Rsyn – A Physical Synthesis Framework for Research and Education

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

***Abstract*—Typically academic physical synthesis algorithms are developed to optimize a particular feature of the circuits. Usually, they are strongly intertwined with the infrastructure and support only the required circuit data. This approach severely restricts integration and extensibility among physical synthesis algorithms. To tackle that problem, we developed the Rsyn framework. It is a modular and extensible platform which provides the required infrastructure for a wide-range of physical synthesis problems. Thus, new features may be easily integrated making Rsyn increasingly valuable as a framework to leverage research. Besides, standard and third party components can be mixed via code or script language to create a comprehensive design flow, which can be used to improve the research impact and results. Sharing and reusability of common components are also one of the main contributions of the Rsyn framework. The netlist data model uses the new features of C++11 providing an easy and efficient way to traverse and modify the netlist. User-defined attributes may be easily mapped to the netlist elements. A notification system alerts components about changes in the netlist elements. The flexibility of the implemented netlist inspired the name Rsyn, which comes from the word resynthesis. Our platform allows researchers and students to focus on the key concepts of the algorithms instead of spending a significant amount of time implementing the required infrastructure. Rsyn is also a valuable environment to teach and learn physical synthesis algorithms. We have successfully used Rsyn platform in several projects including the 2015 ICCAD and 2018 ISPD contests.**

Keywords*—*EDA, Physical Synthesis, Framework, Open Source

I. Introduction

Modern system-on-chips benefit from technology scaling by integrating several functional blocks composed by hundreds of millions transistors in a single die. This comes with the cost of affording an ever-increasing number of tool licenses, teams of engineers, enablements and servers [5]. Moreover, the design flows available in EDA vendor tools do not completely benefit from the technology scaling due to the known suboptimal-nature of the heuristics composing the underlying optimization engines. The use of the modern and powerful machine learning techniques has been explored in recent works targeting a higher flow predictability [5] and therefore reducing the number of design iterations and cost. A higher predictability also helps achieve better outcomes. This shows a brand-new scope of research possibilities but on the other hand raises the question: *"How can academic research groups conduct their research on modern industrial challenges with limit access to technology, such as PDKs and open-source libraries and frameworks?"*

Traditional conferences, such as ISPD [8][12][9][10], ICCAD [13][6][7] and TAU [11] [1] [2], hold programming contests for problems faced by the industry. Small teams of students are asked to propose solutions in a very tight schedule of few months. The contests are organized in collaboration with major industry companies, such as Synopsys, Cadence, Mentor, Xilinx, Intel and IBM. The industry provides a well-formed statement of the problem being addressed, testcases (benchmarks) and evaluation methodology. It has been observed an increase in the number of researches and publications on the contest-related topics in the years that follow the contests. However, the contest benchmarks usually suffer from obfuscations to hide the original design functionalities, clock structure, etc. These obfuscations can misguide the optimization and raise questions about the actual applicability of the research. More recently, a cooperation between ARM and the Arizona State University resulted in the open ASAP7 7nm predictive PDK [17], allowing academia to conduct research in a virtual 7nm environment. To develop optimization tools for either contests or predictive PDKs researchers still implement their own data structures, which takes huge portion of their time that could be spent on algorithm development. The data structures in these cases are commonly designed for performance and support only the information necessary for the optimization being developed. They lack extensibility and interoperability, which discourages the reuse of the code. We believe academia and industry can benefit from *open-source, collaborative and comprehensive* frameworks, data models and parsers[1].

ABC [16] is a traditional environment for logic synthesis that comprehends the qualities aforementioned. Many researchers develop their projects using ABC environment due to its comprehensive API and the many embedded algorithms and utilities. ABC is also known to be used in industrial tools. There are open-source physical design tools [34] [19] [32], but they do not share a common data model or API and their interoperability in these conditions is very likely to be implemented using *scripts*, which is an inefficient and restrictive solution. The physical design community lacks an environment such as ABC that provides a *common API and data model* and works as an *integration tool*.

