CIRCT: 基于 MLIR 的下一代开源硬件编译框架

叶汉辰 - UIUC 伊利诺伊大学香槟分校

hanchenye@gmail.com

2021 年 6 月 20 日

叶汉辰

伊利诺伊大学香槟分校博士生，分别于 2017 年和 2019 年获得复旦大学本科和硕士学位。他的主要研究方向为高层次综合、硬件编译技术、和深度学习的硬件加速。他曾在 Xilinx 和 SiFive 实习，进行编译器相关的开发和研究。他是 CIRCT 项目的主要贡献者之一，负责其中高层次综合流程的开发和维护。

个人主页：hanchenye.com

深度学习的硬件加速
HybridDNN (DAC'20)
DNNExplorer (ICCAD'20)
Being-ahead (MLBench'21)

高层次综合
ScaleHLS (LATTE'21)

硬件编译技术
CIRCT - Handshake &
StaticLogic & FIRRTL

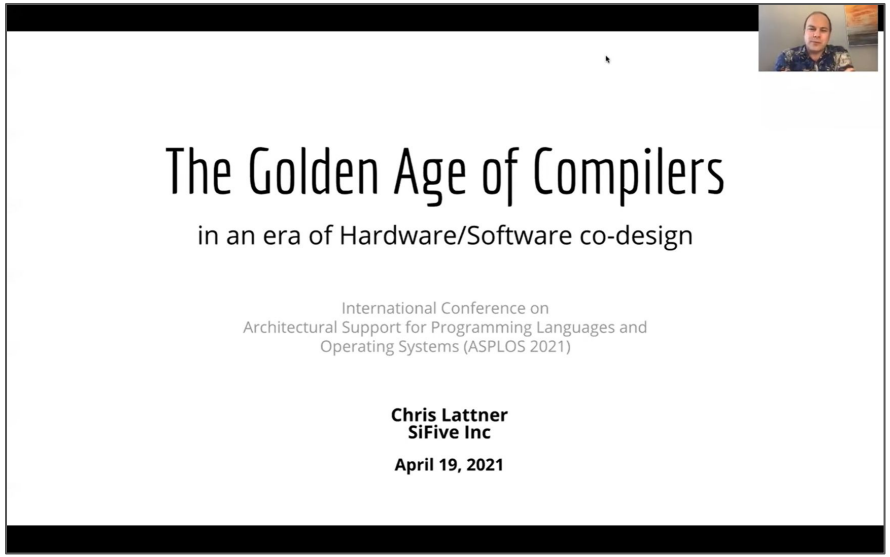
报告大纲

- CIRCT 项目背景
- MLIR: 基于多层次中间表示的开源编译框架
- CIRCT 的主要组成部分
- CIRCT 的应用和影响

报告大纲

- CIRCT 项目背景
- MLIR: 基于多层次中间表示的开源编译框架
- CIRCT 的主要组成部分
- CIRCT 的应用和影响

体系结构和编译器的黄金时代



摩尔定律的终结 → 领域专用架构/加速器 (Domain-Specific Architecture/Accelerator，DSA)

如何对 DSA 进行编程？**如何设计和验证 DSA？**

如何对 DSA 进行编程？以 AI 芯片为例

编程语言

领域专用语言 (Domain-Specific Language，DSL) 或领域专用编程框架

PyTorch

TensorFlow

飞桨

编译器

1. 使用 LLVM 进行优化和代码生成？LLVM 是为编译 CPU 程序而设计，抽象层次太低
2. 使用 AI 芯片专用编译框架？模块化和可扩展性不足，难以处理异构的编译需求
3. 开发定制的编译系统？指令并行、多线程管理、内存管理、异构系统的协同、代码调试、代码生成…

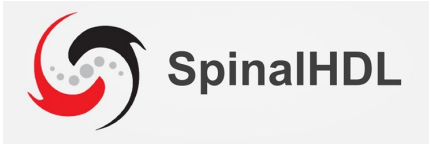
我们需要**模块化、可扩展、支持多种抽象层次**的**统一**编译框架。通过扩展能够对多种不同领域的程序进行中间表示、优化、和代码生成，减少重复造轮子，加快新 DSA 编译系统的开发。

MLIR

如何设计和验证 DSA？

设计语言（编程语言）

1. 高层次综合？主要适用于设计功能性的子模块
2. 领域专用设计语言？高层次综合的领域专用化
3. Verilog/VHDL 是工业界的标准语言，但是：Huge, compilated, incompletely implemented; *Is it an IR? Or a programming language for humans?* [1]
4. Meta HDL？Chisel/SpinalHDL、CλaSH/Bluespec、MyHDL/Migen，敏捷设计的重要发展方向



EDA 工具（编译器）

硬件电路的优化、综合、布局布线、验证可以被映射为软件中的编译过程：

1. 电路抽象为中间表示 (Intermediate Representation, IR)，例如 FIRRTL
2. 优化被映射为 IR 的变换，综合和布局布线被映射为 IR 的降低
3. 验证被映射为 IR 的分析和仿真

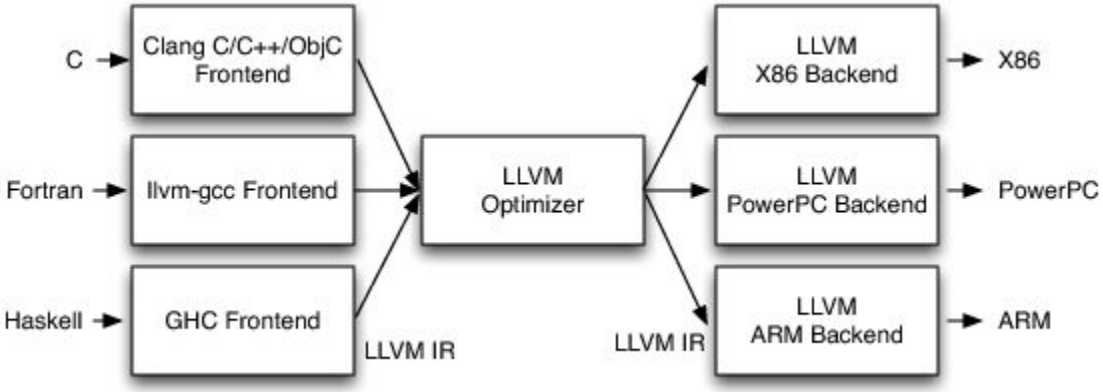
我们需要**模块化、可扩展、支持多种抽象层次**的**硬件编译**框架，对硬件电路进行中间表示、优化、和验证。

[1] Chris Lattner. The golden age of compilers: in an era of hardware/software co-design.

报告大纲

- CIRCT 项目背景
- MLIR：基于多层次中间表示的开源编译框架
- CIRCT 的主要组成部分
- CIRCT 的应用和影响

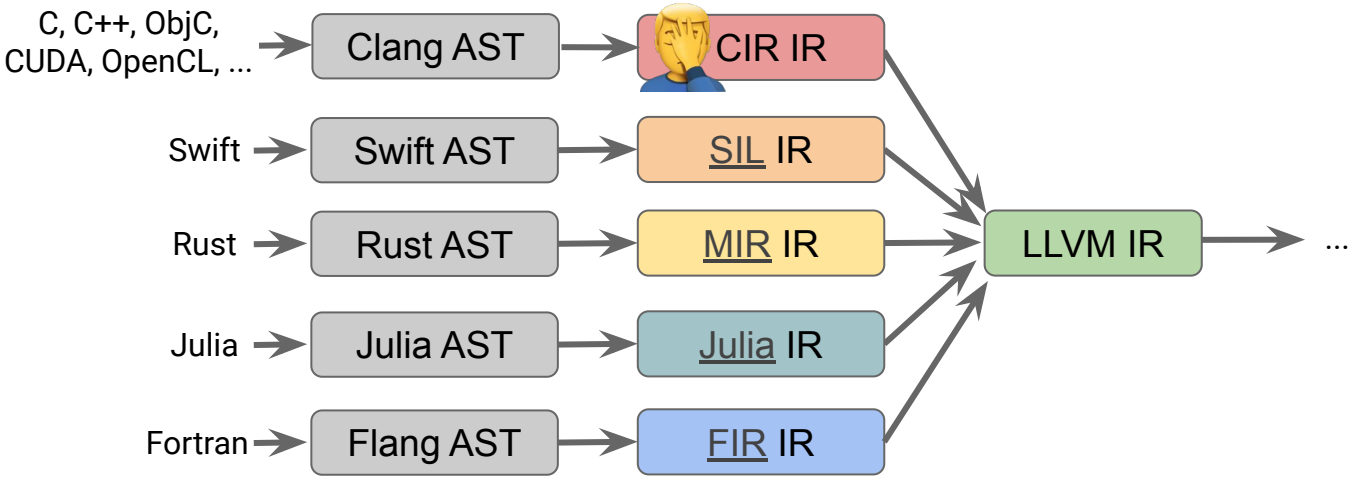
从 LLVM 到 MLIR



- LLVM 用一套 IR 表示所有 CPU 程序
- 基于 LLVM IR 对程序进行优化
- 将编译器的前端、优化、和后端解耦

Source: Chris Lattner. The architecture of open-source applications. LLVM Chapter.

