Validation of Multi-Cycle Path Timing Exceptions in Simulation with Automatically Generated SystemVerilog Assertions

Akshaya Prashanthi Lakshmi Narayanan
Infineon Technologies AG

June 22, 2017
SNUG France

Agenda

Introduction to the Problem

Verification Methodology Used in the Past

Hybrid Verification Flow from Spyglass TXV

Results from Real Life Design Application

Conclusions and Future Work

References

Introduction to the Problem

Multi-Cycle Paths, Risks and Solution

- What are Multi-Cycle Paths (MCPs)?
  - An MCP is defined when more than one clock cycle is required to propagate data from one register to another.
  - Area/timing constraints drive designers to implement large combinational clouds between register stages.

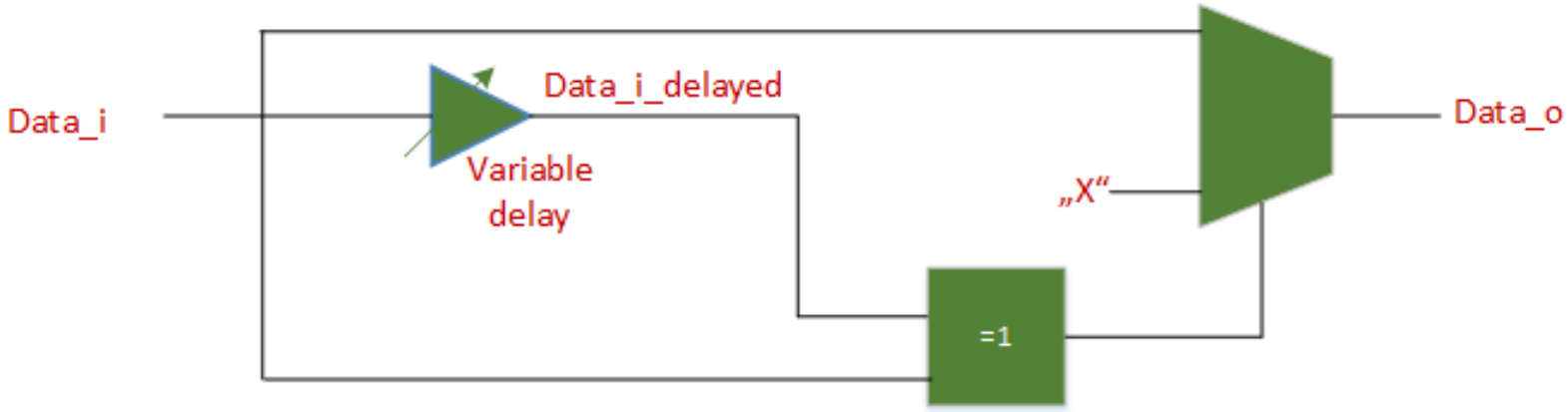
- Risks
  - Incorrectly implemented MCP can lead to functional failure in silicon.

- Solution
  - Verify at Register Transfer Level (RTL) and correct implementation bugs early in the design flow.

Verification Methodology Used in the Past

Insertion of RTL Delay Elements

- Implementation



- Setback
  - Manually inserted ➔ Not robust to changes, susceptible to human errors.
  - No coverage information ➔ No defined confidence level.

Solution : Automated Flow !!!

Hybrid Verification Flow from Spyglass TXV

- Static and Dynamic Verification
- Description of the Flow
- Enable Detection
- SVA Generation
- Cycle vs. Timing Based SVA

Static and Dynamic Verification

- Static verification uses mathematical methods to prove properties of the design.
- Dynamic verification refers to simulation-based verification. It is a test bench based approach.

