OpenPhySyn: An Open-Source Physical Synthesis Optimization Toolkit

Ahmed Agiza
Computer Science Department
Brown University
Providence, RI, 02912

Sherief Reda
School of Engineering
Brown University
Providence, RI, 02912

***Abstract*—Physical synthesis is a crucial phase in modern EDA due to the challenges in achieving timing closure. Many approaches have been presented to solve different timing and electrical violations, such as buffer insertion, gate sizing, pin swapping, gate cloning, and logic transformations; each approach has different overhead costs in terms of area, power, and run-time. This paper describes OpenPhySyn, a new open-source EDA kit that implements and enhances various physical synthesis and logical design algorithms to resolve design violations and perform timing closure. The tool integrates seamlessly with standard EDA flows and tackles different types of violations with minimal human interference and reduced area overhead. We evaluate OpenPhySyn on different industrial designs showing around 78% reduction in the area overhead when compared against existing open-source optimization flow while providing better solution quality.**

***Index Terms*—physical synthesis, optimization, buffering, resizing**

I. Introduction

Solving electrical and timing violations is a challenging problem in modern digital design. The problem complexity increases significantly with the shrinkage of technology nodes and the increase of interconnect delay [1] [2] [3]. Common approaches for solving design violations include buffer insertion, gate sizing, and pin swapping. Additionally, resolving the design violations entitles a significant increase in the design area and power due to the added or upsized cells.

Several approaches have been well-studied to find optimal buffer trees, such as the classical van Ginneken dynamic buffering approach [4] that uses a dynamic programming approach to find the optimal buffer tree for a given pin. Karandikar *et al.* explain a gate resizing approach to solve different types of electric violations [5]. Additionally, other techniques have been proposed to optimize the performance of the classical algorithms, such as the approach by Shi *et al.* that decreases the run-time of the van Ginneken buffering significantly [6].

OpenPhySyn [1] is an open-source toolkit that performs various physical and logic synthesis optimizations to solve design violations while minimizing the area overhead as a secondary objective. OpenPhySyn implements various optimization algorithms with novel enhancements to improve the design quality. OpenPhySyn utilizes modern open-source packages such as OpenSTA [7] for incremental timing analysis and OpenDB [8] for managing the loaded design. The tool also reads and writes using standard LEF/DEF format, facilitating the integration with different EDA flows. Moreover, OpenPhySyn is based on a flexible infrastructure that facilitates the implementation and integration of any optimization algorithm through a dynamic modular architecture, while providing a comprehensive utility library to speed up the development process and direct the developer's effort to the core logic of the optimization.

In this work, our contributions are summarized as follows:

- We describe OpenPhySyn, a novel physical synthesis optimzation tool. Our tool solves different design timing and electrical violation with minimal area overhead.
- We present an overview of the tool's main optimization commands, the tool exports various optimization commands that solve design violations with different techniques.
- We present the tool's optimization flows; the tool is able to composes multiple optimization techniques to solve design timing and electrical violations efficiently.
- We evaluate our tool against different EDA benchmarks and show the optimization results while comparing against the existing OpenROAD flow [9].

The rest of this paper is organized as follows. Section II presents the tool flow, Section III shows the experimental evaluation, and we conclude in Section IV.

II. OpenPhySyn Overview

A. OpenPhySyn Architecture

As shown in Fig 1, OpenPhySyn utilizes a modular architecture to facilitate extensibility and development. The main components of our tool are as follows:

- **Tcl interface:** the tool provides a scriptable Tcl interface for the end-user conforming to the EDA tools standards. The interface acts as the entry point for the users to load and process their design files. Additionally, the tool provides a broad set of optimization commands, with global optimization commands and more granular optimization commands for experienced users.
- **LEF/DEF readers and writers:** the tool provides the user with an interface to read and write design files. The interface uses OpenDB's Si2 LEF/DEF parsers internally to process the design into the database.

