Methodology for Logic Optimization Post Synthesis

Sarala Gumma,
Design Engineer
Intel Corporation
Bangalore, India.
sarala.gumma@intel.com
C2DG Core Group

Makandar, Syed M Md Zaid
Design Engineer
Intel Corporation
Bangalore, India.
syed.m.zaid.makandar@intel.com
C2DG Core Group

Neelam Maniar C
Design Engineer
Intel Corporation
Bangalore, India.
neelam.c.maniar@intel.com
C2DG Core Group

Kousik Debnath
Design Automation Manager
Intel Corporation
Bangalore, India.
kousik.debnath@intel.com
C2DG Core Group

Abstract - ***Timing miscorrelation between DC compiler & timing signoff tool ends up with many setup violations and leaves scope for further logic optimization. Unstable Floorplan & Timing constraints adds to the huge timing convergence effort. In this paper, we have explored the opportunity of logic optimization post synthesis freeze. A methodology for absorbing changes at the correct stage of PnR flow is formulated. An automated way was developed to find logic optimization candidates from the timing signoff tool and improve setup timing by optimizing them. Data showing TNS improvement by 25% was presented.***

Keywords **— Logic Optimization, Timing Convergence, High Speed Design.**

I. Introduction

Timing convergence of multimillion gate designs with aggressive frequency push is a huge challenge. Stringent project timelines to achieve fast time-to-market aggravate the situation. The conventional method for physical design includes RTL synthesis, Placement, Clock tree synthesis, detailed route, and sizing Optimization. Design Compiler, maps technology independent RTL to the given technology specific netlist. RTL, I/O timing and floorplan constraints are inputs to DC. The IC Compiler place and route system does placement, clock tree synthesis, routing and optimization for complex designs on the synthesized netlist.

Timing miscorrelation between the DC/ICC and timing signoff tool exists due to different extraction engines, fill methodologies, timing engines, etc. This leaves behind many residual setup paths. Unstable I/O timing and floorplan constraints require multiple loops of Synthesis and PnR during the project making timing convergence difficult. RTL changes during the course of the project require logical ECO and contribute new paths to TNS.

Additional manual timing convergence effort is required from designer despite having powerful tools like DC/ICC. Designer can control the priority of critical paths by altering cost function of DC/ICC through commands like set_path_margin, group_paths, etc. For optimizing interconnect delay, the designer can provide inputs to ICC like specifying priority nets, placement bounds, etc. Clock tree synthesis stage of ICC takes inputs from the user and adds clock buffer and performs clock tuning for converging critical setup violations. Lagrangian Relaxation (LR) [1] based optimization tools optimally size cells by considering the ratio of load and drive strength. Nevertheless, many setup violations are left encouraging to explore innovative methods.

This paper targets reduction in effort of timing convergence by automating identification and absorption of Logic Optimization opportunities. Instances of back to back connected 2 input static gates are more prevalent in the designs. Core library provides a rich catalog of stacked gates. Opportunities such as optimizing NAND2-NOR2-INV to NAND3 often go unnoticed. This results in saving one stage delay. There are also patterns saving 2 stage delays such as NAND2-NOR2-NAND2 to NAND4. Absorbing inversion is another overlooked opportunity, this involves propagation of inversion to either fan-in or fan-out logic cone. Propagation of inversion to the logic depth of one is easily accomplished by identifying patterns such as INV-XOR to XNOR. Novel quality-check rule called "Logic-Optimization" is introduced. This paper talks about Identifying & Absorbing logic optimizations in an automated fashion and defines a methodology for the same.

II. Pattterns

Reduction of the number of stages in a timing path saves delay. This can be done by merging two simple gates into a complex stacked gate (or) by absorbing inverters into preceding or succeeding gates.