We present *Rsyn*, an open-source framework for physical design research. Rsyn implements an *elegant* and *extensible* data model and a *intuitive* and *user friendly* API. We provide a set of ready-to-use utilities (e.g. routing prediction, parasitic extraction and timing analysis) for fast prototype of ideas. The framework provides a collaborative environment that can leverage integration of tools. For instance, an analytic placer and a timer can be integrated to implement a timing-driven placement. Rsyn is available for download on GitHub [26] under Apache License. We summarize the qualities of Rsyn as follows:

- An elegant and user-friendly data model and API implemented using the modern features of C++11;
- A robust callback system for netlist and layout-related events;
- Support for user-defined attributes (extensibility);
- Timing and parasitic extraction APIs;
- A powerful graphical user interface with support for user-defined overlays;

[1]Refer to Section II for a small review of open source EDA tools.

- Parsers for academic and industrial file formats, such as Bookshelf, Verilog, LEF/DEF, SDC and SPEF.

The remainder of this paper is organized as follows: Section II presents an overview of the existing open-source EDA tools and flows. Section III introduces the elements composing Rsyn framework and how developers can extend the framework with their own functionalities. Section IV discusses the standard components of Rsyn. Finally, Section V give conclusions and directions for future work.

II. Related Works

In EDA community, ABC [16] from Berkley University is one of the most successful and traditional open source projects. ABC is an environment for logic synthesis and verification. It provides a user-friendly and flexible data structure. ABC is a consolidated project which supports different applications. Several state-of-art algorithms are developed in the ABC environment. It supports several industrial and academic file formats.

Yosys [25] is an open source RTL synthesis tools. It synthesizes RTL to both ASIC or FPGA. Logic minimization and technology mapping are performed by ABC which is integrated in Yosys synthesis flow. Yosys is licensed under Internet Systems Consortium license [21]. The project is available at GitHub.

OpenTimer [3] is an open-source STA tool. OpenTimer has won three awards in TAU contests [1][2]. The tool has a scheduler to perfom parallel STA tasks. The scheduler aids to improve runtime up to ten times compared to other academic tools. OpenTimer has features to remove common path pessimism and a fast incremental timing analysis. It is an accurate and fast academic tool. OpenTimer may be integrated into Rsyn. Its source code is available at GitHub.

Parsing tools play a major role in optimization development once they define the interoperability among the optimization engines and commercial flows. Although Verilog [27] is the most common netlist exchange format, only a few open-source parsers exist. We highlight *Icarus Verilog* [20] and *Ben Marshall's Verilog parser* [15], both with limited support for the language syntax. For timing budgeting and constraints Synopsys provides an open-source SDC parser [36] and for physical synthesis Cadence provides LEF and DEF parsers [28]. OpenAcess [29] is a database that supports interoperability with commercial flows and includes a comprehensive set of parsers. However, the coalition responsible for maintening OpenAcess decided to restrict code the availability to a few paying members.

Qflow [31] is a free open-source RTL to GSDII flow. It comprises many underlying tools, which are also open-source. Qflow relies on Yosis to perform synthesizable Verilog to gate-level netlist synthesis. The back end of the flow is made with the classical annealing-based GrayWolf [19] placer and Qrouter [32] routing tool. Qflow enables the designs of circuits with relaxed constraints for those who cannot afford commercial flows. The authors claim the existence of successful tapeouts using Qflow. However, Qflow does not rely on any common infrastructure or database to run its tools and the trend is that boundaries among engines in EDA flows are becoming blurrier. The recent white-paper of *Synopsys Fusion Technology* [35] shows the importance of implementing flows with a single data model for logical and physical optimization and to exchange engines between synthesis, physical implementation and sign-off. We believe Rsyn can be the driver of such view in the open-source EDA.

