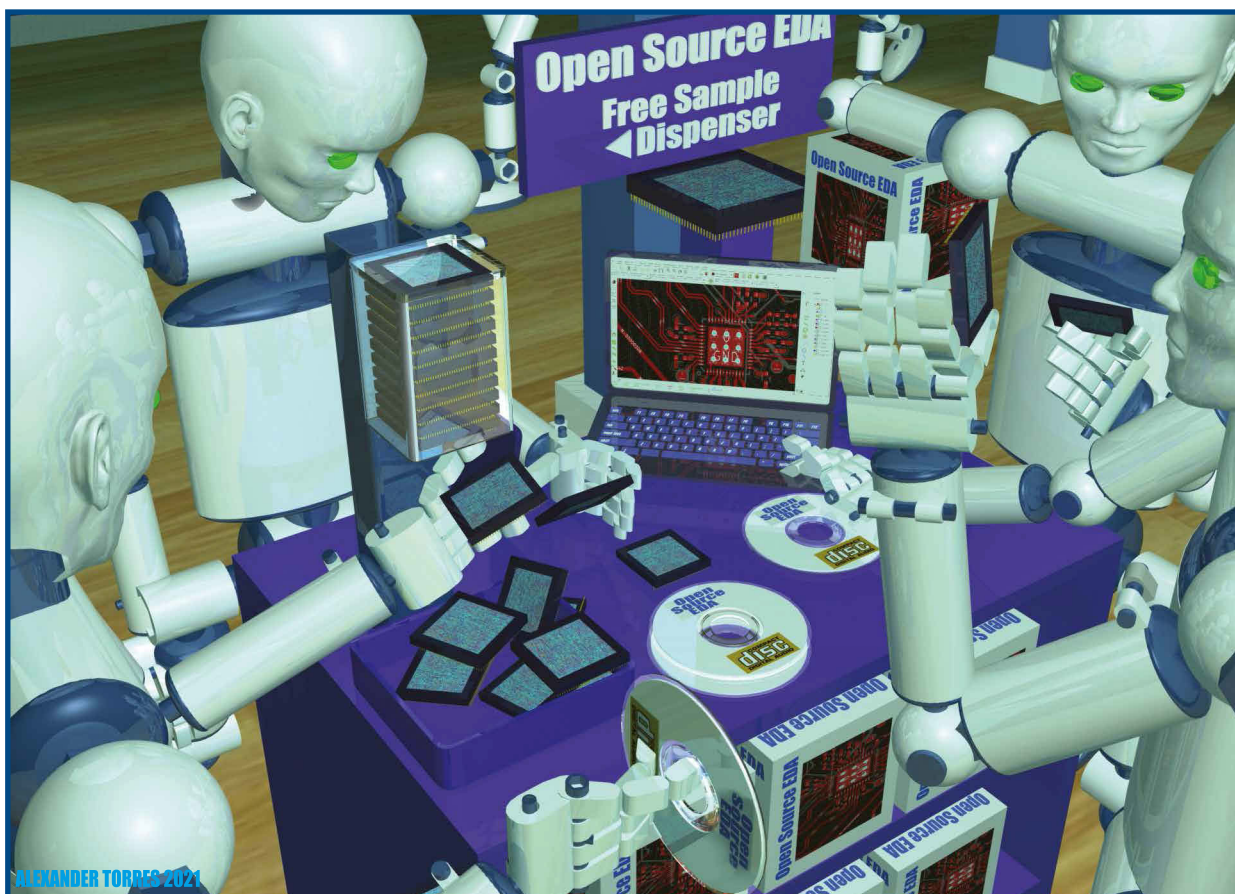


# Design & Test



ALEXANDER TORRES 2021

## Special Issue on Open-Source EDA

- ALIGN: A System for Automating Analog Layout
  - MAGICAL: An Open-Source Fully Automated Analog IC Layout System from Netlist to GDSII
    - An Open-Source EDA Flow for Asynchronous Logic
  - Real Silicon Using Open-Source EDA • Fault: Open-Source EDA's Missing DFT Toolchain
  - PyH2: Using PyMTL3 to Create Productive and Open-Source Hardware Testing Methodologies
    - OpenTimer v2: A Parallel Incremental Timing Analysis Engine
  - CATNAP-Sim: A Comprehensive Exploration and a Nonvolatile Processor Simulator for Energy Harvesting Systems
- General Interest**
- Design of  $\_I\_\_$  Shape Stub-Based Negative Group Delay Circuit • Design of Single-Bit Fault-Tolerant Reversible Circuits





**Connect. Support. Inspire.**

IEEE WIE is one of the world's leaders in changing the face of engineering. Our global network connects over 20,000 members in over 100 countries to advance women in technology at all points in their life and career.

[wie.ieee.org](http://wie.ieee.org)

#IEEEWIElead



**Associate Editor-in-Chief**  
Partha Pratim Pande  
Washington State University, USA  
pande@eecs.wsu.edu

**Editor-in-Chief**  
Jörg Henkel  
Karlsruhe Institute of Technology, Germany  
henkel@kit.edu

**Cover Design**  
Alexander Torres  
unclealextores@aol.com

## TECHNICAL AREAS

### KEYNOTES

Xiaobo Sharon Hu, *University of Notre Dame, USA*  
Sharon.Hu.8@nd.edu  
Patrick Groeneveld  
*Synopsys, USA*  
patrick.r.groeneveld@synopsys.com  
Sri Parameswaran  
*The University of New South Wales, Australia*  
sri.parameswaran@unsw.edu.au

### Analog, Asynchronous and Mixed Signal

Haralamos Stratigopoulos  
*CNRS Sorbonne, France*  
haralamos.stratigopoulos@lip6.fr

### Deep-Submicron Design, DFM and Yield

Vivek De, *Intel, USA*  
vivek.de@intel.com

### Embedded Software

Petru Eles  
*Linköping University, Sweden*  
Petru.eles@liu.se

### Emerging Technologies and Devices

Mircea Stan  
*University of Virginia, USA*  
mircea@virginia.edu

Yiran Chen  
*University of Pittsburgh, USA*  
yic52@pitt.edu

### Interconnect

Partha Pratim Pande  
*Washington State University, USA*  
pande@eecs.wsu.edu

Umit Ogras, *Arizona State University, USA*  
umit@asu.edu

Sudeep Pasricha, *Colorado State University, USA*  
sudeep@colostate.edu

### Low Power Systems

Anand Raghunathan, *Purdue University, USA*  
raghunathan@purdue.edu

### Memories

Said Hamdioui  
*Delft University of Technology, The Netherlands*  
S.Hamdioui@tudelft.nl

Binoy Ravindran, *Virginia Tech, USA*  
binoy@vt.edu

### Multicores

Tulika Mitra  
*National University of Singapore, Singapore*  
tulika@comp.nus.edu.sg  
Chia-Lin Yang, *National Taiwan University, Taiwan*  
yangc@csie.ntu.edu.tw

### Physical Design

David Pan, *University of Texas at Austin, USA*  
dpan@ece.utexas.edu

### Reliability

Sanghamitra Roy, *Utah State University, USA*  
sanghamitra.roy@usu.edu

### Real-Time and Mixed Criticality

Jian-Jia Chen  
*University of Dortmund, Germany*  
jian-jia.chen@cs.uni-dortmund.de

### Security and Trust

Ramesh Karri, *New York University, USA*  
rkarri@nyu.edu

### Self-Aware and Adaptive Systems

Axel Jantsch  
*Vienna University of Technology, Austria*  
axel.jantsch@tuwien.ac.at

Laura Pozzi  
*University of Lugano, Switzerland*  
laura.pozzi@usi.ch

### SoC and 3D

Sung Kyu Lim, *Georgia Tech, USA*  
limsk@ece.gatech.edu

Jin-Fu Li  
*National Central University, Taiwan*  
jfli@ee.ncu.edu.tw

Nicola Nicolici, *McMaster University, Canada*  
nicola@ece.mcmaster.ca

Cecilia Metra, *University of Bologna, Italy*  
cmetra@deis.unibo.it

### Test and Verification Techniques

Vivek Chickermane  
*Cadence, USA*  
vivekc@cadence.com

Fei Su, *Intel, USA*  
fei.su@intel.com

John Carulli  
*Globalfoundries, USA*  
john.carulli@globalfoundries.com

Sandeep Kumar Goel, *TSMC, Taiwan*  
SKGOEL@TSMC.COM

Xin Li  
*Duke University, USA*  
xinli.ece@duke.edu

Shreyas Sen  
*Purdue University, USA*  
shreyas@purdue.edu

### Things and Systems of Systems

Tei-Wei Kuo  
*National Taiwan University, Taiwan*  
ktw@csie.ntu.edu.tw

Paul Bogdan  
*University of Southern California, USA*  
paulbogdan2010@gmail.com

Mohammad Al Faruque  
*University of California Irvine, USA*  
alfaruqu@uci.edu

## DEPARTMENTS

### Interviews

Yao-Wen Chang  
*National Taiwan University, Taiwan*  
ywchang@ntu.edu.tw

Magdy Abadir, *USA*  
magdy.abadir@gmail.com

### Newsletters

CEDA Currents:  
Vasilis Pavlidis  
VP Publicity  
*University of Manchester, U.K.*

TTTC Newsletter:  
Theo Theocharides  
*University of Cyprus, Cyprus*  
theocharides@ucy.ac.cy

### DATC Newsletter:

Joe Damore, *IBM (retired), USA*  
joepdamore@aol.com

### Reports

Yervant Zorian, *Synopsys, USA*  
yervant.zorian@synopsys.com  
Massimo Poncino  
*Politecnico di Torino, Italy*  
massimo.poncino@polito.it

### Perspectives

David Atienza, *Swiss Federal Institute of Technology in Lausanne, Switzerland*  
david.atienza@epfl.ch  
Sri Parameswaran,  
*The University of New South Wales, Australia*  
sri.parameswaran@unsw.edu.au

Ahmed Jerraya, *CEA-Leti, France*  
ahmed.jerraya@cea.fr

Hidetoshi Onodera, *Kyoto University, Japan*  
onodera@i.kyoto-u.ac.jp

### Publicity

Mohammad Al Faruque  
*University of California Irvine, USA*  
alfaruqu@uci.edu

### Reviews

Scott Davidson, *Oracle, USA*  
scott.davidson@oracle.com

### Roundtables

David Yeh, *SRC, USA*  
david.yeh@src.org

Rajesh Gupta  
*University of California San Diego, USA*  
gupta@cs.ucsd.edu

### The Last Byte

Scott Davidson  
*Oracle, USA*  
scott.davidson@oracle.com

### The Road Ahead

Andrew Kahng  
*University of California San Diego, USA*  
abk@ucsd.edu

### Tutorials

Swarup Bhunia  
*Case Western Reserve University, USA*  
skb21@case.edu  
Jürgen Teich, *FAU Erlangen, Germany*  
juergen.teich@fau.de

## LIASONS

### Latin America Liaison

Ricardo Reis  
*Universidade Federal do Rio Grande do Sul, Brazil*  
reis@inf.ufrgs.br

## STEERING COMMITTEE

Luis Miguel Silveira (Chair)  
*Technical University of Lisbon, Portugal*  
lms@ieee-ceda.com

Krishnendu Chakrabarty (Past EiC)  
*Duke University, USA*  
krish@ee.duke.edu

Bruce Hecht, *Analog Devices, USA*  
Bruce.Hecht@analog.com

Enrico Macii, *Politecnico di Torino, Italy*  
enrico.macii@polito.it

Yervant Zorian, *Synopsys, USA*  
yervant.zorian@synopsys.com

## IEEE Publishing Operations

445 Hoes Lane, Piscataway, NJ 08854 USA

Dawn Melley  
Senior Director, Publishing Operations  
d.melley@ieee.org

Kevin Lisankie  
Director, Editorial Services  
k.lisankie@ieee.org

Peter M. Tuohy  
Director, Production Services  
p.tuohy@ieee.org

Jeffrey E. Cichocki  
Associate Director, Editorial Services  
j.cichocki@ieee.org

Neelam Khinvasara  
Associate Director,  
Information Conversion and  
Editorial Support  
n.khinvasara@ieee.org

Patrick Kempf  
Senior Manager, Journals Production  
p.j.kempf@ieee.org

Joanna Gojlik  
Journals Production Manager  
j.gojlik@ieee.org

Theresa L. Smith  
Production Coordinator  
tsmith@ieee.org

Felicia Spagnoli  
Advertising Production Manager  
f.spagnoli@ieee.org

**Submission Information:** Submit a Word, pdf, text, or PostScript version of your submission to ScholarOne Manuscripts, <https://mc.manuscriptcentral.com/dandit>

**Editorial:** Unless otherwise stated, bylined articles and columns, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *IEEE Design & Test* does not necessarily constitute endorsement by IEEE. All submissions are subject to editing for style, clarity, and length.

**IEEE Design & Test** (ISSN 2168-2356) is published bimonthly by the Institute of Electrical and Electronic Engineering, Inc. Responsibility for the contents rests upon the authors and not upon the IEEE, the Society/Council, or its members. IEEE Corporate Office: 3 Park Avenue, 17th Floor, New York, NY 10016. IEEE Operations Center: 445 Hoes Lane, Piscataway, NJ 08854. NJ Telephone: +1 732 981 0060. Price/Publication Information: To order individual copies for members and nonmembers, please email the IEEE Contact Center at [contactcenter@ieee.org](mailto:contactcenter@ieee.org). (Note: Postage and handling charge not included.) Member and nonmember subscription prices available upon request. Copyright and Reprint Permissions: Abstracting is permitted with credit to the source. Libraries are permitted to photocopy for private use of patrons, provided the per-copy fee of \$31.00 is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923. For all other copying, reprint, or republication permission, write to Copyrights and Permissions Department, IEEE Publications Administration, 445 Hoes Lane, Piscataway, NJ 08854. Copyright © 2021 by The Institute of Electrical and Electronics Engineers, Inc. All rights reserved. Periodicals Postage Paid at New York, NY and at additional mailing offices. Postmaster: Send address changes to *IEEE Design & Test*, IEEE, 445 Hoes Lane, Piscataway, NJ 08854-4141. GST Registration No. 125634188. CPC Sales Agreement #40013087. Return undeliverable Canada addresses to: Pitney Bowes IMEX, P.O. Box 4332, Stanton Rd., Toronto, ON M5W 3J4, Canada. IEEE prohibits discrimination, harassment, and bullying. For more information visit <http://www.ieee.org/nondiscrimination>. Printed in U.S.A.

# Contents

March/April 2021  
Volume 38 Number 2

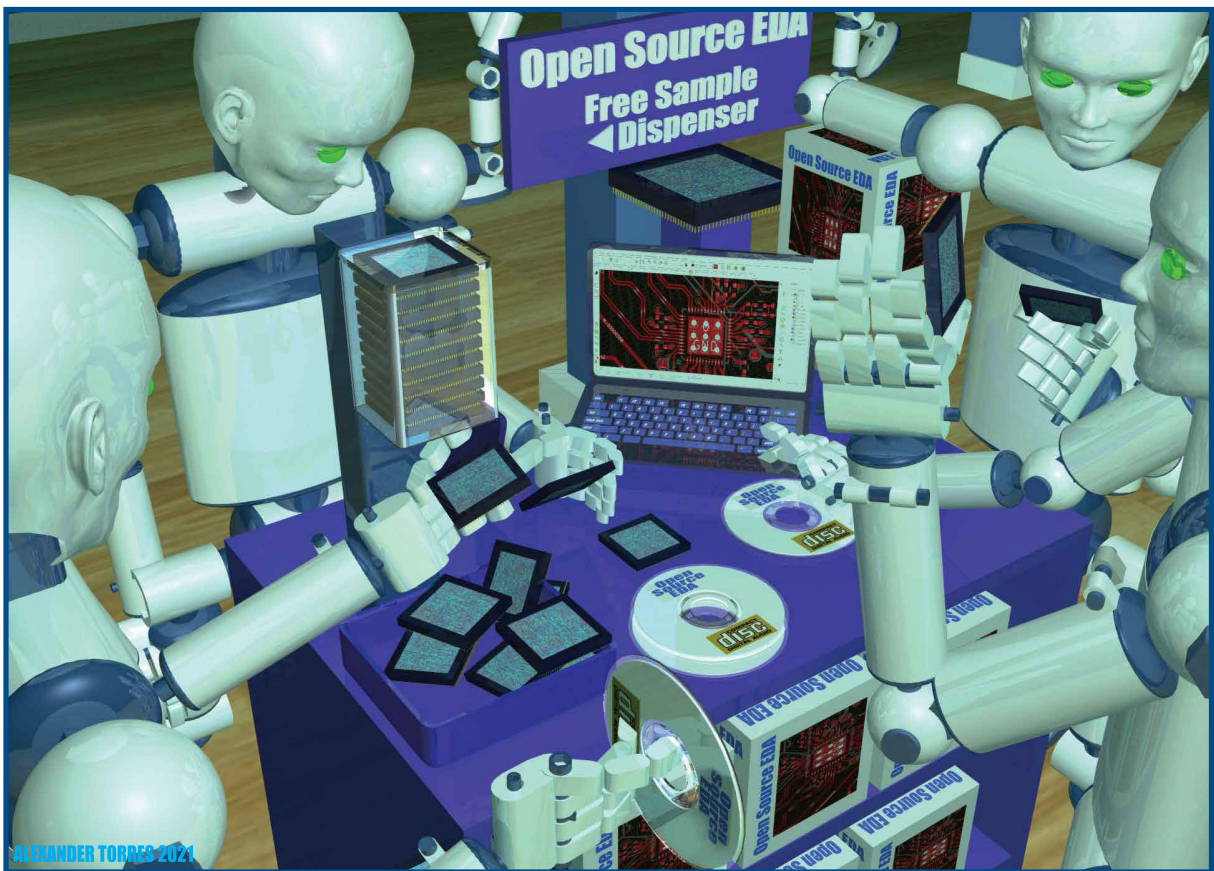
Copublished by the IEEE Council  
on Electronic Design Automation,  
the IEEE Circuits and Systems  
Society, the IEEE Solid-State  
Circuits Society, and the Test  
Technology Technical Council

## Special Issue

- 5** **Guest Editors' Introduction: The Resurgence of Open-Source EDA Technology**  
*Sherief Reda, Leon Stok, and Pierre-Emmanuel Gaillardon*
- 8** **ALIGN: A System for Automating Analog Layout**  
*Tommy Dhar, Kishor Kunal, Yaguang Li, Meghna Madhusudan, Jitesh Poojary, Arvind K. Sharma, Wenbin Xu, Steven M. Burns, Ramesh Harjani, Jiang Hu, Desmond A. Kirkpatrick, Parijat Mukherjee, Soner Yaldiz, and Sachin S. Sapatnekar*
- 19** **MAGICAL: An Open-Source Fully Automated Analog IC Layout System from Netlist to GDSII**  
*Hao Chen, Mingjie Liu, Biying Xu, Keren Zhu, Xiyuan Tang, Shaolan Li, Yibo Lin, Nan Sun, and David Z. Pan*
- 27** **An Open-Source EDA Flow for Asynchronous Logic**  
*Samira Ataei, Wenmian Hua, Yihang Yang, Rajit Manohar, Yi-Shan Lu, Jiayuan He, Sepideh Maleki, and Keshav Pingali*
- 38** **Real Silicon Using Open-Source EDA**  
*R. Timothy Edwards, Mohamed Shalan, and Mohamed Kassem*
- 45** **Fault: Open-Source EDA's Missing DFT Toolchain**  
*Manar Abdelatty, Mohamed Gaber, and Mohamed Shalan*
- 53** **PyH2: Using PyMTL3 to Create Productive and Open-Source Hardware Testing Methodologies**  
*Shunning Jiang, Yanghui Ou, Peitian Pan, Kaishuo Cheng, Yixiao Zhang, and Christopher Batten*
- 62** **OpenTimer v2: A Parallel Incremental Timing Analysis Engine**  
*Tsung-Wei Huang, Chun-Xun Lin, and Martin D. F. Wong*
- 69** **CATNAP-Sim: A Comprehensive Exploration and a Nonvolatile Processor Simulator for Energy Harvesting Systems**  
*Ali Hoseinghorban, Mohammad Abbasinia, Ali Paridari, and Alireza Ejlali*

## General Interest

- 78** **Design of  $\Sigma\Delta$  Shape Stub-Based Negative Group Delay Circuit**  
*Fayu Wan, Ningdong Li, Blaise Ravelo, Wenceslas Rahajandraibe, and Sébastien Lalléchère*
- 89** **Design of Single-Bit Fault-Tolerant Reversible Circuits**  
*Hari M. Gaur, Ashutosh K. Singh, Anand Mohan, Masahiro Fujita, and Dhiraj K. Pradhan*



Cover design by Alexander Torres

## DEPARTMENTS

- |           |  |            |  |
|-----------|--|------------|--|
| <b>4</b>  | <p><b>From the EIC</b><br/> <b>Open-Source Electronic Design Automation (EDA) Tools</b><br/> <i>Jörg Henkel</i></p>  | <b>100</b> | <p><b>Recap of the 39th Edition of the International Conference on Computer-Aided Design (ICCAD 2020)</b><br/> <i>Yuan Xie</i></p> |
| <b>97</b> | <p><b>Conference Reports</b><br/> <b>Report on First and Second ACM/IEEE Workshop on Machine Learning for CAD (MLCAD)</b><br/> <i>Marilyn Wolf, Jörg Henkel, Raviv Gal, and Ulf Schlichtmann</i></p> | <b>102</b> | <p><b>T TTC Newsletter</b></p>   |
|           |  | <b>104</b> | <p><b>The Last Byte</b><br/> <b>The Road to Open-Source EDA</b><br/> <i>Scott Davidson</i></p>                                     |



## From the EIC

# Open-Source Electronic Design Automation (EDA) Tools



■ **OPEN-SOURCE EDA HAS** become a major endeavor in the EDA community as it promises a variety of advances, especially with respect to common infrastructures such as internal representations and databases, as well as interoperability of tools. Its success is also due to eco-systems such as RISC-V, Chips Alliance, and Free Silicon Foundation. It is, therefore, the right time to be covered by *IEEE Design&Test* through our guest editors Sherief Reda, Pierre-Emmanuel Gaillardon, and Leon Stok. The special issue is structured into four articles covering entire design flows as well as further four articles focusing on specific tools. Thanks to the guest editors for this special issue.

In our General Interest section, we have two articles. The article titled “Design of  $\pi$ -Shape Stub-Based Negative Group Delay Circuit” by Wan et al. presents a type of negative group delay (NGD) circuit based on transmission line resonators.

The second General Interest article titled “Design of Single-Bit Fault-Tolerant Reversible Circuits” by

Gaur et al. introduces a generalized architecture for designing fault-tolerant reversible circuits.

ACM/IEEE MLCAD is a new workshop on Machine Learning for CAD. The report covers the first edition from 2019 held in Banff and the second edition from 2020 held in a virtual form. Thanks to the general chairs for the report.

The ACM/IEEE 39th International Conference on Computer-Aided Design (ICCAD 2020) took place as a virtual event. Thanks to Yuan Xie, the General Chair, for his conference report.

And thanks to Massimo Poncino, our Conference Reports editor, for acquiring the reports.

As always, last but not least, thanks to Scott Davidson for The Last Byte titled “The Road to Open-Source EDA.”

Enjoy reading! ■



Jörg Henkel  
Editor-in-Chief  
*IEEE Design&Test*

Digital Object Identifier 10.1109/MDAT.2021.3066119  
Date of current version: 8 April 2021.

# Guest Editors' Introduction: The Resurgence of Open- Source EDA Technology

**Sherief Reda**  
Brown University

**Leon Stok**  
IBM

**Pierre-Emmanuel Gaillardon**  
University of Utah

■ **IN THE 1980s**, the academic community produced several very high-quality electronic design automation (EDA) tools that spawned the EDA industry. Tools such as Spice [1], Espresso [2], and SIS [3] became the foundation of EDA companies. Open-source tools enable rapid innovation and create an ecosystem for scientific development. In recent years, the cost and difficulty involved in the design of integrated circuits (ICs) in advanced nodes have stifled hardware design innovation and have raised unprecedented barriers to bring new design ideas to the marketplace. Unlike the thriving software community, which enjoys a large number of open-source operating systems, compilers, libraries, and applications, the hardware community lacks such a modern ecosystem. With the advent of open silicon IP ecosystems, such as RISC-V, Chips Alliance, and Free Silicon Foundation, the time has come to reinvigorate the open-source movement in EDA tools. Recent programs from governmental agencies aim to jump-start the development of open-source EDA tools to reduce the cost and turnaround time of hardware design.

The availability of open-source EDA tools leads to multiple benefits. First, the availability of open-source tools leads to reproducible research with clear identification of state-of-the-art results. Thus, open-source tools enable unequivocal benchmarking that can quickly identify new EDA solutions that advance the state of the art. Second, open-source tools enable the acceleration of EDA research as innovations can be implemented at a faster rate by building on top of existing open-source tools and components. Thus, open-source tools lower the barrier to entry to the field by new students or practitioners. Third, full-stack open-source tools enable the quantification of improvements across the entire EDA flow. Since it is possible that improvements in one EDA stage are masked by downstream tools, evaluation within the context of a full stack of open-source EDA tools ensures that these improvements stick till the end. Fourth, open-source tools with standard I/O format exchanges enable a healthy ecosystem to develop between open-source tools and closed-source industrial tools, leading to faster dissemination of knowledge between academia and industry. Fifth, open-source EDA tools lead to a more trustworthy design process since the scrutinizing of an open-source tool by a community of developers can identify any backdoors that lead to the capture

*Digital Object Identifier 10.1109/MDAT.2020.3038851*

*Date of current version: 8 April 2021.*

of sensitive design information or potential insertion of hardware Trojans.

Open-source development leads to special challenges. First, there is a need for common infrastructure tools, such as EDA databases that consolidate shared tasks, such as data structures for internal circuit representations and reading/writing of standard I/O EDA formats. Second, there is a need for open-source tools to fully interoperate with physical design kits (PDKs) and libraries. Existing open-source PDKs and libraries (e.g., FreePDK45nm) do not map to any real manufacturing flows. However, recent efforts by Google and Skywater to release a full manufacturing PDK in 130 nm provide a hopeful path [4], where other vendors might follow suit and open-source the PDKs and libraries of some of their mature technology nodes. Third, open-source tools need maintenance beyond their release, requiring long-term commitment and funding. Thus, developing and engaging a community of developers through collaborative platforms, e.g., Github, is essential for long-term success.

In this issue, we have collected papers that touch upon key parts of the design flow. A requirement in the review process was that the code is released as open-source, and all the tools released with the special issue are given in Table 1.

The first series of articles introduces complete flows, addressing a specific design category, namely analog, synchronous digital, and asynchronous digital. First, analog layout tools are a key part of any electronic design system. They are as essential to analog design as they are to digital design where they are used to design the cell libraries, memory cells, and all key analog components. The paper titled “ALIGN: A System

for Automating Analog Layout” describes a correct-by-construction approach to synthesize electrically and designs compliant design. By taking advantage of layout hierarchies the authors are able to apply this to an interesting class of circuits. The second paper on analog design flows is titled “MAGICAL: An Open-Source Fully Automated Analog IC Layout System from Netlist to GDSII,” where it presents MAGICAL, which is a fully automated analog IC layout system. MAGICAL takes a netlist and design rules as inputs and it produces the final GDS layout in a fully automated fashion. For asynchronous logic flows, the paper titled “An Open-Source EDA Flow for Asynchronous Logic” presents an open-source EDA flow for digital asynchronous circuits, capable of supporting many different families of asynchronous circuit families from logic synthesis all the way down to GDSII. Finally, the paper titled “Real Silicon Using Open-Source EDA” demonstrates that complete open-source tooling can be used to design industrial quality digital circuits. Using the OpenLane framework, based itself on the OpenROAD tool [5], the authors show a complete set of RISC-V-based SoC.

In addition to complete flows, the second series of articles introduces specific point tools. Design for test (DFT) is an integral part of the design flow. An open-source DFT flow is therefore essential for any open-source solution. The paper titled “Fault: Open-Source EDA’s Missing DFT Toolchain” describes an approach to fill in this missing piece. The paper titled “PyH2: Using PyMTL3 to Create Productive and Open-Source Hardware Testing Methodologies” proposes a new model testing and verification methodology, PyH2, using property-based random testing in Python. PyH2 leverages the whole Python ecosystem to build test benches and models. The paper “OpenTimer v2: A Parallel Incremental Timing Analysis Engine” introduces a high-quality open-source static timing analysis engine that is capable of parallel incremental timing and that provides an efficient API to facilitate the development of complex EDA tools. Finally, the paper titled “CATNAP-Sim: A Comprehensive Exploration and a Nonvolatile Processor Simulator for Energy Harvesting Systems” introduces an architecture exploration tool to study and understand the tradeoffs of future processor systems using

**Table 1. Open-source tools released for articles in this special issue.**

Tool	Source
ALIGN	<a href="https://github.com/ALIGN-analoglayout/ALIGN-public">https://github.com/ALIGN-analoglayout/ALIGN-public</a>
MAGICAL	<a href="https://github.com/magical-eda/MAGICAL">https://github.com/magical-eda/MAGICAL</a>
Asynchronous logic	<a href="https://github.com/asynvcvsi/">https://github.com/asynvcvsi/</a>
OpenLane	<a href="https://github.com/efabless/openlane">https://github.com/efabless/openlane</a>
Fault	<a href="https://github.com/Cloud-V/Fault">https://github.com/Cloud-V/Fault</a>
PyMTL3 and PyH2	<a href="https://github.com/pymtl/pymtl3">https://github.com/pymtl/pymtl3</a>
OpenTimer	<a href="https://github.com/OpenTimer/OpenTimer">https://github.com/OpenTimer/OpenTimer</a>
CATNAP-Sim	<a href="http://esrlab.ce.sharif.ir/download/CATNAP-Sim.gz.tar">http://esrlab.ce.sharif.ir/download/CATNAP-Sim.gz.tar</a>



nonvolatile memory and help guide the design of the future.

We hope you enjoy the articles and tools that are available with this special issue. ■

## ■ References

- [1] L. W. Nagel and D. O. Pederson, "SPICE (Simulation Program with Integrated Circuit Emphasis)," EECS Dept., Univ. California, Berkeley, Tech. Rep. No. UCB/ERL M382, 1973.
- [2] R. K. Brayton et al., *Logic Minimization Algorithms for VLSI Synthesis* (The Kluwer International Series in Engineering and Computer Science 2). New York, NY, USA: Springer, 1984, ISBN 978-1-4612-9784-0, pp. 1–193.
- [3] E. M. Sentovich et al., "SIS: A system for sequential circuit synthesis," EECS Dept., Univ. California, Berkeley, Tech. Rep. No. UCB/ERL M92/41, May 1992.
- [4] Google Skywater PDK. [Online]. Available: <https://github.com/google/skywater-pdk>
- [5] T. Aja et al., "Toward an open-source digital flow: First learnings from the OpenROAD project," in *Proc. ACM/IEEE Design Autom. Conf.*, 2019, Article no. 76, pp. 1–4.

**Sherief Reda** is a Full Professor with the School of Engineering, Brown University, Providence, RI, USA. His research interests are in the area of hardware systems, with focus on energy-efficient computing, design automation of integrated circuits, embedded systems, and computer architecture. He is a Senior Member of IEEE.

**Leon Stok** is Vice President of EDA IBM. Stok has a PhD degree in electrical engineering from the Eindhoven University of Technology, Eindhoven, The Netherlands. He is a Fellow of IEEE.

**Pierre-Emmanuel Gaillardon** is an Associate Professor with the Electrical and Computer Engineering (ECE) Department, The University of Utah, Salt Lake City, UT, where he leads the Laboratory for NanoIntegrated Systems (LNIS). He is a Senior Member of IEEE.

■ Direct questions and comments about this article to Sherief Reda, School of Engineering, Brown University, Providence, RI 02912 USA; [sherief\\_reda@brown.edu](mailto:sherief_reda@brown.edu).

# ALIGN: A System for Automating Analog Layout

**Tonmoy Dhar and Kishor Kunal**

University of Minnesota

**Yaguang Li**

Texas A&M University

**Meghna Madhusudan, Jitesh Poojary, and**

**Arvind K. Sharma**

University of Minnesota

**Wenbin Xu**

Texas A&M University

**Steven M. Burns**

Intel Labs

**Ramesh Harjani**

University of Minnesota

**Jiang Hu**

Texas A&M University

**Desmond A. Kirkpatrick, Parijat Mukherjee, and Soner Yaldiz**

Intel Labs

**Sachin S. Sapatnekar**

University of Minnesota

## *Editor's notes:*

This article describes a correct-by-construction approach to synthesize electrically and designs compliant design. By taking advantage of layout hierarchies the authors are able to apply this to an interesting class of circuits.

—*Sherief Reda, Brown University*

—*Leon Stock, IBM*

—*Pierre-Emmanuel Gaillardon, University of Utah*

■ **ANALOG LAYOUT**, Intelligently Generated from Netlists (ALIGN) [1] is an open-source layout generator for analog circuits that is currently under development. Version 1 of the software flow was released in August 2020. The ALIGN project engages a joint academic/industry team to translate a SPICE-level netlist into a physical layout, with a 24-hour turnaround and no human in the loop. The ALIGN flow inputs a netlist of the topology and transistor sizes of which have already been chosen, specifications, and a process design kit (PDK), and outputs GDSII.

*Digital Object Identifier 10.1109/MDAT.2020.3042177*

*Date of publication: 3 December 2020; date of current version: 8 April 2021.*

The philosophy of ALIGN is to compositionally synthesize the layout by first identifying layout hierarchies in the netlist, then generating correct-by-construction layouts at the lowest level of the hierarchy, and finally assembling blocks at each level of hierarchy during placement and routing. Thus, a key step in ALIGN is to identify these hierarchies

to recognize the building blocks of the design. In doing so, ALIGN mimics the human designer, who identifies known blocks, lays them out, and then builds the overall layout hierarchically. At the lowest level of this hierarchy is an individual transistor; these transistors are then combined into larger fundamental primitives [e.g., differential pairs and current mirrors], then modules [e.g., operational transconductance amplifiers (OTAs)], up through several levels of hierarchy to the system level [e.g., a radio-frequency (RF) transceiver]. ALIGN uses a mix of algorithmic techniques, template-driven design, and machine learning (ML) to create layouts that are at the level of sophistication of the expert designer.

Unlike digital designs that are built from a composition of a small number of building blocks, analog circuits tend to use a wide variety of structures. Each of these has its own constraints and requirements, and traditionally only the expert designer has been able to build circuits that could deliver high performance. ALIGN targets a wide variety of analog designs, in both bulk and FinFET technologies, covering four broad classes of functionality:

- Low-frequency components that include analog-to-digital converters (ADCs), amplifiers, and filters.
- Wireline components that include clock/data recovery, equalizers, and phase interpolators.
- RF/wireless components that implement transmitters, receivers, etc.
- Power delivery components that include capacitor- and inductor-based DC-to-DC converters.

Each class is characterized by similar building blocks that may have a similar set of performance parameters, although it should be mentioned that there is considerable diversity even within each class. An overview of factors that are important for designs in each category is summarized in Figure 1.

There have been several prior efforts to automate analog layout synthesis [2]–[8], but these methods are not widely deployed in tools today. Some methods address limited classes of designs; others cannot be tuned to handle a wide enough set of variants of the same design class. Moreover, there is a general consensus that prior methods for automating analog layout have been unable to match the expert designer, both in terms of the ability to comprehend and implement specialized layout tricks and the number and variety of topologies with circuit-specific constraints. The ultimate goal for analog layout synthesis is to reach the quality of a hand-crafted design.

In recent years, the landscape has shifted in several ways, making automated layout solutions attractive. First, in nanometer-scale technologies, restricted design rules with fixed pitches and unidirectional routing limit the full freedom of layout that was available in older technologies, thus reducing the design space to be explored during layout, reducing the advantage to the human expert. Second, today more analog blocks are required in integrated systems than before, and several of these require correct functionality and modest performance. The combination of increasing analog content with the relaxation in specifications creates a sweet spot

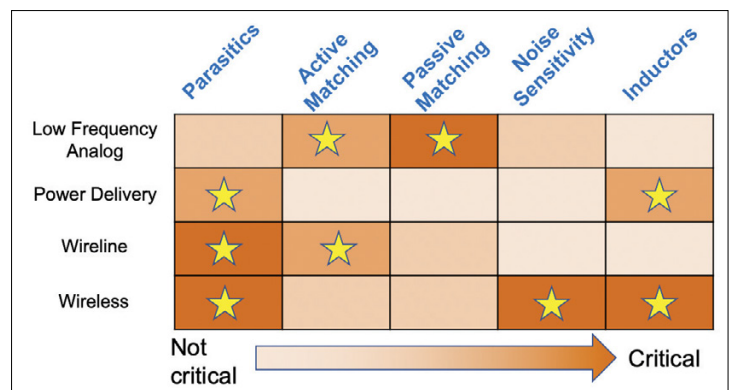
for analog automation. Even for high-performance blocks, an automated layout generator could considerably reduce the iterations between circuit optimization and layout, where layout generation is the primary bottleneck. Third, the advent of ML provides the promise for attacking the analog layout problem in a manner that was not previously possible, and set the stage for no-human-in-the-loop design.

This article provides an overview of the technical details of ALIGN and shows how ALIGN has been used to translate analog circuit netlists to layouts. The core ALIGN engine can be run with no human in the loop, enabled by ML algorithms that perform the functions typically performed by humans, e.g., recognizing hierarchies in the circuit during auto-annotation, or generating symmetry constraints for layout. ML algorithms can also be instrumental in creating rapid electrical constraint checkers, which verify whether a candidate placement/routing solution meets performance constraints or not, and using this to guide the place-and-route engine toward optima that meet all specifications. For more in-depth details, the reader is referred to detailed descriptions given in [9]–[12], and to watch for new publications of ongoing work by our group.

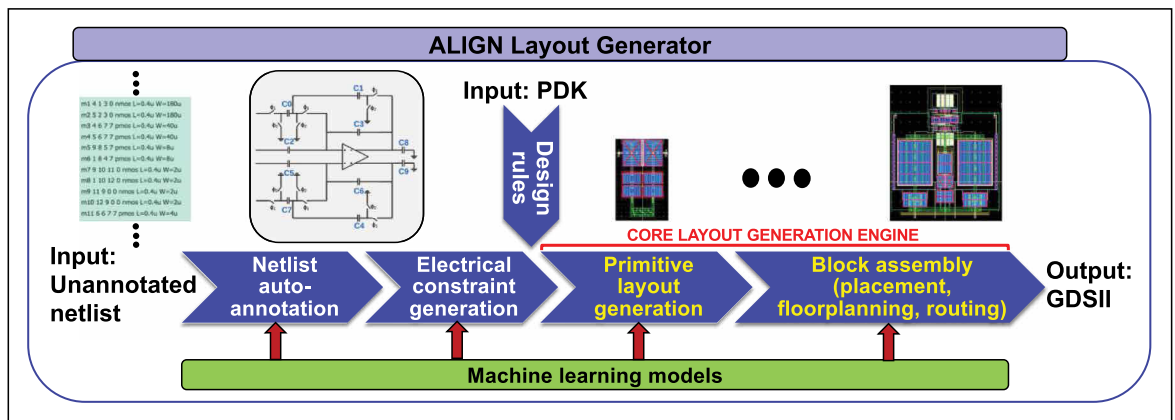
### Technical core of ALIGN

The ALIGN flow consists of five modules, illustrated in Figure 2:

- *Netlist auto-annotation* creates a multilevel hierarchical representation of the input netlist and identifies structural symmetries in the netlist. This is a key step that is used to hierarchically build the layout of the circuit.



**Figure 1. Classification of analog circuits, showing the factors that are important for each category of circuits.**



**Figure 2. Overview of the ALIGN flow.**

- *Design rule capture* abstracts the proprietary PDK into a simplified grid, appended with Boolean constraints as needed, that must be obeyed at all steps during layout.
- *Constraint generation* identifies the performance constraints to be met and transforms them into layout constraints, such as maximum allowable net lengths, or constraints such as matching/common-centroid based on structural information identified during auto-annotation.
- *Parameterized primitive cell generation* automatically builds layouts for primitives, the lowest-level blocks in the ALIGN hierarchy. Primitives typically contain a small number of transistor structures (each of which may be implemented using multiple fins and/or fingers). A parameterized instance of a primitive in the netlist is automatically translated to a GDSII layout in this step.
- *Hierarchical block assembly* performs placement and routing on the hierarchical circuit structure while meeting geometric and electrical constraints.

The flow creates a separation between the open-source code and proprietary data. Proprietary PDK models must be translated into an abstraction that is used by the layout generators. Parts of the flow are driven by ML models: the flow provides the infrastructure for training these models on proprietary data.

The overall ALIGN flow is intended to support no-human-in-the-loop design. However, the flow is modular and supports multiple entry points: for example, the auto-annotation module could be replaced by designer annotation, and the rest of the flow could be executed using this annotation. The flow is flexible to user input: for example, the user can specify new primitives, and

they will be used by the annotation module as well as the layout generator within the flow.

#### Netlist auto-annotation

This step groups transistors and passives in the input netlist into a hierarchical set of building blocks and identifies constraints on the layout of each block. The input to ALIGN is a SPICE netlist that is converted to a graph representation. Next, features of the graph are recognized, and a circuit hierarchy is created. If the input netlist is partitioned into sub-circuits, such information is used during recognition, but ALIGN does not count on the netlist hierarchy. Instead, hierarchies are automatically identified and annotated. It is important to note that the best layout hierarchy may sometimes differ from a logical netlist hierarchy; hence, ALIGN may flatten netlist hierarchies to build high-quality layouts.

Analog designers typically choose from a large number of variants of each design block, e.g., between textbooks and research papers, there are well over 100 widely used OTA topologies of various types (e.g., telescopic, folded cascode, Miller-compensated). Prior methods are library-based (i.e., they match a circuit to prespecified templates) [5] or knowledge-based (i.e., they determine block functionality using a set of encoded rules) [2], or both [13]. Library-based methods require a large library, while rule-based methods must be supported by an exhaustive knowledge base, both of which are hard to build and maintain. ALIGN uses two approaches for annotating circuits blocks, both based on representing the circuit connectivity using a graph representation:

(1) *ML-based methods*: For commonly encountered blocks, the problem of identifying blocks maps on to

whether a subgraph of the larger circuit is isomorphic to a known cell. However, to allow for design variants, ALIGN uses *approximate* graph isomorphism, enabled by the use of graph convolutional neural networks (GCNs) that classify nodes within the circuit graph into classes [e.g., OTA nodes, low-noise amplifier (LNA) nodes, and mixer nodes]. With some minimal postprocessing, it is demonstrated that this approach results in excellent block recognition. Details of the approach are provided in [9]. A training set for the GCN, consisting of 1390 OTA circuits, including bias networks, is available on the ALIGN GitHub repository.

(2) *Graph-traversal-based methods*: It is unrealistic to build a training set that covers every possible analog block, and for blocks that lie outside the scope of the GCN training set, we use graph-based approaches to recognize repeated structures within a circuit. Such structures typically require layout constraints: for example, ADCs may use a set of binary-weighted capacitors or a set of resistors in an R-2R ladder, and these require careful placement in common-centroid fashion and symmetric routing. ALIGN employs methods based on graph traversal and approximate subgraph isomorphism to recognize these array structures.

Once these structures are recognized in a very large circuit graph, they form a level of the hierarchy. Within these blocks, lower hierarchical levels can be detected using conventional subgraph isomorphism methods: sub-blocks at these levels have fewer variants and can be efficiently recognized using library-based approaches.

Figure 3 shows the results of auto-annotation on a switched-capacitor (SC) filter. A GCN-based approach

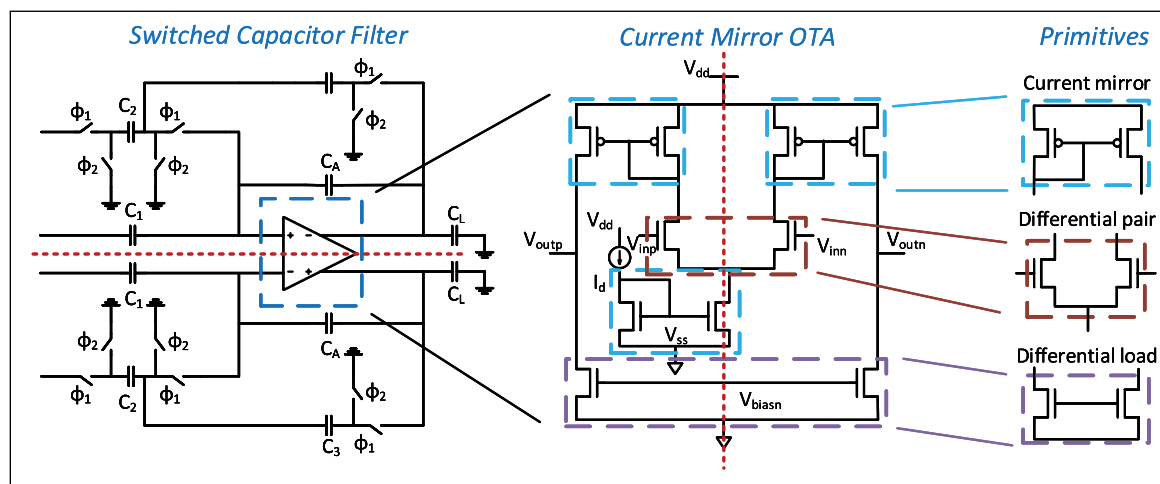
can be used to identify the current-mirror OTA, and then primitives within the OTA can be identified. In the process, lines of symmetry within each structure can be found, as illustrated in the figure. At the primitive level, since the layouts are generated by the parameterized cell generator, these lines of symmetry are implicit in the definition of the primitive. At higher levels, these can be inferred during auto-annotation.

### Design rule abstraction

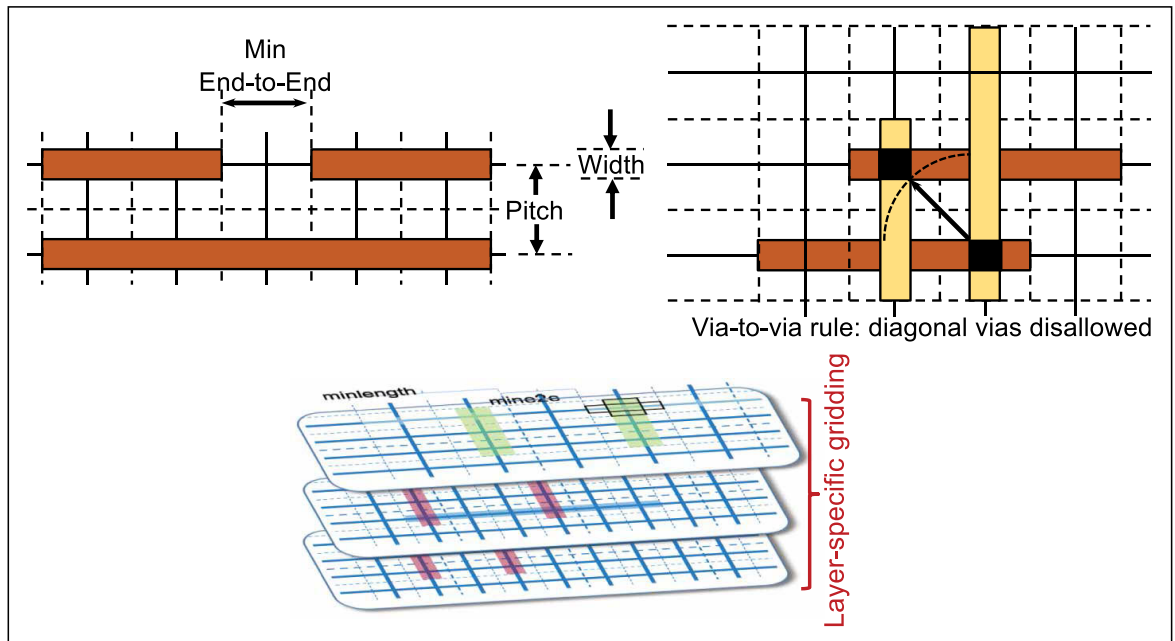
The ALIGN layout tools are guided by process-specific design rules that ensure design rule correctness. The complexity of design rules has grown significantly in recent process generations. Efforts at building generalized abstractions for process rules have previously been proposed (e.g., [14]). ALIGN uses a more efficient design rule abstraction mechanism that creates fixed grid structures in FEOL and BEOL layers, as illustrated in Figure 4. Major grids (bold lines) represent centerlines for routes, while minor grids (dashed lines) correspond to stopping points for features. The gridding structure and basic process information are abstracted into a JSON file. For BEOL layers, this includes:

- default wire dimensions, pitch, and grid offset (*Pitch, Width, MinL, MaxL, Offset*);
- end-to-end spacing design rules (*EndToEnd*);
- metal direction, colors (*Direction, Color*);
- via rules (*Space{X/Y}, Width{X/Y}, VencA\_{L/H}, VencP\_{L,H}*).

While this is superficially similar to traditional  $\lambda$ -rules, our abstraction permits a different



**Figure 3. Extracting netlist hierarchy during auto-annotation.**



**Figure 4. Design rule abstraction using per-layer grids and rules.**

gridding structure that can vary from layer to layer, and the use of major/minor grid lines that represent wire pitches, wire overhangs, as well as the ability to incorporate via rules through Boolean constraints. Our approach reduces the complex set of conditions embedded in thousands of rules in a design rule manual to a massively simplified and much smaller set, enforcing some limitations through the choice of grids. It is found, through comparisons with manual design, that this leads to minimal or zero degradation in layout quality. Advanced commercial process nodes (22, 10, 7 nm, and beyond) have been abstracted into this simplified form. The abstraction enables layout tools to comprehend PDK features such as regular and irregular width and spacing grids (for each layer), minimum end-to-end spacing design rules (between metals in the same track), minimum length design rules, and enforced stopping point grids. For convenience, the JSON file also encodes per unit parasitics for metal layers and vias.

To facilitate further layout research, we have released design rules for mock PDKs based on public-domain information to abstract layout rules at a 14-nm FinFET node [15] and a 65-nm bulk node [16]. Although they do not represent real technologies, they are realistic. Validation of the design tools

on these PDKs, which can be freely shared, helps the software development process.

