Systematic Speedup Techniques for Functional CDC Verification Closure

Sulabh Kumar Khare, Ashish Hari

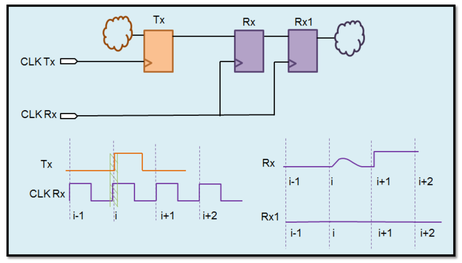
Mentor Graphics Corporation

What is Functional CDC Verification

- Every Clock Domain Crossing (CDC) synchronizer must follow some protocol for functional correctness and to avoid data loss
- Verification of synchronizer protocols using simulation/Formal techniques is called functional CDC verification

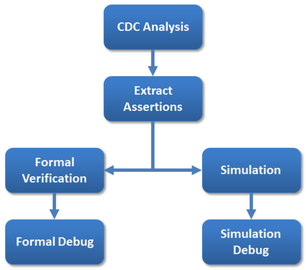
Why Functional CDC Verification

- Modern SOCs depend on multiple asynchronous clock domains to meet increasing high performance and low power needs
- Adding a synchronizer alone does not guarantee that the CDC signal will pass through reliably, without getting lost
- Example of protocol violation (data loss) in a 2-DFF synchronizer:



Existing Functional CDC Verification Flow

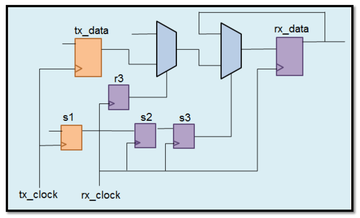
- Starts after the completion of static CDC analysis:



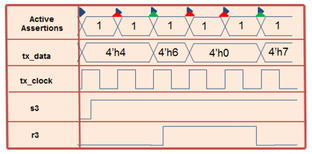
- Generates properties/assertions for CDC paths, which are then verified using simulation or formal techniques
- Results are separately debugged in a Formal or a Simulation environment

Issues With The Existing Flow

- Noise in results due to redundant checking
  - Example of synchronizer with an enable condition for data transfer:



  - Waveform for protocol check in the absence of an enable condition:



- Inaccurate verification with variation in clock frequency:

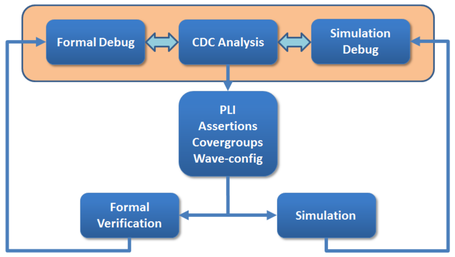
```
property data_stable(data,  tx_clock, reset, areset, NUM_CYCLES);
  @(posedge tx_clock) disable iff(areset)
    !reset && $changed(data) |=> $stable(data)[*(NUM_CYCLES-1)];
endproperty
```

- Increased time to closure due to missing functional coverage
- Limited debug capabilities
- Inconsistent setup for formal verification
- Inaccurate results due to single clock assertion:

```
property data_stable(data, rx_clock, reset, areset);
    @(posedge rx_clock) disable iff(areset)
      !reset && $changed(data) |=> $stable(data)[*2];
endproperty
```

Proposed Functional CDC Verification Flow

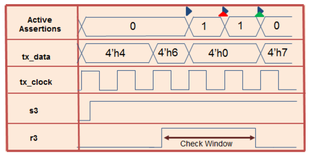
- Starts with the CDC analysis step and applies techniques to accelerate the verification closure:



- Brings functional CDC verification debug closer to CDC analysis

Speedup Techniques

- Check assertions only when required
  - Use design analysis to infer the enable condition for data transfer
  - Waveform for protocol check with the enable condition s3&&r3:

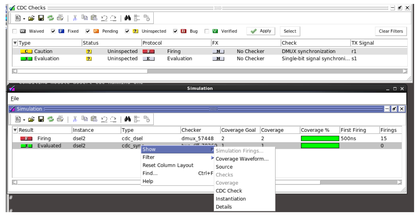


- Ensure consistency in clock frequencies
  - Clock frequency checking module
  - Dynamically calculate frequency at run time
- Define coverage using SVA covergroups:

```
covergroup cg_cdc_gray_coded @(posedge tx_clock);
   cp_values_checked  : coverpoint incr_values_checked            { bins cpb_values_checked = {1'b1}; }
   cp_changed_tx_min : coverpoint incr_changed_tx_min            { bins cpb_changed_tx_min = {1'b1}; }
   cp_all_zero_to_one :  coverpoint &zero_to_one_transitions   { bins cpb_zero_to_one_transitions = {1'b1}; }
   cp_all_one_to_zero : coverpoint  &one_to_zero_transitions    { bins cpb_one_to_zero_transitions = {1'b1}; }
endgroup: cg_cdc_gray_coded

cg_cdc_gray_coded cg_i_cdc_gray_coded = new;
```

- Enhance debug techniques
  - Minimize data for debug using PLI integration to report only unique set of errors
  - Configuration file to load the relevant cone of influence signals in the simulator
  - Integration of protocol coverage with CDC coverage:



- Automatically generate formal run setup automatically that is consistent with the CDC analysis setup
  - Clocks, resets, constants, black boxes
- Use more accurate assertions that handle asynchronous data and clock transitions:

```
property data_stable(data, rx_clock, reset, areset);
  logic data_l;
  @(data) disable iff(areset)
    (1, data_l = !data) |=>
            @(posedge rx_clock)
                    (reset || data_l === data) |=> $stable(data)[*1];
endproperty
```

Case Study: Simulation

- Used a set of SoC designs ranging from 1 to 100 million gates and containing 20 to 240 clocks
- Comparison of simulation results:

Existing flow and methods

Test	Assertion Errors	Run Time (Minutes)	Coverage (%)
1	37563	1:08	Not Available
2	27690	2:44	Not Available
3	819	1:29	Not Available
4	6373	1:20	Not Available

Proposed techniques applied

Test	Assertion Errors	Run Time (Minutes)	Coverage (%)
1	360	0:33	74
2	320	1:11	56
3	191	0:28	83
4	223	0:81	73

- Significant reduction in assertion errors observed with enable condition in the assertion
- Run time improvement observed due to less number of assertion errors with PLI integration
- Integrated covergroups and cover properties helped in measuring the quality of simulation runs

Case Study: Formal

- Used a set of SoC designs ranging from 1 to 100 million gates and containing 20 to 240 clocks
- Comparison of formal verification results (number of assertions):

User defined setup

Test	Total	Proven	Fired	Inconclusive
1	657	81	50	526
2	2274	22	1115	1137
3	354	33	92	229
4	891	2	498	391

Automatically generated setup

Test	Total	Proven	Fired	Inconclusive
1	657	81	544	32
2	2274	273	1064	937
3	354	178	102	74
4	891	323	523	45

- Less iterations required to constraint formal verification tool to get conclusive results with the generated setup
- Reduced number of inconclusive assertions initial runs

Conclusion

- The proposed techniques significantly improve the functional verification closure time of CDC paths
- Simulation performance is improved and the quality of verification is assessed with advanced coverage metrics
- Formal verification setup time is reduced through automatic generation of setup from CDC analysis
- The turnaround time of simulation and formal debug is minimized by the integration of formal debug with CDC results

TEMPLATE DESIGN © 2008
www.PosterPresentations.com