Rsyn – An Extensible Physical Synthesis Framework

Guilherme Flach, Mateus Fogaça, Jucemar Monteiro,
Marcelo Johann and Ricardo Reis
Universidade Federal do Rio Grande do Sul (UFRGS) - Instituto de Informática - PGMicro/PPGC
{gaflach, mpfogaca, jucemar.monteiro, johann, reis}@inf.ufrgs.br

ABSTRACT

Due to the advanced stage of development on EDA science, it has been increasingly difficult to implement realistic software infrastructures in academia so that new problems and solutions are tested in a meaningful and consistent way. In this paper we present Rsyn, a free and open-source C++ framework for physical synthesis research and development comprising an elegant netlist data model, analysis tools (e.g. timing analysis, congestion), optimization methods (e.g. placement, sizing, buffering) and a graphical user interface. It is designed to be very modular and incrementally extensible. New components can be easily integrated making Rsyn increasingly valuable as a framework to leverage research in physical design. Standard and third party components can be mixed together via code or script language to create a comprehensive design flow, which can be used to better assess the quality of results of the research being conducted. The netlist data model uses the new features of C++11 providing a simple but efficient way to traverse and modify the netlist. Attributes can be seamlessly added to objects and a notification system alerts components about changes in the netlist. The flexibility of the netlist inspired the name Rsyn, which comes from the word resynthesis. Rsyn is created to allow researchers to focus on what is really important to their research spending less time on the infrastructure development. Allowing the sharing and reusability of common components is also one of the main contributions of the Rsyn framework. In this paper, the key concepts of Rsyn are presented. Examples of use are drawn, the important standard components (e.g. physical layer, timing) are detailed and some case studies based on recent Electronic Design Automation (EDA) contests are analyzed. Rsyn is available at **http://rsyn.design**.

Keywords

EDA, Physical Synthesis, Framework, Open Source

ISPD '17, March 19-22, 2017, Portland, OR, USA

DOI: http://dx.doi.org/10.1145/3036669.3038249

1. INTRODUCTION

A common challenge faced by academia in physical synthesis research is the lack of an open, collaborative and standard framework in which the work can be conducted. This may be circumvented by narrowing the scope of the research, which may lead to oversimplification, making it hard to assess the real benefits of a technique when inserted in a full design flow.

Some public physical synthesis tools do exist as for timing analysis [14], placement [42], routing [22], but they usually are developed to solve one very specific problem, and are hard to integrate in a cohesive way into a design flow. The underlying data structures are typically tightly tied to the optimization problem and difficult to extend. Even when the tool offers an Application Program Interface (API), there may be memory and runtime overheads due to the maintenance of redundant information (e.g. netlist). Moreover, the code experience gained in one tool is not directly translated to another one making the learning process very timing consuming.

Those drawbacks often push researchers to implement their own infrastructure as the time required to integrate or learn third party tools may not compensate the familiarity and efficiency of an infrastructure developed specifically to solve the research problem of interest. Therefore, researchers end up devoting a lot of time to infrastructure development and less time in the core of the research project. Considering that many physical design problems share common infrastructure requirements, it is clear that a lack of a common and powerful infrastructure hinders the advancement of this field.

In the last decade, renowned conferences such as International Symposium on Physical Design (ISPD), Design Automation Conference (DAC), International Conference on Computer-Aided Design (ICCAD) and International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU) have promoted research on emerging Electronic Design Automation (EDA) challenges [25, 39, 41, 30, 36, 18, 17] by organizing contests where teams compete to achieve the best results. The teams are composed by a small group of students who have few months to develop a tool [1] that solves the proposed problem. These contests usually boost research in the proposed field, which can be verified by the large number of related publications that follow the contests. Even though contests do a great job promoting research and creating standard ways to validate results,

[1]The tools are typically not required to be open-sourced.

there is still plenty room for more horizontal and vertical integration.

In this paper, an open-source, modular and extensible framework, called *Rsyn*, for physical synthesis research is presented. The elegant netlist data model and integrated tools allow fast prototyping of ideas. The modular design of the framework makes it possible to easily extend Rsyn for specific and general purposes. And the collaborative nature leverages it as an increasingly relevant and powerful framework for physical design research.

Rsyn provides support for common industrial file formats used in physical design, such as Library Exchange Format (LEF)/Design Exchange Format (DEF) [19], Liberty [28], Verilog [40] and the popular Bookshelf academic format [3]. Currently, Rsyn is already able to work on the benchmarks from the contests presented in Table 1.

Table 1: Current Contests Supported by Rsyn

Conference	Contest
ISPD 2005	Placement (Bookshelf)
ISPD 2006	Placement (Bookshelf)
ISPD 2012	Discrete Gate Sizing and Vt-Assignment
ISPD 2013	Discrete Gate Sizing and Vt-Assignment
ICCAD 2015	Incremental Timing-Driven Placement

The most relevant features currently provided by Rsyn infrastructure are summarized as follows:

- An elegant and dynamic netlist data model implemented using modern features of C++11.
- A notification system that alerts the modules in the framework when a change is made in the netlist;
- Extensibility of the netlist objects via user-specific attributes.
- Routing estimation and static timing analysis tools that can be configured to use different routing and timing models, including user-specific ones.
- An intuitive Graphical User Interface (GUI) with embedded features to help on analysing and debugging optimization techniques.
- Support for the widely-adopted industrial file formats, such as LEF/DEF [19].

