Methodology for SDF back annotated Gatesims for a Mixed signal IP

Saksham Soni
Advanced Micro Devices
Bengaluru, India

Nilay Desai
Advanced Micro Devices
Bengaluru, India

Amlan Chakrabarti
Advanced Micro Devices
Bengaluru, India

***Abstract*- In SDF back annotated Gatesims, it's a challenge to annotate the analog model delays correctly and reduce TAT significantly for the first test. In this paper, we propose a way of annotating delays for the signals going from the analog block to the digital block. We have developed a tool which takes SDF file as input and dumps out a specify block for the analog model, which is then instantiated inside the behavioral model. The simulator picks up the specify block and annotates IOPATH delays mentioned in the SDF file. This makes sure that timing is met in simulations between F2F across analog and digital portions.**

Keywords

SDF – Standard Delay Format,

TAT – Turnaround time,

F2F – Flop to Flop,

IP – Intellectual Property,

IOPATH delay – delay on path from input to output port,

Specify block – used to specify timing information for the module in which the specify block is used.

I. Introduction

Running Simulations at gate level is an increasing trend in the industry before going into the last stage of chip manufacturing. The main reason for running the Gatesims is to catch multicycle paths in the design and reveal any glitches on edge sensitive signatures. It also helps in checking the critical timing paths of asynchronous designs, finding out if any synchronizers are missing in the design and verify the DFT structure inserted during synthesis. There are multiple steps as shown in Fig.1 and challenges when it comes to running the gate level simulations for a mixed signal IP.

Running zero delay simulations is less challenging. Usually for running zero delay Gatesims, we only need the library paths and non-resettable flop list (if any in the design). Once zero delay simulations are clean, next challenge is to mimic the actual delays in the design and run the simulations to make sure timing wise there is no issue in the netlist. This is achieved using the delays provided by physical design team in SDF files.

To start running SDF back annotated Gatesims, we need to identify all the synchronizers and asynchronous paths along with reset synchronizers present in the design so that timing checks can be switched off as these causes x corruption in the design resulting in redundant debugging. When it comes to SDF back annotated Gatesims, the interaction between digital and analog portion is trickiest problem to deal with. It is quite a challenge to annotate the analog model delays correctly so that signals going into and out of digital portions are correct. There are no straight forward ways of getting analog level delays getting annotated in the simulations.

The traditional way of adding the delays between analog and digital block is a manual effort where assignment delays are added while connecting the analog to digital block. This requires lot of iterations to reach to an optimal delay value. Also, we may end up changing those delays every time we get a new SDF. Furthermore, this is not a scalable approach.