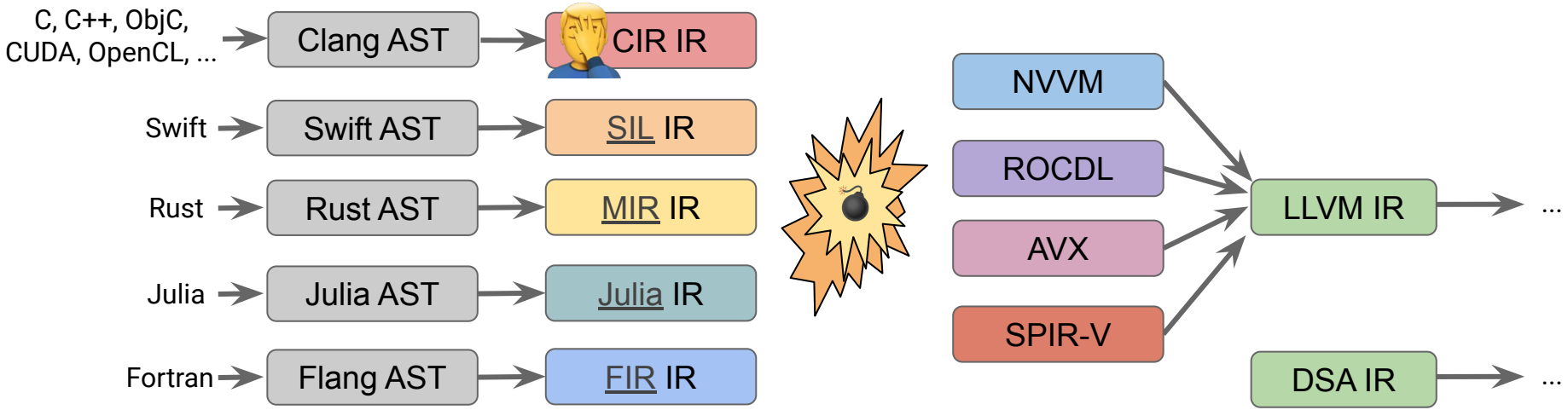
从 LLVM 到 MLIR (Con’t)



- 越来越多的语言需要专用 IR 进行程序优化
- 不同语言的专用 IR 有不同的抽象层次
- 语言专用 IR 可以被降低到 LLVM 以进行后端代码生成

Source: Chris Lattner. MLIR: Multi-Level Intermediate Representation Compiler Infrastructure.

从 LLVM 到 MLIR (Con't)



- 不同的后端需要专用 IR 进行程序优化
- 基于 ASIC/FPGA 的 DSA 可能无法使用 LLVM IR 进行代码生成，需要专用 IR 进行后端代码生成

严重的碎片化：每一种 IR 都有不同的实现方式和“框架”

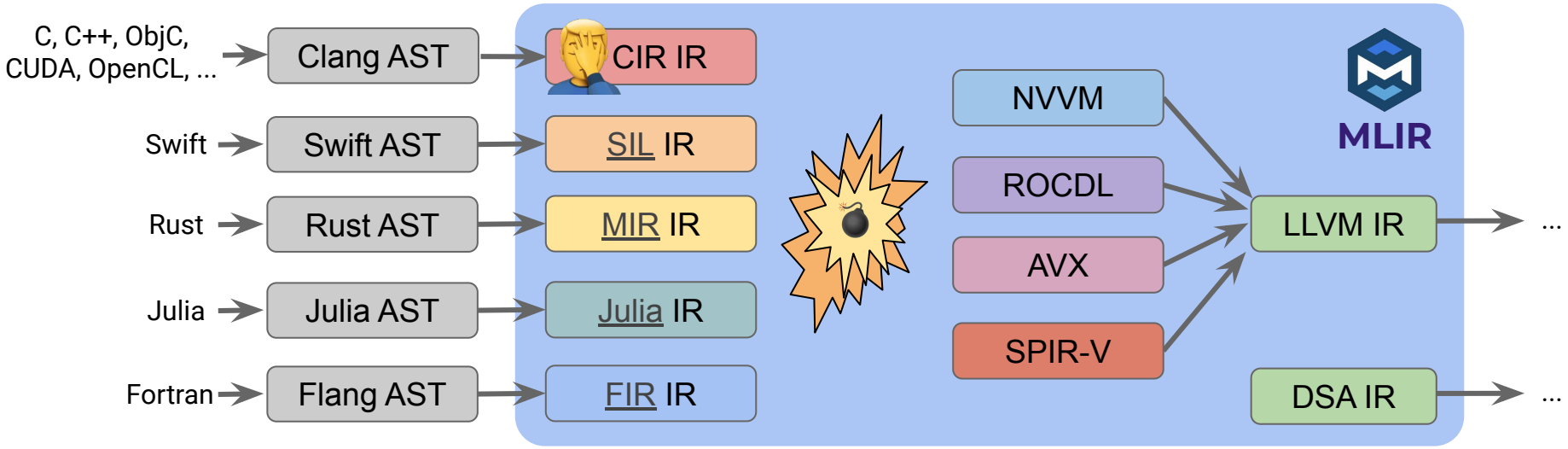
MLIR: Compiler Infrastructure for the End of Moore's Law

- **M**ulti-**L**evel **I**ntermediate **R**epresentation
- State of the art compiler technology
- Built on top of LLVM's open, library based philosophy
- **Modular and extensible**
- Originally created within Google for compiling TensorFlow
- Sufficiently general to compile lots of domains besides ML

https://mlir.llvm.org

Source: Mike Urbach. Applying Circuit IR Compilers and Tools (CIRCT) to ML Applications.

从 LLVM 到 MLIR (Con't)



MLIR 是一个 **Meta IR** 或编译器的基础设施，可以用于：

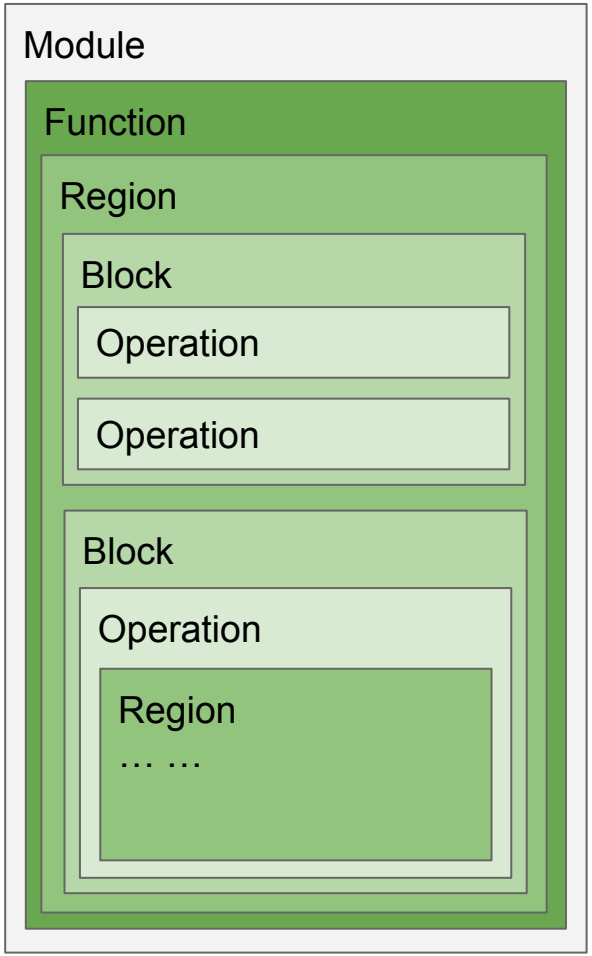
- 设计和实现专用 IR
- **专用 IR 内部**的优化和变换
- **专用 IR 之间**的转换
- 专用 IR 的代码生成

