Journal of Advanced Research in Microelectronics and VLSI
Volume 1, Issue 2 - 2019, Pg. No. 1-3
Peer Reviewed Journal

Article

A Systematic Review on Hyperscale Implementation for Top Level Design

Neeti Agarwal

Physical Design Engineer at Eximius Design Pvt. Ltd., Bangalore, Karnataka, India.

I N F O

E-mail Id: neetidma@gmail.com

How to cite this article:
Agarwal NA. Systematic Review on Hyperscale Implementation for Top Level Design. *J Adv Res Microelec VLSI* 2019; 1(2): 1-35.

Date of Submission: 2019-12-28
Date of Acceptance: 2020-01-13

A B S T R A C T

With the growth in the VLSI industry, Complexity of design has increased both with respect to size and technology. Memory consumed and runtime impact is a major problem for the design especially when it comes to timing closure for the large chips. There have been many techniques to avoid such problems. This paper explains the various approaches in brief and explains in detail the hyperscale implementation approach. It also compares the HS implementation with flat run through a case study of GPU and bolster the adoption of the technique, addressing all the pros and cons.

Keywords: STA, CRPR, Derate, MIM

Introduction

Application specific integrated circuits deals with improvement in three key aspects namely Power , Area and Timing {Performance} .

A tradeoff is done among the three of the above stated parameters.

Timing is a very important factor, the issues can be addressed but the designs such as GPU whose size is big i.e has billions of instances takes a lot of time in generating the data for analysis which becomes a bottleneck in the timing closure and thus the entire design per se.

There are various methods which are used to generate timing data:

- **Flat run -** Netlist, parasitics, constraints are read for both block level and top level.
- **Hierarchical run -** Timing models are taken for block modules and run at toplevel.
- **Hyperscale implementation -** Context is characterised which includes interface logic abstraction, It is better than ETM and ILM. {ETM and ILM are type of Timing models }

Hyperscale methodology is a saviour when it comes to interim timing runs which reduces the runtime to almost half and helps in better analysis when compared to other timing models specific runs. The quality of the run is comparable to that of a flat run. This is checked by comparing slack at various nodes in both cases.

There are various Hyperscale Flow approaches in the industry

Top-down Hyperscale Approach

In this type of approach , a context is created from a top level session and blocks runs are done after reading the context. Later, block context is used to do a full -fledged Hyperscale run.

Bottom -Up Hyperscale Approach

In this no context creation happens and hyperscale models are used in the run.

HS Implementation Procedure

We have followed a top to bottom approach of hyperscale implementation in our case study.

- It involves the creation of toplevel sessions with the constraints , block sparasitics , verilog in consideration.
- The context for block level is written out through this session. This is called MIM {Multiple instantiated

module } extraction.

- MIMs are read in and the block level run is done and block level context is dumped through the block level session.
- The block level context is further read in and toplevel run is done , using MIMs . We need no other information for the same.

The HS implementation can be done in the existing setup easily. Procedure To Incorporate HS implementation in existing setup is below -"set_hier_config -enable_mim -partition" flow is for HyperScale Distributed analysis to control MIM partitioning which is getting changed in the 19.12 version.

We can use below flow to generate HS context for blocks from flat top-level analysis and read back context at block

Top-level:

- Variable/app setting
- **set_app_var hier_enable_analysis true**
  set_app_var hier_create_template_scripts true
- read_verilog
- link_design
- read upf, sdc, parastics, margins
- Characterize context for 2 MIM with instances which need to merge.

characterize_context -block MIM1 -instances {I1 I2 .. I17} -name {MIM1_I1_I17}
characterize_context -block MIM2 -instances {I18 I19 I20} -name {MIM2_I18_I19_I20}

- update_timing
- Write down Merged MIM context in binary format (gbc).

There are other user-readable format (ptsh) but that will miss some information (CRPR, AOCV depth) due to propriety.

write_context MIM1_I1 -output MIM1_I1_I17_context -format gbc

write_context MIM2_I17 -output MIM2_I17_I20_context -format gbc

MIM1 Block-level PT run

- Variable/app setting
- set_app_var hier_enable_analysis true
- read_verilog
- link_design
- read upf, sdc, parastics, margins
- read_context ./MIM1_I1_I17_context -name MIM1_I1_I17
- update_timing

Hyperscale Flow Merits When Compared To Hierarchical / Flat Run

Flat Vs Hyperscale

The memory and data required for timing run is really high for flat timing run. This leads to delay in timing closure.