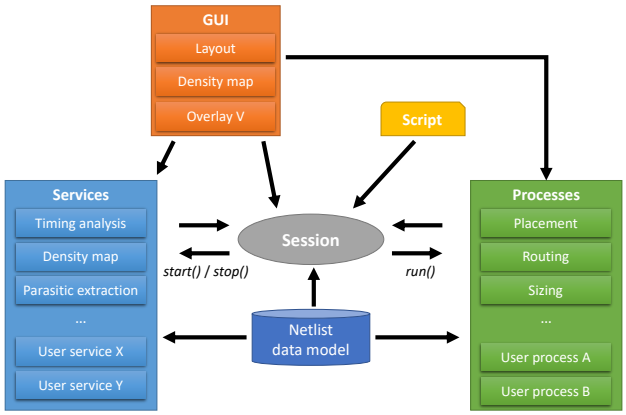

Fig. 1. Anatomy of Rsyn framework

III. Anatomy of Rsyn Framework

The Rsyn framework *comprises* a *session*, *data model*, *services*, *processes* and a *graphical user interface* (GUI). Figure 1 depicts an overview of the framework and how its elements are related. The session is the hub to use the Rsyn framework from which all the components of the framework can be accessed. The data model describes the design in its logical (netlist) and physical implementation (layout) and is the backbone of the framework on top of which services and processes are written. Services and processes both are used to perform analysis and optimization on the designs. In this section, we provide an overview of how the netlist is modeled in the framework data model. We also describe how tools can be implemented as services and processes in Rsyn. Finally, we present a powerful graphical user interface (GUI) that can be used to visualize the circuit layout and help on development of new algorithms.

A. Netlist Data Model

It is crucial for the framework to provide a high-quality modeling of the netlist once it holds the basic information to be consumed and modified by of optimization tools. Rsyn models the netlist as a direct graph in which pins are the nodes and arcs are the hyperedges, as depicted in Figure 2. There are two types of arcs: *net arcs* and *cell arcs*. Net arcs connect drivers to sinks of nets and cell arcs connect input pins to output pins of the cells. An object called design manages the netlist, which is divided into *modules*. Modules contain instances that can be *cells*, *ports* or other *modules*. Objects named *library cells*, *library pins* and *library arcs* store the information of cells, pins and arcs from a same implementation. Optimizations can define callbacks to cope with incremental modifications on the netlist. For instance, if a sizing tool implements its own circuit representation, it can define callbacks to update the main data model each time a cell has the size changed or threshold voltage is switched. This allows to keep the framework and possibly other tools, such as the timer, up-to-date. Another powerful feature our netlist data model is the extensibility. Users can define their own *attributes* for the objects to store additional information on-the-fly.

In Listing 1, a snippet code of Rsyn with user defined attribute is depicted. In Line 1, a new integer attribute called *data* for pins is created. All the nets of the circuit are traversed in topological order (Line 2). All sink pins of the net are transversed (Line 3). Finally, in Line 4, a method *foo* is called and the *foo*'s result is stored in the attribute's associated to pin.

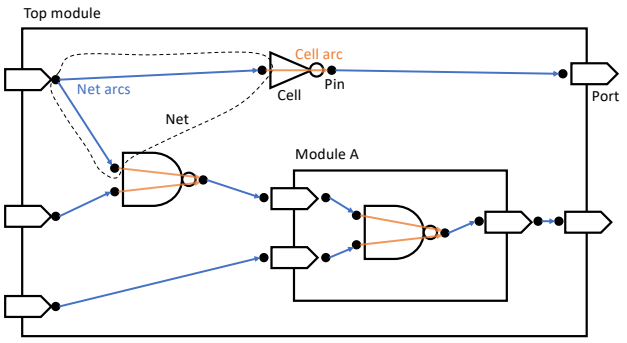

Fig. 2. Netlist data model.

