LiveHD: A Productive Live Hardware Development Flow

Sheng-Hong Wang, Rafael Trapani Possignolo, Haven Blake Skinner, and Jose Renau
University of California, Santa Cruz

***Abstract*—Synthesis and simulation of hardware design can take hours before results are available even for small changes. In contrast, software development embraced live programming to boost productivity. This article proposes LiveHD, an open-source incremental framework for hardware synthesis and simulation that provides feedback within seconds. Three principles for incremental design automation are presented. LiveHD uses an unified VLSI data model, LGraph, to support the implementation of incremental principles for synthesis and simulation. LiveHD also employs a tree-like high-level intermediate representation to interface modern hardware description languages. We present early results comparing with commercial and open source tools. LiveHD can provides feedback for the synthesis, placement, and routing in $<$ 30 s for most changes tested with negligible QoR impact. For the incremental simulation, LiveHD is capable of getting any simulation cycle in under 2 s for a 256 RISC-V core design.**

■ **A RESURGENCE IN** hardware accelerators is thriving with the continued power and performance scaling and the emergence of new specialized domains, such as machine learning. This trend is shown in systems like Apple A13, where CPUs, caches, and GPU represent less than half of the die area, and the rest is composed of accelerators like the NPU and proprietary synthesizable logic blocks. However, current hardware development flow contrast with agile methods, popular in software development. This issue is crucial as application domain experts, who are not hardware designers, turn to

Digital Object Identifier 10.1109/MM.2020.2996508
Date of publication 22 May 2020; date of current version 30 June 2020.

hardware accelerators to make new technologies viable.

We propose a shift in paradigm to improve the hardware design productivity based on open-source and agile hardware development flow. In agile methodologies, designers should implement a feature, get feedback, and react. The sooner in the design-time feedback is available, the easier it will be to react to issues. Feedback should include various design targets, including functional verification, and QoR (frequency, area, and power). Functional changes need to include the cost estimate, and timing/power closure changes need to be functionally validated. Moreover, accurate power analysis depends on toggle factors from the simulation. Therefore, designers need both quick simulation and synthesis. To reach that goal, we advocate for incremental synthesis and simulation techniques in an integrated framework to improve productivity.

Turnaround time of synthesis is one of the major bottlenecks of hardware design. Synthesis is the process of translating the description of a circuit from HDL to gates and then to physical design. Performing synthesis is typically slow and tedious and is often performed on a large design block even after small code changes. The turnaround time for synthesis is typically around hours to days. However, for small code changes, incremental synthesis can prove a valuable tool for designers to get feedback within seconds.

We propose a shift in paradigm to improve the hardware design productivity based on open-source and agile hardware development flow.

There is no open-source incremental synthesis flow, and commercial FPGA flows that support incremental are not ideal. When running two commercial FPGA "incremental" flows in medium-sized designs with only the file time stamp changed, the runtimes were not substantially reduced (see Figure 1). The lack of incremental synthesis forces the industry to have "weekly" synthesis reviews. More academic setups are used to multiple hours for changes in small designs.

Another bottleneck in hardware design is simulation turnaround time. Simulation is used to verify and optimize the logic of a circuit. Commercial flows have incremental compilation

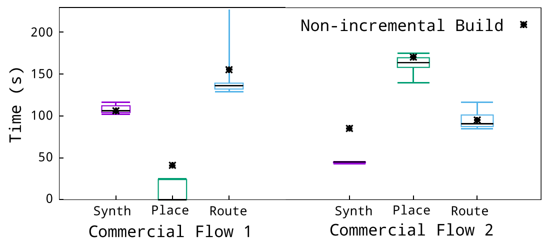

Figure 1. Commercial "incremental" flows have significant overheads even without RTL changes.

options, but even trivially small code changes have a significant compile-time impact. In a large-scale design like modern processors, it is common to spend tens of minutes to recompile with the "fast incremental option." Existing open-source tools are even slower. Verilator,[1] the *de facto* open-source simulator, can be fast for small designs, but does not scale well. While a small RISC-V core can compile <10 s, a design with 64 instances of it takes $40\times$ more time to compile.

