SDC commands. For complete details on each SDC command used in the examples, please refer to the SDC1.7 specs in [1].

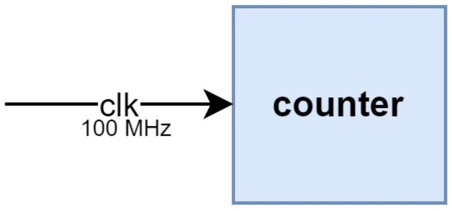
## Clock constraints

This section discusses how to create clock constraints which characterize clocks in a design.

### 1. Creating clocks

#### DESIGN PROBLEM 1

A counter runs at a clock frequency = 100 MHz. The clock comes from an input port.

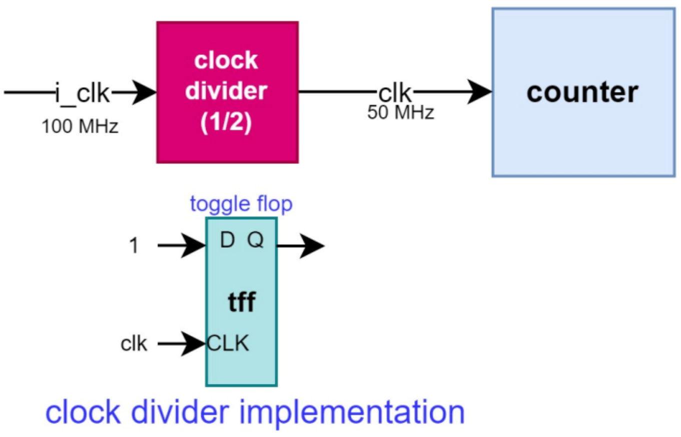


#### SDC

```
# Create clock at the input port clk

# Case 1: The clock has 0° phase shift, 50% duty cycle /``\__/
create_clock -name clk -period 10.00 -waveform {0 5} [get_ports clk]

# Case 2: The clock has a different duty cycle = 60% /```\_/
create_clock -name clk -period 10.00 -waveform {0 6} [get_ports clk]

// Case 3: The clock has 180° phase shift \__/``\
create_clock -name clk -period 10.00 -waveform {5 10} [get_ports clk]
```

#### DESIGN PROBLEM 2

A counter runs at a clock frequency = 100 MHz. The clock is driven by a divide-by-2 clock divider circuit implemented in the RTL. The master clock comes from an input port.



**SDC**

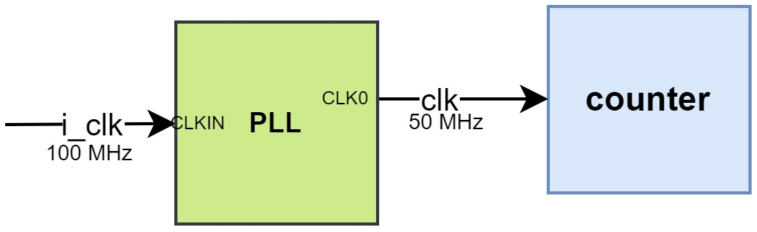
```
# Create base clock at the input port
create_clock -name clkin -period 10.00 -waveform {0 5} [get_ports i_clk]

# Create divided clock from the base clock
create_generated_clock -name clk -divide_by 2 -source [get_ports i_clk] [get_pins {tff/Q}]

# OR
create_generated_clock -name clk -divide_by 2 -source [get_pins {tff/CLK}] [get_pins
{tff/Q}]
```

**Notes**

- Generated clocks are treated by the STA tool as synchronous to master/base clocks and automatically infers source latency = source latency of base clock + latency between the base and generated clock definition points.
- If you create the divided clock as master clock instead of using *create_generated_clock*, the disadvantage is that it will be treated as asynchronous clock. Hence, latencies should be explicitly defined. So, this is recommended to be avoided.

## DESIGN PROBLEM 3

A counter runs at a clock frequency = 100 MHz. The clock is driven by the divide-by-2 output clock from a PLL. The PLL input clock comes from an input port.



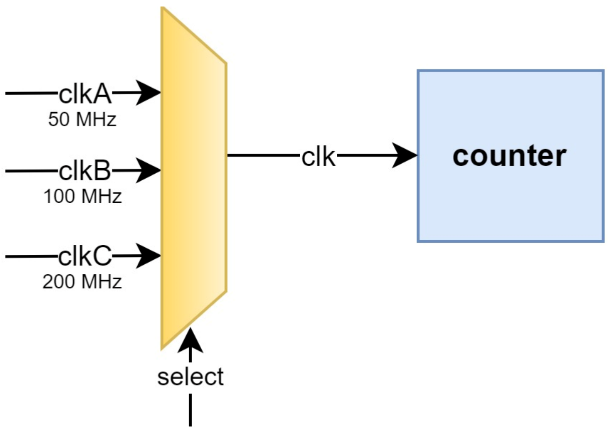
**SDC**

```
# Create base clock at the input port
create_clock -name i_clk -period 10.00 -waveform {0 5} [get_ports i_clk]

# Create PLL output clock from the base clock
```

```
create_generated_clock -name clk -divide_by 2 -source [get_pins {pll/CLKIN}] [get_pins
{pll/CLK0}]

# Create base clock and PLL output clocks automatically
# Applicable only to Quartus for Altera FPGAs only…
derive_pll_clocks -create_base_clocks
```

## DESIGN PROBLEM 4

A counter is clocked by a clock mux which selects between clocks = 50/100/200 MHz, which are driven from input ports. The clock mux is internal to the design and the clocks don't interact outside the mux.