The organization of the paper is as follows: Section 2 and Section 3 provide the description of the Rsyn architecture and the available standard components, respectively. In Section 4, we briefly summarize our recent research works using the Rsyn framework. In Section 5, the related works are shortly presented. Section 6 highlights the most relevant conclusions. The reader may refer to the official website [35] for more information, documentation and the source code.

2. ANATOMY OF RSYN FRAMEWORK

The core of Rsyn is composed of a netlist data model and three main components as shown in Figure 1: an engine, services, and processes.

The *netlist data model* stores structural and logical design data and can be extended via object attributes. It serves as a standard way to exchange and query design information in the framework. The *engine* is the main hub in the Rsyn framework. It manages and provides access to the design, processes and services keeping session information. It is also responsible for command registrations and dispatch.

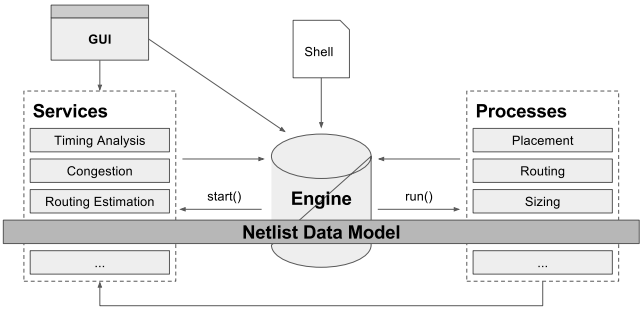

Figure 1: Rsyn Framework

2.1 Netlist Data Model

Rsyn implements a hierarchical [2] netlist data model which is managed by the object *design*. Figure 2 depicts the objects that compose the netlist data model. User-specific *attributes* can easily extend these objects and internally handle netlist modifications. A *notification system* is provided so that *observers* can be aware of changes in the netlist performed by third parties.

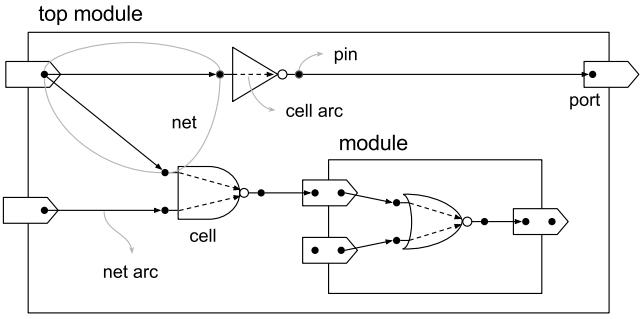

Figure 2: Netlist data model.

The netlist models a hierarchical directed graph where *pins* represent the nodes and *arcs* represent the edges. Arcs are classified into two types: cell and net arcs. Cell arcs represent the logical relation between the input and output pins of *cells*. Net arcs represent the logical relationship between the driver and sink of *nets*. Nets can only connect pins in the same hierarchy level [3]. The topological ordering of the nodes is incrementally updated whenever the netlist is changed allowing fast ordered traversal.

Hierarchies are represented by *modules* and connections between hierarchies by *ports*. *Instances* abstract modules, ports, and cells. The parent of an instance is the module where it is instantiated. The only instance with no parent is the top module.

All pins belong to one and only one instance and no instance has pins in different hierarchy levels. A port is associated to two pins: *inner pin* and *outer pin*. The inner pin

[2]Hierarchy support is still experimental.

[3]The concept of *supernets*, which will handle transparently sinks in different hierarchies, is yet to be implemented.

Listing 1: Rsyn Code Example

```
1 Rsyn::Attribute<Rsyn::Pin, int> data = design.
      createAttribute();
2 for (Rsyn::Net net : module.
      allNetsInTopologicalOrder()) {
3     for (Rsyn::Pin pin : net.allPins(Rsyn::SINK
          )) {
4         data[pin] = foo(net);
5     } // end for
6 } // end for
```

Listing 2: Rsyn Script Example

```
1 start "myService";
2 run "myOptimization1" {
3   "effort" : 1,
4   "debug" : "first pass"
5 };
6 run "myOptimization2" {
7   "effort" : 10,
8   "debug" : "second pass"
9 };
10 myReport "report.txt" -nets -cells;
```

belongs to the port itself and is inside the module where the port is instantiated. The outer pin belongs to the instance that represents the upper module in the hierarchy. Cells are associated with *library cells*. Similarly, cell pins and cell arcs are associated with the respective *library pins* and *library arcs* in the standard-cell library. This allows common data to be shared among cells, pins and arcs of the same type.

The snippet in Listing 1 shows the key concepts of the netlist data model. In the example, at line 1, an integer attribute for pins is created. At line 2 the nets are traversed in topological order. For each net, its sinks pins are swept at line 3. And at line 4 the attribute value is set.