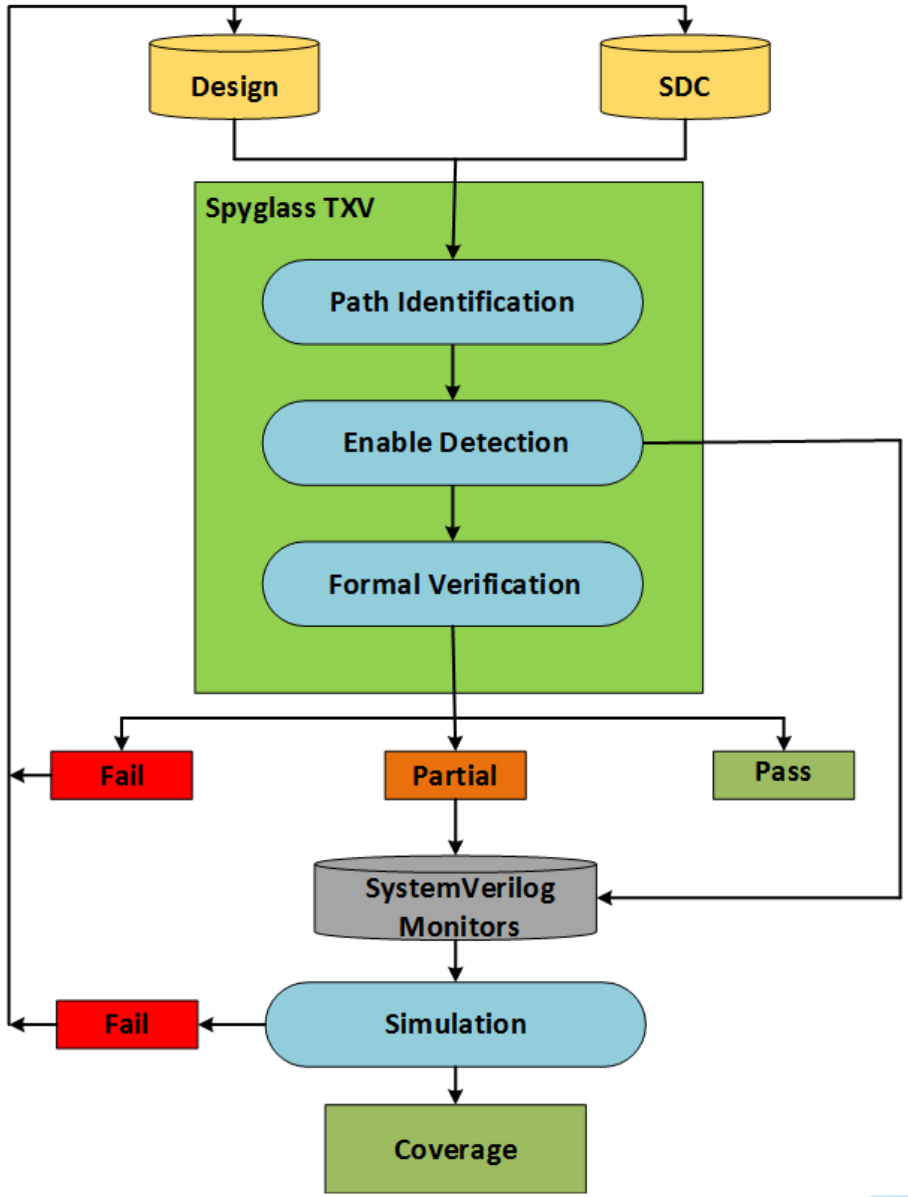
Verification	Dynamic	Static
Depth	Non-Exhaustive	Exhaustive
Test Bench	Yes	Not required
Conclusiveness	Always conclusive	Inconclusiveness increases with increased complexity

- The Hybrid Flow from Spyglass TXV performs static verification on the design and later can be used for dynamic verification to close coverage holes.

Description of the Flow

Inputs to Spyglass TXV
- Design description as RTL/netlist
- MCP timing exceptions
- Definition of design clocks

What Spyglass TXV does :
- Identifies all timing paths between start and end point
- Performs Enable Detection for these paths
- Generates SystemVerilog Assertions (SVA) for each path