## SDC

```
# Create all clocks at the input ports
create_clock -name clkA -period 20.00 -waveform {0 10} [get_ports clkA]
create_clock -name clkB -period 10.00 -waveform {0 5} [get_ports clkB]
create_clock -name clkC -period 5.00 -waveform {0 2.5} [get_ports clkC]

# Define logically exclusive clocking
set_clock_groups -logically_exclusive -group {clkA} -group {clkB} -group {clkC}

# OR
# Define false paths between clock domains (not recommended, deprecated practice…)
set_false_path -from clkA -to clkB
set_false_path -from clkA -to clkC
set_false_path -from clkB -to clkA
set_false_path -from clkB -to clkC
set_false_path -from clkC -to clkA
set_false_path -from clkC -to clkB
```

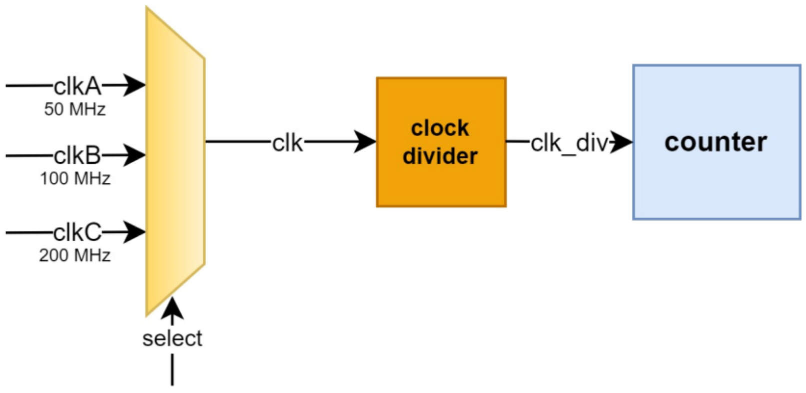
## Notes

- The clocks are "logically exclusive" because even though all the clocks exist together inside the design, only one of the clocks can drive the counter at a time.
- If the select signal is dynamic, the mux requires clock gating checks. Clock gating checks are automatically inferred by the STA tool for simple cells like clock mux.

- If the select signal is static, it can be constrained by set_case_analysis <0/1/2> <select pin>. This performs a constrained timing analysis. If set_case_analysis is not used, all the three clocks are taken into account for timing analysis.
- No need to create generated clock at the mux output as the input clocks should be auto-propagated for simple cells like clock mux.

## DESIGN PROBLEM 5

A counter is clocked by a divide-by-4 clock divider circuit. The master clock of the clock divider is driven by a clock mux which selects between clocks = 50/100/200 MHz, coming from input ports. The clock mux is internal to the design and they don't interact outside the mux.



## SDC

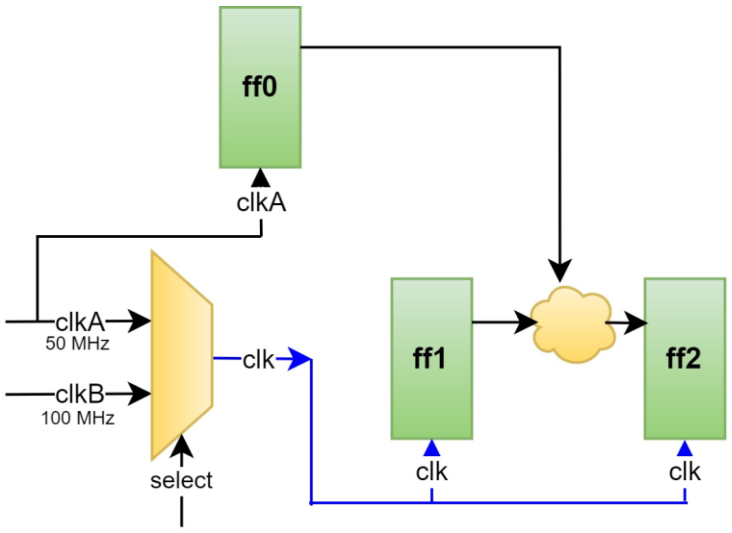
```
# Create all base clocks at the input ports
create_clock -name clkA -period 20.00 -waveform {0 10} [get_ports clkA]
create_clock -name clkB -period 10.00 -waveform {0 5} [get_ports clkB]
create_clock -name clkC -period 5.00 -waveform {0 2.5} [get_ports clkC]

# Create generated clock at the divider output from each base clock
create_generated_clock -name clkA_by_4 -divide_by 4 -source [get_ports clkA] [get_pins
{clkdvd/ff1/Q}]
create_generated_clock -name clkB_by_4 -divide_by 4 -source [get_ports clkB] [get_pins
{clkdvd/ff1/Q}] -add
create_generated_clock -name clkC_by_4 -divide_by 4 -source [get_ports clkC] [get_pins
{clkdvd/ff1/Q}] -add

# Define logically exclusive clocking
set_clock_groups -logically_exclusive -group {clkA clkAgen} -group {clkB clkBgen} -group
{clkC clkCgen}
```

## Notes

- Pay attention to the grouping of clocks in the exclusion constraint. Synchronous clocks (base and generated) are grouped together.

## DESIGN PROBLEM 6

A counter is clocked by a clock mux which selects between clocks = 50/100 MHz, which are driven from input ports. The clock mux is internal to the design and they interact outside the mux as shown in the diagram.

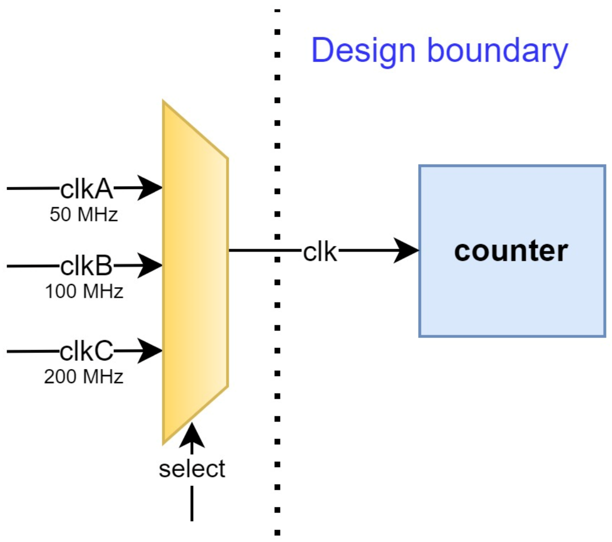


**SDC**

```
# Create all base clocks at the input ports
create_clock -name clkA -period 20.00 -waveform {0 10} [get_ports clkA]
create_clock -name clkB -period 10.00 -waveform {0 5} [get_ports clkB]

# Create generated clock at the mux output for each base clock
create_generated_clock -name clkAmux -divide_by 1 -source [get_ports clkA] [get_pins
{clkmux/o}]
create_generated_clock -name clkBmux -divide_by 1 -source [get_ports clkB] [get_pins
{clkmux/o}] -add

# Define logically exclusive clocking only for generated clocks
set_clock_groups -logically_exclusive -group {clkAmux} -group {clkBmux}
```

**DESIGN PROBLEM 7**

A counter is clocked by a clock mux which selects between clocks = 50/100/200 MHz, which are driven from input ports. The clock mux is external to the design (say, a jumper wire on the PCB selects the clock).



