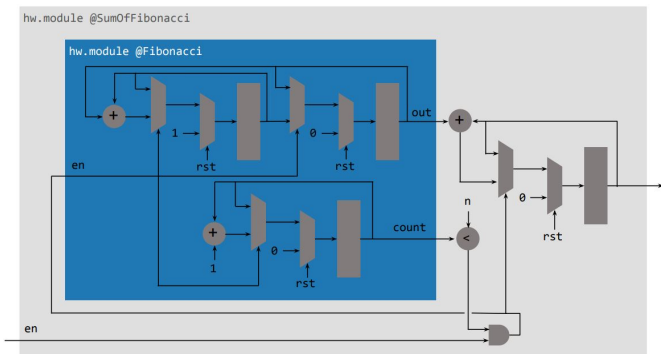


Arcilator

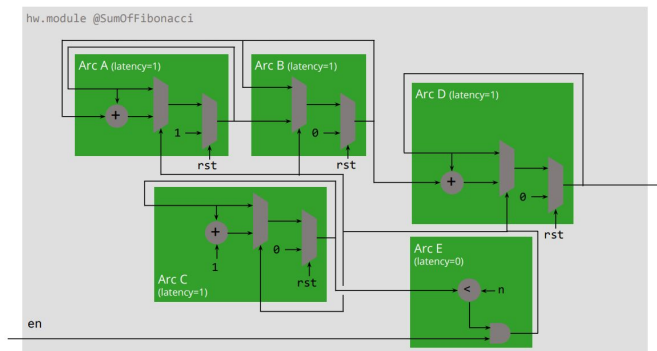
Fast and cycle-accurate hardware simulation in CIRCT

Martin Erhart, Fabian Schuiki, Zachary Yedidia, Bea Healy, Tobias Grosser

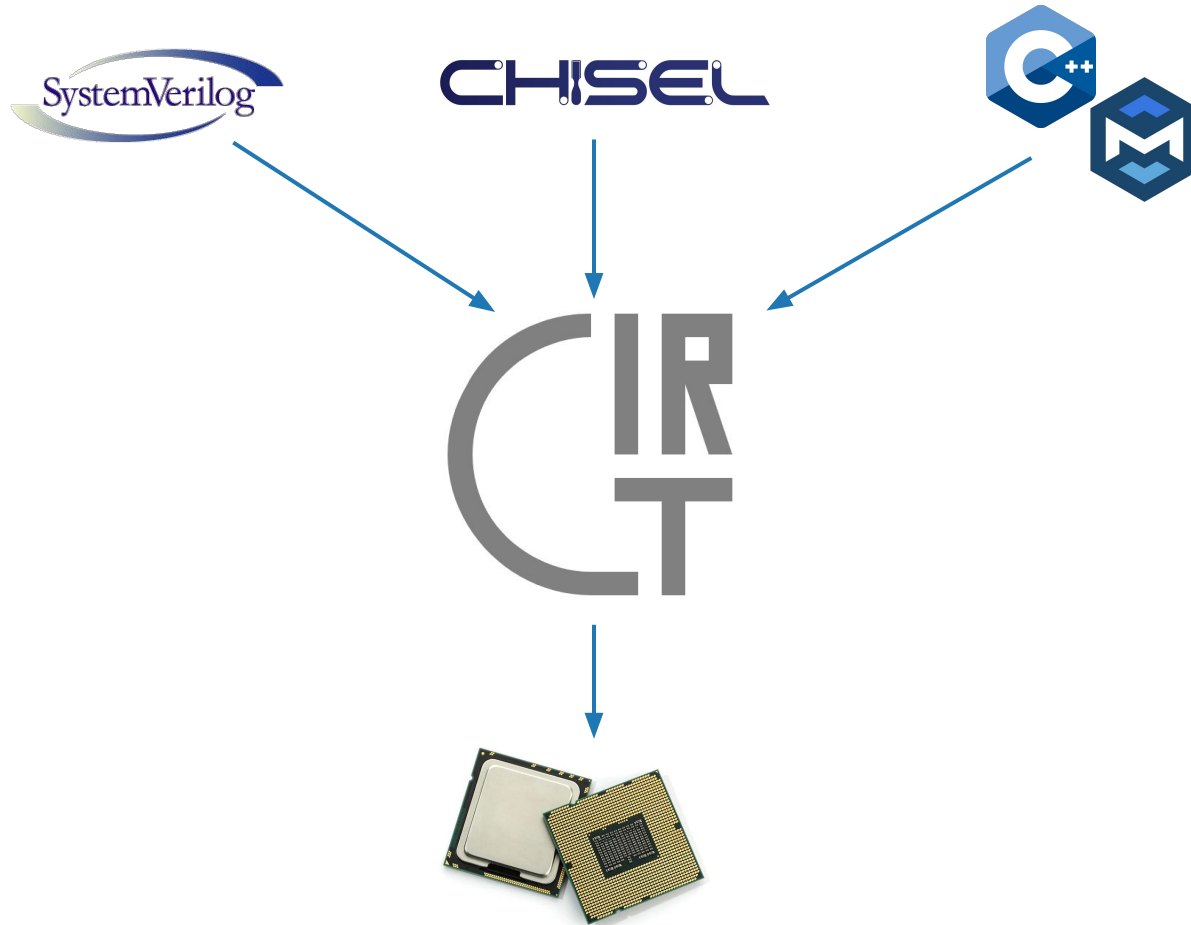
Thanks to Magnus Sjölander for supervising and the CIRCT Community!

