Design Decisions in LiveHD for HDLs Compilation

Sheng-Hong Wang
University of California - Santa Cruz
USA

Jose Renau
University of California - Santa Cruz
USA

ABSTRACT

We present LiveHD, an under-construction open-source framework to build scalable and incremental solutions for hardware design. LiveHD targets new hardware description languages(HDLs) compilation, simulation, synthesis, and other Electronic Design Automation (EDA) steps. LiveHD has two main data structures with associated APIs: LNAST and LGraph. LNAST is a Language Neutral AST, and LGraph is a graph SSA netlist. In this work, we present many important decisions on the LiveHD internal representations like local vs global type inference, or types of gates allowed.

1 INTRODUCTION

Hardware design is a complex task that requires specialized tools. The most popular language, Verilog, is showing its age with close to 40 Years since its design in 1983. More modern versions like SystemVerilog are compatible with older versions which adds complexity. The main commercial tools also lack many modern features. For example, the top-of-the-line ASIC flows do not have incremental compilation. Even more challenging, many tools lack scalable compilation steps like software compilers do.

From the academic point of view, the closed source nature of most tools is a challenge. It does not allow to improve whole flows or to do changes to improve internal steps. Even more problematic, it is difficult to reproduce results, and it is illegal to report benchmarking results.

When combining with the new pressure to create more custom hardware to compensate for Dennard scaling, the result is a new set of open-source flows and languages to challenge the status quo.

Yosys [16] and Verilator [1] were some of the earlier open-source alternatives. Yosys mostly reads Verilog and interfaces with synthesis tools like ABC [12] or SAT/SMT solvers. More recently, other back-ends like C++, low-level FIRRTL [6] have been added to Yosys. Verilator is widely used in academia and industry to perform Verilog simulations. Both tools are great, but they are highly tied to Verilog and cannot interact directly with new HDLs with higher language abstracts.

Besides Yosys and Verilator, there are some newer tools. The most notable/related to our work are FIRRTL [6], LLHD [8], coreIR [7], and google XLS [2]. Each has very different design options and, as a result, characteristics. This work presents LiveHD and explains some of the different tool internal design options and its decisions.

LATTE '21, April 15, 2021, Virtual, Earth

2 LIVEHD INTERNALS

The main goal of LiveHD is to be a "compiler" infrastructure for hardware; as such, it has a rich API for many operations like graph traversals and storing hardware representations. Most of the APIs provide support to main structures LNAST [14] and LGraph[13].

LNAST is the Language Neutral AST. It allows easy translation for new languages to LiveHD. By targeting LNAST instead of LGraph, the language designer does not need to worry about SSA, control flows conversion, variable scopes, and many other constructs shared by most languages. In a way, LNAST is easier to translate to because it is a control flow with several operations.

LGraph is a hierarchical SSA-graph. By hierarchical, we mean that a graph can point to other graphs, and there are many constructs like hierarchical iterators to handle them.

There are many design decisions for both LNAST and LGraph. The following are some of the most important:

SSA Graph: LGraph has an SSA representation, but others like Yosys internals do not. By SSA-graph, we mean that a wire/net has a write from a single source. The SSA representation has been proved very effective in traditional compilers to simplify passes. The fact that in hardware, pointers are not existing and loops are infrequent makes the SSA even more interesting.

Reduced Instruction Set Logic (RISL) Cells: Logic cell means different graph nodes to represent functionality. Most EDA tools have many logic cells or gate options. While some cells are commonly used, such as *add*, *subtract*, *concatenate wires*, *pick bits*, *one_hot_encoding*, many cells only created for a specific function. Yosys has over 100 types, XLS has 60, and FIRRTL 37. LGraph logic cells have been designed to reduce the number of gates. In a way, it tries to be a Reduced Instruction Set Logic (RISL) in a similar way that RISC is a reduced instruction set vs. CISC. The goal has been to have the smallest amount of cells covering all the options without the overhead. The reason for RISL is to reduce the design complexity on passes and optimizations in the IR.