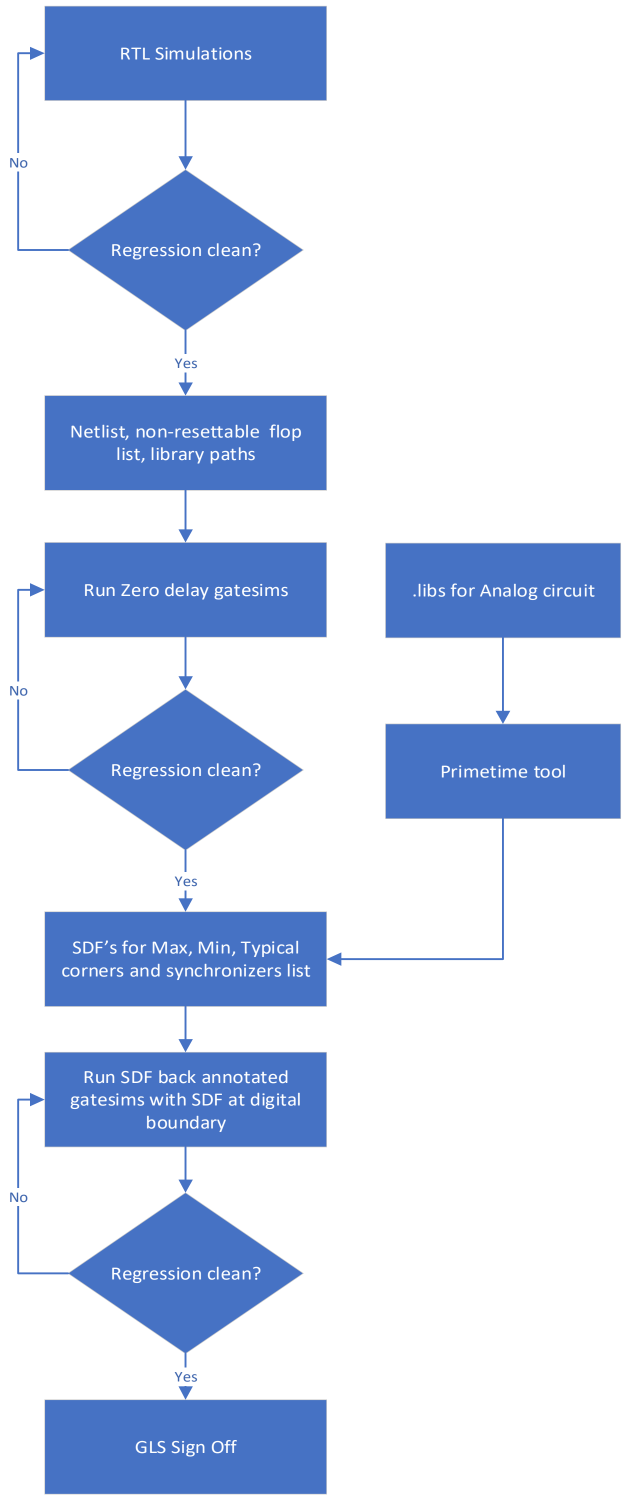

RTL Simulations
No
Regression clean?
Yes
Netlist, non-resettable flop list, library paths
Run Zero delay gatesims
.libs for Analog circuit
No
Regression clean?
Primetime tool
Yes
SDF's for Max, Min, Typical corners and synchronizers list
Run SDF back annotated gatesims with SDF at digital boundary
No
Regression clean?
Yes
GLS Sign Off

Figure 1: Gatesims Flow chart.

II. PROBLEM STATEMENT

A. Background

Annotating delays for analog block in a mixed signal IP are the trickiest part in running SDF back annotated gate level simulations. Major reasons for delays inside analog models not getting annotated is use of zero delay behavioral model for the analog portion, which does not have information of delays. Another reason is simulator is not able to pick top level SDF file and annotate IOPATH delays at analog boundary as there is no specify block for each of the IO's at analog boundary. Without annotating the delays for the analog design, the interaction between the digital domain and analog domain does not happen correctly resulting in multiple timing failures at flops inside the netlist, causing x propagation while running the simulations.

Whenever there is the timing mismatch at a particular flop, there is x injection by the tool during simulations which helps us in knowing that there is a setup or hold violation in the design that needs to be fixed. The cause for these violations can be due to actual issue in closing the timing within the digital design or can be due to signals coming from analog design as the delays for the analog designs are not annotated since we use behavioral model for the analog designs and not the actual spice netlist.

The problem which we are trying to solve in this paper is second one where the signal interaction between digital and analog portion needs to be corrected by adding appropriate delays at analog boundary. This problem is important to resolve as end-to-end timing can be checked only if there is correct timing at analog portion as well. It is challenging problem as there is no standard way of dealing with this problem and traditional way to deal with this problem is by adding delays manually between analog and digital portion which is very tedious task to deal with. Fig. 2 shows the block diagram of a general mixed signal IP and where the exact problem lies.