#### Constraint generation

Two types of constraints are generated to guide layout.

(1) *Geometric constraints*: As the auto-annotation step recognizes known blocks or array structures, it associates geometric requirements with these blocks, such as symmetry, matching, and common-centroid constraints. For instance, Figure 3 shows lines of symmetry in an OTA structure that must be respected during layout. These constraints are extracted naturally as part of auto-annotation. In contrast with prior methods that are based on simulation-intensive sensitivity analysis [17] or graph traversal-based exact matching to templates [5], the approach in the ALIGN method [10] combines graph traversal methods with ML-based methods and is computationally efficient, and capable of finding hierarchically nested symmetry constraints even under approximate matches.

(2) *Electrical constraints*: ALIGN generates a layout based on a fixed netlist, and performance shifts are driven by changes in parasitics from netlist-level estimates to post layout values. Therefore, ALIGN translates electrical constraints to bound the maximum parasitics at any node of the circuit.

For instance, an electrical constraint may be translated to a maximum limit on the resistance of a wire connecting two nodes, which in turn corresponds to a constraint on the maximum length, the number of parallel metal tracks, and the number of vias on the route connecting these nodes. This feature is currently being implemented in ALIGN [11], [12] and is a work in progress. The essential idea is to develop a fast ML inference engine that operates within the inner loop of an iterative placer, and for each placer configuration, determines whether its electrical constraints are satisfied.

These constraints are passed on to the layout generation engine to guide layout at all levels of hierarchy.

### Parameterized primitive layout generation

ALIGN provides the user with a predefined library of parameterizable primitives, as illustrated in Figure 5. Each primitive consists of a small number of transistor or passive units; however, each such unit may consist of multiple replicated structures, such as multifin/multifinger transistors, or resistive/capacitive arrays.

The primitive cell layout follows the gridded abstraction defined by the design rules, and cell generation can be parameterized in terms of the unit cell and the number of unit cells, as shown in Figure 6. For a transistor, a unit cell may be parameterized by the number of fins in a FinFET technology; for a capacitor, parameterization may correspond to the size of the unit capacitor. Additionally, primitive layouts can be parameterized by their aspect ratio, their layout style (common-centroid versus interdigitated transistors), the gate length, the effective widths of critical wires in the cell, etc.





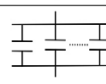








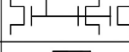






The utility in recognizing primitives and creating parameterized layouts is in enabling ALIGN to create layouts that incorporate the appropriate geometric constraints (e.g., symmetry or common-centroid). In principle, a layout could be built using a “sea of transistors,” where the primitive corresponds to a single transistor, but it would be challenging for such an approach to enforce symmetry requirements beyond the transistor primitives. Prior methods for primitive layout generation [18]–[21] have generally not been as modular or scalable as the ALIGN approach.

### Hierarchical block assembly

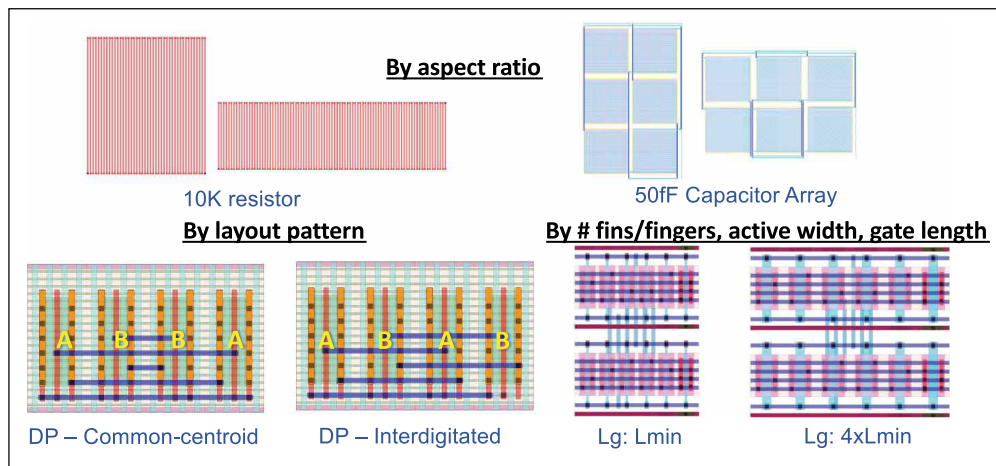
Given the layouts of all primitives and the hierarchical block-level structure of the circuit, extracted during auto-annotation, the placement, and routing step performs hierarchical block assembly that obeys the geometric and electrical constraints described earlier.

Each layout block in the hierarchy can have multiple layout options with different shapes generated for each module. For example, primitives can be parameterized by aspect ratio, and multiple aspect ratios for other blocks may be generated. Flexible shapes drive floorplanning-like placement algorithms that deliver compact layouts under the electrical and geometric constraints passed on to them by the constraint generation step. Routing is integrated into each hierarchical level, accounting for net length/parasitic constraints, shielding and symmetry requirements, and conforming with the design rules embedded into the PDK abstraction. The placer is based on prior work using the sequence pair method [7] and can handle general geometric constraints, such as symmetry, matching, and alignment. Symmetry, shielding, and resistance-constrained routing are supported during routing.

The ALIGN flow can employ one of the two detailed routers.

Primitive	Schematic	Primitive	Schematic
Resistor		Current mirror 1	
Capacitor		Current mirror 2	
Capacitor array		Voltage reference	
Switch		Level shifter	
Diode-connected load		Current mirror load	
Differential pair		Current mirror bank	
Cross-coupled pair 1		Level shifter bank	
Cross-coupled pair 2		Dummy 1	
Differential load		Dummy 2	
Cascode pair		Decap	

**Figure 5. Examples of primitive structures.**



**Figure 6. Parameterization of primitive layouts.**

- A constructive router that uses an integer linear programming formulation and an A\* algorithm; this works particularly well for more sparse designs.
- A satisfiability-based detailed router,<sup>1</sup> released by Intel, is well suited for congested designs.

## Working in an open-source environment

### Why open-source software?

Aside from technical innovations, ALIGN breaks new ground in providing a fully open-source analog layout software flow, which has not been available in the past. The availability of open-source software is crucial for nurturing future innovations in the field. First, further research can build upon a “piece of the puzzle” of analog layout design: for instance, a new cell generator can plug into the open-source ALIGN flow and show end-to-end results from netlist to layout, rather than providing limited results at the end of cell generation. Second, open-source enables a path to ensure that reported results can be reproducible. The traction for open-source is evidenced not only through the efforts in ALIGN, but also in other notable efforts on analog layout [22] and digital layout (including back-end infrastructure such as parasitic extraction on power delivery that is more broadly applicable to any other class of design) [23].

### Open-source designs

Unlike digital designs, where a wealth of designs exists in the public domain, the font of

analog designs is very sparse. Design parameters tend to be closely linked with process nodes, and existing automation flows do not allow robust circuit optimization to meet constraints. Sharing designs based on a commercial PDK over multiple institutions requires a multiway nondisclosure agreement involving the institutions, the foundry, and the foundry access provider. Within the ALIGN team, this issue was complicated by the need for such an agreement to cover both academic and industry team members.

The ALIGN GitHub repository hosts a number of sized analog netlists, a set that is growing, to facilitate open research. These netlists contain test benches that measure the performance parameters of the circuit to verify its adherence to specifications. Moreover, as stated earlier, the repository contains unsized netlist topologies for a variety of OTA circuits.

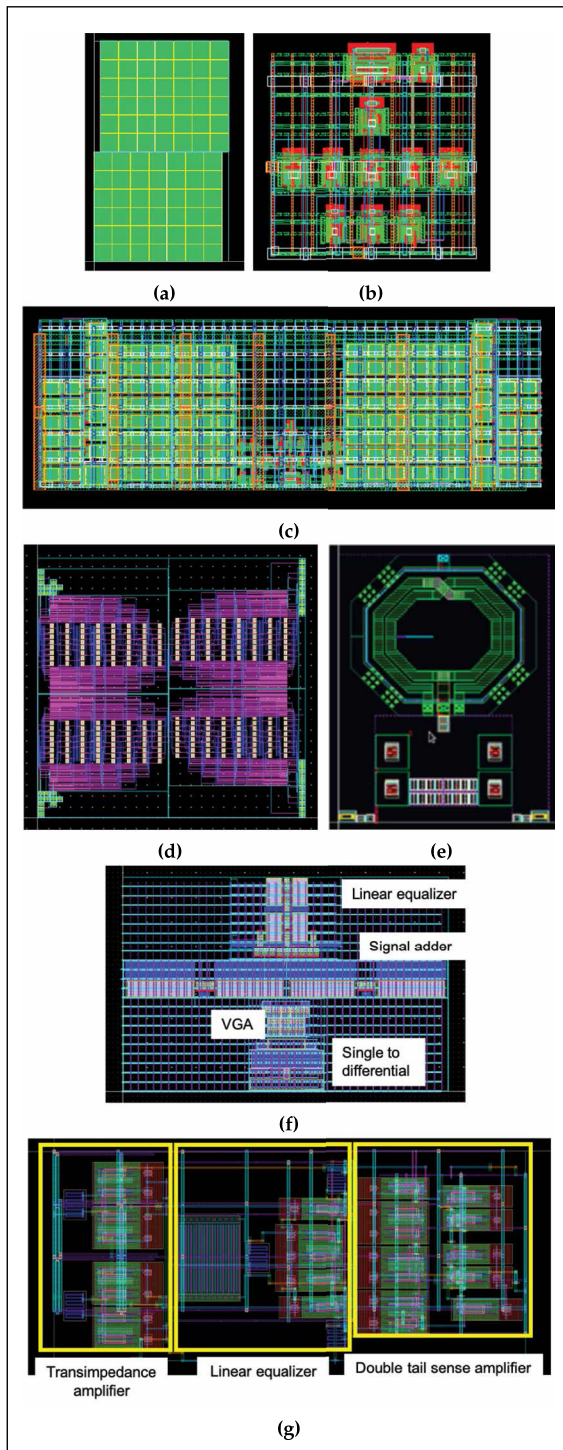
### Software infrastructure

The software flow is maintained on a GitHub repository [24] and maybe downloaded and installed in a native Linux environment. Alternatively, it may be run in a lightweight Docker container that performs operating system virtualization and enables portability and ease of maintenance. ALIGN can leverage the use of other open-source tools such as the KLayout layout viewer. The core software flow is Python-based, and the computationally intensive engines—notably the placer and router—are implemented in C++.

The project is aided by the use of tools that are vital to an open-source infrastructure with continuous

<sup>1</sup> [github.com/ALIGN-analoglayout/AnalogDetailedRouter](https://github.com/ALIGN-analoglayout/AnalogDetailedRouter)





**Figure 7. Sample layouts generated by ALIGN. Note that the block sizes are different; the layouts are not on the same scale. (a) SC DC-to-DC converter. (b) OTA with bias circuitry. (c) SC filter. (d) ADC. (e) Bandpass filter. (f) Equalizer. (g) Optical receiver.**

integration (CI). These include CI build flows, using CircleCI, for automated building of new components as they are added to the repository; unit testing, using pytest, to verify the correctness of individual units of source code that is added to the repository; code coverage to measure how much of the code is executed by the automated tests, using coverage.py with Codecov for tracking; and automated code review for code quality checks using Codacy.

## Results

The ALIGN flow has been applied to generate layouts for circuits that lie in all four classes: 1) low-frequency analog; 2) wireline; 3) wireless; and 4) power delivery. We are unaware of a prior layout generator that has been demonstrated to handle such a broad class of circuits. Figure 7 illustrates a sample set of layouts generated using ALIGN: these include a current-mirror OTA with bias circuitry and its power grid (Figure 7b), an SC filter containing the OTA (Figure 7c), an ADC (all low-frequency analog), a bandpass filter (Figure 7e) (wireless), an SC DC-to-DC converter (Figure 7a) (power delivery), and an equalizer (Figure 7f) and an optical receiver (Figure 7g) (both wireline). The layouts are compact and regular.

A set of representative results for the post layout performance analysis of ALIGN-generated layouts for the OTA (Figure 7b) and the SC filter (Figure 7c) containing the OTA are shown in Tables 1 and 2, respectively. For the larger

**Table 1. Postlayout performance analysis of the ALIGN-generated OTA layout.**

	Schematic	Manual layout (RC extract)	ALIGN Layout (RC extract)
Gain (dB)	24.28	24.22	24.14
3dB frequency (MHz)	24	24	24
UGF (MHz)	199	197	198
Phase margin (°)	89	88	88
Input offset (mV)	~0	0.13	0.10

**Table 2. Postlayout performance analysis of the ALIGN-generated SC filter layout.**

Specification	Schematic	Manual layout (RC extract)	ALIGN layout (RC extract)
Gain (dB)	16.1	15.84	15.59
3dB frequency (KHz)	503	511	524
Unity gain frequency (KHz)	3435	3415	3610
Input offset (mV)	0	0.13	0.10

**Table 3. Comparing the performance of the schematic (S), manual layout (M), and the ALIGN-generated layout (A) of several wireline circuits.**

	SDC		Signal adder	
	Gain (dB) [ $\Delta$ Gain]	BW (GHz) [ $\Delta$ BW]	Gain (dB) [ $\Delta$ Gain]	BW (GHz) [ $\Delta$ BW]
S	-5.6	27.9	2.9	24.5
M	-6.0 [-3.5%]	19.8 [-29.0%]	2.3 [-5.8%]	15.7 [-36.1%]
A	-6.1 [-4.8%]	23.0 [-17.6%]	2.2 [-7.0%]	19.8 [-19.2%]
	VGA		Linear equalizer	
	Gain (dB) [ $\Delta$ Gain]	BW (GHz) [ $\Delta$ BW]	Gain (dB) [ $\Delta$ Gain]	BW (GHz) [ $\Delta$ BW]
S	-10.9dB $\sim$ 5.5	26.25	1.2	18.0
M	-10.9dB $\sim$ 5.6 [1.0%]	12.58 [-52.1%]	0.9 [-3.4%]	13.9 [-22.8%]
A	-11.0dB $\sim$ 5.0 [-5.6%]	13.40 [-49.0%]	0.8 [-4.2%]	15.7 [-12.8%]

block, the SC filter, the extraction results show a good match with the schematic simulation (this level of mismatch between schematic and layout performance is quite normal in analog design), attesting to the quality of the layout. Moreover, the layout respects symmetry constraints that are considered important by analog designers to guard against parasitic mismatch due to systematic variability. For both layouts, the performance of the ALIGN-generated layout is very close to that of the manual layout.

For a set of wireline circuits, Table 3 shows a comparison between the performance of the ALIGN-generated layout and a hand-crafted manual layout and demonstrates that the performance of both layouts is comparable.

**THIS ARTICLE SUMMARIZES** the current state of the ALIGN flow for automated analog layout synthesis. ALIGN is open-source and may be downloaded and used freely [24]. Currently, the project has seen about 24 months of development, and can already synthesize layouts for a wide variety of analog circuits. It is expected that the capabilities of ALIGN will be enhanced significantly over the next few years, handling more sophisticated circuits, more complex constraints, and improved software robustness. The inherent hierarchical approach adopted by ALIGN is key to ensuring scalability of the software to larger designs in the future, while also providing high-quality solutions. ■

### Acknowledgments

This work was supported in part by the DARPA IDEA program under SPAWAR Contract N660011824048.

### References

- [1] K. Kunal et al., "ALIGN – open-source analog layout automation from the ground up," in *Proc. ACM/IEEE Design Autom. Conf.*, 2019, pp. 1–4.
- [2] R. Harjani, R. A. Rutenbar, and L. R. Carley, "OASYS: A framework for analog circuit synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 8, no. 12, pp. 1247–1266, Dec. 1989.
- [3] J. M. Cohn et al., "KOAN/ANAGRAM II: New tools for device-level analog placement and routing," *IEEE J. Solid-State Circuits*, vol. 26, no. 3, pp. 330–342, Mar. 1991.
- [4] H. E. Graeb, Ed., *Analog Layout Synthesis: A Survey of Topological Approaches*. New York, NY, USA: Springer, 2010.
- [5] M. Eick et al., "Comprehensive generation of hierarchical placement rules for analog integrated circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 2, pp. 180–193, Feb. 2011.
- [6] H.-C. Ou, H.-C.-C. Chien, and Y.-W. Chang, "Simultaneous analog placement and routing with current flow and current density considerations," in *Proc. ACM/EDAC/IEEE Design Autom. Conf.*, May 2013, pp. 1–6.
- [7] Q. Ma et al., "Simultaneous handling of symmetry, common centroid, and general placement constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 1, pp. 85–95, Jan. 2011.
- [8] C.-Y. Wu, H. Graeb, and J. Hu, "A pre-search assisted ILP approach to analog integrated circuit routing," in *Proc. IEEE Int. Conf. Comput. Design*, Oct. 2015, pp. 244–250.
- [9] K. Kunal et al., "GANA: Graph convolutional network based automated netlist annotation for analog circuits," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Mar. 2020, pp. 55–60.
- [10] K. Kunal et al., "A general approach for identifying hierarchical symmetry constraints for analog circuit

- layout,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2020, pp. 1–8.
- [11] Y. Li et al., “Exploring a machine learning approach to performance driven analog IC placement,” in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Jul. 2020, pp. 24–29.
- [12] Y. Li et al., “A customized graph neural network model for guiding analog IC placement,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2020, pp. 1–9.
- [13] P.-H. Wu et al., “A novel analog physical synthesis methodology integrating existent design expertise,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 2, pp. 199–212, Feb. 2015.
- [14] G. Soto. (2017). *Discover the Power of OPAL, a New High-Level Design Rule Modeling Language*. Talk at Si2 Event at DAC. [Online]. Available: <http://www.si2.org/events/opal>
- [15] C.-H. Lin et al., “High performance 14 nm SOI FinFET CMOS technology with 0.0174  $\mu\text{m}^2$  embedded DRAM and 15 levels of Cu metallization,” in *IEDM Tech. Dig.*, Dec. 2014, pp. 3–8.
- [16] A. Steegen et al., “65 nm CMOS technology for low power applications,” in *IEDM Tech. Dig.*, Dec. 2005, pp. 64–67.
- [17] E. Malavasi et al., “Automation of IC layout with analog constraints,” *IEEE Trans. Comput.-Aided Design Integr.*, vol. 15, no. 8, pp. 923–942, Aug. 1996.
- [18] S. Bhattacharya et al., “Correct-by-construction layout-centric retargeting of large analog designs,” in *Proc. ACM/IEEE Design Autom. Conf.*, 2004, pp. 139–144.
- [19] N. Lourenco et al., “LAYGEN—Automatic layout generation of analog ICs from hierarchical template descriptions,” in *Proc. Ph.D. Res. Microelectron. Electron.*, 2006, pp. 213–216.
- [20] L. Zhang, U. Kleine, and Y. Jiang, “An automated design tool for analog layouts,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 8, pp. 881–894, Aug. 2006.
- [21] E. Yilmaz and G. Dundar, “Analog layout generator for CMOS circuits,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 1, pp. 32–45, Jan. 2009.
- [22] B. Xu et al., “MAGICAL: Toward fully automated analog IC layout leveraging human and machine intelligence: Invited paper,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2019, pp. 1–8.
- [23] T. Ajayi et al., “Toward an open-source digital flow: First learnings from the OpenROAD project,” in *Proc. ACM/IEEE Design Autom. Conf.*, 2019, pp. 1–4.
- [24] Software Repository. *ALIGN: Analog Layout, Intelligently Generated From Netlists*. Accessed: Aug. 1, 2020. [Online]. Available: <https://github.com/ALIGN-analoglayout/ALIGN-public>

**Tonmoy Dhar** is currently pursuing a PhD with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN. His research interests are in reliability analysis and layout synthesis of analog and mixed-signal circuits. Dhar has a BSc in electrical and electronic engineering from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

**Kishor Kunal** is currently pursuing a PhD with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN. His research interests are in computer-aided design (CAD) of VLSI systems, analog layout automation, and machine learning. Kunal has a BTech from IIT Kharagpur, Kharagpur, India.

**Yaguang Li** is currently pursuing a PhD with Texas A&M University, College Station, TX. His current research interests include analog IC design and machine learning. Li has a BS from the North China University of Technology, Beijing, China (2015) and an MS from ShanghaiTech University, Shanghai, China (2018).

**Meghna Madhusudan** is currently pursuing a PhD with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN. Her current research interests include analog design and layout automation. Madhusudan has a BE from Visvesvaraya Technological University, Belgaum, India (2017).

**Jitesh Poojary** is currently pursuing a PhD with the University of Minnesota, Minneapolis, MN. His research interests are MIMO receivers and mm-Wave receivers. Poojary has a BE from the University of Mumbai, Mumbai, India (2012) and an MTech from the Indian Institute of Technology, Delhi, New Delhi, India (2015).

**Arvind K. Sharma** is currently a Post-Doctoral Associate with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN. His current research interests include device physics, circuit device interaction, layout automation, and variability-aware circuit design. Sharma has a PhD from IIT Roorkee, Roorkee, India (2018).

**Wenbin Xu** is currently a Software Engineer with Cadence Design Systems Inc., San Jose, CA. His research interests include computer-aided design,

approximate computing, and hardware security. Xu has BS and MS in electronic engineering from Shanghai Jiao Tong University, Shanghai, China (2008) and (2011), respectively, and a PhD in computer engineering from Texas A&M University, College Station, TX (2019).

**Steven M. Burns** is a Senior Principal Engineer with Intel Labs, Hillsboro, OR. His current research interests include analog layout synthesis, transformation-based design environments, advanced synthesis algorithms and methods, physical synthesis of standard cells, and CAD for future process technologies. Burns has a PhD from the California Institute of Technology, Pasadena, CA.

**Ramesh Harjani** is the E.F. Johnson Professor with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN. His research interests include analog/RF circuits for wireless communication. Harjani has a PhD from Carnegie Mellon University, Pittsburgh, PA (1989). He is a Fellow of IEEE.

**Jiang Hu** is a Professor of electrical and computer engineering with Texas A&M University, College Station, TX. His research interests include VLSI physical design and machine learning algorithms. He is a Fellow of IEEE.

**Desmond A. Kirkpatrick** is a Principal Engineer with Intel Labs, Hillsboro, OR. Kirkpatrick has an SB in electrical engineering from the Massachusetts Institute of Technology, Cambridge,

MA and a PhD in electrical engineering and computer sciences from the University of California, Berkeley, Berkeley, CA.

**Parijat Mukherjee** is a Research Scientist with Intel Labs, Hillsboro, OR. His research interests include mixed-signal circuit analysis, analog layout synthesis, presilicon prototyping of silicon debug features, provably correct firmware development, and formal methods for platform security verification. Mukherjee has a PhD in computer engineering from Texas A&M University, College Station, TX (2015).

**Soner Yaldiz** has been with Intel Labs, Hillsboro, OR, since 2012. His research focuses on computer-aided design of electrical circuits and systems. Yaldiz has a BS from Sabanci University, Istanbul, Turkey (2004), an MS from Koc University, Istanbul, Turkey (2006), and a PhD from Carnegie Mellon University, Pittsburgh, PA (2012).

**Sachin S. Sapatnekar** is the Henle Chair Professor in electrical and computer engineering and a Distinguished McKnight University Professor with the University of Minnesota, Minneapolis, MN. He is a Fellow of IEEE and ACM.

■ Direct questions and comments about this article to Sachin S. Sapatnekar, Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA; sachin@umn.edu.

# MAGICAL: An Open-Source Fully Automated Analog IC Layout System from Netlist to GDSII

**Hao Chen, Mingjie Liu, Biying Xu,  
Keren Zhu, Xiyuan Tang, Shaolan Li,  
Yibo Lin, Nan Sun, and David Z. Pan**

The University of Texas at Austin

*Editor's notes:*

This article presents MAGICAL, which is a fully automated analog IC layout system. MAGICAL takes a netlist and design rules as inputs, and it produces the final GDS layout in a fully automated fashion.

—Sherief Reda, Brown University

—Leon Stock, IBM

—Pierre-Emmanuel Gaillardon, University of Utah

■ **THE EXPANDING MARKETS** of emerging applications, including the Internet of Things (IoT), 5G networks, advanced computing, healthcare electronics, etc., create large demands for analog and mixed-signal (AMS) integrated circuits. This increasing demand calls for a shorter design cycle and time-to-market. When compared with the tremendous advancements in digital IC layout design automation tools, analog IC layout still remains a heavy manual, time-consuming, and error-prone task. This is due to its high design flexibility and sensitive impact on the circuit performance by even minor changes in the layout implementation.

*Digital Object Identifier 10.1109/MDAT.2020.3024153*

*Date of publication: 14 September 2020; date of current version: 8 April 2021.*

Traditional analog layout synthesis tools rely on various heuristic constraints to guide the layout generation process [1]. These heuristics are based on human layout techniques and enforced during the placement

of devices and routing. Heuristic constraint-based methods face extreme difficulties in practical design flows, where handcrafted constraints are often design and technology dependent, lacking flexibility and generalization when meeting the detailed requirements of different scenarios. There is also the challenge of hard-encoding all such constraints in a legal procedure, especially when numerous contradictory constraints are present. Analytical approaches attempt to uncover the layout design tradeoffs either by deriving closed-form equations in evaluating the layout-dependent effects or sensitivity analysis simulations. With increased device scaling, analytical sensitivity estimates of parasitics and mismatch over performance are no longer accurate.

The most difficult thing above all is the limited availability of analog design tools. In contrast to the booming community of machine learning, where

popular frameworks and data sets are open-sourced, easily available, and heavily relied upon in research by the academic community, commonly used tools in analog design are largely proprietary. The lack of implemented frameworks, benchmark circuit data sets, and publicly available process design kits (PDKs), severely restrict the reproducibility of current research results and impede further improvement and extended research.

In this article, we present our work MAGICAL, a fully automated, end-to-end analog IC layout framework that generates a completed layout from a circuit netlist. Implemented modules include symmetry constraint generation, placement, and routing. These modules are implemented in C++ for optimal software performance, and off-the-shelf with user-friendly Python interface available. The source code<sup>1</sup> is released on GitHub with a number of sanitized benchmark circuits provided.<sup>2</sup> The layouts completed by MAGICAL are validated using industrial standard verification tools, demonstrating circuit performances close to those handcrafted by experienced designers.

Compared with prior on procedural layout generators, such as Berkeley analog generator (BAG) [2], MAGICAL reduces the cost of codifying specific constraints and detailed layout implementation such as circuit floorplan and routing topology. This is achieved with automated symmetry extraction leveraging pattern matching and graph similarity, and area and wirelength driven placement and routing optimization kernel. MAGICAL is also extensible to

handle custom constraints. We also present several of our research and findings that build upon the MAGICAL framework, including applied machine learning techniques, custom constraints, specific design considerations, and leveraging post layout simulation results for performance modeling and optimizations in the “Extensions based on MAGICAL” section. Moreover, we hope to promote research and progress in the analog design automation community, where future researchers could embed new heuristics and algorithms leveraging our framework. Our tool also complements other existing design automation tools in the open-source community [2]–[4].

## Magical framework

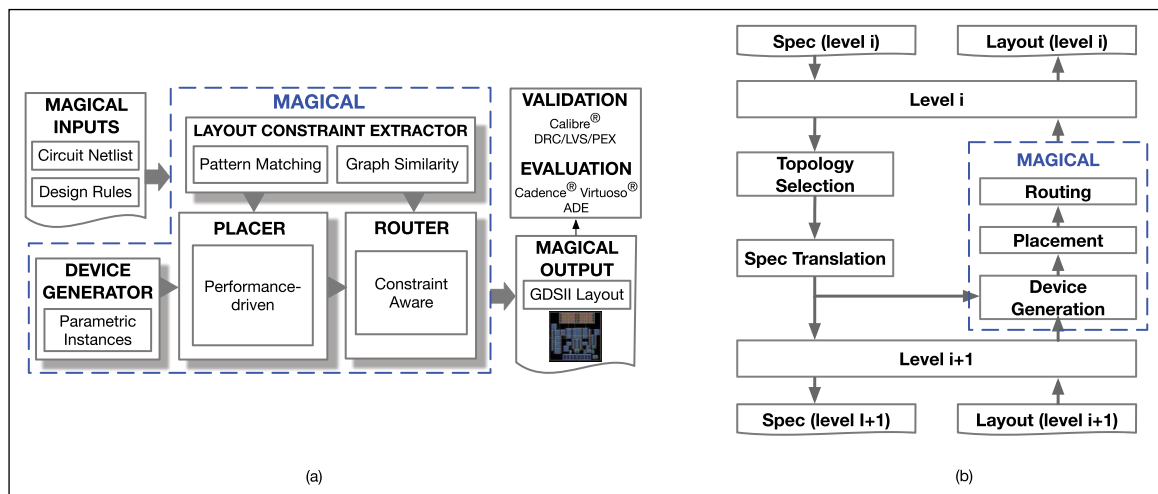
The overall flow of MAGICAL is shown in Figure 1. It takes an unannotated circuit netlist and design rules as inputs, and produces a complete GDSII layout as output fully automatically without human designers in the loop. The entire flow consists of four major modules, with each module being independent with a user-friendly Python interface. The design rules and the extracted layout constraints are honored throughout the entire back-end flow.

## Framework methodology and software architecture

The MAGICAL system includes several individual submodules, i.e., layout constraint extractor, device generator, placer, and router. A top-level MAGICAL flow integrates the individual components and manage the physical synthesis of the

<sup>1</sup><https://github.com/magical-eda/MAGICAL>

<sup>2</sup><https://github.com/magical-eda/MAGICAL-CIRCUITS>



**Figure 1. MAGICAL framework. (a) MAGICAL submodules. (b) MAGICAL hierarchical flow.**

circuits. Figure 1 shows the architecture of the MAGICAL layout system. This architecture is rooted upon the principle of divide-and-conquer nature of the circuit design and motivated by the purposes of creating an extensible and flexible open-source environment.

The designs of complex analog systems, such as phase-locked loops or analog-to-digital converters (ADCs), are typically decomposed into smaller building blocks (e.g., comparators, filters, or amplifiers). Human designers adopt a top-down design methodology, where system-level performance is translated to lower-level building block specifications. The layout design process, on the other hand, is done bottom-up. The building block circuits are first implemented and the performance is optimized and verified. The system is then built with the building blocks. This divide-and-conquer practice avoids optimizing the whole system at once and decomposes it into smaller and more traceable subproblems. MAGICAL adopts this design methodology. The whole physical synthesis is decomposed into multilevel homogeneous subproblems. The top MAGICAL flow manages and schedules the subproblems, while the individual components build the layout in the bottom-up manner.

The separation of submodules also allow easy extension to the default MAGICAL flow. Conventionally, different components in a physical design flow is connected by scripts and exchangeable files, for example [4]. However, the complicated scripting potentially make interaction between different components difficult and hinder the flexibility on the flow. On the other hand, the popularity of machine learning algorithm also raises question that whether the conventional software methodology is suitable for the emerging framework. MAGICAL is developed for easy adaption of new components and changes on the flow. Although the main optimizing kernels are developed in C++ for better efficiency, each submodule has a Python interface. And the top level, MAGICAL flow uses Python interface to assign the subproblems. Such architecture is friendly to adoption of machine learning framework and makes interactions easy. In fact, there have been success on the extensions of default MAGICAL. The “Experimental results” section gives case studies of several MAGICAL extensions.

In the rest of this section, the default MAGICAL submodules are explained in the detail.

## Parametric device generation

Before running the core layout flow, the device generation step first generates the layout of the devices and extracts their pins to facilitate the subsequent placement and routing stages.

The generated GDSII layout is correct by construction based on the design rules. MAGICAL currently supports numerous different device types, including pMOS, nMOS, metal–oxide–metal (MOM) capacitors, and poly resistors. The automatic parametric device generation considers the number of fingers for transistors, the number of segments for resistors, the metal layers for MOM capacitors, etc.

The MAGICAL framework is also extensible for custom-designed devices, digital standard cells, or even subcircuits such as capacitor or resistor arrays.

## Analog layout constraint extraction

The layout constraint extractor takes a circuit netlist as input and generates constraints to guide the later stages. Analog designs frequently use differential topologies to reject common-mode noise and enhance circuit robustness and performance [5]. Thus, correctly identifying symmetry constraints between sensitive devices are crucial for ensuring the quality of placement and routing.

The constraint extraction reads in the input netlist and generates constraints for placement and routing based on the circuit connections. A significant challenge for constraint extraction is in generating high-quality constraints and resolving constraint ambiguity. Since the characteristics of symmetry among devices and between building blocks are vastly different, we use different methods to generate symmetry constraints.

## Device symmetry

Only the symmetry constraints between devices need to be considered for building blocks. We adopt a method similar to the works of Eick et al. [6]. The building block circuit is abstracted into a graph. A pattern library of the commonly used differential topology of transistors is predefined. We use graph isomorphic algorithms to detect matching patterns on the building block circuits with the pattern library. The circuit graph is then traversed from the matched patterns to recognize new symmetry constraints of passive devices, self-symmetry devices, and symmetry routing constraints.

### System symmetry

System symmetry differs from device symmetry because on the system level, template libraries are difficult to generate, and graph isomorphic algorithms are expensive. Since the same building block could be referenced multiple times in system design, we propose to extract graphs that include the neighboring circuit topology of the building blocks to resolve the ambiguity. The extracted graphs are then compared using an efficient graph similarity metric leveraging spectral graph analysis [7]. Self-symmetry constraints and symmetry net constraints could also be extracted similar to the approach in device symmetry.

### Analog placement

Given the placement constraints and devices generated in the previous steps, we develop an analog placement engine. The placer places each device or building block in the layout satisfying the given constraints while optimizing for the wirelength and layout area.

The placement engine follows an analytical framework as in [8]. First, the global placement simultaneously optimizes multiple objectives in a nonlinear objective function to generate a rough legal placement. Then, the legalization step uses linear programming (LP) algorithm to legalize the global placement results honoring input constraints and design rules. Finally, another LP-based detailed placement is used to optimize the wirelength further.

### Analog routing

To determine the wire connections between all the placed devices while satisfying the design considerations for better circuit performance, a constraint-aware analog routing algorithm is applied.

In addition to connectivity and design rules, an analog routing problem is usually imposed with symmetric net constraints, which are specified to ensure matched nets routed symmetrically on some axes. In MAGICAL, the routing engine takes the constraints specification as an input from the layout constraint generator and honors the symmetric and self-symmetric requirement for matched nets.

Our routing framework divides the routing problem into two stages, global routing and detailed routing, similar to the standard digital routing flow. To generate a global routing solution, a sequential symmetry-aware grid-based A\* search routing engine is employed. The circuit is cut into unified grids whose

width and height are decided based on track width on the first metal layer. More specifically, the global routing engine divides the layout into a 3-D graph with grids as the vertices and the connection between neighboring grids as the edges. The capacity of each edge is calculated based on free space modeling and the actual congestion inside the grids. The symmetry-aware global routing algorithm using mirroring techniques then generates the solution for each net.

Given the global routing results as guidance, the detailed routing engine completes the final routing and assigns metal wire geometries. In contrast to digital circuits, analog designs usually have various metal widths and multiple via cuts for different nets, along with a number of special specifications such as symmetric constraints. To solve the detailed routing problem, the symmetric-aware A\* path searching algorithm is performed while satisfying design rules and specific requirements for each net (e.g., wire width). After the detailed routing stage, the final GDSII format layout file is exported.

### Extensions based on magical

In this section, we present several of our research that builds on top of the default MAGICAL framework. We hope to demonstrate the extensibility of the framework and the increased efficiency in the development of novel algorithms and methods by leveraging MAGICAL.

#### Machine learning guided well generation

Generating wells and inserting contacts are required in layout synthesis. The default MAGICAL framework generates separate NWELL contacts for each individual pMOS devices. This provides superior device isolation and reduction in well proximity effect at the increased overhead of area.

However, individual well contacts for transistor is seldom adopted in manual layout strategy. Sharing the same body connection leads to more compact layout. *WellGAN* [9] provides an alternative to the individual well contact. It formulates the well generation problem into a computer vision task and supervisedly learns how human designers draw the well using generative adversarial network (GAN). After the training, the GAN model can generate images of well based on the placement results and a legalization routine can draw the well and insert the contacts based on the model prediction.



## Machine learning guided analog routing

In the default MAGICAL, the routing is targeting wirelength and geometric constraints. However, the router can be easily changed with a machine learning guided algorithm.

Zhu et al. [10] proposed *GeniusRoute* is developed upon the MAGICAL router. It attempts to learn the manual analog routing strategy by leveraging a variational autoencoder (VAE) model. Similar to *WellGAN*, *GeniusRoute* formulates the learning into an image generation task. The VAE models learn where human designers are likely to route the nets in analog circuits and generate prediction on unseen new circuits. The router is modified to adapt the routing prediction.

## Analog placement quality prediction

In the default MAGICAL framework, the analog analytical placement engine only optimizes the wire-length and area. While the wire-length might be a natural surrogate for performance and highly correlated with the power and performance of digital circuits, analog layout performance rarely has strong relevance to the total wire-length. Thus, to satisfy post layout performance requirements and achieve design closure, a feedback loop from performance simulation to the design flow is needed in the development of practical layout synthesis tools.

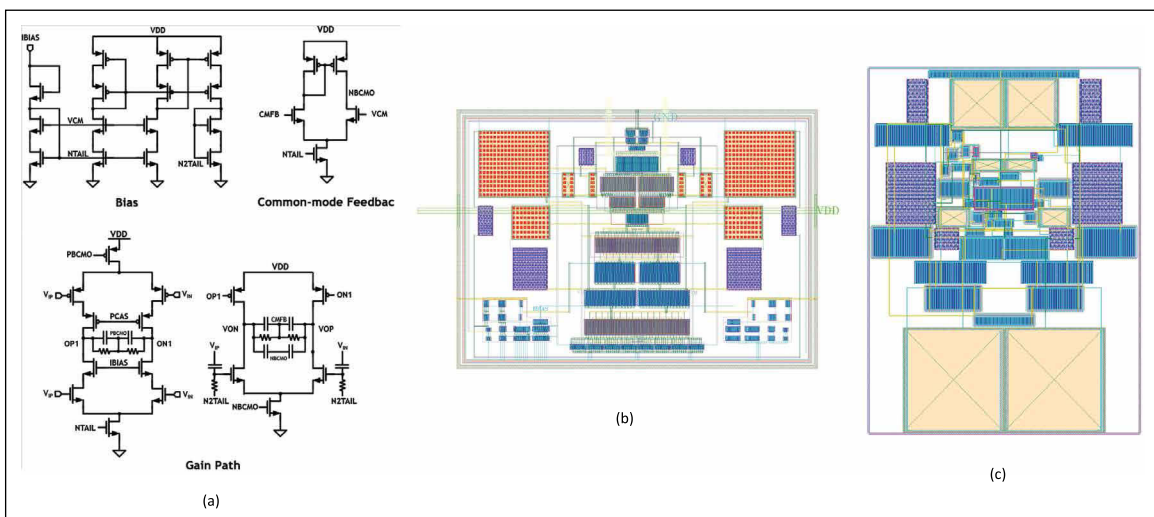
To reduce the design exploration runtime and limit the number of performance simulations, the work of Liu et al. [11] proposes to predict of the layout quality early in the layout design flow. To overcome the difficulty of obtaining high-quality human

layout training data, MAGICAL was used to generate multiple layout solutions for the same circuits automatically. An effective placement feature extraction method with 3-D convolution neural network was developed for effective placement quality prediction. The number of training data needed to obtain satisfactory classification results was significantly reduced by leveraging transfer learning.

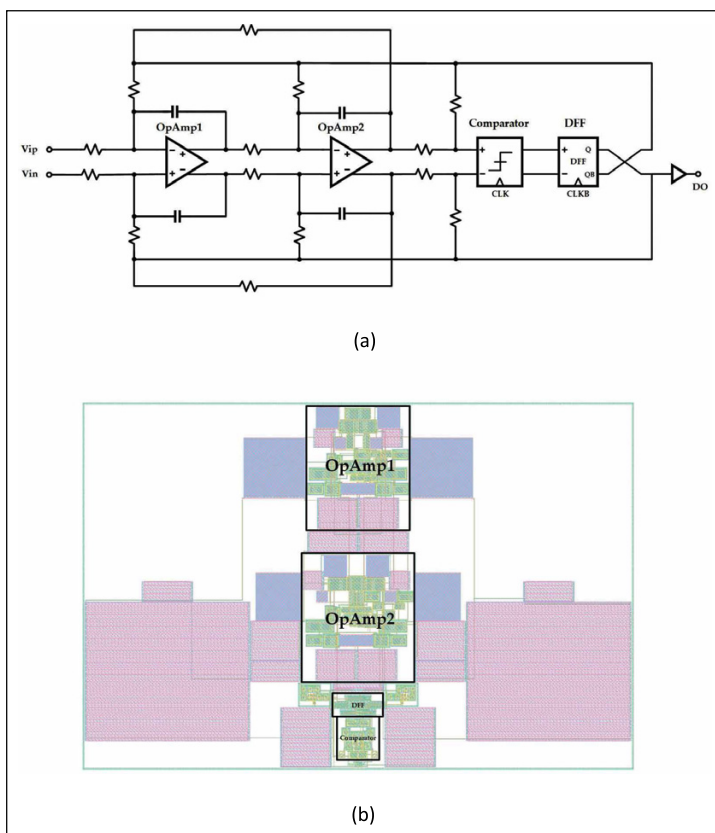
## Efficient layout synthesis with Bayesian optimization

The works of Liu et al. [12] extended the default MAGICAL framework considering custom constraints and design-specific considerations. It leverages post layout simulations in driving the layout implementation process for building block circuits. By formulating the performance optimization as a multiobjective black-box optimization problem, it closes the design loop and guarantees post layout performance through iterative simulations and a data-efficient Bayesian optimization algorithm.

Since system-level transistor simulation is unaffordable, Liu et al. [12] optimized the system-level layout by extending the original MAGICAL framework to include custom constraints and design specific considerations. Specific constraints and considerations include net criticality, routing sequence, net spacing assignments, and regularized signal flow paths. The layout for a complete ADC system with regularized signal flow paths were generated, achieving close to schematic simulation results.



**Figure 2. OTA results. (a) Circuit schematic. (b) Manual layout. (c) MAGICAL layout.**



**Figure 3. CTDSM results. (a) System architecture. (b) MAGICAL layout.**

### Experimental results

The MAGICAL flow is implemented in Python and C/C++, and the experiments are performed on a Linux server with an 8-core 3.4-GHz Intel CPU and 32-GB memory. All designs are in TSMC 40-nm technology. The layout results are validated using Calibre DRC/LVS/PEX, and evaluated using Cadence Virtuoso ADE simulation environment.

The post-layout simulation results for two benchmark circuits, a two-stage operational transconductance amplifier (OTA), and a continuous time  $\Delta\Sigma$  modulator (CTDSM), are shown in Figures 2 and 3, respectively. The circuit performances of the OTA layout results generated by MAGICAL are compared against tape-out quality manual layouts by experienced analog IC designers, under the same test benches. The simulation results are shown in Table 1, where UGB stands for the unity gain bandwidth, PM denotes the phase margin, and CMRR denotes the common-mode rejection ratio. The simulation results for the CTDSM are shown in Table 2, where  $F_s$  denotes the sampling frequency, BW is the bandwidth, SNDR

**Table 1. OTA simulation results.**

Metrics	Gain (dB)	UGB (MHz)	PM (degree)	CMRR (dB)	Offset (mV)
Schematic	69.5	2192	73.5	-	-
Manual	69.8	1567	65.8	97.2	0.012
MAGICAL	69.7	1496	63.2	92.1	0.027

**Table 2. CTDSM simulation results.**

	Schematic	MAGICAL
Supply (V)	1.2	
$F_s$ (MHz)	320	
BW (MHz)	5	
SNDR (dB)	67.8	65.9
SFDR (dB)	84.7	80.5
Power (mW)	0.84	0.86
Area ( $\mu\text{m}^2$ )	-	9450

denotes the signal-to-noise and distortion ratio, and SFDR denotes the spurious-free dynamic range. The results demonstrate that MAGICAL can automatically generate validated layouts from unannotated circuit netlist (both Spectre and HSPICE format), and the post-layout performances are close to the schematic designs. Some performance metrics, including input-referred offset and CMRR, could be further improved by extensively considering layout dependent effects, minimizing coupling to sensitive nets, etc.

### Future directions

Being part of the open-source hardware/EDA ecosystem, the future development of the MAGICAL will both benefit from and contribute to the community. Although the existing components in different open-source EDA tools may have different algorithms and methodologies, there are some overlapping between their functionality. Both analog and digital layout automation flows share many common infrastructural components with MAGICAL. MAGICAL can learn from the recent emerging open-source EDA tools.

Besides the EDA tools, open-sourcing AMS circuit designs is another driving force for analog layout automation. On the one hand, lacking of training data has been a major challenge in machine learning-based EDA algorithm. On the other hand, the lack of a unified test circuit benchmark suite makes it difficult to evaluate and compare different analog EDA tools. Open-source designs will not only make it possible for the EDA tools to have common

evaluation metrics, but also provide training data for machine learning-based EDA algorithms.

While MAGICAL has demonstrated satisfactory results, it currently only minimizes post-layout circuit performance degradation implicitly by considering the analog layout constraints. Although direct optimization methods have been applied and demonstrated to be effective, the overhead of repetitive simulations is still expensive and impractical, especially for system-level designs. In the future research and development, MAGICAL will investigate into the performance-aware techniques, especially machine learning algorithms, throughout its entire flow.

Preliminary simulation results have also demonstrated the potential of generating entire system-level designs with MAGICAL. However, MAGICAL still needs to improve its current placement and routing algorithms for better design rule handling. Furthermore, there is still large room for improvement, especially for system-level designs, including circuit reliability, clock coupling mitigation, IR drop aware routing, and integration with digital flows. Generating tape-out quality layout designs proven with silicon chip measurements will be the future goal of MAGICAL.

**IN THIS ARTICLE**, we presented MAGICAL, an open-source fully automated end-to-end analog IC layout system from circuit netlists to GDSII layouts. Human and machine intelligence are strategically incorporated into MAGICAL by pattern matching and deep learning techniques. The circuit performances of the layouts completed by MAGICAL are close to those handcrafted by experienced designers, while the design cycle is shortened substantially. ■

## Acknowledgments

Hao Chen, Mingjie Liu, Biying Xu, and Keren Zhu contributed equally to this work. This work was supported in part by the National Science Foundation (NSF) under Grant 1704758 and in part by the DARPA IDEA program.

## References

- [1] M. P.-H. Lin, Y.-W. Chang, and C.-M. Hung, "Recent research development and new challenges in analog layout synthesis," in *Proc. 21st Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2016, pp. 617–622.
- [2] J. Crossley et al., "BAG: A designer-oriented integrated framework for the development of AMS circuit generators," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2013, pp. 74–81.
- [3] K. Kunal et al., "ALIGN: Open-source analog layout automation from the ground up," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–4.
- [4] T. Ajayi et al., "Toward an open-source digital flow: First learnings from the OpenROAD project," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–4.
- [5] B. Razavi, *Design of Analog CMOS Integrated Circuits*, 1st ed. New York, NY, USA: McGraw-Hill, 2001.
- [6] M. Eick et al., "Comprehensive generation of hierarchical placement rules for analog integrated circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 2, pp. 180–193, Feb. 2011.
- [7] M. Liu et al., "S3DET: Detecting system symmetry constraints for analog circuits with graph similarity," in *Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2020, pp. 193–198.
- [8] B. Xu et al., "MAGICAL: Toward fully automated analog IC layout leveraging human and machine intelligence: Invited paper," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–8.
- [9] B. Xu et al., "WellGAN: Generative-adversarial-network-guided well generation for analog/mixed-signal circuit layout," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.
- [10] K. Zhu et al., "GeniusRoute: A new analog routing paradigm using generative neural network guidance," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–8.
- [11] M. Liu et al., "Towards decrypting the art of analog layout: Placement quality prediction via transfer learning," in *Proc. Design Autom. Test Eur. Conf. Exhibition (DATE)*, Mar. 2020, pp. 496–501.
- [12] M. Liu et al., "Closing the design loop: Bayesian optimization assisted hierarchical analog layout synthesis," in *Proc. DAC*, Jul. 2020, pp. 1–6.

**Hao Chen** is currently pursuing a PhD with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX. His research interests include physical design, CAD for analog/mixed signal designs, logic synthesis, and emerging technology. Chen has a BS in electrical engineering from the National Taiwan University, Taipei, Taiwan (2019).

**Mingjie Liu** is currently pursuing a PhD with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX. His research interests include applied machine learning

for design automation and layout design automation for analog and mixed-signal integrated circuits.

**Biying Xu** is currently a Lead Software Engineer with Cadence Design Systems, San Jose, CA. Her research interests include physical design automation for digital, analog, and mixed-signal integrated circuits. Xu has a PhD from the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX (2019).

**Keren Zhu** is currently pursuing a PhD with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX. His research focuses on VLSI CAD. Zhu has a BS in electrical engineering with the highest distinction from the University of Wisconsin-Madison, Madison, WI.

**Xiyuan Tang** is currently a Postdoctoral Researcher with The University of Texas at Austin, Austin, TX. His research interests include digitally assisted data converters, low-power mixed-signal circuits, and analog data processing. Tang has a PhD in electrical engineering from The University of Texas at Austin (2019).

**Shaolan Li** is currently an Assistant Professor with Georgia Tech, Atlanta, GA. His research interests include analog and mixed-signal IC design, solid-state medical imaging platform, and artificial

intelligence hardware. Li has a PhD from The University of Texas at Austin, Austin, TX (2018).

**Yibo Lin** is an Assistant Professor with the Computer Science Department, Peking University, Beijing, China. His research interests include physical design, machine learning, and GPU acceleration. Lin has a PhD from the Electrical and Computer Engineering Department, The University of Texas at Austin, Austin, TX (2018).

**Nan Sun** is currently a Professor with the Department of Electronic Engineering, Tsinghua University, Beijing, China. His current research interests include analog, mixed-signal, and RF integrated circuit design, and analog circuit design automation.

**David Z. Pan** is currently a Silicon Laboratories Endowed Chair of electrical engineering with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX. His research interests include cross-layer nanometer IC design for manufacturability, reliability, security, machine learning and hardware acceleration, design/CAD for analog/mixed signal designs and emerging technologies.