2.2 Services and Processes

Services and *processes* provide the basic functionality required to implement a design flow. They are two related concepts that differentiate only by their lifespan. New services and processes created by users and collaborators will typically extend the Rsyn framework features. Services and processes are registered and managed via the *engine*.

A *service* usually is active during several flow steps implementing tasks that are recurrent and/or require a state to be kept. A typical usage of a service would be to create analysis tools (e.g. timing, power) and to implement the shared infrastructure among several flow steps (e.g. incremental legalization used by several optimization steps during detailed placement).

A *process* implements any task that does not require keeping its state after it is finished. They are normally used to implement a flow step. A typical usage of a process would be to create optimization steps (e.g. sizing, legalization, placement, routing). Even though these processes affect the state of the design, their internal data is not required anymore once they have finished. A typical process will rely on several services to perform its task.

2.3 Script

Currently, Rsyn implements its script language focused on simplicity. A script is a sequence of commands. Any command registered in the engine can be called via a Rsyn script. The command syntax mimics the GNU/Linux command line syntax. The parameters are divided into two types: positional and named. Positional parameters are assigned by their position and named parameters by their name. The command syntax is loosely defined as follows:

<command> [<value> ...] [-<param> <value> ...]

where <command> and <param> are any alfa-numeric identifier and <value> can be a number, string or JSON [23]. The JSON data type was chosen due to its flexibility and readability.

In Listing 2, a tiny Rsyn script is presented. Line 1 shows the call to start an Rsyn service while lines 2-5 and 6-9 show two calls to optimization flows and their user-defined parameters. Line 10 presents a call to a circuit report with some parameters.

2.4 Graphical User Interface (GUI)

A modular GUI is provided by the Rsyn framework to aid users to develop, improve, debug and better understand synthesis algorithms. It also has a great impact as an educational tool and in the engagement of students in EDA subjects. In Figure 3, a screen shot of the Rsyn's GUI is presented.

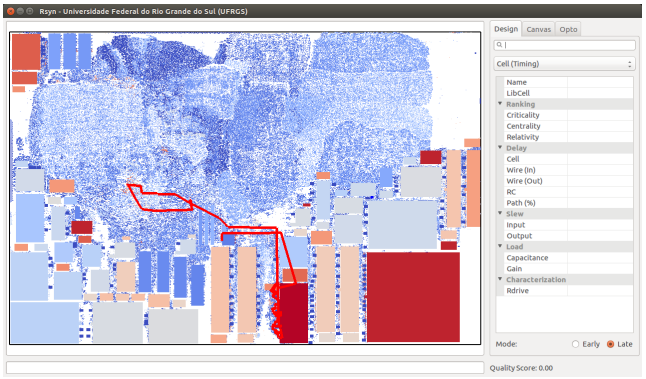

Figure 3: Graphical User Interface of the Rsyn framework showing Superblue18 circuit

The GUI is built using wxWidgets [45] and OpenGL [27] and can be compiled independently of the non-graphical functionalities. The GUI is extended via *overlays*, which can be loaded on demand to present visual data for a new feature (e.g. service).

The main overlay is the one responsible for presenting physical information of the design including standard-cells, macro-blocks, timing paths, and so forth. The circuit elements may be drawn and colorized by user-defined requirements and metrics, such as: criticality (e.g. slack), physical cell type (e.g. combinational, sequential), and so forth.

The property list on the right presents physical and timing information. These fields automatically show cell's data when it is selected. The command line shell location is at the bottom. There the user can issue any registered command in the engine.

2.5 Sandbox

Sandboxes allow creating a draft design that can be optimized independently of the main design. Once the optimization is finished, the changes can be committed. Sandboxes can be created from scratch or directly from a subset of elements of the design, as shown in Figure 4.

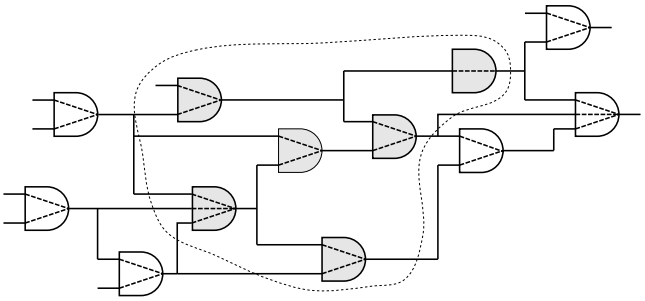

Figure 4: A sandbox extracted from the main design.

A sandbox provides similar API as a module from where instances can be created and connected via nets. However, it does not support hierarchy. Usually, sandbox represents a small design, and hence its implementation can take advantage of the reduced number of elements to improve efficiency.

3. STANDARD COMPONENTS

Section 2 highlighted the basic infrastructure to work and extend the Rsyn framework while this section presents several services, processes, and other standalone components already implemented. These are the parts that transform Rsyn from a simple netlist data model into a powerful framework to research physical synthesis and optimization methods.