Signed Wires: Most EDA flows have to deal with signed and unsigned wires. In a flow like Yosys, the logic cells behave differently with signed and unsigned inputs/outputs. This separation adds even more complexity to the logic cells. In FIRRTL, most of the 37 existing cells also have different behavior regarding cell inputs' signedness. However, signed representation is indeed a superset of unsigned. If the flow supports signed, there is no reason to support unsigned. Choosing a unified signed representation leads to less design overhead

(positive values in an unsigned result have the upper bit zero). For example, the bitwidth inference pass no longer needs to differentiate a cell's different input signedness.

N-ary Gates: LiveHD logic cells are n-ary. N-ary means that cells like *or* or *add* can have an unlimited number of inputs. This allows for simpler graphs and easier optimizations.

Wire Width Semantics:. LGraph wires are designed to be always signed, but maybe equally interesting, the cell semantics are independent of the wire widths. This means that a gate-like *concatenate* is not needed because the output is independent of the input wire width. This design choice is because even the designer specifies a "maximum" number of bits to a wire, the tool should be allowed to optimize the wire's bitwidth. This already happens in synthesis tools that can simplify away wires. LiveHD brings the same concept/ideas to LGraph.

Bitwidth Inference: Verilog specifies the bits for each wire. Other modern HDLs like CHISEL [3] or Pyrope [10] have a different bitwidth semantic depending on the operators. For these languages, the flow can not know all the bits at the elaboration phase; instead, a bitwidth analysis must be performed to gain the knowledge of each gate's bit size before code generation. LiveHD has been designed to allow the per-language bitwidth inference at the transformation from LNAST to LGraph.

Global Inference: Another flexibility that LiveHD provides is the global inference. Some programming languages have global type inference, others have local type inference, and others like Verilog and FIRRTL have no inference. To support a wider superset of languages, LiveHD is designed to perform both global and local inference. The types, bits, and other attribute fields are propagated through the graph hierarchy. Other tools, with the exception of ML-based like Clash [15] and Lava [11], seem to have local type inference or none.

Scope Flexibility: Different HDLs has different scope design. In general, we try to be generic enough to cover different HDLs like Verilog, Pyrope, and FIRRTL efficiently. For example, in Verilog and Pyrope, there could be local and module/function scopes. The design of LNAST variables scope [9] resembles Verilog and Pyrope; Statements from the root node are in the top scope; by creating a new sub-LNAST tree, LNAST could have an additional scope hierarchy inside a module (or a function). In a way, the scope representation in LNAST is a superset to the FIRRTL language. This is because in FIRRTL, the scope is per module, which needs the variables to be instantiated at the module level. Therefore, LNAST could also handle the FIRRTL module scope naively.

When considering the design choice such as global type inference and scope in LiveHD, it is almost impossible that create the most generic IRs for semantics and syntax in all HDLs. To overcome this problem, LiveHD allows extensions with attributes in the IRs passes. For example, if a new language wanted to have dependent types [4], the LNAST could annotate the types with attributes, and a new compiler pass at the LGraph level could enforce the refining types.

Prototype Inference: Current LiveHD still does not have a working object model, but the object methods and attributes are builds following prototype inheritance. This means that an object/struct can be extended and changed. It does not require specifying a fixed type.

	Yosys [16]	FIRRTL [6]	XLS [2]	coreIR [7]	LLHD [8]	Verilator [1]	LiveHD
HDLs	V[0]	FIR[1]	C++ DSLX[2]	V Py[3]	SV[4,5]	SV	V SV[6] FIR Prp[7]
Cell Types	>100	~37	~60	~40	~60	>100[8]	31
Aggr. Type[9]	no	yes	yes	yes	yes	yes	yes
Signedness	S/U[a]	S/U	S/U	S/U	S/U	S/U	S
N-ary Gate	no	no	yes	no	no	no	yes
Global Inf[b]	no	no	no	no	no	no	yes
Formal	yes	todo	yes	yes	no	no	todo
Simulation	yes	no	yes	no	yes	yes	yes
Synthesis	yes	no	no	no	todo	no	yes
FPGA	yes	no	no	no	no	no	yes