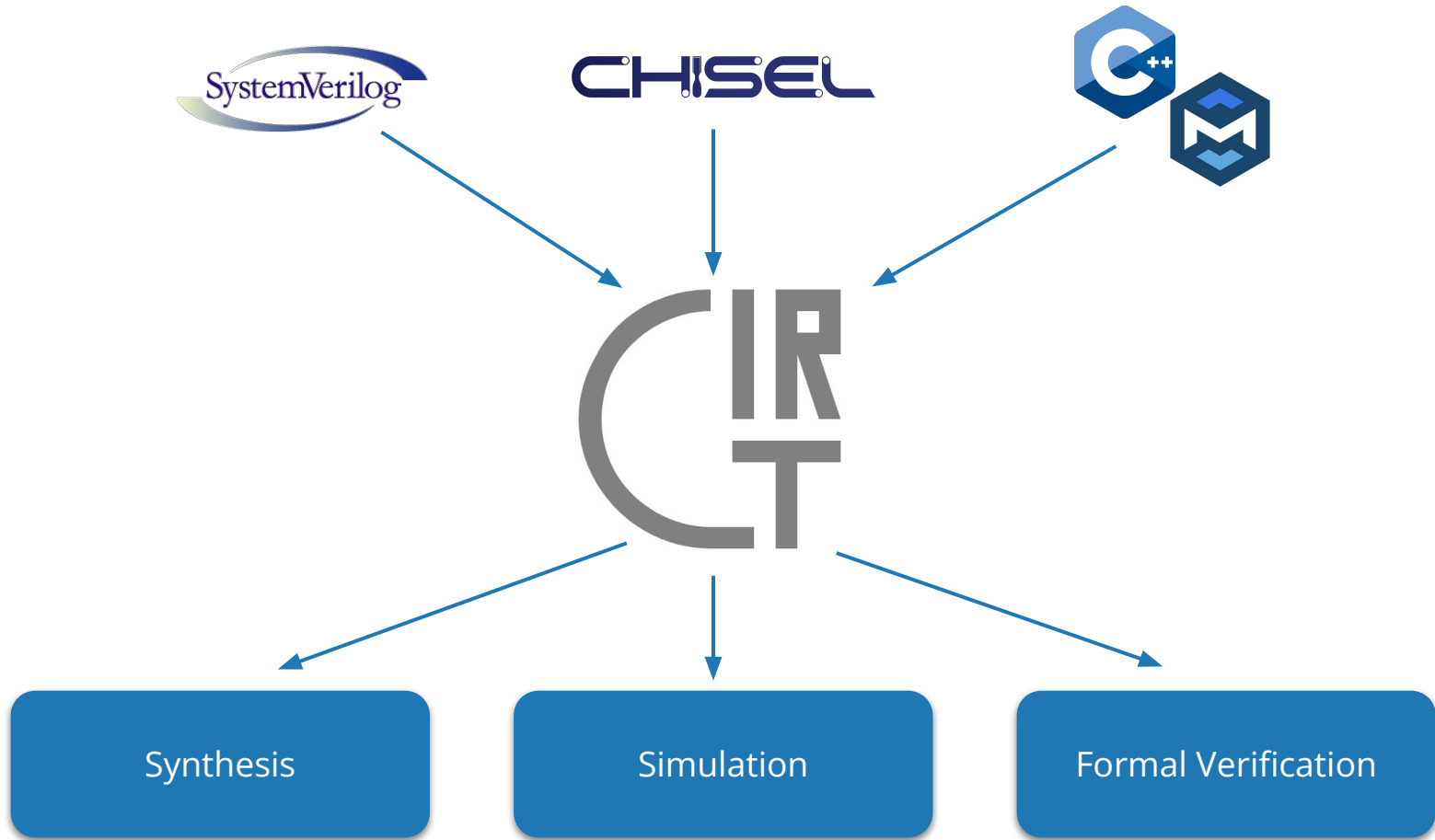


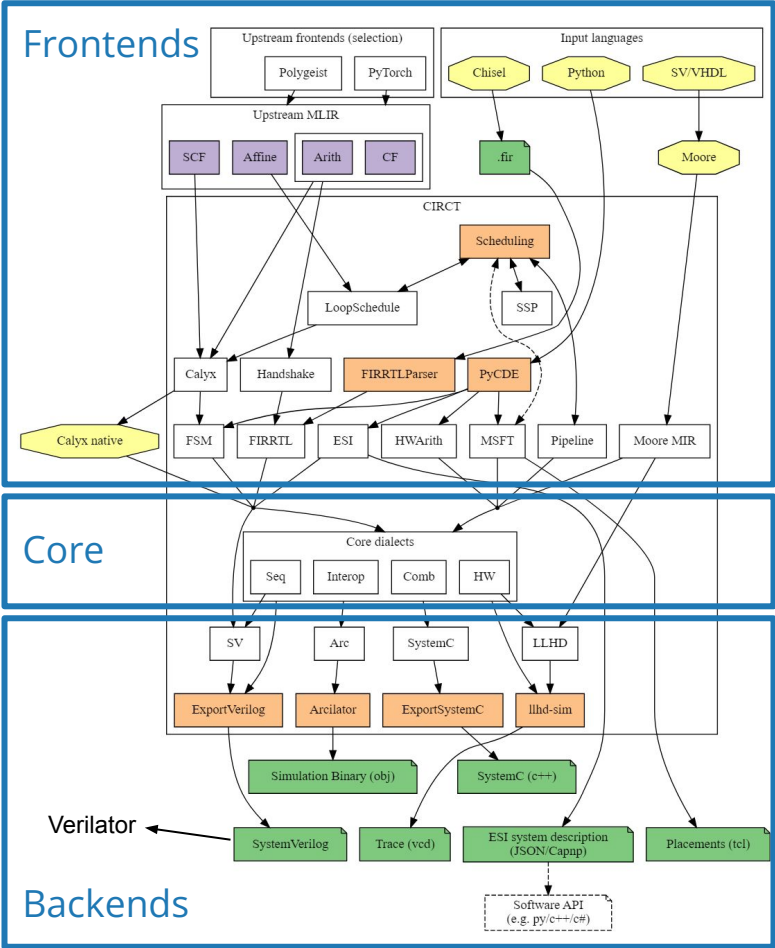
Human module hierarchy



Explicit register transfers







Verilator

README LGPL-3.0 license


Website Verilator.org License LGPL v3 License Artistic 2.0 distro packages 23 docker pulls 17k code quality A

codecov 89% build passing

Welcome to Verilator

Welcome to Verilator, the fastest Verilog/SystemVerilog simulator.

- Accepts Verilog or SystemVerilog
- Performs lint code-quality checks
- Compiles into multithreaded C++, or SystemC
- Creates XML to front-end your own tools





Fast

- Outperforms many closed-source commercial simulators
- Single- and multithreaded output models



Widely Used

- Wide industry and academic deployment
- Out-of-the-box support from Arm and RISC-V vendor IP



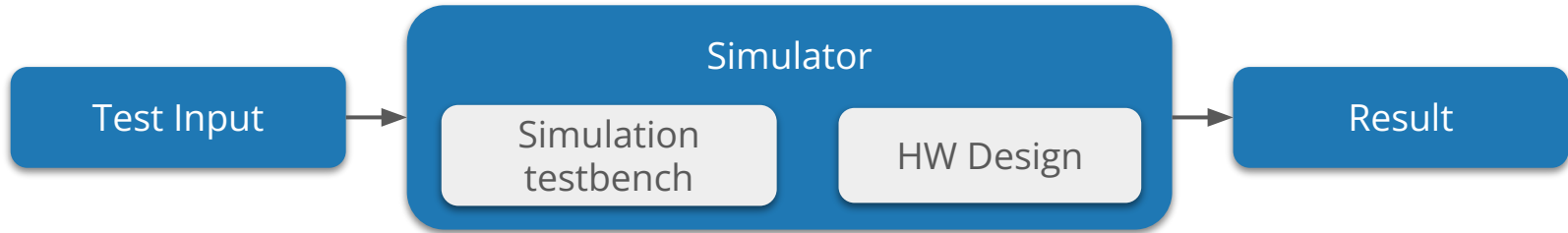
Community Driven & Openly Licensed

- Guided by the CHIPS Alliance and Linux Foundation
- Open and free as in both speech and beer

Why simulate a HW design?