3.1 Physical Design

The *physical design* comprises geometric information about the circuit layout and the technology, as illustrated in Figure 5. Physical design is responsible to aggregate and associate physical layout data to the attributes of the logic netlist. Moreover, the physical design also provides user-defined attributes mapped to the remaining layout elements (e.g. row, obstacle, die boundaries) and to the fabrication technology components. The notification system of the physical design alerts registered observers about third-party modification in the observed physical data. The hierarchical organization of physical design is inspired by the architecture of LEF and DEF formats.

3.2 Routing Estimation

The standard *routing estimation service* provides a way to estimate the physical interconnection among the pins of a net. Given a net, this service generates a Resistance–Capacitance (RC) tree as shown in Figure 6. Consequently, timing analysis, wire length estimation and congesting prediction may use the estimated tree.

Currently, the routing estimation service relies on a *routing estimation model* to generate the interconnection. This separation between the routing estimator and the routing model allows using different methodologies during a flow or

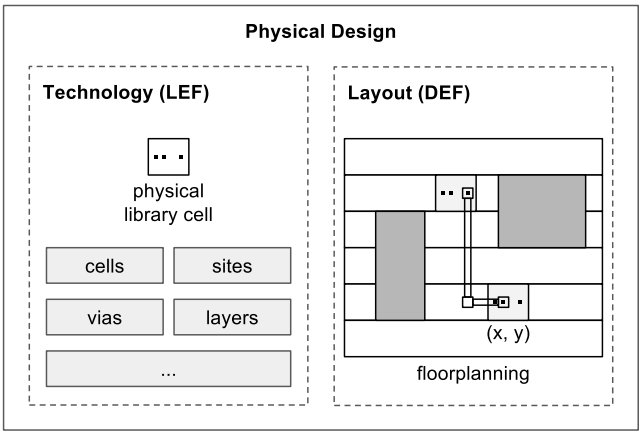

Figure 5: Schematic of the Physical Design Infrastructure. The physical design manages data of the circuit and the fabrication technology.

research project without requiring any additional changes to the service.

The default routing model integrated into Rsyn generates Steiner trees to estimate the interconnection. The Steiner tree is then translated to an RC tree using resistance and capacitance information from metal layers. The physical design service supplies the pin positions.

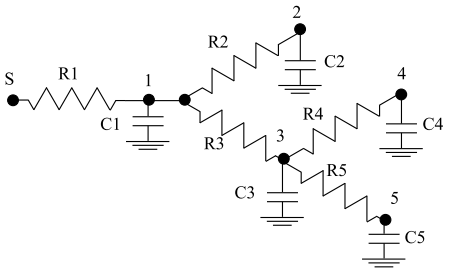

Figure 6: Routing Estimated RC tree.

3.3 Timing Analysis

Static Timing Analysis (STA) is essential to assert the performance of a design and to guide the optimization of a synthesis flow. It is extensively used in several steps and hence requires to be flexible and efficient.

The timer embedded in the Rsyn framework has the common functionality required by any STA tool, calculating the Total Negative Slack (TNS), Worst Negative Slack (WNS), arrival and required time at each pin and their slack, delay of the timing arcs and tracing the critical path. The timing propagation may be estimated for early, and late modes and the analysis is performed incrementally (i.e. only the elements that need the update are updated). Currently, the timer supports only one timing domain.

Similar to the routing estimation service, the timing analysis is independent of a *timing model*. Specific timing models can be used at different steps according to the availability of more accurate routing or sizing information for instance. Combined with the separation of routing and routing model estimation, it provides a flexible scheme for the algorithm

development avoid code rewriting. The high-level organization of the timer is outlined in Figure 7.

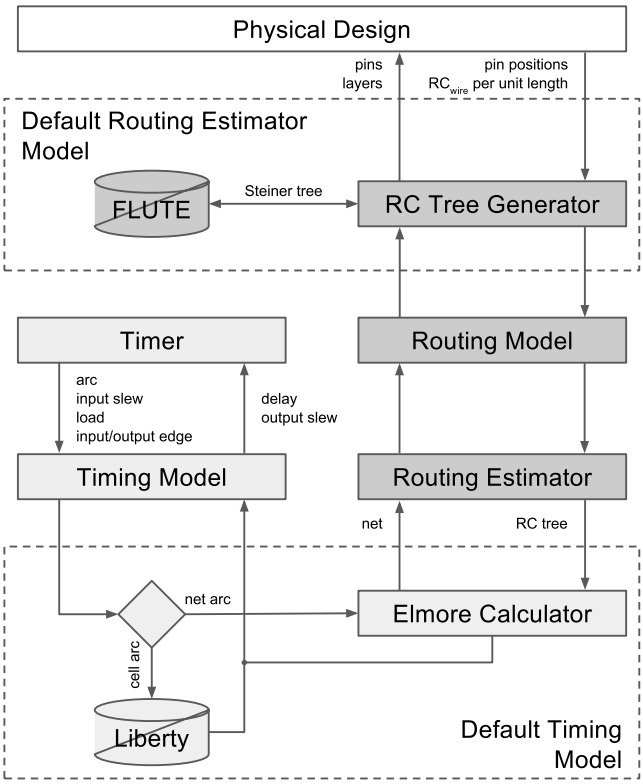

Figure 7: Timing analysis and routing estimator infrastructure.