MLIR 的语法

- 基于 SSA 的 IR 设计，显式类型系统
- Module/Function/Region/Block/Operation 的层次结构
- Operation 可以进一步嵌套若干个 Region

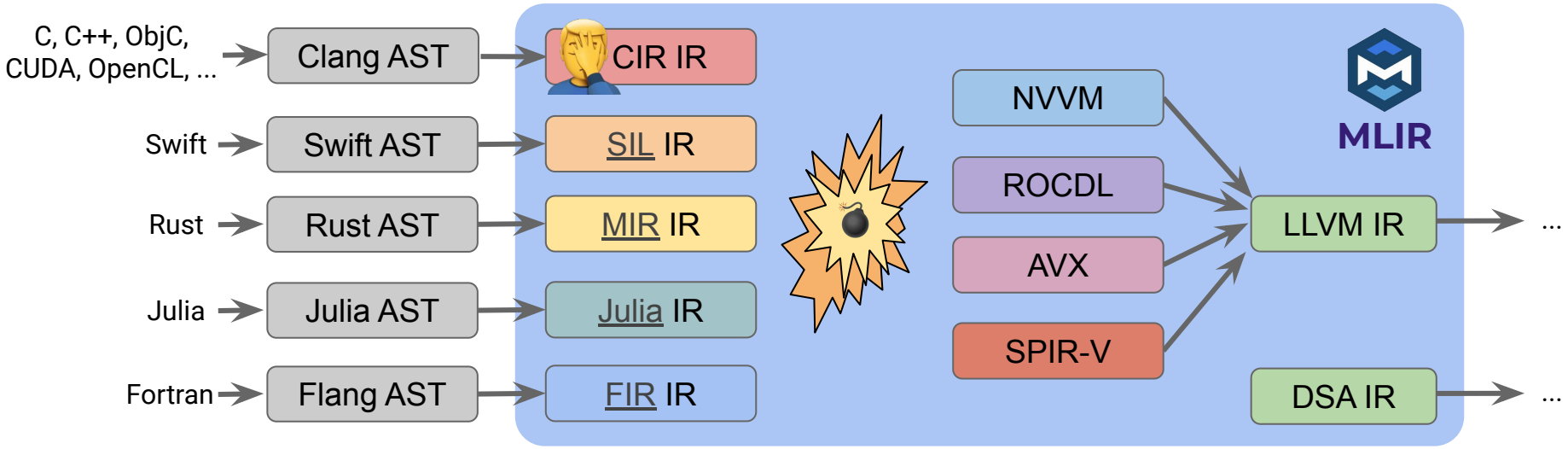
```
func @testFunction(%arg0: i32) {
  %x = call @thingToCall(%arg0) : (i32) -> i32
  br ^bb1
^bb1:
  %y = addi %x, %x : i32
  return %y : i32
}
```

Dialect 包含自定义的 Operation、Type、Attribute 等成员



Source: Chris Lattner. MLIR: Multi-Level Intermediate Representation Compiler Infrastructure.

从 LLVM 到 MLIR (Con't)



MLIR 是一个 **Meta IR** 或编译器的基础设施，可以用于：

- 设计和实现 Dialect
- **Dialect 内部**的优化和变换
- **Dialect 之间**的转换
- Dialect 的代码生成

基于 MLIR 的编译过程 - 以 TensorFlow 为例

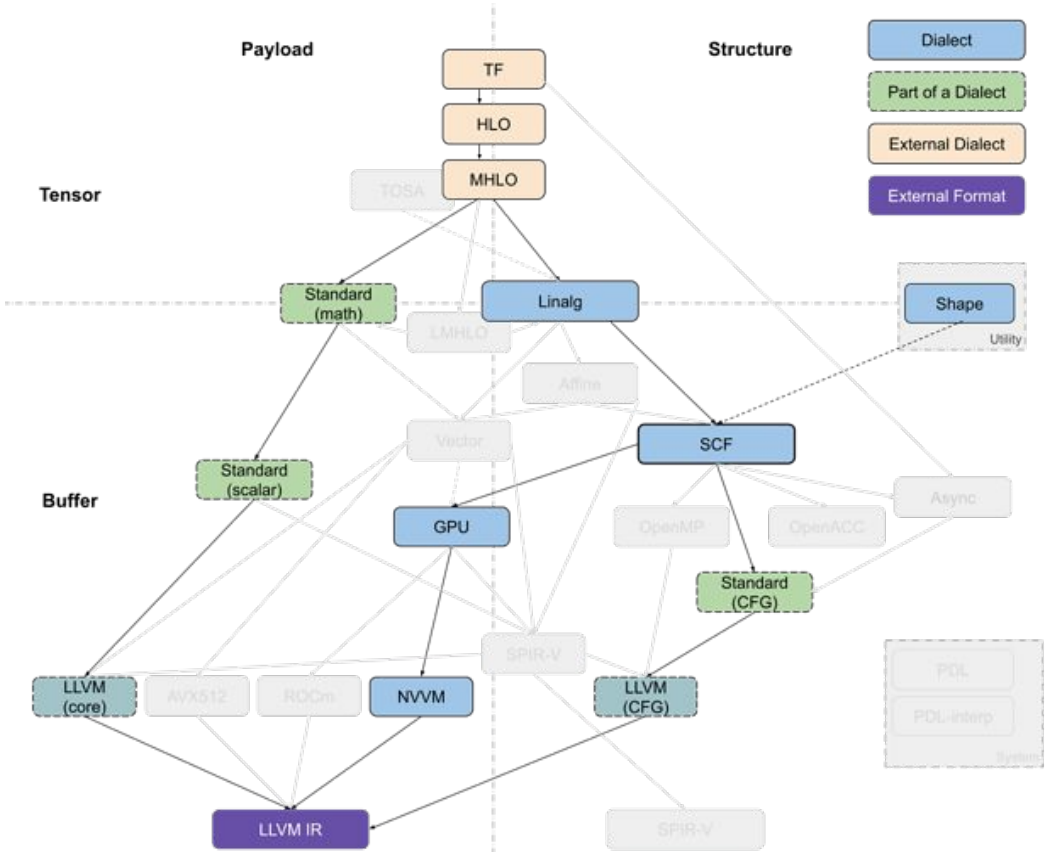


Image source: https://llvm.discourse.group/t/codegen-dialect-overview/2723

报告大纲

- CIRCT 项目背景
- MLIR：基于多层次中间表示的开源编译框架
- CIRCT 的主要组成部分
- CIRCT 的应用和影响

CIRCT: Compiler Infrastructure for the future of EDA

- **C**ircuit **I**ntermediate **R**epresentation **C**ompilers and **T**ools
- Built using MLIR
- LLVM incubator project
- **Composable toolchain for different aspects of electronic design automation (EDA) process**
- Common platform with clean interfaces
- Tools for designing accelerators are relevant for programming accelerators

https://circt.llvm.org

Source: Mike Urbach. Applying Circuit IR Compilers and Tools (CIRCT) to ML Applications.