■ Direct questions and comments about this article to David Z. Pan, Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712 USA; [dpan@ece.utexas.edu](mailto:dpan@ece.utexas.edu).

# An Open-Source EDA Flow for Asynchronous Logic

**Samira Ataei, Wenmian Hua, Yihang Yang,  
and Rajit Manohar**

Yale University

**Yi-Shan Lu, Jiayuan He, Sepideh Maleki,  
and Keshav Pingali**

University of Texas at Austin

*Editor's notes:*

This article presents an open-source EDA flow for digital asynchronous circuits, capable of supporting many different families of asynchronous circuit families from logic synthesis all the way down to GDSII.

—Sherief Reda, Brown University

—Leon Stock, IBM

—Pierre-Emmanuel Gaillardon, University of Utah

■ **SCALABLE COMPUTER SYSTEMS** are designed as a collection of modular components that communicate through well-defined interfaces. The interfaces must be robust to delays and uncertainty in the physical implementation of communication. This view applies to computer systems at many levels of abstraction. The Internet is a collection of communicating computers with message passing through the Internet protocol. A modern datacenter is a collection of servers that communicate via message passing over commodity network hardware. Even large software systems consist of a collection of modules that use well-defined application programming interfaces (APIs) to communicate. Almost all computer systems disciplines have made the wise choice to partition their problem into components that communicate via protocols that *are independent of their physical realization—such as timing, energy, or size.*

Digital Object Identifier 10.1109/MDAT.2021.3051334

Date of publication: 13 January 2021; date of current version:  
8 April 2021.

However, in current chip designs, this modular approach is abandoned in favor of global synchrony. A global synchronization signal (the “clock”) dictates the time budget for every step of the computation—regardless of *what*

is being computed.

Although this clocked design paradigm dominates the design of computers today, engineers are struggling to preserve the fiction of simultaneity required by the clock, even within an individual chip. This struggle is an inevitable result of advancing technology. As transistors get smaller and faster, the delay of communication over wires dominates the cost of local computation with transistors. Such progress renders the clocked paradigm a poorer and poorer abstraction for chip design. Modern application-specific integrated chips (ASICs) are designed as a collection of small-clocked “islands” that communicate via interfaces that break the clocking abstraction.

To address this challenge, we are creating a collection of open-source electronic design automation (EDA) tools that isolate the designer from the details of the physical implementation technology, especially when it comes to delays and timing uncertainty.<sup>1</sup>

<sup>1</sup> It is not possible to entirely decouple the logical correctness of a design from timing to create completely delay-insensitive circuits [1], [2]. However, it is possible to make a very mild and local timing assumption that is easy to satisfy in practice [3].

The approach is based on an *asynchronous, modular, and hierarchical* design methodology for complex chips, and it permits component reuse from one technology to another with little or no modification. While individual modules of the chip can be clocked, the overall system uses an asynchronous integration approach to achieve modular composition. Hence, the EDA flow being developed supports a combination of timing styles in an integrated framework.

The algorithmic complexity of some of the important steps in an EDA flow is higher when analyzing asynchronous circuits, which can have a major impact on the overall runtime of the flow. To reduce the turn-around time for designs, we are implementing parallel versions of the key algorithms in the toolchain, using the Galois system described in the “Parallelism: The Galois framework” section. The Galois system supports the parallelization of irregular algorithms such as those in which the key data structures are graphs and hypergraphs. Since circuits can be viewed as hypergraphs, the Galois system is well-suited for this parallelization effort.

### Asynchronous logic: A unified approach

There are a large number of different asynchronous logic families. Historically, each of these families was developed by different research groups, with different terminology and design methodologies. Note that this is not that different from synchronous logic; any textbook on synchronous digital logic will describe a large number of options for synchronous design such as pseudo-nMOS logic, precharge logic, dual-rail domino logic, and self-resetting logic to name just a few options [4]. In addition, many circuit options for flip-flops and latches are also described, and the merits of each discussed. Heterogeneity of this nature is difficult to incorporate into any automated design flow.

Instead, over the years, the mainstream industrial strength ASIC flow that is provided by the major EDA vendors converged on a core synchronous EDA flow that supported a limited set of options. Today those options include flip-flop and (some) latch-based designs with excellent support for single-clock designs, and limited support for heterochronous designs. Circuit options were ignored in favor of standard-cell libraries with hand-optimized circuit layout for individual cells (an individual CMOS gate or a small collection of gates). This push was driven by industry and resulted in the standardization of what is viewed as the commercially supported ASIC

flow today. Standardization led to interoperability and a rich intellectual property (IP) ecosystem. Modern ASIC design is as much about system integration as it is about writing the detailed hardware description language that describes the chip.

The same cannot be said about asynchronous logic. The convergence that occurred in the synchronous domain did not occur in the asynchronous logic domain, and hence the EDA landscape for asynchronous logic is quite bleak in comparison. While there have been many academic tools developed for individual steps needed to go from a high-level description of an asynchronous design to a chip implementation, only a small number of complete flows have been developed. Example flows developed for asynchronous design include Haste [5] and Balsa [6], and more recently Proteus [7]. Each of these flows supports a restricted style of asynchronous design and uses commercial synchronous tools for physical design automation. Since the synchronous physical design tools do not have a correct view of timing for asynchronous logic, conservative work-arounds are used to constrain the design to ensure a valid implementation. This approach has also been adopted by many academic groups to leverage the investment made by the commercial EDA industry.

### Unified approach to asynchronous logic

Instead of developing an asynchronous logic flow that only supports a particular flavor of asynchronous logic (a survey of some approaches can be found in [8]–[10]), the approach we adopt tackles the problem of heterogeneity of design styles and circuit approaches.

### Commonalities

Since one of the goals of adopting an asynchronous approach is to create a modular design style, all the different approaches generally share the characteristic that a design is partitioned into a collection of concurrently operating hardware modules (we call them *processes*, adopting the term from the concurrent computing literature) that communicate with each other using well-defined protocols on wire bundles (we call the bundles *channels*). Channels are used both for exchanging information as well as synchronization between processes. To support this abstraction, we use a hardware specification notation called CHP (for communicating hardware processes), an extension of Hoare’s communicating

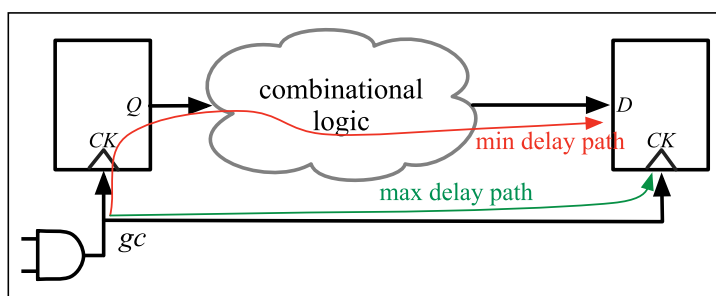
sequential processes (CSP) language [11]. This is the highest level of abstraction and corresponds to a behavioral description of the asynchronous chip. A CHP description can be translated into different asynchronous logic families.

## Differences

The origin of a large number of differences between varying approaches to asynchronous design stems from differing assumptions about timing. Purely delay-insensitive circuits make no assumptions about the delays of gates or wires. This is an extremely robust approach and requires at least two output gates to be expressive enough [12]. No timing constraints have to be specified in this case. Quasi-delay-insensitive circuits and speed independent circuits require a timing assumption called the isochronic fork [2], which translates to a wire delay versus path delay assumption [3]. Bundled-data communication protocols require one wire (the request) to be slower than the data wires. Even though there is a large range of timing requirements, many of them can be expressed using a generalization of two approaches to specific timing constraints. The first is the approach used by synchronous timers to express hold time constraints for generated clocks. If a signal  $gc$  is a generated clock, and it is connected to two flip-flops, then the hold time constraint for the generated clock is a “point of divergence” constraint, where—starting from a root point—the maximum delay through one path has to be slower than the minimum delay through another path (see Figure 1). If the actual delay values are known, then `.sdc` constraints like `set_min_delay` and `set_max_delay` can be used to constrain the maximum and minimum delay on two paths so that they are ordered as required by the hold time constraint [13].

A second approach used by the asynchronous design community is generalized relative timing [14]. In this approach, constraints are specified on *signal transition events*. Hence, a point-of-divergence constraint would be expressed by using  $gc \uparrow$  as the anchor event and then using signal transitions at the input to the flip-flop from Figure 1. The challenge with using events (unlike paths in the synchronous case) is that events might have data-dependent occurrences. Also, if a signal can have switching hazards, it might result in more than one event corresponding to the change in the point of divergence.

To address these issues, we introduce a general version of both these notions that we call *timing forks*.



**Figure 1. Hold-time point-of-divergence constraint for generated clock  $gc$ .**

A timing fork resembles a point-of-divergence constraint, except that it need not be a point of divergence. A timing fork  $a+ : b- < c+$  is a constraint that specifies an *error predicate*. In any execution of the circuit, if the sequence  $a+$  followed by  $c+$  followed by  $b-$  occurs *without an intervening*  $a+$ , then the constraint is violated. This timing fork is based on recent research in the distributed systems literature that argues that events can be ordered only if there is a visible set of timing forks [15]. For isochronic forks, we need a notion of the near and far end of a wire; we augment the syntax of timing forks so that we can specify an input rather than the output of a gate. It should be clear that timing forks can express point-of-divergence constraints in a straightforward fashion. What is less obvious is that even when there is not a local point-of-divergence, one must exist at some point in the execution history in the absence of any absolute notion of time [15].

Bundled data asynchronous communication uses a request line and a data bundle, where the data bundle signals have to be stable once the request goes high in the most straightforward version of the protocol. The communication is initiated by control logic; suppose that  $c$  goes high to initiate the communication, the request signal is  $req$ , and  $di$  is one of the data signals. The timing constraint would be written  $c+ : di < req+$ . Note that a scenario where  $di$  does not change is also permitted by the meaning of a timing fork. We distinguish between multiple uses of the same set of bundled data wires because  $c+$  will occur between each use of the wires.

The synchronous timing constraint shown in Figure 1 can be expressed using this notation in the following way: “ $gc+ : FF.CK+ < FF.D.$ ” This states that any changes in the D pin of FF must occur after the CK pin goes high, where FF is the instance name of the flip-flop on the right-hand side in Figure 1.

Since our timing constraints can specify both synchronous and asynchronous timing constraints, the flow supports a design with a mixture of synchronous and asynchronous components. In particular, timing forks are sufficiently expressive to describe the timing constraints needed for quasi-delay-insensitive circuits, GasP pipelines [16], bundled data communication, and high-speed transition signaling pipelines [17] to name a few circuit families.

#### Asynchronous circuit toolkit framework

The flow we have developed includes a design language called asynchronous circuit toolkit (ACT). ACT is a hierarchical design language that includes communication channels and encoded data values as first-class objects. The language supports representing circuits at multiple levels of abstraction, including CHP, gate-level, and transistor-level descriptions. ACT is strongly typed, and the type system is used to track and specify many design constraints that traditionally are externally specified in commercial flows (e.g., using .sdc files). Timing forks can be included as part of the logic specification.

By using an integrated language that can be used at multiple levels of abstraction, we preserve the relationships between different levels of abstraction in the design throughout the design flow. These relationships are captured using ACT's type system. Timing constraints between modules can be specified in the type-definition of communication channels—i.e., in the interface specification captured by the type signature of a component. Design tools can be viewed as transformations in the ACT framework. For example, logic synthesis elaborates a CHP-level description of a module into a gate-level description of the *same module* without changing its interface. Hence, constraints generated by logic synthesis are made available to the rest of the flow as part of the ACT language, and hence are visible to both timing analysis and place-and-route tools.

The history of this language can be traced to the MiniMIPS project at Caltech, Pasadena, CA (1994–1999), where a simplified version of ACT was developed by Manohar to manage the design complexity of the MiniMIPS asynchronous processor design [18]. This language, called CAST (for Caltech Asynchronous synthesis tools), was used to implement a microprocessor at the gate level of abstraction. This language was used both by Manohar's group at Cornell as well as a startup company (Fulcrum Microsystems). CAST continued

to evolve at Fulcrum Microsystems, which was eventually acquired by Intel in 2012; as part of their development, Fulcrum also developed the Proteus flow [7]. The ACT language was created in 2005 as an evolution of CAST, and to overcome some of its limitations. This language was also used by Achronix Semiconductor, and to develop a number of chips at Cornell and Yale. An open-source version of this early version of ACT was also released [19]; these early versions of ACT were only designed to support quasi-delay-insensitive asynchronous circuits.

The current ACT language [20] is the result of an evolution over almost three decades of research in asynchronous design grounded in the implementation of over a dozen asynchronous VLSI chips ranging in complexity from 0.5M transistors [21] to 5.4B transistors [22], and in technologies ranging from 0.6- $\mu\text{m}$  CMOS to 28-nm CMOS. It is general enough to be able to express a wide range of asynchronous logic families and is the basis for the open-source EDA flow described in the “Open-source flow” section.

#### Keeping designs open

Our implementation of the ACT framework includes a number of configuration files. We have partitioned these files into two disjoint sets: 1) technology-independent and 2) technology-specific. The information in the technology-specific files corresponds to items that may be covered by nondisclosure agreements with foundries, and thus may not be distributed without the appropriate agreements in place. However, the goal is to ensure that the vast majority of the design can be distributed without reference to the details of the underlying technology. Hence, while some of our tools cannot operate without detailed information from the foundry, the original logical design specified in the ACT framework can be distributed without any embedded foundry-specific information because this information is isolated to an input configuration file.

#### Integrating synchronous logic and other tools

The ACT tools are focused on supporting asynchronous logic families. However, we expect that a complex system would require integration with other logic styles—in particular, synchronous logic. Since commercial EDA tools provide outstanding support for design styles commonly used by industry, we do not directly target synchronous logic in the ACT framework. Instead, we provide support for importing



a design as a Verilog netlist. Importing synchronous logic or asynchronous logic from other flows into the ACT framework also requires that timing constraints for imported signals be specified using timing forks.

Module-level integration is the most straightforward way to import a Verilog netlist; this also permits a designer from maximizing the use of mainstream tools. Our timing analysis engine will determine that the synchronous logic and asynchronous logic are in different timing domains [23], and simply report timing for the different components separately. This is analogous to the scenario of an unrelated clock domain crossing that can occur in conventional EDA tools.

Asynchronous logic generated by any other approach can also be imported as long as the design is specified as a Verilog netlist, and the required timing constraints specified as timing forks. The ACT language also supports direct specification of logic using gates specified as pull-up and pull-down networks, by passing the CHP level of circuit description and the Verilog import process.

### Parallelism: The Galois framework

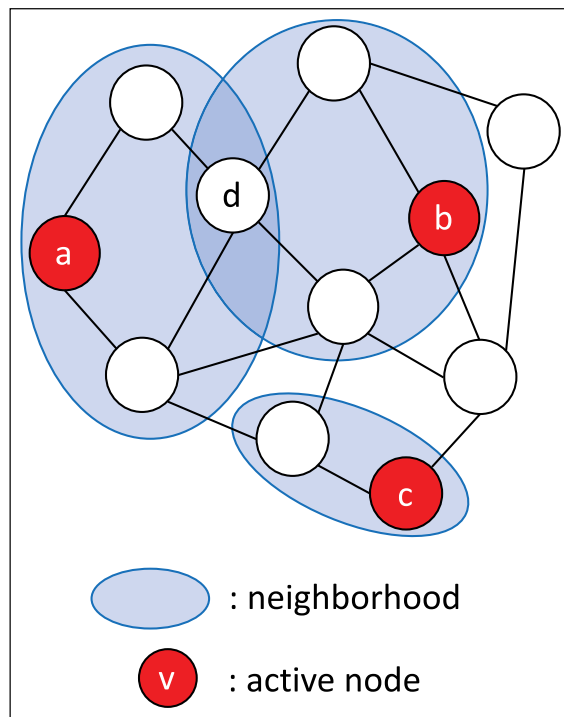
Reducing the turn-around time of this design flow without sacrificing the quality of results is critical for future designs. We believe this goal can be achieved by parallelizing the core EDA algorithms. Since circuits can be viewed abstractly as graphs and hypergraphs, a system for supporting the design and implementation of a parallel EDA tool-chain must have the following characteristics.

- It must support clean abstractions for reasoning about and expressing the available parallelism in graph (and hypergraph) algorithms.
- It must hide parallelization details such as synchronization from EDA algorithm designers.
- It must be scalable; as long as the algorithm has sufficient parallelism, performance should improve if more cores are used.

### Operator formulation of algorithms

A clean abstraction for expressing parallelism in graph algorithms is the *operator formulation*, a *data-centric* abstraction in which algorithms are described as a composition of a *local view* and a *global view* of the computation.

The local view is described by an *operator*, which is a graph update rule applied to an *active node* in the graph (some algorithms have active edges).



**Figure 2. Operator view of algorithms in Galois.**

Each operator application, called an *activity* or *action*, reads and writes a small region of the graph around the active node, called the *neighborhood* of that activity. Figure 2 shows active nodes as filled dots, and neighborhoods as clouds surrounding active nodes, for a generic algorithm.

An active node becomes inactive once the activity is completed. *Morph* operators can modify the graph structure of the neighborhood by adding and removing nodes and edges. And-inverter graph (AIG) rewriting [24] deploys morph operators. *Label computation* operators, in contrast, only update labels on nodes and edges without changing the graph structure. Field programmable gate array (FPGA) routing [25], formulated as a single-source shortest path problem (SSSP) within a routing resource graph, uses label computation operators.

The global view of a graph algorithm is captured by the location of active nodes and the order in which activities must appear to be performed. Topology-driven algorithms make a number of sweeps over the graph until some convergence criterion is met, for example, the Bellman–Ford SSSP algorithm. Data-driven algorithms begin with an initial set of active nodes, and other nodes may become active on the fly when activities are executed. They terminate when there are no more active nodes.

Dijkstra's SSSP algorithm is a data-driven algorithm. The second dimension of the global view of algorithms is ordering [26]. Activities in unordered algorithms such as SSSP can be performed in any order without violating program semantics, although some orders may be more efficient than others.

Parallelism can be exploited by processing active nodes in parallel, subject to neighborhood and ordering constraints. The resulting parallelism is called *amorphous* data parallelism. It is a generalization of the standard notion of data parallelism [27].

### Galois system

The Galois system implements this data-centric programming model (see details in [28]). Application programmers write programs in sequential C++, using certain programming patterns to highlight opportunities for exploiting amorphous data parallelism. The Galois system provides a library of concurrent data structures, such as parallel graph and work-list implementations, and a runtime system. The data structures and runtime system ensure that each activity appears to execute atomically. In this way, the Galois system encapsulates parallelization details and realizes performance scalability at the same time.

The Galois system has been used to implement parallel programs for many problem domains including finite-element simulations,  $n$ -body methods, graph analytics, intrusion detection in networks [29], FPGA routing [25], and AIG rewriting [24].

### Open-source flow

The key steps of the design flow we have developed are as follows:

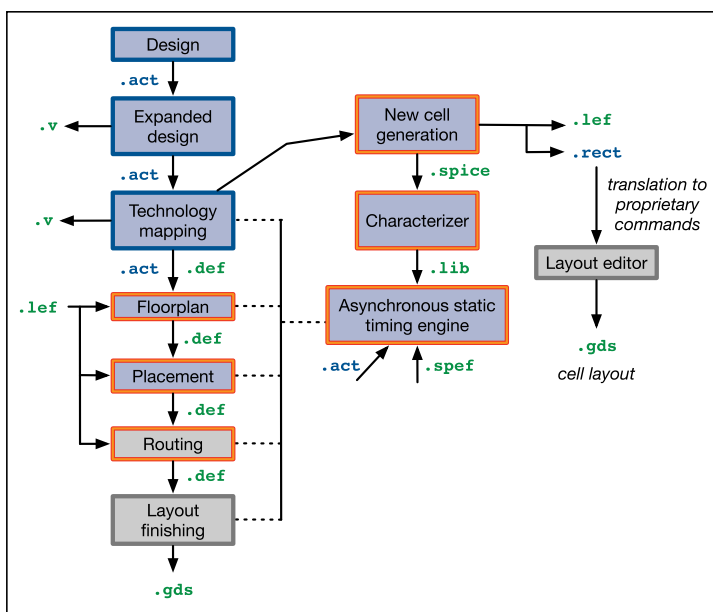
- design elaboration/expansion, which expands the design and customizes it based on parameters specified by the user [20];
- technology mapping and gate generation, which identifies the unique gates needed to implement the asynchronous circuit and generates the layout for the cells, if a new gate is found in the design [30], [31];
- static timing analysis, which implements the asynchronous equivalent of timing analysis, determines the performance/power of the design, and checks any timing constraints needed for correctness [31];
- design partitioning and floor-planning;
- asynchronous timing-driven placement [32], [33];
- timing-driven global routing [34], followed by detailed routing to complete the physical implementation.

The rest of the steps are standard, including inserting fill and adding the pads and seal ring. The flow is summarized in Figure 3. To inter-operate with commercial tools, at key steps we can import/export designs using standard formats such as a Verilog netlist, simulation program with integrated circuit emphasis (SPICE) netlist, and library exchange format (LEF)/design exchange format (DEF). We also accept parasitic information via standard parasitic exchange format (SPEF) files, and timing information using the *.lib* format. All the tools—both those under development and those ready for use—will be distributed at [35].

### Timing and power analysis

We have implemented a static timing analysis engine for asynchronous logic. Since asynchronous logic might have nonstandard gates, we also implemented a cell characterization engine that uses SPICE simulations to create *.lib* files for individual gates. The characterizer computes both delay and power tables for the gates.

Timing is one of the biggest differences between developing an asynchronous EDA flow and a synchronous EDA flow. Asynchronous timing has to handle cyclic circuit structures. Our timing analysis flow includes the following major steps: 1) creating an event-based timing graph for the asynchronous design from a gate-level representation; 2) estimating steady-state slew rates in cyclic



**Figure 3. Design flow for asynchronous logic.**

circuits; 3) analyzing the cycles in the event-based representation, and computing the critical cycle ratio which is a good metric of performance for the asynchronous circuit; 4) computing arrival time and required time for asynchronous circuits, and hence computing the performance slack for each node of the timing graph; and 5) computing the slack for timing forks.

When importing a design from a different flow as a “black box,” the timing graph fragment for the module must be included in the import (analogous to *.lib* files in synchronous logic). If a module is imported using a gate level netlist with timing forks, the event graph can be computed by our timing analysis engine.

Once event transitions are identified, we also compute the power consumption of the circuit. Since many of the steps needed by power analysis are the same as for timing analysis, we have integrated the two into a single unified engine.

The time complexity of timing analysis is much higher than in the synchronous case because of step (3), which computes the critical cycle ratio. We use the parametric shortest path algorithm to compute this ratio [36], which provides better run-time performance than previous approaches that use linear programming when the circuit size is large [31]. Another source of complexity is that the periodicity of an asynchronous circuit may not be from one iteration to the next. Instead, a circuit might have an *unfolding factor*  $M$ , where the circuit timing is only periodic every  $M$  iterations [37], [38]. When  $M$  is high, timing propagation has to be performed on a graph that is logically the  $M$ -fold unfolding of the cyclic graph.

We have implemented our timing analysis engine using the Galois framework described in the “Parallelism: The Galois framework” section. Our current implementation parallelizes the following parts of the full timing analysis: slew rate estimation, arrival time/required time, performance slack, and timing fork slack. The current runtime of our timing engine is shown in Table 1 with example circuits ranging from 0.3M–5.6M pins. The sequential runtime for large designs would be quite prohibitive—above 48 minutes for the largest example. However, the Galois framework can transparently speed up our runtime by large factors; for the largest example, we achieve almost 20× speedup, resulting in a more manageable runtime of roughly 2.5 minutes.

**Table 1. Runtime of static timing analysis implemented using the Galois framework on a 56-core Intel Xeon server. The numbers in parentheses specify the number of threads used to obtain the best runtime. “bd203\*” is a bundled-data sample design; the others are heavily pipelined and desynchronized versions of their synchronous counterparts.**

Name	#Pins	Seq (s)	Best(#thr) (s)	Speedup
bd203*	495	0.03	0.03 (1)	1.0
tv80-a	315,219	65.5	12.22 (42)	5.36
ac97ctrl-a	650,709	102.82	16.59 (28)	6.20
usbfunct-a	798,895	57.96	9.41 (42)	6.16
s38584-a	807,903	51.10	8.88 (35)	5.76
aescore-a	1,017,817	95.38	12.38 (42)	7.70
vgalcd-a	5,689,435	2,889.26	145.18 (56)	19.90

**Table 2. Performance comparison of our timing engine core against a recent open-source synchronous timing analysis engine (OT = OpenTimer). The numbers in parentheses specify the number of threads used to obtain the best runtime. All times are in milliseconds.**

Circuit	# Pins	Best Runtime (#thr)		Speedup over OT
		OT	Ours	
ac97ctrl	40,238	312.0 (21)	35.3 (7)	8.83
aescore	66,221	493.3 (7)	53.0 (7)	9.31
desperf	295,808	2,762.7 (14)	155.0 (14)	17.82
vgalcd	380,730	3,660.7 (28)	187.7 (14)	19.51
desperf*10	2,958,071	29,923.3 (14)	1,366.7 (14)	21.90
vgalcd*10	3,807,291	31,212.7 (35)	1,708.7 (14)	18.27

The same timing propagation core can be used to perform timing analysis for synchronous circuits, and hence, we can compare the performance obtained using the Galois approach to parallelization versus existing synchronous timing analysis engines that support multithreaded execution. Table 2 shows the result of this comparison against OpenTimer, an open-source synchronous timing analysis engine that supports multithreaded execution [39], demonstrating that our parallel timing analysis core achieves good parallel performance.

### Partitioning and floorplanning

For large designs, we have implemented a min-cut-based approach to floorplanning and design partitioning. To this end, we have developed a deterministic, parallel hypergraph-based partitioner using the Galois framework.

Our implementation uses the multilevel graph partitioning framework, where the original hypergraph is subjected to a number of *coarsening* steps

to create a much smaller hypergraph. The small hypergraph is then subjected to an initial partitioning. Finally, the small hypergraph is expanded out to the original graph by inverting the coarsening steps, and in each step, the partition is further refined.

Experimental results on a 28-core Intel Xeon show that our partitioner achieves 7× speedup for hypergraphs with roughly ten million nodes and 4–6× speedup for hypergraphs with 2–3 million nodes.

### Placement

Asynchronous circuits make use of a wide range of gates, especially state-holding gates that have pull-up and pull-down networks that are not complementary. The unbalanced pull-up and pull-down combined with keeper circuits for state-holding gates can lead to an inefficient layout using traditional single/double height standard cells. To alleviate this potential inefficiency, we have adapted existing standard-cell-based placement algorithms to account for cell heights that need not be uniform. We call this approach *gridded-cell* layout, where cells can have both height and width that is an integer multiple of a routing track [32]. Many techniques have been adapted for this nonstandard height cell layout approach, and new algorithms developed for both fast legalization and well-alignment in the presence of nonuniform cell heights.

Experimental results show that our current placer implementation is capable of handling large designs, with a performance that is almost an order of magnitude faster than commercial placers while suffering a 13% (geometric mean) quality loss when measured in terms of half-perimeter wire length.

When comparing standard cell versus nonstandard height cells for asynchronous logic, we found that our placement approach can improve density by 10%–17% compared to commercial standard-cell placers [32].

### Global routing

The last major piece of the flow that we are developing is a parallel global router. We have developed SPRoute, a Galois-based parallel implementation of the FastRoute [40] global router. We use FastRoute because it has good sequential performance for the global routing problem.

SPRoute uses a novel two-phase parallel scheme to achieve good speedup. In the initial phase, SPRoute exploits net-level parallelism. In this approach, different nets are routed in parallel. This proceeds until there

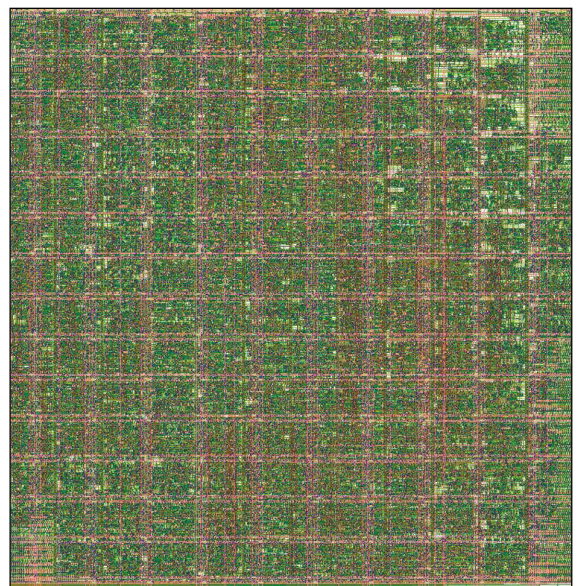
is congestion because of a lack of routing resources. This phase can achieve significant speedup because uncongested regions do not have any resource conflicts and thus net routing can proceed in parallel. Once congestion is detected, SPRoute switches to fine-grained parallelism where parallelism is exploited for frontier exploration during maze routing.

This scheme achieves 11× speedup with 0.6% quality of results reduction on a 28-core machine when compared to the baseline FastRoute implementation [34].

After global routing, the rest of the flow can proceed using standard tools since all the major decisions that impact timing have been accounted for during placement and global routing. Figure 4 shows the routed design of a simple asynchronous benchmark circuit, where the detailed router used was a commercial tool. Note that the placement does not use the standard cell rows.

### Memory compiler

The last major missing ingredient is a high-quality memory compiler. Almost every digital ASIC requires memory, and asynchronous designs are no different. While many commercial memories include self-timed internal access, standard memory compilers provide a “black box” implementation that only provides a synchronous interface.



**Figure 4. Routed example design in a 65-nm process. Detailed routing was performed using a commercial EDA tool.**

To address this challenge, we have built an asynchronous memory compiler (AMC) [41] that is based on the OpenRAM framework [42]. AMC makes a number of changes to the design of the SRAM it generates compared to its baseline OpenRAM implementation: 1) it uses asynchronous logic to implement the control, and therefore provides an asynchronous interface to the core memory; 2) it supports pipelined memory access for multi-banked memories with multiple in-flight transactions, where bank access is interleaved to improve effective throughput; 3) it supports subbanking with a hierarchical word-line structure to improve access time and reduce power consumption; 4) it supports technologies up to 28 nm, including thin cell and foundry cell bit-cells; and 5) it supports an atomic read-modify-write operation, which takes significantly less time than a read followed by a write. AMC includes a built-in self-test engine, as well as synchronous wrapper circuits (at reduced performance). Our comparisons show that the memories generated by AMC are competitive with published designs in the literature, as well as the memories available from the foundry [41].

#### Current status

We currently have a flow that can be used to design and implement quasi-delay-insensitive asynchronous circuits, as well as a restricted set of bundled-data circuits. The memory compiler has successfully been used to build memories in 65, 28, and 12 nm process technologies, as well as older technology nodes. The full flow has been exercised to design a mixed quasi-delay-insensitive/bundled data 65 nm ASIC and a 28 nm ASIC is in progress. The key additional work needed to support a richer class of asynchronous circuits is a more general timing analysis engine frontend. While the core timing analysis engine implements the algorithms needed for asynchronous timing analysis, the frontend that generates the input for the analysis engine is currently being improved so that it can support a richer set of asynchronous circuit families. The rest of the physical design flow supports any asynchronous logic family. Finally, we are working on tighter integration of the timing analysis engine with all the steps in the design flow.

**WE HAVE EMBARKED** on developing a high-quality open-source design flow for asynchronous circuits. In doing so, we developed a unified timing methodology

that can handle both synchronous and a number of different asynchronous circuit families. By building this timing abstraction into all the key EDA tools, our goal is to create an extensible framework where EDA developers can easily support new circuit families.

Significant work is still required both for improving the run-time performance of certain aspects of the flow, as well as improving the quality of results of the design. Some of the major ongoing efforts include: improving the accuracy of timing analysis, as well as its run-time performance in the presence of millions of timing constraints; better incorporation of timing information into both placement and routing, as well as buffer insertion when timing constraints cannot be met during place and route; improved cell generation when a circuit is not found in the standard library; and extending configuration files and algorithms to support the requirements of sub-10-nm designs. ■

#### Acknowledgments

This work was supported by the DARPA IDEA Program under Contract FA8650-18-2-7850.

#### References

- [1] R. Manohar and Y. Moses, "The eventual C-element theorem for delay-insensitive asynchronous circuits," in *Proc. 23rd IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2017, pp. 102–109.
- [2] A. J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," in *Proc. 6th MIT Conf. Adv. Res. VLSI*, W. J. Dally, Ed., 1990, pp. 263–278.
- [3] R. Manohar and Y. Moses, "Analyzing isochronic forks with potential causality," in *Proc. 21st IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2015, pp. 69–76.
- [4] N. H. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. London, U.K.: Pearson, 2010.
- [5] S. F. Nielsen et al., "A behavioral synthesis frontend to the haste/TiDE design flow," in *Proc. 15th IEEE Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2009, pp. 185–194.
- [6] D. Edwards and A. Bardsley, "Balsa: An asynchronous hardware synthesis language," *Comput. J.*, vol. 45, no. 1, pp. 12–18, 2002.
- [7] P. A. Beerel, G. D. Dimou, and A. M. Lines, "Proteus: An ASIC flow for GHz asynchronous designs," *IEEE DesignTest. Comput.*, vol. 28, no. 5, pp. 36–51, Sep. 2011.

- [8] S. Hauck, "Asynchronous design methodologies: An overview," *Proc. IEEE*, vol. 83, no. 1, pp. 69–93, Jan. 1995.
- [9] S. M. Nowick and M. Singh, "Asynchronous design—Part 1: Overview and recent advances," *IEEE Design Test. Comput.*, vol. 32, no. 3, pp. 5–18, Jun. 2015.
- [10] S. M. Nowick and M. Singh, "Asynchronous design—Part 2: Systems and methodologies," *IEEE Design Test. Comput.*, vol. 32, no. 3, pp. 19–28, Jun. 2015.
- [11] C. A. R. Hoare, "Communicating sequential processes," *Commun. ACM*, vol. 21, no. 8, pp. 666–677, 1978.
- [12] R. Manohar and Y. Moses, "Asynchronous signalling processes," in *Proc. 25th IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2019.
- [13] S. Gangadharan and S. Churiwala, *Constraining Designs for Synthesis and Timing Analysis*. New York, NY, USA: Springer, 2013.
- [14] S. A. Seshia, R. E. Bryant, and K. S. Stevens, "Modeling and verifying circuits using generalized relative timing," in *Proc. 11th IEEE Int. Symp. Asynchronous Circuits Syst.*, 2005, pp. 98–108.
- [15] A. Dan, R. Manohar, and Y. Moses, "On using time without clocks via zigzag causality," in *Proc. ACM Symp. Princ. Distrib. Comput.*, Jul. 2017, pp. 241–250.
- [16] I. Sutherland and S. Fairbanks, "GasP: A minimal FIFO control," in *Proc. 7th Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, 2001, pp. 46–53.
- [17] M. Singh and S. M. Nowick, "MOUSETRAP: Ultra-high-speed transition-signaling asynchronous pipelines," in *Proc. IEEE Int. Conf. Comput. Design VLSI Comput. Processors (ICCD)*, 2001, pp. 9–17.
- [18] A. J. Martin et al., "The design of an asynchronous MIPS R3000 microprocessor," in *Proc. 17th Conf. Adv. Res. VLSI*, 1997, pp. 164–181.
- [19] D. Fang, "Hierarchical asynchronous circuit compiler toolkit." Accessed: Jan. 8, 2021. [Online]. Available: <https://github.com/fangism/hack/>
- [20] R. Manohar. "Asynchronous circuit toolkit." Accessed: Jan. 8, 2021. [Online]. Available: <https://github.com/asynvlsi/act/>
- [21] C. T. O. Otero et al., "ULSNAP: An ultra-low power event-driven microcontroller for sensor network nodes," in *Proc. 15th Int. Symp. Qual. Electron. Design*, Mar. 2014, pp. 667–674.
- [22] F. Akopyan et al., "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.
- [23] R. Manohar, "Exact timing analysis for asynchronous circuits with multiple periods," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 3134–3138, Oct. 2020.
- [24] V. Possani et al., "Unlocking fine-grain parallelism for AIG rewriting," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2018, pp. 1–8.
- [25] Y. O. M. Moctar and P. Brisk, "Parallel FPGA routing based on the operator formulation," in *Proc. 51st Annu. Design Autom. Conf. (DAC)*, 2014, pp. 1–6.
- [26] M. A. Hassaan, M. Burtscher, and K. Pingali, "Ordered vs. unordered: A comparison of parallelism and work-efficiency in irregular algorithms," in *Proc. 16th ACM Symp. Princ. Pract. Parallel Program. (PPoPP)*, 2011, pp. 3–12.
- [27] K. Pingali et al., "The tao of parallelism in algorithms," in *Proc. PLDI*, 2011, pp. 12–25.
- [28] D. Nguyen, A. Lenharth, and K. Pingali, "A lightweight infrastructure for graph analytics," in *Proc. 24th ACM Symp. Operating Syst. Princ. (SOSP)*, Nov. 2013, pp. 456–471.
- [29] A. Lenharth, D. Nguyen, and K. Pingali, "Parallel graph analytics," *Commun. ACM*, vol. 59, no. 5, pp. 78–87, Apr. 2016.
- [30] R. Karmazin, C. T. O. Otero, and R. Manohar, "CellTK: Automated layout for asynchronous circuits with nonstandard cells," in *Proc. IEEE 19th Int. Symp. Asynchronous Circuits Syst.*, May 2013, pp. 58–66.
- [31] W. Hua et al., "Cyclone: A static timing and power engine for asynchronous circuits," in *Proc. 26th IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2020, pp. 1–9.
- [32] Y. Yang, J. He, and R. Manohar, "Dali: A gridded cell placement flow," in *Proc. 39th Int. Conf. Comput.-Aided Design*, Nov. 2020, pp. 1–9.
- [33] R. Karmazin et al., "Timing driven placement for quasi-delay-insensitive circuits," in *Proc. 21st IEEE Int. Symp. Asynchronous Circuits Syst.*, May 2015, pp. 45–52.
- [34] J. He et al., "SPRoute: A scalable parallel negotiation-based global router," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–8.
- [35] S. Ataei et al. "Asynchronous VLSI and architecture group." Accessed: Jan. 8, 2021. [Online]. Available: <https://github.com/asynvlsi/>
- [36] N. E. Young, R. E. Tarjant, and J. B. Orlin, "Faster parametric shortest path and minimum-balance algorithms," *Networks*, vol. 21, no. 2, pp. 205–221, Mar. 1991.
- [37] W. Hua and R. Manohar, "Exact timing analysis for asynchronous systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 203–216, Jan. 2018.

- [38] H. Hulgaard et al., "An algorithm for exact bounds on the time separation of events in concurrent systems," Dept. Comput. Sci. Eng., Univ. Washington, Seattle, WA, USA, Tech. Rep. 94-02-02, 1994.
- [39] T.-W. Huang and M. D. F. Wong, "OpenTimer: A high-performance timing analysis tool," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015, pp. 895–902.
- [40] M. Pan et al., "FastRoute: An efficient and high-quality global router," *VLSI Design*, vol. 2012, pp. 1–18, Aug. 2012.
- [41] S. Ataei and R. Manohar, "AMC: An asynchronous memory compiler," in *Proc. 25th IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2019, pp. 1–8.
- [42] M. R. Guthaus et al., "OpenRAM: An open-source memory compiler," in *Proc. 35th Int. Conf. Computer-Aided Design*, Nov. 2016, pp. 93:1–93:6.

**Samira Ataei** is an Associate Research Scientist with Yale University, New Haven, CT. Her research interests include memory design for end-of-the-roadmap silicon, memory compiler, in-memory/near-memory computing, and computer architecture. Ataei has a PhD in electrical engineering from Oklahoma State University, Stillwater, OK (2017).

**Wenmian Hua** is currently with Synopsys Inc. Mountain View, CA. His research interest includes timing and performance analysis of asynchronous circuits. Hua has a PhD in electrical and computer engineering from Cornell University, Ithaca, NY (2020).

**Yihang Yang** is currently pursuing a PhD with Yale University, New Haven, CT. His research interest includes asynchronous VLSI design and its physical design automation. Yang has an MASC in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada (2017).

**Rajit Manohar** is the John C. Malone Professor with the electrical engineering and computer science, Yale University, New Haven, CT. His research

interest includes the design and implementation of asynchronous circuits and systems. Manohar has a PhD in computer science from Caltech.

**Yi-Shan Lu** is currently pursuing a PhD with the Department of Computer Science, University of Texas at Austin, Austin, TX. His current research focuses on parallelization and language design for domain specific computation, for example, EDA. Lu has a master degree in computer science from NTHU, Hsinchu, Taiwan (2011). He is a Student Member of ACM.

**Jiayuan He** is currently pursuing a PhD in computer science from the University of Texas at Austin, Austin, TX. His research interests include parallel computing on multicore CPUs and GPUs, graph analytics, and place and route in EDA. He has a BE in electrical engineering and a second bachelors in economics from Tsinghua University, Beijing, China (2014).

**Sepideh Maleki** is currently pursuing a PhD (fifth year) with Computer Science Department, University of Texas at Austin, Austin, TX. Her research interests includes graph analytics, high performance computing, electronic design automation (EDA), and programming languages. Maleki has a master's degree in computer science from Texas State University, San Marcos, TX.

**Keshav Pingali** is the W.A. "Tex" Moncrief Chair of Computing and the CEO of Katana Graph. Pingali has a PhD from MIT, Cambridge, MA. He is a Foreign Member of the Academia Europaea, a Distinguished Alumnus of IIT Kanpur, India, and a Fellow of ACM, IEEE, and AAAS.

■ Direct questions and comments about this article to Rajit Manohar, Computer Systems Lab, Yale University, New Haven, CT 06520-8267 USA; rajit.manohar@yale.edu.

# Real Silicon Using Open-Source EDA

**R. Timothy Edwards**

Efabless Corporation

**Mohamed Kassem**

Efabless Corporation

**Mohamed Shalan**

The American University in Cairo

*Editor's notes:*

This article demonstrates that complete open-source tooling can be used to design industrial quality digital circuits. Using the OpenLane framework, based itself on the OpenROAD tool 2, the authors show a complete set of RISC-V-based SoC.

—Sherief Reda, Brown University

—Leon Stock, IBM

—Pierre-Emmanuel Gaillardon, University of Utah

■ **OVER THE PAST** 35 years, a number of academic open-source hardware design tools have been available; most notably SPICE, SIS/VIS, and Magic from the Berkeley EECS Department, which has a long history in open-source EDA tools. Other notable academic contributions include IRSIM from Stanford, netcomp from Caltech, and TimberWolf from Yale (originally also Berkeley) [1], [2]. Some of these tools remain popular among integrated circuit (IC) designers and used over the years to address specific design tasks. But complete design flows and methodologies were missing until recently. The past five years have seen rapid development and emergence of several important, complete open-source EDA tool flows for the development of end-to-end ASIC designs. These flows have been enabled by the coincident emergence of open-source EDA tools for field-programmable gate array (FPGA) synthesis, as well as repositories of open-source digital designs both general (as found on the website [opencores.org](https://opencores.org)) and specific to the RISC-V open instruction set architecture

Digital Object Identifier 10.1109/MDAT.2021.3050000

Date of publication: 27 January 2021; date of current version: 8 April 2021.

(ISA) from Berkeley EECS. FPGA design enjoys the benefit of a more software-like tool flow, where designs can be tested and debugged in simulation or in situ. Both FPGA and digital ASIC design share the same front-end flow of digital synthesis. The robust and capable synthesis tool Yosys [3] has formed the backbone of open-source EDA tool flows for both FPGA and ASIC develop-

ment. For ASIC development, additional tools are needed to complete additional critical tasks such as memory (RAM) compiling, analog modeling and simulation, floor planning, chip integration, power analysis, design for test (DFT), design rule checking (DRC), and layout versus synthesis (or schematic) checking (LVS).

In this article, we show a progression of work in tools, flows, reference designs, and an overall ecosystem in which we have been able to demonstrate the practical use of available open-source EDA tool flows to design, manufacture, and validate chips with first-time success. We have attempted to keep these designs open-source from end to end; this fact distinguishes the authors' efforts from other groups working on ASIC flows, as we start with open hardware descriptions (e.g., the RISC-V ISA), and use open-source tools (e.g., Qflow), and make all of the tools and the reference designs available for use on an open platform (<https://efabless.com>) that is free (as in beer, as the saying goes) to register, access, and use, and free (as in freedom) of the usual confidentiality agreements constraints of that typically exist between the ASIC designer, commercial EDA tool vendor, and foundry.



Although we make use of many tools and designs from many sources, inevitably we cannot fit them all into a single working flow. So while our work emphasizes Qflow and OpenROAD, we do not wish in any way to discount or detract from other important flows in use and under development, such as Coriolis [5], and tools both new (such as DrCU [6]), which we have not had the opportunity to investigate, and established (such as OpenTimer) for which we happen to have an acceptable alternative.

## Qflow

One of the authors (Edwards) has been instrumental in keeping a number of the long-surviving open-source EDA tools alive and modernized with continual development and publicly available repositories for the past 25 years. Noting the lack of open-source digital synthesis flows (of which bits and pieces existed, but no complete end-to-end flow), he put together some tools (such as the aforementioned Berkeley SIS/VIS, TimberWolf, and Magic) evolved from a script-based system written by then-colleague Steve Beccue, good for making small digital blocks in the context of mixed-signal systems. He posted this on his website [opencircuit-design.com](http://opencircuit-design.com) [2] and called it “Qflow” (Figure 1).

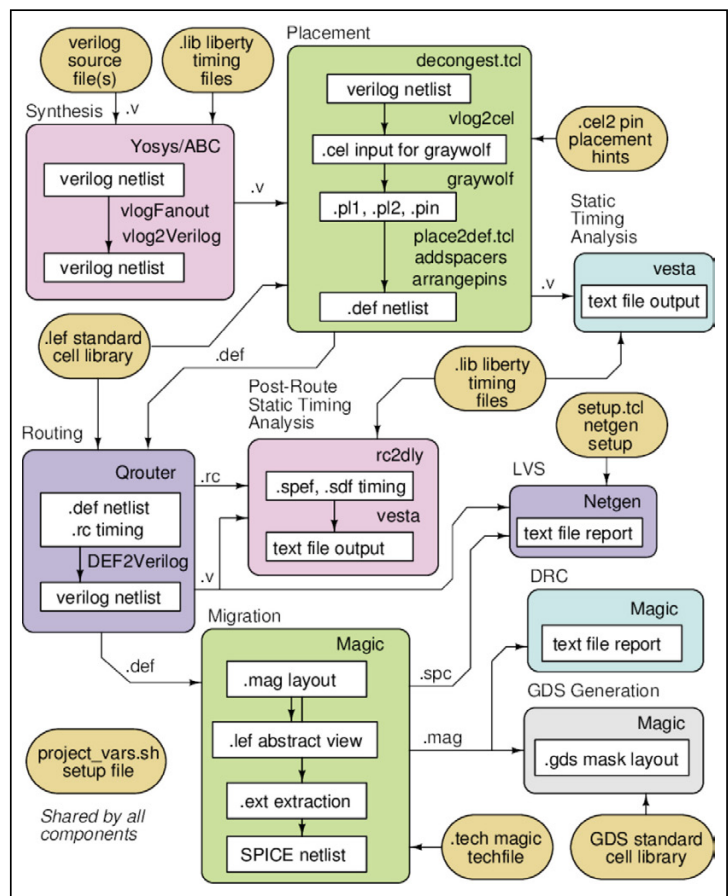
In short, Qflow is a complete tool chain for synthesizing digital circuits starting from verilog hardware description language (HDL) and ending in physical layout (in industry-standard GDSII format) for a specific target fabrication process. Qflow is an evolving and adaptable set of tools that aims to make use of any open-source component that can be made to work within the flow. Qflow incorporates Yosys for the front-end synthesis, Graywolf (a fork of TimberWolf) for digital standard cell placement, and clock tree synthesis (CTS, which unusually for digital synthesis flows is integrated into the placement tool); Vesta (an original tool written for Qflow) for static timing analysis (STA), and Qrouter (also an original tool written for Qflow) for global and detailed routing. Layout is completed in the editing tool Magic, which does DRC, circuit extraction and netlist generation, and final GDS format output generation. Extracted layout is checked against the synthesized netlist using the tool Netgen for final verification.

## Raven microcontroller

Raven is one of the designs that are implemented using Qflow. The choice of architecture for the

Raven processor (Figure 3a) was dictated by design IP availability and the tool flow capacity. To keep fabrication costs down, we selected an established 0.18  $\mu\text{m}$  process; in addition to the low cost, the mature node is well suited for analog design, and a number of silicon-proven analog circuits are available from the foundry. The addition of analog to digital converters (ADCs), digital to analog converters (DACs), comparators, and temperature sensors makes the offering of a small embedded processor appealing. For the processor architecture, we chose the RISC-V ISA because there is a growing body of open-source designs available for use. Although typically designed for FPGAs, they are easily adapted to an ASIC design.