[1] https://github.com/scale-lab/OpenPhySyn

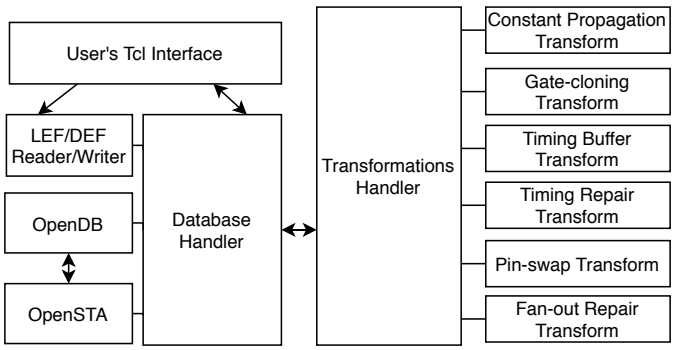

Fig. 1: Overview of OpenPhySyn Architecture.

- **Transforms Handler:** this component provides a modular interface for any optimization implemented in the tool. The handler has a dynamic interface to allow loading any optimization implementation in compile-time or runtime. OpenPhySyn uses its internal Transforms Handler to implement an essential set of optimization algorithms; the optimizations can be easily chained and evaluated.
- **Database Handler:** this module acts as the primary layer in OpenPhySyn's architecture. Firstly, it provides a generic interface to edit the design structure with access to different components. Secondly, it provides an interface to query the incremental timers or analyze various design violations giving the user the flexibility to interchange the design database or the incremental timer with any other third-party tools with minimal changes to the optimization algorithms. Thirdly, it provides a broad set of utilities and helpers to facilitate the development of any new algorithm. Finally, it encompasses a Boolean logic simulator that can extract and analyze any standard cell's logic functionality and perform Boolean simulations used in logic optimization. Fig. 2 summarizes the different modules provided by the database handler to provide the mentioned functionality:
  - **Design Utilities & Helper Algorithms:** The module provides a broad set of algorithms and utilities to interface with the loaded design and implementation for common useful algorithms. The provided built-in optimizations use the exported methods from the database handler, demonstrating the reuse of the shared infrastructure between different transforms
  - **Steiner Tree Heuristics:** The module provides methods to construct and use Steiner trees using the estimation heuristics from the FLUTE package [10]. The constructs provide different computation methods for the Steiner trees, such as the total wire length estimation and total capacitance/resistance for the given wire trees.
  - **Constraints & Violation Checkers:** The module provides a set of different flexible functionalities that can detect and highlight different types of violations in the design. The checkers are used by the built-in optimizations to extract the targeted violations to fix while supporting variable constraints provided by the user.
  - **Boolean Simulator:** The module provides a Boolean evaluator that uses the Boolean functions defined in the liberty file to simulate the design's logic functionality. The simulator can analyze the different functionality of each cell, extract the liberty cells for a given cell type (e.g., buffers, inverters, AND gates), or simulate any given input values to the cells.
  - **Timer (OpenSTA) and Parasitcis Engines Interfaces:** The module provides a timer and parasitics interfaces to interact with any provided time or parasitics extractor. The interface exposes several methods to perform any timing-based computations through OpenSTA internally, or query the parasitic information for different design components.
  - **Legalizer Interface:** The module provides an interface to integrate a placer to run incremental legalization throughout the optimizations. While OpenPhySyn is not equipped with a built-in legalizer, the optimization commands support the legalization interface allowing the user to easily plugin their placer to do incremental legalization throughout the optimization phases.
  - **Database (OpenDB) Interface:** The module provides an interface to the database infrastructure to load and manipulate the different design data structures. The interfaces provide access to different design blocks implemented by OpenDB without the need for internal knowledge of the database's complexities.