Meanwhile, agile software development relies on fast feedback; e.g., a trivial change in a large project like the Linux kernel recompiles in less than 1 min. Recently, live programming received attention in the software community. In live programming, developers see the output from their code changes immediately as the program is always running. Faster feedback leads to a more productive and less frustrating development experience.[2]

In this article, we advocate for an interactive hardware design experience, much like live programming, by providing feedback within a few seconds. This is possible by applying three principles for incremental hardware design: 1) divide the job into partition regions or checkpoints; 2) incrementally transform these partition regions where code change happens; and 3) hot-reload the partition regions into a running program without restarting.

To that end, we present LiveHD (see Figure 2), a new live hardware development framework. By "live," we mean that for small code changes, LiveHD provides design feedback within few seconds. LiveHD follows the three incremental principles; namely, only handle the subjobs related

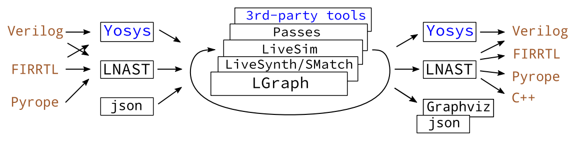

Figure 2. Overview of LiveHD flow.

to the code change, then merge results into the background program without rerunning from the beginning. It builds upon three techniques for incremental hardware design, LiveSynth[3] for logic synthesis, SMatch[4] for P&R for FPGA, and LiveSim[5] for simulation. Table 1 compares LiveHD features with prior art.

LiveHD is built on top of Live Graph (LGraph),[10] a sparse intermediate representation (IR) that serves as a design database and is carefully crafted for live hardware development. LGraph is an LLVM-like infrastructure of hardware design tools by allowing representing large scale designs, performing elaboration, synthesis, and simulation incrementally and leveraging multicores and cloud. LGraph uses an in-house memory-mapped library for fast netlist load/unload and supports hierarchical cross-module traversal to run algorithms on large designs. To allow modern HDLs, LiveHD introduces language-neutral abstract syntax tree (LNAST),[11] which interfaces high-level HDLs with LiveHD.

LiveHD can be coupled with different underlying open-source or commercial synthesis flows and is $5-21\times$ faster than a commercial incremental flow. LiveSynth and SMatch finished the FPGA synthesis, and P&R in less than 30 s for most of the changes tested. For simulation, LiveSim is faster in raw simulation speed than Verilator. When evaluating a 16 node large Partitioned Global Address Space (PGAS) RISC-V multicore, Verilator in single-thread mode has a speed of 51.4 kHz, and LiveSim has a speed of 93 kHz. Moreover, in incremental mode and when using checkpoints, LiveSim can get to any simulation cycle in under $2\ \mathrm{s}$ for a 256 RISC-V core design.

LiveHD

This section describes LiveHD, a hardware design framework that supports simulation, synthesis, and P&R. LiveHD combines several techniques built on top of common IR infrastructures. LiveHD is in active development, but results point to the possibility of interactive hardware design.

Live Synthesis

To perform live synthesis, LiveHD leverages two distinct but complementary ideas: LiveSynth and SMatch. LiveSynth is an incremental logic synthesis flow that updates the existing postsynthesis netlist after a code change. SMatch structurally compares two postsynthesis netlists to find matching gates to reuse placement and routing, thus reducing the amount of P&R work needed. Both techniques focus on highly optimizing a small subregion of the design.

Table 1. Database capability comparison between relevant open-source and commercial tools.