Currently, the default timing model implemented in Rsyn is based on Elmore [6] delay following the guidelines presented in the ICCAD 15 contest. The default timing model uses the RC tree generated by the default routing estimator and cell characterization information obtained from a Liberty file.

3.4 Standard-Cell Library Characterization

Rsyn framework provides a simple *library characterization service* to compute the driver resistance and logical effort [38] of standard-cell timing arcs. Library characterization is performed by sampling the delay of the timing arc at different capacitance loads and estimating the delay via least squares regression as shown in Figure 8.

The drive resistance provides a linearized timing model, which is useful for analytical optimization algorithms. The drive resistance can also be used to sort cells by their drive strength and reduce the number of candidates for a gate-sizing algorithm, for instance.

3.5 Standard-Cell Legalization

Jezz legalizer [31] provides standard-cell legalization for minimum cell displacement based on Abacus [37]. It supports cell legality evaluation, to legalize cells in full and incremental modes, and cell legalization subject to maximum displacement. Jezz features are available in Rsyn framework via the service or process call systems.

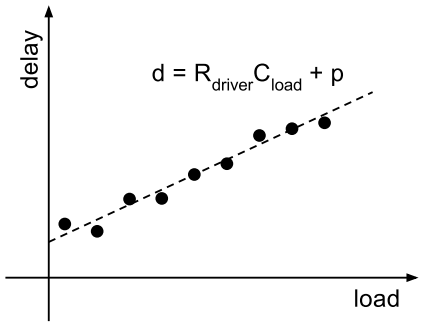

Figure 8: Driver Resistance Computation

3.6 Utilities

The utilities are features inside of Rsyn framework that provide shared resources to any element of the framework. The main idea of utilities is to abstract and encapsulate simple and very common required operations that are systematically performed inside of Rsyn components. A simple example is to compute the boundary area for the physical elements. There are several similar operations that are frequently used and are required in practically all the framework. The utilities may also be used to pass complex data, like boundaries definition, between independent Rsyn elements. The most relevant utilities already available in the framework are the following: colorize the design based on a user-defined metrics; customized step and stopwatches; command line parser; execution logger; float point comparator; definitions and operations for polygons and rectangles; and Cartesian point.

3.7 Third Party Software

Rsyn framework integrates a set of third party projects. They provide several features to the Rsyn framework. Below it is shortly described their main features. FLUTE [4] is an algorithm to build Steiner tree topologies that may be used to estimate routing. NCTUgr [22] is a global router tool. LEF and DEF [19] are parsers to recover data of the fabrication technology and circuit layout, respectively. Liberty [28] is a parser to recover timing and power information of the cell library from the Liberty files. JSON for Modern C++ [23] (Json) is a JavaScript Object Notation parser. Json scripts control the initialization of parameters and execution order for algorithms and flow. LEMON [20] is a graph library with a focus on combinational optimization mainly linked to graphs and networks. CPLEX [5] is a solver to mixed integer linear programming.

4. EXPERIMENTAL VALIDATION

Rsyn was created to fulfil the need to have a stable and versatile infrastructure upon which our research projects and contest tools could be developed. It was motivated to cope with our frustration of having to spend more time coding for infrastructure rather than optimization. Although our research group had been very successful in recent EDA contests, it was clear that the infrastructure developed for one contest was too tied to the contest problem and was not easily adapted to other contest or research projects.

So, after participating in the ICCAD 2014 Incremental Timing-Driven Placement Contest [18], we started to ag-

gregate and refactor several years worth of coding into the Rsyn framework. The goals were to create a platform where all physical design related projects could be developed and that would grow incrementally to form a full academic design flow. The framework should be intuitive to improve code readability and allow fast prototyping of ideas, ultimately freeing researchers to focus on the core of the research project.

In this section, we explore some usages of Rsyn framework in our research projects, which already show the benefits of having such framework.

4.1 ICCAD 2015 Contest

The 2015's edition of the ICCAD contest [17] served as an opportunity to consolidate the new framework while developing the desired algorithms. The development and enhancement of the Rsyn framework overlapped with the development of our research on incremental timing-driven placement, which turned out to be a win-win relation. While using Rsyn as the platform for the optimization algorithms, several bugs were found and fixed, and new features were implemented. It became easier to enhance our research projects while consolidating the Rsyn framework.

We conceived a total of 9 analytical techniques to mitigate both early and late timing violations which will be released as default optimization methods of Rsyn. The techniques to mitigate early timing violation rely on useful clock skew, iterative cell spreading, register swaps and register-to-register path fix. In late timing violation, techniques go through clustered movements (based on [2]) and single cell movements aiming to reduce the load capacitance in the critical nets, and balance driver load capacitance based on the cells drive strength. Furthermore, moving non-critical cells away from over-utilized regions minimizes area utilization overflow. The large number of different techniques implemented was directly related to the flexibility to try new ideas provided by Rsyn.