## SDC

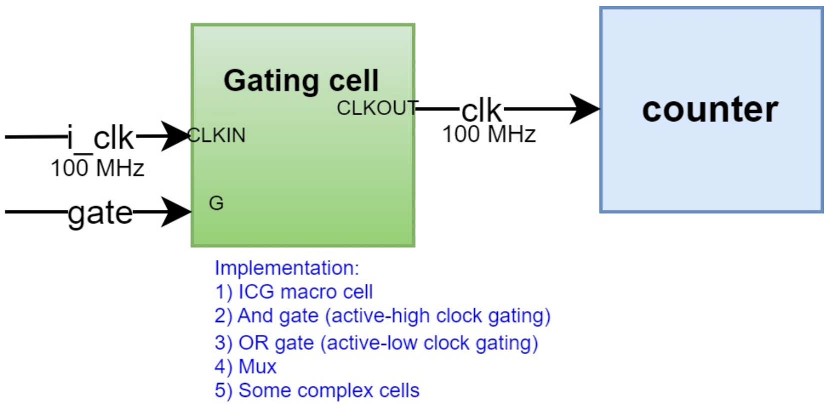
```
# Create all clocks at the master clock input port
create_clock -name clkA -period 20.00 -waveform {0 10} [get_ports clk]
create_clock -name clkB -period 10.00 -waveform {0 5} [get_ports clk] -add
create_clock -name clkC -period 5.00 -waveform {0 2.5} [get_ports clk] -add

# Define physically exclusive clocking
set_clock_groups -physically_exclusive -group {clkA} -group {clkB} -group {clkC}
```

## Notes

- The clocks are "physically exclusive" because only one clock can exist inside the design and drives the design at any given time.

## DESIGN PROBLEM 8

A counter is clocked by a gated clock = 100 MHz from a clock gating cell. The gate signal is dynamically driven in the design. The ungated master clock comes from an input port.



## SDC

```
# Create base clock at the input port
create_clock -name clkin -period 10.00 -waveform {0 10} [get_ports i_clk]

# Create gated clock at the output of the gating cell
create_generated_clock -name clk -divide_by 1 -source [get_pins igc/CLKIN] [get_pins
{igc/CLKOUT}]

# Set clock gating check on the gating cell
set_clock_gating_check -high [get_cells ANDGATE]
set_clock_gating_check -low [get_cells ORGATE]
```

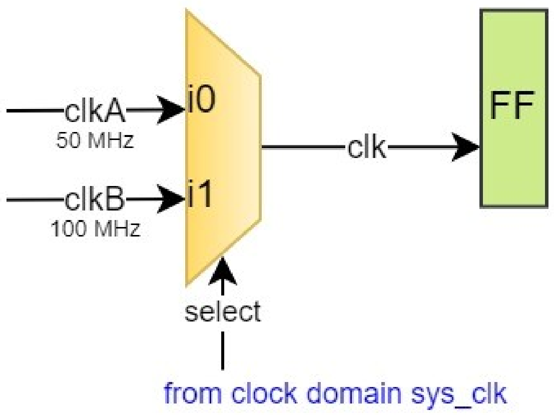
## Notes

- In case of simple gating cells like AND, OR gate, or ICG macro cells, no need to specify clock gate check as it is automatically inferred.
- In case of simple gating cells like AND, OR gate, or multiplexers, or ICG macro cells, the tool should be able to auto-propagate the clock to the output. In such cases, no need to create generated clock.
- In case of multiplexers, or latch-based gating cells, clock gating check should be explicitly specified.
- In case of complex cells where the setup/hold relation is unknown, the setup and hold margins should be explicitly mentioned for the clock gating check with appropriate flags *(-setup, -hold, -high, -low)*.

- In case of multiplexers, the clock gating check happens on select pin. One or more of the input clocks, which you know by design cannot cause glitch on switching from/to, can be disabled from gating check by *set_disable_clock_gating_check <pin>*. And enable *set_enable_clock_gating_check -high/low* on the other pins.

## DESIGN PROBLEM 9

A design is clocked by a clock mux which selects between two clocks = 50/100 MHz, which are driven from input ports. The select signal is dynamically generated from another clock domain running at 25 MHz. By design, clkB is ensured to be low while switching the select signal. Derive the clock gating checks.

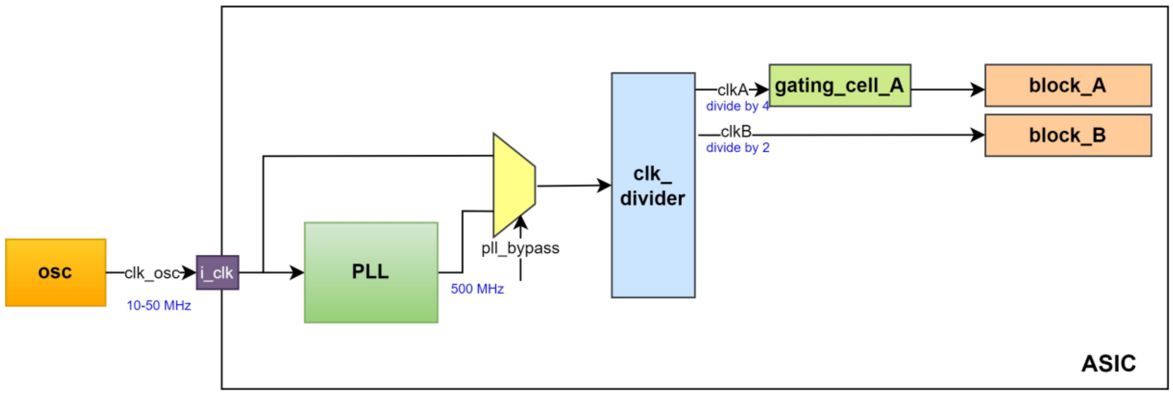


## SDC

```
# Create all base clocks at the input port
create_clock -name clkA -period 20.00 -waveform {0 10} [get_ports clkA]
create_clock -name clkB -period 10.00 -waveform {0 5} [get_ports clkB]

# Set clock gating check only on i0 pin of the mux (active-high)
set_clock_gating_check -high [get_cells mux0]
set_disable_clock_gating_check [get_pins mux0/i1]
```

## DESIGN PROBLEM 10

A design has clock distribution as shown in the diagram. Assume that the PLL settings are dynamic, and gating cells are standard ICG macro cells. Constrain all clocks of this design.