	Simulation	Synthesis	FPGA P&R	ASIC P&R	Live	Open-Source
Qflow[6]	No	Yes	No	Yes	No	Yes
OpenDB[7]	No	Yes	No	Yes	No	Yes
Yosys[8]	No	Yes	No	No	No	Yes
NextPnR[9]	No	No	Yes	No	No	Yes
Verilator	Yes	No	No	Yes	No	Yes
DC/ICC	No	Yes	No	Yes	No	No
VCS	Yes	No	No	No	No	No
Vivado	No	Yes	Yes	No	No	No
LiveHD	Yes	Yes	Yes	No	Yes	Yes

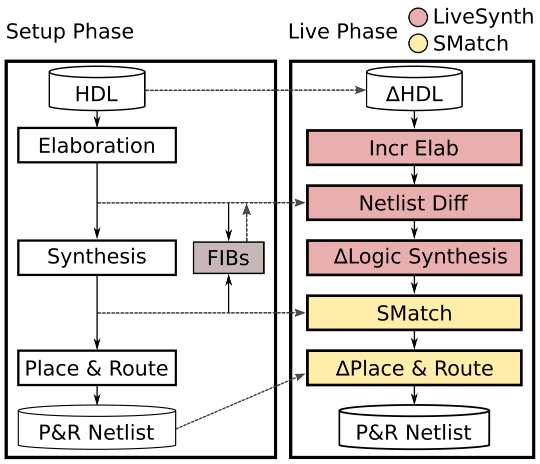

Figure 3. LiveSynth and SMatch operate only on parts of a design that were changed during incremental synthesis, which allows feedback within seconds.

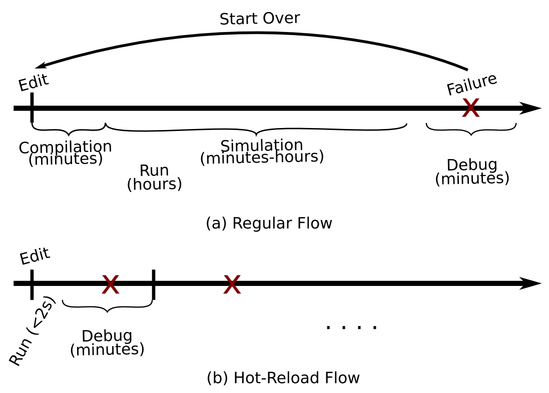

Figure 4. LiveSim uses incremental plus hot-reload to deliver simulation results within seconds. Traditionally, designers wait minutes to hours to reach the point of failure or point where the code must be debugged.

LiveSynth divides the design into multiple regions with function invariant boundaries (FIB),[3] i.e., regions whose boundaries' functionality has not changed during synthesis. These regions are smaller than user-defined modules, but cross-module boundaries. When a change is made in the code, the synthesis flow finds which regions were affected and replace them with the newly synthesized netlist.

SMatch leverages existing P&R and only replaces and reroutes gates as needed. It is possible since: 1) P&R are agnostic to logic function and only depend on netlist structure and the physical dimensions of components; and 2) in FPGAs, the elements of a netlist are of a handful of types. Thus, there are large numbers of equal objects in the netlist. Therefore, two structurally similar netlists will be optimally placed and routed in similar ways, indicating that P&R can be reused. SMatch could also be used in ASICs; however, it would need to match gate size.

LiveHD's synthesis has two phases: Setup and Live (see Figure 3). The Setup phase performs initial synthesis and P&R of the whole design while identifying FIBs between elaborated and synthesized netlists. The Live phase is triggered at every code change and updates existing results based on change.

The LiveSynth portion of the Live phase consists of three steps. The *Incremental Elaboration* re-elaborates only the changed source files. Then, the *Netlist diff* algorithm[3] finds which portions of the netlist have been logically changed and require synthesis. *Netlist diff* traverses the newly elaborated netlist, starting at each FIB and going backward until a new FIB is found. In the third step, ΔLogic Synthesis, only the marked regions are synthesized incrementally and being merged back through the algorithm of *Netlist stitch*.[3]

After a synthesized netlist is generated, the SMatch algorithm[4] identifies nodes, where the newly synthesized netlist is structurally equivalent to the original synthesized netlist. In FPGA netlists, it is plausible to change the logic implemented by a LUT without the need to replace and reroute it, as long as the connections with adjacent LUTs are unchanged. For unmatched or newly added LUTs, SMatch performs a Delta P&R, where most of the design is kept fixed.