The techniques were integrated into the flow depicted in Fig. 9. A diamond shape indicates that the steps run until the quality of the result stops improving while the circle shape means that the quality of the result may degrade a certain number of times before exiting. The flow produces the best-known results for the ICCAD 2015 contest infrastructure, reducing on average the late WNS slack by 11% and the late TNS by 33% w.r.t. the initial placement solution. This Timing-driven placement flow completely removes early timing violation on 62.5% of the benchmarks. For more details about the techniques, please refer to [7].

4.2 Routing-Aware Incremental Timing-Driven Placement

We have also extended our previous work [7] to avoid cell movement towards to routing bins that have routing overflow violation. Therefore, this flow mitigates early and late timing violations aware of routing congestion regions. Moreover, Rsyn infrastructure was extended to support routing data and a third party router. The 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) paper [24] addresses the proposed Routing-Aware Incremental Timing-Driven Placement flow.

This project shows how Rsyn can be enhanced incrementally. Although the congestion infrastructure still needed to be developed, it is now part of the Rsyn framework being available to other projects.

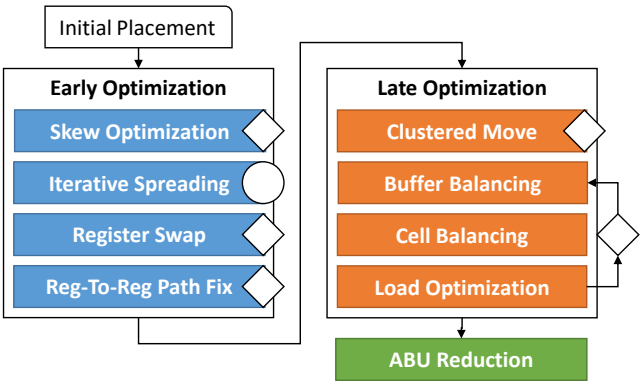

Figure 9: Our timing-driven flow

4.3 Incremental Timing-Driven Quadratic Placement

One drawback of the proposed flow in [7] is the absence of global timing-driven placement techniques. The clustered cell movement is the only technique that moves more than one cell at the same time, and yet the limit of cells per cluster and the search area are small. To cope with that, we proposed an incremental quadratic formulation for timing-driven placement.

Quadratic formulations have been explored for global wirelength-driven placement [42, 21] and timing-driven placement [43, 34]. However, the existing techniques are purely constructive, i.e., do not consider previous solutions, a behavior which is not desired during incremental placement. In [8] a method is proposed to apply quadratic placement incrementally, which is called neutralization.

Figure 10 presents the key ideas of such technique. The original netlist (Fig. 10(a)) is transformed into a graph using the methodologies presented in [42] (Fig. 10(b)). Neutralization forces are added to avoid a major disturbance in the initial solution (Fig. 10(c)) and the critical paths are identified (Fig. 10(d)). Then extra edges are added between nodes of the critical path (Fig. 10) making the quadratic placement align the critical paths. The weights of these edges are set to address Elmore delay [6], which is another contribution of that work. Applying this formulation jointly with the flow presented in Section 4.1 improves the results, on average, by 9.4% and 7.6% regarding WNS and TNS, respectively.

In this project, we already could take advantage of the infrastructure implemented inside Rsyn, finally being able to fully focus on the optimization process. Many analysis and experimentation tasks were done exploring much of the infrastructure provided by Rsyn, such as critical path tracing, legalization and visualization.

5. RELATED WORKS

One of the most traditional open source projects among EDA community is ABC [1] from Berkley University. ABC is a logic synthesis and verification environment on which users may rely on user-friendly and flexible data structures. Just like Rsyn, ABC aims to support different applications. The project is consolidated, providing implementations of

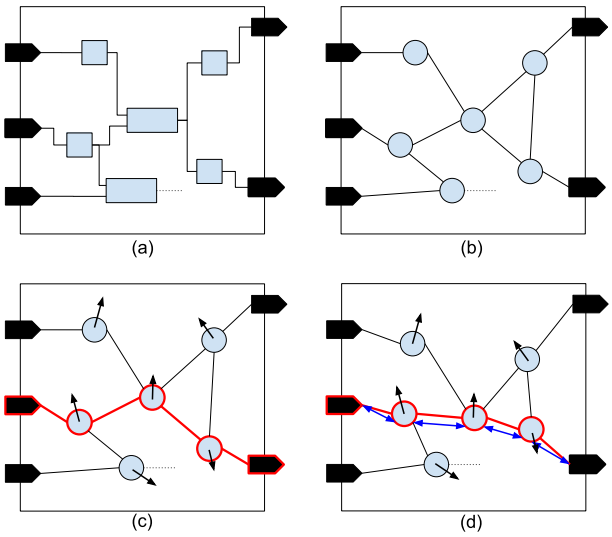

Figure 10: Incremental timing-driven quadratic placement strategy.The neutralization forces are drawn as the black single-edged arrows, the critical path as the bold red line and the additional forces as the double-edged blue arrows.

many state-of-art algorithms and support for many industrial and academic file formats [1].