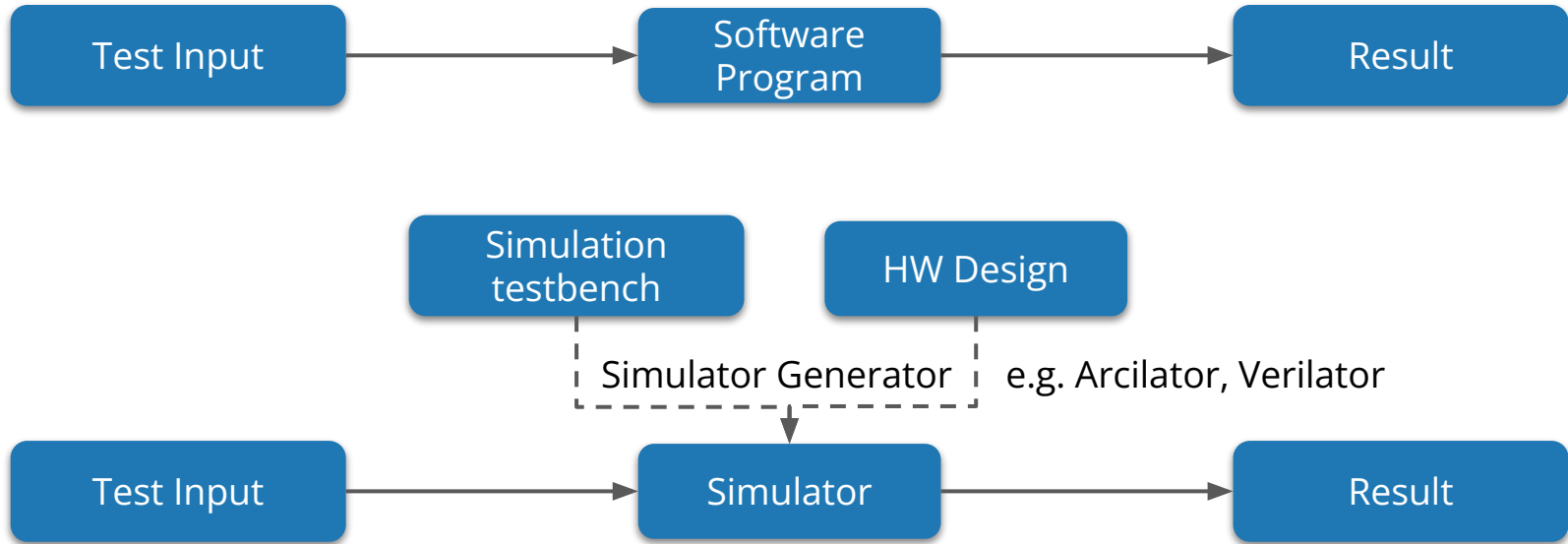


Hardware Simulation Overview

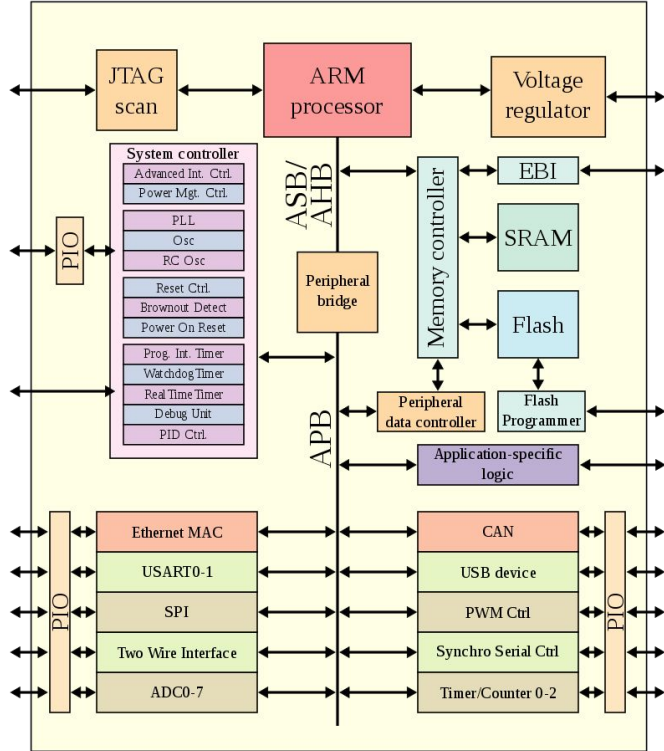
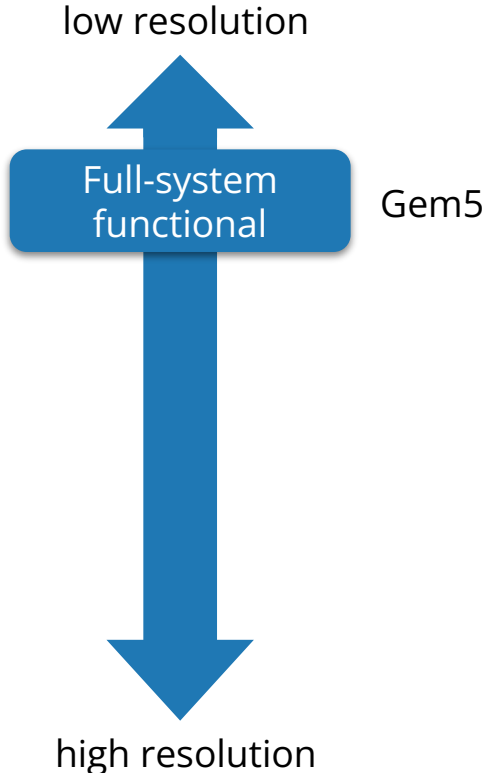