Live Simulation

LiveHD uses LiveSim (see Figure 4) as its live simulation framework, it focuses on the RTL simulation, but gate-level simulation could also leverage LiveSim. LiveSim's goal is to drastically reduce the latency between a code change and the availability of results. LiveSim also relies on incrementally parsing the modified design and updating a partitioned internal representation to regenerate output only for the code changed subregions. After that, LiveSim hot reloads the modified subregion while the simulation is running.

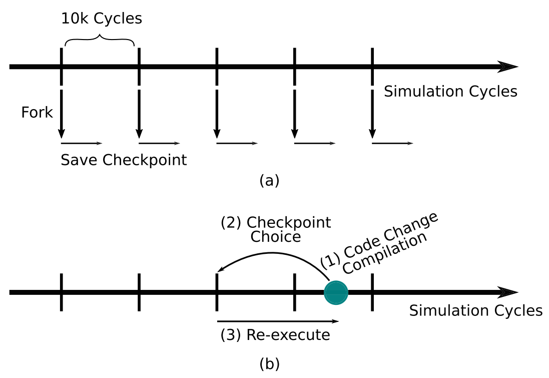

Figure 5. In Baseline mode, checkpoints are regularly created. Then, in Hot-Reload mode, the code changes are recompiled, hot reloaded, a checkpoint is selected, loaded, and the execution continues until the cycle before the code change was inserted.

To reduce compile time, LiveSim avoids duplicating instantiations, which generally leads to slow compilation. If multiple instances of a module use different parameters, LiveSim passes parameters as runtime inputs instead of creating an object. Our experiments show that this helps both in compilation time, but also simulation performance because of the reduced instruction footprint.

LiveSim has two operation modes. The Baseline mode, where compilation and simulation are run regularly for the full code since cycle 0, and the Hot-Reload mode for incremental compilation and simulation. In the Baseline mode, checkpoints are regularly created in every 10,000 cycles to be used during the Hot-Reload mode (see Figure 5).

LiveSim creates multiple dynamic shared libraries instead of a statically linked binary, which allows designers to load/unload modified modules at run time. The checkpoints save the state from all the registers and memories to disk and can be reloaded as simulation start points, as opposed to simulating from the start. When a new state is added to the design being simulated, a default value for new registers is used. This allows for rapid exploration, and the designer may choose to go back to the start of the simulation in cases where this does not work.

Deep functional bugs of complex systems can occur millions of cycles into the simulation, e.g., after booting an operating system and loading an application. Checkpoints are essential to debug these cases. Hot-reload adds on top of checkpointing as a way to further reduce the feedback loop to interactive levels. Loading a large binary for simulation can take several minutes, and hot-reloading specific modules can cut that down to a few seconds. Moreover, checkpoint convergence can be checked in parallel to guarantee that the checkpointed state is still valid. This shifts the traditional paradigm of designers changing the code and having to wait a few days until a validation engineer tests the changes in long simulation runs.

LGraph and LNAST Infrastructure

The lack of a unified database/IR is a challenge to integrate open-source EDA tools for the ASIC design. LiveHD overcomes this obstacle by proposing LGraph and LNAST as the standard IRs, and actively integrates third-party tools and HDLs. LGraph is an SSA and graph-like representation, mostly used by passes in LiveHD. LNAST is a tree-like control flow representation, mostly used to interface between the external HDLs and LiveHD.

LGraph is optimized to support typical synthesis and simulation development from elaboration to layout. LGraph is implemented in C++17 and exposed to developers as an API to manipulate the data structure. The LGraph role of the common data model avoids reparsing netlist and libraries files between design steps. It is especially meaningful when the project goes into the debug or optimization phase, where the changes applied in multiple flow iterations are small, but designers have to wait for the same reparsing time repeatedly.