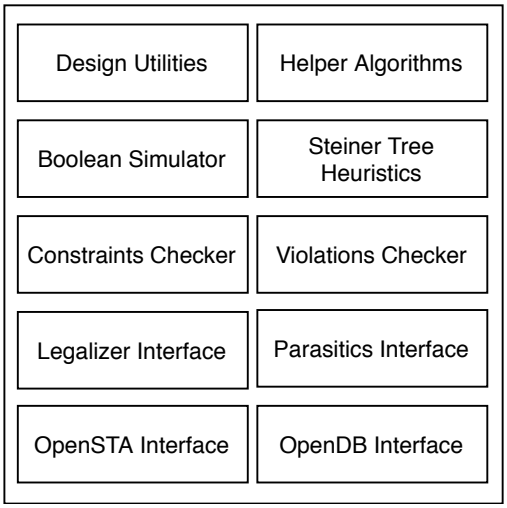

Fig. 2: OpenPhySyn Database Handler modules.

B. OpenPhySyn Optimization Transforms

OpenPhySyn aims at tackling design violations with minimal area overhead. Hence, the tool includes a standard set of optimizations that addresses different types of violations while minimizing the added area as a secondary objective. The

algorithms use several design optimization techniques such as buffer insertion, gate sizing, commutative pin swapping, gate cloning, and logic transformation. The tool exports commands for the following optimizations and utilities included by default in the tool's source code:

- **Buffer library selection**: this approach automatically selects a set of a representative buffer from the library with a given size using the buffer clustering technique by Alpert *et al.* [11]. The algorithm relies on clustering the buffers based on their intrinsic characteristics, resulting in a representative set of buffers to solve the optimization problems more time-efficiently. The selected buffers are used by further buffering algorithms or exported to be utilized by other optimization flows.
- **Timing-driven buffering**: this optimization performs fast buffer cell insertion [6] across the design to improve timing and solve violations.
- **Logic transformation**: this optimization performs logic transformation to replace different blocks from the designs with other logically-equivalent components to solve violations in a more area-efficient manner. Fig. 3 shows an example of the logic transformations performed to optimize the design while using the logic function to reduce the area overhead.

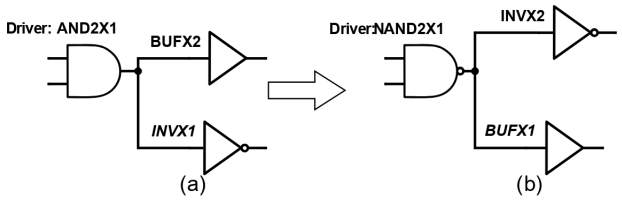

Fig. 3: Example of optimization of the block (a) through logic transformations to block (b) while keeping the logic functionality intact.

- **Fan-out repair**: this optimization performs a fast pass to break down large fan-out trees through buffer tree insertion to distribute the high fan-out and solve maximum fan-out violation.
- **Commutative pin swapping**: this optimization utilizes the Boolean simulator provided in the database handler to extract logically-commutative pins across the critical design paths followed by applying pin swapping to enhance the path delays without affecting the logic functionality of the circuit.
- **Constant propagation**: this optimization performs Depth-Fist-Search-based constant propagation across the design to eliminate redundant logic and save area. The optimization can be run after the logical synthesis, especially in hierarchical designs, to reduce the cell area by eliminating unnecessary cells.
- **Gate cloning**: this optimization performs load-driven gate cloning to reduce the load for cells driving high capacitance [3] [12].
- **Comprehensive timing repair**: this comprehensive optimization utilizes the previously described approaches: buffer selection, buffer insertion, gate sizing, pin swapping, gate cloning, and logic transformation through a composition strategy to solve electrical and timing violations efficiently. The optimization selects one or more techniques to solve each violation based on the state of the design. The command provides different control flags to customize the optimization's behavior as needed, such as controlling the pessimism level for optimization, specifying the minimum accepted gain, enabling or disabling different techniques, and run-time versus quality trade-off. Additionally, the optimization supports using a detailed placer interface to perform incremental legalization during the optimization if the user links any third-party placer.