PATTERNS	**Optimized PATTERN**
AND_AND, AND_NAND, NAND_OR, NAND_NOR_INV, NAND_NOR_XNOR, NAND_NOR_XOR, XNOR_NAND_NOR, XOR_NAND_NOR	NAND03
OR_OR, OR_NOR, NOR_AND, NOR_NAND_INV, NOR_NAND_XNOR, NOR_NAND_XOR, XNOR_NOR_NAND, XOR_NOR_NAND	NOR03
AND_OR, AND_NOR, NAND_AND, NAND_NAND_INV, NAND_NAND_XNOR, NAND_NAND_XOR, XNOR_NOR_NOR, XOR_NOR_NOR	AOI012
OR_AND, OR_NAND, NOR_OR, NOR_NOR_INV, NOR_NOR_XNOR, NOR_NOR_XOR, XNOR_NAND_NAND, XOR_NAND_NAND	OAI012

Table 1: Patterns saving one stage delay

PATTERN	Optimized PATTERNS
NAND_NOR_NAND	NAND04
NAND_NOR_NOR	AOI013
NOR_NAND_NAND	OAI013
NOR_NAND_NOR	NOR04

Table 2: Patterns saving two stage delay

Patterns having the scope of optimization such as AND-AND are searched in synthesized design and reported. For example here, AND_NAND indicates serially connected 2 input AND gate with 2 input NAND gate. This can be converted to 3 input NAND gate (NAND03).

PATTERNS	Optimized PATTERNS
AND_INV, OR_INV, AND_BUF, OR_BUF	NAND, NOR, NAND-INV, NOR-INV
XOR_BUF, XNOR_BUF	XNOR INV, XOR INV
INV_XOR, INV_XNOR, BUF_XOR, BUF_XNOR	XNOR, XOR, INV-XNOR, INV-XOR
AND_XOR, OR_XNOR	NAND XNOR, NOR XOR
AND_XNOR, OR_XOR	NAND XOR, NOR XNOR
[INV-PG-INV Latch] - BUF	[INV-PG Latch]- INV
BUF-[INV-PG-INV Latch]	INV-[PG-INV Latch]

Table 3: Patterns where we can absorb inversion

Absorbing inversion in preceding or succeeding stages results in reduction of number of stages. Based on observation that both XOR and XNOR gates take similar delays, any redundant inverters in their inputs can be removed. Inverters connected to latch also can be removed since Core library provides PassGate-INV, INV-PassGate type latches in addition to conventional INV-PassGate-INV latch.

III. IDENTIFICATION & ECO PATCH GENERATION

A. *Pattern Identification*

Complex designs generally have more than millions of gates. Finding all the possible patterns out of the millions of gates cannot be checked manually. A proper mechanism has been defined to find possible patterns discussed in section II for the optimization. Pattern definition requires to specify *pattern, topology, interface, constraint and action*.

Then we iterate over the design's nets using Search and Query command "if_enet" and print all nets for which patterns are obeyed. Algorithm is listed below

Algorithm 1 Identification of LogicOpt patterns

```
1: Pattern matching
2:     Define: Pattern → pattern name
3:     Define: Topology                          //Identify gates connected serially
       Cell gate1 Inputs: n1 Outputs: n2
       Cell gate2 Inputs: n2 Outputs: n3
4:     Define: Interface → {n1,n3}
5:     Define: Constraints
       gate1 elemtype == NAND2                   //State the pattern to be matched
       gate2 elemtype == NOR2
       Ensure:Fanout of n2 is 1                  //To avoid having to duplicate gate1 for other fanouts
6:     Define: Action
       Set property LogicOpt of net n2 to "True"
7: Reporting identified patterns
8:     Using Search and Query command if_enet, print all nets satisfying LogicOpt == True
```

Results by the developed utility are reported as shown in Table 4. More properties of the net such as hold margin, route length, etc. can be added to report. This report can be used to implement ECOs on selective nets manually to converge highly critical paths, instead one can absorb all ECOs automatically using ICC shell based script which will be discussed in the following section.

#NetName	Reason	Setup Slack	Is_Clk	Total_cap
module:n21	XNORGATE_BUF	0.0088	0	0.043193

Table 4: Utility Report of identified LogicOpt Patterns

One can also code ICC shell based scripts to identify the patterns for logic optimization right after logic synthesis.