FIRRTL 编译器: FIRRTL Parser

```
module Foo:
  input clk: Clock
  input bus: {valid: UInt<1>, data: UInt<32>}

  reg dataReg: UInt, clk

  when bus.valid:
    dataReg <= bus.data
```

.fir file

circt-translate -import-firrtl

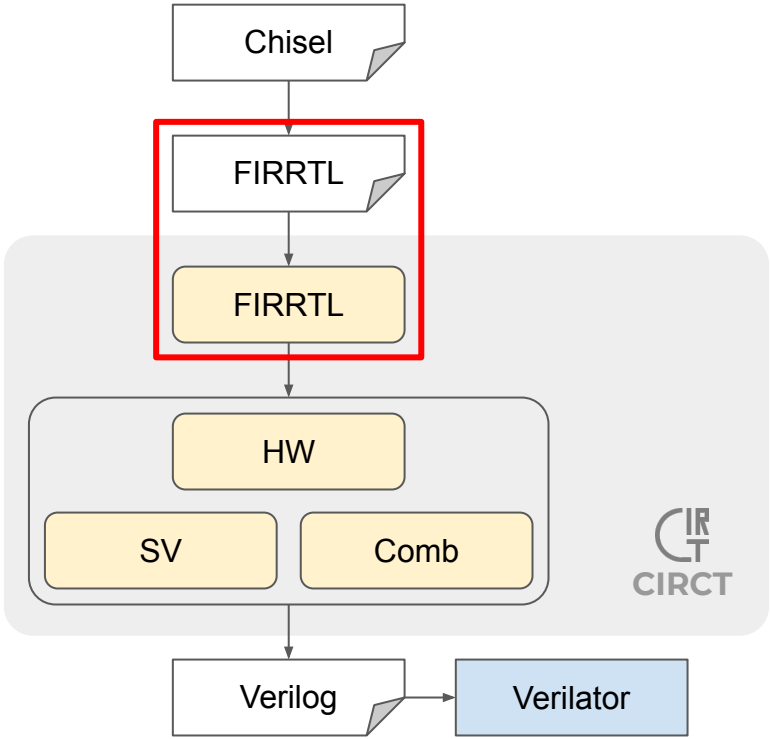
```
firrtl.module @Foo(in %clk: !firrtl.clock, in %bus:
                   !firrtl.bundle<valid: uint<1>, data: uint<32>>) {
  %dataReg = firrtl.reg %clk  : (!firrtl.clock) -> !firrtl.uint

  %0 = firrtl.subfield %bus("valid") :
       (!firrtl.bundle<valid: uint<1>, data: uint<32>>) -> !firrtl.uint<1>

  firrtl.when %0  {
    %1 = firrtl.subfield %bus("data") :
         (!firrtl.bundle<valid: uint<1>, data: uint<32>>) -> !firrtl.uint<32>

    firrtl.connect %dataReg, %1 : !firrtl.uint, !firrtl.uint<32>
} }
```

.mlir file



FIRRTL 编译器: FIRRTL Dialect

```
firrtl.module @Foo(in %clk: !firrtl.clock, in %bus:
                   !firrtl.bundle<valid: uint<1>, data: uint<32>>) {
  %dataReg = firrtl.reg %clk  : (!firrtl.clock) -> !firrtl.uint

  %0 = firrtl.subfield %bus("valid") :
       (!firrtl.bundle<valid: uint<1>, data: uint<32>>) -> !firrtl.uint<1>

  firrtl.when %0  {
    %1 = firrtl.subfield %bus("data") :
         (!firrtl.bundle<valid: uint<1>, data: uint<32>>) -> !firrtl.uint<32>

    firrtl.connect %dataReg, %1 : !firrtl.uint, !firrtl.uint<32>
} }
```

.mlir file: High FIRRTL

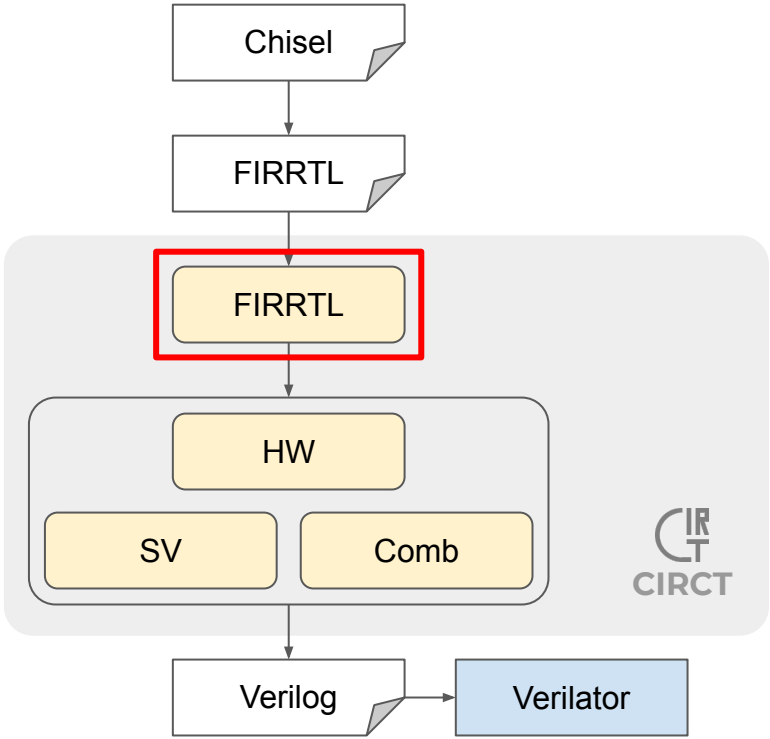
circt-opt -firrtl-lower-types -firrtl-infer-widths -firrtl-expand-whens

```
firrtl.module @Foo(in %clk: !firrtl.clock, in %bus_valid: !firrtl.uint<1>,
                   in %bus_data: !firrtl.uint<32>) {
  %dataReg = firrtl.reg %clk  : (!firrtl.clock) -> !firrtl.uint<32>

  %0 = firrtl.mux(%bus_valid, %bus_data, %dataReg) :
    (!firrtl.uint<1>, !firrtl.uint<32>, !firrtl.uint<32>) -> !firrtl.uint<32>

  firrtl.connect %dataReg, %0 : !firrtl.uint<32>, !firrtl.uint<32>
}
```

.mlir file: Low FIRRTL

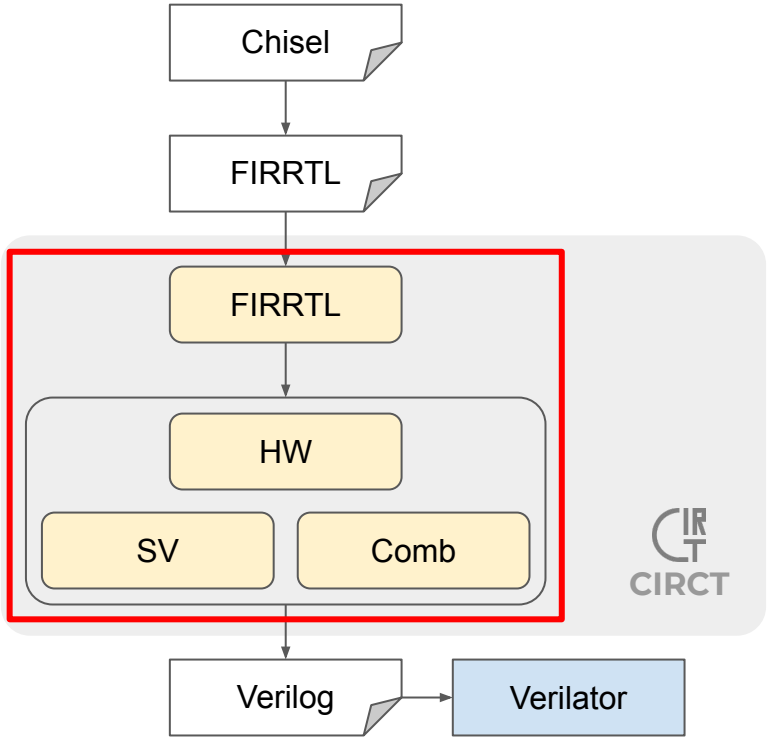


FIRRTL 编译器: HW+SV+Comb Dialect

circt-opt -lower-firrtl-to-hw

```
hw.module @Foo(%clk: i1, %bus_valid: i1, %bus_data: i32) {
  %dataReg = sv.reg : !hw.inout<i32>
  sv.ifdef "SYNTHESIS"  {
  } else  {
    sv.initial  {
      sv.verbatim "`INIT_RANDOM_PROLOG_"
      sv.ifdef.procedural "RANDOMIZE_REG_INIT"  {
        %RANDOM = sv.verbatim.expr "`RANDOM" : () -> i32
        sv.bpassign %dataReg, %RANDOM : i32
      }
    }
  }
  %0 = sv.read_inout %dataReg : !hw.inout<i32>
  %1 = comb.mux %bus_valid, %bus_data, %0 : i32
  sv.alwaysff(posedge %clk)  {
    sv.passign %dataReg, %1 : i32
  }
  hw.output
}
```

