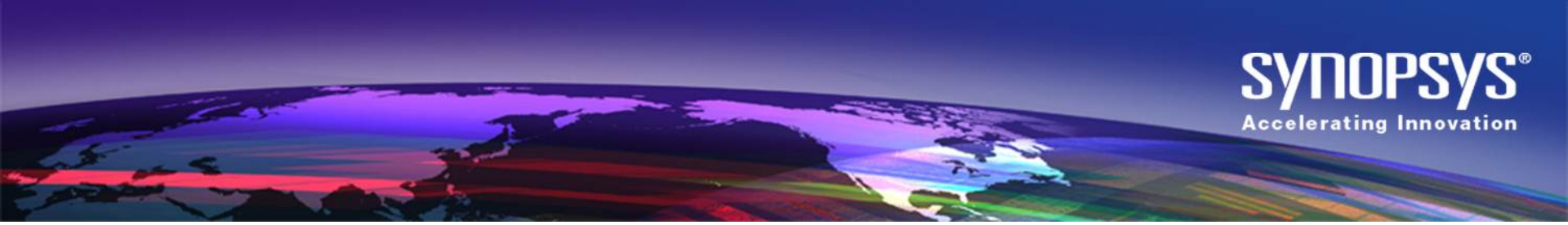


Equivalence Checking for Timing Constraints

Subramanyam Sripada

March 6, 2014

Agenda

- Motivation
- Existing Solutions
- Multi-pass approach
- Results
- Conclusion

Motivation

- Today's timing constraints (SDC) are quite complex
  - 10M lines of SDC for a 25M instance design
- Constraint management is a big issue:
  - Different constraints through different stages of the design flow
  - Constraint modifications and optimizations
  - Hierarchical methodology with multiple teams
  - ECOs
  - Large number of modes

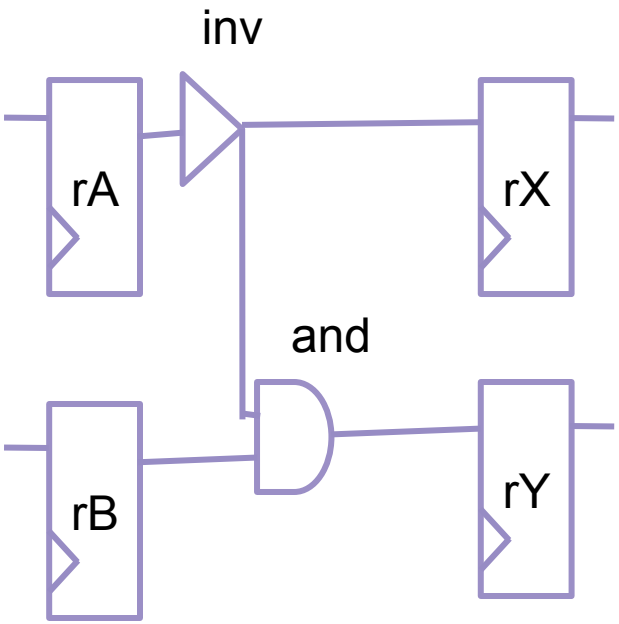
Terminology

- “Equivalence” of constraints is defined as identical timing characteristics (clocks, exceptions etc) for each and every path of the design
- 1-design/2-SDC: Same netlist, two sets of SDC
- 2-design/2-SDC: Logically equivalent designs can have different timing constraint behavior
- Block vs top: Constraint correlation between blocks and chip-level
- Verification of Merged Modes: Verifying correctness of automated or manually merged constraints

Existing Solutions

1. Manual Checking: Manually inspect the constraint/ design changes
2. Structural Checking: Iterative approach based on pairwise comparison of set_case_analysis, clocks, input/output delays, exceptions, clock latency, clock uncertainty, input transition, output load, etc…
3. STA: Compare timing slacks

Motivation



SDC1:
set_multicyle_path 3 –from [get_pins rA/CP]

Equivalent

SDC2:
set_multicycle_path 3 –through [get_pins inv/Z]

SDC3:
set_multicyle_path 3 –from [get_pins rA/CP]
set_multicycle_path 2 -through [get_pins and/Z]