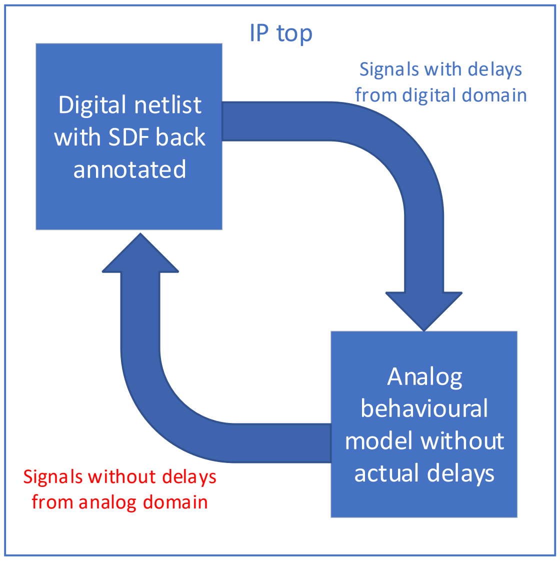

IP top
Digital netlist with SDF back annotated
Signals with delays from digital domain
Analog behavioural model without actual delays
Signals without delays from analog domain

Figure 2: Block diagram of Mixed signal IP

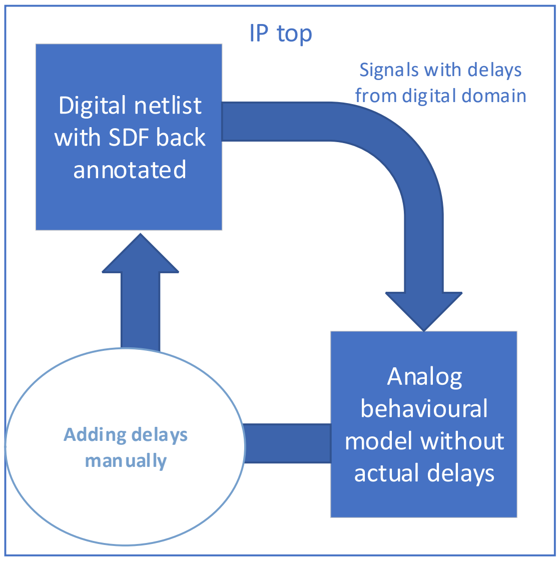

IP top
Digital netlist with SDF back annotated
Signals with delays from digital domain
Analog behavioural model without actual delays
Adding delays manually

Figure 3: Block diagram with manual delays.