We chose the PicoRV32 [7] core, written by Claire Wolf, due to its popularity and proven verification using FPGAs. The implementation is well-parameterized, with options for a number of RISC-V features such as a hardware multiplier, barrel shifter, and compressed instruction set. With most



**Figure 1. QFlow.**

**Table 1. Design flow evolution.**

DESIGN STEP	Qflow	CloudV SoC	OpenLane
System Design	N/A	CloudV	CloudV
RTL Lint	Verilator	Verilator	Verilator
RTL Simulation	iverilog	iverilog	iverilog
Logic Synthesis	Yosys	Yosys	Yosys
DFT Scan Insertion	none	none	Fault
DFT ATPG	none	none	Fault
Formal Verification	none	none	none
Placement	graywolf	graywolf	OpenROAD
Routing	qrouter	qrouter	OpenROAD
CTS	Qflow	Qflow	TritonCTS
Dynamic EMIR	none	none	none
Extraction	Magic	Magic	Magic
Timing Analysis	Vesta	Vesta	OpenSTA
Floorplanning	Magic	efabless	OpenROAD
Top-Level Placement	Magic	efabless	RePLAce
Top-Level Routing	Magic	Magic	OpenROAD
LVS	Netgen	Netgen	Netgen
DRC	Magic	Magic	Magic
GDS	Magic	Magic	Magic
SoC	Raven	Raptor	StriVe

options selected, the size of the processor core (including registers but excluding SRAM) is around 15k gates, well within the range of Qflow's placement and routing tools to handle. The primary design work on the Raven chip consisted of moving the memory array out of the core Verilog module, allowing use of an SRAM circuit provided by the foundry through use of an online memory compiler. We made selections from available analog and digital circuits and memory-mapped them to the processor core. We chose not to use a bus architecture for this design, mainly for simplicity and a faster time to tape-out. We used Qflow to synthesize the PicoRV32 core and a house-keeping SPI slave used to query and control signals related to power and clocking, mainly as a risk-mitigation measure, synthesized with 3.3 V-tolerant logic so that it could be outside of the core 1.8 V-voltage domain. Some additional tasks were to make the crystal oscillator and voltage regulator compatible with the six-metal back-end stack (selected because of its support in X-FAB multiproject wafer runs), and support 3.3-V domain circuits with padframe spacers and fast level shifters. To verify the design, we created "real-valued" verilog modules for the analog

components and simulated the chip and environs [external SPI flash chip for program storage, and universal asynchronous receiver/transmitter (UART) for communications] using iverilog, the popular and capable open-source verilog simulation tool developed by Stephen Williams [8]. We leveraged the RISC-V community's work in creating a GNU gcc compiler toolchain for the RISC-V architecture, creating a suite of testbench programs written in C and compiled to machine code files which are "loaded" onto the SPI flash chip emulation model during simulation. The suite of verification tests exercise the various analog circuits, digital general purpose digital input/output (GPIO), SRAM memory access, and all other major system components.

Qflow does not have a floorplanner and its router is not configured for top-level routing. Floorplanning consisted mostly of determining how the analog blocks would be best arranged around the digital core, with ample room for power routing, then applying pin position constraints on the core and resynthesizing. The chip top level was routed by hand using the Magic layout editor, a task that took about three days of work. Magic was used to verify the layout for DRC and LVS, with final verification done by the foundry during tape-in. The only design checks not caught by Magic were antenna violations; these checks have since been added both to Qrouter and to Magic.

We submitted the Raven chip for fabrication in August 2019 and received samples in December. We had the device packaged and designed a test circuit board, and had verified working silicon by the end of January. The overall design time was long due to the concurrent development of EDA tools and the efabless platform, but the entire design cycle from concept to working silicon was less than a year. The Raven SoC and its test benches including those that use software programming are published on Github (<https://github.com/efabless/raven-picorv32>).

### CloudV SoC workflow

While Qflow suffices to generate layout of single digital blocks up to the complexity of a small processor, it does not handle high-level system descriptions. A number of tools are being developed for that purpose. Since they generate standard verilog code from a high-level description, they are not on the roadmap of Qflow development. To introduce high-level system descriptions to

the flow, we made use of CloudV [9], a software system developed by one of the authors (Shalan) which introduces templates for selecting and assembling the components of an SoC architecture, and a GUI-based front-end for the SoC design process. In the CloudV flow, Qflow was retained for the back-end design with the corresponding manual labor for floorplanning and top-level routing.

As shown in Figure 2, the SoC design workflow in CloudV provides means for hardware IP designers, software developers, SoC architects, and SoC verification engineers to collaborate on a new design. The CloudV web-based IDE can be used to capture, simulate, and verify RTL designs in verilog HDL. After synthesis and mapping to a standard cell library, CloudV reports estimates of the layout area, clock speed, and power. The verified design can be packaged and placed in a common repository to be used by the SoC editor.

The SoC editor is a rule-based editor that enables SoC architects to design an SoC around a specific CPU and bus architecture (currently supporting the ARM Cortex M0 CPU and AMBA busses) using a library of open-source and proprietary IPs, as well as IPs created by designers collaborating on the project. The supported AMBA busses are AHB-Lite, AXI4-Lite, and APB. The architect need not worry about the details of how the IPs are wired as it is handled “behind the scenes” by the SoC editor. The architect may configure the SoC components if applicable; for example, the base address of each peripheral, the peripheral interrupt request line (IRQ) number(s), etc. Once the SoC design is completed, the SoC editor generates a set of outputs that help hardware and software development.

- The top-level verilog net list needed for simulation and SoC assembly.
- Intermediate files to communicate with other CloudV modules (e.g., memory map, linker script, and device driver configuration).

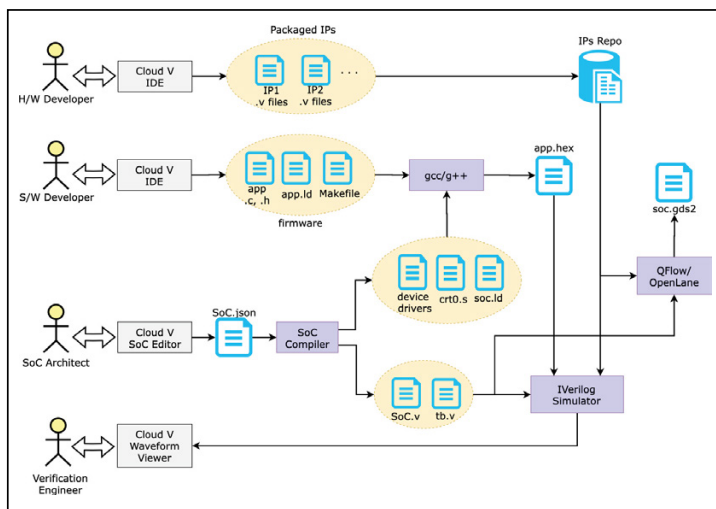
The SoC compiler converts the captured SoC architecture into HDL files need for implementation as well as C language header files and device drivers for the selected peripherals. Also, it generates startup code and the make file and linker scripts needed for software development. Using CloudV IDE, the software engineer can develop firmware for the designed SoC using the GNU C/C++ tool chain.

Finally, CloudV simulates the SoC running the compiled firmware, automating the validation step.

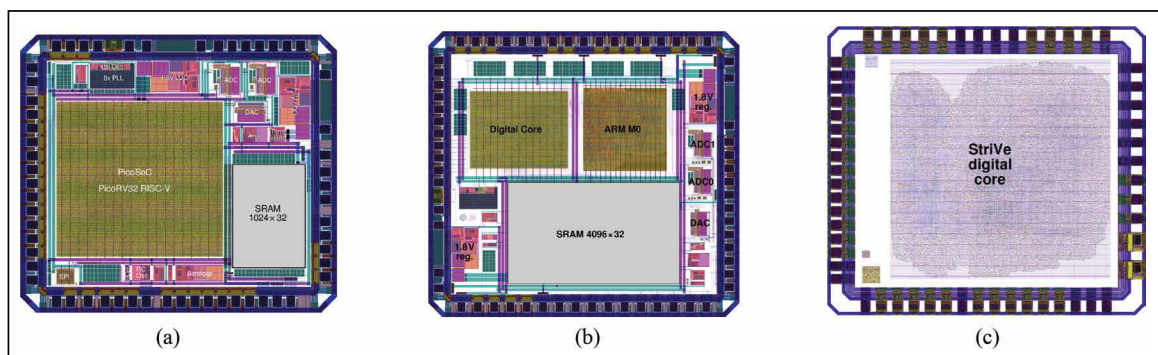
## Raptor microcontroller

We created the Raptor microcontroller as a reference design for the SoC flow using CloudV. Also, we wanted to demonstrate the implementation of a closed-source core (the ARM Cortex M0) within a larger open-source digital design, using a bus architecture to bridge the two. We contracted a third-party design house to synthesize, place, and route the M0 processor using a commercial tool flow. We then created CloudV templates for use with the M0 core and bus architecture, describing a number of blocks in a mixedmode architecture similar to the Raven chip. The templates comprised both purely digital subcircuits such as SPI and I2C bus masters, and timer/counter circuits; and mixed mode circuits with bus interfaces to the analog circuit blocks (ADC, DAC, comparator, etc.) provided by the foundry. The SoC compiler then generated a verilog netlist for the core digital design, including bus interfaces, and a verilog netlist describing the chip top level, including both digital and analog components, and the entire padframe. CloudV validated the design with system-level simulation of a full software test suite.

We used a new GUI python script to do floorplanning. Although this method lacks automated optimization, we could position and orient the padframe pad cells and core cells, and generate the (unrouted) top level layout automatically.



**Figure 2. SoC flow.**



**Figure 3. Open-source SoC designs using open-source EDA tools. (a) Raven SoC. (b) Raptor SoC. (c) StriVe SoC.**

We trained a third-party partner design house in the use of Magic, and they were able to complete the top-level signal and power routing in the span of approximately one week.

The Raptor chip (Figure 3b) was part of an aggressive design schedule to get three microcontroller designs onto a dedicated wafer run on the X-Fab XH018 process. This gave us less time than

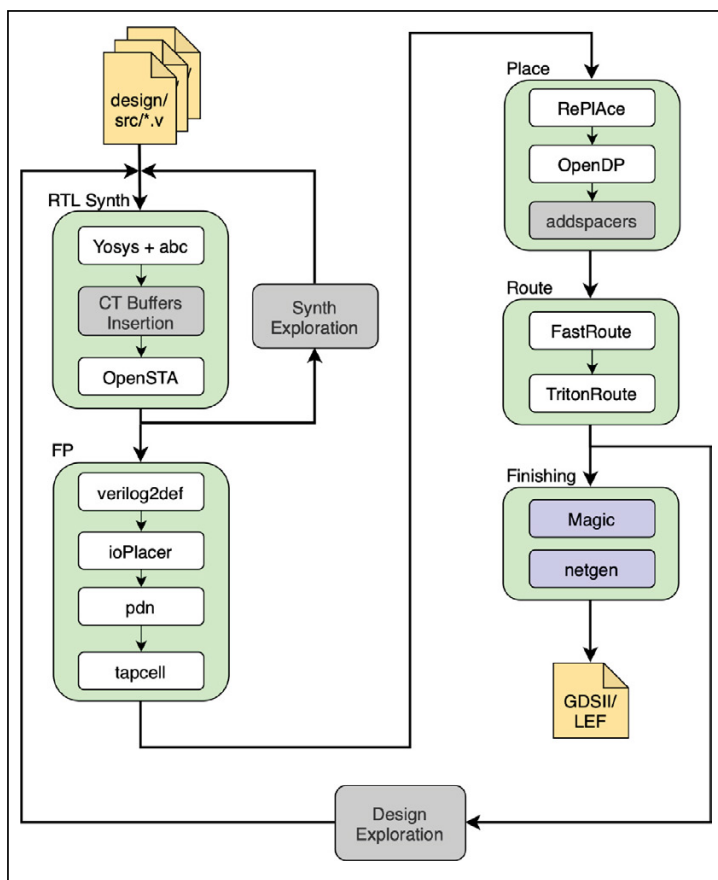
the Raven chip for the entire design flow, which again included both flow development (as the SoC generator and templates were developed concurrently) and chip design. The designs taped out in September 2019 and returned from fabrication in December. We had all three microcontrollers packaged, assembled on demonstration boards, and tested by the end of January. All were successful, although the most important success was the Raptor chip with the new design flow.

### OpenLane flow

Throughout the design phase of the Raven and Raptor chips, we were aware of the development of a new set of open-source tools called OpenROAD with the capacity to perform synthesis, placement, and routing on designs orders of magnitude larger than the small embedded processors we had built with Qflow. As the OpenROAD developers began releasing tools to the public, we began investigating how to integrate them onto our platform. One of the authors (Shalan) created a separate flow that is an amalgam of OpenROAD and Qflow tools, which we call “OpenLane” (Figure 4).

It is essentially a wrapper around OpenROAD tools with integration into the efabless file structures for projects and foundry processes. Because the OpenROAD project targets very high-end foundry nodes, it was critical to ensure that we could use the same tools with the mature processes (mainly 0.18 and 0.13  $\mu\text{m}$ ) we had available on the efabless platform.

The OpenLane flow eliminates most of the manual labor on the flow back-end required with Qflow. OpenLane makes use of the efabless floorplanning script, makes use of the STA, placement, and



**Figure 4. OpenLane design flow.**

routing in OpenROAD, and includes automated design space exploration. Because the stated purpose of the OpenROAD project is under-24-hour, no-human-in-the-loop end-to-end design, the OpenLane derivative will be the most advanced and automated flow to date.

## StriVe microcontroller

We are currently in progress making a reference design called “StriVe” (Figure 3c), another RISC-V architecture, to demonstrate the capabilities of the OpenLane flow. StriVe itself is not (yet) a fixed definition architecture, but is expected to evolve as we develop and refine the flow.

While the concurrent development will necessarily cause a long design cycle, the promise of the OpenROAD tools is that eventually the design cycle using this flow can be made much shorter than any open-source EDA flow has ever achieved to date. In addition to exercising the OpenLane flow, we are collaborating with other groups to be able to have the most open-source hardware ASIC design ever created. This includes replacing the proprietary foundry SRAM with open-source SRAM circuits created using the OpenRAM tool [10]; and open-source analog circuit designs in collaboration with the company Blue Cheetah, developing software and hardware around the Berkeley analog generator (BAG) system [11]. Ultimately, our goal is to have a design that can be open-sourced completely end to end, from the verilog source code and SoC templates to the GDS format layout data. We fully expect the StriVe microcontroller to revolutionize the concept of open hardware.

## Design flow future improvements

It is clear that the open-source EDA tools do not have a one-to-one coverage of every design step in the modern design methodologies (Table 1). While we continue to collaborate with the community to extend the coverage of the open-source EDA point tools, we diligently apply design best practices and precharacterized rule of thumb.

## Design for test

DFT support is under development. Functionality to support scan chain insertion will be added is under development. Automatic test pattern generation (ATPG) functionality will leverage existing open-source tools. We are currently

evaluating fault [12] as a potential open-source DFT solution.

## Formal verification

By constraining the target application to embedded industrially capable microcontroller, the design performance and gate count we can reasonably perform RTLvs-Gate-level verification by simulation with sufficient test bench coverage.

## Dynamic IR drop and signal integrity

Based on the constrained scope of the design and target application, which may vary depending on the choice of target performance metrics such as clock frequencies <100 MHz, and relatively low interconnect resistance for the 130/180 nm process nodes, best practices for power grid design and decoupling capacitor placement are applied along with sufficient timing margins to ensure design closure.

**WE AIMED TO** prove that any skepticism around the use of open-source EDA tools and flows to generate custom ASIC designs is misplaced. We have successfully designed, manufactured, and tested multiple SoC microprocessor reference designs and have a track record of proven first time-working silicon using end-to-end open-source EDA flows. The EDA tool development community is alive and growing, and we have been able to leverage new tools and methods as they have become available, incorporating them into existing flows with a flexibility that commercial tools are not capable of. Every generation of our flows and designs has cut the time to tape-out roughly in half. With the newest generation of open-source EDA tools, we fully expect a design flow that is competitive with any commercial flow, but better able to leverage community input and involvement. We firmly believe that open-source is the future of ASIC design. ■

## Acknowledgments

We would like to thank the following groups and individuals for their contributions to this project: the design team at Sankalp Semiconductor; Claire Wolf, SymbioticEDA; Andrew Kahng, UCSD, and the OpenROAD project group; Matt Guthaus, UCSC, and the OpenRAM project group; X-Fab foundry, Erfurt, Germany; Tim Whitfield, ARM.

## References

- [1] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *IEEE J. Solid-State Circuits*, vol. SSC-20, no. 2, pp. 510–522, Apr. 1985.
- [2] *QFlow*. Accessed: Jan. 1, 2021. [Online]. Available: <http://opencircuitdesign.com/qflow/>
- [3] C. Wolf and J. Glaser, "Yosys—A free Verilog synthesis suite," in *Proc. Austrochip*, 2013, pp. 1–6.
- [4] T. Ajayi et al., "Toward an open-source digital flow: First learnings from the OpenROAD project," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–4.
- [5] *Coriolis VLSI Backend Tools*. Accessed: Jan. 1, 2021. [Online]. Available: <https://www-soc.lip6.fr/equipecian/logiciels/coriolis/>
- [6] G. Chen et al., "Dr. CU: detailed routing by sparse grid graph and Minimum-Area-Captured path search," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 9, pp. 1902–1915, Sep. 2020.
- [7] *PicoRV32*. Accessed: Jan. 1, 2021. [Online]. Available: <https://github.com/cliffordwolf/picorv32>
- [8] *Icarus Verilog*. Accessed: Jan. 1, 2021. [Online]. Available: <http://iverilog.icarus.com>
- [9] M. Shalan and S. Reda, "Open-source SoC workflow in cloud V," in *Proc. Workshop Open-Source EDA Technol. (WOSET)*, Nov. 2018. [Online]. Available: <https://woset-workshop.github.io/WOSET2018.html>
- [10] M. R. Guthaus et al., "OpenRAM: An open-source memory compiler," in *Proc. 35th Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2016, pp. 1–6.
- [11] J. W. Crossley, "BAG: A designer-oriented framework for the development of AMS circuit generators," EECS Dept., Univ. California at Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2014-195, Dec. 2014.
- [12] M. Gaber, M. Abdelatty, and M. Shalan, "Fault, an open source DFT toolchain," in *Proc. Workshop Open-Source EDA Technol. (WOSET)*, Nov. 2019. [Online]. Available: <https://woset-workshop.github.io/WOSET2019.html>

**R. Timothy Edwards** is the VP of Analog and Platform at Efabless Corporation, San Jose, CA, as well as a developer and maintainer of open-source EDA tools for over 25 years, and the owner of the website [opencircuitdesign.com](http://opencircuitdesign.com). Edwards has a PhD in electrical engineering from Johns Hopkins University, Baltimore, MD.

**Mohamed Shalan** is an Associate Professor (with tenure) with the Department of Computer Science and Engineering, American University, Cairo, Egypt. His research interests are in the areas of hardware/software codesign, embedded real-time systems, digital hardware design, energy-efficient computing systems, and electronics design automation. Shalan has a PhD in computer engineering from Georgia Institute of Technology, Atlanta, GA.

**Mohamed Kassem** is the co-founder and CTO of Efabless Corporation, San Jose, CA. Kassem has an MEng in electrical engineering from the University of Waterloo, Waterloo, ON, Canada.

Direct questions and comments about this article to R. Timothy Edwards, Efabless Corporation, San Jose, CA 95131 USA; [tim@efabless.com](mailto:tim@efabless.com); and/or Mohamed Shalan, Department of Computer Science and Engineering, American University in Cairo, Cairo 11835, Egypt; [mshalan@aucegypt.edu](mailto:mshalan@aucegypt.edu).

# Fault: Open-Source EDA's Missing DFT Toolchain

**Manar Abdelatty, Mohamed Gaber, and Mohamed Shalan**

The American University in Cairo (AUC)

*Editor's notes:*

An open-source DFT flow is essential for any open-source solution. This article describes an approach to fill in this missing piece.

—Sherief Reda, *Brown University*

—Leon Stock, *IBM*

—Pierre-Emmanuel Gaillardon, *University of Utah*

■ **However foolproof current** EDA tools happen to be, fabricated circuits may not function correctly because the manufacturing process itself is imperfect. Defects, such as short circuits and open circuits, may be introduced because of these imperfections. Therefore, manufactured circuits are typically tested against defects before packaging, as it is crucial to identify faulty circuits as early as possible; when the faulty chip is soldered on a printed circuit board, the cost of fault remedy would be multiplied by ten. This cost factor continues to apply at every step until the system that uses the chip is delivered to the end user, referred to in the industry as “the rule of ten.”

Regardless the wide variety of proprietary and commercial design-for-testability (DFT) toolchains available, there is a surprising dearth of open-source DFT toolchains. There are some freely available tools with limited utility, such as the Atalanta [1] automatic test pattern generator, which, even then, is not open source as it has a number of usage restrictions. In addition to the source code being freely customizable for research purposes, the reality is that the primary

are typically available at a nominal cost to educational institutions, in the industry, they command outrageous, cost-prohibitive prices to startups; erecting barriers in the EDA space and inhibiting innovation in the field. Fortunately, the barrier to fabricating the actual chips is slowly being broken down by projects like OpenROAD and Google Open-Process Design Kit (PDK), which necessitates the existence of an open-source automated flow for DFT.

Testing of digital logic circuits involves the application of test data (test pattern/vector) to the device under-test (DUT) and the comparison of the resulting response to an expected one as produced by a known-good model [the golden model (GM)]. Should a manufacturing defect be able to alter the behavior of a circuit, a discrepancy would appear between the DUT and the GM, which allows for the DUT to be quickly pointed out as a defective chip. At the core of this process is what is known as test pattern generation, which aims to find a set of input sequences that would be able to detect faulty circuits.

Test pattern generation is a complex process with two primary aspects to optimize: 1) the cost of test application (proportional to the testing time) and 2) the quality of the tests (coverage). In essence, automatic test pattern generation (ATPG) software is

*Digital Object Identifier 10.1109/MDAT.2021.3051850*

*Date of publication: 14 January 2021; date of current version: 8 April 2021.*

designed to minimize the number of generated test vectors (TVs) (and therefore, lessen the amount of time spent in testing) while maximizing the number of covered fault sites to ensure that in that testing, as many defects as possible are covered. A less important but still significant nonrecurring engineering cost is the time spent on generating the ATPGs: for larger, more complex circuits especially with millions, if not billions of fault sites, the ATPGs should also aim to minimize the amount of time taken to generate the TV set.

DFT tools also typically include the infrastructure required for a circuit to support such testing: because the many possible issues that may arise during the fabrication of a hardware design require almost any hardware written to be designed with testability in mind. To this extent, standards have been introduced to assist with automated testing, most famously IEEE 1149.1 [2], which is commonly known as Joint Test Access Group (JTAG).

The EDA research team at the American University in Cairo has thus endeavored to create such a tool: one that leverages existing open-source tools such as the Yosys Open SYnthesis Suite [3], the Icarus Verilog simulator [4], and the Pyverilog [5] to deliver a cohesive experience encompassing netlist cutting, ATPG, static compaction all the way to scan chain insertion, JTAG interface stitching, and verification. We succinctly call this toolchain “Fault.” Fault is designed and implemented to support standard EDA formats; hence, it can be integrated into any industrial RTL to graphic design system II (GDSII) flow.

## Design-for-testing overview

### Fault models

Because of the diversity of VLSI defects, it is difficult to generate tests for real defects. Fault models are necessary for generating and evaluating a set of TVs. *Fault modeling* is a process by which possible fault sites can be represented and simulated behaviorally, regardless the actual cause. Also, it should be computationally efficient in terms of fault simulation and test pattern generation. Many fault models have been proposed, but, unfortunately, no single fault model accurately reflects the behavior of all possible defects that can occur. However, the single stuck-at fault model is widely used and considered the *de facto* standard fault model as it has been used successfully for decades.

### Stuck-at-faults

A stuck-at fault affects the state of logic signals on connections in a logic circuit, including primary inputs (PIs), primary outputs (POs), internal gate inputs and outputs, fanout stems (sources), and fanout branches. We refer to them as *fault sites*. Generally, the number of fault sites is equal to the sum of number of PIs, the number of gates, and the number of fanout branches. A stuck-at fault transforms the correct value on the faulty signal line to appear to be stuck at a constant logic value, either a logic 0 or a logic 1, referred to as stuck-at-0 (SA0) or stuck-at-1 (SA1), respectively.

### Delay faults

A delay fault is described as a fault that is not inherently structural, rather, it is a fault that prevents the device from operating at the desired clock speed. While chips with structural imperfections are obviously undesirable, chips that cannot run at the specified clocking requirements are also functionally useless, that is, outside of a limited class of general-purpose computing hardware that does not need determinism, making stuck-at faults not completely sufficient for circuit testing. Of particular interest is the so-called transition fault model, which is easy to implement by modifying stuck-at ATPG software [6].

### Combinatorial equivalence

As the vast majority of VLSI circuits are sequential, the stuck-at faults are, by nature, limited as a register would typically lie between an input and an output, making it so a stuck-at fault would typically not propagate from an input to output, therefore, making tests useless. Therefore, a method must be utilized to somehow bypass the registers for the purposes of TV generation and testing. There are two methods used: one while generating TVs, which is *register cutting*, and another for fabricating the actual chip, which is the use of a scan chain.

### Cutting

“Cutting” the circuit refers to the act of replacing every D-flipflop with an input and an output so they can be easily accessed while generating test patterns. As shown in Figure 1, each flip-flop’s output becomes an input for the rest of the circuit and each flip-flop’s input becomes an output for the rest of the circuit. In that manner, it is possible to test all inter register logic by manipulating the register outputs as desired and evaluating the register inputs alongside testing regular inputs and outputs. The circuit



generated by this process is ephemeral; however, it is only used for pattern generation and then discarded.

### Scan chain generation

To utilize these test patterns, which also include the registers with actual circuits, the circuits must include some manner of infrastructure to allow the manipulation of register values: both writing the “inputs” and reading the “outputs.” This is typically approached by connecting *all the registers in a circuit* in a process known as scan chain stitching, where every register is serially connected to another register in the circuit forming a full serial chain. A test-mode pin is usually added to multiplex between the regular interregister logic and scan chain logic. The scan chain is self-verifying: simply scanning a pattern in and out fully is sufficient to ensure that not only the scan chain is functional but also ensure that all registers in the circuit are functional as well: failing to scan a pattern in and out is in itself a quick way to realize that a circuit is faulty.

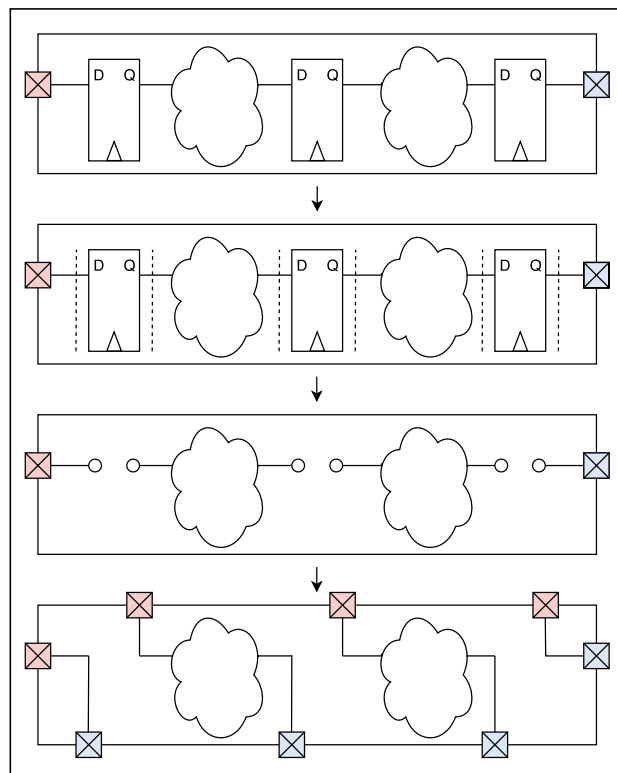
### Automatic test pattern generation

Test generation is the task of producing an effective set of TVs that will achieve high fault coverage for a specified fault model. In general, ATPG tools differ by the used fault model and the approach or algorithm used for the TVs generation approach.

The other key characteristic of ATPG is the method used to generate the TVs. Random test generation (RTG) is the simplest method for generating TVs. TVs are randomly generated and fault-simulated (simulated in the presence of faults) till a reasonable high fault coverage by the TVs is achieved.

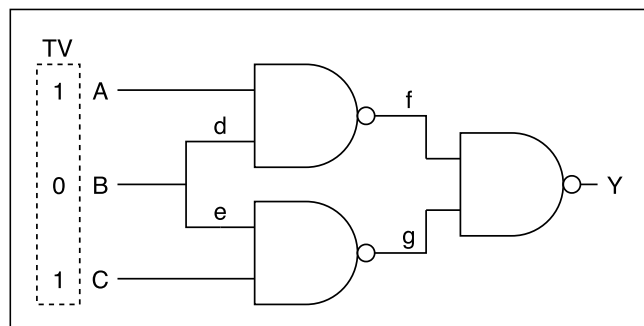
There are also a number of ATPG algorithmic methods that are in wide use today, including the D-algorithm, the path-oriented decision making (PODEM), and the fan-out oriented algorithm (FAN) [7]. Pattern generation through any of these algorithmic methods requires what is known as path sensitization; which refers to fault activation and propagation by finding a path in the circuit that will allow a fault to show up and propagate to an observable output of the circuit if it is faulty.

For the example circuit shown in Figure 2, the ATPG process identifies eight fault sites {A, B, C, d, e, f, g, Y}, and 16 stuck-at faults. To simulate stuck-at faults, the ATPG constructs a faulty model for each generated TV by forcing each fault site to be stuck-at zero and stuck-at one. Then, the output of the faulty model is compared to the GM output



**Figure 1. Converting a sequential circuit to a combinational circuit by “cutting.” The inputs are marked in red and outputs are in blue.**

and discrepancies mark a fault as detectable by the applied TV. For example, applying the input sequence {1,0,1}, while forcing node e to be stuck-at one, would produce an incorrect circuit result which means e’s stuck-at one fault is detectable by the applied TV. However, forcing e to be stuck-at zero would result in a correct circuit output meaning that e’s stuck at zero fault is not detectable by the applied vector. This process is repeated for all sites, for each generated vector, and then coverage is computed.



**Figure 2. ATPG example. The fault sites are lettered.**

## Fault toolchain

Fault operates on synthesized netlists in Verilog and is made up of five components: Cut, PGen, Compact, Chain, and Tap. Figure 3 shows the typical design flow of Fault. First, the flattened netlist is converted into a pure combinational design using Cut. This modified netlist is used for the ATPG process done by PGen which outputs the final coverage and the generated TVs in a javascript object notation (JSON) format. The generated TV set is then compacted by Compact that reduces the number of the generated TVs without affecting the coverage. Finally, scan chain insertion is done by Chain and a JTAG controller is stitched to the inserted scan chain by Tap.

### Cut

Using Pyverilog [5], the flattened netlist flip-flops are removed converting the sequential design into a pure combinational design. The new netlist has an extra input port for every removed flip-flop output pin and an extra output port for every removed flip-flop input pin.

### PGen

PGen is used to perform ATPG for stuck-at faults. The stuck-at fault model assumes that manufacturing defects cause nodes to be stuck at logic zero or logic one. PGen uses pseudorandom ATPG coupled with fault simulation. This is a simpler alternative to algorithmic methods such as PODEM and D algorithms. Algorithmic methods require “path sensitization,” which makes them complex to handle netlists mapped using any arbitrary standard cell library. In PGen, TVs are pseudorandomly generated then simulated. PGen

generates a testbench, for every generated TV, that compares a GM to a model where fault sites are progressively injected using Verilog force statements. The outputs of both models are compared, and any fault site that can be marked as detectable using said TV is sent back to Fault to be marked as covered. PGen stops generating TVs when either the target coverage or the maximum number of TVs has been reached. Final coverage is then output to a file in a ubiquitous and easy to manipulate JSON format.

### Compact

Reduces the size of the TV set using static compaction while maintaining the coverage percentage of the initial test set. The compaction is needed to reduce the testing time; hence, reducing the cost. The compaction process is illustrated in Algorithm 1. It starts with two sets: the initial TV set, generated by the ATPG, along with its covered faults and an empty compacted TV set. First, essential TVs (i.e., ones that cover at least one fault not covered by any other TV in the set) are added to the compacted set unadulterated, while the initial test set has the essential fault points removed. Then, the TV with the highest number of detectable faults is inserted in the compacted set and the faults covered by that vector are removed from the initial test set. This process is repeated until the compacted set covers all previously detected faults.

### Chain

Performs scan chain insertion. Chain converts a netlist’s flip-flop cells to scan cells by adding an abstract multiplexer definition to every flip-flop input port using Pyverilog [5]. This definition is then mapped by Yosys [3] to either a multiplexer (MUX) cell from the standard cell library or a scannable

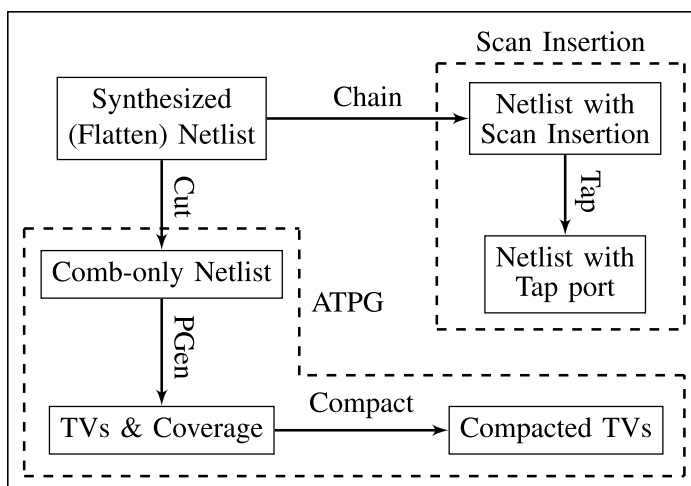


Figure 3. Fault design flow.

---

### Algorithm 1: Static TV Compaction.

---

**Data:** S1 – TV set

F1 – Detectable Faults

**Result:** S2 – Compacted TV set

initialize S2 ;

add essential TVs to S2;

**while** S2 does not cover all faults in F1 **do**

    Find TV that covers the highest number of faults;

    add TV to S2;

    Remove faults covered by TV from F1;

**end**

---

flip-flop cell if the library has one. The chain also adds boundary scan cells for the netlist's inputs and outputs, then stitches all the scan cells into one scan chain. The chained netlist has extra ports for the serial-in and serial-out of the constructed scan chain in addition to the testing control signals. Additionally, Chain offers an option to automatically generate a testbench to verify the scan chain integrity.

## Tap

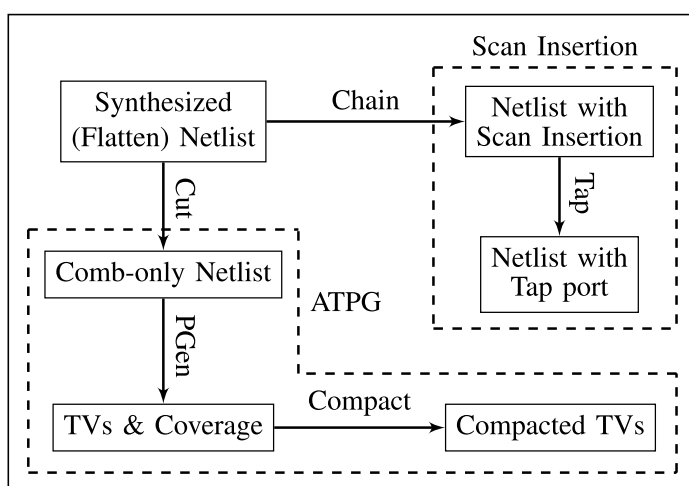
Tap adds the JTAG interface to a chained netlist using Pyverilog. This is accomplished by adding its five namesake test access ports (TAPs): serial test data in (TDI), test mode select (TMS), serial test data out (TDO), test clock (TCK), and active low test reset (TRST). As illustrated in Figure 4 the serial-in and serial-out of the constructed scan chain are stitched to the TDI line and the input of the TDO line multiplexer respectively. Selecting a register between TDI and TDO lines is done by shifting an instruction code on the TDI line. For the purpose of doing on-chip testing, a custom instruction is defined to select the internal scan chain. The TAP controller is verified by automatically generating a testbench wherein TVs are shifted through the scan chain, then applied to the onchip logic by deasserting shift for one clock cycle, then the captured output is shifted out and compared to the fault-free response.

## Implementation

The fault is implemented in the Swift programming language [8] as it is a statically typed, safe, native programming language that could also interact with and use Python-based libraries idiomatically.

Fault leverages the Swift–Python interoperability developed by Google Inc. as part of their Swift for Tensorflow project [9], allowing it to interface seamlessly with the Pyverilog library which produces an abstract syntax tree for direct manipulation, necessary for the cutting behavior, scan chain stitching, and other things. Most logic is implemented in pure Swift, which, while compiled Python yields a negligible speed boost, the native programming language is more lenient on memory usage, which is a great boon when simulating large hardware designs.

Interfacing with Yosys and Icarus Verilog is done via simple calls to the UNIX shell: Fault would generate the synthesis script or testbench, respectively, call the tool responsible and parses its output, which may be written to either a pipe or a file, which is then read by



**Figure 4. JTAG controller connection to a chained netlist.**

Fault. The fault runs many simulations in parallel and benefits greatly from a multithreaded environment.

Despite that, however, a practical problem is that setting up the Swift language, let alone Swift/Python interop, is cumbersome on Linux, which is a great concern considering cloud infrastructure overwhelmingly runs Linux—not to mention that the problem becomes greater while distributing for users as Windows support for Swift is immature and Apple does not yet provide official binaries for the language on Windows. To help alleviate these problems, lightweight Docker images were created so one may run Fault on any platform of their choice in a reliable, relatively configuration-free setting.

## Benchmarking and performance

We have evaluated the performance of Fault's flow on a number of open-source designs frequently used as benchmarks. The coverage and runtime are used as metrics for the ATPG process. Coverage results were obtained by running the ATPG process with a ceiling of 5000 TVs, an increment of ten TVs per iteration, and minimum coverage of 97% such that the ATPG process stops when the ceiling count is encountered or the minimum coverage is achieved. As shown in Figure 5a, the fault is able to achieve 96.6% coverage on average. As shown in Figure 5b, Fault's runtime is moderately low for smaller designs, but it is significantly higher for larger designs. This is largely attributed to the ATPG problem being NP-complete [10] thus needing to generate a larger TV set to reach a reasonable coverage;

hence, more Iverilog simulations. The fault simulations were carried out on Intel Xeon-based station running Ubuntu 18.04 long-term support (LTS) with Fault's native installation with an allocated RAM of 32 GB and ten threads.

Additionally, we experimented with Atalanta to validate Fault's stuck-at simulator. The experiment involved generating the TVs with Atalanta, then simulating the generated vectors using Fault's stuck-at simulator. Since Atalanta does not support standard EDA formats and is only compatible with a netlist in the International Symposium of Circuits and Systems (ISCAS) bench format, we introduced an optional utility to Fault's flow, bench, that takes a gate-level netlist and the standard cell library Verilog models as an input and generates the netlist in bench format. This netlist was then used as an input to Atalanta. Figure 5c shows the coverage results for Fault's simulator and Atalanta. Fault's coverage is slightly lower than Atalanta; however, Fault shows an accurate coverage percentage of the gate-level netlist because the bench circuit format does not support some circuit constructs such as being permanently grounded.

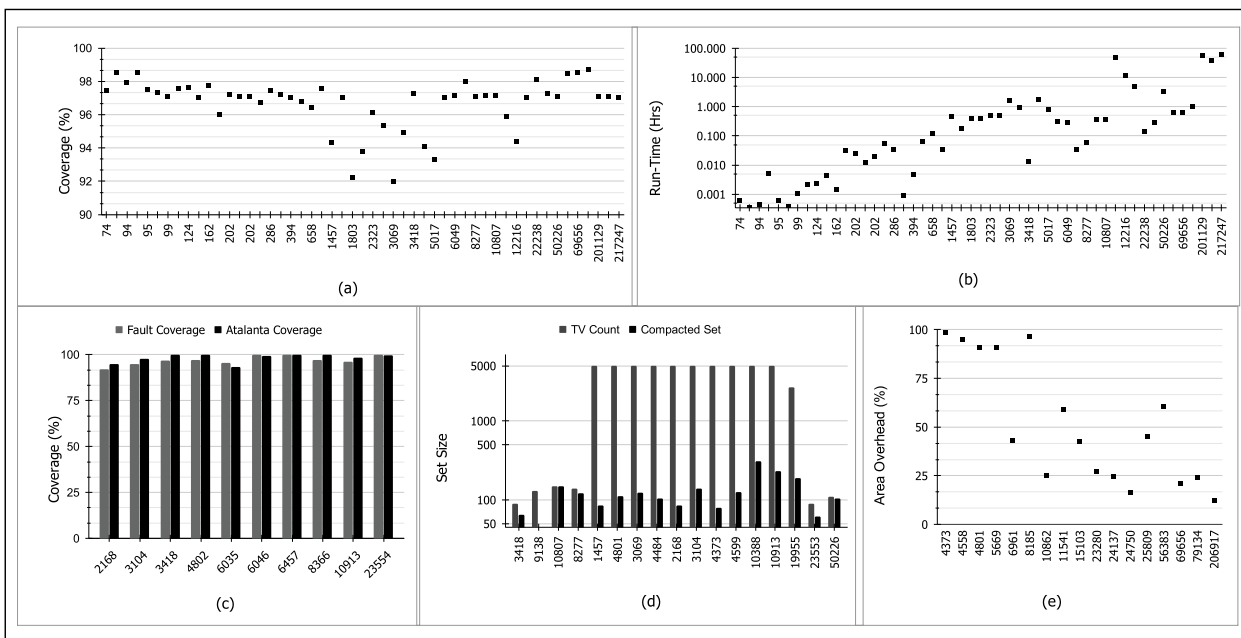
For the compaction process, the metric used is the size of the initial TV set versus the compacted set. As shown in Figure 5d, Fault is able to significantly reduce large TV sets (5000) with a reduction percent

of 97.2% on average. For smaller sets, the reduction percentage is lower because most of the generated TVs are essential. The compaction process was also verified by rerunning the fault simulations with the compacted set to ensure that the coverage percentage is not reduced.

For the scan chain insertion and JTAG controller stitching, the area overhead is calculated to evaluate the penalty of DFT. As shown in Figure 5e, for smaller designs, the area overhead is quite large (i.e., >90%) rendering it unnecessary, however, for larger designs, the cost of adding extra testability logic is insignificant making the strong case of the necessity of having DFT structures for complex designs especially.

### Using fault for ASIC tape-out

With the advent of open-source EDA movements, the release of Google SkyWater PDK, and openLane project, we are able to work on a potential tape-out of striVe6 chip [11], a PicoRV32 SoC with testability structure automatically injected by Fault. The fault was able to achieve a coverage percentage of 91% on striVe6 with 5000 TVs that were compacted to 311 TVs and verified to have the same coverage as the original set. The scan chain and JTAG controller were automatically constructed and verified by Fault with an estimated area overhead of 16%.



**Figure 5. Fault's flow results. (a) Coverage versus gate count. (b) Run-time versus gate count. (c) Fault and Atalanta coverage versus gate count. (d) Initial TV set versus compacted set count. (e) Area overhead versus gate count.**

The striVe6 exercise has revealed several challenges with Fault's flow. First of which is how to deal with unscannable black-box modules (like SRAM) in the ATPG process. This was solved by extending the Cut option to support removing black-box modules and, like flip-flop cells, exposing the blackbox module PIs as output ports and POs as input ports; thus bypassing the SRAM block while testing. Additionally, the Chain option was extended to support bypassing black-box modules. This was achieved by adding a wrapper scan cell to the black-box PIs and POs that were eventually stitched to the flip-flops scan chain. The second challenge was having flipflops with different clock-edge sensitivity, which affected the scan chain integrity. This was solved by also extending the Chain option to add an inverter to the TCK supplied to the negative-edge triggered flip-flops.

**IN THIS ARTICLE**, we introduced Fault the first and only practical open-source DFT toolchain compatible with HDL designs. Fault toolchain provides all needed utilities to generate TVs, simulate faults, and insert scan chains. The fault is aiming at filling some of the gaps in the emerging open-source EDA ecosystem. Also, Fault provides all the needed infrastructure for research activities in digital application specific integrated circuits (ASIC) testing.

Future work includes ATPG acceleration by generating TVs algorithmically to improve the coverage as well as the run-time. While the pseudo-random number generator (PRNG) demonstrated the capability of reaching high coverage in a reasonable amount of time, targeted TVs may yield coverage improvement by covering fault sites the RNG failed to detect. Considerations also include adding support for fault collapsing, use of compiled HDL simulators such as Verilator [12]. We have plans for supporting BIST for memory and logic, TVs compression, and scanchain reordering. Also, we are planning to extend Fault to support a larger variety of fault models like the transition fault model and to add support for fault diagnostics to locate defects and improve the yield. Additionally, we plan to extend fault to support testability checking to detect DFT violations like the generated clock and combinational feedback loops. Finally, we will be adding support for IEEE P1687 and IEEE 1500 standards. ■

## Acknowledgments

The Fault is publicly available under the Apache 2.0 license at <https://github.com/Cloud-V/Fault>,

including the benchmarks used for testing. Full installation and usage instructions are available in the GitHub Wiki and the Readme files. It has been tested to work with macOS 10.15 "Catalina" and Ubuntu 18.04 "Bionic Beaver." Because of the complicated set of dependencies required to run the toolchain, a Docker container based on the latter platform has been made available at <https://hub.docker.com/r/cloudv/fault>. Fault is part of the CloudV Project at the American University in Cairo (AUC), an initiative to reshape digital design and system-on-a-chip design education around open-source software and cloud technologies.

## References

- [1] H. K. Lee and S. D. Ha, "On the generation of test patterns for combinational circuits," Dept. Elect. Eng., Virginia Polytech. Inst., Blacksburg, VA, USA, Tech. Rep. 12 93, 1993.
- [2] *IEEE Standard for Test Access Port and Boundary-Scan Architecture*, IEEE Standard 1149.1-2013 (Revision of IEEE Std 1149.1-2001), 2013.
- [3] C. Wolf and J. Glaser, "Yosys—A free Verilog synthesis suite," in *Proc. Austrochip*, 2013, pp. 1–6.
- [4] S. Williams. *Icarus Verilog*. Accessed: Feb. 5, 2020. [Online]. Available: <http://iverilog.icarus.com>
- [5] S. Takamaeda-Yamazaki, "Pyverilog: A Python-based hardware design processing toolkit for Verilog HDL," in *Proc. Int. Symp. Appl. Reconfig. Comput.*, Apr. 2015, pp. 451–460.
- [6] A. Krstić and K.-T. Cheng, "Delay fault models," in *Delay Fault Testing for VLSI Circuits*. New York, NY, USA: Springer, 1998, pp. 23–31.
- [7] Z. Navabi, "Test pattern generation methods and algorithms," in *Digital System Test and Testable Design Using HDL Models and Architectures*. New York, NY, USA: Springer, 2011.
- [8] Chris Lattner and Apple Inc. *The Swift Programming Language*. Accessed: Feb. 5, 2020. [Online]. Available: <https://swift.org/>
- [9] Google Inc. *Swift for Tensorflow*. Accessed: Feb. 5, 2020. [Online]. Available: <https://www.tensorflow.org/swift>
- [10] M. K. Prasad, P. Chong, and K. Keutzer, "Why is ATPG easy?" in *Proc. 36th Annu. ACM/IEEE Design Autom. Conf.*, Jun. 1999, pp. 22–28.
- [11] M. Shalan and T. Edwards, "Building OpenLANE: A 130 nm OpenROAD-based tapeout-proven flow," in *Proc. Int. Conf. Comput. Aided Design (ICCAD)*, 2020, pp. 1–6.
- [12] Wilson Snyder. *Verilator*. Accessed: Jun. 15, 2020. [Online]. Available: <https://www.veripool.org/wiki/verilator>.

**Manar Abdelatty** is a Research Associate with the Computer Science and Engineering Department, The American University in Cairo, Cairo, Egypt. Her research interest includes embedded systems and development of electronic design automation software. Abdelatty has a bachelor's degree from Computer Engineering Department, The American University in Cairo (2020).

**Mohamed Gaber** is a Research Associate with the Computer Science and Engineering Department, The American University in Cairo, Cairo, Egypt. His research interest includes the development of electronic design automation software. Gaber has a bachelor's degree from Computer Engineering Department, The American University in Cairo (2020).

**Mohamed Shalan** is an Associate Professor (with tenure) with the Department of Computer Science and Engineering, The American University in Cairo (AUC), Cairo, Egypt. His research interests include OpenSource EDA, embedded systems, Internet of Things (IoT), and low-power computing systems. Shalan has a PhD in computer engineering from Georgia Institute of Technology, Atlanta, GA (2003).

■ Direct questions and comments about this article to Manar Abdelatty, School of Sciences and Engineering, The American University in Cairo, New Cairo 11835, Egypt; manarabdelatty@aucegypt.edu.

# PyH2: Using PyMTL3 to Create Productive and Open-Source Hardware Testing Methodologies

**Shunning Jiang, Yanghui Ou, Peitian Pan,  
Kaishuo Cheng, Yixiao Zhang,  
and Christopher Batten**

Cornell University

*Editor's notes:*

This article proposes a new model testing and verification methodology, PyH2, using property-based random testing in Python. PyH2 leverages the whole Python ecosystem to build test benches and models.

—Sherief Reda, Brown University

—Leon Stock, IBM

—Pierre-Emmanuel Gaillardon, University of Utah

■ **AS DENNARD SCALING** is over and Moore's law continues to slow down, modern system-on-chip (SoC) architectures have been moving toward heterogeneous compositions of general-purpose and specialized computing fabrics. This heterogeneity complicates the already challenging task of SoC design and verification. Building an open-source hardware community to amortize the nonrecurring engineering effort of developing highly parametrized and thoroughly verified hardware blocks is a promising solution to the heterogeneity challenge. However, the widespread adoption of open-source hardware has been obstructed by the scarcity of such high quality blocks. We argue that a key missing piece in the open-source hardware ecosystem is comprehensive, productive, and open-source verification methodologies that reduce the effort required to create

Digital Object Identifier 10.1109/MDAT.2020.3024144

Date of publication: 14 September 2020; date of current version: 8 April 2021.

thoroughly tested hardware blocks. Compared to closed-source hardware, verification of open-source hardware faces several significant challenges.

First, closed-source hardware is usually owned and maintained by compa-

nies with dedicated verification teams. These verification engineers usually have many years of experience in constraint-based random testing using a universal verification methodology (UVM) with commercial SystemVerilog simulators. However, open-source hardware teams usually follow an agile test-driven design approach stemming from the open-source software community, where the designer is also responsible for creating the corresponding tests. Moreover, the steep learning curve, in conjunction with very limited support in existing open-source tools, makes the UVM-based approach rarely used by open-source hardware teams. We argue that the open-source hardware community is in critical need of an alternative route for testing open-source hardware, instead of simply duplicating closed-source hardware testing frameworks.