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
```

B. Services and Processes

Rsyn is primarily designed to work as a *framework*, meaning users develop their optimization and analysis engines and integrate (*register*) them as new functionalities for the framework. In Rsyn, the users can register their engine as a *process* or a *service* in the session. *Processes* are engines that perform optimization, such as placement, clock-tree synthesis and routing implement well-defined algorithms that modify the netlist or the layout and will be called a limited number of times in the optimization flow. Every time a process is called, a new object is created and destroyed after its execution. Engines such as timing analysis tools and density maps are expected to be called several times inside other engines and their information updated incrementally. We calls these engines *services*. Once a service is *started*, it will remain available for other services and process until explicitly *stopped*.

C. Script

Once processes and services are registered in the framework they become available to be called using Rsyn commands. The commands in Rsyn follow the GNU/Linux syntax, defined as:

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

where `<command>` and `<parm>` are alphanumeric identifiers and `<value>` may assume string, numeric or Json [23] values. Rsyn scripts are composed of a set of commands that defines the user execution flow and do not present support for loops of functions to preserve the execution flow simple. The script uses Json syntax to transmite processes and services input parameters due to Json's readability and flexibility

In Listing 2, we present a snippet of Rsyn script. In Line 1, a service called *densityMap* is started with the parameters *numCols* equals to 50 and *numRows* equals to 60. Line 2-5 runs a process called *incrementalPlacement* with parameters *effort* equals to 5 and *target* equals to *density*. Line 6 calls

Listing 2. RSYN SCRIPT EXAMPLE