**SDC**

```
# Create oscillator (OSC) clock = 50 MHz at the input port
create_clock -name clk_osc -period 200.00 -waveform {0 100} [get_ports i_clk]

# Create PLL output clock
create_clock -name pll_clk -period 2.00 -waveform {0 1.00} [get_pins {pll/CLKOUT}]

# Define logically exclusive clocking between OSC and PLL clocks
set_clock_groups -logically_exclusive -group {clk_osc} -group {pll_clk}

# Created generated clock clkA at the divider output from each base clock
create_generated_clock -name clkA_from_osc -divide_by 4 -source [get_ports i_clk] [get_pins
{clk_divider/ff1/Q}]
create_generated_clock -name clkA_from_pll -divide_by 4 -source [get_pins {pll/CLKOUT}]
[get_pins {clk_divider/ff1/Q}] -add

# Created generated clock clkB at the divider outputs from each base clock
create_generated_clock -name clkB_from_osc -divide_by 2 -source [get_ports i_clk] [get_pins
{clk_divider/ff1/Q}]
create_generated_clock -name clkB_from_pll -divide_by 2 -source [get_pins {pll/CLKOUT}]
[get_pins {clk_divider/ff1/Q}] -add

# Create gated clock for block_A at the gating cell output from each master clock
create_generated_clock -name clkA_from_osc_gated_ -divide_by 1 -source [get_pins
{gating_cell_A/CLKIN}] -master_clock [clkA_from_osc] [get_pins {gating_cell_A/CLKOUT}]
create_generated_clock -name clkA_from_pll_gated -divide_by 1 -source [get_pins
{gating_cell_A/CLKIN}] -master_clock [clkA_from_pll] [get_pins {gating_cell_A/CLKOUT}] -add
```

## 2. Creating clock uncertainties

Clock jitter and other pessimisms can be modelled as setup and hold uncertainties.

**SDC**

```
# Intra-clock uncertainties
set_clock_uncertainty -setup 0.20 [get_clocks clk]
set_clock_uncertainty -hold 0.20 [get_clocks clk]

# Inter-clock uncertainties (across different clock domains)
set_clock_uncertainty -setup 0.20 -from [get_clocks clk1] -to [get_clocks clk2]
set_clock_uncertainty -hold 0.20 -from [get_clocks clk1] -to [get_clocks clk2]
```

## 3. Creating clock transition times/slews

At the output of PLL models or an input port, the STA tool may not be able to compute the transition times automatically, and hence leads to less accurate timing analysis. In such cases, the transition time/slew can be explicitly specified at the source of the clock.

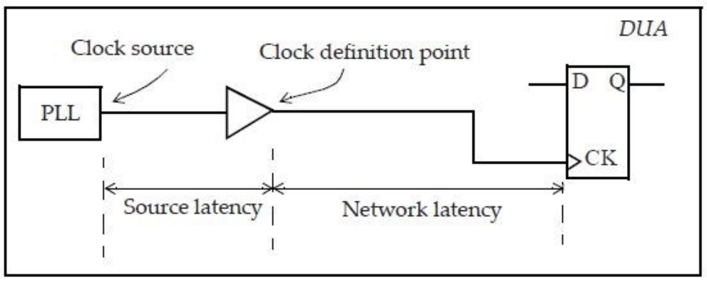
**SDC**

```
# Clock transition times (rise/fall)
set_clock_transition -rise 0.10 [get_clocks clk]
set_clock_transition -fall 0.12 [get_clocks clk]
```

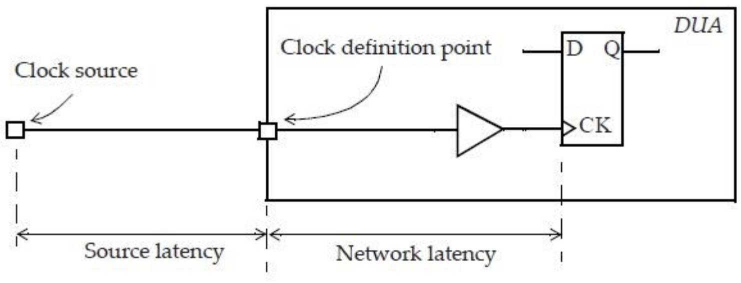
This specification applies only for ideal clocks and is disregarded once the clock trees are built, at which point, actual transition times at the clock pins are used. If a clock is defined at an input port, use the *set_input_transition* specification to specify the slew on the clock [1].

## 4. Creating clock latencies

Source and network latencies of clock can be specified. Source latency (insertion delay) is the on-chip or off-chip delay from the actual source of clock to the clock definition point. Network latency is the delay from clock definition point to a clock pin in a flip-flop or some clocked element.



(a) On-chip clock source.



(b) Off-chip clock source.

*Image source: [1]*

**SDC**

```
# Network latency
set_clock_latency 1.80 -rise [get_clocks clk]
set_clock_latency 1.90 -fall [all_clocks]
# Use set_propagated_clocks in SDC after CTS in ASICs
# Source latency
set_clock_latency 1.80 -max -source [get_clocks clk]
set_clock_latency 1.20 -min -source [get_clocks clk]
```

For ASICs, network latency specification is just an estimate used for STA before CTS and is disregarded once the clock trees are built. At this point after CTS, (actual network latency + source latency spec) is used as the net clock latency if *set_propagated_clocks* is set.

## Timing Exceptions

Timing exceptions are constraints that modify the default behavior of the STA tool when it evaluates the timing of paths in a design. Most commonly used timing exceptions are discussed in this section.