Second, unlike closed-source hardware's development cycle where most engineers focus on a specific design instance for the next generation product, open-source hardware blocks

usually exist in the form of design *generators* to maximize reuse across the community [1]. However, design generators are significantly more difficult to verify than design instances due to the combinatorial complexity in the multidimensional generator parameter space. There is a critical need to create an open-source framework that systematically and productively tests design generators and automatically simplifies both failing test cases and failing design instances to facilitate debugging.

Third, performing random testing can be difficult in important hardware domains. There has been a major surge in open-source RISC-V processor implementations. However, due to limited human resources, most of these implementations only include a few directed tests, randomly generated short assembly sequences, and/or very large scale system-level tests (e.g., booting Linux). There is a critical need to create an automated random testing framework to improve the fidelity of open-source processor implementations.

Fourth, many open-source hardware blocks are designed to improve reusability by exposing well-encapsulated timing-insensitive hand-shake interfaces that can provide an object-oriented view of the hardware block (e.g., a hardware reorder buffer exposes three object-oriented “method” interfaces: *allocate*, *update*, and *remove*). However, it is very hard to perform random testing to test the behavior of concurrent hardware data structures that have multiple interfaces accepting “transactions” in the same cycle. Converting a random transaction sequence into cycle-by-cycle test vectors using traditional testing approaches requires a cycle-accurate golden model. Manually creating multitransaction test-vectors only works for directed testing. One possible solution is to execute only one random transaction in each cycle, yet the inability to stress intracycle concurrent behavior harms the quality of the tests. There is critical need to create a novel testing approach for object-oriented hardware using concurrent intracycle transactions.

To address these challenges, we introduce PyH2,<sup>1</sup> our vision for a productive and open-source testing methodology for open-source hardware, which is significantly different from state-of-the-art closed-source

<sup>1</sup>Python’s hypothesis for hardware.

hardware testing. Leveraging open-source software, PyH2 attempts to solve the open-source hardware testing challenge by holistically using property-based testing (PBT) in Python to significantly reduce designer effort in creating high-quality tests. The advantage of PBT over constraint-based random testing is as follows.

- PBT does not draw all of the random data beforehand, making it possible to leverage runtime information to guide the random data generation.
- PBT can automatically shrink the failing test case to a minimal failing case once a bug is discovered.

Compared to BlueCheck [2], a prior PBT framework for hardware, the key distinctions are as follows.

- PyH2 enables using a high-level behavioral specification written in Python as the reference model instead of requiring the reference model to be synthesizable.
- The random byte-stream internal representation of hypothesis provides more sophisticated auto-shrinking, while BlueCheck simply removes transactions along with *ad hoc* iterative deepening.
- PyH2 can auto-shrink not only the transactions but also the design itself by unifying the design parameter space and the test-case space.

We see coverage-guided mutational fuzzing (e.g., RFUZZ [3]) as complementary to PBT. PBT can be used to quickly find bugs with moderate complexity, while RFUZZ can be used to very slowly find potentially more complex bugs. Overall, PyH2 is able to combine the advantages of complete-random testing (CRT) and iterative-deepened testing (IDT) to identify a failing test case quickly and then provide a minimal failing case to facilitate debugging.

PyH2 is supported by the whole Python ecosystem, among which three main packages form the foundation of PyH2 (PyMtl3, pytest, and hypothesis). PyH2 users can use over 100,000 open-source Python libraries to build test benches and golden models. PyH2 leverages PyMtl3 [4], [5] to build Python test benches to drive register-transfer-level (RTL) simulations with PyMtl3 models and/or external SystemVerilog models leveraging PyMtl3’s Verilator cosimulation support. PyH2 adopts pytest, a mature full-featured Python testing tool, to collect, organize, parametrize, instantiate, and refactor test cases for



testing open-source hardware. PyH2 also exploits pytest plugins to evaluate hardware-specific testing metrics. For example, PyH2 tracks the line coverage of behavioral logic blocks of PyMTL3 models during simulation using coverage.py, a line coverage tool for normal Python code. The key component of PyH2 is hypothesis, a PBT framework to test Python programs by intelligently generating random test cases and rapidly auto-shrinking failing test cases.

PyH2 is realized by a collection of PyH2 frameworks which are discussed in depth in the rest of this article: PyH2G (PyH2 for RTL design generators), PyH2P (PyH2 for processors), and PyH2O (PyH2 for object-oriented hardware).

## Background

This section briefly introduces PyMTL3, pytest, and hypothesis, the three key Python libraries that form the foundation of PyH2.

### PyMTL3

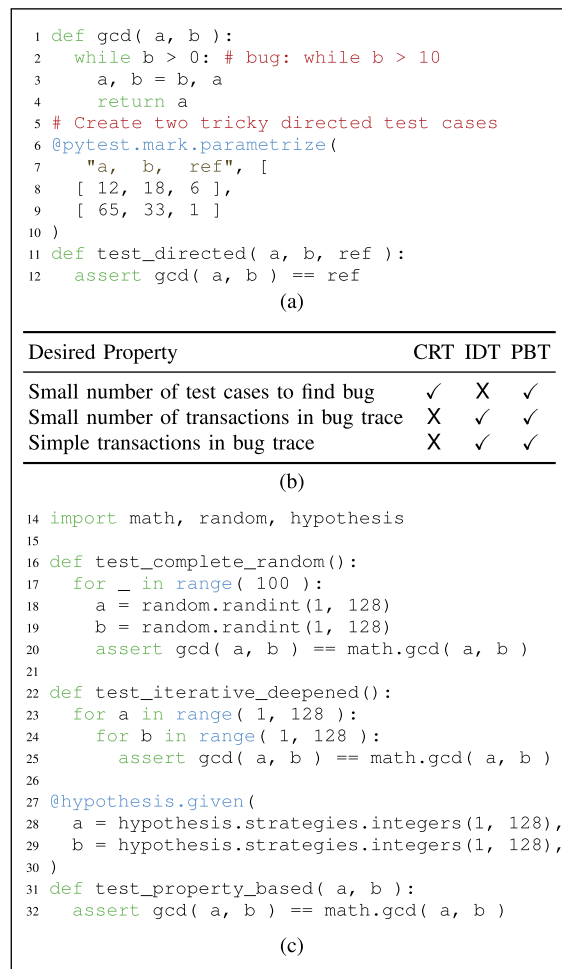
PyMTL3 is an open-source Python-based hardware modeling, generation, simulation, and verification framework. PyMTL3 supports multilevel modeling for RTL, cycle-level, and functional-level models. To provide productive, flexible, and extensible workflows, PyMTL3 is designed to be strictly modular. Specifically, PyMTL3 separates the PyMTL3 embedded domain-specific language that constructs PyMTL3 models, the PyMTL3 in-memory intermediate representation (IMIR) that systematically stores hardware models and exposes APIs to query/mutate the elaborated model, and PyMTL3 passes that are well-organized programs to analyze, instrument, and transform the PyMTL3 IMIR.

PyMTL3 aims at creating an evolving ecosystem with its modern software architecture and high interoperability with other open-source tools. PyMTL3 emphasizes performing simulation in the Python runtime and automatic Verilator black-box import for cosimulation. Driving the simulation from Python test benches to test both PyMTL3 designs and external SystemVerilog modules enables PyMTL3 to combine the familiarity of Verilog/SystemVerilog with the productivity features of Python. Tools that take the opposite approach (e.g., cocotb) embed Python in a Verilog simulator and drive the simulation from the Verilog runtime, but this complicates the ability to leverage the full power of Python. RTL designs built in PyMTL3 can be translated to SystemVerilog accepted by

commercial EDA tools, or Yosys-compatible Verilog accepted by OpenROAD, a state-of-the-art open-source EDA flow [6].

### PyTest

pytest is a mature full-featured tool for testing Python programs. Using pytest, the programmer can create small tests with little effort and also parametrize numerous complex tests with compositions of pytest decorators succinctly as shown in Figure 1a. pytest also provides lightweight command line options to print out different kinds of error messages varying from a list of characters indicating



**Figure 1. Background on testing methodologies. (a) Parametrizing directed tests using a pytest decorator. (b) Comparison of different testing techniques. (c) Code for testing a greatest common divisor function using CRT, IDT, and PBT.**

whether each test fails, to per-test full stack traces. `pytest` has hundreds of plugins, such as `pytest-cov` that leverages `coverage.py` to track line coverage.

### CRT, IDT, and hypothesis PBT

Traditional testing methodologies usually use a mix of CRT and IDT. As shown in Figure 1b, CRT can detect errors quickly because it randomly samples the input space, but can produce very complicated failing test cases which are difficult to debug. IDT finds bugs more slowly because it gradually samples the input space, but can produce simple counterexamples. PBT, first popularized by QuickCheck [7], is a high-level, black-box testing technique where one only defines *properties* of the program under test and uses *search strategies* to create randomized inputs. The original QuickCheck paper also discussed the integration with Lava [8] to test circuits. Properties are essentially partial specifications of the program under test and are more compact and easier to write and understand than full system specifications. Users can make full use of the host language when writing properties and thus can accurately describe the intended behavior. Most PBT tools support *shrinking*, a mechanism to simplify failing test cases into a minimal reproducible counterexample. With these features, PBT can achieve the benefits of both CRT and IDT.

Hypothesis [9] is a state-of-the-art Python PBT library that includes built-in search strategies for different data types and supports integrated auto-shrinking of failing test cases. All hypothesis strategies are built on top of a unified random byte-stream representation, and each strategy internally repurposes random bytes to produce the target random value. Search strategies in hypothesis are integrated with methods that describe how to simplify certain types of data, which makes shrinking effective. Users can compose built-in search strategies for any user-defined data type and shrinking will work out-of-the-box.

Complicated stateful systems can also be tested with `RuleBasedStateMachine` in hypothesis. The user inherits from the `RuleBasedStateMachine` class to add variables, a prologue, and an epilogue to create a new test class. The user needs to define rules and their preconditions and invariants, which describes conditional state transitions. For stateful testing, usually the user creates Python assertions inside the rule to compare against a golden reference model. Hypothesis repeatedly instantiates

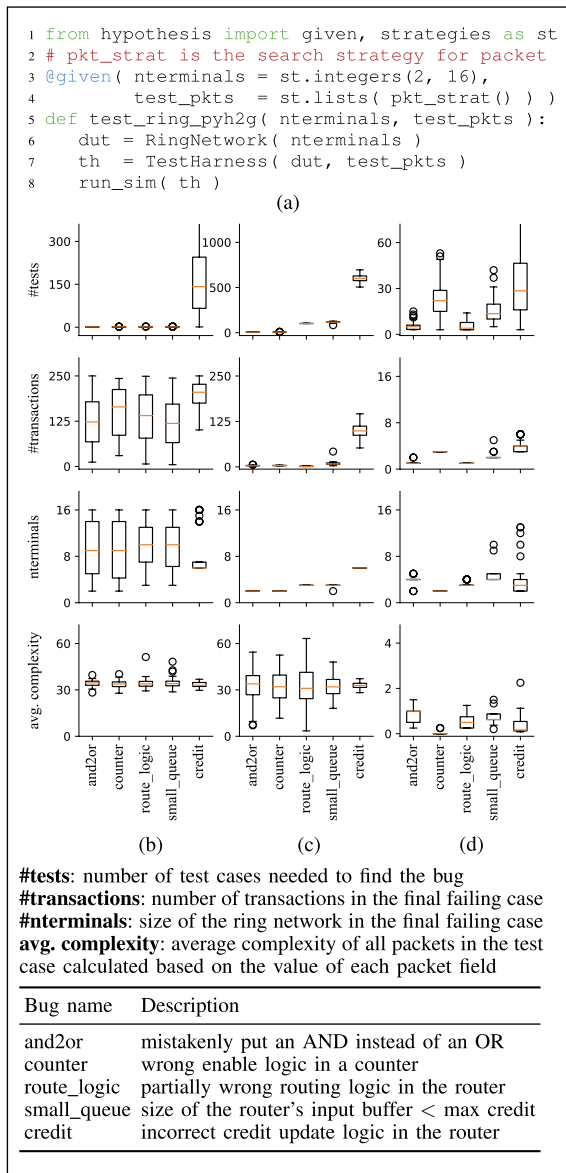
the test class and executes a sequence of rules on the state machine.

Figure 1c shows examples of testing the greatest common divisor function using CRT, IDT, and hypothesis PBT against `math.gcd`. The CRT test (lines 16–20) includes 100 random samples. The IDT test (lines 22–25) iteratively tries all possible values for  $a$  and  $b$  from 1 to 128. We use the `@hypothesis.given` decorator to transform a normal function test `_property_based` that accepts arguments, into a randomized PBT test. Consider a bug where line 3 in Figure 1a is changed to `while b > 10`. CRT can find the bug quickly, but the failing test case involves relatively large numbers. IDT finds the bug in exactly 11 test cases [i.e., `gcd(1,11)`]. PBT can find the bug quickly with large numbers, but then auto-shrink the inputs to a minimal counterexample [i.e., `gcd(2,1)`].

### PyH2G: PyH2 for RTL design generators

PyH2G is a PyH2 framework to productively and effectively test *RTL design generators*. We envision that future open-source SoC designs are heavily based on chip generators which are composed of numerous highly parametrized RTL design generators. Unfortunately, verifying design generators is significantly more challenging than verifying design instances due to the *combinatorial explosion* in the multidimensional generator parameter space. Traditional testing techniques such as CRT and IDT face new challenges when testing design generators. CRT can find a bug quickly with a few test cases but often leads to a complicated failing test case with numerous transactions *and a complex design*, which makes it more difficult to debug. IDT can produce a simple failing case with a small design instance, but may take a very long time to detect the error due to the iterative deepening required for the generator parameters.

In response to these challenges, PyH2G uses PBT to obtain the benefits of both CRT and IDT. Specifically, PyH2G creates composite search strategies in hypothesis to interpret part of the generated random byte stream as the design parameters and the rest as the test case (see lines 3–4 of Figure 2a). Unifying the design parameter space and the test case space allows hypothesis to simultaneously shrink the design parameters (i.e., reducing the complexity of the generated design instance), the length of the input transaction sequence, and the complexity of each transaction to a minimal failing test case.



**Figure 2. PyOCN RingNet generator case study. (a) PyH2G example. (b) CRT. (c) IDT. (d) PyH2G.**

### Case study: on-chip network generator

We quantitatively evaluated CRT, IDT, and PyH2G using the PyOCN [10] ring network generator against four real-world bugs. PyOCN is a multitopology, modular, and highly parametrized on-chip network generator built in PyMTL3. Figure 2a illustrates an example of a PyH2G test that uses search strategies to configure the ring network and generate the test packets. When a test case fails, hypothesis can simultaneously shrink the design instance and the packet sequence. We ran

50 trials for each bug, and the results are shown as box-and-whisker plots in Figure 2b–d. Overall, PyH2G detects errors quickly with a small number of test cases and produces a simple failing test case that has a short sequence of transactions and a simple design. PyH2G also significantly reduces the transaction complexity. PyH2G sometimes runs slightly more test cases than CRT because hypothesis will first generate explicit examples to stress-test the boundary conditions before exploring values randomly. However, this also help PyH2G discover the credit bug more quickly than CRT.

### PyH2P: PyH2 for processors

PyH2P is a PyH2 framework to automatically generate random assembly instruction sequences to test processors, which makes the case for effective domain-specific random testing methodologies. Different from existing work, PyH2P is able to automatically shrink a failed long program to a minimal instruction sequence with a minimal set of architectural registers and memory addresses. It is possible to combine auto-shrinking with other sophisticated random program generators [11] by carefully using PyH2P random strategies. PyH2P can also leverage Symbolic-QED [12] by applying QED transformations to generated random programs and performing bounded model checking to accelerate bug discovery.

PyH2P creates composite hypothesis strategies to generate random assembly programs for effective auto-shrinking. Specifically, PyH2P creates a hierarchy of strategies for arithmetic, memory, and branch instruction strategies using substrategies for architectural registers, memory addresses, and immediate values. PyH2P currently implements a block-based mechanism which first instantiates a control-flow template of branches, and then fills random instructions between branches. PyH2P ensures that each generated assembly program has well-defined behavior across the test and reference models. For arithmetic instructions, PyH2P constrains the range of the immediate value strategy to avoid overflow. For memory instructions, PyH2P constrains the range of the memory address strategy to avoid unaligned and out-of-bound memory accesses. For branch instructions, PyH2P first generates a sequence of branch instructions and their corresponding labels, and then randomly shuffles them to form the control-flow template. This eliminates the possibility of branch

out-of-range errors. Additionally, a set of registers are dedicated to loop bounds and loop variables to avoid infinite loops.

### Case study: PicoRV32 processor

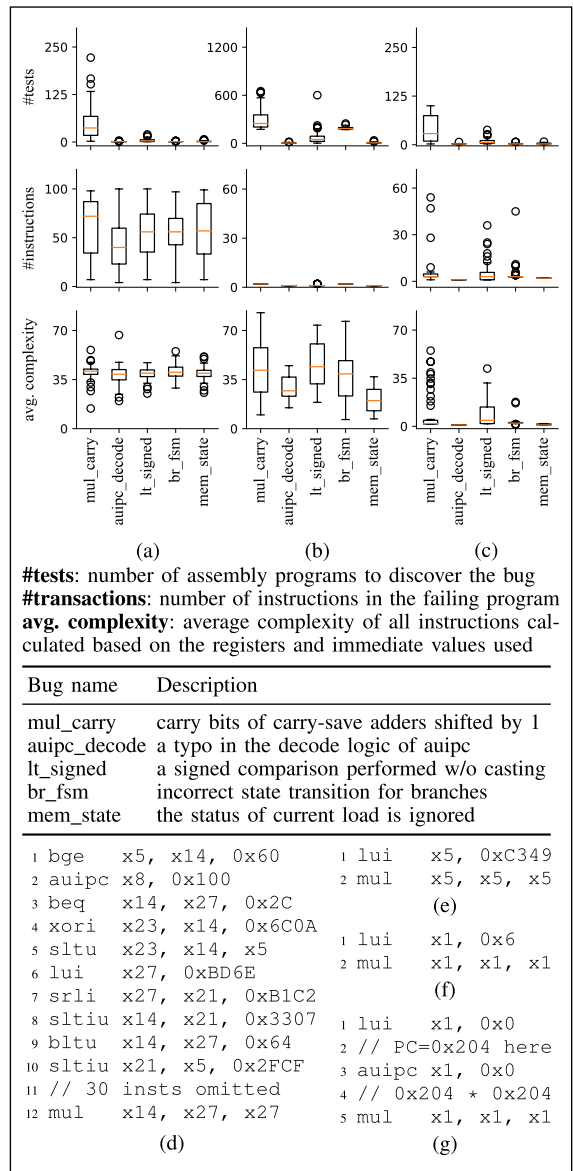
We demonstrate the effectiveness of PyH2P using PicoRV32, an open-source, area-optimized RV32IMC processor implemented in Verilog. We leverage PyMTL3's Verilator support to drive the cosimulation using a PyMTL3 testbench. The imported processor is connected to a PyMTL3 cycle-level test memory which stores the assembly program generated by PyH2P. After executing the program, we extract and compare the value of PicoRV32 architectural registers and the test memory against an instruction set simulator written in PyMTL3.

We inject five directed bugs into the Verilog code, and ran 50 trials for each methodology and bug combination. The results are shown as box-and-whisker plots in Figure 3a–c. CRT generally requires a small number of tests (less than to discover a bug, but the failing cases usually include more than 50 complex instructions. IDT significantly reduces the number of instructions in the failing test case, but needs significantly more cases to find the failing case. Note that IDT generates instructions of similar complexity to CRT because we have to generate random immediate values to avoid prohibitively long runtimes to find these bugs. PyH2P is able to discover the failing test case using a similar number of trials to CRT and can shrink it to a minimal case with similar length to the cases found by IDT. Moreover, PyH2P is able to shrink the immediate value so that the average instruction complexity is significantly reduced.

Figure 3d–g shows the failing cases for the `mul_carry` bug discovered by each methodology. This bug can only be triggered by specific operands. Figure 3d is the example found by CRT with 41 instructions, seven unique architectural registers, and large immediate values. Figure 3e shows the example found by IDT which uses only one register but a large random immediate value. Figure 3f and g includes two minimal failing cases from different PyH2P trials, which are significantly simpler.

### PyH2O: PyH2 for object-oriented hardware

PyH2O is a PyH2 framework that enables using method calls to test RTL hardware components with object-oriented latency-insensitive interfaces.



**Figure 3. PicoRV32 processor case study. (a) CRT. (b) IDT. (c) PyH2P. (d) CRT example. (e) IDT example. (f) PyH2P example 1. (g) PyH2P example 2.**

The key contribution of PyH2O is a novel testing methodology for concurrent hardware data structures that are difficult to thoroughly test using traditional approaches. PyH2O proposes a novel simulation mechanism called *auto-ticking*, which has been implemented as a new PyMTL3 simulation pass. With merely “transaction-accurate” Python data structures as reference models, PyH2O uses the rule-based stateful testing features in hypothesis to perform a sequence of random method calls on both

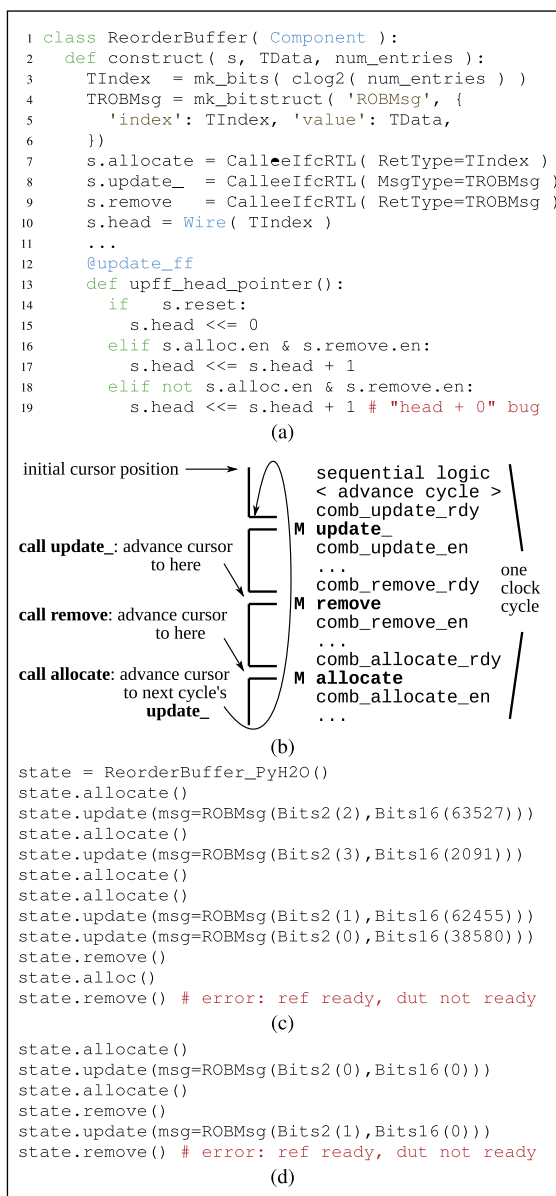
the reference model and the auto-ticking simulator of the RTL model, and then checks if the outcomes match for each method call.

PyH2O is based on method-based interfaces which are decoupled handshake interfaces with four ports: 1) enable; 2) ready; 3) arguments; and 4) return value. Essentially, setting the enable signal high after making sure the ready signal is high is equivalent to calling the corresponding ready method, checking if it returns true, and then calling the actual method. Converting an RTL method interface to a Python method involves an adapter that provides a method and a ready method to the user and sets/modifies the signals inside the adapter. PyH2O leverages Python reflection to automatically wrap the RTL method interfaces with a generated top-level PyMTL3 wrapper with Python methods.

PyH2O applies the AutoTickSimPass to create an auto-ticking simulator for the wrapped model. Conceptually, auto-ticking is more fine-grained than the classical delta cycle approach. Auto-ticking divides the combinational logic into multiple parts based on logic related to the method interfaces. When the user calls the enhanced top-level method, not only the method but also all the logic between this method and the next method is executed. If the executed method is the last method of the cycle, the simulator advances to the first method of the next cycle. If the user skips a method in this cycle and calls another method later in the cycle or a previous method that is already skipped/called in the current cycle, the simulator ignores the in-between methods and executes all the logic until it reaches the called method. Unlike trivial one-method-per-cycle testing, this auto-ticking scheme is able to execute multiple methods in the same cycle if they are called in a specific order.

#### Case study: reorder buffer data structure

Figure 4a shows an RTL reorder buffer implementation which exposes the three methods called interfaces. `allocate` is ready if the buffer is not full. It returns the entry index and advances the tail pointer. `update_` is ready if the buffer has valid elements. It takes an index/value pair to update the buffer. `remove` is ready if the buffer head is valid and already updated, and returns the index/value pair. Note that `remove` and `allocate` can occur in the same cycle even if the reorder buffer is full, because the implementation combinational



**Figure 4. PyH2O reorder buffer case study. (a) PyMTL3 reorder buffer code snippet. (b) Auto-tick execution schedule for reorder buffer. (c) First falsifying example found by PyH2O. (d) Minimized failing case after auto-shrinking.**

factors whether `remove` is called into `allocate`'s ready signal. Figure 4b shows the execution schedule generated by the AutoTickSimPass. The auto-ticking simulator guarantees that a sequence of three method calls in the order of `update_ < remove < allocate` will occur in the same cycle.

To show the effectiveness of PyH2O, we replace head+1 with head+0 in line 19 of Figure 4a. This subtle bug needs at least six transactions in a specific order to trigger because it requires six transactions to allocate, update and remove two entries, but must not remove the first one and allocate the second one in the same cycle. After trying several sequences with varying length from 5 to 19, PyH2O discovers a 11-transaction failing case as shown in Figure 4c. After auto-shrinking, PyH2O successfully finds one of the minimum failing case as shown in Figure 4d.

**THIS ARTICLE HAS** introduced PyH2, which leverages PyMTL3, pytest, and hypothesis to create a novel open-source hardware testing methodology. We believe PyH2 is an important first step toward addressing four key challenges in open-source hardware testing as follows.

- PyH2 is more accessible to open-source hardware designers compared to complex closed-source hardware testing methodologies.
- PyH2G is well-suited for testing not just design instances but also design generators which are critical to the success of the open-source hardware ecosystem.
- PyH2P can improve the random testing of open-source processor implementations compared to the more limited directed and random testing currently used in many open-source projects.
- PyH2O can more effectively test object-oriented hardware data structures.

We have open-sourced PyMTL3 and PyH2 at <https://github.com/pymtl/pymtl3>. ■

## Acknowledgments

Shunning Jiang and Yanghui Ou contributed equally to this work. This work was supported in part by NSF CRI under Award 1512937, in part by DARPA POSH under Award FA8650-18-2-7852, a research gift from Xilinx, Inc., and the Center for Applications Driving Architectures (ADA), one of six centers of JUMP, a Semiconductor Research Corporation program cosponsored by DARPA, as well as equipment, tool, and/or physical IP donations from Intel, Xilinx, Synopsys, Cadence, and ARM. We acknowledge and thank Derek Lockhart for his initial thoughts on combining PyMTL with hypothesis, and Cheng

Tan for his contributions to PyH2G. We would like to acknowledge and thank David MacIver and Zac Hatfield-Dodds for their work on the Hypothesis framework and thoughtful discussions on how to leverage Hypothesis for hardware. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of any funding agency.

## References

- [1] O. Shacham et al., "Avoiding game over: Bringing design to the next level," in *Proc. 49th Annu. Design Autom. Conf. (DAC)*, Jun. 2012, pp. 623–629.
- [2] M. Naylor and S. Moore, "A generic synthesizable test bench," in *Proc. ACM/IEEE Int. Conf. Formal Methods Models for Codesign (MEMOCODE)*, Sep. 2015, pp. 128–137.
- [3] K. Laeuffer et al., "RFUZZ: Coverage-directed fuzz testing of RTL on FPGAs," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2018, pp. 1–8.
- [4] S. Jiang, B. Ilbeyi, and C. Batten, "Mamba: Closing the performance gap in productive hardware development frameworks," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.
- [5] S. Jiang et al., "PyMTL3: A Python framework for open-source hardware modeling, generation, simulation, and verification," *IEEE Micro*, vol. 40, no. 4, pp. 58–66, Jul. 2020.
- [6] T. Ajayi et al., "Toward an open-source digital flow: First learnings from the OpenROAD project," in *Proc. 56th Annu. Design Autom. Conf. (DAC)*, Jun. 2019, pp. 1–4.
- [7] K. Claessen and J. Hughes, "QuickCheck: A lightweight tool for random testing of Haskell programs," in *Proc. Int. Conf. Funct. Program. (ICFP)*, Sep. 2000, pp. 268–279.
- [8] P. Bjesse et al., "Lava: Hardware design in Haskell," in *Proc. Int. Conf. Funct. Program. (ICFP)*, Sep. 1998, pp. 174–84.
- [9] D. MacIver et al., "Hypothesis: A new approach to property-based testing," *J. Open Source Softw.*, vol. 4, no. 43, p. 1891, Nov. 2019.
- [10] C. Tan et al., "PyOCN: A unified framework for modeling, testing, and evaluating on-chip networks," in *Proc. Int. Conf. Comput. Design (ICCD)*, Nov. 2019, pp. 437–445.

[11] F. Corno et al., “Fully automatic test program generation for microprocessor cores,” in *Proc. Design Autom. Test Eur. (DATE)*, Mar. 2003, pp. 1006–1011.

[12] F. Corno et al., “Fully automatic test program generation for microprocessor cores,” in *Proc. Design Autom. Test Eur. (DATE)*, Mar. 2018, pp. 55–60.

**Shunning Jiang** is currently pursuing a PhD in electrical and computer engineering with Cornell University, Ithaca, NY. Jiang has a BS in computer science from Zhiyuan College, Shanghai Jiao Tong University, Shanghai, China (2015). He is a student member of IEEE.

**Yanghui Ou** is currently pursuing a PhD in electrical and computer engineering with Cornell University, Ithaca, NY. Ou has a BEng in electrical and computer engineering from the Hong Kong University of Science and Technology, Hong Kong (2018). He is a student member of IEEE.

**Peitian Pan** is currently pursuing a PhD in electrical and computer engineering with Cornell University, Ithaca, NY. Pan has a BS in computer science from Shanghai Jiao Tong University, Shanghai, China (2018). He is a student member of IEEE.

**Kaishuo Cheng** is currently a junior undergraduate student in computer science with Cornell University, Ithaca, NY.

**Yixiao Zhang** has a BS and an MEng in electrical and computer engineering from Cornell University, Ithaca, NY (2018 and 2019, respectively).

**Christopher Batten** is currently an Associate Professor of Electrical and Computer Engineering with Cornell University, Ithaca, NY. Batten has a BS in electrical engineering from the University of Virginia, Charlottesville, VA (1999), an MPhil in engineering from the University of Cambridge, Cambridge, U.K. (2000), and a PhD in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, MA (2010). He is a member of IEEE.

■ Direct questions and comments about this article to Christopher Batten, School of Electrical and Computer Engineering, Cornell University, Ithaca, NY 14853 USA; cbatten@cornell.edu.

# OpenTimer v2: A Parallel Incremental Timing Analysis Engine

**Tsung-Wei Huang**

University of Utah

**Chun-Xun Lin and Martin D. F. Wong**

University of Illinois at Urbana-Champaign

*Editor's notes:*

This article introduces a high-quality open-source static timing analysis engine that is capable of parallel incremental timing and that provides an efficient API to facilitate development of complex EDA tools.

—Sherief Reda, Brown University

—Leon Stock, IBM

—Pierre-Emmanuel Gaillardon, University of Utah

■ **STATIC TIMING ANALYSIS** (STA) is a pivotal step in the overall chip design flow. It verifies the expected timing behaviors and prevents chips from malfunctioning after tape-out [1]. Of all timing analysis applications, *incremental timing* is imperative for the success of timing-driven optimization flows, such as placement, routing, logic synthesis, and physical synthesis [2]. Optimization tools often call a timer millions of times in their inner loop to evaluate a transform or an algorithm. The timer must quickly and accurately answer timing queries to ensure slack integrity and timing closure after the circuit experiences one or more changes. The capability of a timer on both speed and accuracy fronts is crucial for reasonable turnaround time and performance.

To this end, we developed *OpenTimer*, a high-performance timing analysis tool in 2015 [4]. OpenTimer is an award-winning tool in the ACM

TAU Timing Analysis Contest (2014 through 2016) and has received many recognitions in the CAD community (golden timers in the IEEE/ACM ICCAD CAD Contests and the ACM TAU Contests [2]). OpenTimer is open-source, and we are committed to free sharing of our technical innovation to make EDA a better and open place

to engage more talented people contributing to the community [3]. So far, OpenTimer has been used in many industrial and academic projects such as Qflow, VSDflow, CloudV, DARPA IDEA, OpenDesign, LGraph, and Ophidian [5]–[9]. After four years of development, we announced a major release OpenTimer v2 [3]. We rewrote the codebase in modern C++17 and developed a new software architecture to facilitate the parallelization of incremental timing. The overview of the OpenTimer v2 software stack is shown in Figure 1. We summarize our contributions as follows.

- *New parallel task programming model:* We developed a new task-based programming model that enables efficient implementations of parallel decomposition strategies. The new model allows us to go beyond the traditional loop-based parallelization of incremental timing, thereby leading to more asynchrony and faster runtime.
- *New software architecture and API concept:* We developed the core timing routines around three

Digital Object Identifier 10.1109/MDAT.2021.3049177

Date of publication: 5 January 2021; date of current version: 8 April 2021.



concepts, *builder*, *action*, and *accessor*. This separation allows OpenTimer v2 to exploit parallelism from both intra and inter operations, followed by efficient lazy evaluation.

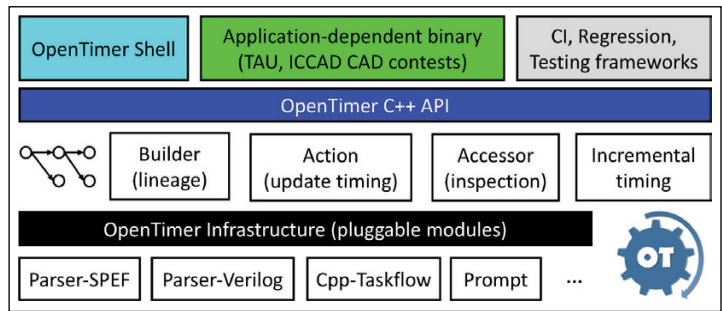
- *New parallel incremental timing framework:* We developed a task-based incremental timing framework that propagates timing naturally with the structure of the timing graph. Our framework can simultaneously perform both graph-based analysis and path-based analysis in parallel while keeping accurate results without breaking complex dependencies between different timing propagation tasks.

Compared with the previous generation, OpenTimer v2 is faster and more scalable in increasing the graph size and the CPU count. The programming interface is also more succinct due to the new API concept. We have made many components modular to make OpenTimer v2 user-friendly and easier for developers to contribute to the codebase. These components include not only the core parallel incremental timing algorithms but also supporting readers/writers for SDC, liberty, and SPEF that can be beneficial for other EDA applications. We believe OpenTimer v2 stands out as a unique system considering the technical innovations and ensemble of software tradeoff and architecture decisions we have made. Recently, OpenTimer was selected as the Best Open-source EDA Tool Award in the 2018 WOSET at ICCAD (one out of 30) [10].

### Challenges of incremental timing

Developing an efficient parallel incremental timing engine is a notoriously challenging job, requiring in-depth knowledge of circuit, graph theory, parallel programming, and software engineering. We highlight the three aspects of the challenge we face:

- *Complex task dependencies:* Updating a timing graph takes on load capacitance, parasitics, slew, delay, arrival time, required arrival time, and more. These quantities are interdependent and are not economical to compute. The resulting task dependency in terms of encapsulated function calls is very large and complex.
- *Irregular compute pattern:* Updating a timing graph involves highly diverse computation patterns. We need to capture different forms of



**Figure 1. OpenTimer v2 software architecture [3].**

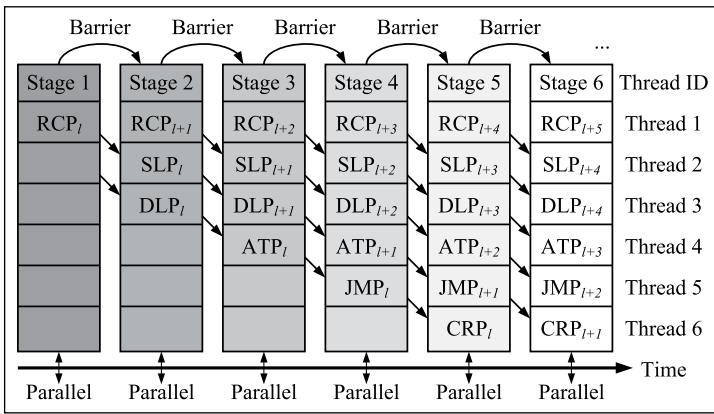
timing data whether it is structured in a local block or is flat in the global scope, to implement different delay calculators and pruning heuristics.

- *Unknown API practices:* Our user experience led us to believe that the *API concept* dominates the usability of a timer. When things go incremental, users and developers are often confused by the effect of each operation, such as the per-call complexity, parallelism, and consistency. This can significantly lift up the turnaround time and result in performance pitfall due to misunderstanding of API.

The extensibility and scalability to new technology is also an important factor to take into consideration while developing a general incremental timing framework. We are not only interested in technical innovations but also in the modularity of the software to provide a better user experience.

### Bottleneck in OpenTimer v1 and existing timers

One of the major differences between v1 and v2 is the parallelization of incremental timing. OpenTimer v1 and existing timers [11]–[13] dealt with incremental timing using *loop-based* parallelism [4]. In a rough view, we levelized the circuit into a topological order and applied the OpenMP “parallel for” directive to each node set level by level. This level-based decomposition is advantageous in its simple *pipeline* concept and is by far the most implementation in existing timers, including industrial tools. Figure 2 illustrates this strategy as an example of forward timing propagation. For each node, we update a number of dependent tasks including parasitics (RCP), slew (SLP), delay (DLP), arrival time (ATP), jump points (JMP), and pessimism reduction (RCP) [4]. However, this paradigm suffers from many performance drawbacks. For example, the



**Figure 2. Loop-based parallel timing propagations. Each level applies a parallel\_for to update timing from the fanin of each node [4].**

number of nodes can vary from level to level, resulting in highly unbalanced thread utilization. Also, there is a synchronization barrier between successive levels to impose task dependencies. The overhead can be large for graphs with long data paths. Furthermore, we found it difficult to add to the pipeline other analysis frameworks that require diverse modeling techniques, for example, signal integrity and cross-talk analysis.

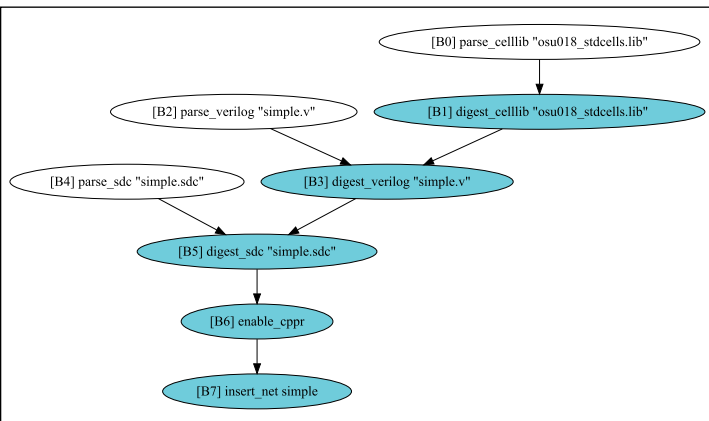
Big idea 1: A new parallel task programming model using modern C++

After many years of research, we came to a conclusion that the biggest hurdle to a scalable parallel timer is a suitable *parallel programming model*. In addition to the traditional loop-based approach, the programming model must be capable of *task-based* parallelism. In fact, we have tried multiple options, such as OpenMP

4.5 tasking and Intel Threading Building Blocks (TBB), that are commonly used in EDA applications. We found them unsuitable to our workload for various reasons. For instance, OpenMP 4.5 tasking is *static*. Unfortunately, it is difficult to decide the timing graph at the time of programming. The problem of TBB is the programmability. Users need to understand complex task constructs and templates that are often at low level and hard to maintain. Similar reasons exist in other libraries as well. Therefore, we decided to develop a new parallel task programming model using modern C++ technology. Although the original purpose was for incremental timing, we later generalized it to a standalone open-source project called Taskflow to benefit generic C++ developers [14]. Note that the proposed parallel task programming model is different from that mentioned in [15], which relies on a specialized scheduler to insert tasks dynamically into shared work queues. We focus on *static* modeling that maps the entire timing propagation graph into a task computation graph. When the graph is ready, the scheduler can perform whole-graph optimization and schedule tasks using work-stealing to achieve dynamic load balancing.

Big idea 2: A new API concept and software architecture

With Taskflow in place, we develop a new software architecture in OpenTimer v2 to enable efficient parallel incremental timing. We group each timing operation into one of the three categories, *builder*, *action*, and *accessor*. A timing operation can be either a C++ method in the timer class or command in our shell. Hereafter, the term OpenTimer refers to v2 unless otherwise specified.



**Figure 3. OpenTimer lineage example of five builder operations (cyan). Three parsing tasks run in parallel.**

Builder: OpenTimer lineage

A builder operation builds up a timing analysis environment, for example, reading cell libraries and a verilog netlist. OpenTimer maintains a lineage graph of builder operations to create a *task execution plan* (TEP). A TEP starts with no dependency and keeps adding tasks to the lineage graph each time users call a builder operation. It records what transformations need to be executed when an action operation is called.

Figure 3 shows an example of OpenTimer lineage. The lineage is made of five builder operations, `read_celllib`, `read_verilog`, `read_sdc`, `enable_cprr`, and `insert_net`. Each time users call a builder operation, the timer adds one or

multiple tasks to the lineage graph. These operations are not evaluated until an action operation is issued. The advantage of this is *fine-grained* task parallelism. An operation is divided into several smaller tasks that can run in parallel with other counterparts. For example, reading an input file can be broken into two subtasks, parsing the file and digesting the data into OpenTimer's in-memory model. It is obvious the parsing part can run in parallel with others as long as it precedes its corresponding digesting task. Maintaining a lineage of builder operations enables us to exploit both intra and interoperation parallelism, followed by efficient lazy evaluation. Another side benefit of the lineage is the engineering change order (ECO) capability. We can easily keep track of the modifiers for state recovery or debugging.

#### Action: Update timing

A TEP is materialized and evaluated when users request the timer to perform an action operation, for example, reporting the arrival time and the slack value of a pin. Calling an action operation triggers a timing update from the earliest task to the one that produces the result of the action call. Internally, we create a task dependency graph and update timing in parallel, including forward propagation (slew and arrival time) and backward propagation (required arrival time). Figure 4 shows an example of task dependency graph to update a timer. The bottom-most call of every action operation is the method `update_timing`. The method explores a minimum set of nodes in the timing graph as propagation candidates and constructs a task dependency graph to carry out the timing update. Our tasking model can incorporate different types of timing propagation into a task. Unlike the level-based approach in v1, a task can start immediately after all its preceding tasks finish. This largely enhances asynchrony, giving rise to higher CPU utilization, and faster runtime.

#### Accessor: Inspect OpenTimer

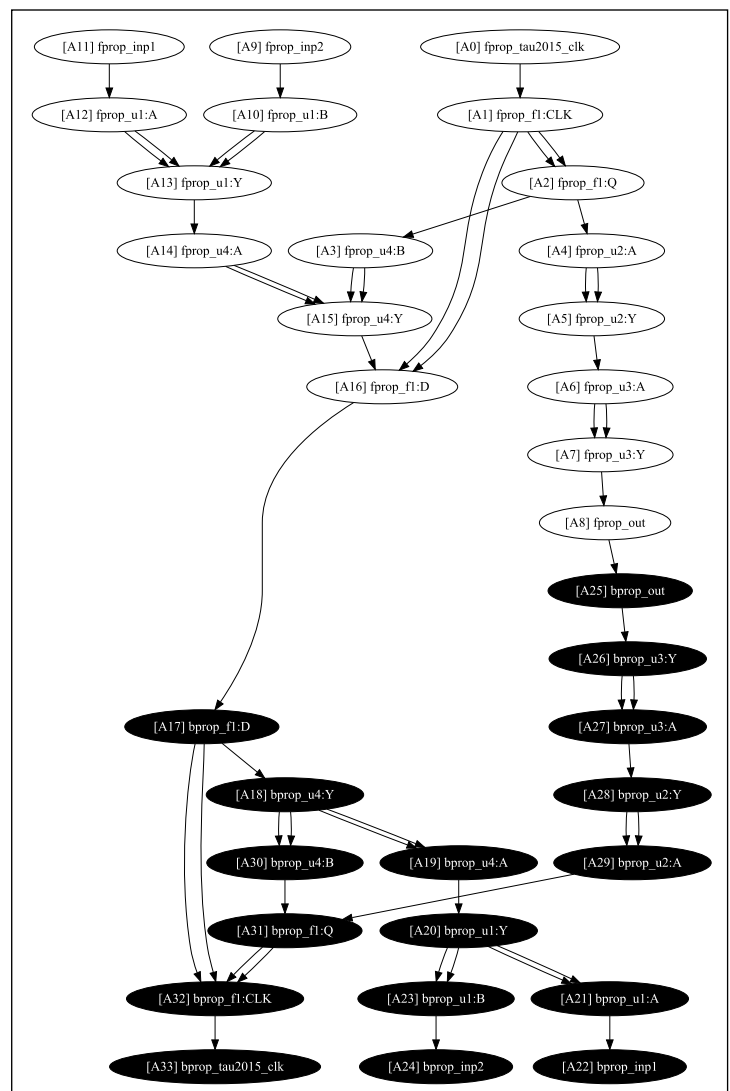
An accessor operation lets users inspect the timer status and dump *static* timing information, for example, dumping the timing graph for visualization purposes or dumping the design statistics. All accessor operations are declared as *constant* methods in the timer class. Calling an accessor method does not alter any internal data structures of a timer.

### Big idea 3: Parallel incremental timing analysis algorithms

We discuss, in this section, how OpenTimer performs graph-based analysis and path-based analysis.

#### Graph-based analysis

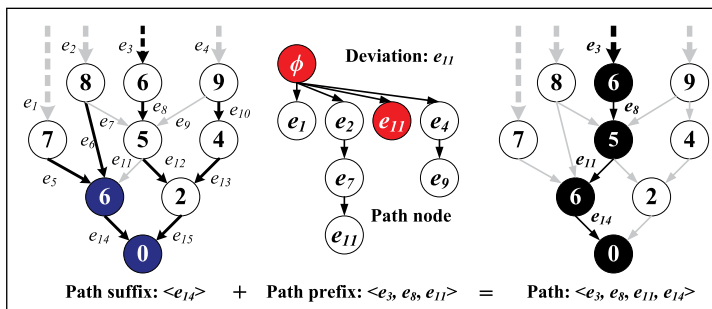
At the bottom of every action operation, OpenTimer calls `update_timing` to perform graph-based timing updates. The timer first evaluates the lineage (e.g., Figure 3) and discovers a list of *frontier* pins from which incremental timing should begin after a modification is applied [4]. We then identify the



**Figure 4. Example task dependency graph to carry out an action operation. The graph consists of forward propagation tasks (white) and backward propagation tasks (black).**

**Table 1. Accuracy comparison between OpenTimer v1 and v2 on TAU15 contest benchmarks [2].**

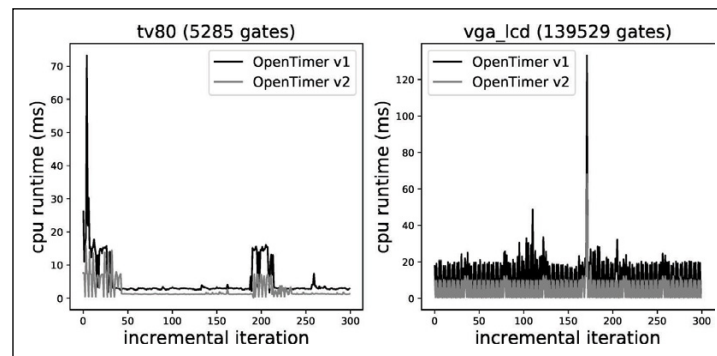
Circuit	#Gates	#Nets	#Ops	v1	v2
b19	255.3K	255.3K	5641.5K	99.95%	100%
cordic	45.4K	45.4K	1607.6K	98.88%	100%
des_perf	138.9K	139.1K	4326.7K	99.73%	100%
edit_dist	147.6K	150.2K	3368.3K	98.30%	100%
fft	38.2K	39.2K	1751.7K	99.77%	100%
netcard	1496.0K	1497.8K	11594.6K	99.99%	100%
softusb_navre	6.9K	7.0K	427.8K	99.97%	100%
tip_master	37.7K	38.5K	1300.4K	97.04%	100%

**Figure 5. OpenTimer applies implicit path representation based on a suffix tree and a prefix tree data structures per query to perform path-based analysis [16].**

propagation candidates (downstream and upstream of frontier pins) and derive a task dependency graph for graph-based timing update (e.g., Figure 4). Executing the task dependency graph autonomously triggers a parallel incremental timing update.

#### Path-based analysis

We developed our path-based analysis using the path generation algorithm by Huang and Wong [16]. To our best knowledge, this is by far the fastest

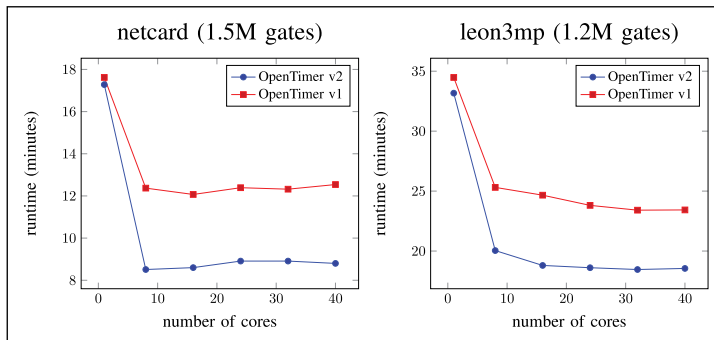
**Figure 6. Runtime comparison of incremental timing.**

algorithm in the literature. The algorithm consists of two complementary data structures, *suffix tree* and *prefix tree*. Each path is transformed into an implicit representation that takes constant space and time. The suffix tree represents the shortest path tree rooted at a given endpoint of the design. The prefix tree is a tree order of timing arcs each representing a unique path deviated from a timing arc. Generating the top-k critical paths across all endpoints is extremely efficient under this data structure. It also largely facilitates the parallelization as each pair of suffix tree and prefix tree is independent of each other at different endpoints. An example of the implicit path representation is shown in Figure 5.