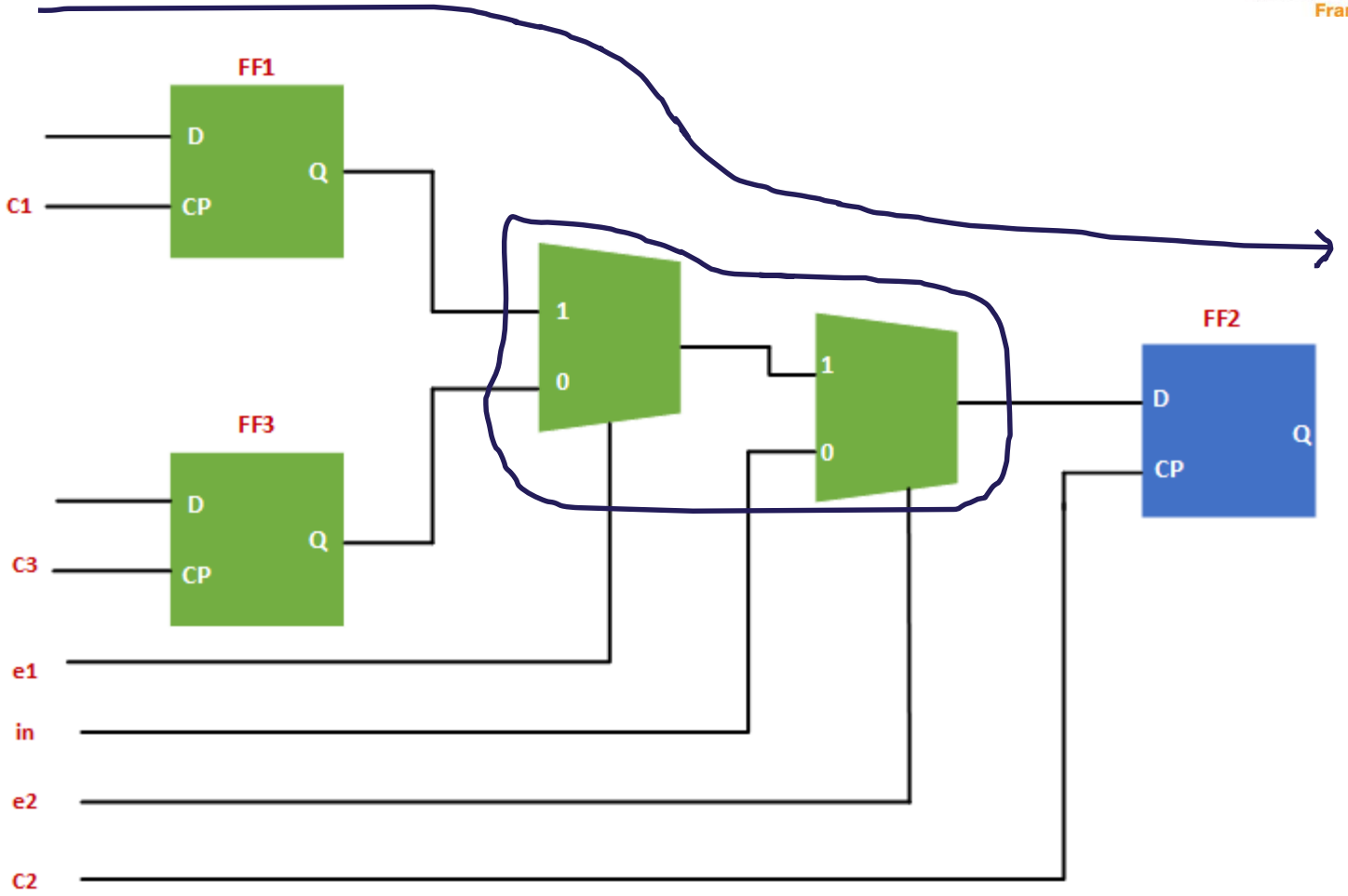


Enable Detection

Spyglass TXV identifies enabling conditions for each data path using control logic structures