e.g. LLHD-Sim

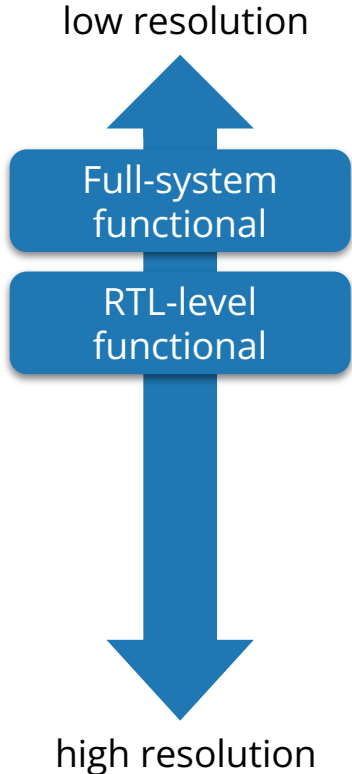
Hardware Simulation Overview



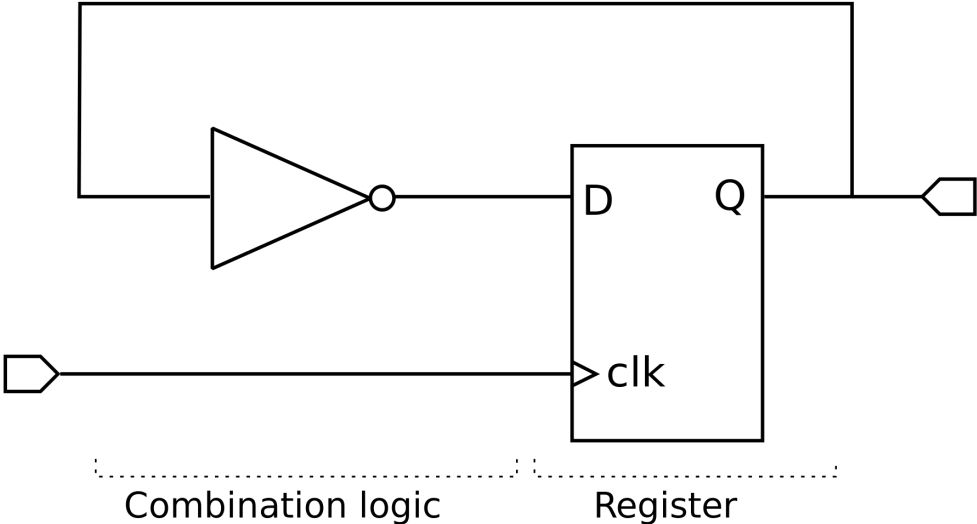
Hardware Simulation Overview



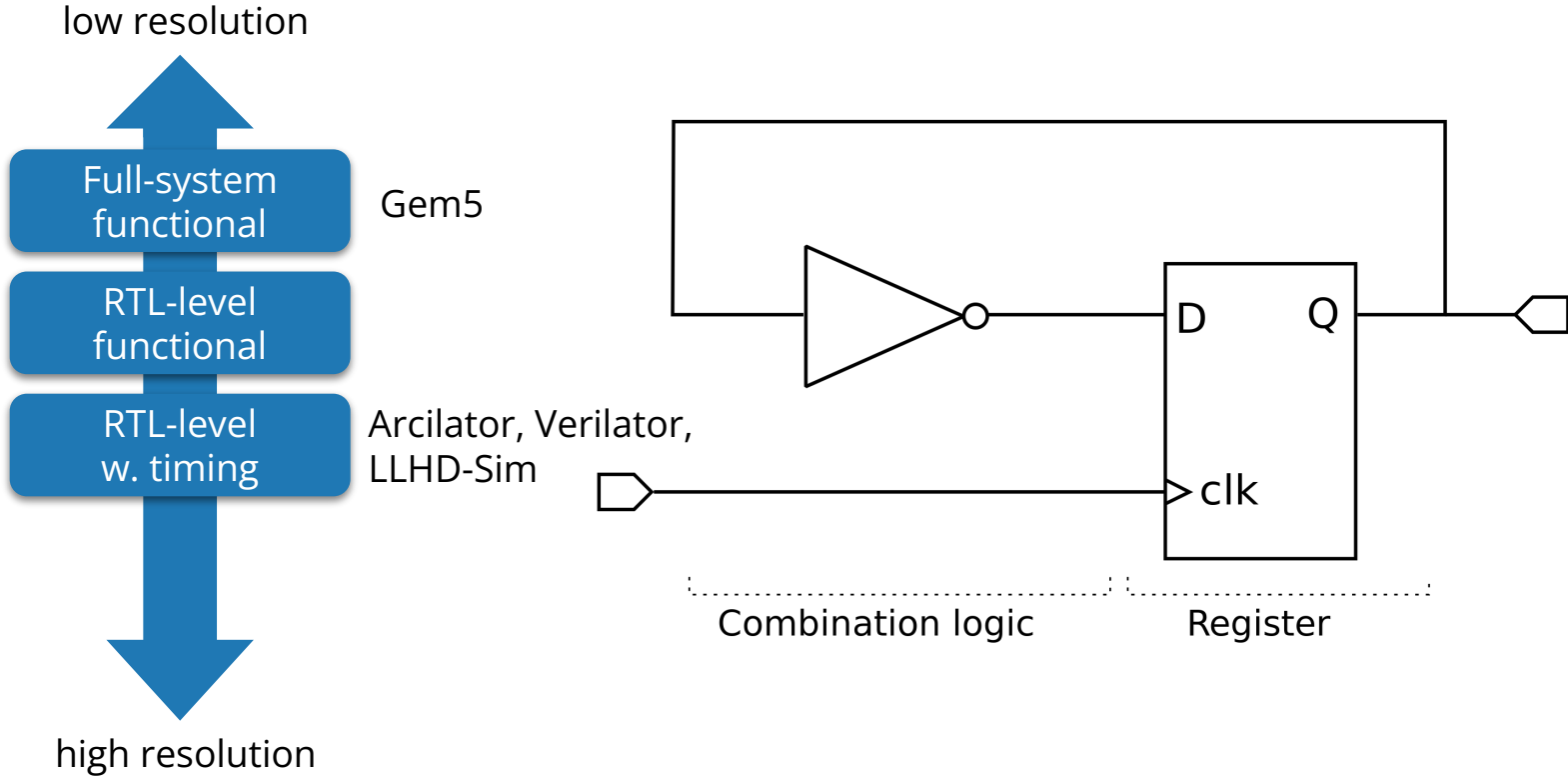
Hardware Simulation Overview



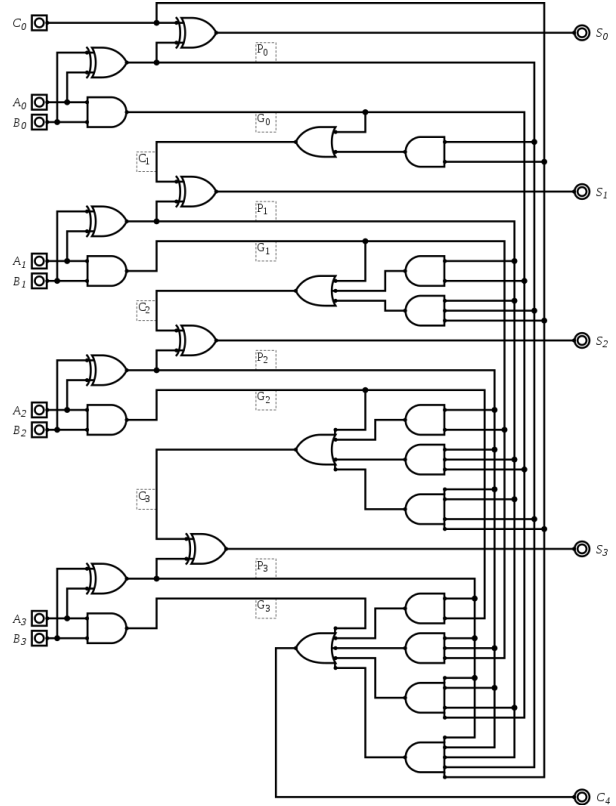
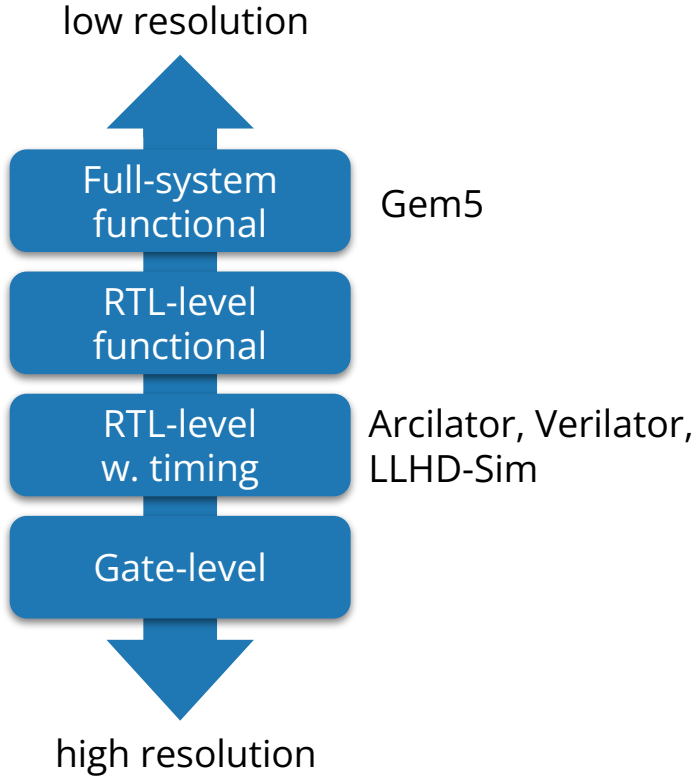
Gem5



