2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

A step towards Zero Silicon Bugs using Assertion based Assumption Validation

Rohit Kumar Sinha, Intel India, Bangalore, rohit.kumar.sinha@intel.com

Babu Christie, Intel India, Bangalore (christie.babu@intel.com)

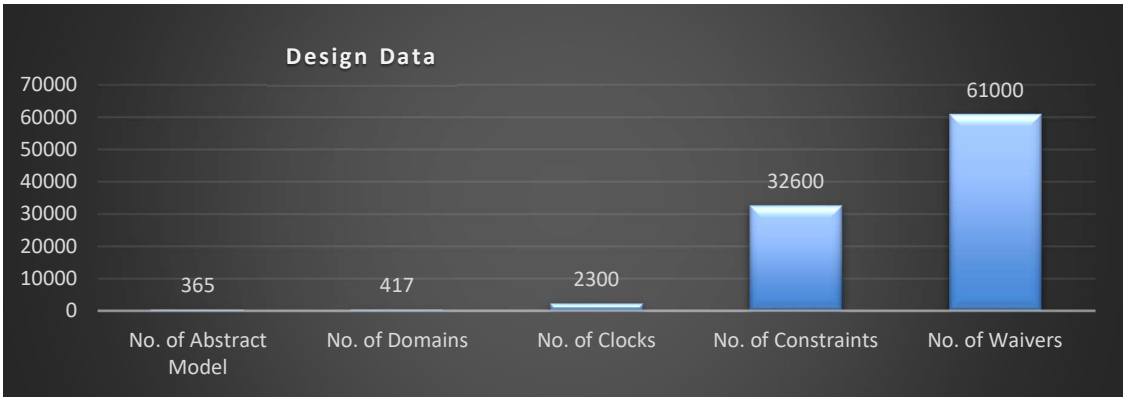
***Abstract*— As the complexity of SoCs is exponentially increasing and IPs are being sourced from multiple external and internal channels, the validation of asynchronous designs and ensuring that there are no potential misses in the SoC integration have become a daunting task. During the SoC integration, we often get cases wherein pre-silicon or sometimes post-silicon bug cost an entire respin due to meta-stability issues or due to glitches in the clock-reset paths. That's the reason there is an absolute necessity mainly in the SoC design to ensure that the assumption used for signing off CDC or RDC design challenges are validated using an autonomous flow. The flow should ensure that the assumptions that have been taken to close CDC analysis are validated with respect to the design intent such that constraints added because of the wrong interpretation can be detected upfront to avoid costly iterations**

***Keywords*—** *SVA, assertions, formal, verification, asynchronous, validation, protocols, SoC, CDC*

I. INTRODUCTION

In the ongoing ARM based SOC design, there are multiple flavors of IPs being integrated and IP team often signs off their design using their own TFM and it is nearly impossible for the SoC team to expect from IP vendor-either internal or external to follow the TFMs as per the SoC guidelines. As a result, SoC integration quality sign off becomes very challenging and probability of wrong constraints being used in the CDC closure is quite high.

In the previous ARM based SOC, we typically have the following CDC data which provides an overview of the complexity of the SoC complexity.



As mentioned in the graphs, handling hue number of constraints and waivers are very challenging and hence it requires an additional validation mechanism which ensures that the assumption used for the CDC analysis are indeed correct in the actual design implementation.

There are essentially below motivations for enabling SVA based assumption protocol validation.

A. SoC Complexity

As we know that yesterday's SoCs are becoming today's IP and SoC architecture is increasingly becoming complex and that's the reason clocking architecture, reset architecture complexity calls for a detailed analysis for the asynchronous crossings.

B. Vulnerability

At times, it is difficult to understand the interpretation of the constraints, waivers and violations reported by the CDC tool and because of which few real issues are either waived or ignored. RTL integration engineer is often vulnerable to miss some critical crossing because of incorrect assumptions

C. Costly Respin

Every designer understands that cost associated with 1 Si respin and companies have got out of business in the past because of the missing TTM because of the CDC issues. Mainly CDC issues are often random in nature and it leads to fatal issues if it is detected post Si. So one must be extremely careful in handling metastability issues during the CDC analysis.