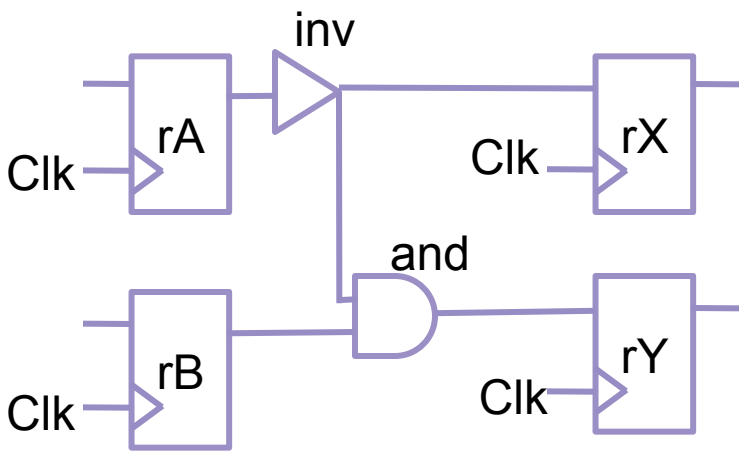
Not Equivalent

SDC4:
set_multicycle_path 3 –through [get_pins inv/Z]
set_multicycle_path 2 –through [get_pins and/Z]

Multi-Pass Approach

- Intention to compare “timing behavior” on each and every path of the design
- Adaptive multi-pass approach to achieve good performance
  - Gradual refinement to move to higher pass only if ambiguous in previous pass

1-design/2-SDC



SDC1:
set_multicyle_path 2 –through [get_pins rA/CP]
set_false_path -through [get_pins and/B]

SDC2:
set_multicyle_path 2 –through [get_pins rA/CP]
set_false_path -through [get_pins rB/CP]

Timing SP	Launch clock	Timing EP	Capture clock	SDC1	SDC2	Result
*	Clk (r)	rX/D	Clk(r)	MCP (2)	MCP (2)	Match
*	Clk (r)	rY/D	Clk(r)	FP, MCP (2)	FP, MCP (2)	Ambiguous
rA/CP	Clk (r)	rY/D	Clk(r)	MCP (2)	MCP (2)	Match
rB/CP	Clk (r)	rY/D	Clk(r)	FP	FP	Match

Pass 1 characteristics: Endpoint matches/ mismatches

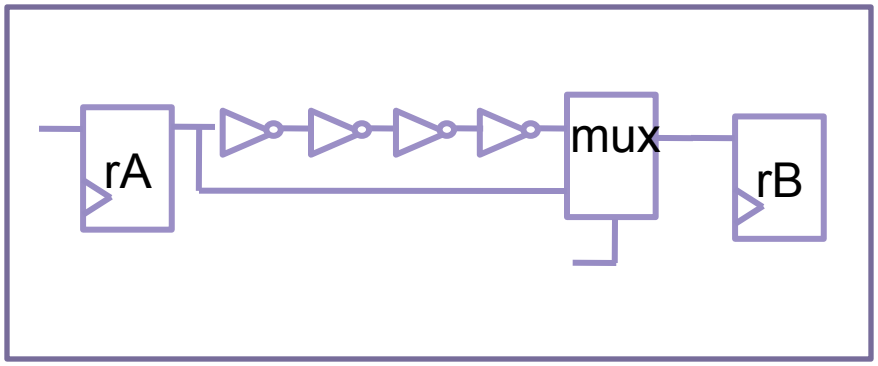
Pass 2 characteristics: Startpoint matches/ mismatches for given endpoint

Path set

Exception State

1-design/2-SDC

Re-convergence example



SDC1:
set_multicycle_path 2 –through mux/A

SDC2:
set_multicycle_path 2 –through mux/B

Timing SP	Launch clock	Timing EP	Reconv ergence point	Captur e clock	SDC1	SDC2	Result
*	Clk (r)	rB/D	*	Clk(r)	Valid, MCP (2)	Valid, MCP (2)	Ambiguou s
rA/CP	Clk (r)	rB/D	*	Clk(r)	Valid, MCP (2)	Valid, MCP (2)	Ambiguou s
rA/CP	Clk (r)	rB/D	mux/A	Clk(r)	MCP (2)	Valid	Mismatch
rA/CP	Clk (r)	rB/D	mux/B	Clk(r)	Valid	MCP (2)	Mismatch

