# A High-level Synthesis Toolchain for the Julia Language

Benedict Short
Imperial College London
London, UK

Ian McInerney
i.mcinerney17@imperial.ac.uk
Imperial College London
London, UK

John Wickerson
j.wickerson@imperial.ac.uk
Imperial College London
London, UK

## The Problem

A key factor in the slow adoption of application-specific hardware designs is the so-called "two-language problem," where scientific algorithms are usually prototyped and developed in high-level languages, but then must be translated into lower-level languages (such as RTL for FPGA designs). This leads to extra development effort, since two implementations generally need to be developed and maintained concurrently, and different teams might be responsible for the two layers.

Solving the two-language problem would mean taking the high-level implementation, written by the domain scientist/engineer, and directly compiling it into a hardware accelerator using High-Level Synthesis (HLS) tools. Most existing HLS tools consume a C/C++-like "high-level" language and are based on compiler stacks developed for compiling and optimizing a stream of sequential operations, such as the LLVM framework. While there has been success deploying HLS in fields such as video processing, graph processing and genomics, this traditional framework is not ideal for FPGA designs, and can lead to HLS-generated designs performing suboptimally and also non-portable source code [2].

One reason for this is a lack of high-level data about the design being available to the HLS compiler when using compiler stacks designed for sequential C/C++. The designer can hint at this missing information by adding pragmas, but those are generally toolchain-specific and non-portable.

## Our contribution

We propose to overcome this lack of information by starting from a higher-level language than C/C++, and by raising the level of abstraction in the compiler toolchain's IR.

To this end, we present an open-source and permissively licensed toolchain that compiles Julia,[1] a language built for science and mathematics that includes native support for common math operations/concepts (e.g., linear algebra), into SystemVerilog by going through MLIR using the CIRCT framework.[2]

Using Julia as the source language for HLS was first advocated by Biggs et al. [1], who presented several advantageous features. For instance, Julia integrates mathematical concepts such as linear algebra as first-class language features; these can be optimised directly by the compiler, without the need to reconstruct them. Moreover, its package ecosystem contains many 'pure Julia' packages, which instead of simply providing a thin wrapper on top of a Fortran or C library, actually implement the algorithms in Julia. This makes it possible to simulate custom numerics formats inside existing algorithms, and could also allow for accelerator cores to be created by just using the algorithms in existing packages.

Our toolchain — called *JuliaHLS* — has advanced analyses and optimisations, such as operator fusion for linear algebra or tiling, that were not previously possible without fragile 'lifting passes' or programmer-annotated compiler directives. We also leverage unique features of the Julia compiler infrastructure to build this tool, including the `AbstractInterpreter` interface and method table overlays. The `AbstractInterpreter` interface allows for us to create a new `MLIRInterpreter` compiler flow that operates alongside the standard Julia compiler while reusing many parts from it (e.g., AST lowering, type analysis, optimisation passes, etc.) without relying on internals of the Julia compiler stack.

JuliaHLS is able to generate high-performance designs that correctly pass timing analysis at 100MHz and achieve up to 82.6% of the throughput of Dynamatic v2.0 [3]. Additionally, in our testing, our toolchain was able to correctly compile several programs that Dynamatic was unable to, and was able to produce designs capable of running at higher clock frequencies.

JuliaHLS can already generate functionally correct hardware from a significant subset of Julia, but it does remain an early-stage toolchain and requires substantial development to achieve performance competitive with hand-written RTL and mature HLS tools. Our plans for future work involve making improvements within Julia's compiler infrastructure, advancing CIRCT's HLS capabilities, and enabling JuliaHLS-generated accelerators to be not just *designed* in Julia, but *used* from Julia too.

For more information about the JuliaHLS project, please see our full paper on arXiv [4] and our organisation on Github:
https://github.com/JuliaHLS

## References

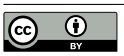
[1] Benjamin Biggs, Ian McInerney, Eric C. Kerrigan, and George A. Constantinides. 2022. High-Level Synthesis Using the Julia Language. In *Proceedings of the 2nd Workshop on Languages, Tools, and Techniques for Accelerator Design (LATTE'22)*. arXiv:2201.11522

[2] Jason Cong, Jason Lau, Gai Liu, Stephen Neuendorffer, Peichen Pan, Kees Vissers, and Zhiru Zhang. 2022. FPGA HLS Today: Successes, Challenges, and Opportunities. *ACM Transactions on Reconfigurable Technology and Systems* 15, 4 (2022). doi:10.1145/3530775

[3] Lana Josipovic, Andrea Guerrieri, and Paolo Ienne. 2022. From C/C++ Code to High-Performance Dataflow Circuits. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 41, 7 (2022). doi:10.1109/TCAD.2021.3105574

[4] Benedict Short, Ian McInerney, and John Wickerson. 2025. A High-level Synthesis Toolchain for the Julia Language. arXiv:2512.15679 [cs.SE] https://arxiv.org/abs/2512.15679

[1] https://julialang.org/
[2] https://github.com/llvm/circt




*FPGA '26, Seaside, CA, USA*

ACM ISBN 979-8-4007-2079-6/2026/02
https://doi.org/10.1145/3748173.3779564