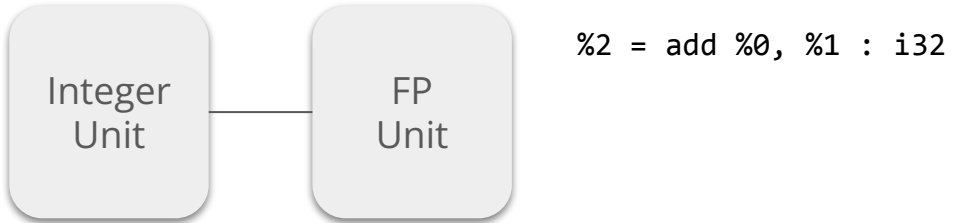
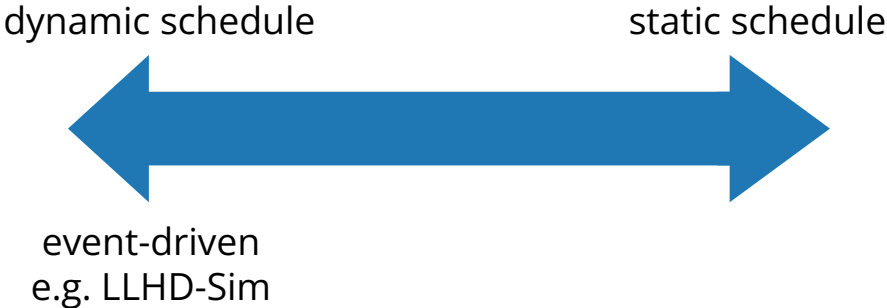
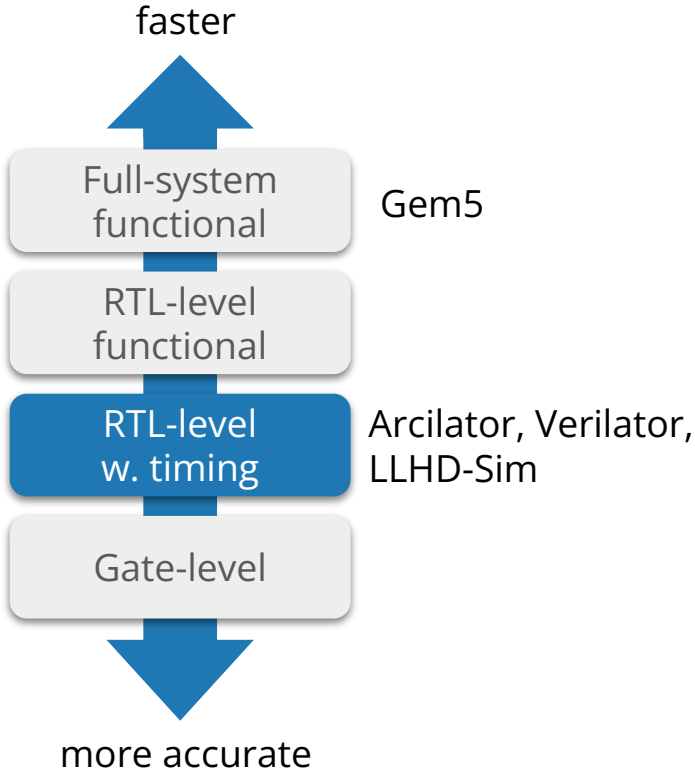
Hardware Simulation Overview



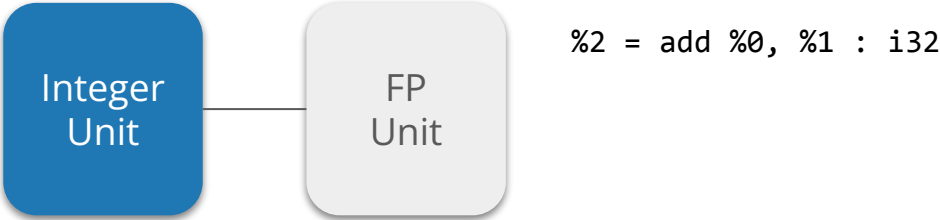
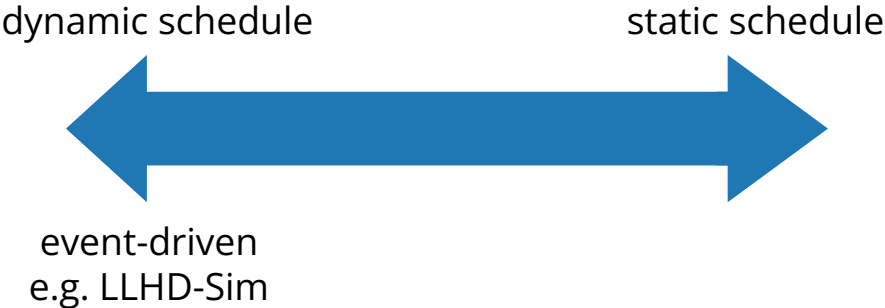
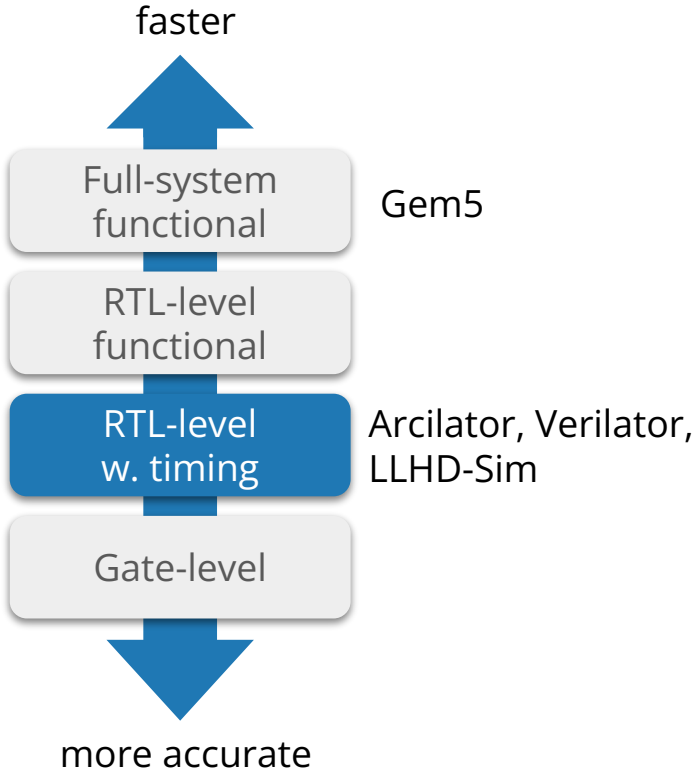
Hardware Simulation Overview