Modern designs usually contain hundreds of millions to billions of gates. To quickly load/store such large netlists, LGraph memory maps the objects for persistence. Memory mapping maps a disk file directly to virtual memory and, thus, reduces buffer copy operations and manipulating text files. It has the speed advantage for large file processing.[10] We implement a fast memory-mapped library with basic data structures such as vector, hash map, bidirectional hash map, set, and tree. These fundamental containers form the skeleton of LGraph's codebase. They are used extensively for constructing

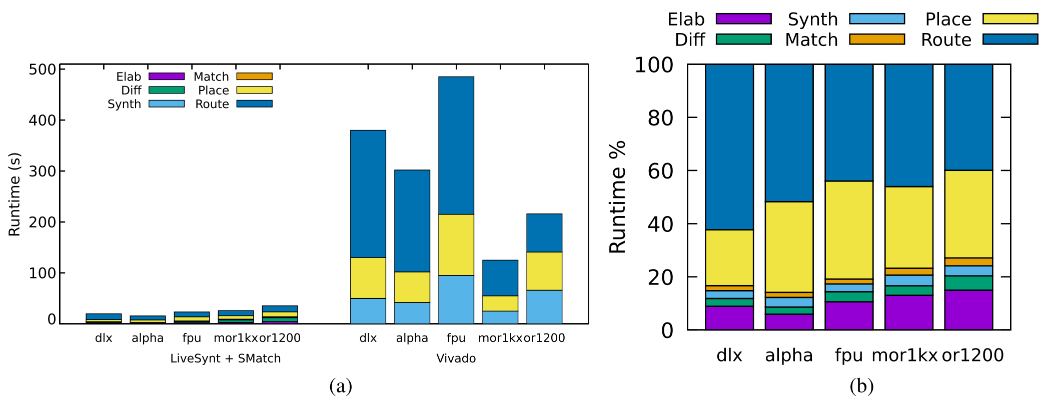

Figure 6. LiveHD performs incremental FPGA logic synthesis and P&R in under 30 s for most changes in the Anubis benchmark suite. (a) Runtime. (b) Breakdown.

graph networks and attributes. As the program gets completed, LGraph's database is automatically synchronized to the disk by the OS.

Performing optimization and transformation hierarchically is vital for efficiency. Nonetheless, many open-source tools lack hierarchical design support. Designers must flatten the hierarchy before using these tools, which increases runtime. LGraph supports hierarchical traversal for integrated tools; each submodule instantiation is represented as a subgraph node in the parent module. When an iterator encounters a subgraph instance, it will traverse the subgraph recursively. This cross-module traversal treats the hierarchical netlist just like a flattened design. Besides, as a module instantiation could have different attributes at the different hierarchy, the LGraph designs an attribute structure for developers to annotate both nonhierarchical and hierarchical characteristics.

LNAST is the higher level representation of a design than LGraph. LGraph and LNAST representations can be seamlessly converted to each other. As a high-level IR, LNAST can capture high-level semantics from different HDLs in the common tree structure. This enables HDLs to leverage the live synthesis and simulation framework in LiveHD. Another critical motivation for designing LNAST is the generation of human-readable C++, Verilog, and other HDLs, which will be an essential infrastructure in LiveHD for fast verification and simulation.

EVALUATION

In this section, we discuss the benefits of LiveHD for synthesis and simulation.

Setup

LiveSynth and SMatch were implemented upon LGraph in C++17, compiled with CLANG 5.0.0. Synthesis was performed with YOSYS version 0.7+312, targeting Xilinx FPGAs. P&R were done using Xilinx Vivado 2017.2. QoR results are reported after routing. Anubis benchmark[12] was used, including five designs (DLX, ALPHA, FPU, MOR1KX, OR1200) and real code changes. We compared QoR between incremental and full synthesis independently for each change. The experiments were run on 2 Intel(R) Xeon(R) E5-2689 CPUs at 2.60 GHz, with 64-GB memory, ArchLinux 4.3.3-3 server.