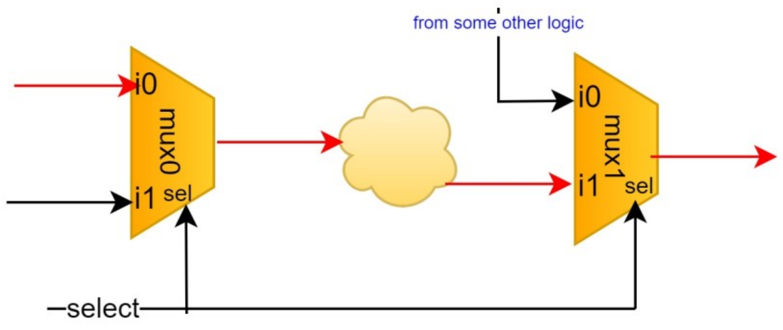
*Note: You may have already noticed the use some of these constraints in the earlier design examples.*

### 1. False Paths

False paths are the timing paths in a design that need not be timed due to several reasons. The STA tool ignores timing of the paths when they are constrained as false paths. Described below are some examples where this timing exception may be used.

**DESIGN PROBLEM 1**

A path (red colored path in the diagram) is a functionally/structurally impossible timing path in the design. This path should be exempted from timing analysis.

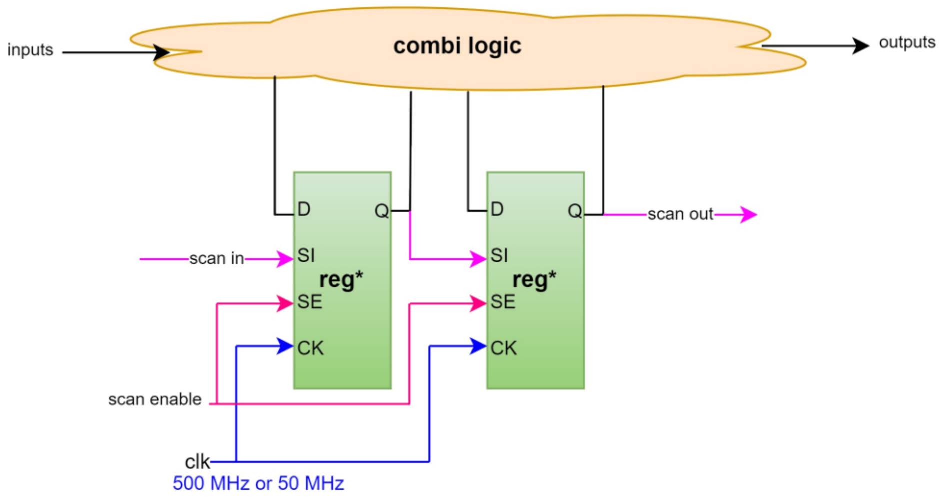


**SDC**

```
# Network latency
set_false_path -from [get_pins {mux0/i0}] -to [get_pins {mux1/o}]

# OR
set_false_path -through [get_pins {mux0/i0}] -through [get_pins {mux1/o}]
```

**DESIGN PROBLEM 2**

A design can function in two modes: functional, scan (non-functional). Functional mode uses max clock = 500 MHz, while scan mode uses clock = 50 MHz. Scan inputs of all registers are unused in functional mode, and hence not required to be timed for functional clock. Convey this requirement to the STA tool.



**SDC**

```
# Create all clocks
create_clock -name clk_func -period 2.00 [get_ports clk]
create_clock -name clk_scan -period 20.00 [get_ports clk] -add

# Set false path to scan inputs
set_false_path -from clk_func -to [get_cells {reg*/SI}]
```

**DESIGN PROBLEM 3**

A design has an asynchronous reset (active-low). The reset de-assertion is synchronous. Since the assertion is asynchronous, it can be ignored from timing. The design also has a test mode input, which is quasi-static and doesn't change during functional operation. The timing paths driven by this signal can be ignored safely. Convey these requirements to the STA tool.

**SDC**

```
# Set false path only for reset assertion
set_false_path -from [get_ports i_rstn] -fall

# Set false path for quasi-static signals
set_false_path -from [get_ports i_testmode]
```

**Notes**

- Using *set_false_path* gives complete freedom to the implementation tools to take as much delay as they want. To avoid this, designers prefer to put an upper-bound delay either through *set_multicycle_path* through a *set_max_delay*. [2]

## 2. Clock Groups

The *set_clock_groups* constraint is used to specify unrelated clock domains in a design. We have already seen some of its use cases (*physically_exclusive*, *logically_exclusive*) in previous examples.

**DESIGN PROBLEM 1**

A design has 3 clock domains. The domains clkA, clkB are synchronous, but the domain clkC is unrelated to both clkA and clkB. Hence, all the clock domain crossing paths between clkC and {clkA, clkB} can be ignored.

**SDC**

```
# Set clock groups
set_clock_groups -asynchronous -group {clkA clkB} -group {clkC}
```

**Notes**

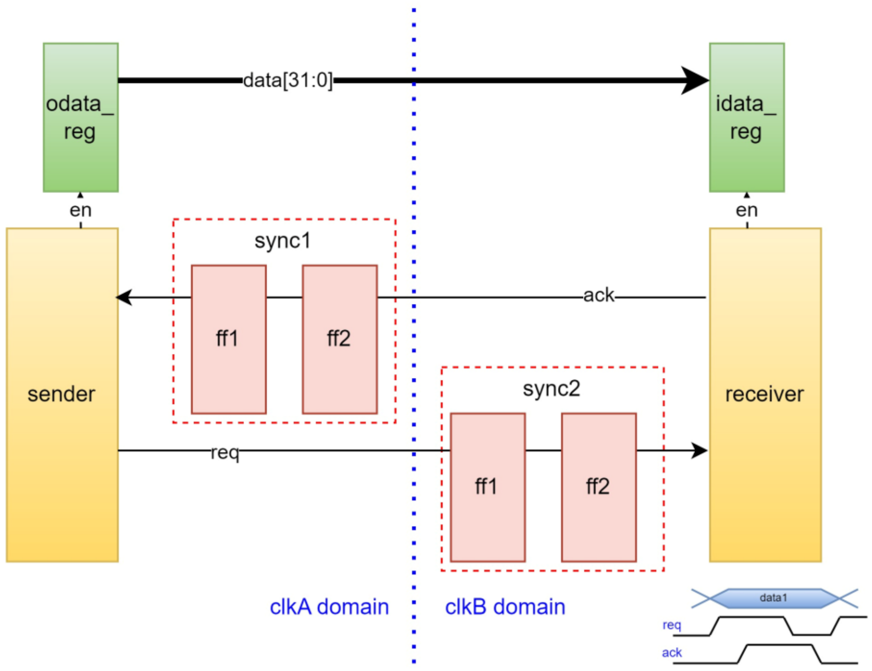
- Using *set_clock_groups -asynchronous* also gives complete freedom to the implementation tools to take as much delay as they want. To avoid this, designers prefer to put an upper-bound delay either through *set_multicycle_path* through a *set_max_delay*.