II. IMPLEMENTATION OF SVA PROCTOCL VALIDATION

The purpose of this paper is to address the mentioned issues by providing a comprehensive methodology to ensure that the constraints and assumptions that are used for CDC closure are validated against the design intent and there are no misses leading to potential silicon escapes.

There are essentially four major steps involved in the SVA protocol validation.

A. Translation of CDC waivers into Constraints

Firstly, we were required to associate all flavors of waiver with the corresponding SGDC constraints as none of the industry standard CDC tool directly converts waivers into assertions. As shown in the table below, we captured most common types of waivers and then we mapped to the equivalent SGDC constraints.

Type of Waivers	**Related Constraints**
Stable and non-glitch prone signals	Quasi_static
Pulse extender in the crossing path	Clock_relation
MetaFlop in the crossing path - enable_multiflop_sync = yes (sync_cell)	- enable_multiflop_sync = no, add synchronize_cell “instance_name”
Debug modules (VISA & IDV) network signals not Impact on CDC(crossings between test clock & functional clock are waived) - set_clock_groups	Set_clock_groups
Xover in the crossing path - Clock_relation (posedge/negedge)	Clock_relation
Signal going to Power control unit -	Quasi_static
clkack/clkreq are safe	Handshake protocol; qualifier - enable
PwrGood signal is stable.	Quasi_static
Rx samples the signal once Tx settles down - qualifier can be used	Data_hold_check
There is no activity/transactions happening during the time of reset - **After the reset deassertion, clock is cut off because of the gating logic.**	Reset desertion; reset_filter_path
Registers in bypass mode	Quasi_static
Both TX and RX clocks are aligned.	Clock_relation

Going to config register that is polled by SW	Quasi_static
Mutually exclusive clocks	Set_clock_group
Enable signal asserts long before the valid data is accumulated	qualifier
initial stage mux clocks won't be running or gated during reset de-assertion	Quasi_static_rdc
As per usecase, the d input of the flops will be stable during reset deassertion and will have the value same as reset value	qualifier –src_stable

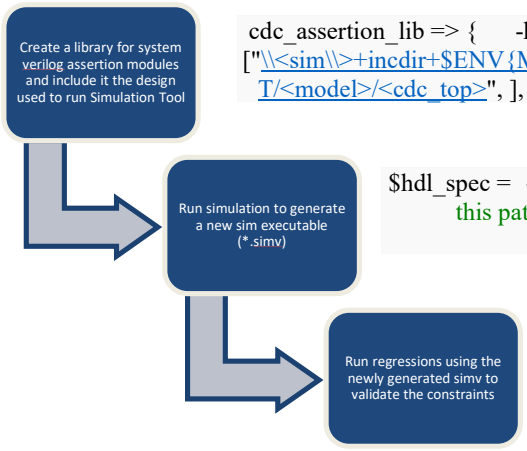
B. Generation of SVA

Secondly, we need to generate the assertions for all the constraints which are used for the CDC closure. In order to generate the assertions, we need to run "cdc_verify_funct" goal which ensure all the CDC constraints SVAs are dumped for the further analysis. Below is the snapshot of the assertions used in the design

Type	SGDC Constraint Name	Purpose	Generated SVAs for the CDC constraints
Assumption	quasi_static -name	• the value of signals is predominantly static • If a clock on the destination flip-flop is stopped • If a reset is active on a destination flip-flop • If the logic in the clock domain crossing path is not sensitive to any metastability issue	• property(@(posedge reset) disable iff (~local_enable || ~Assumption_mod.*_assert)(srcout==initVal)) else $error(msg);
Assumption	set_case_analysis -value	Checks if sca is set to specified value	(~Assumption_mod.*_assert || ~local_enable || (signal==value)) else begin $error(msg);
Assumption	clock_relation –from/-to_clock	Verifies relation between from_clock, to_clock and ref_clock	ADVCDC_CLOCK_POS: assert (~Assumption_mod.*_assert || ~local_enable || neg_period) else begin $error(msg); end
Assumption	reset_filter_path –type reset_sync02 –from_rst/-to_rst –clock	CDC Verifies the sequencing between from_rst and clocks	ADVCDC_DETECT_RFP: assert (~Assumption_mod.*_assert || ~local_enable || (toReset == toReset_val))
Assumption	qualifier –src_stable	Verifies the src_qual !== dest_qual	ADVCDC_DETECT_SRC_STABLE: assert (~Assumption_mod.*_assert || ~local_enable || (src_qual != des_qual))
Assumption	quasi_static_rdc	Verifies that whenever reset changes, flop is already at its reset value	@(data or posedge des_clk) disable iff (~Assumption_mod.*_assert || ~local_enable || ~des_en || des_rst) not ($changed(des_clk) within (1'b1[*margin] ##1 $changed(data) ##1 1'b1[*margin]));
Assumption	reset_filter_path –type rdc –from_rst/-to_rst –clock	RDC Verifies the sequencing between from_rst and to_rst and clocks	ADVCDC_DETECT_RFP: assert (~Assumption_mod.*_assert || ~local_enable || (toReset == toReset_val))