B. *LogicOpt ECO generation*

ICC Shell-based script reads patterns produced by the utility and generates an ECO Patch. It places new complex gate (U12 in Figure 3) at average location of existing gates (U1, U2 in Figure 3), deletes existing gates and addresses connectivity. Drive strength of new cell is set to the drive strength of the last cell (U2 in Figure 3) in cells involved in optimization. Due to the reduction in the number of stages, preceding stages must be upsized and this will be taken care of by sizing tools.

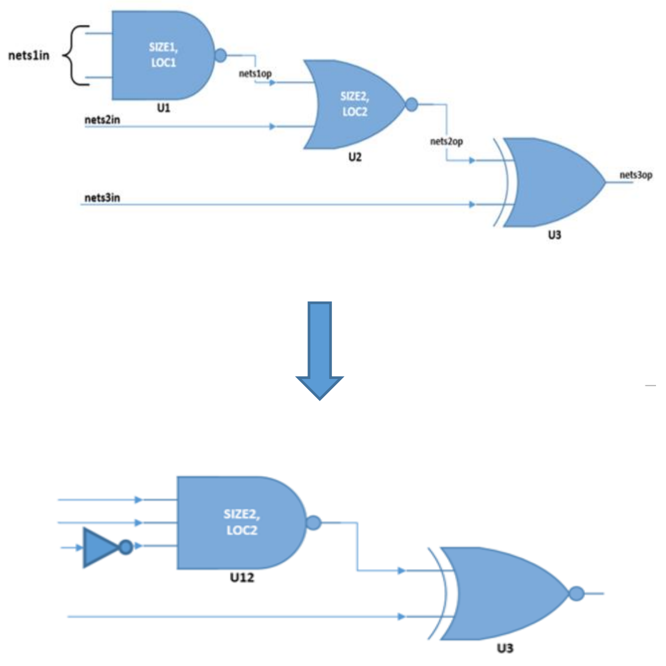

Figure 1: Example of Logic Optimization

Additional constraints are checked before generating ECOs for a given pattern

- Proximity: If gates involved (U1, U2 in Figure 1) in Optimization are far, it is better to have more stages to reduce interconnect delay and maintain signal integrity.
- Route length: Many nets will be touched during absorption of LogicOpt pattern. If a touched net is too

long, zroute might not do optimal routing, hence degrades the overall delay. One way to solve this issue is to avoid ECO generation, More will be discussed in the following section

- Margin: Script is setup and hold slack aware. It does not target logic optimization at the cost of potential hold violation.
- Fanin nets to the optimized complex gate where inverters will be added (nets2in in Figure 1) should have enough positive margin.

IV. METHODOLOGY

Methodology shown below in Figure 2 is formulated to get best gains out of logic optimization.