Table 1: Capability comparison between relevant tools

[0] Verilog-2005, [1] hi-FIRRTL, [2] Google's high-level synthesis language, [3] Python, [4] SystemVerilog, [5] As the time of writing, SystemVerilog is implemented [6] we are now integrating Slang [5] to bridge SystemVerilog, [7] Pyrope, [8] includes many simulation only constructs, [9] High-level aggregate types like tuple, vector [a] Signed/Unsigned, [b] Global type inference,

3 CONCLUSIONS

LiveHD builds an end-to-end hardware flow from front-end HDLs compilation/simulation, to mid-end logic synthesis, and to back-end FPGA place and route. Furthermore, it pursuits for a scalable and incremental compilation. In order to achieve these goals, we have considered and design a different set of options in LiveHD. As discussed in section 2, many decisions/options have broad implications. Certainly, some will prove to be not the best decisions, but exploring and evaluating is the right way to improve.

REFERENCES

[1] 2021. Verilator 4.034, Open-source tool for Verilog HDL simulation. https://www.veripool.org/wiki/verilator. Online; accessed on 5 February 2021.

[2] 2021. XLS: Accelerated HW Synthesis. https://github.com/google/xls. Online; accessed on 9 February 2021.

[3] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. 2012. Chisel: constructing hardware in a scala embedded language. In *DAC Design Automation Conference 2012*. IEEE, 1212–1221.

[4] Jeremy Condit, Matthew Harren, Zachary Anderson, David Gay, and George C Necula. 2007. Dependent types for low-level programming. In *European Symposium on Programming*. Springer, 520–535.

[5] Yong He, Kayvon Fatahalian, and Tim Foley. 2018. Slang: language mechanisms for extensible real-time shading systems. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–13.

[6] Adam Izraelevitz, Jack Koenig, Patrick Li, Richard Lin, Angie Wang, Albert Magyar, Donggyu Kim, Colin Schmidt, Chick Markley, Jim Lawson, et al. 2017. Reusability is FIRRTL ground: Hardware construction languages,

compiler frameworks, and transformations. In *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 209–216.

[7] Cristian Mattarei, Makai Mann, Clark Barrett, Ross G Daly, Dillon Huff, and Pat Hanrahan. 2018. CoSA: Integrated Verification for Agile Hardware Design. In *2018 Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 2–5.

[8] Fabian Schuiki, Andreas Kurth, Tobias Grosser, and Luca Benini. 2020. Llhd: A multi-level intermediate representation for hardware description languages. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 258–271.

[9] Sheng-Hong Wang and Jose Renau. [n.d.]. LNAST API. https://masc.soe.ucsc.edu/lnast-doc/. Online; accessed on 3 April 2021.

[10] Sheng-Hong Wang, Haven Skinner, Sakshi Garg, Hunter Coffman, Kenneth Mayer, Akash Sridhar, Rafael T. Possignolo, and Jose Renau. [n.d.]. Pyrope. https://masc.soe.ucsc.edu/pyrope.html. Online; accessed on 16 February 2021.

[11] Satnam Singh and Philip James-Roxby. 2001. Lava and JBits: From HDL to bitstream in seconds. In *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)*. IEEE, 91–100.

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

[13] Sheng-Hong Wang, Rafael Trapani Possignolo, Qian Chen, Rohan Ganpati, and Jose Renau. 2019. LGraph: A Unified Data Model and API for Productive Open-Source Hardware Design. In *Open-Source EDA Technology, Proceedings of the Second Workshop on (WOSET'19)*.

[14] Sheng-Hong Wang, Akash Sridhar, and Jose Renau. 2019. LNAST: A Language Neutral Intermediate Representation for Hardware Description Languages. In *Open-Source EDA Technology, Proceedings of the Second Workshop on (WOSET'19)*.

[15] Rinse Wester, Christiaan Baaij, and Jan Kuper. 2012. A two step hardware design method using CλaSH. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 181–188.

[16] Clifford Wolf. 2021. Yosys Open SYnthesis Suite. http://www.clifford.at/yosys/. Online; accessed on 10 February 2021.