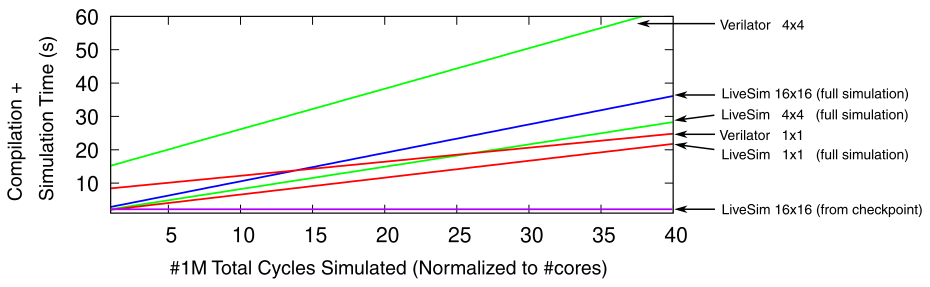

Figure 7. LiveSim has a faster compile and simulation time than Verilator for large designs. When restarting from checkpoints, the advantages is even greater.

Table 2. Compilation time for LiveSim and Verilator.

	1×1	2×2	4×4	8×8	16×16
LiveSim Hot-Reload	1.5	1.5	1.5	1.6	2
LiveSim Full	4.9	4.7	4.8	15.6	176
Verilator	8	14	63	327	NA

Hot-reload can swap a module in under 2 s. Even full compilation with LiveSim is faster than with Verilator.

LiveSim was prototyped in Ruby and is being ported to LGraph. We used a partitioned global address space (PGAS) of RISC-V cores, 1×1, 2 × 2, 4 × 4, 8 × 8, and 16 × 16 meshes. Each PGAS node is a five-stage RISC-V RV64i core with 32 K internal memory. We coded PGAS in Pyrope,[13] an HDL that maps well to LNAST. Both LiveSim and Verilator (version 4.018) generate C++, compiled with g++ 7.3 using the same O2 optimization flag. Experiments were performed in an Intel I7-6700 K Linux server with 32-GB memory. Verilator was shown to outperform commercial simulation tools.[14]

Synthesis Results

LiveHD, using LiveSynth plus SMatch, had a runtime of under 30 s for logic synthesis and P&R, for most changes in the Anubis benchmark suite, making it $5-21\times$ faster than the incremental commercial FPGA flow. Figure 6 reports the runtime for each flow (a) and runtime percentage for LiveHD (b), for each benchmark, averaged across all changes. From the results, it seems the commercial flow only performs incremental P&R (not incremental synthesis), which would explain the large portion of synthesis for the commercial flow runtime results. Even though the proposed flows minimize the amount of P&R needed, they still rely on Vivados placer and router, which, even in incremental mode, is meant to maximize QoR at all costs.

The LiveHD speedup comes from two different places. First, LiveSynth reduces the amount of work during logic synthesis. Then, only changed blocks enter SMatch flow. SMatch further reduces the number of gates for P&R by identifying structural matches in the synthesized netlist.

Simulation Results

Figure 7 demonstrates the accumulated benefits of improved compile times and simulation rates in LiveSim. For a given simulation length (*x*-axis), the difference between the lines for the same design (*y*-axis) shows the reduction in wall-clock time to complete the compilation and simulation. In the best case for LiveSim, incremental simulation and hot-reload are used, and the simulation starts from a checkpoint. In this case, the compilation and simulation time is $\approx$ 2 s from code change to reach virtually any simulation cycle count.

Even the baseline compilation of LiveSim is faster than Verilator (see Table 2). LiveSim full recompilation time ranges from 4.9 s to 176 s depending on the design, and LiveSim Hot-Reload is always achieved in less than 2 s, even for 256 core design. Verilator was $1.5\times$ to $20\times$ slower and

Table 3. For LiveSim (L) and Verilator (V), simulation rate (normalized per simulated core), and microarchitectural measurements of the host processor as the number of simulated cores is increased.