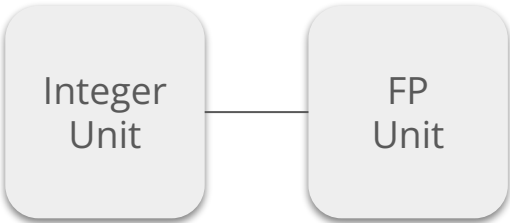
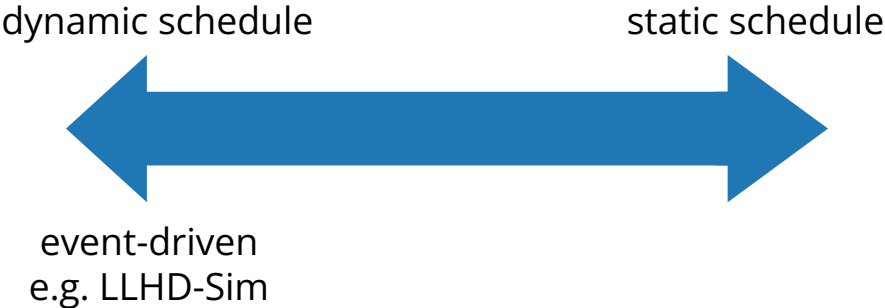
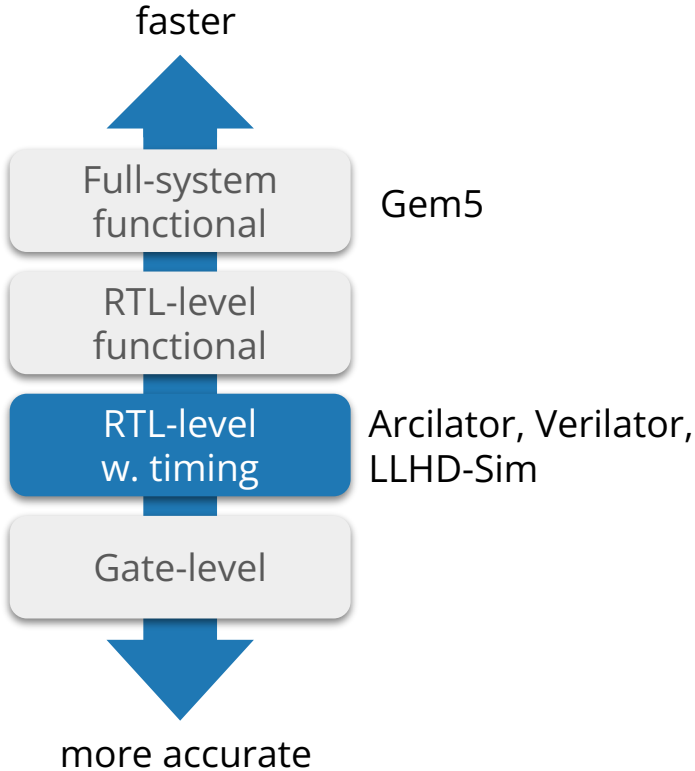
Hardware Simulation Overview



Hardware Simulation Overview

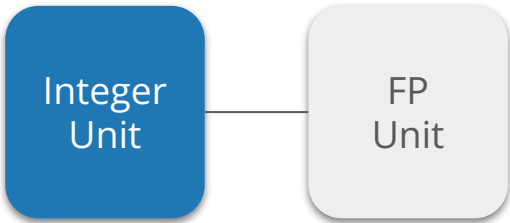
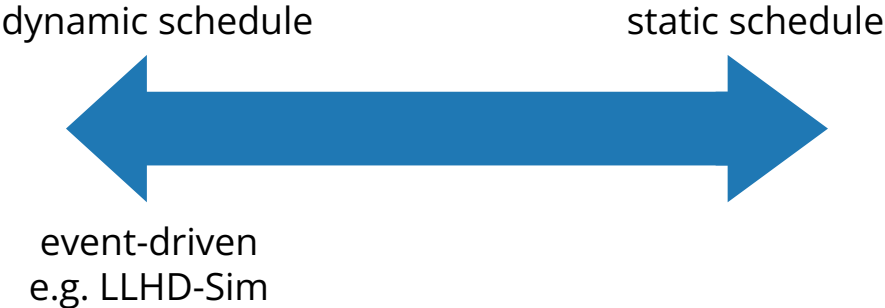
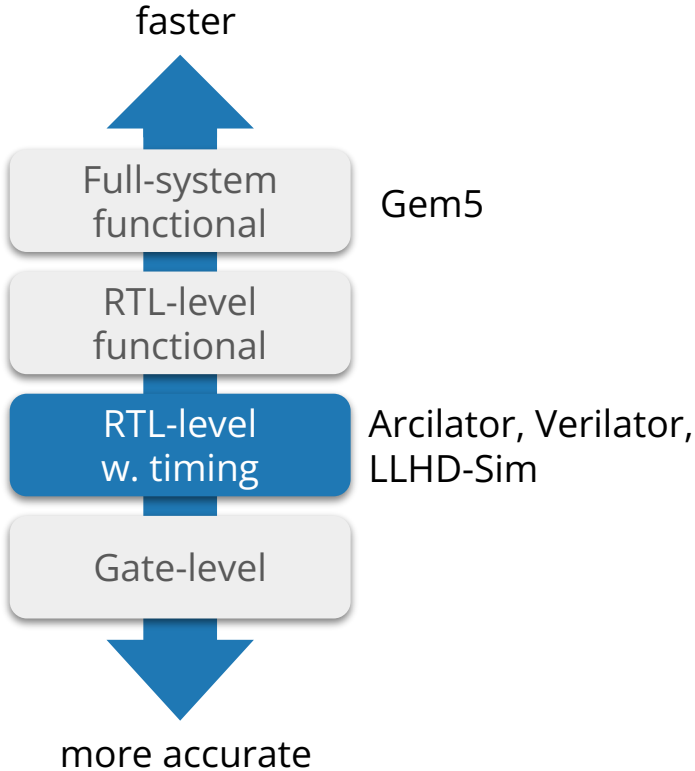


Hardware Simulation Overview



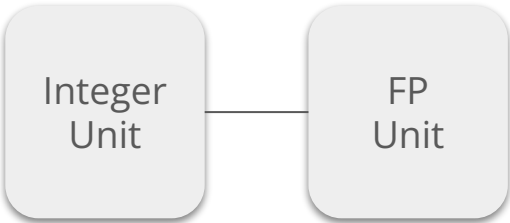
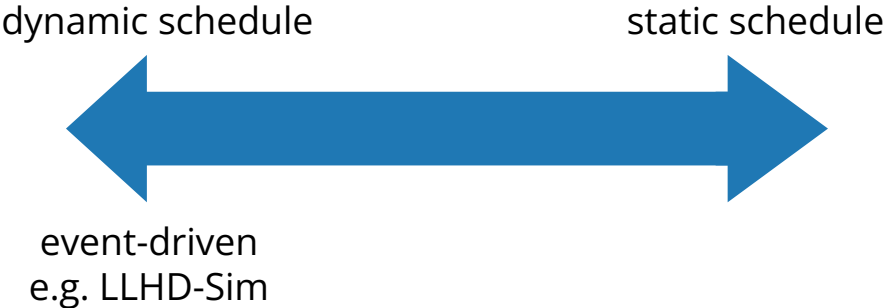
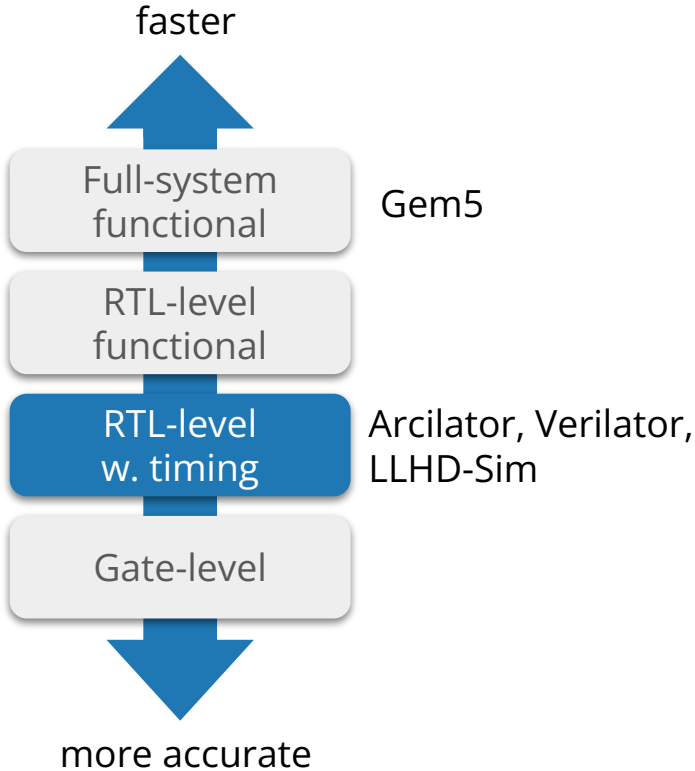
```
%2 = add %0, %1 : i32  
%3 = mul %0, %2 : i32
```

