CoreIR : LLVM-Inspired Hardware Intermediate Representation

Ross Daly, Leonard Truong, Jeff Setter, Pat Hanrahan

CoreIR Overview

- CoreIR refers to:
  - Language/Backend-independent hardware IR
  - A C++-based compiler toolchain
- Static, strongly-typed connections
  - Allows for easy static analysis and debugging
- Verilog-style modules and instances with parameterization
- Declaration, instantiation, and linking of modules and generators
- Universal set of hardware primitives

Compiler Toolchain

- C++ codebase inspired by LLVM
- API for Constructing, Traversing, and Manipulating IR
  - Available in C++, C, and Python
- Ability to attach arbitrary metadata for debugging and analysis
- Rich LLVM-Inspired compilation pass framework
- Serialization of IR to compact standard JSON format

Primitives

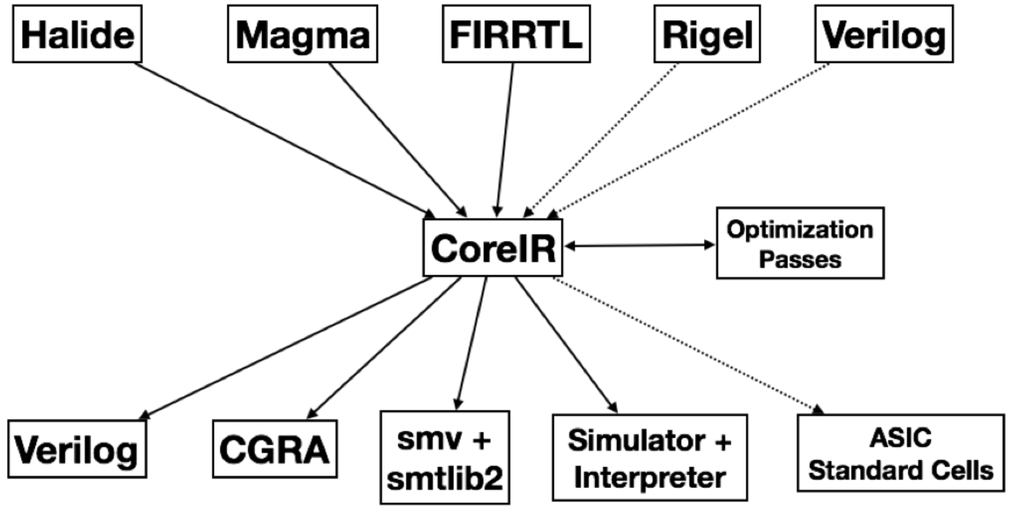
- Equivalent semantics to SMT bit vectors
  - Allows for easy formal verification integration
- Has base primitives and extensions similar to RISC-V ISA
- Current extensions: Memories, Counters
- Future extensions: Fixed Point, Floating Point
- Easy technology mapping by linking backend-specific definitions to each CoreIR primitive

Compilation Pass Framework

- Extensible framework for creating passes similar to LLVM
- Unique views and iterators for easy traversing and manipulating
- Analysis and Transformation Passes
- Allows specification of pass dependencies to automatically remove redundant analysis
- Current passes: flattening, global wiring, generic CSE, debugging, graph culling, graph statistics, etc…
- Future passes: Register Retiming

Hardware Language Interoperability

- Cross language parameter passing for generators
- Common circuit ABIs using CoreIR type system
- Lazy generator evaluation allowing Staged Generator Compilation
- Linking/swapping generator implementations



ISTC Agile