	1 core		4 cores		16 cores		64 cores		256 cores	
	L	V	L	V	L	V	L	V	L	V
Simulation Rate (kHz/core)	1974	2378	1957	2351	1492	823	1223	718	1172	N/A
IPC	2.50	2.86	2.47	2.71	1.94	0.96	1.62	0.80	1.24	N/A
Inst. Cache MPKI	0.25	0.01	0.02	8.45	0.01	20.57	0.01	24.37	0.01	N/A
Data Cache MPKI	0.96	0.01	10.78	0.01	36.42	84.04	39.09	42.99	48.08	N/A
Branch MPKI	1.54	0.01	1.33	0.01	2.83	0.01	3.62	0.29	4.27	N/A

Verilator's performance degrades with a larger design (more simulated cores). Verilator is unable to compile the largest design within a day.

was not able to compile the 256 core design in 24 h regardless of optimization options. The reason is that LiveSim reduces code size by avoiding code replication. The result shows that we can recompile incrementally large designs orders of magnitude faster than Verilator.

LiveSim's simulation has a faster raw simulation speed. Performance counter analysis (see Table 3) reveals a significant increase in miss rates for caches and branch predictor as the number of simulated cores increase in Verilator. LiveSim simply reuses the same object files for the repeated cores, reducing instruction cache miss rate. By reducing the performance detriments from instruction cache misses, LiveSim is more scalable than Verilator, in particular for designs with a large number of instances of the same module.

Hardware development has been plagued by notoriously long synthesis and simulation times, which forms a bottleneck to productivity as developers spend much of their time waiting for feedback. We believe it is time for hardware design to achieve the same level of productivity that exists in software design.

CONCLUSION

Hardware development has been plagued by notoriously long synthesis and simulation times, which forms a bottleneck to productivity as developers spend much of their time waiting for feedback. We believe it is time for hardware design to achieve the same level of productivity that exists in software design. This article advocates for agile hardware design through incremental design, based on three principles: 1) job partitioning; 2) incremental transformation; and 3) hot-reloading. LiveHD is an embodiment of these principles and the first incremental hardware design framework aiming to provide live feedback for small changes for both synthesis and simulation.

The three principles prevent restarting synthesis and simulation from the beginning every time the designer makes trivial changes. Moreover, the use of a common representation, LNAST/LGraph, reduces code replication among EDA steps and various open-source tools, while maintaining high-level semantics of modern HDLs. All this work is available with an open-source BSD-3 license. LiveHD opens research opportunities in many areas, particularly in agile hardware design.

ACKNOWLEDGMENTS

This work was supported by both the Center for Research in Open Source Software (CROSS) at UC Santa Cruz and the National Science Foundation under Grant CCF-1514284. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the NSF. This material is based upon work supported by, or in part, by the Army Research Laboratory and the Army Research Office under Contract/Grant W911NF1910466.

REFERENCES

1. Verilator 4.018, Open-source tool for Verilog HDL simulation. Sep. 5, 2019. [Online]. Available: https://www.veripool.org/wiki/verilator. .
2. J. Dabrowski and E. V. Munson, "40 years of searching for the best computer system response time," *Interacting Comput.*, vol. 23, no. 5, pp. 555–564, 2011.
3. R. T. Possignolo and J. Renau, "LiveSynth: Towards an interactive synthesis flow," in *Proc. 54th Annu. Design Autom. Conf.*, 2017, pp. 1–6.
4. R. T. Possignolo and J. Renau, "SMatch: Structural matching for fast resynthesis in FPGAS," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, Art. no. 75.
5. H. Skinner, R. T. Possignolo, S.-H. Wang, and J. Renau, "LiveSim: A fast hotreload simulator," in *Proc. Int. Symp. Perform. Anal. Syst. Softw.*, Apr. 2020, p. 10.
6. "Qflow," [Online]. Available: http://opencircuitdesign.com/qflow/. Accessed: Apr. 20, 2020.
7. T. Ajayi *et al.*, "Toward an open-source digital flow: First learnings from the openroad project," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, Art. no 76.
8. C. Wolf, "Yosys open SYnthesis suite." [Online]. Available: http://www.clifford.at/yosys/, Accessed: Apr. 10, 2020.
9. D. Shah, E. Hung, C. Wolf, S. Bazanski, D. Gisselquist, and M. Milanovic, "Yosys+ nextpnr: An open source framework from verilog to bitstream for commercial FPGAS," in *Proc. IEEE 27th Annu. Int. Symp. Field-Program. Custom Comput. Mach.*, 2019, pp. 1–4.