C. Binding the SVA into Simulation Environment

Next Step is to bind the SVAs into the simulation environment so that the targeted tests are run and ensure that all the required assertions are validated. Below are the steps to bind the SVA into functional simulation environment. In the current design, we needed to discuss with Verification Architect to identify the appropriate tests which will ensure that all the required assertions are hit




```
cdc_assertion_lib => {      -hdl_spec => ["cfg/cdc_sva.hdl", ],     -vlog_opts =>
["\\<sim\\>+incdir+$ENV{MODEL_ROOT}/target/<dut>/cdc/*_sva/$PROJECT/<model>/<cdc_top>", ],  ## This is where the assertions will be dumped },

$hdl_spec =  {   -vlog_files => [ "tools/*cdc/<model>/cdc_assertions.sv", ],  ## Add this path to the search paths if it does not already exist   -vhdl_files =>
["verif/tb/cdc_assertions/hybrid_model_sim.vhd", ],

`ifdef INTEL_CDC_ASSERTION module cdc_assertions;    `include "sva_assumptions_<cdc_top>_bind.sv"  // Eg: sva_assumptions_adl_scf_io_top_bind.sv      `include "sva_rules_prop_<cdc_top>_bind.sv"   // these bind files will bind the assertions to the design Endmodule `include "sva_assumptions_<cdc_top>_sim.sv
```