Path set

Exception State

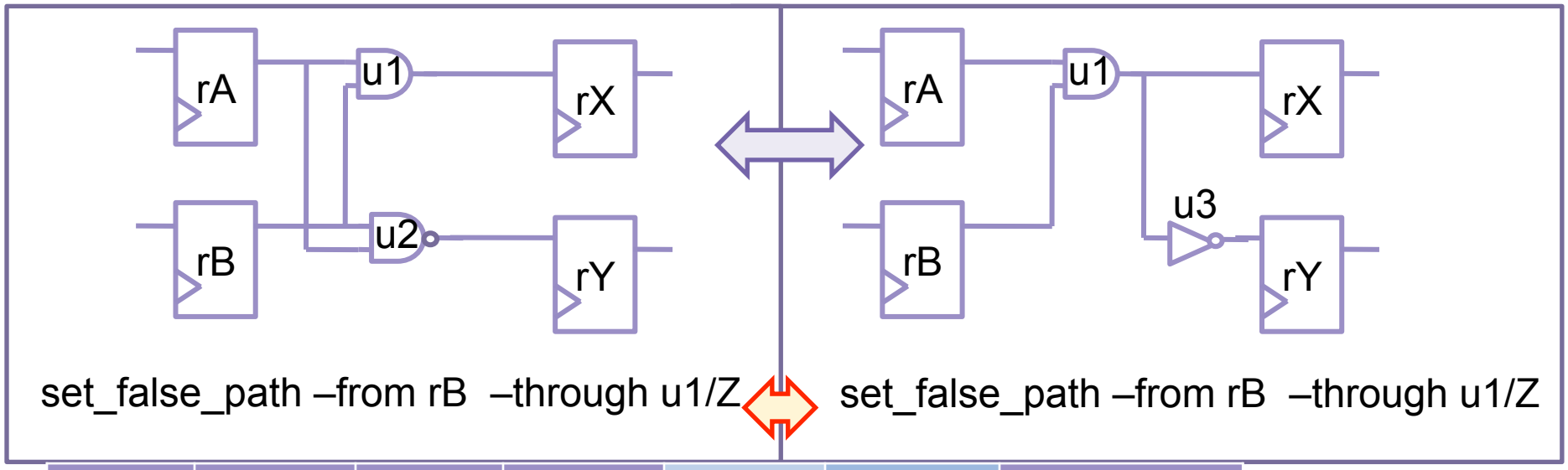
Pass 1 characteristics

Pass 2 characteristics

Pass 3 characteristics
Re-convergent points within one {startpoint, endpoint} pair

2-design/2-SDC

Object matching done by
- names
- custom mapping



Timing SP	Launch clock	Timing EP	Capture clock	SDC1	SDC2	Result
*	Clk (r)	rX/D	Clk(r)	Valid, FP	Valid, FP	Ambiguous
*	Clk (r)	rY/D	Clk(r)	Valid	Valid, FP	Mismatch
rA/CP	Clk (r)	rX/D	Clk(r)	Valid	Valid	Match
rB/CP	Clk (r)	rX/D	Clk(r)	FP	FP	Match