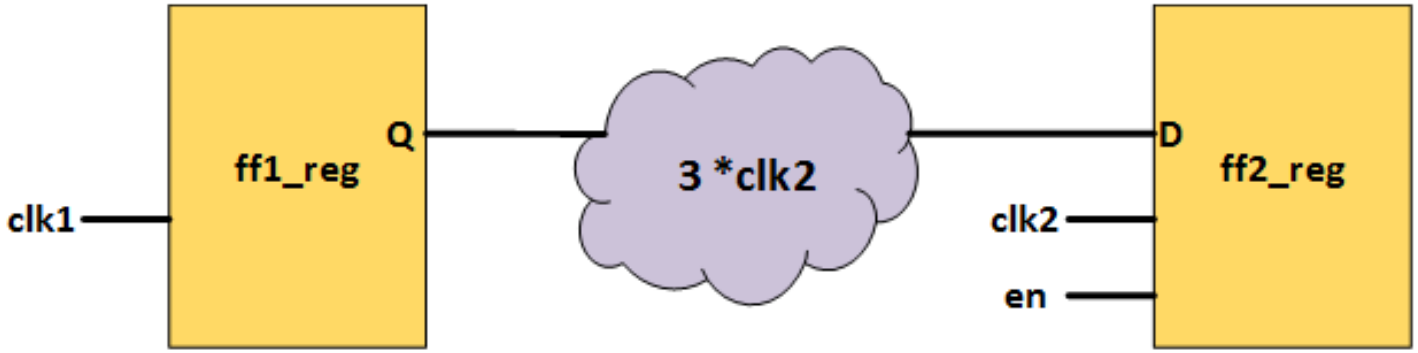
- MUX based enable
- AND gate based enable
- NAND gate based enable
- OR gate based enable
- NOR gate based enable

MCP path ➔ C1 to C2
data path ➔ from FF1 to FF2
Enable condition ➔ e1 & e2



SVA Generation

MCP Timing Constraint and Clock Definition




```
set_multicycle_path 3 -from clk1 -through ff1_reg/Q -through
ff2_reg/D -to clk2
create_clock -period 10 -name clk2 [get_ports clk2]
```

SVA Generation

Corresponding Assertion and Property

```
MCP_Check_mod #("Spyglass - MCP failure (test.sdc:3)", 2, 1)
TXV_MCP_s0_m1_p1(
.mcpclk(clk2),
.from(ff1_reg),
.to(ff2_reg),
.den(en ));
```

```
property mcp_user;
@(posedge mcpclk)
(from != $past(from)) |=> (!((to != $past(to)) &&
$past(den)))[*PATH_MULT];  // for a (PATH_MULT + 1) cycle mcp
Endproperty
```

```
TXV_MCP : assert property (mcp_user)
```

The generated SVA can be Cycle-Based or Time-Based

Cycle vs. Timing Based SVA

Cycle-Based SVA	Timing-Based SVA

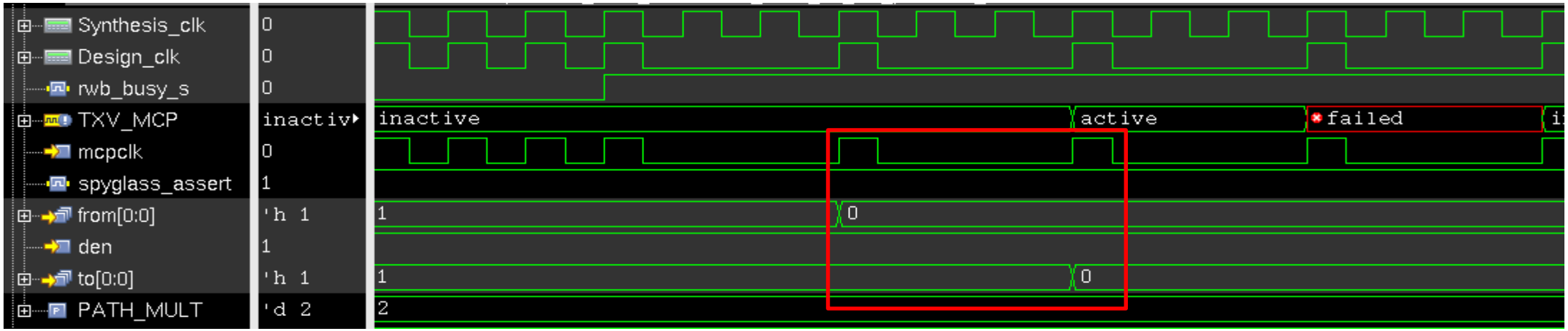
`TXV_MCP_s0_m1_p1(`
`.mcpclk(clk2),`
`.from(ff1_reg),`
`.to(ff2_reg),`
`.den(en ));)`
 	
clk2: Actual design clock.
	`TXV_MCP_s0_m1_p1(`
	`.mcpclk(TXV_clk2),`
	`.from(ff1_reg),`
	`.to(ff2_reg),`
	`.den(en ));)`
	
	TXV_clk2: Tool generated clock.

clk2 need not have a fixed frequency throughout the simulation as it is tied to design.	TXV_clk2 will have a fixed frequency throughout the simulation. The frequency will be taken from the original clock definition for clk2.
Can result in false fails.	Cannot result in false fails.