.mlir file: HW+SV+Comb



FIRRTL 编译器: HW+SV+Comb Dialect

HW Dialect

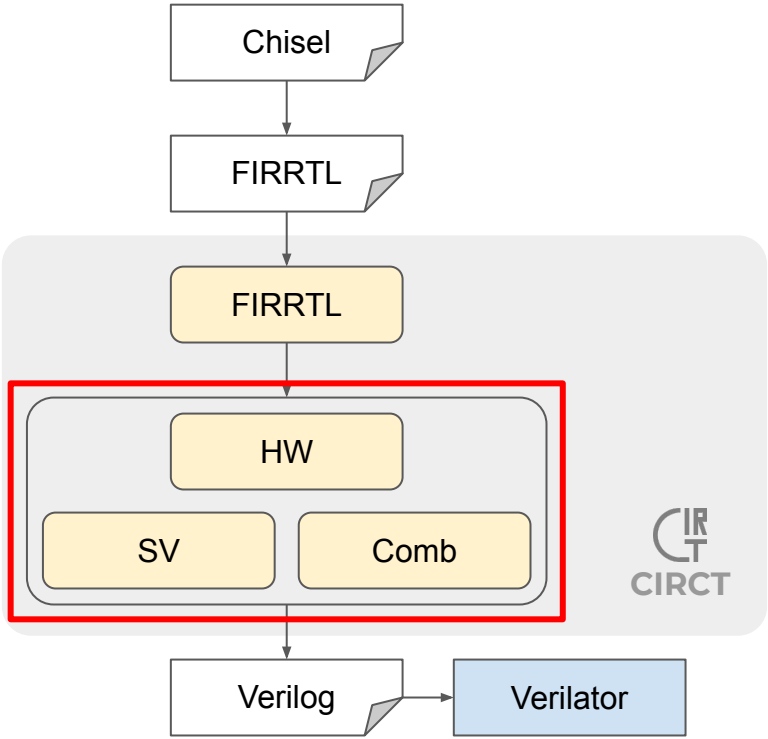
- 抽象硬件电路的结构，例如 (Ext)Module/Instance，以及类型，例如 InOut、Array、Struct、Union
- Module port 可以支持多种类型，例如 SystemVerilog Interface，便于表示不同层次的硬件电路抽象
- 可以与除 SV 和 Comb 之外的其他 Dialect 进行组合
- 便于进行 IR 的分析和变换

SV Dialect

- 表示 SystemVerilog 语言中的声明和结构
- 打印美观的 SystemVerilog 输出文件

Comb Dialect

- 表示电路中的组合逻辑



FIRRTL 编译器: Verilog Emitter

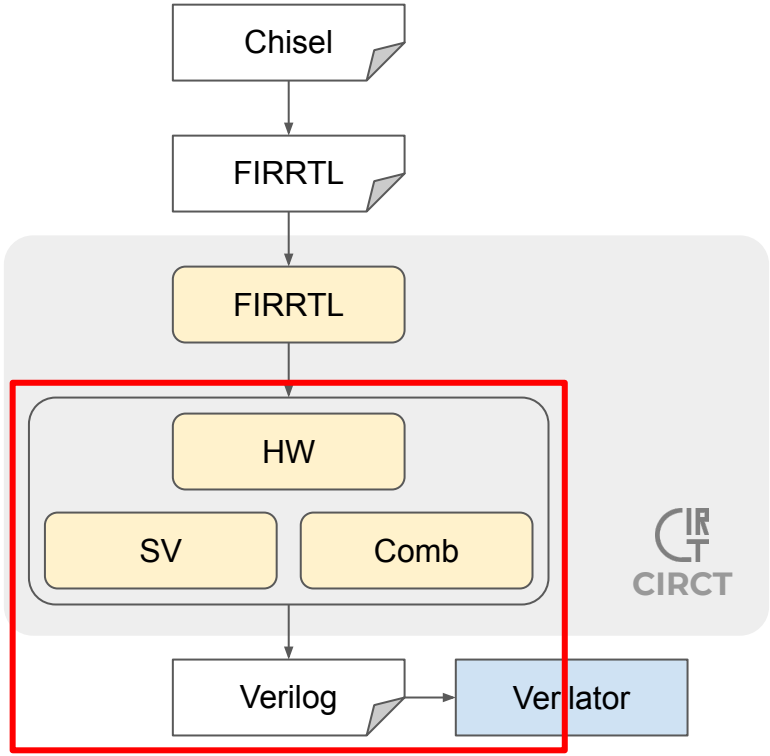
circt-translate -export-verilog

```
module Foo(
 input        clk, bus_valid,
 input [31:0] bus_data);

 reg [31:0] dataReg;    // Foo.mlir:32:16

 `ifndef SYNTHESIS     // Foo.mlir:33:5
   initial begin       // Foo.mlir:35:7
     `INIT_RANDOM_PROLOG_      // Foo.mlir:36:9
     `ifdef RANDOMIZE_REG_INIT // Foo.mlir:37:9
       dataReg = `RANDOM;      // Foo.mlir:38:21, :39:11
     `endif
   end // initial
 `endif
 wire [31:0] _T = bus_valid ? bus_data : dataReg;
// Foo.mlir:43:10, :44:10
 always_ff @(posedge clk)      // Foo.mlir:45:5
   dataReg <= _T;      // Foo.mlir:46:7
endmodule
```

.sv file



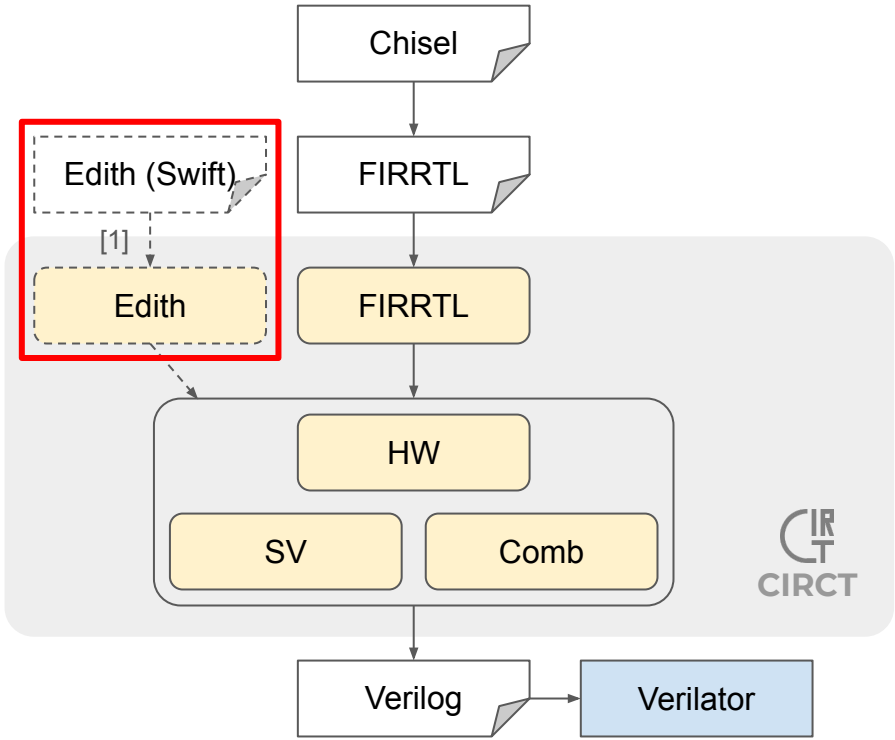
模块化和可扩展性：支持新的 Meta HDL

与 Scala FIRRTL 编译器的对比

- **快！**相比于 Scala FIRRTL 编译器，CIRCT 将编译速度提升了 15-25 倍
- **可复用:** 理论上 HW+SV+Comb 可以用于编译任何 Meta HDL，进行电路优化和 Verilog 生成

Edith [1]

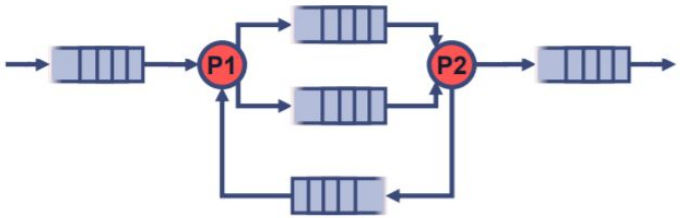
- 基于 Swift 的硬件描述语言
- 通过 Swift binding 在 Swift 中生成 Edith IR
  a. **Self-contained Edith Dialect:** 类似于 FIRRTL Dialect，可以降低到 HW+SV+Comb
  b. **HW+SV+Comb+Edith:** 通过 partial lowering 将 Edith 专用的操作去除，以进行 Verilog 的生成