Hyperscale reduces the run time by creating contexts of smaller modules instantiated at the top. It includes the creation of constraints, abstraction of interface logic and stub logic in case there is any . This results in decrease in runtime as full flat extraction is avoided using this approach.

Hierarchical Vs Hyperscale

The hierarchical run is used when we want less runtime. The library models for blocks are used in this approach. This therefore creates a problem as the modelling of the interface is not realistic . We model it in the form of input/output delays not knowing what actual logic is.

Hyperscale is better in this approach. It extracts the interface logic, thus catering to this requirement of a realistic timing analysis and the runtime also does not affect much in one level of logic getting extracted .

Hyperscale is thus a midway out. It helps in resolving both runtime, memory and realistic timing analysis related issues.

Hyperscale Flow Challenges

Wire Variation - When comparing Flat vs Hyperscale approach , It is found that when we consider the interface, the timing analysis varies a little especially at lower technology nodes ~5nm .There is a considerable change in the wire-via variation of the net when it goes from block to top level at the interface . This change in delay is because of slew rates changing because of wire variation .

GPD stitching - The flat approach reads parasitics in the form of spef, whereas in Hyperscale model , Galaxy parasitic Data (GPD) is read . GPD is automatic stitching of contexts in the design by tool. If a comparison is made with SPEF read , We need to make sure that no offset is provided while reading the parasitics. This would lead to change in physical location {Bounding box created by tool } and issue in related analysis.

Derate Changes - It is seen that distance derates changes in the clock network in the two runs i.e. flat and Hierarchical run .

Clock mapping - Top level clock needs to be mapped with respect to block level. This can be checked by report_clock -map. Failure with respect to this leads to wrong analysis .

Constraint mismatch/overwriting - If there are more than one similar contexts , constraints can be overwritten . This

leads to wrong constraints associated with the path which leads to faulting timing analysis .

Hyperscale Flow Solution To The Challenges

Wire-Via variation - Newe Version of tool had changes which solved the issue

GPD Stitching - For this issue , we never use offset in our run and allow the tool to stitch it . This leads to no discrepancies with respect to physical location calculation .

Derate Change - Derate changes are encountered but gets adjusted in CRPR calculation.

Clock Mapping - If there is any issue wherein source clock is not mapped wrt block level clock. Create_clock command is used to create it and map to block level clock.

Constraint mismatch/Overwriting - Here , we need to be cautious when similar context is created. All HS related warnings and Errors should be reviewed and fixed. Issues wrt to case value mismatch should be reported and fixed as the intent of architecture is.

Case Study

We took a GPU design and performed Flat run on the toplevel. Following is the study

On the setup corner, It was found that Hyperscale Implementation is more pessimistic when compared to flat implementation. The only differences were coming because the tech node was 5nm , so the differences were accounting for wire -via variation . Furthermore , with the newer version of the STA tool, These differences were solved.

Thus, Hyperscale implementation was good to go for intermediate runs. We can close our blocks without any worry .

On the hold side, we saw optimistic results on the Hyperscale Implementation than flat implementation . This was because of two reasons :

We saw that the net delay was high in Hyperscale implementation which resulted in more delay , thus making Hyperscale timing optimistic than flat implementation. This was a major problem but was solved using proper modeling at interface.

Other was via variation. It was solved with a new tool version change.

Thus, We could use this approach for hold timing as well.

Conclusion

Thus, we could use this Hyperscale Top down approach to help cater to the runtime and memory issues without compromising the quality of Timing closure.

Acknowledgement

I would like to acknowledge Pranav Murthy, who gave me an opportunity to work on this technology/ Method. Secondly, Arun Venkatesan, who helped me understand and supported me to write this paper. I would especially thank Abhishek Banthia , the CAD engineer who helped us understand the intricacies of the tool.

I am very thankful to these people who have helped me and considered me to work on cutting edge technologies.

References

1. Synopsys Solvnet page : https://solvnetplus.synopsys.com/s/
2. Synopsys explanation of HS implementation : https://solvnetplus.synopsys.com/s/article/PrimeTime-HyperScale-Training-1576148300228
3. Primetime HS training, Synopsys