C. Optimization Modes and Violation Types

OpenPhySyn optimizations tackle different types of violations, including electrical violation (caused by capacitance and transition limits), hold and setup violations (caused by path delays), and fan-out violations (caused by high fan-out pins). To accommodate for parasitics and estimation error, OpenPhySyn optimizations support pessimistic, ideal, and optimistic modes. This allows OpenPhySyn to meet tighter constraints, increasing the design tolerance for violations introduced in further flow stages. Additionally, the tool accommodates for more user-defined constraints such as the maximum utilization or custom dont-use cells.

D. OpenPhySyn Flow Integration

OpenPhySyn is designed and structured to integrate seamlessly with other EDA flow tools. As mentioned earlier, the tool provides flexible modules to interface with an incremental timer or structural design database. Additionally, the tool provides an interface to integrate placement tools to enable incremental legalization, which is supported with different modes by the provided optimization. Moreover, the tool supports different modes for wire parasitics estimation: providing average custom wire parasitics, automatic extraction for average wire parasitics from the LEF file, or interfacing with any external parasitics engine. By default, the wire length is estimated using Steiner trees [13] from FLUTE package [10] and used throughout different optimizations.

III. Evaluation

A. Setup

For the experimental evaluation, we selected six designs from different fields with different sizes and characteristics. We ran our tests using a commercial 65nm technoloy. We used Yosys [14] and ABC [15] for the logic synthesis, followed by OpenROAD [9] for floorplanning and placement. The parasitics were estimated from the third metal layer in the LEF file. We passed the placed designs to OpenPhySyn to perform the various optimizations to solve timing and electrical violations. Finally, we performed a legalization pass using OpenDP [9]. We evaluated the results in comparison with OpenROAD's optimization flow [9] using the same setup conditions. Table I

	Initial State of the Benchmark							OpenROAD Optimization Flow					OpenPhySyn Optimization Flow				
Design	**#INST**	**AREA (um^2)**	**CLK (ns)**	**#TRNS Viols.**	**#CAP Viols.**	**WNS (ns)**	**TNS (ns)**	**ΔAREA (%)**	**#TRNS Viols.**	**#CAP Viols.**	**WNS (ns)**	**TNS (ns)**	**ΔAREA (%)**	**#TRNS Viols.**	**#CAP Viols.**	**WNS (ns)**	**TNS (ns)**
gcd	294	1271	3.50	1	1	-1.44	-33.34	55.09	0	0	0.00	0.00	6.94	0	0	0.00	0.00
dn	8523	51943	7.50	153	157	-2.82	-16.56	44.99	0	0	0.00	0.00	0.61	0	0	0.00	0.00
ibex	22729	100106	23.00	361	313	-1.85	-53.08	78.52	1	1	0.00	0.00	0.53	0	0	0.00	0.00
tinyRocket	26300	122354	14.00	458	468	-1.51	-548.39	82.08	1	2	0.00	0.00	0.19	0	1	0.00	0.00
bp_be	42785	386850	20.00	3076	3032	-4.97	-91.76	89.37	480	494	0.00	0.00	8.47	0	1	0.00	0.00
jpeg	60440	262284	41.00	1763	1607	-3.90	-51.12	94.99	263	268	0.00	0.00	8.12	0	0	0.00	0.00
Average								74.17	124.17	127.50	0.00	0.00	4.14	0.00	0.33	0.00	0.00

TABLE I: Evaluation of the optimization flow on the selected benchmarks. The first section shows the names of the evaluated benchmarks, total number of cells, design area in square micrometers, clock period in nanoseconds, initial number of transition violations, initial number of capacitance violations, initial worst negative slack in nanoseconds, initial total negative slack in nanoseconds. The following section shows the percentage increase in design area, total number of transition violation, total number of capacitance violations, worst negative slack in nanoseconds, total negative slack in nanoseconds after optimizin the commands using OpenROAD. The last section shows the same metrics after optimizing the designs through OpenPhySyn.

gives the initial state of the chosen benchmarks after placement listing their respective number of cells, area in square micrometers, clock period in nanoseconds, initial number of transition violations, initial number of capacitance violations, initial worst negative slack in nanoseconds, initial total negative slack in nanoseconds, percentage increase in the design area, the number of transition violations, the number of capacitance violations, worst negative slack in nanoseconds, and total negative slack in nanoseconds when optimized by OpenROAD's flow compared against OpenPhySyn's optimization's flow. As shown from Table I, OpenPhySyn gives superior results where it solves most of the presented violations with an average 4% area overhead compared to OpenROAD's flow that adds an average 19% area overhead while failing to solve many of the presented violations.