## 3. Max and Min Delays

These two constraints are used to override the setup and hold relationships. The constraint set_max_delay overrides the setup, while set_min_delay overrides the hold relationship. One popular use case of this constraint is in CDC paths, as a better alternative to false path/clock groups constraint, which have no upper-bound on timing path delays.

**DESIGN PROBLEM 1**

A design has 2 asynchronous clock domains, clkA@100 MHz), clkB@50 MHz. The domain clkA transfers 32-bit data to the domain clkB. A four-way request-ack handshaking scheme is implemented between the domains. Constrain all the CDC paths for synthesis.



**SDC**

```
# Create all clocks
create_clock -name clkA -period 10.00 [get_ports clkA]
create_clock -name clkB -period 20.00 [get_ports clkB]
```

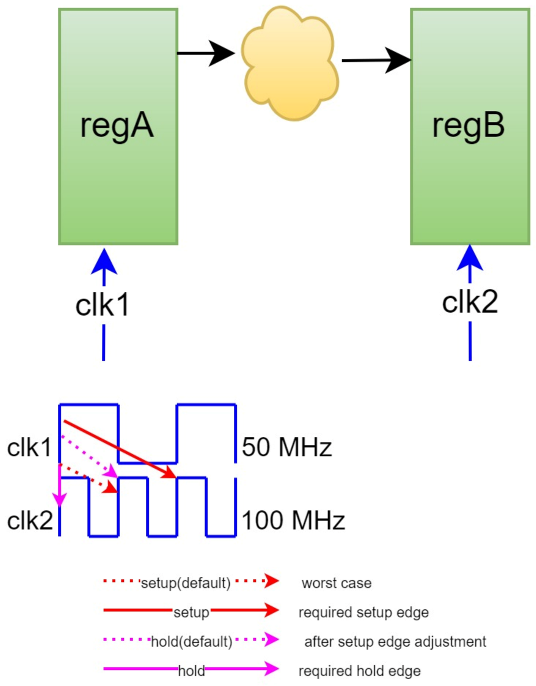
```
# Set max delay to have upper-bound delay on CDC paths…
# In this case, max delay = min(TclkA, TclkB) is a good thumb rule to follow to
# place the meta flop (ff1 in sync1/2) as close as possible…
set_max_delay -from [get_clocks clkA] -to [get_clocks clkB] 10.00
set_max_delay -from [get_clocks clkB] -to [get_clocks clkA] 10.00

# In Xilinx FPGAs: only the datapath can be constrained to a max delay…
set_max_delay -datapath_only -from [get_clocks clkA] -to [get_clocks clkB] 10.00
set_max_delay -datapath_only -from [get_clocks clkB] -to [get_clocks clkA] 10.00

# In Altera FPGAs: Set max skew to have upper-bound on skew between data bus bits.
# In this case, we can use min(TclkA, TclkB)
set_max_skew -from [get_cells odata_reg[*]] -to [get_cells idata_reg[*]] 10.00

# In Xilinx FPGAs: Set bus skew to have upper-bound on skew between data bus bits.
# In this case, we can use min(TclkA, TclkB)
set_bus_skew -from [get_cells odata_reg[*]] -to [get_cells idata_reg[*]] 10.00
```

## 4. Multi-cycle Paths (MCP)

In some cases, the timing path between flops can be allowed to take more than one cycle. Hence, the capture edge can be shifted forward by a specific number of cycles. In such cases, it is desirable to change the default setup-hold relationship assumed by the STA tool. Multi-cycle path constraints can be used to relax the timing on such paths.

### DESIGN PROBLEM 1 – SLOW TO FAST MCP (integer multiple)

In a design, data is launched at every rising edge of clk1 = 50 MHz and it is captured at rising edge of clk2 = 100 MHz every two clock cycles. These paths are by default analyzed pessimistically as single-cycle paths. Timing can be relaxed in these paths by enforcing the design-intended setup-hold relationship. Convey this requirement to the STA tool. Assume both clocks are synchronous and driven from input ports.



**SDC**

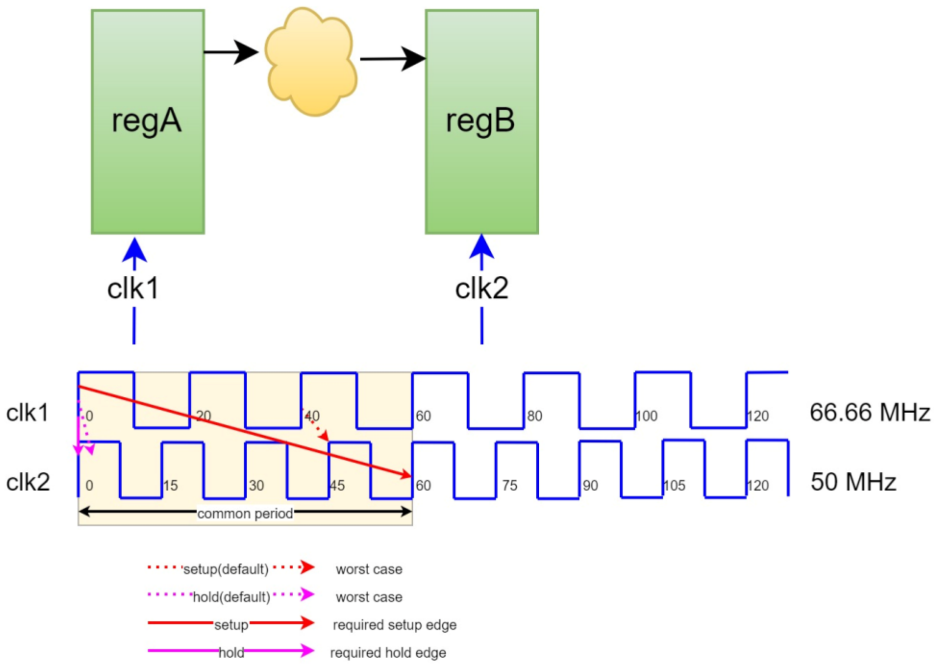
```
# Create all clocks
create_clock -name clk1 -period 20.00 -waveform {0 10} [get_ports clk1]
create_clock -name clk2 -period 10.00 -waveform {0 5} [get_ports clk2]

# Set MCP to shift the setup edge to 2nd edge of clk2, default = 1st edge
set_multicycle_path -setup 2 -from [get_clocks {clk1}] -to [get_clocks {clk2}]
-end
# Set MCP to shift the hold edge by -1 clk2 cycle
set_multicycle_path -hold 1 -from [get_clocks {clk1}] -to [get_clocks {clk2}]
-end

# OR
set_multicycle_path -setup 2 -from [get_pins {regA/c}] -to [get_pins {regb/d}]
-end
set_multicycle_path -hold 1 -from [get_pins {regA/c}] -to [get_pins {regb/d}]
-end
```