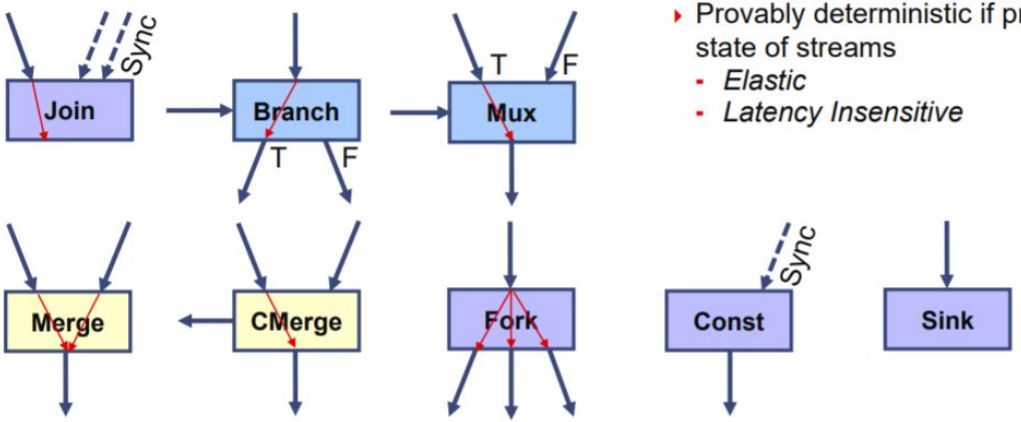
[1] Edith: https://github.com/circt/edith

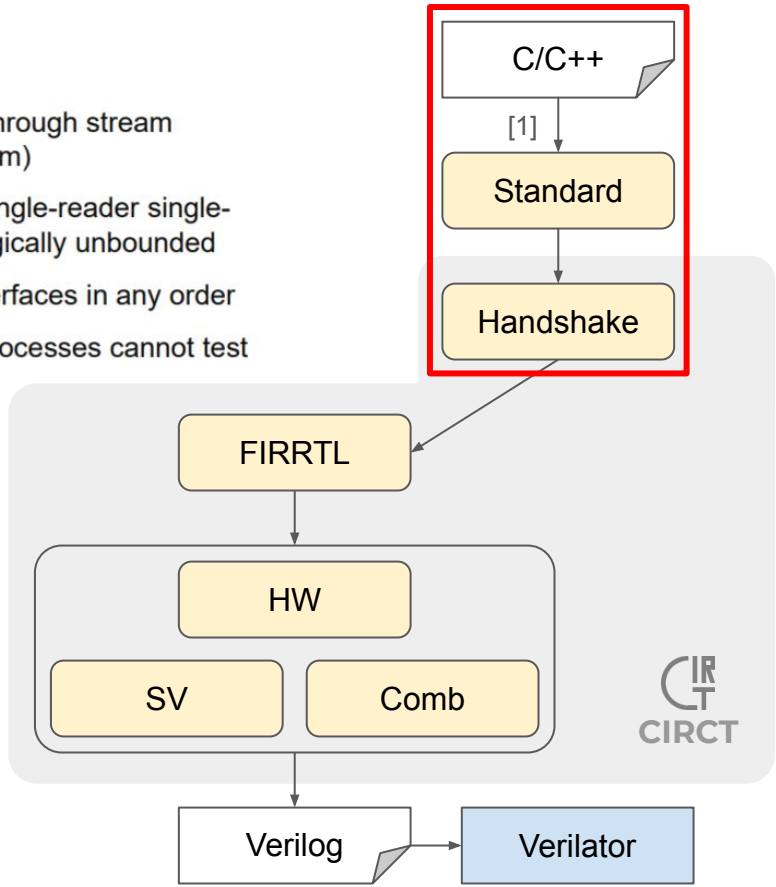
模块化和可扩展性：高层次综合和 DSL

Handshake Dialect



- Processes communicate through stream interfaces (e.g. AXI4-Stream)
- Interfaces connected by single-reader single-writer FIFOs, which are logically unbounded
- Processes can access interfaces in any order
- Provably deterministic if processes cannot test state of streams
  - *Elastic*
  - *Latency Insensitive*





Source: Hanchen Ye. Handshake-based HLS in CIRCT

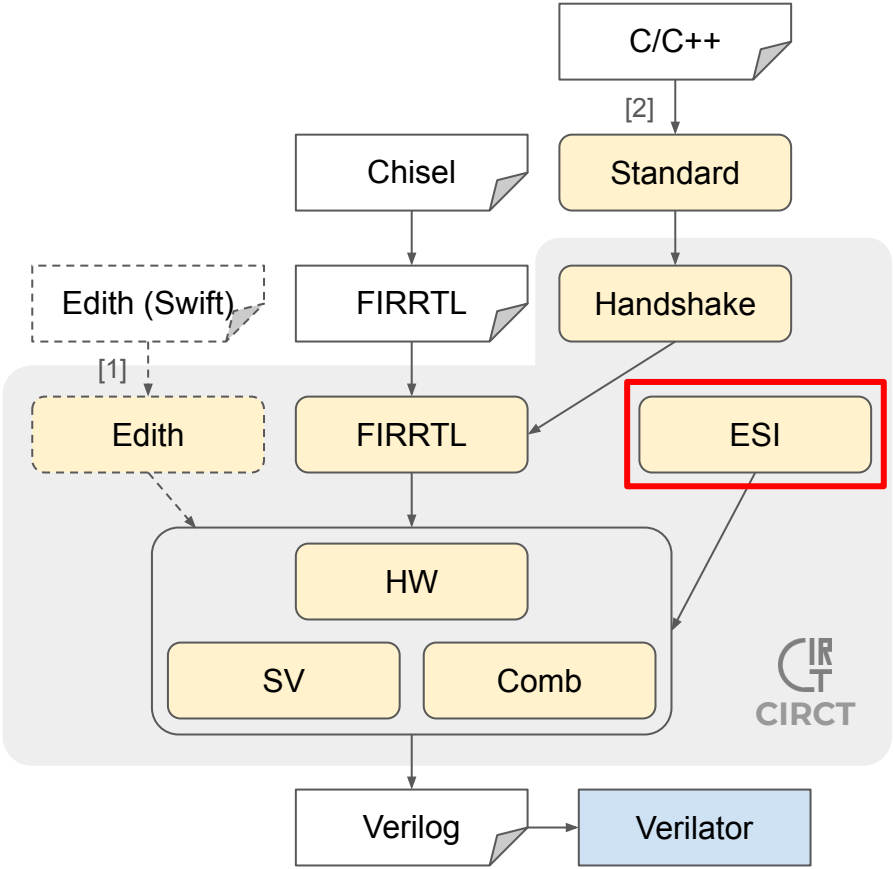
[1] Polygeist: https://github.com/wsmoses/Polygeist

模块化和可扩展性：自定义 Interface

```
hw.module @Sender(%clk: i1) -> (%x: !esi.channel<i4>, %y: i8)
hw.module @ArrSender() -> (%x: !esi.channel<!hw.array<4xi64>>)
hw.module @Reciever(%a: !esi.channel<i4>, %clk: i1)
```