Hardware Simulation Overview



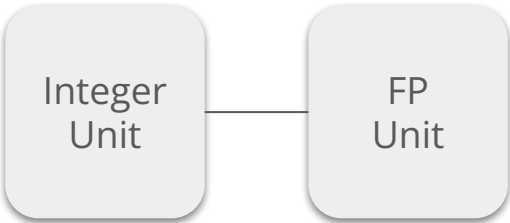
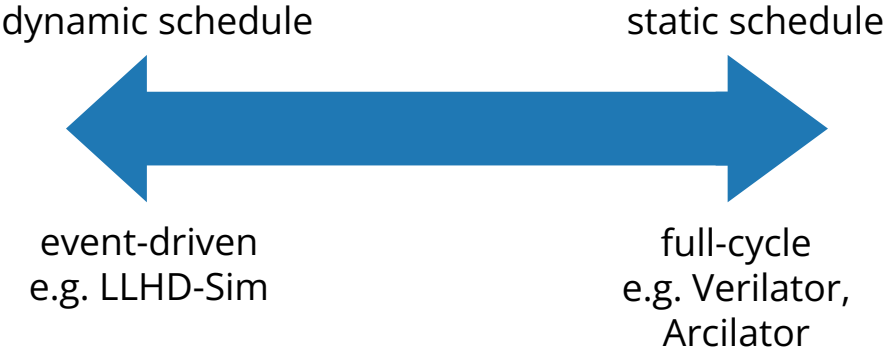
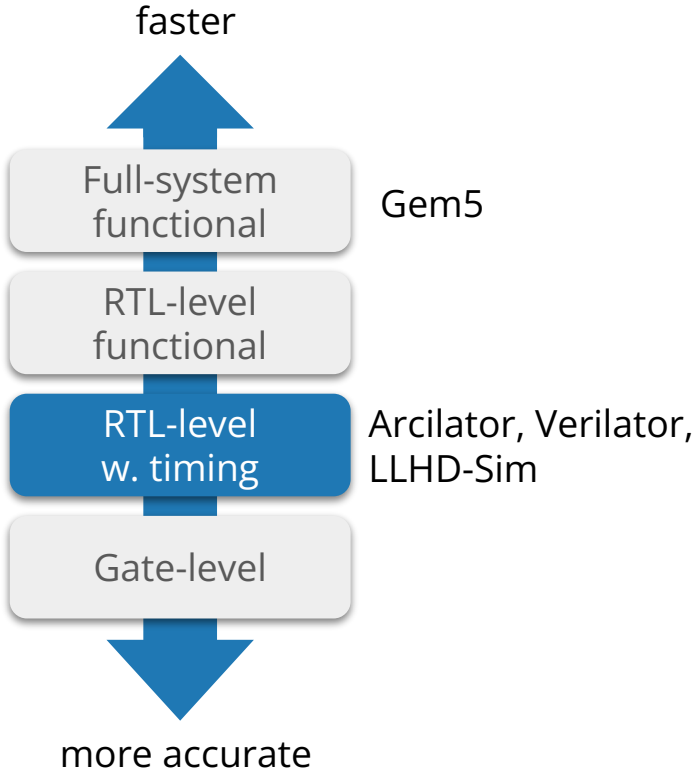
```
%2 = add %0, %1 : i32  
%3 = mul %0, %2 : i32
```


Hardware Simulation Overview



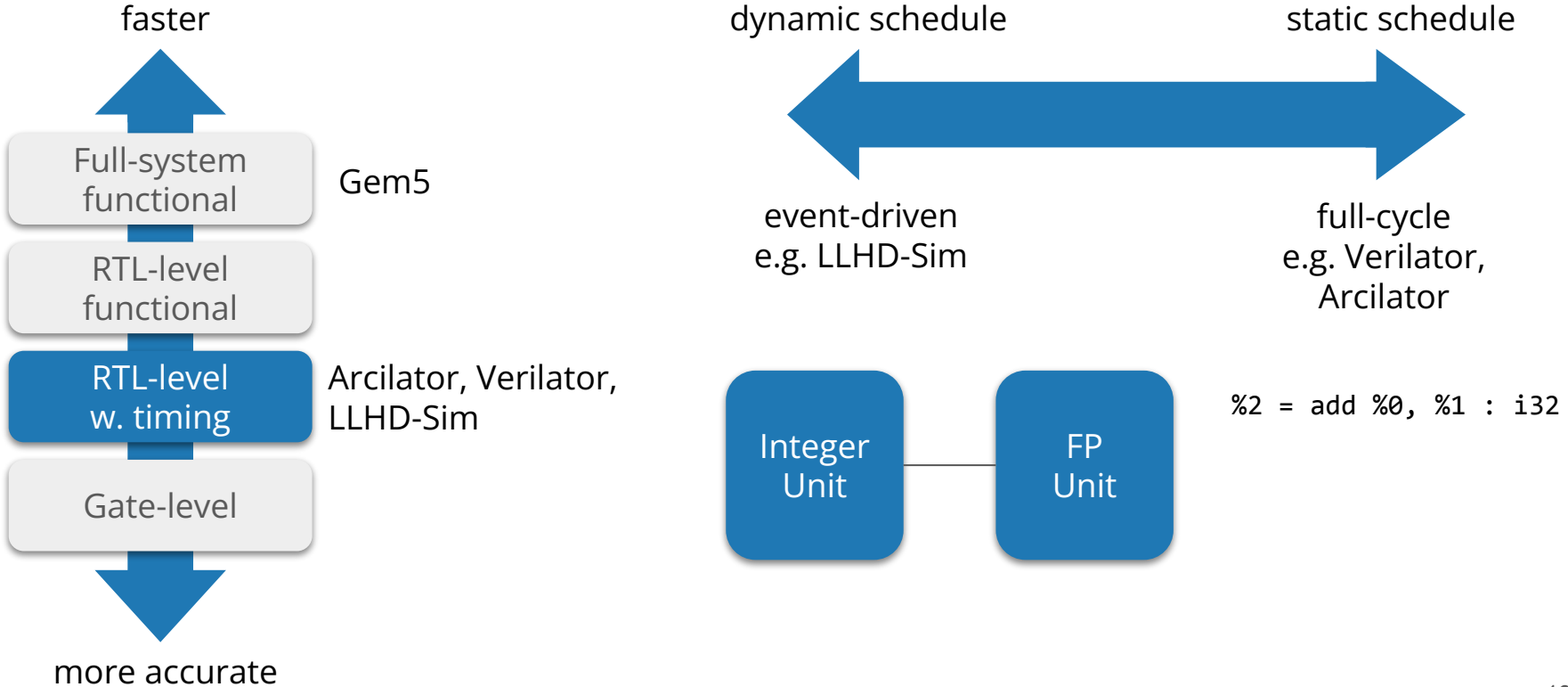
```
%2 = add %0, %1 : i32  
%3 = mul %0, %2 : i32
```