**Notes**

- Setup path MCP modifies the default hold relation as well. In summary, if a setup multicycle of N cycles is specified, then most likely a hold multicycle of N-1 cycles should also be specified. A good rule of thumb for multi-frequency multicycle path specification for slow to fast clock domain paths is to use the -end option. [1]

## DESIGN PROBLEM 2 – SLOW TO FAST MCP (non-integer multiple)

In a design, data is launched at rising edge of Tclk1 = 20 ns and it is captured at rising edge of Tclk2 = 15 ns. The clock relationship is therefore in a non-integer ratio of 4:3. For no data loss and matched throughput, the sender and receiver are operating in sync where launching happens every 3 clock cycles and capturing happens every 4 clock cycles. These paths are by default analyzed pessimistically with most restrictive setup-hold relationship. Timing can be relaxed in these paths by enforcing the design-intended setup-hold relationship. Convey this requirement to the STA tool. Assume both clocks are synchronous and driven from input ports.



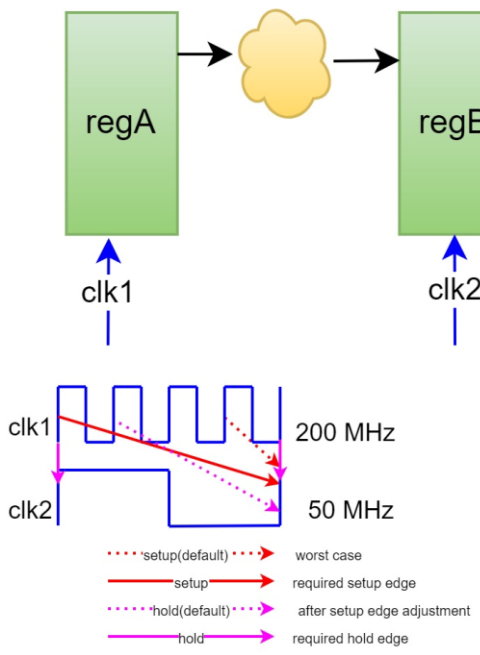
**SDC**

When clocks are in a non-integer ratio, the exact setup and hold requirement may not always be possible to be established with MCP constraints.

```
# Create all clocks
create_clock -name clk1 -period 20.00 -waveform {0 10} [get_ports clk1]
create_clock -name clk2 -period 15.00 -waveform {0 7.5} [get_ports clk2]

# Set MCP to shift the setup edge to 4th edge of clk2 for 40->90 = 50ns Tsetup
set_multicycle_path -setup 4 -from [get_clocks {clk1}] -to [get_clocks {clk2}] -
end
# Set MCP to shift back the hold edge by 3 clk2 for 0->0 = 0ns Thold
set_multicycle_path -hold 3 -from [get_clocks {clk1}] -to [get_clocks {clk2}]
-end

# The best solution:
# With MCP, hold relationship looks fine, but we still have 60-50 = 10ns pessimism
# on setup requirement, hence use set max/min delay constraints…
set_max_delay -rise_from [get_clocks clk1] -rise_to [get_clocks clk2] 60.00
set_min_delay -rise_from [get_clocks clk1] -rise_to [get_clocks clk2] 0
```

**DESIGN PROBLEM 3 – FAST TO SLOW MCP (integer multiple)**

In a design, data is launched at rising edge of clk1 = 200 MHz every four cycles, and it is captured at every rising edge of clk2 = 50 MHz. These paths are by default analyzed pessimistically as single-cycle paths. Timing can be relaxed in these paths by enforcing the design-intended setup-hold relationship. Convey this requirement to the STA tool. Assume both clocks are synchronous and driven from input ports.

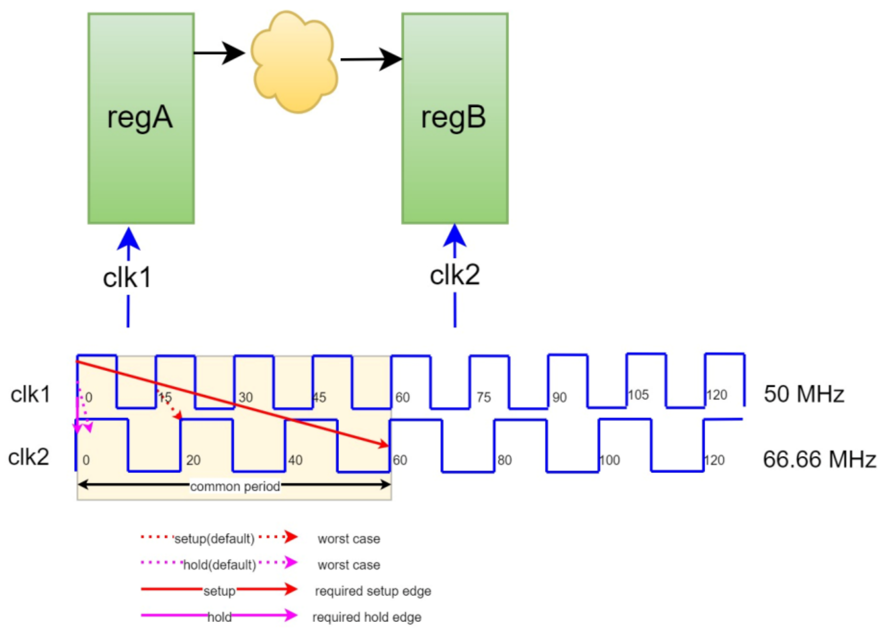


**SDC**

```
# Create all clocks
create_clock -name clk1 -period 5.00 [get_ports clk1]
create_clock -name clk2 -period 20.00 [get_ports clk2]

# Set MCP to shift the setup edge to 4th edge of clk1, default = 1st edge in -ve
# direction
set_multicycle_path -setup 4 -from [get_clocks {clk1}] -to [get_clocks {clk2}] -
start
# Set MCP to shift the hold edge by +3 clk1 cycle
set_multicycle_path -hold 3 -from [get_clocks {clk1}] -to [get_clocks {clk2}] -
start
```