ESI (Elastic Silicon Interface) Dialect

- 适用于硬件接口的强类型系统
- 兼容片上和芯片间的通信
- 可以降低到 SV interface
- 可以实现为延时不敏感的电路
- 支持软硬件联合仿真



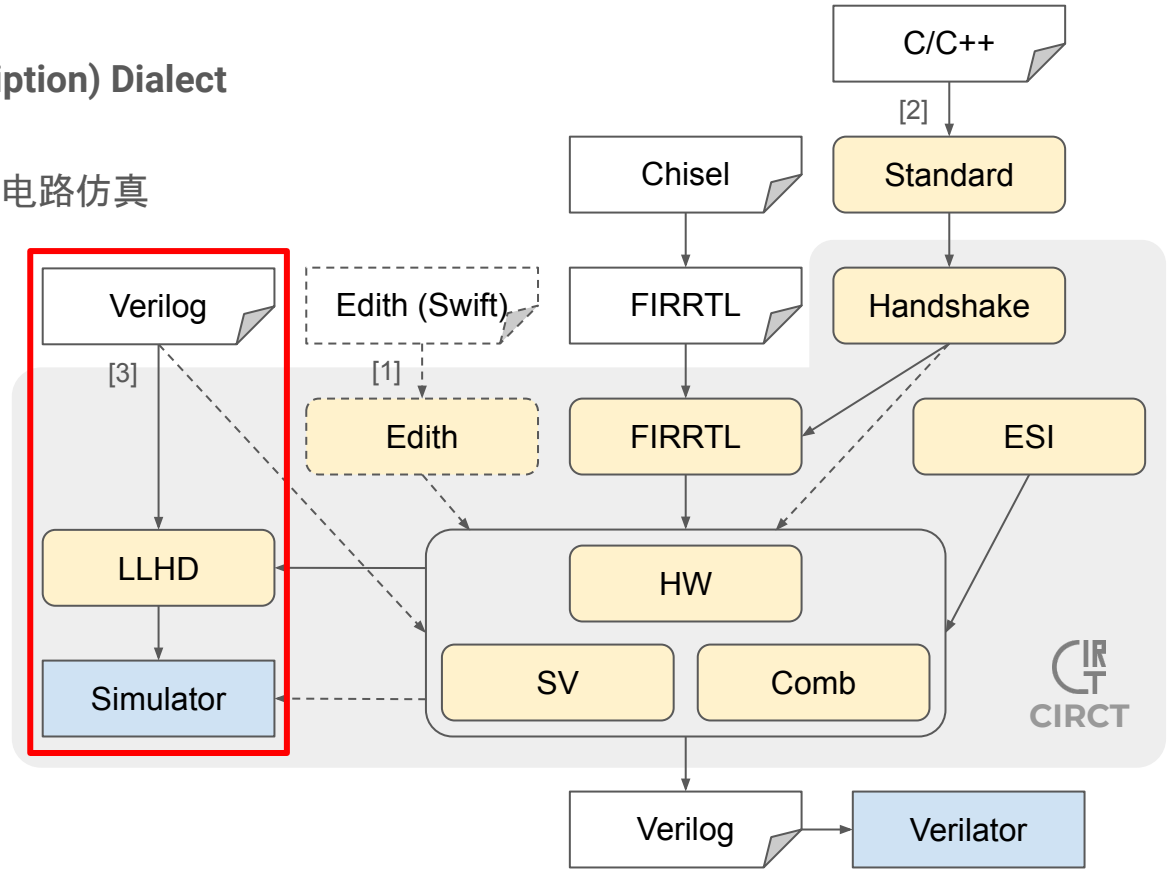
[1] Edith: https://github.com/circt/edith
[2] Polygeist: https://github.com/wsmoses/Polygeist

模块化和可扩展性：多层次电路验证

LLHD (Low Level Hardware Description) Dialect

- 专用于数字电路的中间表示
- 支持 Verilog-to-LLHD parsing 和电路仿真

- 支持 Verilog-to-HW parsing
- 支持 HW+SV+Comb 仿真
- 支持 behavioral、structural、和 netlist 级别硬件 IR 以进行多层次电路验证



[1] Edith: https://github.com/circt/edith
[2] Polygeist: https://github.com/wsmoses/Polygeist
[3] Moore: https://github.com/fabianschuiki/moore

报告大纲

- CIRCT 项目背景
- MLIR:基于多层次中间表示的开源编译框架
- CIRCT 的主要组成部分
- CIRCT 的应用和影响

编译器全景图

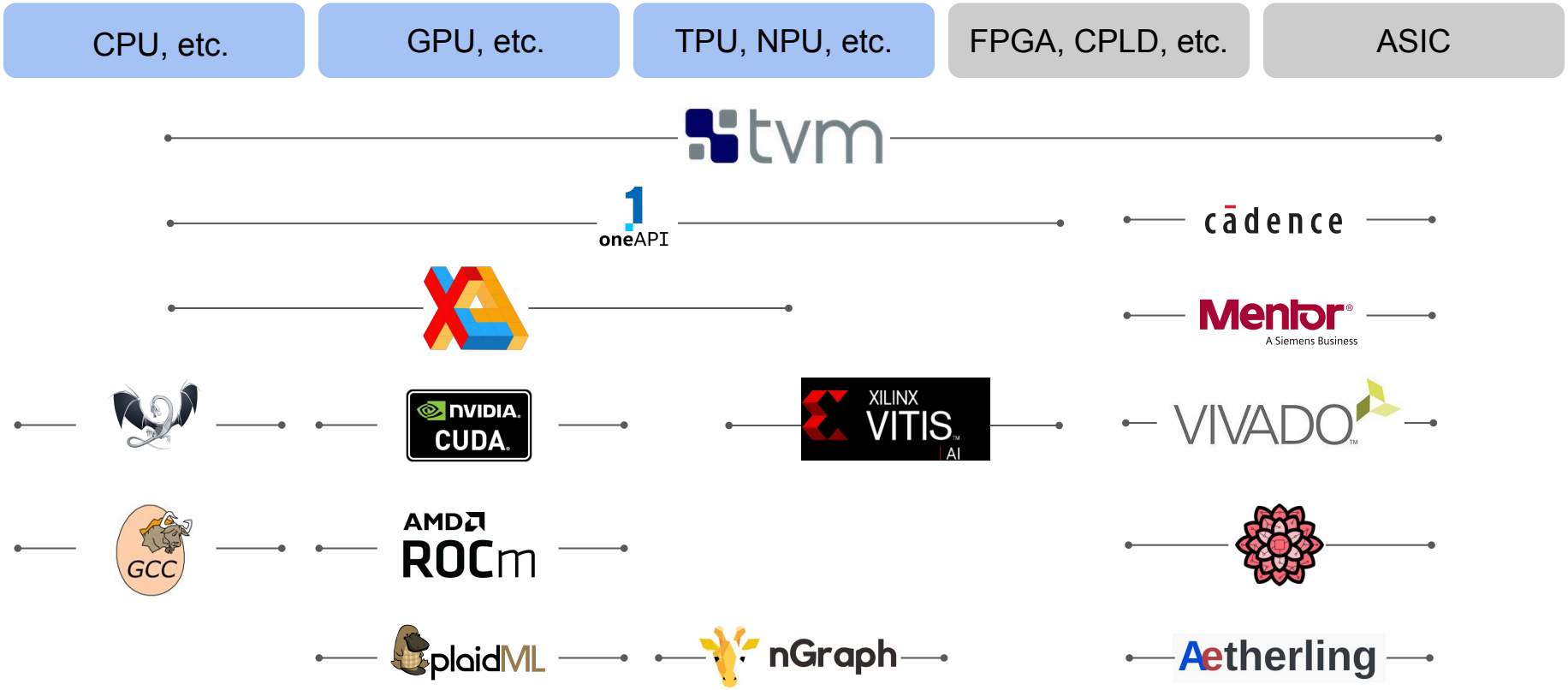


Image source: Mike Urbach. Applying Circuit IR Compilers and Tools (CIRCT) to ML Applications.

编译器全景图 (Con't)