D. Identify the Right Functional Simulation Test

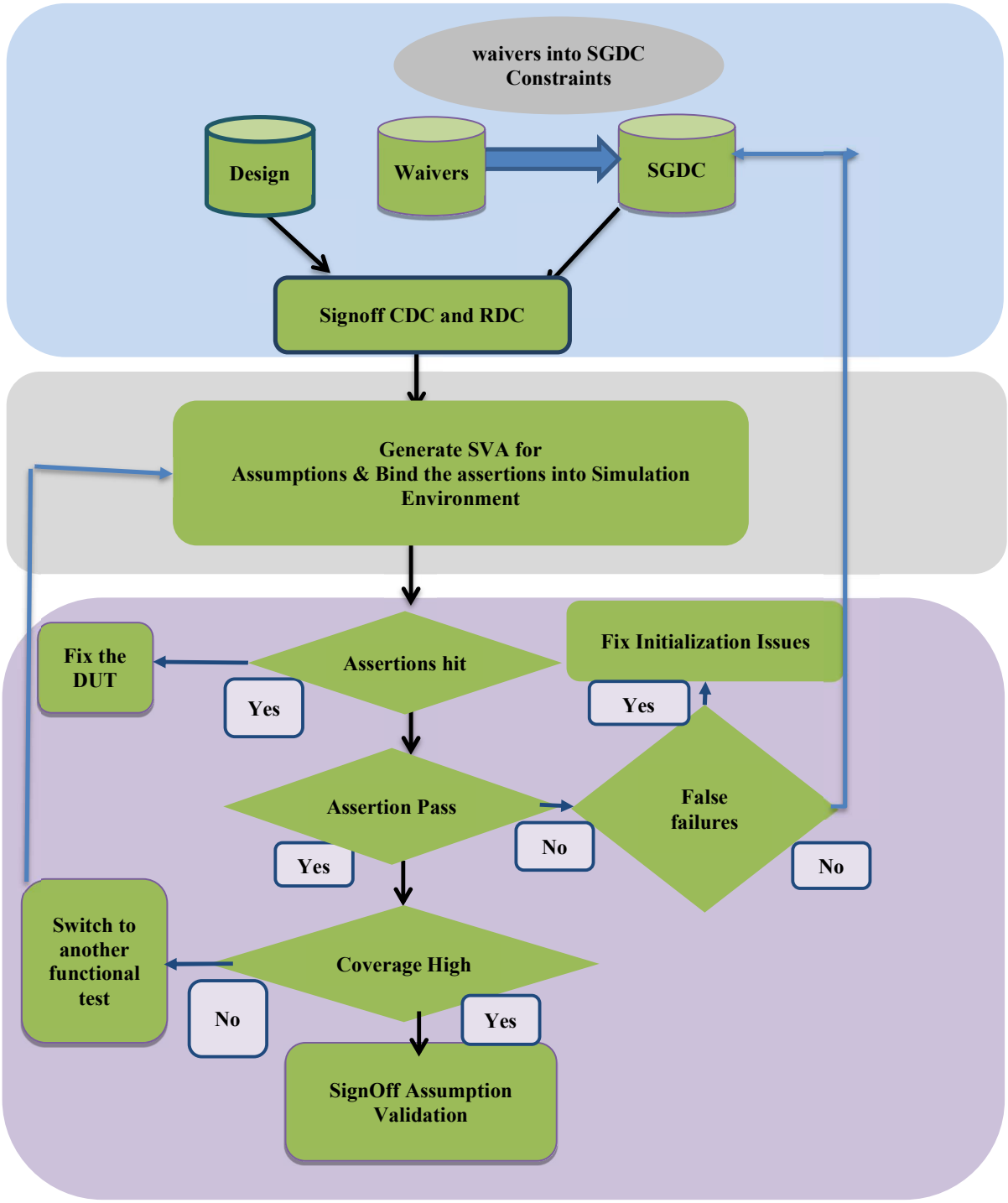
One of the critical steps is to identify the right functional simulation test so that all the assertions get hit and get validated. In the CPUSS design, we used the cluster based test and it helped us to get the assertions analyzed. Below is the summary report which provides the information about assertions

```
"cpuss_tb.Assumption_mod_cpuss.ADVCDC_quasi_static_9.ADVCDC_DETECT_TOGGLE", 885 attempts, 881 successes, 0 failures, 1 incompletes

"cpuss_tb.Assumption_mod_cpuss.ADVCDC_reset_filter_path_0.ADVCDC_DETECT_RFP", 1 attempts, 1 successes, 0 failures, 0 incompletes

"cpuss_tb.Assumption_mod_cpuss.ADVCDC_set_case_analysis_0.ADVCDC_WRONG_VALUE_INIT.unnamed$$_0", 1
```

Next step is to ensure that SVA assertions are analyzes once the design is "active" and out of resets so that assertions don't fail before reset is deasserted. Below are the steps to define the reset conditions.

```
`ifdef INTEL_CDC_ASSERTION
 module cdc_assertions;
    `include "sva_assumptions_<cdc_top>_bind.sv"
    `include "sva_rules_prop_<cdc_top>_bind.sv"
 always@(<reset_signal>)
    begin
if(!(<reset signal> === 1'bx))
        begin
        Assumption_mod.*assert = <signal signal>;
        Assertion_mod.*assert = <signal signal>;
        end else
        begin
        Assumption_mod.*assert = 1'b0;
        Assertion_mod.*assert = 1'b0;
        end
  end
 endmodule
    `include "sva_assumptions_<cdc_top>_sim.sv"
```

III. RECOMMENDED FLOW



IV. RESULTS

The recommended flow was implemented in the ARM based SoC design and the run results are presented for the A53 CPUSS. In this case we had more 5000+ CDC constraints which were used to close the CDC analysis. Below are some of the example of the critical issues that were identified