IV. Conclusion and Future Direction

We presented OpenPhySyn, a novel open-source physical synthesis optimization kit that tackles modern EDA timing closure challenges. The tool utilizes a modular architecture and provides an infrastructure to facilitate the development of physical optimizations. OpenPhySyn is designed to work seamlessly within a full EDA flow by using standard input and output formats for the processed design. Moreover, OpenPhySyn includes and enhances many of the standard EDA optimization algorithms. Additionally, OpenPhySyn performs intelligent compositions for the packaged algorithms to optimize the loaded designs efficiently. We evaluated OpenPhySyn against different benchmarks and compared them against OpenROAD's optimization flow showing superior results in both the area overhead and the number of solved violations. Our future work is to extend OpenPhySyn to design and implement more EDA algorithms to solve more complex violations with better efficiency.

V. Acknowledgment

OpenPhySyn is available open-source in the public domain under the BSD-3-Clause License accessible at https://github.com/scale-lab/OpenPhySyn. The tool is tested on Linux and macOS platforms using different commercial libraries, including 65nm, 45nm, and 14nm technology nodes. The authors would like to acknowledge Prof. Andrew B. Kahng and Tom Spyrou from UCSD for their feedback on OpenPhySyn.

References

[1] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick, "Repeater scaling and its impact on cad," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 4, pp. 451–463, 2004.

[2] N. D. MacDonald, "Timing closure in deep submicron designs," *DAC Knowledge Center Article*, 2010.

[3] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *VLSI physical design: from graph partitioning to timing closure*. Springer Science & Business Media, 2011.

[4] L. P. Van Ginneken, "Buffer placement in distributed rc-tree networks for minimal elmore delay," in *IEEE International Symposium on Circuits and Systems*. IEEE, 1990, pp. 865–868.

[5] S. K. Karandikar, C. J. Alpert, M. C. Yildiz, P. Villarrubia, S. Quay, and T. Mahmud, "Fast electrical correction using resizing and buffering," in *2007 Asia and South Pacific Design Automation Conference*. IEEE, 2007, pp. 553–558.

[6] W. Shi and Z. Li, "A fast algorithm for optimal buffer insertion," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 6, pp. 879–891, 2005.

[7] "Opensta: Static timing analyzer, https://github.com/the-openroad-project/opensta."

[8] "Opendb: Database and tool framework for eda, https://github.com/the-openroad-project/opendb."

[9] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem *et al.*, "Toward an open-source digital flow: First learnings from the openroad project," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–4.

[10] C. Chu and Y.-C. Wong, "Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70–83, 2007.

[11] C. J. Alpert, R. G. Gandham, J. L. Neves, and S. T. Quay, "Buffer library selection," in *Proceedings 2000 International Conference on Computer Design*. IEEE, 2000, pp. 221–226.

[12] A. Srivastava, R. Kastner, C. Chen, and M. Sarrafzadeh, "Timing driven gate duplication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 1, pp. 42–51, 2004.

[13] T. Okamoto and J. Cong, "Buffered steiner tree construction with wire sizing for interconnect layout optimization," in *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*. IEEE Computer Society, 1997, pp. 44–49.

[14] C. Wolf, "Yosys open synthesis suite," 2016.

[15] A. Mishchenko, "Abc: A system for sequential synthesis and verification, berkley logic synthesis and verification group," *URL http://www. eecs. berkeley. edu/alanmi/abc/abc. html*, 2012.