```
1 start "densityMap" {"numCols": 50, "numRows": 60};
2 run "incrementalPlacement" {
3   "effort" : 5,
4   "target" : "density"
5 };
6 reportDensityMap "report.txt" -verbose true;
```

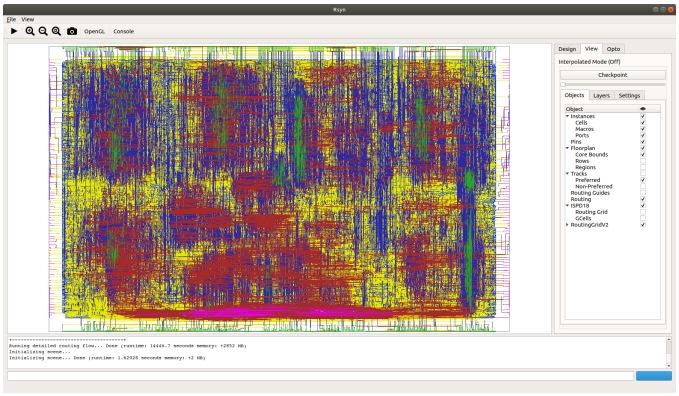

Fig. 3. Graphical User Interface of Rsyn

a command called *reportDensityMap* followed by string parameter *report.txt* and the boolean parameter *verbose* equals to true.

D. Graphical User Interface (GUI)

A GUI may have many practical applications. In Rsyn, our primary goal is to enable the visualization of the layout produced by optimization engines and therefore make easier to assert whether the algorithms are behaving as expected. However, we also provide tools to add new information in the GUI so algorithmic-specific information can be drawn (E.g. users can easily change the colors of the instances to represent the clusters produced by a clustering engine or draw arrows to depict the displacement of a legalization engine). We use Qt [33], which is a well-known cross-platform for development of desktop applications, as the backbone of Rsyn GUI. The many embedded features in Qt and its extensive online documentation makes of our GUI a powerful tool.

Figure 3 presents an overview of Rsyn GUI. Most of the window area is dedicated to the *canvas* where the layout and other information are drawn. The drawing is a task performed by a set of *overlays*, which makes our GUI *modular* and *extensible*. For example, the instances, rows, ports, pins, wires and vias are drawn by the *layout overlay* and users may implement their own overlays so new information can be added to the canvas. A menu in the right lists all overlays and allows to show and hide the drawing of a specific overlay. The users can rely on the top menu to find helpful features, such as *run script*, *zoom* and *snapshot* and in the bottom we provide an *interactive console* similar to the ones present in EDA commercial tools.

IV. STANDARD COMPONENTS

A. Physical Design

In Rsyn framework, Physical Design is the environment to store and manage layout data. It provides a user-friendly and intuitive API to access layout data of the physical objects. Physical design is composed of several physical elements to cope with specific layout data. Each physical design element

has its API. Physical Design handles physical elements defined in the technology library, the physical elements which are extension from logic elements, and physical objects which are present only in the layout design.

The technology library elements have the predefined layout which can be instantiated in design core. Therefore, physical synthesis algorithms can access and reference them while optimizing the design. The technology library elements provide guide and restrictions for the algorithms. For instance, routing algorithms can rely on library layers to compute paths to connect net's pins. The typical physical library elements are library cells, sites, vias, layers, and so forth.

The layout data of logic elements are stored and managed by physical objects. These objects are mapped to the respective logic objects. In the mapped physical objects, cell positions, ports/pins layout data, net topologies and so forth are stored and managed. In the Physical Design API, the reference to the logic objects is required to access the respective physical objects.

Several physical objects are defined to aid physical synthesis algorithms. They are only present in the physical design. These structures are regions, rows, routing grid, and so forth. For some of them, e.g., rows, an attribute API is available. Users can map with physical attribute their data to the physical objects. This map infrastructure is similar to the one present in the logic netlist.

In Physical Design, a notification system is provided. Users can register their functions to be called when the interested physical element is modified. Physical Design environment is developed in C++11 with the proxy design pattern. The environment is continuously extended to provide required features for new research projects. Recently, the detailed routing support has been added to the physical design. Therefore, Rsyn supports circuit with already routed nets. It also enables users to implement their detailed routing algorithms.

B. Parsers

The Rsyn framework has integrated several parsers to academic and industrial file formats. An intuitive and user-friendly API provides the mechanism to parse circuit files. Rsyn has integrated parsers for LEF/DEF, Verilog, Liberty, SDC and SPEF formats. Parsers for the bookshelf and some contest (e.g., ISPD 18) formats are also available. The required parsers are automatically called when the circuit load flows is executed.

C. Utilities