Another open project on synthesis is Yosys [44], which translates Verilog code to an equivalent netlist, supporting both Application-Specific Integrated Circuit (ASIC) and Field-Programmable Gate Array (FPGA) flows. One of the Yosys goals is to make up for the lack of the extensibility of commercial synthesis tools. It features flexible data structures and tools for coarse grain synthesis. ABC is integrated into Yosys flow to perform logic minimization and technology mapping. Yosys is available on GitHub licensed under Internet Systems Consortium (ISC) license [16]. Once Rsyn and Yosys are both open sources and collaborative tools, a middleware could be build using both projects to provide logic and physical design.

OpenTimer [14] is an open-source tool for static timing analysis, winner of three awards in TAU contests [12, 11]. It implements a scheduler that aids to perform different tasks in parallel, achieving a runtime ten times smaller than other academic tools. Other features include Common Path Pessimism Removal (CPPR) and a fast incremental timing analysis. OpenTimer code is available on its website [13] under a General Public License (GPL) license [9]. Unfortunately, it is outside of any control version repository for the EDA community to share its contributions to the project. However, since OpenTimer is an accurate and fast academic tool, it may be integrated as part of Rsyn tools and repository and extended according to the community needs.

Parsing tools may compose the largest group of the available open projects. We highlight Icarus Verilog [15] for Hardware Description Language (HDL), Liberty Parser [28] for standard cell libraries and OpenAcess [26]. The OpenAccess is a C++ API that provides a solution for the wide range of complex file formats and syntax present today in the commercial design flow, such as LEF/DEF. Its source code is open, so the industry and academic community may propose extensions. By adopting an open and verified API, the EDA engineers may avoid coding their parsers, leading to fewer errors in the design flow.

Until now, we addressed only individual tools. Qflow [32] proposes an entirely open source design flow, from Verilog description to physical layout. It takes advantage of other academic tools, like Yosys [44] for Verilog parsing, Graywolf [10] for placement and Qrouter [33] for detailed routing. The authors claim that small commercial circuits were already designed using Qflow and highlight that small startups which cannot afford commercial tools may take advantage of the design flow.

Ophidian [29] is an open source project focused on research and teaching to physical synthesis. However, the project has only support to parsing few types of circuit files, netlist, and placement.

We believe there is a wide space for projects like ABC and Yosys on the physical design domain. Since nowadays there is no open framework addressing physical design where EDA developers may share their implementations. Rsyn fits this task and it goes further providing tools to report results (e.g. graphics) and a user interface to aid in the debugging process.

6. CONCLUSION

This paper presented Rsyn, an open-source framework that provides a versatile and modular infrastructure for physical synthesis research. Rsyn is designed to be extended incrementally and already contains several components that allow researchers to focus on the core of the research project rather than on the infrastructure to support it. Rsyn provides a standard and collaborative platform to share implementations, optimization techniques and code reuse.

With Rsyn, we intend to build a comprehensive, but intuitive physical design environment where the EDA community can develop and analyze new techniques. Rsyn also aims at getting more students interested in EDA research by allowing a better interaction with optimization algorithms.

Our current experience with the framework, already shows its potential to increase productivity. Rsyn allowed us to enhance our current results and explore different research topics much faster than what would be possible without it.

Acknowledgments

This work is partially supported by Brazilian Coordination for the Improvement of Higher Education Personnel (CAPES) and by the National Council for Scientific and Technological Development (CNPq).

7. REFERENCES

[1] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification. http://www.eecs.berkeley.edu/~alanmi/abc/.

[2] A. Bock, S. Held, N. Kammerling, and U. Schorr. Local search algorithms for timing-driven placement under arbitrary delay models. In *DAC*, pages 1–6, June 2015.

[3] Bookshelf. http://vlsicad.eecs.umich.edu/BK/ISPD06bench/BookshelfFormat.txt.

[4] C. Chu and Y. C. Wong. Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design. *TCAD*, 27(1):70–83, Jan 2008.

[5] Cplex. http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud/.

[6] W. C. Elmore. The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers. *Journal of Applied Physics*, 19(1):55–63, 1948.

[7] G. Flach, M. Fogaça, J. Monteiro, M. Johann, and R. Reis. Drive strength aware cell movement techniques for timing driven placement. In *ISPD*, 2016.

[8] M. Fogaça, G. Flach, J. Monteiro, M. Johann, and R. Reis. Quadratic timing objectives for incremental timing-driven placement optimization. In *ICECS*, 2016.

[9] Gnu general public license. https://www.gnu.org/licenses/gpl-3.0.en.html.

[10] Graywolf. https://github.com/rubund/graywolf.

[11] J. Hu, G. Schaeffer, and V. Garg. Tau 2015 contest on incremental timing analysis. In *ICCAD*, pages 882–889, Nov 2015.

[12] J. Hu, D. Sinha, and I. Keller. Tau 2014 contest on removing common path pessimism during timing analysis: Special session paper: Common path pessimism removal (cppr). In *ICCAD*, pages 591–591, Nov 2014.

[13] T.-W. Huang and M. D. F. Wong. Opentimer: An open-source high-performance timing analysis tool. https://web.engr.illinois.edu/~thuang19/software/timer/OpenTimer.html.