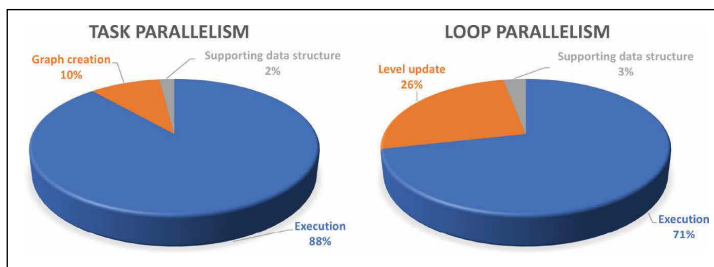
#### Experimental results

OpenTimer v2 is implemented in C++17 on a 40-core 3.2-GHz 64-bit Linux machine with 64-GB memory. We used G++ 8.0 with `-std=c++17` to compile the source. Experiments are undertaken on the TAU15 contest benchmarks with a golden reference generated by IBM Einstimer under static mode [2]. Table 1 compares the accuracy between OpenTimer v1 and v2 on a set of TAU15 contest benchmarks [2]. These benchmarks are where OpenTimer v1 failed to achieve full accuracy due to an implementation compromise between path generation and parallelization. The new software architecture in v2 lets us manage to resolve these issues and we are able to match the golden results completely. We did not observe too much runtime and memory difference between v1 and v2 on these benchmarks.

The TAU15 contest benchmarks have fewer than ten incremental timing iterations, making it hard to profile the performance. Therefore, we modified two circuits tv80 and vga\_lcd, on which both v1 and v2 acquire full accuracy, to incorporate 300 incremental timing iterations. In each iteration, we randomly modify the designs (e.g., `repower_gate`) and call `report_timing` to trigger incremental timing updates. As shown in Figure 6, v2 is consistently faster than v1 (2.14x on tv80 and 2.19x on vga\_lcd). About 64% of the speed-up came from replacing the pipeline-based parallelism with the new tasking framework. Figure 7 plots the runtime scalability of v1 and v2 over an increasing number of cores. Regardless of the core count, v2 is always faster than v1. Both saturates at about 8–12 cores.



**Figure 7. Runtime scalability with increasing number of CPU cores on two large circuits, netcard, and leon3mp.**



**Figure 8. Runtime profiling for task parallelism in OpenTimer v2 and loop parallelism in v1.**

The scalability is affected by many factors such as the graph structure and the size of incremental timing. A primary reason that prevents v2 from scaling beyond 12 cores is the data size. Most data for incremental timing are sparse. They do not span across large cones, as full timing, which produces a large amount of data for higher parallelism.

Figure 8 shows the runtime profiling for task-based approach in OpenTimer v2 and loop-based levelization in v1. We measure the time each significant portion of update\_timing takes in a piechart. Creating a task graph occupies about 10% of the entire runtime and executing the graph takes the majority of 88%. On the other hand, the loop-based approach spent up to 26% on updating the level list and the parallel execution of tasks across all levels takes 71%.

**IN THIS ARTICLE**, we presented OpenTimer v2—a new parallel incremental timing analysis tool. We have developed a new parallel task programming model and applied it to design an efficient parallel incremental timing framework. Also, we have introduced a new API concept that defines a clear

operation effect on top of our parallelization framework. The source of OpenTimer v2 is available at [3]. ■

## Acknowledgments

We would like to thank all OpenTimer users in providing their feedback, suggestions, and requests.

## References

- [1] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*, 2009th ed. New York, NY, USA: Springer, 2009, ISBN-13: 978-0387938196.
- [2] J. Hu, G. Schaeffer, and V. Garg, “TAU 2015 contest on incremental timing analysis,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015, pp. 895–902.
- [3] *OpenTimer*. Accessed: 2020. [Online]. Available: <https://github.com/OpenTimer/OpenTimer>
- [4] T.-W. Huang and M. D. F. Wong, “OpenTimer: A high-performance timing analysis tool,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015, pp. 895–902.
- [5] *Qflow*. Accessed: 2020. [Online]. Available: <http://opencircuitdesign.com/qflow/>
- [6] *CloudV*. Accessed: 2020. [Online]. Available: <https://cloudv.io/>
- [7] J. Jung et al., “DATC RDF: An academic flow from logic synthesis to detailed routing,” in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2018, pp. 37:1–37:4.
- [8] *LGraph*. Accessed: 2020. [Online]. Available: <https://github.com/masc-ucsc/lgraph>
- [9] *Ophidian*. Accessed: 2020. [Online]. Available: <https://gitlab.com/eclufsc/ophidian>
- [10] *WOSET*. Accessed: 2020. [Online]. Available: <https://github.com/woset-workshop/woset-workshop.github.io>
- [11] *OpenSTA*. Accessed: 2020. [Online]. Available: <https://github.com/abk-openroad/OpenSTA>
- [12] P.-Y. Lee, I. H.-R. Jiang, and T.-C. Chen, “FastPass: Fast timing path search for generalized timing exception handling,” in *Proc. 23rd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2018, pp. 172–177.
- [13] K. E. Murray and V. Betz, “Tatum: Parallel timing analysis for faster design cycles and improved optimization,” in *Proc. Int. Conf. Field-Programmable Technol. (FPT)*, Dec. 2018, pp. 110–117.

- [14] T.-W. Huang et al., “Cpp-taskflow: Fast task-based parallel programming using modern C++,” in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2019, pp. 974–983.
- [15] A. Mark Lavin et al., “Decentralized dynamically scheduled parallel static timing analysis,” U.S. Patent 20120311 514 A1, Jul. 8, 2014.
- [16] T.-W. Huang and M. D. F. Wong, “UI-timer 1.0: An ultrafast path-based timing analysis algorithm for CPPR,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 11, pp. 1862–1875, Nov. 2016.

**Tsung-Wei Huang** is currently an Assistant Professor with the Electrical and Computer Engineering (ECE) Department, University of Utah, Salt Lake City, UT. His current research interests focus on timing analysis and parallel processing. Huang has a BS and an MS from the Department of Computer Science, National Cheng Kung University (NCKU), Tainan, Taiwan (2010 and 2011, respectively), and a PhD in ECE from the University of Illinois at Urbana–Champaign (UIUC), Champaign, IL.

**Chun-Xun Lin** is currently pursuing a PhD with the Department of Electrical and Computer

Engineering (ECE), University of Illinois at Urbana–Champaign (UIUC), Champaign, IL. His research interests include VLSI CAD and parallel processing. Lin has a BS in electrical engineering from National Cheng Kung University, Tainan, Taiwan (2009) and an MS in electronics engineering from the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan (2011).

**Martin D. F. Wong** is currently the Dean of the Faculty of Engineering, Chinese University of Hong Kong (CUHK), Hong Kong. Wong has a BS in mathematics from the University of Toronto, Toronto, ON, Canada, and an MS in mathematics and a PhD in computer science (1987) from the University of Illinois at Urbana–Champaign (UIUC), Champaign, IL.

■ Direct questions and comments about this article to Tsung-Wei Huang, Department of Electrical and Computer Engineering, University of Utah, Salt Lake City, UT 84112 USA; [tsung-wei.huang@utah.edu](mailto:tsung-wei.huang@utah.edu).

# CATNAP-Sim: A Comprehensive Exploration and a Nonvolatile Processor Simulator for Energy Harvesting Systems

**Ali Hoseinghorban, Mohammad Abbasinia,  
Ali Paridari, and Alireza Ejlali**  
Sharif University of Technology

*Editor's notes:*

This article introduces an architecture exploration tool to study and understand the tradeoffs of future processor systems using nonvolatile memory and help guide the design of the future.

—Sherief Reda, Brown University

—Leon Stock, IBM

—Pierre-Emmanuel Gaillardon, University of Utah

**ENERGY EFFICIENCY SERVES** as a critical factor in designing embedded systems. Furthermore, the growing dominance of energy harvesting systems (EHSs) in sensor-rich applications has eventuated an emerging trend in wearable-systems and IoT devices as an alternative to battery-powered embedded systems. Most of these applications are designed for resource-constraint environments, and EHSs utilize ambient-energy sources such as solar, piezoelectric, RF, etc. [1]–[3].

State-of-the-art EHSs exploit nonvolatile memories (NVMs), such as ferroelectric random access memory (FRAM) instead of FLASH memory, to improve

*Digital Object Identifier 10.1109/MDAT.2021.3049176*

*Date of publication: 5 January 2021; date of current version:  
8 April 2021.*

energy efficiency and performance [4], [5]. NVMs have interesting features such as byte accessibility, low access latency, and low energy consumption compared to FLASH memories, which improves cost,

weight, and energy efficiency of state-of-the-art EHSs [6]. On the other hand, SRAM has lower access latency and energy consumption compared to NVMs, while it dissipates higher leakage power. Furthermore, SRAM cannot hold data in the case of power failure in EHSs [5]. To this end, the state-of-the-art EHSs exploit hybrid NVM-SRAM memory to take advantage of both memories. For instance, the MSP430FR family, commercially off-the-shelf microcontrollers by Texas Instruments, use hybrid FRAM-SRAM memory.

In EHSs, the processor runs an application, until the stored energy in the capacitor is higher than the cutoff threshold. When the capacitor's energy depletes, the system turns off the processor and turns it on again when the capacitor accumulates enough

energy. When the system turns off the processor, NVM keeps the data, while SRAM loses them; so, the processor must consider an appropriate check-pointing policy to avoid data inconsistency issues [1], [7]. Data inconsistency occurs when the processor fails to complete the program while some data are changed in the NVM. So, merely re-executing from the last check-point with modified data would result in wrong functionality.

In addition to check-pointing policy, several factors influence the performance and energy efficiency of a given EHS, such as ambient source power trace, capacitor size, and application memory access patterns. The use of real hardware platforms, i.e., MSP430FR microcontrollers for experiments, gives more reliable and accurate results that rule out all simulation errors. However, examining a diverse range of experimental conditions on hardware is difficult and time-consuming. Furthermore, the extraction of useful information such as the number of memory accesses, the number of check-points, and charging time of the system is either impossible or imposes a significant overhead.

In this article, we presented CATNAP-Sim,<sup>1</sup> a simulator for EHSs, with NVP and hybrid NVM-SRAM memory. We have integrated the energy and performance characteristics of MSP430FR5969 and Wang's power system model [6] into SimpleScalar [8]. We also implemented several check-pointing policies in CATNAP-Sim [1], [2], [7]. Furthermore, we comprehensively explore the design space and discuss the effects of exploiting SRAM, solar power strength, capacitor size, benchmark, and check-pointing policy on the EHSs.

The novel contributions of this work are listed as follows.

- Energy consumption of MSP430FR5969, a commercial off-the-shelf platform, is measured using Energy-Trace++, and the results are integrated into CATNAP-Sim.
- The low overhead, capacitor-less, converter-less, and state-of-the-art power system model [6] is integrated into CATNAP-Sim.
- CATNAP-Sim provides an exhaustive design exploration for various parameters, such as capacitor size, ambient power trace, check-pointing policy, and applications. We provide a set of

scripts to run simulations in parallel to improve the simulation time.

- We show how CATNAP-Sim marks important tradeoffs between charging time, execution time, failure rate, check-pointing overhead, and energy consumption for different check-pointing policies.

## Background and related works

NVPs exploit hybrid SRAM-NVM memory because the use of SRAM in the memory hierarchy improves the efficiency of the system significantly; however, SRAM loses its data in the case of a power failure. So, the system must consider an appropriate check-pointing policy to reduce the check-pointing overheads while eliminating the data inconsistency in the system. The inconsistency problem occurs when the processor modifies some data on the NVM, and cannot back up the system successfully. Therefore, re-execution from the last successful check-point might produce incorrect results. The check-pointing policy in the literature can be classified into four groups.

In the *software* check-pointing policy, the programmer divides the application into a set of tasks (functions), and the system backs up after the successful execution of each task [2]. In this policy, the programmer identifies each task's sensitive data during the development phase; in the online phase, the system copies the sensitive data to a temporary memory before the execution of each task. Therefore, the system can resolve the inconsistency problem with low energy and area overhead. However, setting the size of tasks requires precise knowledge about the system's energy consumption and capacitor, which might not be available in the development phase.

In the *compiler* check-pointing policy [1], the compiler analyzes the control flow graph of an application and inserts check-pointing instructions in different application lines. The compiler detects write-after-read (WAR) pairs targeting the same address and adds a check-point instruction between them to deal with the inconsistency problem. This approach guarantees the program's consistency without user interaction, while it imposes too many check-points to the application [7].

In the *low-voltage-threshold* check-pointing policy [5], the processor executes the program until it receives the capacitor's low-voltage interrupt from the capacitor's voltage controller; then, the system

<sup>1</sup>We released CATNAP-Sim as an open-source simulator, and it could be downloaded from <http://esrlab.ce.sharif.ir/download/CATNAP-Sim.gz.tar>.



starts to back up the SRAM. In this approach, the system performs backups only when necessary; so, the number of check-points is ideal. However, because of some nonideal factors like imperfect knowledge of the state-of-charge, unknown backup size, and input power fluctuations, setting a precise threshold to guarantee successful backup is a challenge. Finally, in the *watchdog* check-pointing policy [7], the system statically or dynamically sets a watchdog timer threshold, and when the timer overflows, the system creates a backup of the SRAM. The inconsistency problem in *low-voltage-threshold* and *watchdog* check-pointing policies is hard to resolve because the check-point locations are unknown in the development phase. In the case of failure, the processor needs to re-execute the application from scratch [5] or monitor the data modification in NVM [7], which increases the area and energy overheads of the system.

Bazzaz et al. [9] proposed an energy estimation model and a simulator to estimate battery-powered embedded systems' energy consumption. However, this simulator is not suitable for EHSs because there are many indispensable parameters, such as solar power fluctuation, capacitor size, and check-pointing policy, which are not considered in MEET and other simulators for battery-powered devices. Recent studies presented several models for EHSs and IoT devices [3], [6]. However, they did not provide a simulator to examine the behavior of state-of-the-art EHSs.

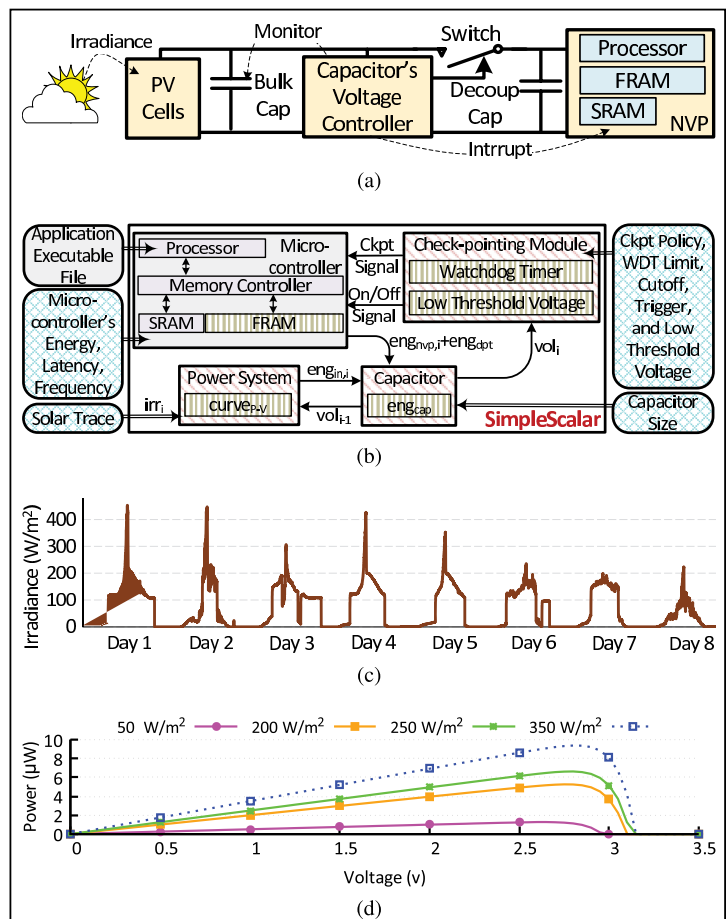
Several in-house simulators used in previous research studies are customized SimpleScalar [7] or gem5 [1] specifically for the focus of their particular research purposes; therefore, they cannot be used in general, all-inclusive approaches. Additionally, most of them modified the default architecture of the processor or memory subsystem; so, they could not evaluate their simulator's accuracy with real hardware. NVPSim [10] is a gem5-based simulator that analyzes the effects of cache memory for nonvolatile processors in EHSs. NVPSim considered an ideal capacitor as the energy buffer of the system, which can store any amount of energy while real capacitors saturate after accumulating a specific amount of energy [6]. NVPSim employs *low-voltage-threshold* as the system's check-pointing policy, but it does not investigate the inconsistency challenge. But CATNAP-Sim considers the effects of inconsistency on different check-pointing policies and supports all

four check-pointing policies. So, the user can apply a hybrid policy by configuring the simulator.

## CATNAP-Sim structure

EHSs have four main components, including the source of ambient power, processing unit, power system, and energy storage (Figure 1a). The power system scavenges the ambient solar power and accumulates it in the bulk capacitor. The controller monitors the voltage of the bulk capacitor and closes the switch when the voltage rises above the trigger voltage or opens it when the capacitor voltage falls below the cutoff voltage. The controller raises an interrupt to the processor when the capacitor's voltage falls below the low threshold.

As we mentioned previously, data inconsistency is an important challenge in EHSs with NVM



**Figure 1. Overview of typical EHS components and the structure of CATNAP-Sim. (a) Components of an EHS, (b) structure of CATNAP-Sim, (c) solar power traces during eight days [11], and (d) P-V curve [6] of the power system.**

[1], [2], [7]. To address and solve data inconsistency challenges, a simulator is required to monitor changes in the memories. Furthermore, to consider the effect of high access latency of NVMs, a simulator is needed to evaluate the performance and energy efficiency of different strategies such as scratchpad memory (SPM) allocation, cache memory architectures, and eviction policies in EHSs [2], [5]. In the SimpleScalar timing model, all instructions except for loads and stores are executed in one cycle. The latency of load and store instructions depends on which memory type (SRAM or NVM) the requested data are accessed [8]. In this article, in line with previous simulators, such as MEET [9] and NVPSim [10], we modified an existing ARM simulator called SimpleScalar. So, we kept the SimpleScalar simulator modeling intact, but provided additional functions for modeling the input power, check-pointing policies, and NVM. We used Sim-profile mode of the SimpleScalar version 3.0. This version supports ARM7 integer instruction set and FPA floating-point extensions.

The structure of CATNAP-Sim is depicted in Figure 1b, and the modified and additional components to the SimpleScalar simulator are outlined with patterns. Harvesting and embedded systems exploit ultralow-power microcontrollers such as ARM Cortex-M, ARM7, or TI MSP430 with scratchpad memories to reduce energy consumption. Therefore, in line with recent studies targeting embedded and EHSs [7], [9], CATNAP-Sim is based on SimpleScalar, which models ARM7 cores (three-stage pipeline RISC cores without cache memory). CATNAP-Sim is able to work with thumb or regular instruction sets. We also have a plan

to extend CATNAP-Sim to support simulation of EHSs with ARM Cortex-A processors with zero, one, or two cache levels using gem5 simulator.

#### Source of ambient power

Several surrounding ambient energy sources can be utilized to provide electrical energy through various designated harvesting technologies like solar and RF [3]. Among the ambient power sources, harvesting solar energy through photovoltaic (PV) cell technology is desirable due to the high availability and remarkable energy density [6], [11]. Light energy patterns can be predictable or stochastic, responding to numerous environmental circumstances such as time, location, and mobility of the device. Figure 1c depicts the input irradiance for eight days [11], and it shows that the radiant power is significantly low throughout most of the day. Therefore, energy consumption in EHSs is a critical challenge that directly affects the forward progress of the system.

#### Processing unit

The advantages of NVPs with NVM-SRAM memory, e.g., fast backup, fast restore, and low leakage energy make them suitable for EHSs with unreliable energy sources. However, the state-of-the-art NVPs exploit nonvolatile registers, which consist of a nonvolatile storage cell attached to a CMOS flip-flop [4]. During the ordinary execution of the program, the processor works with the CMOS flip-flops; during backup, data in flip-flops are stored in nonvolatile cells; during boot-up, flip-flops restore their data from the nonvolatile cells. Although emerging NVMs considerably reduce the energy consumption of backups and restores compared to FLASH memories [4], backup and restore operations in NVPs are still expensive because processors need to back up all the registers of the system [2], [7]. In addition to registers, the main memory in an NVP consists of hybrid SRAM-NVM memory to improve the efficiency and performance of both execution and backups because access latency and energy consumption of NVMs (especially write accesses) are higher than SRAM.

To validate the energy consumption and execution time of the CATNAP-Sim, we executed 15 applications on MSP430FR5969 and CATNAP-Sim, and the results are presented in Table 1. Although ARM7 and MSP430 have different instruction-sets, both are

**Table 1. Comparison between the energy and execution time of 15 applications on msp430fr5969 and CATNAP-Sim.**

Application	Execution Time (ms)			Energy Consumption (nJ)		
	CATNAP	MSP430	Error(%)	CATNAP	MSP430	Error(%)
edge	285.52	306.91	7.49	224.83	241.92	7.60
smooth	4038.07	4301.33	6.52	2622.45	2740.46	4.50
corner	154.36	166.50	7.86	120.57	130.58	8.30
epic	2427.20	2256.71	7.02	1654.51	1743.86	5.40
dijkstra	812.36	868.33	6.89	641.85	684.86	6.70
ndes	13.53	12.44	8.06	11.41	12.47	9.30
ludcmp10	15.31	16.49	7.75	10.69	11.85	10.90
matmult10	742.17	793.46	6.91	473.14	506.26	7.00
insertsort	7302.82	7777.08	6.49	666.90	680.90	2.10
adpcm	1.68	1.54	8.21	1.47	1.67	13.30
qsort	25106.88	26680.20	6.27	2839.93	2871.17	1.10
bitcount	141.45	151.40	7.04	109.70	118.26	7.80
patricia	3333.25	3105.49	6.83	2430.92	2550.03	4.90
stringsearch	3470.91	3684.29	6.15	2855.04	2974.95	4.20
crc32	21.95	20.40	7.02	16.44	18.02	9.60

RISC-based microcontrollers optimized for low-power applications. Therefore, most of the common instructions in both microcontrollers have similar behavior. Table 1 shows that for small applications, such as *crc32*, *adpcm*, and *ludcmp10*, the energy consumption error is more than 9.6%, whereas, for large applications, such as *insertsort*, *stringsearch*, and *qsort*, the error is less than 4.2%. So, compared to a real platform, the energy consumption and execution time error of CATNAP-Sim is less than 13.30% and 8.21%, respectively. NVPSim did not provide any report regarding the energy consumption error, but the execution time error in this simulator is less than 9.31%.

### Power system

The power system accumulates the ambient energy in the system's energy storage and turns on/off the processor. PV cells are among the most popular and convenient technologies to scavenge ambient solar power and convert it into electrical energy. CATNAP-Sim exploits Wang's power system model [6] for absorbing solar energy and storing it in the system's energy storage. In this model, dc-dc converters, which are necessary for conventional EHSs, are replaced with a simple controller to improve energy efficiency. Furthermore, the use of a bulk capacitor as the system's energy storage provides higher energy efficiency, less complexity, and improves the area, cost, and weight of the system compared to super-capacitors and rechargeable batteries.

Wang's model is evaluated with a real hardware prototype. The evaluation shows that the leakage and other nonideal factors have a more significant effect on lower irradiances. So, the input power model's error decreases from 8.2% to 2.7% when irradiance increases from 100 to 440 W/m<sup>2</sup>.

NVPSim considers an ideal capacitor as the energy buffer of the system, which can store any amount of energy while real capacitors saturate after accumulating a specific amount of energy. Wang's power model (Figure 1d) shows that the incoming current from the PV cells drops when the capacitor's voltage goes above the saturation voltage. So, the absorbed power depends on the accumulated energy (voltage) of the capacitor. Therefore in NVPSim, the model for absorbing and accumulating energy is not realistic. NVPSim did not report any error, but in CATNAP-Sim, we used Wang's model [6] with less than 8.2% error.

There is a switch for turning the processor on/off by connecting and disconnecting the processor from

the capacitor. If the capacitor's voltage drops below the cutoff voltage, the controller opens the switch to recharge the capacitor. When the stored energy exceeds a predefined trigger threshold, the controller closes the switch to turn on the processor. The system exploits a small decoupling capacitor (in the scale of a few nano-farads) to tolerate noise and create backups from processor registers in the case of power failures. Figure 1a outlines the components of an EHS proposed by Wang et al. [6]. Figure 1d shows the power-voltage curve of the absorbed power, which peaks when the capacitor's voltage is in the range 2.5–3 V. In CATNAP-Sim, the user can set the cutoff and trigger voltage to scavenge the maximum power from the ambient source.

### Energy storage

The energy storage is responsible for accumulating absorbed energy and providing energy for the NVP. The size of the capacitor is an important parameter in EHSs because using small capacitors would increase the number of power interrupts, and using large capacitors results in longer charging time and response time in the system [7], [10]. In CATNAP-Sim, the user can configure the size of the capacitor to provide a fair tradeoff between charging time and the number of power interrupts. The available energy in the capacitor in cycle *i* is calculated as follows:

$$eng_{cap,i} = eng_{cap,i-1} + eng_{in,i} - eng_{nvp,i} - eng_{dpt} \quad (1)$$

**Table 2. Setup configurations used CATNAP-Sim.**

Simulator Configurations	MSP430	Cortex-M0
Processor Average Energy of Non-memory Inst	0.23 nJ	0.16 nJ
Processor Backup Energy	4.7 nJ	7.2 nJ
Processor Restore Energy	1.3 nJ	2.4 nJ
Processor Backup Latency	320 ns	400 ns
Processor Restore Latency	384 ns	400 ns
Processor Frequency	24 MHz	8 MHz
Processor Leakage Power	675 μW	0 μW
FRAM Read Access Energy		0.22 nJ
FRAM Write Access Energy		0.34 nJ
SRAM Access Energy		0.05 nJ
FRAM Access Latency		125 ns
SRAM Access Latency		1 Cycle
Platform Power Dissipation (excluding processor)	450 μW	
Bulk Capacitor Size	50-500 μF	
Cut-Off Voltage	2.50 v	
Low Threshold Voltage	2.60 v	
Trigger Voltage	2.80 v	
Solar Irradiance		see Fig. 2a

where  $eng_{cap,i}$ ,  $eng_{in,i}$ ,  $eng_{nvp,i}$  and  $eng_{dpt}$  represent the accumulated energy in the capacitor, absorbed energy, energy consumed by NVP, and the dissipated energy of all of the modules in the system except the processor, respectively.

Considering the frequency of NVP ( $freq$ ), the capacitance of the bulk capacitor ( $C$ ), and the power-voltage curve (Figure 1d) of the power system ( $curve_{p-v}$ ), the absorbed power in cycle  $i$  could be estimated using the irradiance strength in cycle  $i$  ( $irr_i$ ), and the voltage of the capacitor in cycle  $i-1$  ( $vol_{i-1}$ ) as follows:

$$eng_{in,i} = curve_{p-v}(irr_i, vol_{i-1}) \times \frac{1}{freq} \quad (2)$$

$$vol_{i-1} = \sqrt{\frac{2 \times eng_{cap,i-1}}{C}}$$

The energy consumption of NVP consists of leakage power ( $pow_{nvp,leak}$ ) and dynamic power, and both NVM and SRAM have a notable impact on the dynamic energy consumption of NVPs. Therefore, the energy consumption of NVP in cycle  $i$  is calculated as follows:

$$eng_{nvp,i} = \left( pow_{nvp,leak} \times \frac{1}{freq} \right) + eng_{ins,i} + \sum_{mem \in \{sram, nvm\}} \left\{ (rd_{mem,i} \times eng_{rd,mem}) + (wr_{mem,i} \times eng_{wr,mem}) \right\} \quad (3)$$

The  $eng_{ins,i}$ ,  $eng_{rd,mem}$ , and  $eng_{wr,mem}$  show the energy consumption of executing an instruction on the NVP, reading from memory  $mem$ , and writing to memory  $mem$ , respectively. The  $mem$  is either NVM or SRAM, and the  $wr_{mem,i}$  ( $rd_{mem,i}$ ) is one if the NVP executes store (load) instruction from  $mem$  at cycle  $i$ ; otherwise, it remains zero. It is important to mention that the NVP only consumes energy when the switch is closed, and the processor is on.

## Experiments

We explore the design space of EHSs using CATNAP-Sim.

### System setup

CATNAP-Sim takes the configuration of processor and memory as input. We measured average energy consumption and latency of backup, restore, non-memory instructions, access to FRAM, and access to SRAM from MSP430FR5969 microcontroller using EnergyTrace++. CATNAP-Sim simulates a three-stage pipeline processor, and it can work with both thumb and regular instruction sets. However, in the evaluations, we used 16-bit thumb instruction set because MSP430 is a three-stage pipeline 16-bit processor. Furthermore, we used energy and latency parameters of an NVP with the ARM Cortex-M0 proposed by Bartling et al. [12], and we perform experiments with both microcontrollers (Table 2). We add various input solar traces from Gorlatova et al. [11] to study the effects of input power's strength and stability on EHSs (Figure 2a). All experiments are done with these 30 s traces, and users can import custom solar traces to the CATNAP-Sim. The users can implement their approaches with various processors and platforms by changing these values in the simulator's configuration file.

CATNAP-Sim is based on SimpleScalar, and NVPSim is based on gem5 simulator. The reports show that both SimpleScalar and gem5 are capable of executing more than several hundred million instructions per hour. Table 3 shows the simulation time for simulating with CATNAP-Sim and NVPSim on a server with ten 3 GHz Intel Xeon E5-2690 cores. The results show that for a single execution, NVPSim is faster than CATNAP-sim by up to eight times. However, we provide a set of scripts in CATNAP-Sim to run simulations in parallel and improve the simulation time. So, for the execution of ten different scenarios, CATNAPSim improves the simulation time by up to 7.4 times over NVPSim.

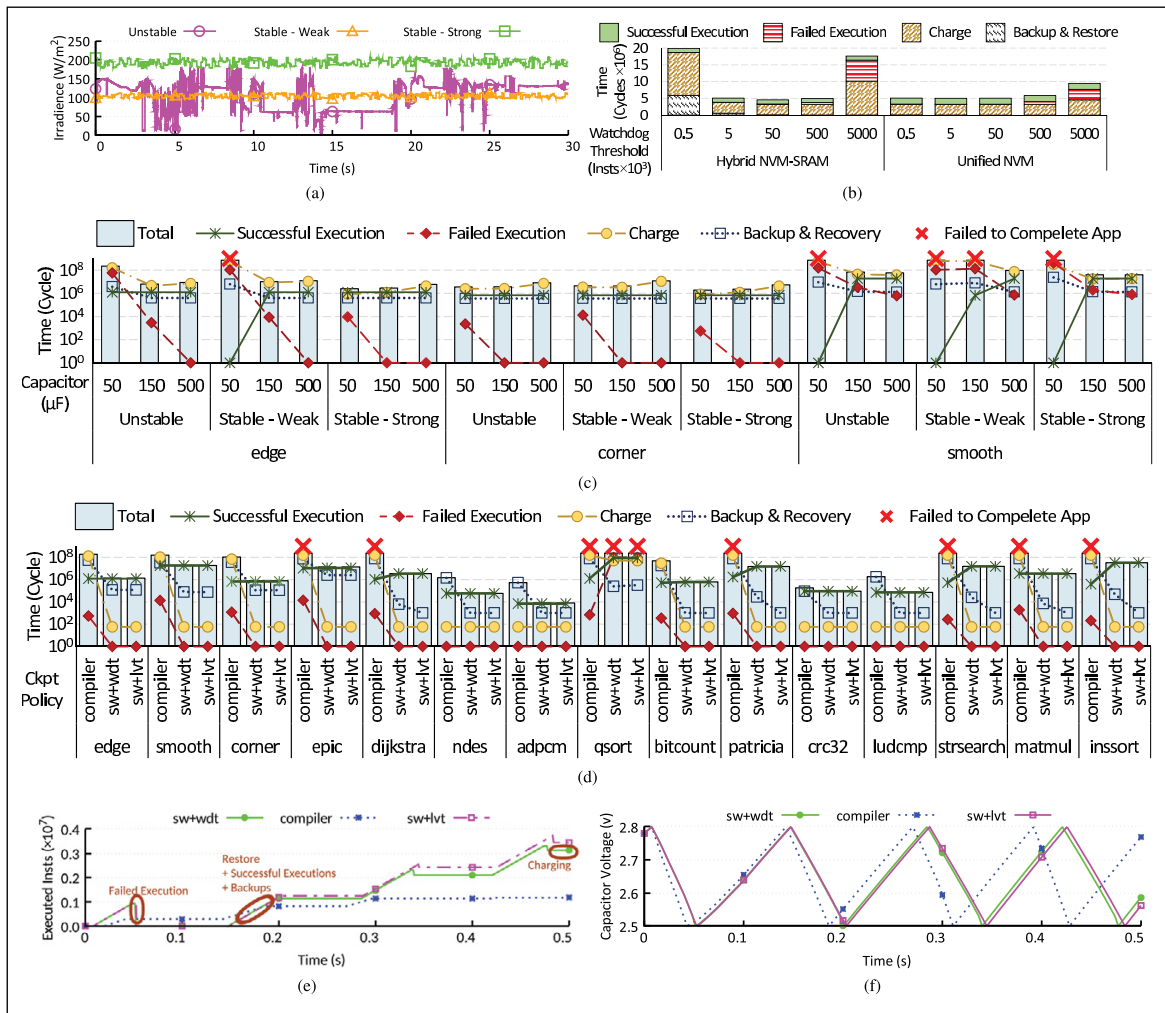
## Results

In the following, we discuss the effects of various parameters on the performance and energy efficiency of EHSs.

*Input power trace:* Input power changes both in amount and pattern over time. For example, solar energy usually has its peak, moderate, and minimum power at noon, morning (or afternoon), and after

**Table 3. Comparison between simulation time of CATNAP-Sim and NVPSim.**

Number of Simulated Cycles	Single Scenario		Ten Scenarios	
	NVPSim (s)	CATNAP (s)	NVPSim (s)	CATNAP (s)
$10^5$	7	56	72	63
$10^6$	47	105	437	119
$10^8$	194	213	2005	272



**Figure 2. Comprehensive design exploration using CATNAP-Sim. (a) Three weak, strong, and unstable solar traces used in simulations. (b) Effect of SRAM and check-pointing interval (epic application, 500  $\mu\text{F}$  capacitor, unstable trace, watchdog check-pointing policy, and MSP430 microcontroller). (c) Effects of input power trace, capacitor size, and benchmarks on EHSs (watchdog check-pointing policy with 50,000 instructions thresholds and MSP430 microcontroller). (d) Effects of check-pointing policy and benchmarks on EHSs (unstable trace, 500  $\mu\text{F}$  capacitor, and Cortex-M0 microcontroller). (e) Forward progress trace of EHS for the first 500 ms of execution (edge application, 500  $\mu\text{F}$  capacitor, unstable trace, and MSP430 microcontroller). (f) Voltage trace of EHS for the first 500 ms of execution (edge application, 500  $\mu\text{F}$  capacitor, unstable trace, and MSP430 microcontroller).**

sundown (or at night), respectively. Another factor to consider would be the mobility of the device, whether the device is in motion or in a stable position, which can profoundly affect the predictability of the input power.

*Benchmarks:* Energy consumption of accesses to memory (especially NVMs) has a significant contribution to the total energy consumption of

the system. The number and pattern of accesses to memory are different between benchmarks. Therefore, to yield more accurate simulation results, we added 15 applications from MiBench and Mälardalen to CATNAP-Sim. Figure 2c shows that for *corner*, the number of failed executed instructions becomes almost zero when the system exploits 150  $\mu\text{F}$  capacitor, while for *edge*, this happens when the system

uses a 500  $\mu\text{F}$  capacitor. The contribution of failed executed instructions in *smooth* application on response time is less than 2% for the system with a 500  $\mu\text{F}$  capacitor.

*Capacitor size:* The system can accumulate more energy with a larger capacitor, which reduces the number of power interrupts in the system, but the system requires more time to charge a larger capacitor [7], [10]. Thus, the large capacitor results in longer response time in the system, especially when the input power is weak. The results in Figure 2c show that the system with 50  $\mu\text{F}$  is unable to complete *smooth* in 30 s when it works under all solar traces depicted in Figure 2a. The system with 150  $\mu\text{F}$  also fails to complete *smooth* under weak solar irradiance. For *corner*, the system with 50 and 150  $\mu\text{F}$  capacitors improves the average response time of the system with 500  $\mu\text{F}$  capacitor by 61% and 58%, respectively.

*Check-pointing policy:* We compared three check-pointing policies using CATNAP-Sim for 15 applications, and the results are presented in Figure 2d. Furthermore, Figure 2e and f displays the first 500 ms of the execution of one sample only to clearly show the detailed forward progress and voltage trace, respectively (for the sake of visibility, we cannot show the detailed forward progress and capacitor voltage trace for 30 s). In *compiler* check-pointing policy [1], many unnecessary backups are imposed on the system, which increases the system's energy and performance overhead. Therefore, the forward progress is slow, and the capacitor energy is depleted faster than other approaches. The *watchdog* [7] and *low-voltage-threshold* [5] policies make backup decisions during the execution time; hence, these policies need hardware modifications or operating system supports to guarantee the consistency of the application; otherwise, they need to re-execute the application from scratch. To this end, we implemented two policies: 1) *software plus watchdog* (*sw+wdt*) and 2) *software plus low-voltage-threshold* (*sw+lv*). So, in the case of failure, the processor resumes application from the last software check-point to keep consistency without hardware modifications. Figure 2d shows that *sw+wdt* and *sw+lv* have faster forward progress than *compiler* check-pointing policy. Furthermore, the system failed to complete seven applications with *compiler* policy. However, in *sw+wdt* and

*sw+lv*, the system only failed to complete one application.

The important challenge in check-pointing is to achieve an optimal point in a tradeoff between the total number of check-points and the total number of failed instructions caused by power failures. To this end, we used the *watchdog* policy with different thresholds, and the results are presented in Figure 2b. The results for hybrid NVM-SRAM show that when the watchdog threshold is set to 500, 5000, 50,000, 500,000, and 5,000,000 instructions, the contribution of backup and restore (failed executed instructions) to total response time is 29.63% (0.02%), 11.71% (0.10%), 1.36% (0.61%), 0.18% (9.85%), and 0.03% (35.49%), respectively. It is noteworthy to mention that charge time. Results depict that the charging time of the system is 532, 134, and 422 ms, when the watchdog threshold is set to 500, 50,000, and 500,000 instructions, respectively.

*Memory type:* The use of SRAM improves the energy efficiency and performance of embedded systems [1], [2], [5]. However, EHSs should backup the SRAM data to NVM when the backup signal is raised. Therefore, when the backup signals the backup, restore, and failed executed instructions increase the energy inefficiency in the system and result in longer a system with unified NVM (e.g., QuickRecall [4]) and a system with hybrid NVM-SRAM memory with various backup intervals. The results show that when the system sets the threshold too low (500) or too high (5,000,000), the hybrid NVM-SRAM increases the system's response time by 74% and 46%, respectively. The best result of the hybrid NVM-SRAM approach is achieved when the watchdog timer threshold is set to 50,000 instructions, which improve the best result of the unified NVM approach (5000 instructions watchdog threshold) by 10%.

**IN THIS ARTICLE**, we proposed CATNAP-Sim, a nonvolatile processor for EHSs. We discussed the important components of state-of-the-art EHSs and the model of components which are utilized in CATNAP-Sim. CATNAP-Sim provides an opportunity for users to explore the design space exhaustively, find appropriate values for various parameters, and improve the performance and energy efficiency of EHSs. We also investigated the effects of various parameters, such as input power strength, benchmarks, capacitor size, check-pointing policy, and impact of SRAM in EHSs. ■

## References

- [1] M. Xie et al., "Avoiding data inconsistency in energy harvesting powered embedded systems," *ACM Trans. Design Autom. Electron. Syst.*, vol. 23, no. 3, p. 38, 2018.
- [2] K. Maeng, A. Colin, and B. Lucia, "Alpaca: Intermittent execution without checkpoints," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, p. 96, 2017.
- [3] B. Martinez et al., "The power of models: Modeling power consumption for IoT devices," *IEEE Sensors J.*, vol. 15, no. 10, pp. 5777–5789, Oct. 2015.
- [4] H. Jayakumar et al., "QuickRecall: A HW/SW approach for computing across power cycles in transiently powered computers," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 12, no. 1, p. 8, 2015.
- [5] H. Li et al., "An energy efficient backup scheme with low inrush current for nonvolatile SRAM in energy harvesting sensor nodes," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*. San Jose, CA, USA: EDA Consortium, 2015, pp. 7–12.
- [6] Y. Wang et al., "Storage-less and converter-less photovoltaic energy harvesting with maximum power point tracking for Internet of Things," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no.2, pp. 173–186, Feb. 2016.
- [7] A. Hoseinghorban, M. Abbasinia, and A. Ejlali, "PROWL: A cache replacement policy for consistency aware renewable powered devices," *IEEE Trans. Emerg. Topics Comput.*, early access, Oct. 14, 2020, doi: 10.1109/TETC.2020.3031114.
- [8] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, 2002.
- [9] M. Bazzaz, M. Salehi, and A. Ejlali, "An accurate instruction-level energy estimation model and tool for embedded systems," *IEEE Trans. Instrum. Meas.*, vol. 62, no. 7, pp. 1927–1934, Jul. 2013.
- [10] Y. Gu et al., "NVPsim: A simulator for architecture explorations of nonvolatile processors," in *Proc. 21st Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2016, pp. 147–152.
- [11] M. Gorlatova, A. Wallwater, and G. Zussman, "Networking low-power energy harvesting devices: Measurements and algorithms," *IEEE Trans. Mobile Comput.*, vol. 12, no. 9, pp. 1853–1865, Sep. 2013.
- [12] S. C. Bartling et al., "An 8 MHz 75  $\mu$ A/MHz zero-leakage non-volatile logic-based Cortex-M0 MCU SoC exhibiting 100% digital state retention at VDD=0V with <400ns wakeup and sleep transitions," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2013, pp. 432–433.

**Ali Hoseinghorban** is currently pursuing a PhD with the Computer Engineering Department, Sharif University of Technology, Tehran, Iran. He is also a Visiting Researcher with the Chair for Processor Design, CFAED, Technische Universität Dresden, Dresden, Germany. His research interests include energy harvesting systems and emerging nonvolatile memories.

**Mohammad Abbasinia** has a BSc in computer engineering from Shahid Beheshti University, Tehran, Iran (2015) and an MSc in computer engineering from the Sharif University of Technology, Tehran (2020). His research interests include energy harvesting systems and emerging nonvolatile memories.

**Ali Paridari** is currently pursuing an MSc with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. His research interests include energy harvesting systems and emerging nonvolatile memories. Paridari has a BSc from the Department of Computer Engineering, Shahid Beheshti University, Tehran, Iran (2018).

**Alireza Ejlali** is an Associate Professor of Computer Engineering with Sharif University of Technology, Tehran, Iran. His research interests include low power design and fault-tolerant embedded systems.

■ Direct questions and comments about this article to Alireza Ejlali, Department of Computer Engineering, Sharif University of Technology, Tehran, Iran; ejlali@sharif.edu.

# Design of $\pi$ -Shape Stub-Based Negative Group Delay Circuit

**Fayu Wan, Ningdong Li, and Blaise Ravelo**

Nanjing University of Information Science and Technology (NUIST)

**Sébastien Lalléchère**

Université Clermont Auvergne (UCA), CNRS

**Wenceslas Rahajandraibe**

Aix-Marseille University, CNRS,  
University of Toulon

*Editor's notes:*

This article presents a type of negative group delay (NGD) circuit based on transmission line resonators. In order to obtain the circuit's S-parameters, the article uses a combination of ABCD and Z-parameters. Analytical design equations are presented, which are verified using circuit simulations.

—Binoy Ravindran, Virginia Tech

■ **SINCE ITS FIRST** experimentation in the microwave frequencies [1], [2], the negative group delay (NGD) circuits have attracted much attention of electronic, RF, and microwave research engineers. Because of its counterintuitive effect, the NGD function remains unfamiliar and causes skepticism among RF and microwave engineering communities.

Further understanding about the NGD phenomenon intuitively occurred during the microwave metamaterial revolutions in the 2000s [3]–[5]. The NGD effect was found with double negative index periodical structures constituted by several periodical left-handed (LH) cells. However, the metamaterial structure-inspired NGD passive circuits suffer from high attenuation losses which may reach 20 dB [3]–[5] even with a single cell. Various designs, such as microstrip line-based [6] and dual band NGD circuit [7],

were performed for illustrating of this unfamiliar microwave circuit.

To solve the circuit application problems caused by the high insertion loss, some interesting NGD passive networks, for example, a general-purpose gain amplifier, were employed to compensate high signal attenuation (see

[8]–[11]). Active transversal filter-based NGD topologies using non-Foster elements were initiated [9], [10]. Encouraging results guaranteeing stability in the NGD bandwidth were obtained. A simpler low-noise amplifier (LNA) cascaded with  $RL$  and  $RC$ -network passive circuit is introduced in [11] and [12]. Nevertheless, the embedded amplifiers can increase the out-of-band noise and design complexity. Also, the NGD active circuits require further research work to reduce the design complexity considerably and to maintain the operation stability in the NGD bandwidth. Moreover, the active NGD circuits were still needed for further investigation notably about the nonlinearity and the definition of active NGD circuit Figure-of-Merit (FoM). In addition to design complexity, the applications of NGD-active microwave circuits are not yet potentially guaranteed in this decade. Therefore, the design of a low-loss microwave NGD-passive circuits is still a challenging and attractive task.

Digital Object Identifier 10.1109/MDAT.2020.3002149

Date of publication: 12 June 2020; date of current version: 8 April 2021.



To avoid the design complexity, more efforts have been put since the early 2010s on the design of low attenuation NGD-passive circuits [13]–[22]. Among the technical solutions for the passive NGD circuit design, innovative topologies of distributed elements such as microstrip coupling lines (CLs) and transmission lines (TLs) were developed in [13]–[18]. Two compact and self-matched NGD microstrip circuits [18] constituted by CL loaded lossy TLs and resistor were designed to obtain the reflection coefficient greater than 30 dB. Some NGD topologies employ complicated microwave functions as power dividers or power combiners to improve the circuits' bandwidth. However, the NGD applications remain practically limited by the design complexity.

In parallel to the research on low attenuation NGD-passive circuits, diverse NGD functions were developed [13]–[20]. NGD circuit inspired from absorptive band-stop filter topology was introduced in [16]. Further works on the design of compact NGD circuits were also made [18]–[21]. A multiband NGD circuit operating simply with tri-parallel uncoupled lines was designed in [20]. To meet the maturity of applications [22], [23], the NGD microwave circuit researchers must overcome the limitations [24] notably in terms of attenuation and still need to explore the different topologies of NGD microwave circuits.

For this reason, an unfamiliar microwave-passive topology of reverse T-shape stub is initiated in this article. It will be investigating whether the topology may be able to generate NGD function guaranteeing low signal attenuation. The NGD network is composed of fully distributed elements as two identical CLs and identical TLs.

This article is organized in three main sections as follows.

- To gain a good familiarity about the NGD analysis, theoretical investigation based on the  $S$ -parameter modeling is developed in the “NGD theory of the under design intercoupled branch reverse T-stub topology” section. The bandpass NGD analysis will be performed by identifying the NGD center frequency. The NGD existence condition will be established from the analytical group delay expression derived from transmission coefficient.
- In the “Design, simulation, and experimental validations” section, the calculated results, simulations, and experimental validations will be

discussed. To verify the feasibility of the NGD theory, a proof-of-concept of reverse T-stub microstrip circuit design will be described.

- Finally, in the last section, a conclusion is provided.

## NGD theory of the under design intercoupled branch reverse T-stub topology

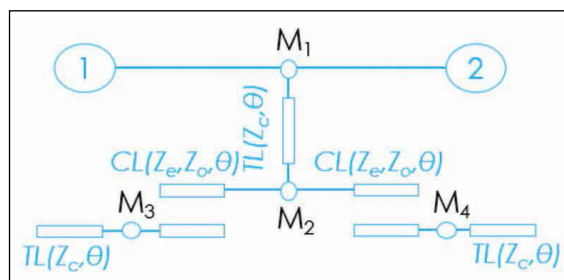
In this section, the modeling of the unfamiliar NGD topology constituted originally by reverse T-shape stub with branch line intercoupled will be introduced. Compared to the existing work, in particular, the NGD CL topology investigated in [19], a new distributed reverse T-shape stub topology is investigated in this article. Accordingly, completely new analytical expressions governing the NGD analysis are established.

### Analytical modeling methodology of reverse T-topology

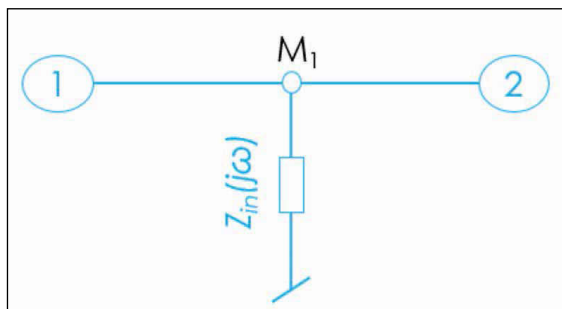
After topological description, the global circuit input impedance will be analytically investigated based on the approach reported in [25] and [26]. The equivalent  $S$ -parameter analytical model is established. Then, the analytical group delay expression is extracted from the reverse T-stub topology transmission coefficient. Then, the NGD analysis will be established.