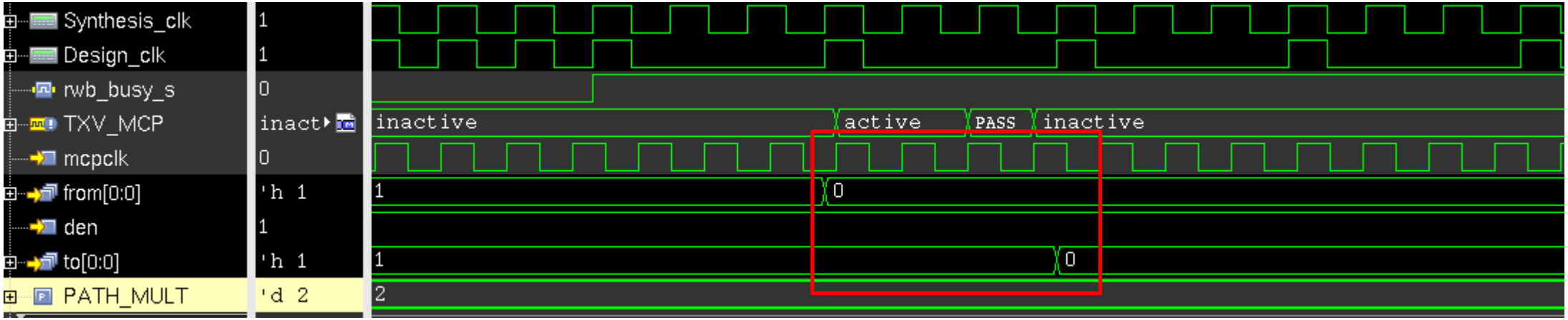
Cycle Vs Timing Based SVA

False fails from simulation

Cycle-Based SVA



Timing-Based SVA



Application on Real Life Design

Results and Coverage Information

Design Description

- In our design, most of the enable signals for the MCPs are driven from software.
- MCP Verification by SVA generation was more beneficial than Formal Verification.

Type	No. of MCP Exceptions	No. of Generated SVA
Clock to Clock	3	725
Register ⟷ Register(bit)	152	441

- Both the types of MCP exceptions included paths which are not functionally relevant. Hence we couldn't achieve 100% coverage with our test cases.

Coverage Results

Assertion coverage metrics was obtained using Cadence Vmanager regression tool

- Assertion Coverage :
  - We were able to cover 66.19% of the assertions.
  - It is not possible to achieve more than this with the current setup.
- Assertion Status Grade:
  - 66.1% of the total assertions passed
  - 99% of the covered assertions passed.
  - One failing assertion could be ignored as it had no functional consequence.
- We had fixed a bug in the design due to an assertion failure!

Current Node: /SG_SVA_PROP

Exclusion Rule Type	UNR	Name	Assertion Average Grade	Assertion Covered	Assertion Status Grade	Valid Metrics
None	none	SG_SVA_PROP	66.19%	736 / 1112 (66.19%)	66.1%	0

Sub-Nodes:

Exclusion Rule Type	UNR	Name	Assertion Average Grade	Assertion Covered	Assertion Status Grade	Valid Metrics
None	none	TXV_MCP_s0_m1_p1	100%	1 / 1 (100%)	100%	256
None	none	TXV_MCP_s0_m1_p2	100%	1 / 1 (100%)	100%	256
None	none	TXV_MCP_s0_m1_p3	100%	1 / 1 (100%)	100%	256
None	none	TXV_MCP_s0_m1_p4	100%	1 / 1 (100%)	100%	256
None	none	TXV_MCP_s0_m1_p5	100%	1 / 1 (100%)	100%	256
None	none	TXV_MCP_s0_m1_p6	100%	1 / 1 (100%)	100%	256
None	none	TXV_MCP_s0_m107_p2	100%	1 / 1 (100%)	0%	256

Conclusions and Future Work

- Conclusions
  - Automated approach to verify MCP exceptions leads to better design quality and faster closure of design verification.
  - Obtain better verification coverage information .The quality and number of the test cases determines the simulation coverage for the MCP assertions.
  - Detect errors early in the design flow.