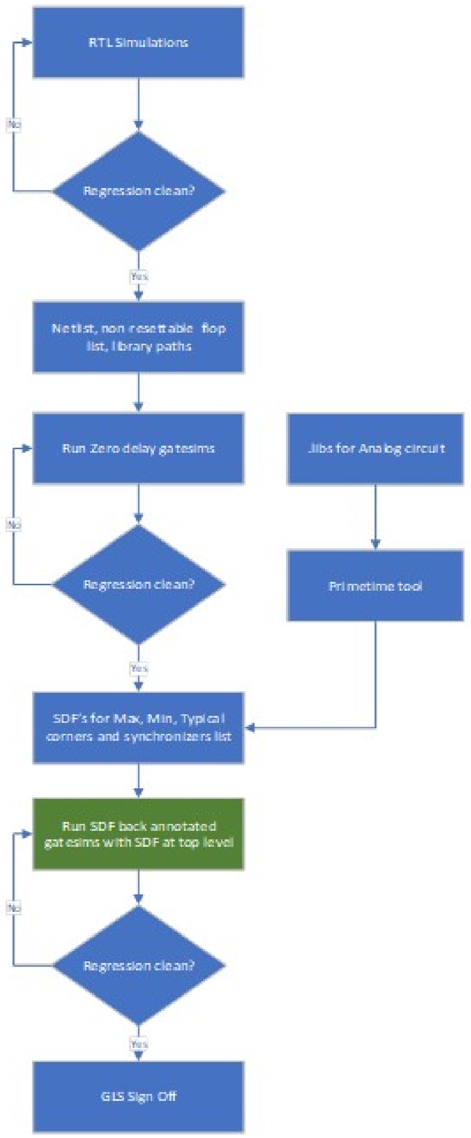
B. Previous Work

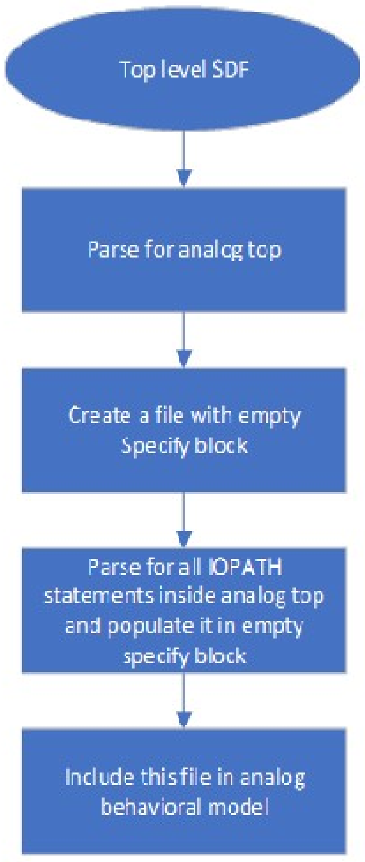
There is no standard way of solving this problem and what has been previously used is adding up the delays manually on the signals going between the analog and digital domain as shown in Fig. 3. This addition of manual delays, as previously been used, is furthermore very tedious and is more prone to mistakes.

III. PROPOSED SOLUTION

To deal with this problem of annotating the delays for the analog model, we have come up with a scheme which takes care of generating the specify block containing all the IOPATHs mentioned in the SDF shown in Fig. 4.

With the traditional approach, the simulator reports warnings that IOPATH delays can't be annotated as there is no specify block available. This generated specify block, once instantiated inside the design helps to resolve all the warnings. Also signals coming out of the analog model is delayed with respect to the inputs to the analog model. These signals then can be directly consumed in the digital domain.


RTL Simulations
No
Regression clean?
Yes
Netlist, non resettable flop list, library paths
Run Zero delay gatesims
Libs for Analog circuit
No
Regression clean?
Primetime tool
Yes
SDF's for Max, Min, Typical corners and synchronizers list
Run SDF back annotated gatesims with SDF at top level
No
Regression clean?
Yes
GLS Sign Off
Figure 4: Updated Gatesims Flow chart.


Top level SDF
Parse for analog top
Create a file with empty Specify block
Parse for all IOPATH statements inside analog top and populate it in empty specify block
Include this file in analog behavioral model
Figure 5: Proposed flow steps

For this activity we developed a flow shown in Fig. 5. The input to the flow is the top-level SDF which contains IOPATH delays for the signals going from the analog to the digital domain. Another input is the analog domain module name for which we need to generate the specify block. Once these two inputs are given to the flow, it searches for the module name inside the SDF file and generates the specify block for that module. This specify block contains all the signals for which the IOPATH delays exist in the SDF. Once this specify block has been generated, it must be instantiated inside the analog module for which specify block is intended.

Fig. 6 shows the block diagram of the design after running through this flow.

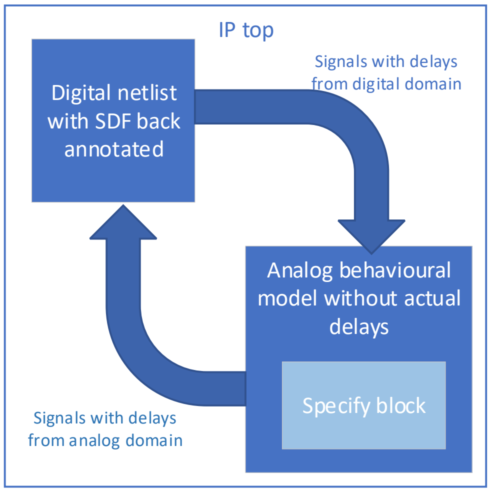
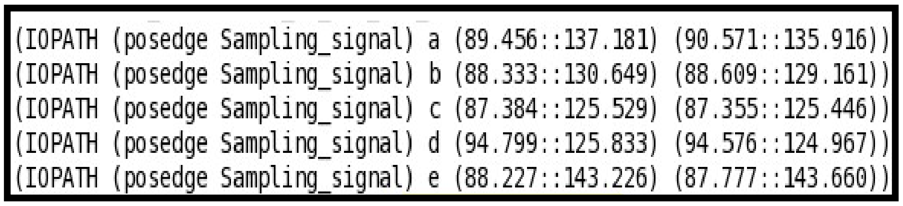