### Topological description

Figure 1 sketches the electrical circuit configuration of the “ $\_|\_$ ” or reverse T-stub-based NGD topology under study. It acts as a two-port circuit built with fully distributed passive structure. It is built with three identical TLs. The secondary branches of “T” are constituted by two CLs and TLs. The constituting elementary CLs and TLs are assumed having the same characteristic impedance  $Z_c$  and electrical length  $\theta$ .



**Figure 1. Reverse T-stub-based topology under study.**



**Figure 2. Equivalent reduced circuit of the topology shown in Figure 1.**

Acting as a microwave topology, our analysis and design must be based on the  $S$ -parameters. As shown in Figure 2, the overall topology can be reduced to a parallel impedance  $Z_{in}$ . To determine the global  $S$ -parameter, we can proceed with the equivalent impedance matrix of  $Z_{in}$ .

**Methodological description**

Acting as a passive circuit, the NGD design method including the theoretical analysis can be summarized by the flow chart depicted in Figure 3. The key point of the NGD analysis of our reverse T-topology is the calculation of the  $S$ -parameters. The schematic of the topology shown in Figure 1 with equivalent matrix blocks is shown in Figure 4. According to the microwave circuit theory, the  $S$ -parameter can be obtained from the two-port circuit impedance matrix via the  $Z$ -to- $S$  transform.

**Elementary parameter definition**

The constituting elementary lines are assumed with physical length  $d$ , which corresponds to the delay  $\tau$  and the electrical length  $\theta$ . We denote the propagation wave speed by  $v$ , and the relationship between  $d$  and  $\tau = d/v$ . We recall that the electrical length is given by  $\theta(\omega) = \omega\tau$  the basic elementary CLs and TLs. For the sake of analytical simplification of the  $S$ -parameter and group delay expressions, the reverse T-stub constituting CLs and TLs are considered lossless.

**Analytical approach and modeling of reverse T-topology input impedance**

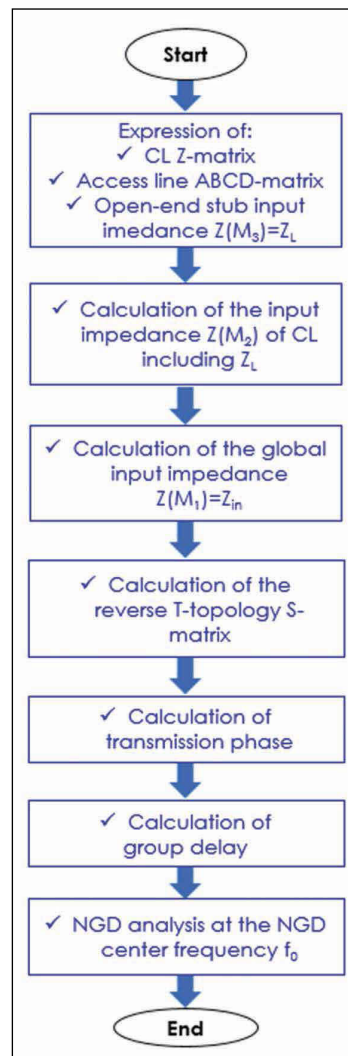
The calculation of  $Z_{in}$  will be realized from the operation between the ABCD- and Z-matrices of the constituting elements of the reverse T-stub circuit. By means of ABCD matrix, the NGD topology global input impedance can be explicated as

$$Z_{in}(j\omega) = \frac{jZ_c \left\{ \begin{aligned} &(x-1)^2 \cos^2[\theta(\omega)] - \\ &(x+1)^2 \sin^4[\theta(\omega)] - 16x^2 \cos^4[\theta(\omega)] \\ &+ 10x(x^2+1) \sin^2[\theta(\omega)] \cos^2[\theta(\omega)] \end{aligned} \right\}}{(x+1)^2 \cot[\theta(\omega)] \left\{ \begin{aligned} &(x+1)^2 \cos^4[\theta(\omega)] + \\ &2(x^2+1)[1-3\cos^2[\theta(\omega)]] \end{aligned} \right\}} \quad (1)$$

and the term  $x = \sqrt{(1+C)/(1-C)}$  where  $C$  is the voltage coupling coefficient of the coupler.

**S-parameter modeling**

The following paragraphs will explore the NGD analysis of our T-stub-based topology. Acting as a microstrip circuit, the analysis is established with  $S$ -parameter modeling.



**Figure 3. Flow chart of NGD modeling applied to the reverse T-topology.**

### Extraction method of global S-parameter

First, according to the circuit and system theory, we remember that the impedance matrix of the reduced circuit of Figure 2 is given by

$$[Z_{\perp}(j\omega)] = \begin{bmatrix} Z_{in}(j\omega) & Z_{in}(j\omega) \\ Z_{in}(j\omega) & Z_{in}(j\omega) \end{bmatrix}. \quad (2)$$

Knowing  $Z_{in}$ , the associated S-matrix can be obtained via the Z-to-S transform matrix relationship

$$[S_{\perp}(j\omega)] = ([Z_{\perp}(j\omega)] - R_0 [1]_2) \times ([Z_{\perp}(j\omega)] + R_0 [1]_2)^{-1} \quad (3)$$

with  $R_0 = 50 \Omega$  is the terminal load reference impedance and the two-dimension identity matrix

$$[1]_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \quad (4)$$

### Frequency-dependent expressions of reflection and transmission coefficients

Substituting input impedance  $Z_{in}$  introduced in (12) into the determined S-matrix, the reflection and transmission coefficients as a function of  $Z_c$ ,  $\theta(\omega)$ , and  $x$  were, respectively, written as

$$S_{11}(j\omega) = \frac{j(x+1)^2 R_0 \cos[\theta(\omega)] \left\{ \begin{array}{l} 2(x^2+1) \sin^4[\theta(\omega)] - x \sin[2\theta(\omega)] \\ -2(x^2-1) \cos^2[\theta(\omega)] \end{array} \right\}}{2\xi(j\omega)} \quad (5)$$

$$S_{21}(j\omega) = \frac{jZ_c \sin[\theta(\omega)] \left\{ \begin{array}{l} 2(x^2+1)^2 \sin^4[\theta(\omega)] - 2(x^2-1)^2 \cos^2[\theta(\omega)] \\ -5x(x^2+1) \sin^2[2\theta(\omega)] + 32x^2 \cos^2[\theta(\omega)] \end{array} \right\}}{2\xi(j\omega)} \quad (6)$$

with

$$\xi(j\omega) = \left\{ \begin{array}{l} R_0 \cos[\theta(\omega)] \left[ \begin{array}{l} \chi_2 \sin^4[\theta(\omega)] \\ +\chi_4 \cos^2[\theta(\omega)] \sin^2[\theta(\omega)] \\ +\chi_7 \cos^2[\theta(\omega)] \end{array} \right] \\ +jZ_c \sin[\theta(\omega)] \left[ \begin{array}{l} \chi_1 \sin^4[\theta(\omega)] + \\ \chi_3 \cos^2[\theta(\omega)] \sin^2[\theta(\omega)] \\ +\chi_5 \cos^4[\theta(\omega)] \\ +\chi_6 \cos^2[\theta(\omega)] \end{array} \right] \end{array} \right\} \quad (7)$$

and

$$\begin{cases} \chi_1 = (x+1)^2 \\ \chi_2 = -(x^4+2x^3+2x^2+2x+1) \\ \chi_3 = -10x(x^2+1) \\ \chi_4 = 2x(x+1)^2 \\ \chi_5 = 16x^2 \\ \chi_6 = -(x^2+1)^2 \\ \chi_7 = (x^2+1)^2. \end{cases} \quad (8)$$

### Magnitudes of reflection and transmission coefficients

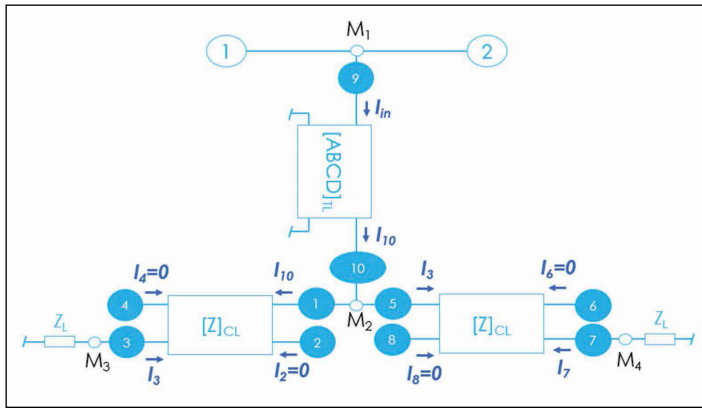
Similar to the classical microwave circuit analyses, before the NGD analysis, it is crucial to perceive the frequency responses of the transmission coefficient. Accordingly, the associated magnitude of the reflection and transmission coefficients is, respectively, given by

$$s_{11}(\omega) = |s_{21}(j\omega)| = \frac{\left| \begin{array}{l} (x+1)^2 R_0 \cos[\theta(\omega)] \\ \left\{ \begin{array}{l} 2(x^2+1) \sin^4[\theta(\omega)] \\ -x \sin[2\theta(\omega)] \\ -2(x^2-1) \cos^2[\theta(\omega)] \end{array} \right\} \end{array} \right|}{2\xi(\omega)} \quad (9)$$

$$s_{21}(\omega) = |s_{21}(j\omega)| = \frac{\left| \begin{array}{l} Z_c \sin[\theta(\omega)] \\ \left\{ \begin{array}{l} 2(x^2+1)^2 \sin^4[\theta(\omega)] - \\ 2(x^2-1)^2 \cos^2[\theta(\omega)] - \\ 5x(x^2+1) \sin^2[2\theta(\omega)] \\ +32x^2 \cos^2[\theta(\omega)] \end{array} \right\} \end{array} \right|}{2\xi(\omega)} \quad (10)$$

with

$$\xi(\omega) = \sqrt{R_0^2 \cos^2[\theta(\omega)] \left\{ \begin{array}{l} \chi_2 \sin^4[\theta(\omega)] \\ +\chi_4 \cos^2[\theta(\omega)] \sin^2[\theta(\omega)] \\ +\chi_7 \cos^2[\theta(\omega)] \end{array} \right\}^2 + Z_c^2 \sin^2[\theta(\omega)] \left\{ \begin{array}{l} \chi_1 \sin^4[\theta(\omega)] + \\ \chi_3 \cos^2[\theta(\omega)] \sin^2[\theta(\omega)] \\ +\chi_5 \cos^4[\theta(\omega)] \\ +\chi_6 \cos^2[\theta(\omega)] \end{array} \right\}^2}. \quad (11)$$



**Figure 4. Schematic of the topology shown in Figure 1 with equivalent matrix blocks.**

Description of NGD analysis and NGD circuit design

It is worth emphasizing that still many efforts are needed to make the NGD engineering familiar to electronic and RF/microwave device designers. Similar to classical electronic circuits (filters, phase shifters, couplers, power dividers, etc.), the NGD engineering can be openly performed in a familiar manner. The two following paragraphs describe the way to analysis and the methodological design of the NGD circuit under investigation.

### Frequency-dependent group delay

The phase shift associated to the transmission coefficient is defined by  $\varphi(\omega) = \angle S_{21}(j\omega)$ . We have

$$\varphi(\omega) = \frac{\pi}{2} - \arctan \left\{ \frac{Z_c \sin[\theta(\omega)] \begin{bmatrix} \chi_1 \sin^4[\theta(\omega)] + \\ \chi_3 \cos^2[\theta(\omega)] \sin^2[\theta(\omega)] \\ + \chi_5 \cos^4[\theta(\omega)] \\ + \chi_6 \cos^2[\theta(\omega)] \end{bmatrix}}{R_0 \cos[\theta(\omega)] \begin{bmatrix} \chi_2 \sin^4[\theta(\omega)] \\ + \chi_4 \cos^2[\theta(\omega)] \sin^2[\theta(\omega)] \\ + \chi_7 \cos^2[\theta(\omega)] \end{bmatrix}} \right\} \quad (12)$$

According to the circuit and system theory, the group delay can be derived from the transmission coefficient by the relation

$$\tau(\omega) = \frac{-\partial\varphi(\omega)}{\partial\omega} \quad (13)$$

Knowing the transmission phase introduced in (12), the reverse T-stub topology group delay can be calculated analytically from this previous expression.

### NGD analysis at very low frequency

One of natural particular frequency, which can be investigated for the NGD existence, is the lowest one or DC. The NGD analysis can be performed from the group delay expressed in (24). At very low frequency, the reverse T-topology presents the following group delay:

$$\tau(\omega \approx 0) = \frac{\pi Z_c [4x^2 - (x^2 - 1)^2]}{2R_0 \omega_0 (x^2 - 1)^2} \quad (14)$$

We can emphasize that this expression is always positive for any parameters  $x$  and  $Z_c$ . Therefore, the topology under study cannot behave as a low-pass NGD circuit.

### NGD analysis at resonance frequency

The second particular frequency of the topology under study can be the resonance  $\omega = \omega_0 = \pi/(2\tau)$ . At this frequency, the group delay is shown as

$$\tau(\omega_0) = \frac{-\pi R_0 (x + 1)^2}{2Z_c \omega_0 (x^2 + 1)} \quad (15)$$

This group delay is unconditionally negative for any value of the topology parameters. Therefore, the circuit can behave as bandpass NGD function. The NGD center frequency is equal to  $\omega = \omega_0$ .

### Design and test methodology of NGD circuit

The methodology to design and to test the NGD circuit must begin with the expected fabrication technology and the desired value of NGD at the center frequency. Then, we can follow the design guideline indicated by the flow chart of Figure 5 until the NGD circuit prototype fabrication and test.

The NGD design method can be divided into three successive phases.

- *Phase 1:* At the beginning of the design, the NGD function around the expected working frequency must be specified. Then, the analytical computation can be realized based on the ideal model of the  $S$ -parameters. Some parametric analyses can also be performed in this phase to check the better comprehension of influence of parameters constituting the NGD topology.
- *Phase 2:* In this intermediate step, the NGD engineer must take care on the available technology for fabricating the NGD prototype. For example, in the present study as it will be explored in the next validation section, we will deal with microstrip technology to design and implemented our NGD

prototype. Therefore, in this step, the NGD circuit can be designed with the design and simulation tools (as in the present study, we will utilize with the RF and microwave circuit ADS designer and simulator from Keysight Technologies). The simulations are focused on the realistic effects on the NGD prototype. It consists of simulating and optimizing the NGD circuit by taking into account the realistic effects as the TL widths and lengths. In this step, we can also add the extra interconnect lines as the input/output access lines, the “T” or “+” interconnects and also the substrate material parameters provided by the manufacturer.

- **Phase 3:** In this last phase, the NGD prototype must be fabricated from the circuit layout drawn from the optimized design performed in Phase 2. Acting as an RF and microwave circuit, the NGD prototype can be tested with a vector network analyzer (VNA). Before the tests, the VNA must be calibrated with consideration of the working frequency. The test must start with the *S*-parameter measurements. Then, the group delay can be calculated from (12).

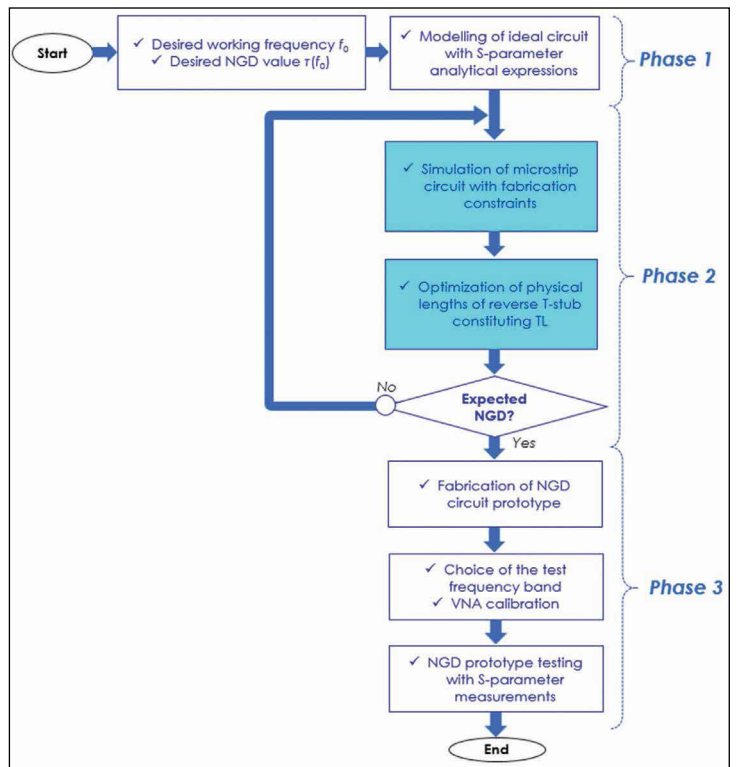
To verify more realistically the efficiency of the developed NGD theory, the proof-of-concept will be investigated in the next section.

### Design, simulation, and experimental validations

As proof-of-concept, NGD circuit was designed, simulated, fabricated, and tested to verify the relevance of the theory established in the previous section. The design process is implemented in a manner similar to that of the classical and familiar electronic analog circuits. All the simulation results presented in this article were obtained from simulations with the microwave electronic circuit designer and simulator ADS from Keysight Technologies. The measurements are performed with a VNA.

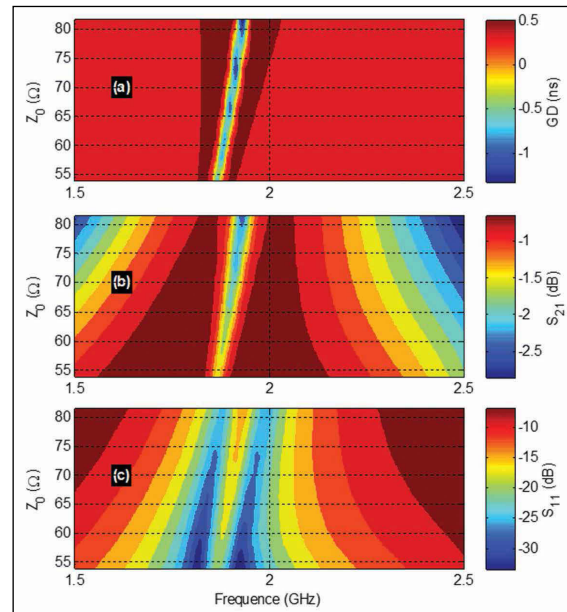
#### Parametric analysis

The proposed parametric analyses aim to predict the influences of the reverse T-stub parameters  $Z_0$ ,  $C$ , and  $\theta$  on the NGD topology under study. To do this, three cases of *S*-parameter parametric simulations from 1.5 to 2.5 GHz were performed by varying characteristic impedance  $Z_0$ , coupling coefficient  $C$  and the physical lengths  $d$ . The group delay, transmission, and reflection coefficient results are keenly



**Figure 5. Methodological flow chart of NGD circuit design and test.**

mapped in cartography in function of both the CL and TL characteristics and the operation frequency.



**Figure 6. Parametric simulated results versus  $Z_0$ : (a) group delay, (b)  $S_{21}$ , and (c)  $S_{11}$  with fixed  $\tau = 0.129$  ns.**

**Table 1. Time delay and the associated microstrip line physical length.**

$d$ (mm)	18	19	20	21	22
$\tau$ (ns)	0.123	0.130	0.136	0.143	0.150

It is noteworthy that the ideal circuit investigated in this section presents the same specifications as the FR4 substrate used to design and fabricate the proof of concept tested in the following section. Acting as an ideal configuration, the calculated results explored in the present subsection may differ from the electromagnetic computations and measurements because of the TL characteristic impedance, effective permittivity and also the metallization skin effect. Those effects are not taken into account in our NGD theory.

**Parametric analysis with respect to  $Z_0$**

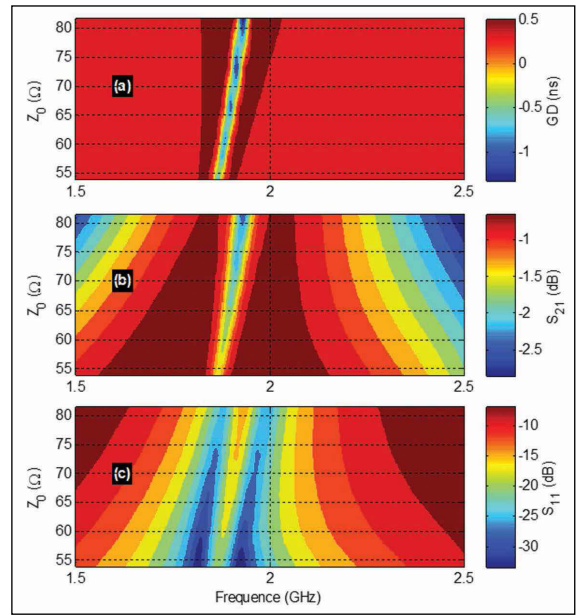
The present parametric analysis is dedicated to the influence of  $Z_0$  on the NGD aspect. As can be seen in Figure 6a–c, by varying  $Z_0$  from 81 to 54  $\Omega$ , the optimal GD decreases from  $-0.57$  to  $-1.32$  ns at the NGD center frequency. Then,  $S_{21}$  and  $S_{11}$  vary from  $-1.41$  to  $-2.44$  dB and from  $-18.75$  to  $-12.96$  dB, respectively.

**Parametric analysis with respect to  $C$**

The present paragraph analyses the influence of coupling coefficient  $C$  varied from varies from  $-14$  to  $-20$  dB on the NGD performance by fixing  $Z_0 = 64 \Omega$  and  $\tau = 0.129$  ns. It can be underlined that the GD and  $S_{21}$  are improved with the increase with  $Z_0$ . However,  $S_{11}$  is worst with the increase of characteristic impedance. The  $S$ -parameter and GD versus  $C$  are simulated from 1.5 to 2.5 GHz. It is found in Figure 7 that the NGD center frequency which is of about 1.886 GHz is insensitive to  $C$ . However, the GD and  $S_{21}$  decrease proportionally with  $C$ . Then, reflection coefficient  $S_{11}$  (in the vicinity of center frequency) becomes better when  $C$  decreases.

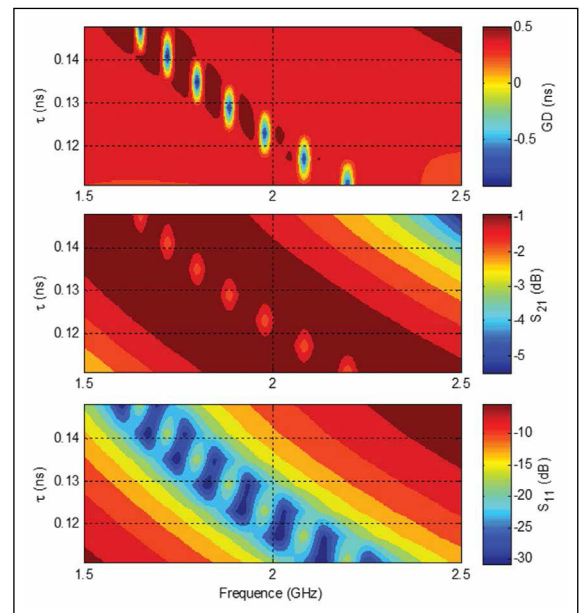
**Parametric analysis with respect to  $\tau$**

The NGD circuit performance depends on time delay  $\tau$ . Therefore, parametric analysis has been indirectly carried out with respect to  $\tau$  via microstrip lengths  $d$  summarized in Table 1. As seen in Figure 8a, when  $\tau$  changes from 0.123 to 0.150 ns, the GD varies from  $-0.90$  to  $-1.07$  ns. The NGD center frequency

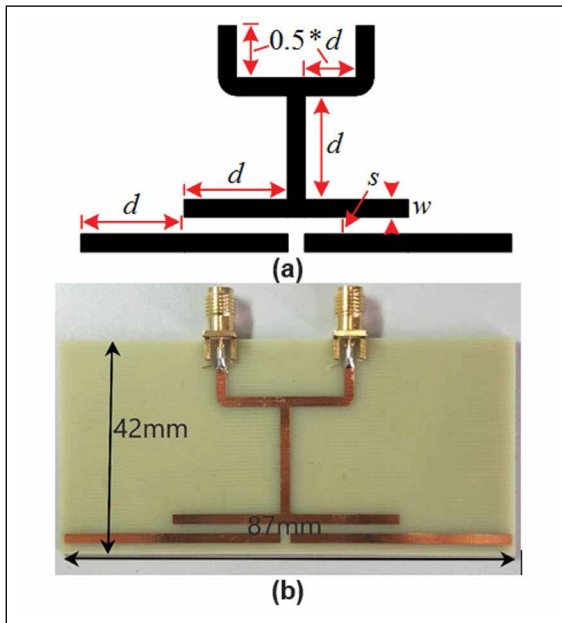


**Figure 7. Parametric analyses of (a) GD, (b)  $S_{21}$ , and (c)  $S_{11}$  in function of  $C$  with fixed  $Z_0 \approx 61.09 \Omega$  and  $\tau = 0.129$  ns.**

shifts significantly from approximately 1.98 and 1.65 GHz. Furthermore, as depicted in Figure 8b and c,  $S_{21}$  and  $S_{11}$  do not change significantly in the range of varied  $\tau$ . Therefore, to start the design, we can choose from the present parametric analysis, the T-stub appropriated physical length according to the designed NGD center frequency.



**Figure 8. Parametric analyses of (a) GD, (b)  $S_{21}$ , and (c)  $S_{11}$  in function of  $\tau$ .**



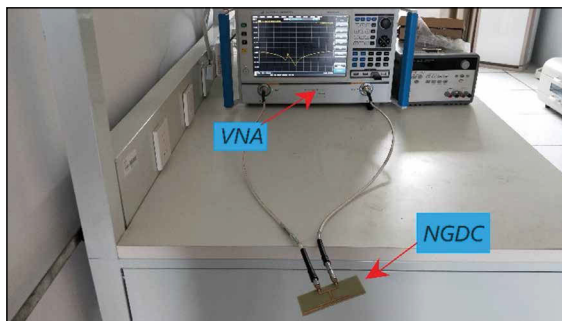
**Figure 9. Fabricated NGD circuit prototype (a) layout and (b) photograph.**

**Experimental results**

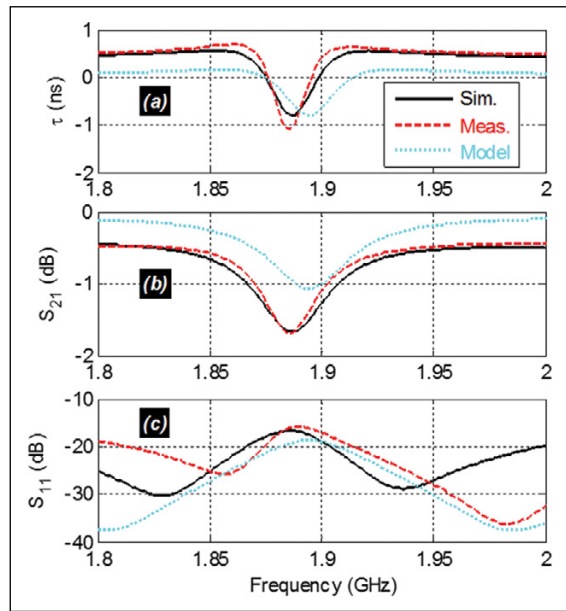
To validate experimentally the NGD function with the under-investigation reverse T-stub topology, the microstrip prototype design will be described in the next paragraph. Then, comparisons of results from simulations and measurements will be discussed.

**Design description of tested reverse T-stub-based NGD prototype**

To verify the rationality of the previously developed theory and parametric analyses, a proof-of-concept of NGD circuit was designed, fabricated, and measured. The prototype is a passive distributed circuit. This prototype is implemented in fully distributed microstrip



**Figure 10. Photograph of the reverse T-stub NGD prototype experimental setup.**



**Figure 11. (a) Group delay, (b) transmission, and (c) reflection coefficients of the fabricated NGD circuit shown in Figure 10b.**

technology without using lossy lumped circuits. The NGD circuit prototype was realized on the FR4 substrate presenting characteristics addressed in Table 2. Before the fabrication, the T-stub TLs and CLs were slightly optimized to reach more significant NGD values over low-attenuation losses. It should be emphasized that the TL connecting the access ports could

**Table 2. NGD circuit prototype parameters.**

Constituting element	Description	Parameter	Value
Metallization strip line	Conductivity	$\sigma$	58 MS/m
	Thickness	$t$	35 $\mu\text{m}$
	Width	$w$	1.9 mm
Dielectric substrate	Thickness	$h$	1.6 mm
	Relative permittivity	$\epsilon$	5
	Loss tangent	$\tan(\delta)$	0.02
CL	Width	$w$	1.9 mm
	Interspace	$s$	1.8 mm
	Length	$d$	21 mm
	Electrical length	$\theta$	90°
	Even-mode impedance	$Z_e$	68.25 $\Omega$
	Odd-mode impedance	$Z_o$	52.72 $\Omega$

**Table 3. Performance comparison (Y: yes, N: no).**

Ref.	$f$ (GHz)	$GD$ (ns)	$BW$ (MHz)	$S_{21}$ (dB)	$S_{11}$ (dB)	$FoM$	Extra lumped components?
[13]	2.14	-6.85	60	-15.4	-19	0.07	Y
[14]	2.14	-6.33	30	-20.7	-29	0.0175	Y
[18]	1.57	-8.75	60	-20.5	-32	0.0496	Y
[19]	1.016	-2.09	144	-18.1	-33	0.038	Y
[20]	2.3	-2.59	36	-4.1	-10	0.0581	N
[27]	1	-1.5	363	-33	-25	0.0133	Y
This work	1.89	-1	20	-1.7	-15	0.0164	N

be neglected because of the well-matching effect and its time-delay notable small compared to the targeted NGD value. The final parameters are indicated in Table 2.

The TL and CL physical width  $w$  corresponds to characteristic impedance  $Z_0 = 61.09 \Omega$ . The TL and CL quarter wavelength ( $\theta = 90^\circ$ ) is set at the NGD center frequencies of about 1.89 GHz. The considered CLs have the same coupling coefficients  $C$  of about  $-17.83$  dB.

The ADS design layout of the fabricated prototype is displayed in Figure 9a. The associated photograph is presented in Figure 9b which has a physical size  $42 \text{ mm} \times 87 \text{ mm}$ .

### Discussion on simulated and measured results

The NGD prototype was measured using a VNA provided by Rohde and Schwarz (ZNB 20, frequency band 100 kHz–20 GHz). The  $S$ -parameter measurement experimental setup is shown in Figure 10.

The comparative results between the calculations from (9), (10), and (13), simulations and measurements are performed from 1.8 to 2 GHz as depicted in Figure 11:

- simulations with ADS tool represented by the “Sim.” legend plotted in black solid line;
- experimental tests and measurements represented by the “Meas.” legend plotted in red dashed line;
- ideal calculated results represented by the “Model” legend plotted in blue dotted line.

The modeled results are slightly shifted because of the substrate imperfection in the considered working frequency. As plotted in Figure 11a, the NGD optimal value is of about  $-0.8$  ns in simulation against  $-1$  ns in measurement. The NGD center frequency is of about 1.89 GHz. The slight differences between the GD calculated from (12), simulations and experimental results, notably observed around the NGD

center frequency is notably due to the substrate dispersion loss and also the metallization skin effect. As expected, this result proves the validity of the bandpass NGD function generated by the reverse T-stub topology introduced earlier in Figure 1. The NGD prototype has a bandwidth of about 20 MHz. Moreover, Figure 11b introduces that the designed NGD prototype ensures a very low attenuation loss only of about  $-1.7$  dB in simulation and measurement around the center frequency. Additionally, as depicted in Figure 11c, the reflection coefficient is better than  $-15$  dB within the NGD bandwidth.

### Performance comparison

The comparison between the basic performances of the proposed NGD turtle topology and the existing ones available in the literature [13], [14], [18]–[20], [27] are summarized in Table 3. The comparison includes the NGD FoM defined in [19]. It is noteworthy that the introduced NGD topology presents the following advantages: 1) significant design flexibility; 2) implemented with fully distributed elements without lossy lumped component; 3) low signal attenuation less than 1.7 dB; and 4) the reflection loss better than  $-15$  dB without additional and external matching networks in the NGD bandwidth.

**AN NGD THEORY** of reverse T-stub shape inducing interbranch coupling effect is developed. The proposed NGD topology is composed of fully passive distributed elements with three identical TLs and two identical CLs. The  $S$ -parameter model of the topology is established from ABCD- and Z-matrices. The NGD analysis that allows identifying the NGD existence condition is described.

The relevance of the NGD theory was approved by simulations and measurements. Parametric analyses were conducted in function of the reverse T-stub physical and electrical characteristics. The NGD



performance of the topology in function of elementary CLs and TLs characteristic impedance and coupling coefficients and the operation frequency was cartographed.

More importantly, comparisons between the calculated, simulated, and measured results are also discussed. As proof-of-concept, the test and validations were performed with an NGD circuit prototype designed and implemented in microstrip technology. An excellent agreement between simulations and measurements was observed. The reverse T-stub NGD prototype achieved an excellent performance compared with the literature [13], [14], [18]–[20]. Measured group delay value of  $-1$  ns and transmission coefficient better than  $-2$  dB were occurred at the center frequency of about 1.886 GHz.

The modeling methodology offers valuable information to the electronic designer, for instance exploring the sensitivity of the T-stub NGD system to input variability. ■

## Acknowledgments

This work was supported in part by NSFC under Grant 61971230 and Grant 61601233, in part by the Jiangsu Distinguished Professor program and Six Major Talents Summit of Jiangsu Province (2019-DZXX-022), in part by the Postgraduate Research and Practice Innovation Program of Jiangsu Province under Grant SJKY19\_0974, and in part by the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD) fund.

## References

- [1] S. Lucyszyn and I. D. Robertson, "Analog reflection topology building blocks for adaptive microwave signal processing applications," *IEEE Trans. Microw. Theory Techn.*, vol. 43, no. 3, pp. 601–611, Mar. 1995.
- [2] C. D. Broomfield and J. K. A. Everard, "Broadband negative group delay networks for compensation of oscillators, filters and communication systems," *Electron. Lett.*, vol. 36, no. 23, pp. 1931–1933, Nov. 2000.
- [3] G. V. Eleftheriades, O. Siddiqui, and A. K. Iyer, "Transmission line models for negative refractive index media and associated implementations without excess resonators," *IEEE Microw. Wireless Compon. Lett.*, vol. 13, no. 2, pp. 51–53, Feb. 2003.
- [4] O. F. Siddiqui, M. Mojahedi, and G. V. Eleftheriades, "Periodically loaded transmission line with effective negative refractive index and negative group velocity," *IEEE Trans. Antennas Propag.*, vol. 51, no. 10, pp. 2619–2625, Oct. 2003.
- [5] O. F. Siddiqui et al., "Time-domain measurement of negative-index transmission-line metamaterials," *IEEE Trans. Microw. Theory Techn.*, vol. 52, no. 5, pp. 1449–1453, May 2004.
- [6] G. Chaudhary, Y. Jeong, and J. Lim, "Microstrip line negative group delay filters for microwave circuits," *IEEE Trans. Microw. Theory Techn.*, vol. 62, no. 2, pp. 234–243, Feb. 2014.
- [7] H. Choi et al., "A novel design for a dual-band negative group delay circuit," *IEEE Microw. Wireless Compon. Lett.*, vol. 21, no. 1, pp. 19–21, Jan. 2011.
- [8] C.-T.-M. Wu et al., "A dual-purpose reconfigurable negative group delay circuit based on distributed amplifiers," *IEEE Microw. Wireless Compon. Lett.*, vol. 23, no. 11, pp. 593–595, Nov. 2013.
- [9] T. Zhang, R. Xu, and C.-T.-M. Wu, "Unconditionally stable non-foster element using active transversal-filter-based negative group delay circuit," *IEEE Microw. Wireless Compon. Lett.*, vol. 27, no. 10, pp. 921–923, Oct. 2017.
- [10] M. Zhu and C.-T. Michael Wu, "A tunable non-foster T-network loaded transmission line using distributed amplifier-based reconfigurable negative group delay circuit," in *Proc. Asia-Pacific Microw. Conf. (APMC)*, Kyoto, Japan, Nov. 2018, pp. 720–722.
- [11] F. Wan et al., "The design method of the active negative group delay circuits based on a microwave amplifier and an RL-series network," *IEEE Access*, vol. 6, pp. 33849–33858, Jun. 2018.
- [12] F. Wan et al., "Time-domain experimentation of NGD ActiveRC-network cell," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 4, pp. 562–566, Apr. 2019.
- [13] G. Chaudhary and Y. Jeong, "Low signal-attenuation negative group-delay network topologies using coupled lines," *IEEE Trans. Microw. Theory Techn.*, vol. 62, no. 10, pp. 2316–2324, Oct. 2014.
- [14] G. Chaudhary and Y. Jeong, "Transmission-type negative group delay networks using coupled line doublet structure," *IET Microw., Antennas Propag.*, vol. 9, no. 8, pp. 748–754, Feb. 2015.
- [15] B. Ravelo, "Theory of coupled line coupler-based negative group delay microwave circuit," *IEEE Trans. Microw. Theory Techn.*, vol. 64, no. 11, pp. 3604–3611, Nov. 2016.
- [16] L.-F. Qiu et al., "Absorptive bandstop filter with prescribed negative group delay and bandwidth," *IEEE Microw. Wireless Compon. Lett.*, vol. 27, no. 7, pp. 639–641, Jul. 2017.

- [17] G. Liu and J. Xu, "Compact transmission-type negative group delay circuit with low attenuation," *Electron. Lett.*, vol. 53, no. 7, pp. 476–478, Mar. 2017.
- [18] T. Shao et al., "A compact transmission-line self-matched negative group delay microwave circuit," *IEEE Access*, vol. 5, pp. 22836–22843, Oct. 2017.
- [19] Z. Wang et al., "A negative group delay microwave circuit based on signal interference techniques," *IEEE Microw. Wireless Compon. Lett.*, vol. 28, no. 4, pp. 290–292, Apr. 2018.
- [20] F. Wan et al., "S-parameter model of three parallel interconnect lines generating negative group-delay effect," *IEEE Access*, vol. 6, pp. 57152–57159, Oct. 2018.
- [21] T. Shao et al., "A compact dual-band negative group delay microwave circuit," *Radioengineering*, vol. 27, no. 4, pp. 1070–1076, Dec. 2018.
- [22] B. Ravelo, A. Pérennec, and M. Le Roy, "Synthesis of frequency-independent phase shifters using negative group delay active circuit," *Int. J. RF Microw. Comput.-Aided Eng.*, vol. 21, no. 1, pp. 17–24, Jan. 2011.
- [23] L. He et al., "A 24-GHz source-degenerated tunable delay shifter with negative group delay compensation," *IEEE Microw. Wireless Compon. Lett.*, vol. 28, no. 8, pp. 687–689, Aug. 2018.
- [24] M. Kandic and G. E. Bridges, "Asymptotic limits of negative group delay in active resonator-based distributed circuits," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 8, pp. 1727–1735, Aug. 2011.
- [25] B. Ravelo, A. Normand, and F. Vurpillot, "Modelling of interbranch coupled 1:2 tree microstrip interconnect," *ACES J.*, vol. 33, no. 3, Mar. 2018, pp. 285–292.
- [26] B. Ravelo, "Theory on asymmetrical coupled-parallel-line transmission and reflection zeros," *Int. J. Circuit Theory Appl.*, vol. 45, no. 11, pp. 1534–1551, Nov. 2017.
- [27] C.-T.-M. Wu and T. Itoh, "Maximally flat negative group-delay circuit: A microwave transversal filter approach," *IEEE Trans. Microw. Theory Techn.*, vol. 62, no. 6, pp. 1330–1342, Jun. 2014.

**Fayu Wan** is currently a Full Professor with the Nanjing University of Information Science and Technology, Nanjing, China. From 2011 to 2013, he was a Postdoctoral Fellow with the Electromagnetic Compatibility Laboratory, Missouri University of Science and Technology, Rolla, MO. His current research interests include negative group delay circuits, electrostatic discharge, electro-magnetic

compatibility, and advanced RF measurement. Wan has a PhD in electronic engineering from the University of Rouen, Rouen, France (2011).

**Ningdong Li** is currently pursuing an MS from the Nanjing University of Information Science and Technology, Nanjing, China. His research interests include abnormal wave propagation in dispersive media and microwave circuits. Li has a BSc in electrical engineering from Anhui Polytechnic University, Wuhu, China (2017).

**Blaise Ravelo** is currently a Professor with the Nanjing University of Information Science and Technology, Nanjing, China. He is a pioneer of the negative group delay (NGD) concept and its applications. His Google Scholar in 2020 was with an h-index of 20. He is a (co-)author of more than 250 scientific research papers published in international conferences and journals.

**Wenceslas Rahajandraibe** is currently a Professor with the University of Aix-Marseille, Marseille, France. He heads the Integrated Circuit Design Group of the IM2NP Laboratory. His research is focused on the design of analog and RF IC's for telecommunication systems and for smart sensor ultralow power IC interfaces.

**Sébastien Lalléchère** is currently an Associate Professor with Institut Pascal and Université Clermont Auvergne, Clermont-Ferrand, France. His research interests cover the fields of electromagnetic compatibility including antennas and propagation, complex and reverberating electromagnetic environments, electromagnetic coupling, computational electromagnetics, stochastic modeling, and sensitivity analysis in electrical engineering. Lalléchère has a PhD in electronics/electromagnetism from Université Blaise Pascal, Clermont-Ferrand (2006).

■ Direct questions and comments about this article to Blaise Ravelo, Electronics and Information Engineering College, Nanjing University of Information Science and Technology (NUIST), Nanjing 210044, China; blaise.ravelo@nist.edu.cn.

# Design of Single-Bit Fault-Tolerant Reversible Circuits

**Hari M. Gaur**

ABES-IT Ghaziabad

**Ashutosh K. Singh**

NIT Kurukshetra

**Anand Mohan**

IIT (BHU) Varanasi

**Masahiro Fujita**

University of Tokyo

**Dhiraj K. Pradhan**

University of Bristol

*Editor's notes:*

This article introduces redundant design approaches for reversible circuits that have the ability to detect and tolerate single-bit fault without the need of conventional voting scheme. Experiments performed show that the proposed scheme reduces the gate cost on average with up to 28% as compared with tri-modular redundant circuits.

—Said Hamdioui, Delft University of Technology

■ **FAULT TOLERANCE IS** the architectural attribute of a digital system that maintains proper functioning of a machine while encountering various kinds of failures. It facilitates the realization of explicit parts of a system that involve a higher degree of safety and critical problems [1]. The use of redundant circuits in collaboration with majority voter scheme is one of the effective methods to achieve fault tolerance in digital system design. The method provides fault-free output at the cost of large number of gates and wires. In spite of having several advantages of this technique, some problems still exist. The worst situation can arise in majority voting when all the redundant circuits produce faulty output.

Motivated by a variety of applications in several emerging technologies toward reduction of power consumption and sizes, reversible circuits (RCs)

Digital Object Identifier 10.1109/MDAT.2020.3006808

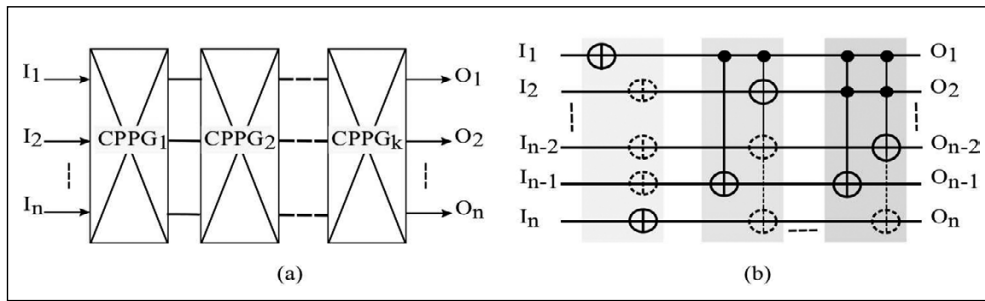
Date of publication: 3 July 2020; date of current version: 8 April 2021.

have received significant attention since a decade. RCs have direct relation with quantum computation which are largely to loss of energy levels due to the phenomenon of quantum decoherence that cause single point

failures [2]. In addition to the remarkable work in the field of testing, development of fault-tolerant designs are also finding grounds for these circuits [3]. The evolution commences from error correcting codes generation, majority multiplexing, error correction for finite field using parity check, majority voter scheme to recent Clifford+T quantum gate library [4]. Fault-tolerant redundant logic circuits in-hold lesser design complexity and has the ability of online repair and diagnose, however, it is not addressed in the literature for RCs.

Fault detection in parity preserving circuits

Parity preserving (PP) circuits have the capability to detect the faults occurred due to unusual change of bits in RCs. A number of testable design methodologies were presented in the past utilizing this characteristics. Unfortunately, if PP gates or complex PP gate (CPPG) are used to design a circuit, the statement will not be true. CPPG, shown in Figure 1a, is built with a group of fundamental gates and considered as a single



**Figure 1. PP architectures: (a) CPPG circuit and (b) fundamental gates-based circuit.**

gate (building block) in the circuit formation [5]. Parity preserved architectures ensures the detection of single-bit flip faults for such design methodology which produces PP circuits by the use of fundamental gates as presented in Figure 1b for example [6]–[8]. The scheme is based on the method of multiple controlled Toffoli (MCT) gates placement either during design methodology or modification technique and provides full coverage of single-bit faults [9], [10]. PP circuits, however, provide only fault detection. However, they can also be used to design fault-tolerant circuit by further modifications.

**Contribution**

This article introduces a generalized architecture for designing fault-tolerant RCs that can be scalable up to  $N$  modular redundant circuits using parity preservation and generation technique. The designed circuit using this model can detect and tolerate single point failures by means of bit fault. The presented model also have ability to deal with the occurrence of worst situation with redundant logic when all adjacent circuits produce faulty output and have fault diagnosis and repair provisions. Nevertheless, efficient methodologies for designing testable RCs have been adopted which were found superior in terms of gates, quantum cost (QC), garbage output (GO), ancilla input (AI), and fault coverage [9], [10].

**Proposed fault-tolerant model**

The insight of the scheme is the development of PP circuits followed by the inclusion of a parity checker to form testable circuits (TCs). These circuits generate an error signal at the output during any single-bit faulty operations. This signal is utilized to design redundant circuits for fault tolerance.

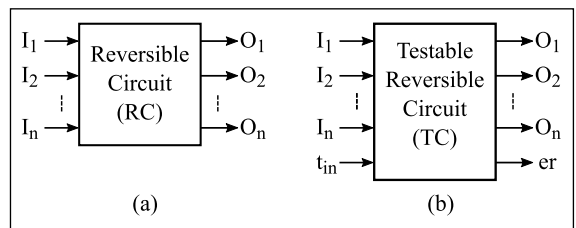
**Error signal generation**

In an  $n$  wire PP RC shown in Figure 2a with inputs  $(I_1, I_2, \dots, I_n)$  and outputs  $(O_1, O_2, \dots, O_n)$ , parity checking can be achieved by cascading controlled NOT (CNOT) gates from each wire to a new wire before and after the complete circuit. Considering  $t_{in}$  as a new test wire on which the parity checking has to be done for the formation of a TC, as shown in Figure 2b. The corresponding output of this wire is called as error signal ( $er$ ) in this article and is given by

$$er = (I \oplus O) \oplus t_{in} \tag{1}$$

where  $I = (I_1 \oplus I_2 \oplus \dots \oplus I_n)$  and  $O = (O_1 \oplus O_2 \oplus \dots \oplus O_n)$ . As RC is PP circuit,  $I \oplus O = 0$ . This means that, for nonerroneous functionality of RC,  $er = t_{in}$ .

The MCT gates placement methodology is utilized for the realization of RC and are converted into respected testable cells (TCs) for the generation of error signal ( $er$ ) [9], [10]. Considering a single-bit flip fault occurred at any level of the circuit. Each set (containing two MCT gates) in TC design scheme is PP and the same parity information will be transferred to next level. Hence, the values at the output will be inverted in odd numbers. Considering  $O_1 \rightarrow \overline{O_1}$ ,  $er$  will be inverted for faulty operations as calculated in



**Figure 2. Conversion of RC into TC: (a) CPPG circuit and (b) fundamental gates-based circuit.**

$$er = [(I_1 \oplus I_2 \oplus \dots \oplus I_n) \oplus (\overline{O_1} \oplus O_2 \oplus \dots \oplus O_n)] \oplus t_{in} = [O_1 \oplus \overline{O_1}] \oplus t_{in} = 1 \oplus t_{in} = \overline{t_{in}}. \quad (2)$$

Hence, the logic behind the generation of error signal can be summarized as  $er = t_{in}/\overline{t_{in}}$ , for nonfaulty/faulty operations of TC which is used to detect the occurrence of these faults in the circuit.

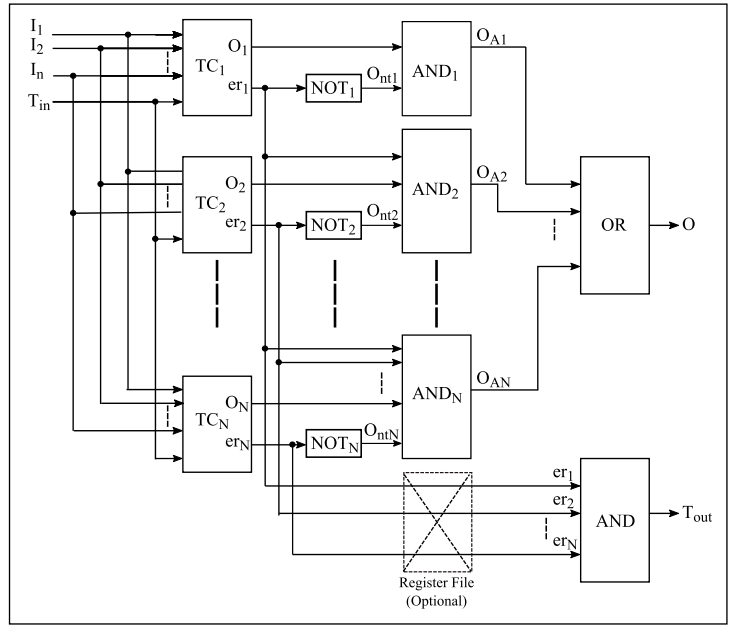
#### Fault tolerance

Depending on the required degree of accuracy of a circuit, multiple redundant TCs can be connected in parallel. The array of these TCs are cascaded with an AND-OR network to form a fault-tolerant circuit. Considering the following characteristics in a multiple-input-single-output (MISO) TC for the designing fault-tolerant model:

- $n$  input ( $I_1, I_2, \dots, I_n$ ) including AIs and one test input  $T_{in}$ ;
- one required output  $O$  and rest  $(n - 1)$  output are garbage;
- assigning  $T_{in} = 0$  which implies  $er = 0$  for non-faulty and  $er = 1$  for faulty operations.