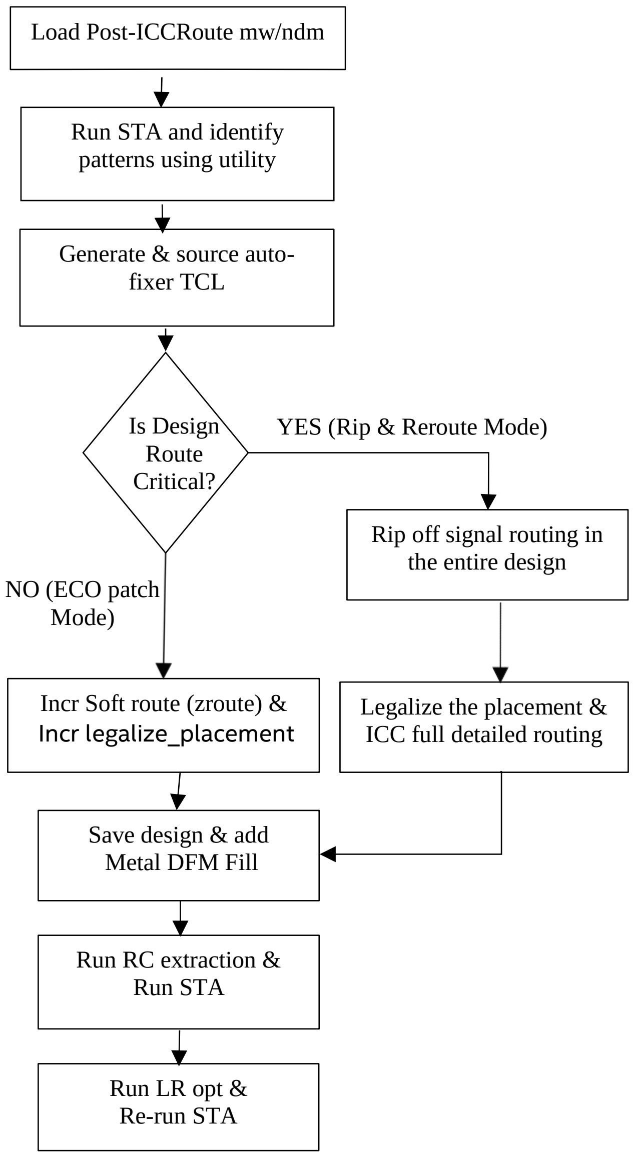
Load Post-ICCRoute mw/ndm
Run STA and identify patterns using utility
Generate & source auto-fixer TCL
Is Design Route Critical?
YES (Rip & Reroute Mode)
Rip off signal routing in the entire design
NO (ECO patch Mode)
Incr Soft route (zroute) & Incr legalize_placement
Legalize the placement & ICC full detailed routing
Save design & add Metal DFM Fill
Run RC extraction & Run STA
Run LR opt & Re-run STA

Figure 2. Flowchart defining Logic Optimization Utility

LogicOpt patterns are identified on an optimally sized place and route database. A report containing patterns gets dumped and this is used an input for the auto-fixer file generation. User can inspect the patch manually and absorb changes selectively for handful of patterns of their interest. For back annotating all probable LogicOpt patterns, based on route criticality of the design following modes are to be followed.

ECO mode: If design complexity is low and sufficient routing resources are available, LogicOpt patterns can be back annotated in ECO mode. After the back annotation, placement of the newly added cells is legalized. Opens, shorts & DRCs are cleaned with soft route, incremental search and repair. Due to reduction in number of stages, stages preceding optimized gates need to be upsized, this will be addressed with LR optimization flow.

Rip and Route mode: ECO patch of logical optimization requires cleanup of many opens, shorts & DRCs. In high complex designs, due to non-availability of routing resources, incremental ICC-route creates jogs and ends up choosing lower metal resources for nets connecting newly added gates. Hence back-annotation in ECO mode is not suitable. Back – annotation pre CTS, pre Route isn't recommended since clock latencies are ideal and path profile will change substantially. Post Route, to facilitate ICC with availability of all tracks, all signal routes are ripped off post back annotation of LogicOpt ECO patch. After signal route rip-off, design will be taken through all phases of ICC Route. After route stage, sizing based sizing optimization is applied.

V. RESULTS

As defined in Section II, patterns are categorized as Absorbing inversion, 1Stage gain & 2Stage gain. Table 5 shows No. of patterns identified in various designs per category. The No. of patterns in each design makes it evident that even after DC, there is scope for Logic Optimization. Cell count reduction is proportional to No of patterns found in design.

Design	Absorbing Inversion	1Stage Delay Gain	2 Stage Delay Gain
DUT1	148	82	34
DUT2	187	162	142
DUT3	96	117	19
DUT4	111	78	25
DUT5	103	271	162
DUT6	109	156	83

Table 5: Number of patterns per-category

Figure 3 shows significant left over in setup TNS in sign off tools after DC/ICC and LR optimization. The logic optimization utility discussed reduces average TNS across designs and voltage corners by approximately 25%. Average leakage reduction by 3% and Area reduction by 0.5% is observed across the designs.