A. Issues with reset_filter_path constraints

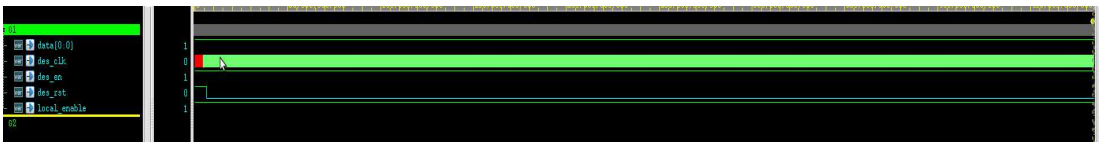
Deassertion filter paths assumption at block level doesn't hold good
example- reset_filter_path –from reset <rst1> -to_clock <clk1>
*Assumption failure (constraint 'reset_filter_path' at /nfs/sc/disks/tbh_rtl_010/rohitks/cpuss-tbh-a0/verif/tests/static_checks/*_cdc/cpuss/cpuss.sgdc:910): Reset cpuss.par_noc_cpuss.par_noc_north_cpu01.cpuss_cpr_wrap.cpuss_cpr.cpuss_cpr_tap_ovrd.tap_mux_dbg_rst_n.o asserted after cpuss.par_noc_cpuss.par_noc_north_cpu01.cpuss_cpr_wrap.cpuss_cpr.cpuss_cpr_tap_ovrd.tap_mux_a53_ncpureset_1.o*

B. Issues with set_case_analysis constraint

Modal constraints defined for DFT Signals (MBIT, TAP) found to be incorrect
*Assumption failure (constraint 'set_case_analysis' at /nfs/sc/disks/tbh_rtl_003/aanto/CDC/func_cdc_ww45_4/cpuss-tbh-a0/verif/tests/static_checks/*_cdc/cpuss/cpuss_clocks.sgdc:404): Value 1 on signal*

* Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors).*

C. Issues with set_case_analysis constraint



Powergood assumptions defined as the quasi-state are wrong
*Assumption failure (constraint 'set_case_analysis' at /nfs/sc/disks/tbh_rtl_003/aanto/CDC/func_cdc_ww45_4/cpuss-tbh-a0/verif/tests/static_checks/*_cdc/cpuss/cpuss_clocks.sgdc:404): Value 1 on signal 3. cpuss_dut.CSTR_SVA_U1_cpuss.ADVCDC_quasi_static_31.ADVCDC_DETECT_TOGGLE: started at 484220000000fs*

Additionally, if you have run simulation tool compile, elab and regressions with the switches mentioned as below, simulation test will dump summary.log in the same area that has acerun.log or <test_name>.log

```
trex  -ace_args -mcrd -simv_args "+fsdb+sva_success" -ace_args- -ace_args -simv_args ""-assert summary -assert report=summary.log"" -ace_args- -fsdb -fsdb_sva_full
```

This file will have summary of all the assertions exercised, passed and failed in that test. You can use cdc_sva_coverage.pl as follows to get the coverage numbers for all CDC SVAs from all the tests you ran. The output of this script will be a text file that has the info about total number of CDC SVAs, total number of failed SVAs (along with their names) and total number of unattempted SVAs (along with their names).

V. SUMMARY

In this paper, we recommended a methodology for the validation of assumption constraint that could be wrongly added during the CDC closure. This methodology addresses the error-prone waiver handling mechanism and it provides design-intent based CDC closure wherein all the waivers are translated into respective constraints and all the assumption constraints are validated using SVA protocol methodology. Below is the summary-

- CDC constraint Assumption validation using SVA Protocol is a MUST especially in SoC Design
- It uncovers the corner cases bug which is impossible through manual reviews
- Binding into the regression suite helps designer to reduce TAT
- Assertion Coverage Data ensures the Sign-off Quality of the validation
- Identifying the appropriate Functional Tests ensure that assertions are hit and validated
- Future Enhancement Recommended covering more constraints for SVA protocol validation
- Formal Technology need to be used for CDC Assumption Validation

ACKNOWLEDGMENT

Thanks to my mentors, peers, CAD team as well as vendors for helping in the whole process

REFERENCES

[1] Rohit Kumar Sinha, Babu Christie, “A step towards zero Silicon Bugs: SVA Protocol Based Assumption Validation