Considering  $N$  number of TCs which are connected in parallel to produce  $N$  redundant outputs ( $O_1, O_2, \dots, O_N$ ). The corresponding error signal produced by TCs are ( $er_1, er_2, \dots, er_N$ ). GOs and constant input of all the building blocks are ignored. Each TC is cascaded with respective reversible AND gates ( $AND_1, AND_2, \dots, AND_N$ ). The outputs of these AND gates are used as input to a reversible OR gate which produces fault free output  $O$ , as depicted in Figure 3. The inputs to the AND gates are the output of respective TCs, complement of respective error signal ( $O_{nt1}, O_{nt2}, \dots, O_{ntN}$ ), error signals of  $N$ th TC and of all previous TCs. For instance,  $AND_1$  has three inputs ( $C, O_1, O_{nt1}$ ),  $AND_2$  has four inputs ( $C, O_2, er_1, O_{nt2}$ ) and  $AND_N$  has  $(N+2)$  inputs ( $C, O_N, er_1, er_2, \dots, er_{N-1}, O_{ntN}$ ). Here,  $C$  is a constant input for producing logical AND operation using MCT gates. Hence,  $AND_1$  is of size  $(3 \times 3)$ ,  $AND_2$  is of size  $(4 \times 4)$  and  $AND_N$  is of size  $\{(N+2) \times (N+2)\}$ . All the error signal output are also taken as the inputs to another AND gate. The output of this gate is given by  $T_{out} = er_1 \cdot er_2 \cdot \dots \cdot er_N$  after ignoring AI and GO.

The important aspect behind the designing is to probe the final output of fault-tolerant MISO TC circuit to be nonfaulty for the true functioning of at least any one of the TCs. The final output  $O$  of proposed MISO TC circuit is given by the following equation:



**Figure 3. Fault-tolerant MISO TC model.**

$$\begin{aligned} O &= O_{A1} + O_{A2} + \dots + O_{AN} \\ &= O_1 \cdot O_{nt1} + O_2 \cdot er_1 \cdot O_{nt2} + \dots + O_N \\ &\quad \cdot er_1 \cdot er_2 \cdot \dots \cdot er_{N-1} \cdot O_{ntN} \\ &= O_1 \cdot \overline{er_1} + O_2 \cdot er_1 \cdot \overline{er_2} + \dots + O_N \\ &\quad \cdot er_1 \cdot er_2 \cdot \dots \cdot er_{N-1} \cdot \overline{er_N}. \end{aligned} \quad (3)$$

The error signal outputs will be  $er_1 = 0$  and  $er_2 = er_3 = \dots = er_N = 1$  for  $T_{in} = 0$ , when all the TCs produce faulty output except  $TC_1$ . The final output can be calculated as  $O = O_1$  using (3), which is fault free output of  $TC_1$ . Based on the functioning of MISO TC and utilizing (3), the functional and fault patterns for Trimodular redundant fault-tolerant circuit containing three inputs ( $I_1, I_2, I_3$ ) and one output TC ( $O$ ) can be shown in Table 1. Here, ( $er_1, er_2, er_3$ ) are the three error signal outputs, which indicates the faulty/nonfaulty behavior of respective TCs for  $T_{in} = 0$ .

Hence, the error signal output will be at logic 0 when any one of the TCs is performing correct operation. The test output  $T_{out} = er_1 \cdot er_2 \cdot \dots \cdot er_N = 0$ . For the worst case when all the TCs are performing erroneous operations,  $er_1 = er_2 = \dots = er_N = 1$  which flips  $T_{out}$  to 1. Hence  $T_{out}$  will be 1 only when all TCs are in erroneous operations, else be 0. The provision for the inclusion of a register file can also be included on the wires of all the error signals to store the fault patterns of all the TCs. These patterns can be utilized for fault diagnosis and repair purposes.

**Table 1. Functional/fault pattern table of fault-tolerant MISO TC.**

Inputs				Error signal			NOT Gates Output			AND Gates Output			Final Output	Test Output	Fault Location
$I_3$	$I_2$	$I_1$	$T_{in}$	$er_3$	$er_2$	$er_1$	$O_{n13}$	$O_{n12}$	$O_{n11}$	$O_{A3}$	$O_{A2}$	$O_{A1}$	$O$	$T_{out}$	
$I_3$	$I_2$	$I_1$	0	0	0	0	1	1	1	0	0	0	$f_{n,fl}$	0	all circuit un-faulty
$I_3$	$I_2$	$I_1$	0	0	0	1	1	1	0	0	0	0	$f_{n,fl}$	0	$TC_1$ faulty
$I_3$	$I_2$	$I_1$	0	0	1	0	1	0	1	0	0	0	$f_{n,fl}$	0	$TC_2$ faulty
$I_3$	$I_2$	$I_1$	0	0	1	1	1	0	0	0	0	0	$f_{n,fl}$	0	$TC_1$ and $TC_2$ faulty
$I_3$	$I_2$	$I_1$	0	1	0	0	0	1	1	0	0	0	$f_{n,fl}$	0	$TC_3$ faulty
$I_3$	$I_2$	$I_1$	0	1	0	1	0	1	0	0	0	0	$f_{n,fl}$	0	$TC_1$ and $TC_3$ faulty
$I_3$	$I_2$	$I_1$	0	1	1	0	0	0	1	0	0	0	$f_{n,fl}$	0	$TC_2$ and $TC_3$ faulty
$I_3$	$I_2$	$I_1$	0	1	1	1	1	0	0	0	0	0	0	1	all circuit faulty

$f_{n,fl}$  = non-faulty output function

**Fault-tolerant model for MIMO TC**

The work is extended for the circuits which produces more than one output at a time. The number of outputs in a single TC will increases the design complexities as the number of NOT and will be equal to the multiple of number of redundant circuit and number of output of the TC. It will increase the gate cost as well as the QC which are the key metrics of any RC designs. Considering the following characteristics for the designing of multiple input multiple output (MIMO) TC:

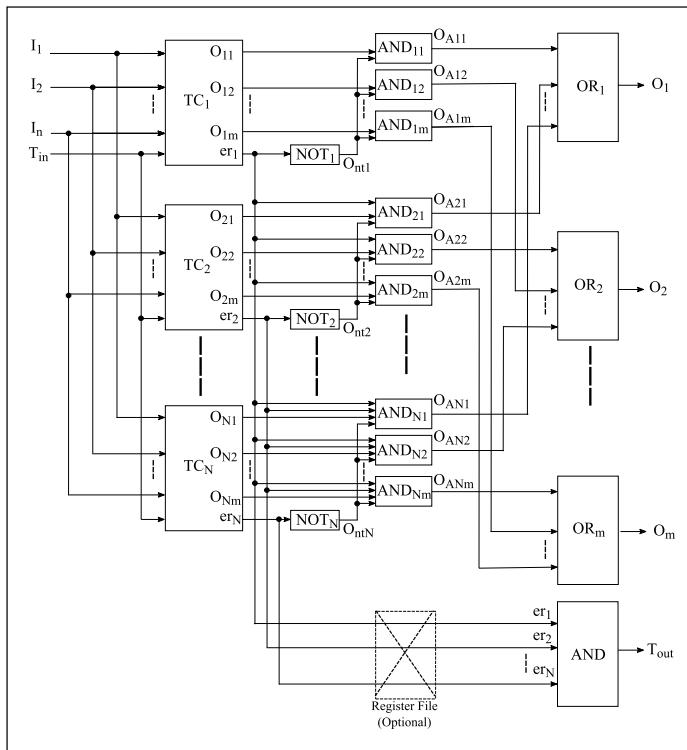
- $n$  input ( $I_1, I_2, \dots, I_n$ ) including AIs and one test input  $T_{in}$ ;
- $m$  required output ( $O_1, O_2, \dots, O_m$ ) and rest ( $n - m$ ) output are garbage;

- assigning  $T_{in} = 0$ , this implies,  $er = 0$  for nonfaulty and  $er = 1$  for faulty operations.

Considering  $N$  number of TCs are taken in parallel, which produce  $N$  redundant  $m$  output vectors  $[(O_{11}, O_{12}, \dots, O_{1m}), (O_{21}, O_{22}, \dots, O_{2m}), \dots, (O_{N1}, O_{N2}, \dots, O_{Nm})]$ . The corresponding error signal of TCs are ( $er_1, er_2, \dots, er_N$ ). GOs and constant input of all the building blocks are ignored. All the outputs of each TC is cascaded with corresponding reversible AND gates  $[(AND_{11}, AND_{12}, \dots, AND_{1m}), (AND_{21}, AND_{22}, \dots, AND_{2m}), \dots, (AND_{N1}, AND_{N2}, \dots, AND_{Nm})]$ . The outputs of these AND gates are used as input to corresponding reversible OR gates ( $OR_1, OR_2, \dots, OR_m$ ) which produces fault free output ( $O_1, O_2, \dots, O_m$ ), as depicted in Figure 4. All the connections are similar to MISO TC, except ( $m \times N$ ) AND gate arrays are used in place of single AND gate and  $m$  OR gates for the TCs. The inputs to the AND gates arrays are the outputs of respective TCs, complement of corresponding error signals ( $O_{n1}, O_{n2}, \dots, O_{nN}$ ) and error signals of all previous TCs. The  $O_{ANm}^{th}$  output (output of AND gates) from each AND gate array are fed as input to  $O_m^{th}$  OR gate. The  $m$ th output of the circuit can be calculated using

$$\begin{aligned}
 O_m &= O_{A1m} + O_{A2m} + \dots + O_{ANm} \\
 &= O_{1m} \cdot O_{n11} + O_{2m} \cdot er_1 \cdot O_{n12} \\
 &\quad + \dots + O_{Nm} \cdot er_1 \cdot er_2 \cdot \dots \cdot er_{N-1} \cdot O_{n1N} \\
 &= O_{1m} \cdot \overline{er_1} + O_{2m} \cdot er_1 \cdot \overline{er_2} \\
 &\quad + \dots + O_{Nm} \cdot er_1 \cdot er_2 \cdot \dots \cdot er_{N-1} \cdot \overline{er_N}. \quad (4)
 \end{aligned}$$

For the nonerroneous functioning of any of the TC, the final output  $O_m$  of fault-tolerant MIMO TC circuit is nonfaulty. All the error signal outputs are also taken as inputs to another AND gate for the detection of worst case situation and a register file can also be added to store the fault patterns of  $N$ th TC. The functional and fault patterns for trimodular redundancy (TMR)-based fault-tolerant circuit for three inputs ( $I_1, I_2, I_3$ ) and three outputs ( $O_1, O_2, O_3$ ) TC is also shown in Table 2 when  $T_{in} = 0$ .



**Figure 4. Fault-tolerant MIMO TC model.**

**Table 2. Functional/fault pattern table of fault-tolerant MIMO TC.**

Error signal			NOT Gates Output			AND Gates Output			Final Output	Test Output	Fault Location
$er_3$	$er_2$	$er_1$	$O_{n,t3}$	$O_{n,t2}$	$O_{n,t1}$	$O_{A31} - O_{A33}$	$O_{A21} - O_{A23}$	$O_{A11} - O_{A13}$	$O_1 - O_3$	$T_{out}$	
0	0	0	1	1	1	0	0	$f_{n,fl}$	$f_{n,fl}$	0	all circuit un-faulty
0	0	1	1	1	0	0	$f_{n,fl}$	0	$f_{n,fl}$	0	$TC_1$ faulty
0	1	0	1	0	1	0	0	$f_{n,fl}$	$f_{n,fl}$	0	$TC_2$ faulty
0	1	1	1	0	0	$f_{n,fl}$	0	0	$f_{n,fl}$	0	$TC_1$ and $TC_2$ faulty
1	0	0	0	1	1	0	0	$f_{n,fl}$	$f_{n,fl}$	0	$TC_3$ faulty
1	0	1	0	1	0	0	$f_{n,fl}$	0	$f_{n,fl}$	0	$TC_1$ and $TC_3$ faulty
1	1	0	0	0	1	0	0	0	$f_{n,fl}$	0	$TC_2$ and $TC_3$ faulty
1	1	1	0	0	0	0	0	0	0	1	all circuit faulty

**Features of proposed model**

Based on the above explanations, the concluding features of fault-tolerant MISO and MIMO TC circuit models are listed as follows.

- The output(s) is nonfaulty, for true performance of at least any one of the TCs.
- The operational preference of  $TC_{N-1}$  is higher than  $TC_N$  during operations.
- The worst case, when all the TCs are performing faulty operations, can be detected.
- Fault detection in AND-OR network can be done by using PP structures of AND OR gates, then constructing separate TC for each array, as shown in Figure 5 [7], [9].
- Fault diagnosis of respective TC can also be achieved by examining the faulty patterns from register file.

**Performance assessment**

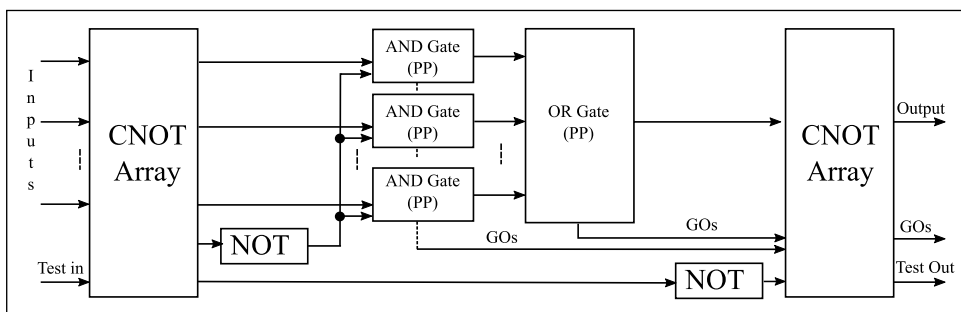
The performance of the proposed methodology is analyzed by its realization on a number of RCs and obtaining the effective cost measures. A comparison is also provided with the results of a best fit prior methodology on the same platform. Arbitrary RCs are converted into corresponding fault-tolerant version utilizing the three essential steps of the proposed methodology.

- Design preserving circuit (RC) using fundamental Toffoli and Fredkin gates.
- Modify RC to form corresponding TC which produce an error signal during a single-bit faulty operation. MCT gates placement and cascading [9], [10] method is considered for the development of TCs in the present case. The circuits are synthesized by creating a set of rules accordingly using transformation-based synthesis algorithm [11].
- Develop fault-tolerant circuits using redundant circuits in accordance with proposed conventions.

Final cost measures are calculated by creating Toffoli Fredkin cascade files and implementing on RC-viewer tool [12]. There are several proven methodologies are provided for the development of TCs in the literature. In spite of the comparing the work with that fault-tolerant methodologies for the direct comparison, the most efficient TC design method in terms of cost metrics is considered for comparison [13].

**Cost measures**

Number of wires ( $n$ ), gate count (GC), QC, GO, and AI are the major parameters which are considered in this correspondence. General equations to calculate these measures in accordance with the proposed model are given by



**Figure 5. Fault detection in AND-OR network.**

$$n_N = Nn_{TC} + m(2N - 1) + 1 \quad (5)$$

$$GC_N = NGC_{TC} + (N - 1)n_{TC} + (2N - 1)m + N + 1 \quad (6)$$

$$GO_N = n_N - m - 1 \quad (7)$$

$$AI_N = n_N - n_v \quad (8)$$

where  $N$  is the number of redundant circuits are used,  $m$  is the number of required outputs of TC, and  $n_v$  is the number of input variables, i.e.,  $(n_{TC} - AI_{TC})$ . Equation for QC is not provided as it does not follow linear relation when similar circuits are cascaded, the results may differ on different tools available for its calculation.

Brief review on TC designing

Numerous methodologies have been proposed which utilize the concept of parity preservation and

generation for the detection of faults in RCs. The construction of TCs has been achieved using novel gates, original circuit modification and designing with built-in testability features [3]. TC using  $R_1/R_2$ , CTSG using online testable gate (OTG), dual rail I/O testable gates and modification using testable reversible circuit (TRC) produces two complementary error signal outputs, respectively [14]–[17], which increase design complexities as well as operating costs. The method which utilizes the concept of modification of RC into modified testable cell (MTC) is only meant for PP gates [7]. Apart from our prior proposed methods [9], [10], extended Toffoli gates (ETGs)-based modification requires only double gates as presented in the original circuit to form corresponding TCs with single error signal output [13].

**Table 3. Implementation results and comparison.**

Circuit ↓ Design → Capability → Cost Measure→	TC Design using [13]						Present Design Method					
	TC		DMR		TMR		TC		DMR		TMR	
	FD		FD+FT				FD		FD+FT			
	GC	QC	GC	QC	GC	QC	GC	QC	GC	QC	GC	QC
nthprime3	18	22	55	95	86	144	14	20	47	91	74	138
rd32	20	28	57	97	89	147	15	27	47	95	74	144
ham3	21	25	61	101	95	153	16	22	51	95	80	144
permanent2×2	21	54	73	195	116	297	18	84	67	255	107	387
xor5	22	22	59	75	92	114	18	18	51	67	80	102
3_17	24	32	67	123	104	186	18	34	55	127	86	192
4mod5	25	29	65	89	101	135	20	26	55	83	86	126
majority	36	149	88	330	136	497	27	253	70	538	109	809
c17	41	131	117	321	183	487	30	210	95	479	150	724
5mod5	42	101	100	234	154	353	31	147	78	326	121	491
hwb4_d1	44	56	111	175	171	265	32	56	87	175	135	265
hwb4_d2	47	59	117	181	180	274	34	58	91	179	141	271
4_49_d1	47	67	117	197	180	298	34	74	91	211	141	319
mth_prime4_d1	47	71	117	205	180	310	34	82	91	227	141	343
4_49_d2	50	70	123	203	189	307	36	74	95	211	147	319
nth_prime4_d2	50	70	123	203	189	307	36	76	95	215	147	325
2of5	50	78	117	189	180	286	36	84	89	201	138	304
ex2	51	167	118	366	181	551	37	269	90	570	139	857
rd53_d1	52	84	128	224	198	340	37	93	98	242	153	367
mod5adder_d1	57	125	145	337	224	510	42	178	115	443	179	669
mod5adder_d2	61	121	153	329	236	498	46	166	123	419	191	633
rd53_d2	62	111	147	277	226	419	46	144	115	343	178	518
rd74	80	136	186	330	286	500	57	157	140	372	217	563
6sym	80	132	180	300	276	454	56	150	132	336	204	508
con1	81	266	184	578	282	871	58	428	138	902	213	1357
cm82a	82	214	188	484	288	730	56	316	136	688	210	1036
ham7	89	113	213	325	327	493	64	112	163	323	252	490
9symd1	105	181	232	400	355	605	75	213	172	464	265	701
9sym	108	188	238	414	364	626	77	223	176	484	271	731
rd84	114	198	262	462	403	701	79	233	192	532	298	806

N=Number of Redundant Circuits, PP: Parity Preserving, FD: Fault Detection, FT: Fault Tolerant



## Results and comparison

A set of benchmark circuits are taken for the experimentation of proposed method from one of the most reliable web pages for RCs [12]. RCs are designed and transformed into respective TCs, which are used to design respective fault-tolerant circuit for double modular redundancy (DMR) and TMR. Note that the MIMO model is applied for designing fault-tolerant circuits. For instance, the number of input variables ( $n_i$ ) in rd32 circuit is 3 and the number of required outputs ( $m$ ) are 2.

After analyzing the prior work on TC designing, ETG-based method to design PP circuits is preferred and corresponding TCs for comparison. The implementation results are listed in Table 3, where number of wires, AIs, and GOs are not listed since they are same in both of the methodologies. These results are obtained without considering fault detection circuit for AND–OR network, AND gates are implemented using MCT gates and OR gates are implemented using Fredkin gates ( $3 \times 3$  MCF gates for simplicity). The sum of all the values listed in each column is calculated and taken for the evaluation of the change in cost metrics in respective cases of TC, DMR, and TMR. Due to the fact that, there are two CNOT gates are used per gate for achieving testability in [13] whose total QC is 2. However in the present method, MCT gates are used which slightly increases the QC by 23%, 19%, and 18% in case of TC, DMR, and TMR circuits, respectively. But, an excellent reduction in gate cost by 28%, 23%, and 22% in respective TC, DMR, and TMR fault-tolerant circuits has been analyzed due to the use of two gates in [13] for testability and the proposed work utilizes only one gate for the construction of TCs.

**PARITY PRESERVATION AND** generation is one of the techniques of achieving testability in RCs as these circuits performs fully controllable and observable operations. Utilizing this technique, TCs are realized which facilitate nonfaulty/faulty information during circuit operation. These TCs are used to design a generalized architecture fault-tolerant RCs with the use of redundant logic. The operation of redundant circuits is governed by AND–OR network that has the capability to deal with the worst situation when all the circuits turned faulty. The model guarantees full coverage of single-bit faults and provides real-time operational behavior TCs. A number of circuits are realized using the proposed methodology where

an excellent reduction is observed in the GC when compared to prior TC development methodologies. The reduction in test overheads by exploring new PP circuit design methodologies will be the primary objective for future extension of the work. ■

## References

- [1] A. Avizienis, "Fault-tolerance: The survival attribute of digital systems," *Proc. IEEE*, vol. 66, no. 10, pp. 1109–1125, Oct. 1978.
- [2] M. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge, U.K.: Cambridge Univ. Press, 2000.
- [3] H. M. Gaur, A. K. Singh, and U. Ghanekar, "A comprehensive and comparative study on online testability for reversible logic," *Pertanika J. Sci. Technol.*, vol. 24, no. 2, pp. 245–271, 2016.
- [4] M. Amy et al., "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 6, pp. 818–830, Jun. 2013.
- [5] H. M. Gaur et al., "Computational analysis and comparison of reversible gates for design and test of logic circuits," *Int. J. Electron.*, vol. 106, no. 11, pp. 1679–1693, 2019.
- [6] H. M. Gaur, A. K. Singh, and U. Ghanekar, "Design of reversible arithmetic logic unit with built-in testability," *IEEE Design Test*, vol. 36, no. 5, pp. 54–61, Oct. 2019.
- [7] H. M. Gaur, A. K. Singh, and U. Ghanekar, "Testable design of reversible circuits using parity preserving gates," *IEEE Design Test*, vol. 35, no. 4, pp. 56–64, Aug. 2018.
- [8] A. K. Singh, H. M. Gaur, and U. Ghanekar, "Fault detection in multiple controlled Fredkin circuits," *IET Circuits, Devices Syst.*, vol. 13, no. 5, pp. 723–729, 2019.
- [9] H. M. Gaur and A. K. Singh, "Design of reversible circuits with high testability," *IET Electron. Lett.*, vol. 52, no. 13, pp. 1102–1104, 2016.
- [10] H. M. Gaur, A. K. Singh, and U. Ghanekar, "A new DFT methodology for  $k$ -CNOT reversible circuits and its implementation using quantum-dot cellular automata," *Int. J. Light Electron Opt.*, vol. 127, no. 22, pp. 10593–10601, 2016.
- [11] D. M. Miller, D. Maslov, and G. W. Dueck, "A transformation based algorithm for reversible logic synthesis," in *Proc. Design Autom. Conf.*, Jun. 2003, pp. 318–323.

- [12] D. Maslov, G. Dueck, and N. Scott. (2004). *Reversible Logic Benchmark Page*. Accessed: Aug. 2018. [Online]. Available: <http://www.cs.uvic.ca/?dmaslov/>
- [13] N. M. Nayeem and J. E. Rice, "Online testable approaches in reversible logic," *J. Electron. Test.*, vol. 29, no. 6, pp. 763–778, 2013.
- [14] D. P. Vasudevan et al., "Reversible-logic design with online testability," *IEEE Trans. Instrum. Meas.*, vol. 55, no. 2, pp. 406–414, Apr. 2006.
- [15] N. Farazmand, M. Zamani, and M. B. Tahoori, "Online fault testing of reversible logic using dual rail coding," in *Proc. 16th IEEE Int. Symp. On-Line Test. (IOLTS)*, Jul. 2010, pp. 204–205.
- [16] H. Thapliyal and A. P. Vinod, "Designing efficient online testable reversible adders with new reversible gate," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2007, pp. 1085–1088.
- [17] S. K. N. Mahammad and K. Veezhinathan, "Constructing online testable circuits using reversible logic," *IEEE Trans. Instrum. Meas.*, vol. 59, no. 1, pp. 101–109, Jan. 2010.

**Hari M. Gaur** is currently working as an Associate Professor with the Department of ECE, ABES Institute of Technology, Ghaziabad, India. His research interests include reversible logic, online and offline testing, and fault-tolerant digital design. Gaur has a PhD from NIT Kurukshetra, Kurukshetra, India.

**Ashutosh K. Singh** is working as a Professor and the Head of the Department of Computer Applications, NIT Kurukshetra, Kurukshetra, India. He is also a Chartered Engineer from U.K. His research interests include verification, synthesis, design, and testing of digital circuits, data science, cloud

computing, machine learning, security, and big data. Singh has a PhD in electronics engineering from Indian Institute of Technology (BHU) Varanasi, Varanasi, India, and a Post Doc from the Department of Computer Science, University of Bristol, U.K.

**Anand Mohan** is currently working as an Institute Professor at Indian Institute of Technology (BHU) Varanasi, Varanasi, India. His current areas of research interest are intelligent instrumentation, fault-tolerant design, robust watermarking algorithms, and information security.

**Masahiro Fujita** is currently working as a Professor at VLSI Design and Education Center, University of Tokyo, Tokyo, Japan. He has done innovative work in the areas of hardware verification, synthesis, testing, and software verification mostly targeting embedded software and web-based programs. Fujita has a PhD in information engineering from the University of Tokyo (1985).

**Dhiraj K. Pradhan** is currently the Chair of Computer Science at the University of Bristol, Bristol, U.K. More recently serving as the Founding CEO of Reliable Computer Technology, Inc. He is a Fellow of both ACM and the Japan Society of Promotion of Science and also a Fellow of IEEE.

■ Direct questions and comments about this article to Hari M. Gaur, Department of Electronics and Communication Engineering, ABES Institute of Technology Ghaziabad, Ghaziabad 201009, India; [leoharimohan84@gmail.com](mailto:leoharimohan84@gmail.com).

# Conference Report

## Report on First and Second ACM/IEEE Workshop on Machine Learning for CAD (MLCAD)

### Marilyn Wolf

University of Nebraska, Lincoln, NE 68588 USA

### Jörg Henkel

Karlsruhe Institute of Technology,  
76131 Karlsruhe, Germany

### Raviv Gal

IBM Haifa, Haifa, Israel

### Ulf Schlichtmann

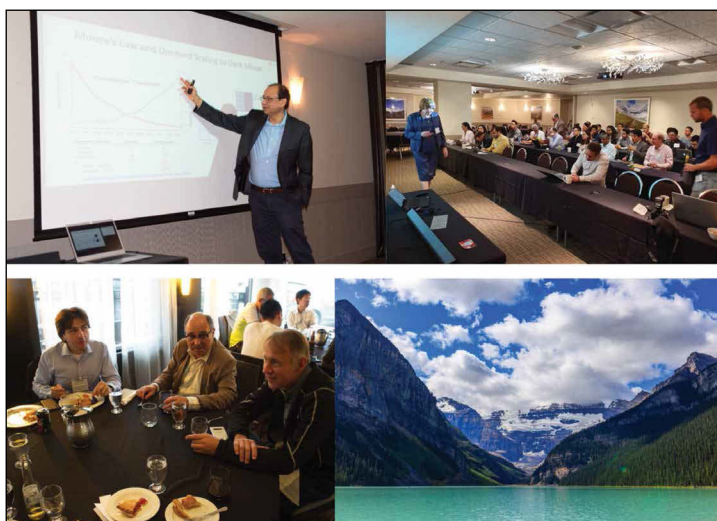
Technical University of Munich, 80333 Munich,  
Germany

■ **THE FIRST** ACM/IEEE Workshop on Machine Learning for CAD (MLCAD) was held on September 2–4, 2020 in Canmore, AB, Canada. The location at the entrance to Banff National Park maintained a long tradition of mountain locations for technical meetings (Figure 1). The workshop welcomed 52 participants including eight graduate students. The program committee was cochaired by Hussam Amrouch of Karlsruhe Institute of Technology and Bei Yu of Chinese University of Hong Kong. General Chairs were Marilyn Wolf and Jörg Henkel. The program included 30 contributed presentations based on submissions to the program committee as well as five invited talks. The program included talks from both industry and academia; participants were based in Asia, Europe, and North America. The program provided time for in-depth discussion; topics included appropriate types of ML methods for various types of CAD problems and challenges associated with training data.

The second edition of MLCAD turned out to be quite different. Originally scheduled to be held physically in September in Iceland, it became apparent in the spring of 2020 that this option was not realistic any more. Initially, it was postponed by two months to November, hoping that by then travel options would be available again. In May, it was decided that an online version was the only

realistic option (Figure 2). At that time, ICCAD was still planning on a physical conference, so a “Highlights of MLCAD” physical meeting as an ICCAD Thursday workshop was planned, to enable physical meeting which the EC considered to be important for interaction among MLCAD researchers. The very helpful support of ICCAD is gratefully acknowledged. However, ICCAD went virtual as well, so that plan was dropped.

Various options were considered for conducting MLCAD virtually. The usual well-known problems had to be dealt with. Discussions with



**Figure 1. Impressions from the First ACM/IEEE MLCAD Workshop in Banff.**

Digital Object Identifier 10.1109/MDAT.2021.3066137

Date of current version: 8 April 2021.

participants in earlier virtual conferences helped us guide our decisions, as did an ACM Best Practices report on virtual conferences and a virtual meeting of all SIGDA-sponsored conferences. In the end, we decided to extend MLCAD into an entire week (Nov. 16–20, 2020), with roughly 3 hours of the live program every day. Talks for contributed articles were prerecorded. The ACM Digital Library turned out to be an excellent choice for hosting not only MLCAD articles, but also the prerecorded videos. The live program each day was weighted toward invited keynote and plenary talks, as well as a panel on the final day. Seventy-five minutes were dedicated to contributed articles each day. Authors would present live highlights of their research for 5 minutes, followed directly by discussion. Zoom was used for the live presentations. Discussion was a mixture of chat-based question (using both Zoom chat as well as a dedicated MLCAD forum we set up in Slack) and questions directly asked orally. Four days of the MLCAD week were focused on Pacific Time, as most invited speakers came from this region, and contributed articles were also dominated by U.S. 7–10 A.M. PST worked well for the U.S. and Europe, but Asia participation suffered.

We specifically did conduct one day on a different schedule (morning in Asia, afternoon/evening in the U.S.). It turned out that this arrangement was not as successful. It effectively shut out Europe (midnight—3 A.M.), and being early afternoon in Pacific time zone, may have interfered with regular meetings which attendees were involved with in their companies or universities. Attendance on this day turned out to be lower compared to other days by about one-third. Overall, discussions were quite lively, and a lot of positive feedback was received both on the contents of the workshop as well as the logistical arrangements.

The workshop featured four keynote and four plenary presentations, mostly from a broad range of industrial companies (Cadence, Huawei, Infineon, Nvidia, Qualcomm, Synopsys, and Xilinx). The opening keynote by Andrew Kahng of UCSD explored the relation of learning, optimization, and scaling in the context of MLCAD. It attracted a lot of discussion. On the final day, a panel with mixed industry and academic representation pondered the state and future of MLCAD.

A total of 26 contributed articles was featured in the program. The accepted articles underwent

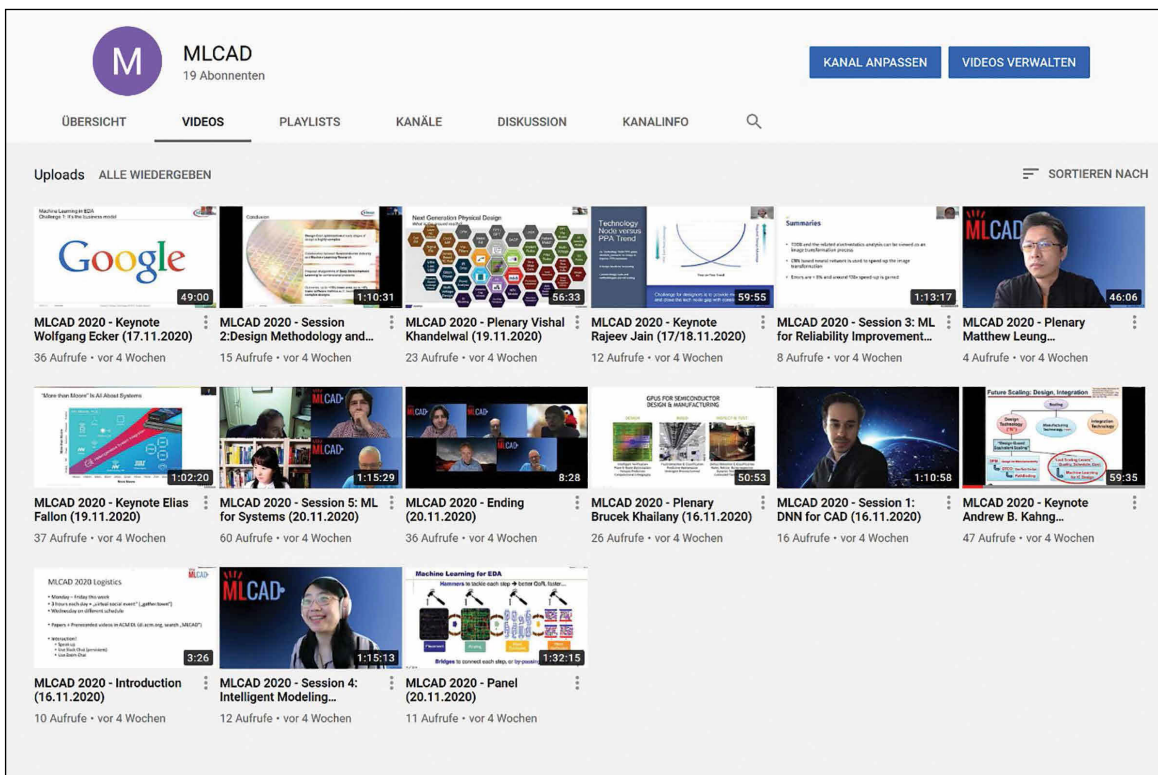


Figure 2. The Second ACM/IEEE MLCAD Workshop, virtual.

a rigorous review by an expert Technical Program Committee of 32 researchers, requiring at least three reviews per article. The contributed articles were dominated by North America and Europe, with 16 and 8 articles, respectively. Article submissions doubled compared to the 2019 edition of the workshop.

Registration more than doubled as well, to a total of 141 registered participants. Most sessions were attended by 70–90 attendees at any time. The available expertise resulted in substantial and lively discussions. Attendee demographics were similar to contributed articles—dominated by the U.S. participation, but a very strong showing from Europe as well, especially from Germany.

MLCAD'2020 also featured a “virtual social event” every day. Jian-Jia Chen (Technical University of Dortmund) set up some environments in the gather.town system. These allowed participants to interact in a manner somewhat resembling a physical social event. Participants explored this option and it did result in some nice conversations, but such a virtual

social event cannot yet match the level of interaction which we expect from physical social events. Certainly, this is also impacted simply by the fact that many participants in a virtual workshop will have their next regular meetings scheduled right after the workshop, whereas in a physical setting, they are actually away from their offices for some days.

**THE PROCEEDINGS OF** MLCAD'2020 are available in both ACM Digital Library and IEEEExplore. Recordings of most invited talks have been made available in a YouTube channel as shown in the figure (see the workshop website at [mlcad.itec.kit.edu](http://mlcad.itec.kit.edu) for details). ■

■ Direct questions and comments about this article to Jörg Henkel, Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany; [henkel@kit.edu](mailto:henkel@kit.edu).

# Conference Report

## Recap of the 39th Edition of the International Conference on Computer-Aided Design (ICCAD 2020)

**Yuan Xie**

University of California at Santa Barbara (UCSB)

■ **THE INTERNATIONAL CONFERENCE** on Computer-Aided Design (ICCAD) is jointly sponsored by IEEE and ACM, and it is the premier forum to explore emerging technology challenges in electronic design automation, present leading-edge R&D solutions, and identify future roadmaps for design automation research areas.

The majority of the past 38 editions of the ICCAD were hosted in California, either in the bay area or in southern California such as San Diego or Irvine. Recently, the executive committee also tried to explore other locations such as Austin or Denver. Unfortunately, 2020's edition is a special one: owing to the global COVID-19 pandemic, many conferences were forced to make the event work in the online world. Without exception, the ICCAD 2020 Executive Committee also decided to move forward with a virtual conference due to the continued uncertainty surrounding the COVID-19 situation.

Nevertheless, we were very excited to test out this new online edition for the first time in ICCAD's 39-year history. Even though the virtual events lack the kind of interpersonal communications attendees get from in-person events, with industry sponsorship and much lower expense due to the virtual platform, we may offer a much lower registration fee to the attendees, and with no travel overheads, it can boost the number of participants. Furthermore, a carefully tuned schedule with a virtual platform can make it a

true "global" event for anyone around the world to attend ICCAD.

The members of the executive committee, the technical program committee, and numerous volunteers have spent several months preparing an exciting program. Despite the global pandemic, we have a record-high in terms of the number of regular article submissions, with 471 regular articles submitted for review by our technical program committee, an almost 20% increase compared to last year's 394 submissions. Submissions in hardware security and neural networks were particularly popular; however, traditional EDA topics such as system design, physical design, verification/validation, and logic synthesis were also well represented. The submissions were divided into 17 tracks and reviewed by 144 outstanding technical program committee members from both industry and academia worldwide. For the first time, the TPC meeting was held online without compromising the quality of the double-blind review process. Finally, the program committee has selected 127 articles spreading over 35 sessions on diverse topics. We also had a record number of special session proposals submitted to ICCAD this year. Altogether, we had 11 special sessions and two embedded tutorials on topics that complement the regular sessions.

The committee carefully planned ICCAD's first-ever virtual conference. With generous industry sponsorship from Alibaba, Cadence, Synopsys, and SMARCO, we were able to offer more than a 10x reduction in registration rate compared to previous ICCAD conferences (for example, \$50 for regular

*Digital Object Identifier 10.1109/MDAT.2021.3051483*

*Date of current version: 8 April 2021.*

rate and \$25 for student rate). Presenters also have prerecorded video uploaded before the conference, so that the live presentation can be much shorter, enabling a modified conference schedule to accommodate various time zones as much as possible. All these efforts helped a huge boost of the registered attendees—compared to the previous year’s ~400 attendees, and we ended up with approximately 900 attendees for the first-ever virtual ICCAD—more than double with respect to any previous ICCAD.

Two best article awards were presented at the opening ceremony. In the “front-end” category, Sujit Kumar Muduli, Gourav Takhar, and Pramod Subramanian were recognized for their article titled “HyperFuzzing for SoC Security Validation.” The “backend” best article award went to Adam Issa, Valeriy Sukharev, and Farid N. Najm for work presented in their article “Electromigration Checking Using a Stochastic Effective Current Model.”

We were delighted to host several distinguished keynote speakers: the Monday morning keynote on AI for enterprises was delivered by IBM Fellow Dr. Ruchir Puri. On Tuesday, Professor Birgit Vogel-Heuser from the Technical University of Munich presented the IEEE CEDA Luncheon Distinguished Lecture on Cyber Physical Systems. Finally, Professor Yao-Wen Chang from National Taiwan University presented the Wednesday keynote on EDA for More-Moore and More-than-Moore Designs.

On Thursday, we have five interesting workshops planned, on a variety of both new and established topics. Some of these workshops are long-time staples of ICCAD, while others test the waters for the first time. Additionally, a workshop addressing system-level interconnect problems is further colocated with ICCAD.

Once again, ICCAD promises to be an ultimate destination for those working on cutting-edge EDA research. I would like to thank the organizing committee and program committee members, the authors and speakers, as well as attendees from all around the world, to make this first-ever virtual ICCAD event a great and memorable one. Finally, we are grateful to our ICCAD 2020 sponsors and numerous supporters for making this year’s conference another successful event.

**HOPEFULLY, THE PANDEMIC** will be over in 2021, and based on that assumption, the next ICCAD will be the first-ever edition to be held outside of U.S., taking place in Munich, Germany, from November 1 to 4, 2021. The executive committee is excited to locate ICCAD for the first time in Europe. See <http://www.iccad.com/> for more details. Once again, ICCAD promises to be an ultimate destination for those working on cutting-edge EDA research. We hope to see you in Munich in November! ■

■ Direct questions and comments about this article to Yuan Xie, ECE Department, University of California at Santa Barbara (UCSB), Santa Barbara, CA 93106 USA; [yuanxie@ucsb.edu](mailto:yuanxie@ucsb.edu).



## TTTC News

The TTTC website always lists the latest features and information for its visitors! To find out more, please visit the website at <http://www.ieee-tttc.org/>.

## PAST TTTC EVENTS

### **The IEEE International Test Conference (ITC 2020)**

November 3–5, 2020

Washington, DC, USA—Virtual Conference

<http://www.itctestweek.org/about-itc/>

ITC is the world's premier venue dedicated to the electronic test of devices, boards, and systems—covering the complete cycle from design verification, design-for-test, design-for-manufacturing, silicon debug, manufacturing test, system test, diagnosis, reliability and failure analysis, and back to process and design improvement. At ITC, design, test, and yield professionals can confront challenges faced by the industry and learn how these challenges are being addressed by the combined efforts of academia, design tool and equipment suppliers, designers, and test engineers. ITC, the cornerstone of the test week event, offers a wide variety of technical activities targeted at test and design theoreticians and practitioners, including formal paper sessions, tutorials, panel sessions, case studies, invited lectures, commercial exhibits and presentations, and a host of ancillary professional meetings.

### **The 24th Design, Automation, and Test in Europe (DATE) Conference**

February 1–5, 2021

Grenoble, France

<https://www.date-conference.com/>

The 24th DATE conference and exhibition is the main European event bringing together design-

ers and design automation users, researchers and vendors, as well as specialists in the design, test, and manufacturing of electronic circuits and systems hardware and software. DATE puts a strong emphasis on both technology and systems, covering ICs/ SoCs, reconfigurable hardware and embedded systems, as well as embedded software. The five-day event consists of a conference with plenary invited papers, regular papers, panels, hot-topic sessions, tutorials, workshops, special focus days, and a track for executives. The scientific conference is complemented by a commercial exhibition showing the state of the art in design and test tools, methodologies, IP and design services, reconfigurable and other hardware platforms, embedded software, and (industrial) design experiences from different application domains, such as automotive, wireless, telecom, and multimedia applications. The organization of user group meetings, fringe meetings, a university booth, a PhD forum, vendor presentations, and social events offers a wide variety of extra opportunities to meet and exchange information on relevant issues for the design automation, design, and test communities. Special space will also be reserved for EU-funded projects to show their results.

## UPCOMING TTTC EVENTS

### **The IEEE VLSI Test Symposium**

April 25–28, 2021—Virtual Live Event

[http://tttc-vts.org/public\\_html/new/2021/](http://tttc-vts.org/public_html/new/2021/)

The IEEE VLSI Test Symposium (VTS) explores emerging trends and novel concepts in testing, debug, and repair of microelectronic circuits and systems.

The VTS Program Committee invites original, unpublished paper submissions for VTS 2021. Proposals for innovative practices and special session tracks are also invited. Paper submissions should be complete manuscripts, up to six pages (inclusive of figures, tables, and bibliography) in a standard IEEE two-column format; papers exceeding the page limit will be returned without review. Authors

*Digital Object Identifier 10.1109/MDAT.2021.3052369*

*Date of current version: 8 April 2021.*



should clearly explain the significance of the work, highlight novel features, and describe its current status. On the title page, please include author name(s) and affiliation(s), and the mailing address, phone number, and e-mail address of the contact author. A 50-word abstract and five keywords identifying the topic area are also required.

The 2021 edition of VTS will be an online virtual interactive live event. The program includes keynotes, scientific paper presentations, short industrial application paper presentations, special sessions, and innovative practices sessions.

### **The 25th IEEE European Test Symposium (ETS'20)**

May 24–28, 2021

Belgium—Virtual Live Event

<http://ets2021.eu/>

The IEEE European Test Symposium (ETS) is Europe's premier forum dedicated to presenting and discussing scientific results, emerging ideas, applications, hot topics, and new trends in the area of electronic-based circuits and system testing, reliability, security, and validation.

In 2021, ETS will be organized virtually online. The symposium is organized by KU Leuven and IMEC that cosponsor the event jointly with the IEEE Council on Electronic Design Automation (CEDA).

The program includes excellent keynotes, scientific papers, and highlights from the industry. In addition to regular paper submissions, ETS offers

a track for informal contributions dedicated to early hot ideas and relevant case studies as well as a PhD forum. A Test Spring School and Fringe Workshops will be organized in conjunction with ETS'21.

### **NEWSLETTER EDITOR'S INVITATION**

I would appreciate input and suggestions about the newsletter from the test community. Please forward your ideas, contributions, and information on awards, conferences, and workshops to Theocharis (Theo) Theocharides, Department of Electrical and Computer Engineering, University of Cyprus, 75 Kallipoleos Avenue, PO Box 20537, Nicosia 1678, Cyprus; [ttheocharides@ucy.ac.cy](mailto:ttheocharides@ucy.ac.cy).

Theo Theocharides  
Editor, TTTC Newsletter

### **BECOME A TTTC MEMBER**

For more details and free membership, browse the TTTC web page: <http://tab.computer.org/tttc>.

- *CONTRIBUTIONS TO THIS NEWSLETTER: Send contributions to Theocharis (Theo) Theocharides, Department of Electrical and Computer Engineering, University of Cyprus, 75 Kallipoleos Avenue, PO Box 20537, Nicosia 1678, Cyprus; [ttheocharides@ucy.ac.cy](mailto:ttheocharides@ucy.ac.cy). For more information, see the TTTC web page: <http://tab.computer.org/tttc>.*

# The Last Byte

## The Road to Open-Source EDA

Scott Davidson

■ **THE VERY THEME OF** this special issue of Design&Test, open-source EDA, resonates with me.

Thirty years ago or so, many large vertically integrated electronics companies had internal EDA departments. They created a lot of innovative tools and algorithms that designers use even today, driven by the advanced ICs used in their products. I worked in this environment.

Although the EDA tools these divisions created were proprietary, they were the open source for those inside the company with a need to know. As mentioned in the Guest Editors' Introduction, being able to build on an already created infrastructure speeds the development of new techniques. I built a Fault simulator that used C models instead of just gates from an excellent and well-documented logic simulator someone else had done, and a colleague added a new Fault model to our test generator.

But I also learned some of the pitfalls of this environment. These should be considered for a successful open-source EDA infrastructure.

Open-source EDA has two uses. One, as mentioned above, is as a base for researching new algorithms. The second, also mentioned in the Guest Editors' Introduction and the articles in this issue, is as a design environment for academic research in design and EDA.

A tool that is part of a design environment must be stable and must be powerful enough to handle the designs it will see. Both can be difficult. Those who fund research want to see innovation. Will they pay for maintenance? Also, doing maintenance does not fit in well with academia. Students do not stay all that long, and tool maintenance is not a good thesis topic, though it might make one quite employable.

Then there is the problem of making the tool capable of handling real designs. The DFT tool Fault, described in one of the articles in this issue, is an excellent start but not powerful enough to handle anything but benchmarks. It takes a long time to get a tool ready for industrial-strength designs. Do funding agencies (and students) have the patience for this?

Another issue is how to incorporate innovations into a stable toolset. While the design database mentioned in the Introduction should help, tools often interact in unexpected ways. A suite of EDA tools must be managed, whether open source or proprietary. The product manager who approves or denies requests for changes may be annoying to developers and even some customers, but without them chaos can ensue. Coordinating tools in a suite developed in many different universities on unsynchronized schedules will be challenging.

None of these issues make open-source EDA a bad idea, but I'd recommend that research concentrate on the hard problems, not problems that commercial tool vendors have already solved. This means that developers of open-source EDA need to understand the capabilities of commercial EDA. Be as familiar with EDA vendor websites as you are with IEEE Transactions on CAD. Ask vendors for copies of manuals. Visit exhibitors at DAC and ITC, either virtual or physical.

And EDA vendors, visit universities and tell them what your dream list is: capabilities you want to include but have no idea of how to implement.

**AND BEST WISHES** to those working on open-source EDA. If I were back in grad school, that's where I'd be. ■

■ Direct questions and comments about this article to Scott Davidson; davidson.scott687@gmail.com; Twitter: @scott687.

Digital Object Identifier 10.1109/MDAT.2021.3053219  
Date of current version: 8 April 2021.



# *Present a world of opportunity*

- **Professional Growth**
- **Collaboration**
- **Global Network**

***When you give the gift of an IEEE Membership,***

you're helping someone you care about build a platform for success.

That's because IEEE delivers access to the industry's most essential technical information, provides both local and global networking opportunities, offers career development tools, plus discounts on conference registrations and insurance programs, as well as many other exclusive member benefits.

***Get started today at***  
**[www.ieee.org/gift](http://www.ieee.org/gift)**



# Technology insight on demand on IEEE.tv

## Internet television gets a mobile makeover

A mobile version of IEEE.tv is now available for convenient viewing. Plus a new app for IEEE.tv can also be found in your app store. Bring an entire network of technology insight with you:

- Convenient access to generations of industry leaders.
- See the inner-workings of the newest innovations.
- Find the trends that are shaping the future.

IEEE Members receive exclusive access to award-winning programs that bring them face-to-face with the what, who, and how of technology today.

**Tune in to where technology lives [www.ieee.tv](http://www.ieee.tv)**