- Future Work
  - Develop a post-processing tool to remove the functionally irrelevant SVA.
  - Extend the application of the tool to more complex designs with deeper levels of hierarchy.
  - Generation of a schematic for every enable logic that is inferred by the tool. This would help in easier debugging.

References

- Ping Yeung, “Advanced static verification for SoC designs,” SoC Design Conference (ISOCC), 2009.
- Walter Soto Encinas, “Infrastructure for formal and dynamic verification of peripheral programming model,” Test Symposium (LATS), 2016.
- Sudeep Mondal, “Hybrid Verification Flow,” Atrenta Technical Conference 2013.
- Hongbin Zheng, “High-level synthesis with behavioral level multi-cycle path analysis,” Field Programmable Logic and Applications (FPL), 2013.

I would like to thank my co-authors:

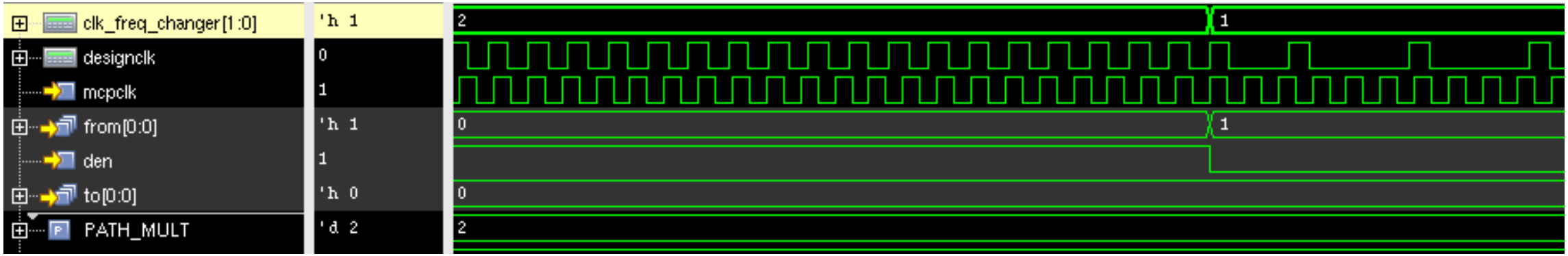
- James Gillespie, Synopsys GmbH, Munich, Germany
- Abhinav Singla, Synopsys India Pvt Ltd, Noida, India
- Sudeep Mondal, Synopsys India Pvt Ltd, Noida, India
- Barsneya Chakrabarti, Synopsys India Pvt Ltd, Noida, India

Thank You

TXV_clk2 defn

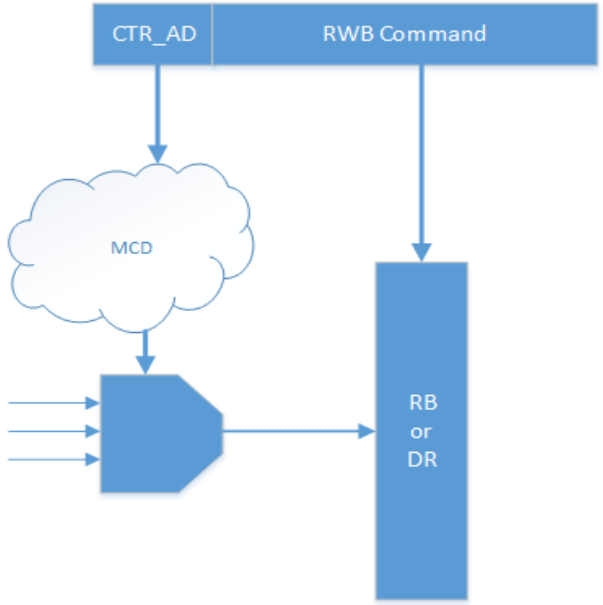
```
reg TXV_C1;
initial begin
    TXV_C1 <= 1'b0;
end
always #10.000000 TXV_C1 = ~TXV_C1;
 mcpclk = TXV_C1;
```