[14] T.-W. Huang and M. D. F. Wong. Opentimer: A high-performance timing analysis tool. In *ICCAD*, ICCAD '15, pages 895–902, Piscataway, NJ, USA, 2015. IEEE Press.

[15] Icarus verilog. http://iverilog.icarus.com/.

[16] Isc license (isc). https://opensource.org/licenses/ISC.

[17] M.-C. Kim, J. Hu, J. Li, and N. Viswanathan. Iccad-2015 cad contest in incremental timing-driven placement and benchmark suite. In *ICCAD*, ICCAD '15, pages 921–926, Piscataway, NJ, USA, 2015. IEEE Press.

[18] M.-C. Kim, J. Hu, and N. Viswanathan. Iccad-2014 cad contest in incremental timing-driven placement and benchmark suite. In *ICCAD*, ICCAD '14, pages 361–366, Piscataway, NJ, USA, 2014. IEEE Press.

[19] Lef/def. http://www.si2.org/. 2016-11-21.

[20] Library for efficient modeling and optimization in networks (lemon). https://lemon.cs.elte.hu/trac/lemon.

[21] T. Lin, C. Chu, J. R. Shinnerl, I. Bustany, and I. Nedelchev. POLAR: Placement based on novel rough legalization and refinement. In *ICCAD*, Nov 2013.

[22] W. H. Liu, W. C. Kao, Y. L. Li, and K. Y. Chao. Nctu-gr 2.0: Multithreaded collision-aware global routing with bounded-length maze routing. *TCAD*, 32(5):709–722, May 2013.

[23] N. Lohmann. Json. https://github.com/nlohmann/json. 2016-11-21.

[24] J. Monteiro, N. K. Darav, G. Flach, M. Fogaça, R. Reis, A. Kennings, M. Johann, and L. Behjat. Routing-aware incremental timing-driven placement. In *ISVLSI*, pages 290–295, July 2016.

[25] G.-J. Nam. Ispd 2006 placement contest: Benchmark suite and results. In *ISPD*, ISPD '06, pages 167–167, New York, NY, USA, 2006. ACM.

[26] Openaccess coalition. https://projects.si2.org/oac_index.php.

[27] Opengl. https://www.opengl.org/. 2016-11-21.

[28] Open source liberty. http://www.opensourceliberty.org/.

[29] Ophidian - open-source library for physical design research and teaching. https://github.com/eclufsc/ophidian.

[30] M. M. Ozdal, C. Amin, A. Ayupov, S. M. Burns, G. R. Wilke, and C. Zhuo. An improved benchmark suite for the ispd-2013 discrete cell sizing contest. In *ISPD*, ISPD '13, pages 168–170, New York, NY, USA, 2013. ACM.

[31] J. C. Puget, G. Flach, R. Reis, and M. Johann. Jezz: An effective legalization algorithm for minimum displacement. In *SBCCI*, pages 1–5, Aug 2015.

[32] Qflow. http://opencircuitdesign.com/qflow/.

[33] Qrouter. http://opencircuitdesign.com/qrouter/.

[34] B. M. Riess and G. G. Ettelt. SPEED: fast and efficient timing driven placement. In *ISCAS*, volume 1, 1995.

[35] Rsyn. http://rsyn.design.

[36] D. Sinha, L. Guerra e Silva, J. Wang, S. Raghunathan, D. Netrabile, and A. Shebaita. Tau 2013 variation aware timing analysis contest. In *ISPD*, ISPD '13, pages 171–178, New York, NY, USA, 2013. ACM.

[37] P. Spindler, U. Schlichtmann, and F. M. Johannes. Abacus: Fast legalization of standard cell circuits with minimal movement. In *ISPD*, ISPD '08, pages 47–53, New York, NY, USA, 2008. ACM.

[38] I. Sutherland, B. Sproull, and D. Harris. *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[39] C. N. Sze, P. Restle, G.-J. Nam, and C. Alpert. Ispd2009 clock network synthesis contest. In *ISPD*, ISPD '09, pages 149–150, New York, NY, USA, 2009. ACM.

[40] Ieee standard verilog hardware description language. *IEEE Std 1364-2001*, pages 01–856, 2001.

[41] N. Viswanathan, C. Alpert, C. Sze, Z. Li, and Y. Wei. Iccad-2012 cad contest in design hierarchy aware routability-driven placement and benchmark suite. In *ICCAD*, pages 345–348, Nov 2012.

[42] N. Viswanathan and C. C. N. Chu. FastPlace: efficient analytical placement using cell shifting, iterative local refinement,and a hybrid net model. *TCAD*, 24, 2005.

[43] N. Viswanathan, G.-J. Nam, J. A. Roy, Z. Li, C. J. Alpert, S. Ramji, and C. Chu. ITOP: Integrating Timing Optimization Within Placement. In *ISPD*. ACM, 2010.

[44] C. Wolf. Yosys open synthesis suite. http://www.clifford.at/yosys/.

[45] wxwidgets. http://www.wxwidgets.org/. 2016-11-21.