IP top
Digital netlist with SDF back annotated
Signals with delays from digital domain
Analog behavioural model without actual delays
Specify block
Signals with delays from analog domain

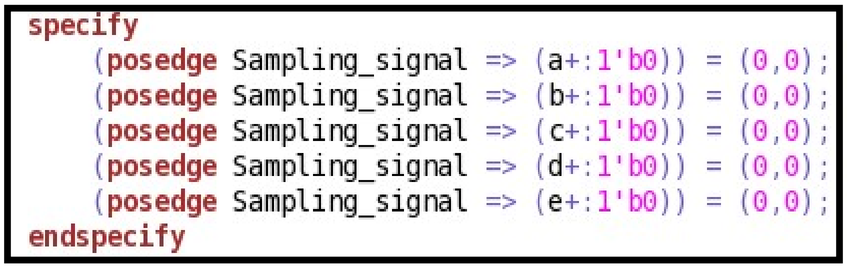
Figure 6: Block diagram with specify block instantiated.

Fig. 7 shows the SDF provided by the PD team and Fig. 8 shows the specific block which is the output by the flow that can be directly included.



```
(IOPATH (posedge Sampling_signal) a (89.456::137.181) (90.571::135.916))
(IOPATH (posedge Sampling_signal) b (88.333::130.649) (88.609::129.161))
(IOPATH (posedge Sampling_signal) c (87.384::125.529) (87.355::125.446))
(IOPATH (posedge Sampling_signal) d (94.799::125.833) (94.576::124.967))
(IOPATH (posedge Sampling_signal) e (88.227::143.226) (87.777::143.660))
```

Figure 7: SDF from PD team



```
specify
    (posedge Sampling_signal => (a+:1'b0)) = (0,0);
    (posedge Sampling_signal => (b+:1'b0)) = (0,0);
    (posedge Sampling_signal => (c+:1'b0)) = (0,0);
    (posedge Sampling_signal => (d+:1'b0)) = (0,0);
    (posedge Sampling_signal => (e+:1'b0)) = (0,0);
endspecify
```

Figure 8: Specify block generated by tool.

Fig. 9 shows the signal transitions when there is no specify block instantiated in the analog model. Fig. 10 shows the signal transitions post addition of the specify block. The difference between the two is that the second one is having some delays being added to the signals, which then meets the setup and hold timing at the flop inside the digital domain. This doesn't happen if the specify block is not present.

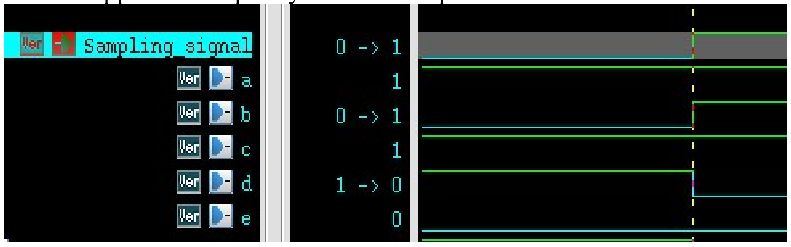


Figure 9: Signal transition without Specify block.