Pass 1 characteristics

Pass 2 characteristics

Path set

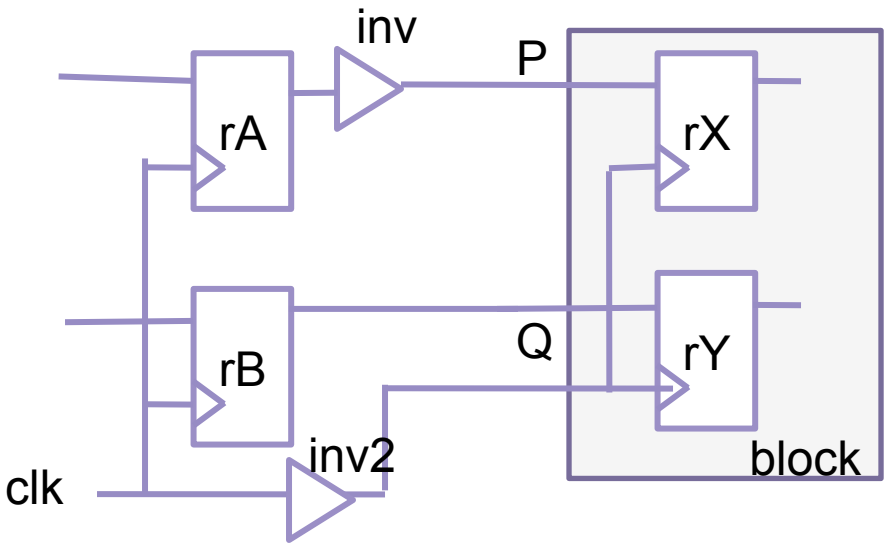
Exception State

2-design/2-SDC

- Difficult to handle netlist changes that change the timing paths:
  - Different registers: The extra/missing startpoints (pass 2) and endpoints (pass 1) will be reported as inconclusive or mismatch
  - Different re-convergence points: The set of paths between the startpoint and endpoint can be reported as inconclusive or mismatch
  - Might be alleviated with additional inputs

Block vs Top

- Netlist object matching done by names
- Clock matching is done by waveform checks



Top SDC:

```
create_clock –p 10 –waveform {0 5} clk
set_multicyle_path 2 –through [get_pins rA/CP]
```

Block SDC:

```
create_clock –p 10 –waveform {5 0} clk
create_clock –p 10 –waveform {0 5} –name clk_virtual
set_input_delay 2 -clock clk_virtual [get_ports {P Q}]
set_multicycle_path 2 –from [get_ports P]
```

Timing SP	Launch clock	Timing EP	Capture clock	Top SDC	Block SDC	Result
*	clk/ clk_virtual (r)	rX/D	clk(f), clk	MCP (2)	MCP (2)	Match
*	clk/ clk_virtual (r)	rY/D	clk(f)	Valid	Valid	Match

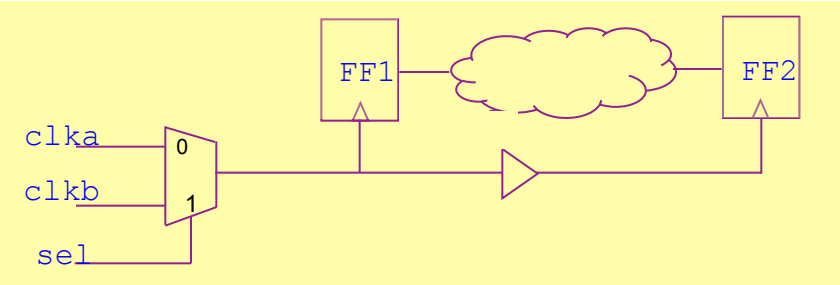
Pass 1 characteristics

Path set

Exception State

Verification of Merged Modes

Union of timing characteristics of individual modes vs characteristics of merged mode



Mode 1 SDC:

```
set_case_analysis 0[get_port SE]
```

Mode 2 SDC:

```
set_case_analysis 1[get_port SE]
```

User Merged SDC:

```
set_false_path –from [get_clocks clkb] –to
    [get_clocks clka]
```

Timing SP	Launch clock	Timing EP	Capture clock	SDC1	SDC2	Merged SDC	Result
*	clka (r)	FF2/D	clka(r)	Valid	Not present	Valid	Match
*	clkb (r)	FF2/D	clka(r)	FP	FP	FP	Match
*	clka (r)	FF2/D	clkb(r)	Not present	Not present	Valid	Mismatch
*	clkb (r)	FF2/D	clkb(r)	Not present	Valid	Valid	Match

Pass 1 characteristics

Path set

Exception state

Comparison to Existing Solutions

	Category		
	Coverage	Runtime	Memory Required
Manual Checking	Limited	Expensive	Not applicable
STA	Only critical paths/ endpoints	Requires delay calculation	Loads one constraint set at a time but keeps timing numbers like arrival times
Structural Checking	All timing paths, but limited SDC changes	No calculation, no enumeration, quite fast	Loads multiple designs/ constraint sets at the same time
Muti-Pass Approach	All timing paths	Path enumeration might be needed for a very small portion of the design	Loads multiple designs/ constraint sets at the same time

Conclusions

- Efficient, non-iterative solution that compares two constraint behaviors for either one or two netlists
  - Compare compressed vs. non-compressed exceptions
  - Compare false paths vs. {case analysis, disable timing, clock groups}
  - Honor exception precedence
- Compared to STA:
  - Covers full design
  - Runs faster
  - Consumes more memory
- Is nicely extensible for all constraint equivalence problems
- Compact set of user-friendly reports based on mismatched constraints and corresponding paths