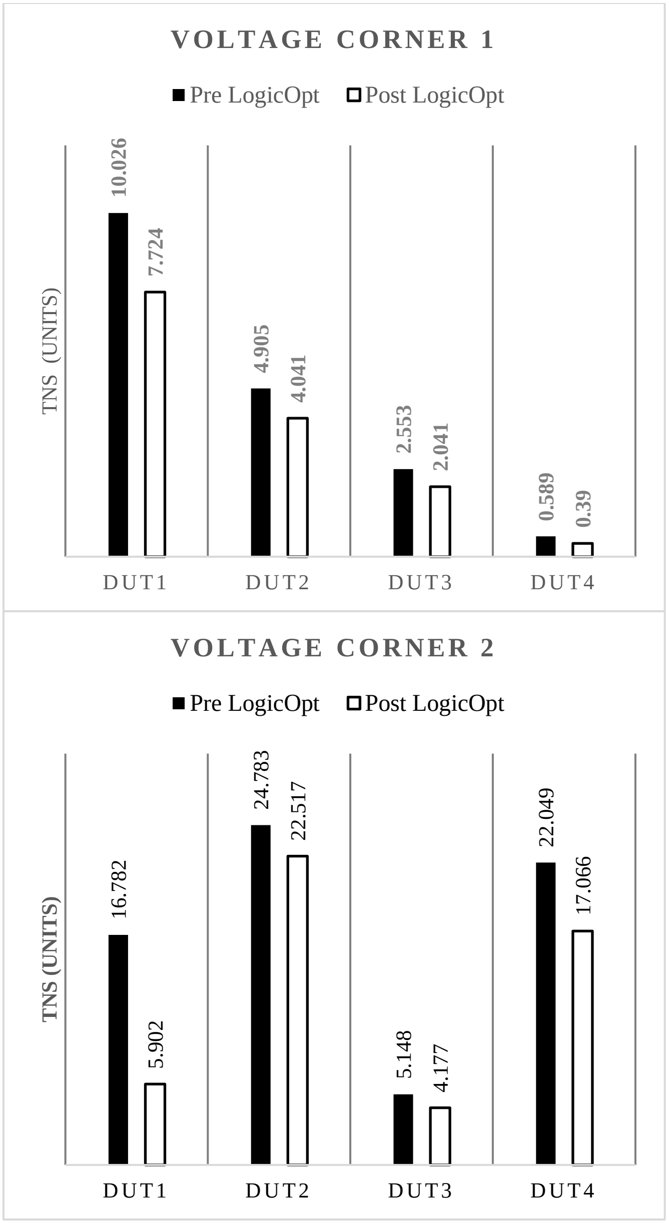

Figure 3: Results of TNS reduction seen in various designs

VI. Summary

Many LogicOpt patterns are left un-optimized by DC due to various reasons as discussed. This is evident from Table 5 which shows the number of patterns identified per-category in various designs. With correct flow & automation, these patterns can be back annotated for good TNS gains (25%). Since all possible patterns in design are reported, designer can also opt to fix patterns of their interest selectively.

As discussed in Section IV, fixing all patterns can result in timing degradations, if routing, sizing and timing criticality of other fanins is overlooked. Automation of ECO patch generation is made aware of these variables. As part of future work, more patterns will be identified and added. LogicOpt flow will be integrated to Synthesis and PnR Flow reducing effort of designer.

VII. Acknowledgments

The authors would like to acknowledge Vikas Gadi, Satish Sethuraman, Arpit Gandhi, Pratik Kamat, Eashwar Raghuraman, Arun Seetharaman, Akanksha Singh, Satish Cheekatla, and Ranganatha Gowda for their contributions to the methodologies, flows, enhancements, data generation and helping in make right decisions during the course of development of the algorithm.

VIII. References

1. Chung-Ping Chen, C.C Chu, D.F. Wong, "Fast exact simultaneous gate and wire sizing by Legrangian relaxation", ICCAD, 1998.
2. https://solvnet.synopsys.com