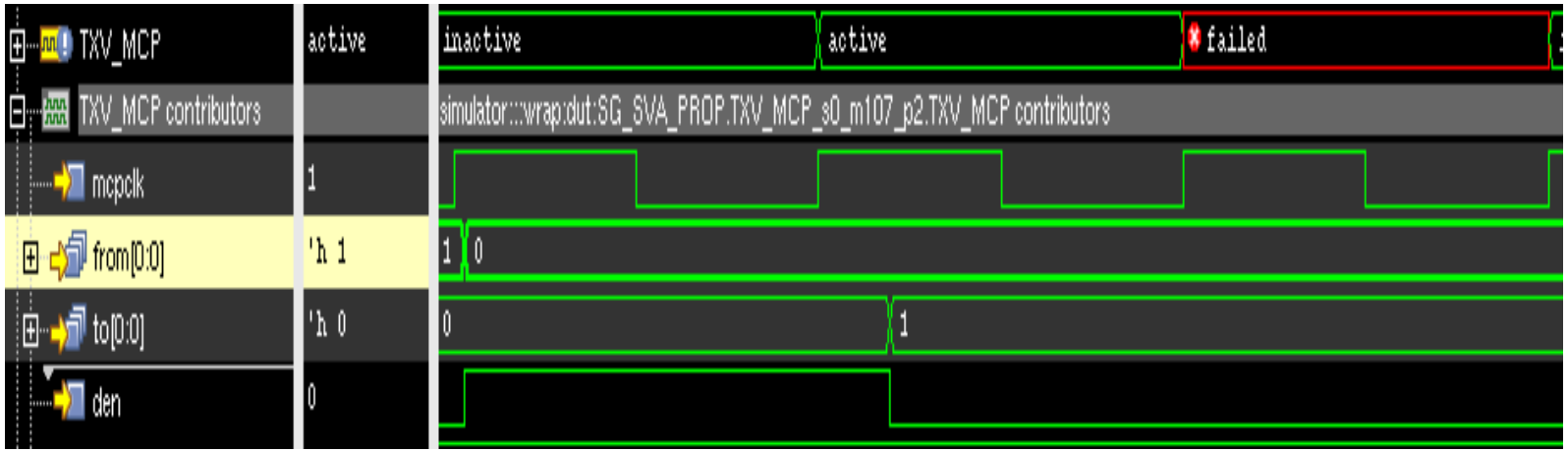
- Example from our Actual Design to illustrate the problem:



Detection of a Failing MCP

- In our setup , the software was fetching data from “RB or DR” register in the very next clock cycle after “CTR_AD” was fed.

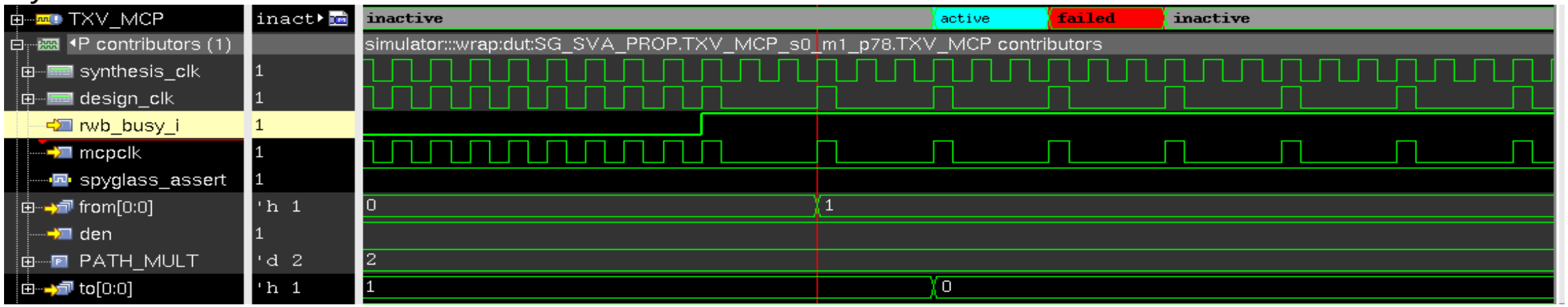




Cycle Vs Timing Based SVA

False fails from simulation

Cycle-Based SVA



Timing-Based SVA