## DESIGN PROBLEM 4 – FAST TO SLOW MCP (non-integer multiple)

In a design, data is launched at rising edge of Tclk1 = 15 ns and it is captured at rising edge of Tclk2 = 20 ns. The clock relationship is therefore in a non-integer ratio of 4:3. For no data loss and matched throughput, the sender and receiver are operating in sync where launching happens every 4 clock cycles and capturing happens every 3 clock cycles. These paths are by default analyzed pessimistically with most restrictive setup-hold relationship. Timing can be relaxed in these paths by enforcing the design-intended setup-hold relationship. Convey this requirement to the STA tool. Assume both clocks are synchronous and driven from input ports.



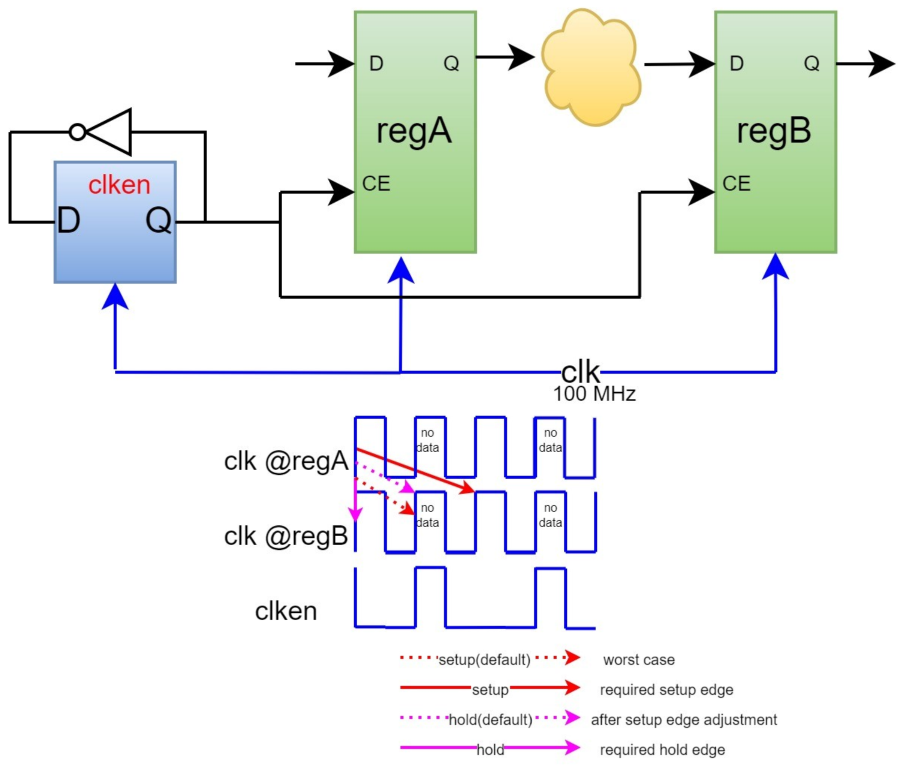
## SDC

When clocks are in a non-integer ratio, the exact setup and hold requirement may not always be possible to be established with MCP constraints.

```
# Create all clocks
create_clock -name clk1 -period 15.00 [get_ports clk1]
create_clock -name clk2 -period 20.00 [get_ports clk2]

# Set MCP to shift the setup edge to 4th edge of clk1 in -ve direction for 30->80
# = 50 ns Tsetup
set_multicycle_path -setup 4 -from [get_clocks {clk1}] -to [get_clocks {clk2}] -
start
# Set MCP to shift the hold edge by +3 clk2 from default hold edge at 15 for 0->0
# = 0 ns Thold
set_multicycle_path -hold 3 -from [get_clocks {clk1}] -to [get_clocks {clk2}] -
start

# The best solution:
# With MCP, hold relationship looks fine, but we still have 60-50 = 10ns pessimism
# on setup requirement, hence use set max/min delay constraints…
set_max_delay -rise_from [get_clocks clk1] -rise_to [get_clocks clk2] 60.00
set_min_delay -rise_from [get_clocks clk1] -rise_to [get_clocks clk2] 0
```

## DESIGN PROBLEM 5 – MCP at HALF CLOCK RATE

A design runs at clock frequency = 100 MHz. Internally, the design generates a clock enable pulse every two clock cycles. The registers in the design launches/captures data only when the clock enable is high. Therefore, effectively, the data is launched/captured at half clock rate. These paths are by default analyzed pessimistically as single-cycle paths. Timing can be relaxed in these paths by enforcing the design-intended setup-hold relationship. Convey this requirement to the STA tool. Assume the clock is driven from an input port.



**SDC**

```
# Create clock
create_clock -name clk -period 10.00 [get_ports clk]

# Set MCP to shift the setup edge to 2nd edge of clk, default = 1st edge at all
# enable-driven destination registers
set_multicycle_path -setup 2 -to [get_fanouts [get_pins clken|q] -through
[get_pins -hierarchical *|ena]] -end

# Set MCP to shift the hold edge by -1 clk cycle at all enable-driven destination
# registers
set_multicycle_path -hold 1 -to [get_fanouts [get_pins clken|q] -through [get_pins
-hierarchical *|ena]] -end
```

## Epilogue

Currently, this document discusses only clock constraints and timing exceptions. I am planning to add IO path constraints for system & source synchronous designs as well and update the document in future.

## References

[1] "Static Timing Analysis for Nanometer Designs – A Practical Approach", by J. Bhasker, Rakesh Chadha, *Springer*

[2] "Principles of VLSI RTL Design", by Sanjay Churiwala, Sapan Garg, *Springer*