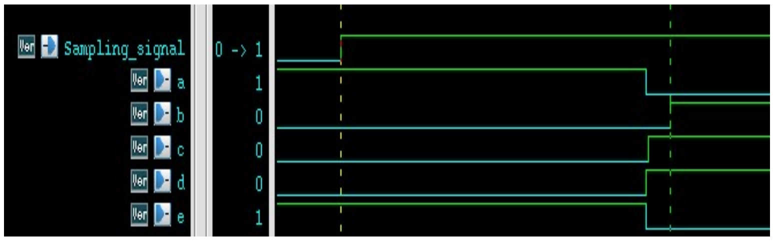


Figure 10: Signal transition with specify block.

IV. EXPERIMENTAL RESULTS

Fig. 11 shows the comparison of efforts needed for adding delays for the signals in the analog domain for 2 different IP's with ~40 signals at analog interface. Adding delays manually needs much more time as number of signals increases. With the proposed approach, adding delays is independent of how many signals are present in the design. As a result, the proposed approach is scalable. We compared 2 different designs and results shows that we need lot more time for the manual approach as compared to the proposed solution.

Although it takes few days to add the delays in the proposed solutions, still it takes significantly less effort than the manual approach. Along with turnaround time improvement, this proposed approach makes sure that all the delays are getting annotated and not just the few which are usually done in manual approach. The danger with not adding all the delays and just adding manual delays as and when the tests fail is missing out corner case bugs which might be present and getting masked due to delays not being added in the path. In one of the projects, we discovered an issue in the constraint after adding up the delays for all the paths which did not show up when delays were added manually.

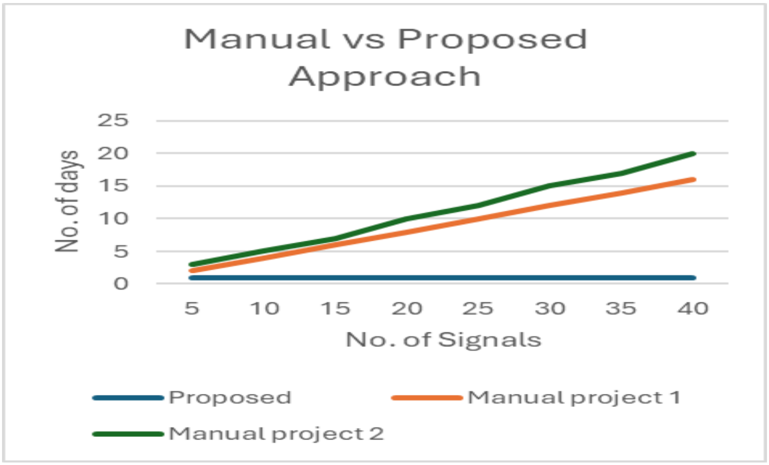


Figure 11: Comparison for the efforts in terms of days

V. CONCLUSION AND FUTURE WORK

To summarize, the proposed approach to add the delays automatically instead of manual addition is very efficient and easy to deploy resulting in better verification quality in much lesser time. This flow is scalable and is independent of any number of signals at the analog boundary.

While using this approach, we had to modify few checkers being written for analog domain signals. The reason is that the checkers did not take care of the delays when they were developed. So, as an enhancement, we would like to identify all the checkers present in the testbench for analog domain signals and enhance the flow to automatically tweak the checkers to take care of these delays.

ACKNOWLEDGMENT

The authors would like to acknowledge Mr. Ujjal Bose for his valuable suggestions in developing this methodology. The authors would also like to acknowledge our colleagues, Rahul PP, Robert Childs, Saket Doshi and Kakoli Bhattacharya for their contributions and suggestions during the development.

REFERENCES

[1] https://www.edn.com/gate-level-simulations-verification-flow-and-challenges.

[2] https://www.multimediadocs.com/assets/cadence_emea/documents/gatelevel_simulation_methodology.pdf

[3] Kai-Hui Chang, and Chris Browy, “Improving gate-level simulation accuracy when unknowns exist,” *DAC Design Automation Conference 2012*, San Francisco, CA, USA, 2012, pp. 936-940.