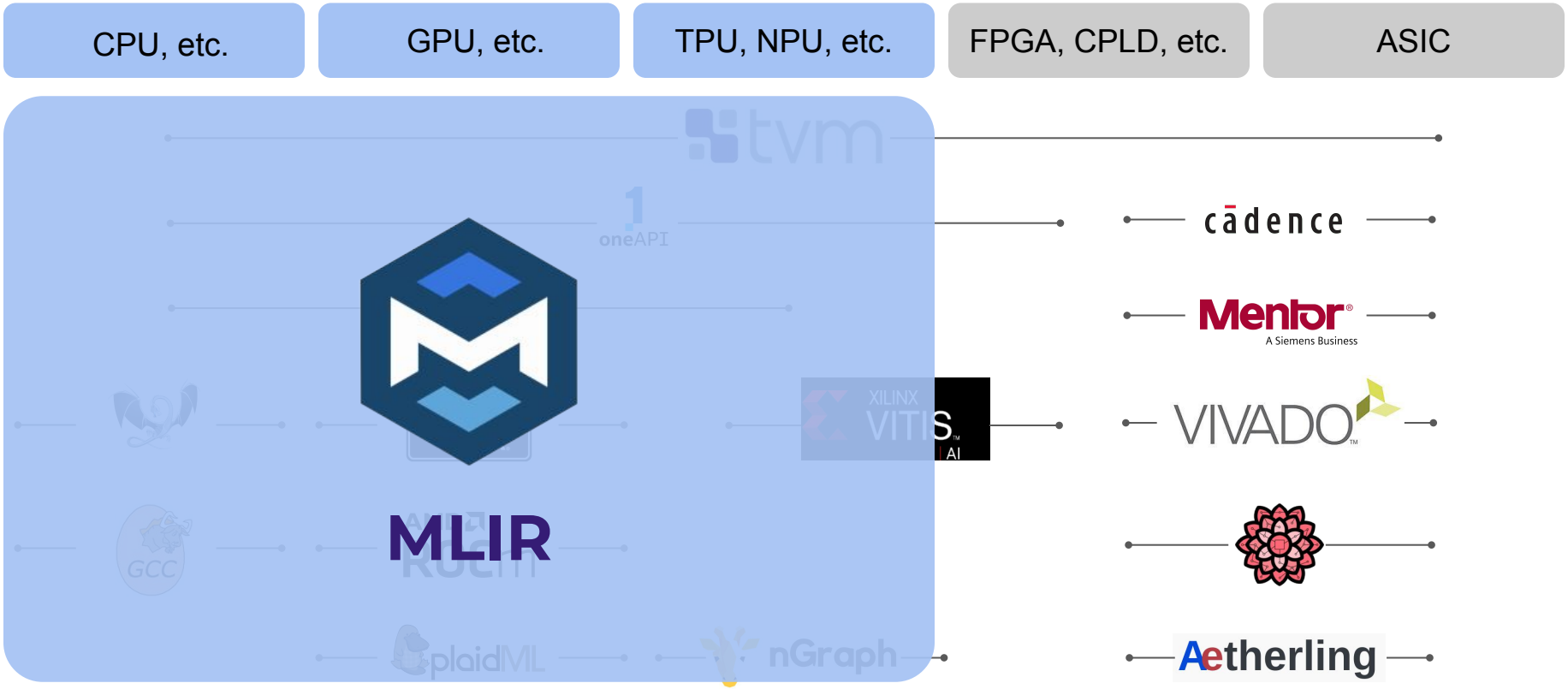


Image source: Mike Urbach. Applying Circuit IR Compilers and Tools (CIRCT) to ML Applications.

编译器全景图 (Con't)

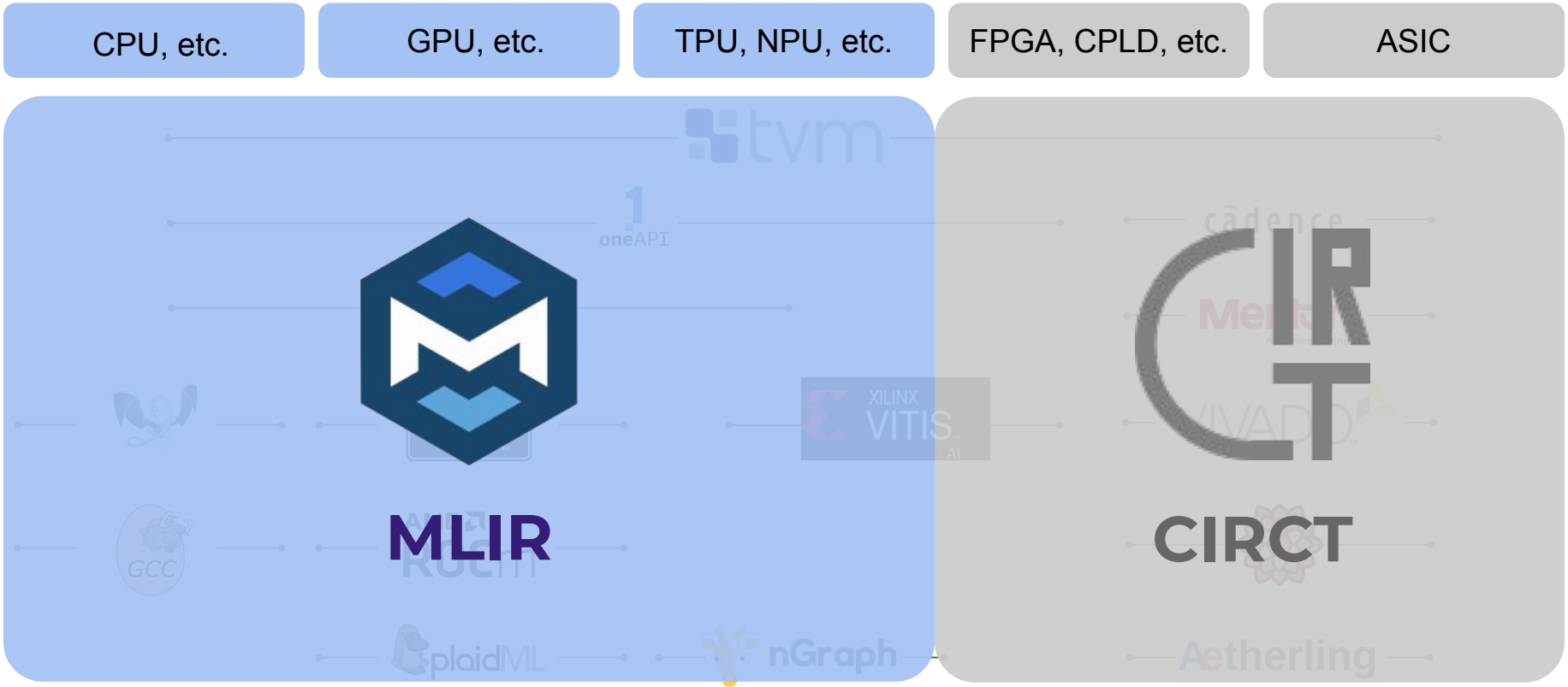


Image source: Mike Urbach. Applying Circuit IR Compilers and Tools (CIRCT) to ML Applications.

软硬件协同设计

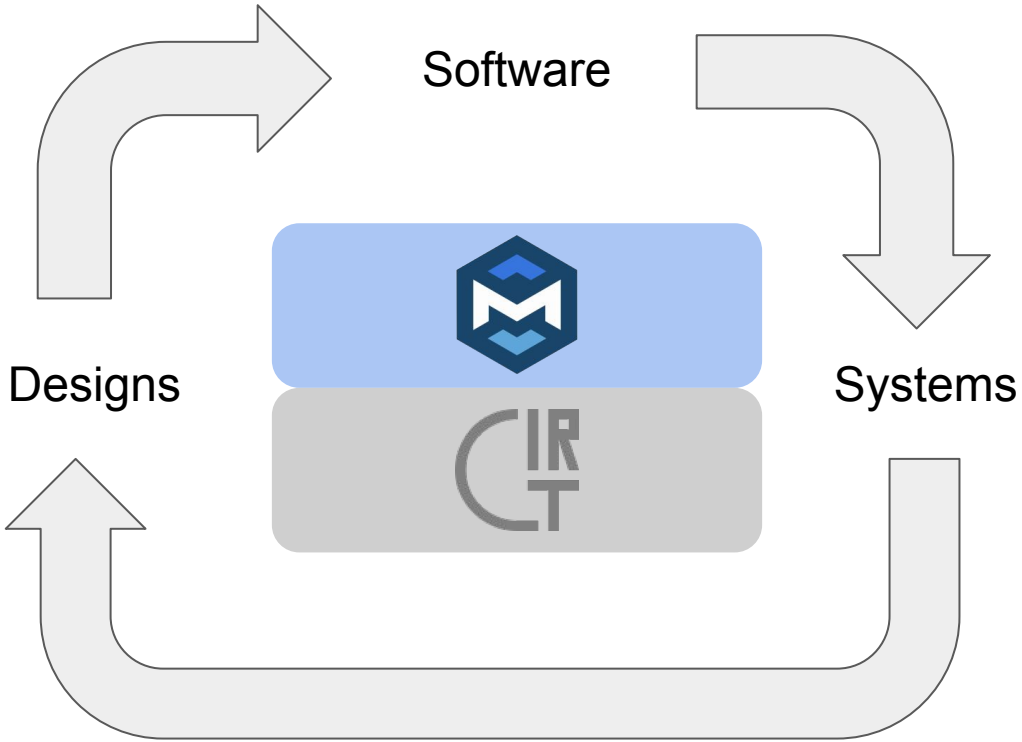


Image source: Mike Urbach. Applying Circuit IR Compilers and Tools (CIRCT) to ML Applications.

加入 CIRCT 社区！

- Website: circt.llvm.org
- GitHub: github.com/llvm/circt
- Forums: CIRCT on LLVM Discourse
- Chat: CIRCT on LLVM Discord

谢谢！

叶汉辰 - UIUC 伊利诺伊大学香槟分校

hanchenye@gmail.com

2021 年 6 月 20 日