10. S.-H. Wang, R. T. Possignolo, Q. Chen, R. Ganpati, and J. Renau, "LGraph: A unified data model and API for productive open-source hardware design," in *Proc. 2nd Workshop Open-Source EDA Technol.*, Nov. 2019, p. 5.
11. S.-H. Wang, A. Sridhar, and J. Renau, "LNAST: A language neutral intermediate representation for hardware description languages," in *Proc. 2nd Workshop Open-Source EDA Technol.*, Nov. 2019, p. 4.
12. R. T. Possignolo, N. Kabylkas, and J. Renau, "ANUBIS: A new benchmark for incremental synthesis," in *Proc. Int. Workshop Logic Synthesis*, Jun. 2017, p. 8.
13. H. Skinner, "Pyrope: A latency-insensitive digital architecture toolchain," Ph.D. dissertation, Univ. California, Santa Cruz, CA, USA, Dec. 2018.
14. P. Bannon, G. Venkataramanan, D. D. Sarma, and E. Talpes, "Computer and redundancy solution for the full self-driving computer," in *Proc. IEEE Hot Chips 31 Symp.*, 2019, pp. 1–22.

Sheng-Hong Wang is currently working toward the Ph.D. degree in computer science and engineering with the University of California, Santa Cruz (UCSC). Before joining UCSC, he had 5 years of experience in digital circuit design at Novatek Microelectronics, Taiwan. His current research focus includes Compiler/EDA tools for productive hardware design and computer architecture. Wang received the B.S. and M.S. degrees in electrical engineering from the National Cheng Kung University. Contact him at swang203@ucsc.edu.

Rafael Trapani Possignolo is currently a Research Scientist with Intel Corporation, researching computer architecture and methodology for hardware design. He coauthored papers spanning from computer architecture and languages to circuit design, with an overall focus on techniques and tools to improve productivity. Trapani Possignolo received the Ph.D. degree in computer engineering from the University of California, Santa Cruz, the B.S. degree in computer engineering and the M.S. degree in electrical engineering from the University of Sao Paulo, and the M.S. degree in embedded systems engineering and an Engineering degree from École Nationale Supérieure des Techniques Avancées. Contact him at rafaeltp@gmail.com.

Haven Blake Skinner currently works at LinkedIn on the Feed Infrastructure team. His research focus was computer architecture and programming languages. Skinner received the Ph.D. degree from the University of California, Santa Cruz (UCSC). During his time at UCSC, he worked on the Pyrope Project, developing an early implementation of the compiler and exploring digital architecture design with fluid pipelines. Contact him at hskinner@ucsc.edu.

Jose Renau is a Professor with the CSE Department, University of California Santa Cruz. His area of research is computer architecture, focusing on productive hardware design flows (live hardware design flow or LiveHD, architectural simulators like ESESC, new hardware description language like Pyrope, new design methodologies like Fluid Pipelines), out-of-order cores, and RISC-V verification. His past projects included thread-level speculation, infrared thermal measurements and power modeling, and design effort metric. For more details visit (http://www.soe.ucsc.edu/renau). Contact him at renau@ucsc.edu.