Rsyn utilities are a set of basic and shareable resources. The objective of utility elements is to abstract and encapsulate simple and common features and operations. The simple example is the rectangle shape. It can be used in cells, pins, core, and so forth. Basically, in all the circuit elements, the data and requires operations related to the element's boundaries are similar. Therefore, the common operations are encapsulated in an element which can be used in several distinct places. The utilities also may be used as method's attributes to pass complex data to functions. In Rsyn, the most used utilities are customized runtime profile watches, execution logger, float pointer comparators, polygons, rectangles, Cartesian points, and so forth.

V. Conclusion

In this paper, the Rsyn framework is presented. Rsyn is an open-source, versatile, extensible and modular physical synthesis platform. Rsyn provides the required infrastructure to researchers implement their optimization algorithms. The infrastructure is composed by academic and industrial parsers, netlist graph, timer, routing estimator, Graphics User Interface, and so forth. Researchers can dedicate their time to develop optimization algorithm instead to implement the required infrastructure. Moreover, they can use third party algorithms to provide some features for their projects. These algorithms can be easily integrated in Rsyn by using service or process features. The platform minimizes the fragmentation and improves the integration of physical synthesis algorithms. Rsyn platform is a friendly environment to teach, learnand research physical synthesis algorithms. We have successfully used Rsyn platform in several projects including the 2015 ICCAD and 2018 ISPD contests.

Acknowledgment

This work is partially supported by Brazilian Coordination for the Improvement of Higher Education Personnel (CAPES) - Finance Code 001 and by the National Council for Scientific and Technological Development (CNPq).

References

[1] J. Hu, D. Sinha and I. Keller, "TAU 2014 contest on removing common path pessimism during timing analysis", *Proc. TAU*, 2014, pp. 153–160.
[2] J. Hu, G. Schaeffer and V. Garg, "TAU 2015 Contest on Incremental Timing Analysis: Incremental Timing and CPPR Analysis", *Proc. TAU*, 2015, pp. 882–889.
[3] T.-W. Huang and M. D. F. Wong, "OpenTimer: A High-Performance Timing Analysis Tool", *Proc. ICCAD*, 2015, pp. 895–902.
[4] 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", *IEEE Trans. CAD* 32(5) (2013), pp. 709–722.
[5] A. B. Kahng, "Reducing Time and Effort in IC Implementation: A Roadmap of Challenges and Solutions", *Proc. DAC*, 2018, pp. 36:1-36:6.
[6] M.-C. Kim, J. Hu and N. Viswanathan, "ICCAD-2014 CAD Contest in Incremental Timing-driven Placement and Benchmark Suite", *Proc. ICCAD*, 2014. pp. 361–366.
[7] M.-C. Kim, J. Hu, J. Li, and N. Viswanathan, "ICCAD-2015 CAD Contest in Incremental Timing-driven Placement and Benchmark Suite", *Proc. ICCAD*, 2015, pp. 921–926.
[8] G.-J. Nam, "The ISPD2005 Placement Contest and Benchmark Suite", *Proc. ISPD*, 2005, pp. 216–220.
[9] M. M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke and C. Zhuo, "The ISPD-2012 Discrete Cell Sizing Contest and Benchmark Suite", *Proc. ISPD*, 2012, pp. 161–164.
[10] M. M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke and C. Zhuo, "An Improved Benchmark Suite for the ISPD-2013 Discrete Cell Sizing Contest", *Proc. ISPD*, 2013, pp. 168–170.
[11] D. sinha, L. G. e Silva, J. Wang, S. Raghunathan, D. Netrabile and A. Shebaita, "TAU 2013 variation aware timing analysis contest", *Proc. TAU*, 2013, pp. 171–178.
[12] C. N. Sze, P. Restle, G.-J. Nam and C. Alpert, "ISPD2009 Clock Network Synthesis Contest", *Proc. ISPD*, 2009, pp. 149–150.
[13] 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", *Proc. ICCAD*, 2012, pp. 345–348.
[14] N. Viswanathan and C. C. N. Chu, "FastPlace: efficient analytical placement using cell shifting, iterative local refinement,and a hybrid net model", *IEEE Trans. CAD* 24 (2005), pp 722–733.
[15] A Flex/Bison Parser for the IEEE 1364-2001 Verilog Standard. http://github.com/ben-marshall/verilog-parser
[16] ABC: A System for Sequential Synthesis and Verification. http://www.eecs.berkeley.edu/~alanmi/abc/
[17] ASAP: Arizona State Predictive PDK. http://asap.asu.edu/asap/
[18] Bookshelf. http://vlsicad.eecs.umich.edu/BK/ISPD06bench/BookshelfFormat.txt
[19] Graywolf. http://github.com/rubund/graywolf
[20] Icarus Verilog. http://iverilog.icarus.com/
[21] ISC License. https://opensource.org/licenses/ISC
[22] *International Technology Roadmap for Semiconductors*. http://www.itrs2.net/itrs-reports.html
[23] Json. https://github.com/nlohmann/json
[24] LEF/DEF. http://www.si2.org/
[25] Yosys Open SYnthesis Suite. http://www.clifford.at/yosys/
[26] Rsyn. http://rsyn.design
[27] IEEE Standard Verilog Hardware Description Language. https://ieeexplore.ieee.org/document/954909/
[28] LEF/DEF. http://www.si2.org/
[29] OpenAccess. http://projects.si2.org/oac_index.php
[30] Open Source Liberty. http://www.opensourceliberty.org/
[31] Qflow. http://opencircuitdesign.com/qflow/
[32] Qrouter. http://opencircuitdesign.com/qrouter/
[33] Qt. https://www.qt.io/
[34] RePlAce. https://github.com/abk-openroad/RePlAce
[35] "Synopsys Vision for the New Wave of Chip Design", https://www.synopsys.com/implementation-and-signoff/fusion-technology.html
[36] TAP-in. http://www.synopsys.com/community/interoperability-programs/tap-in.html