Hardware Simulation Overview

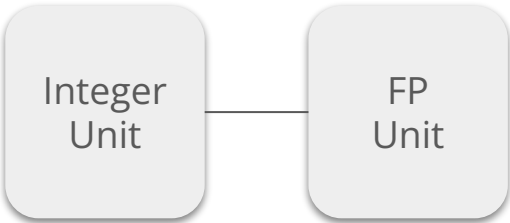
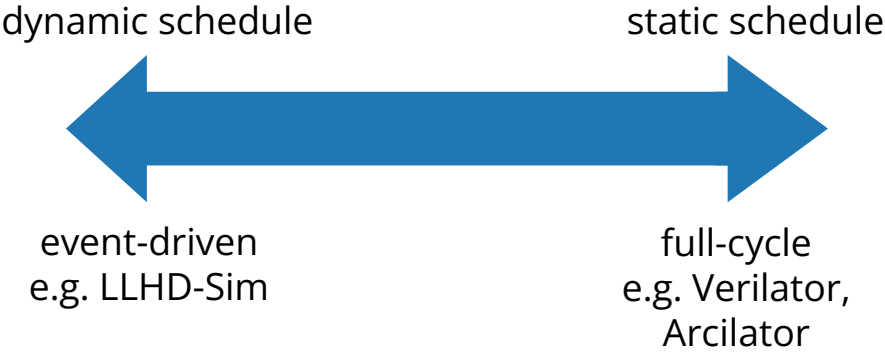
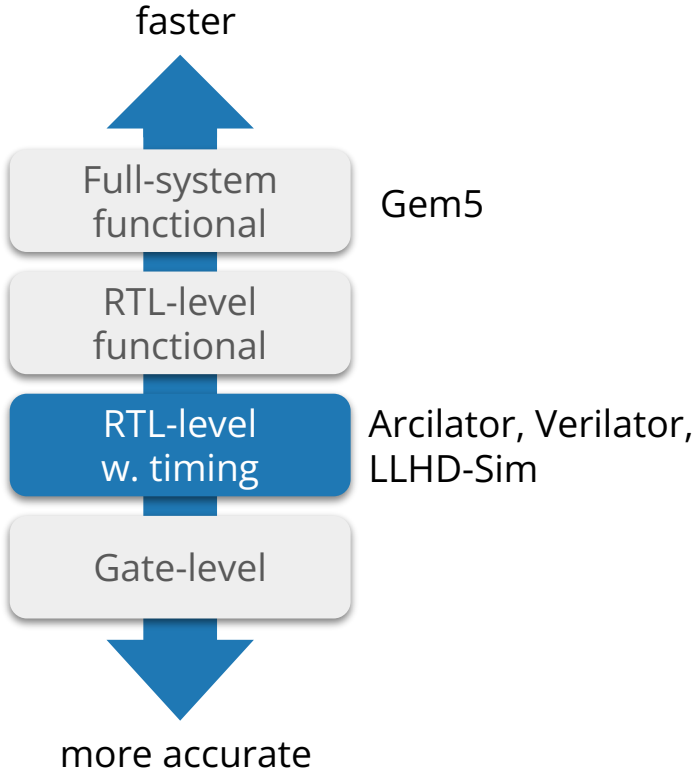


```
%2 = add %0, %1 : i32
```

Hardware Simulation Overview

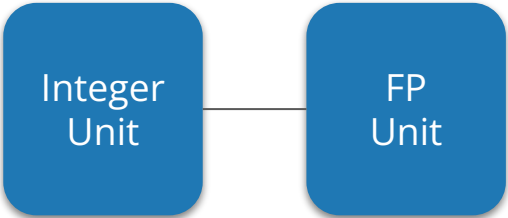
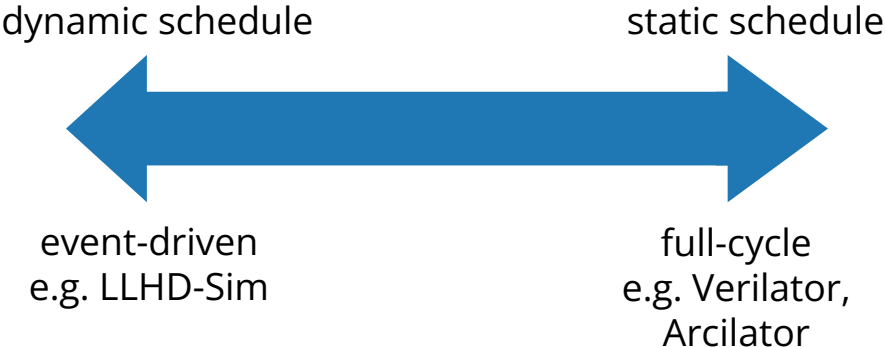
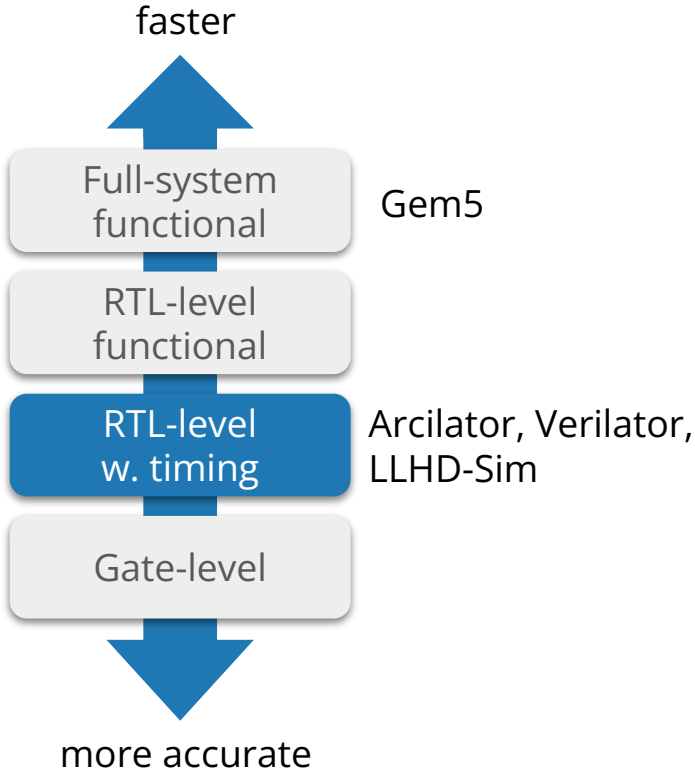


Hardware Simulation Overview



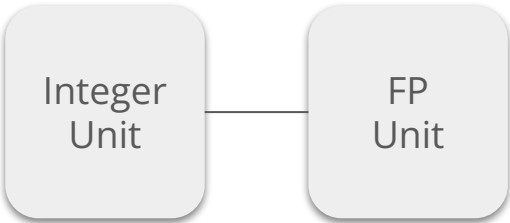
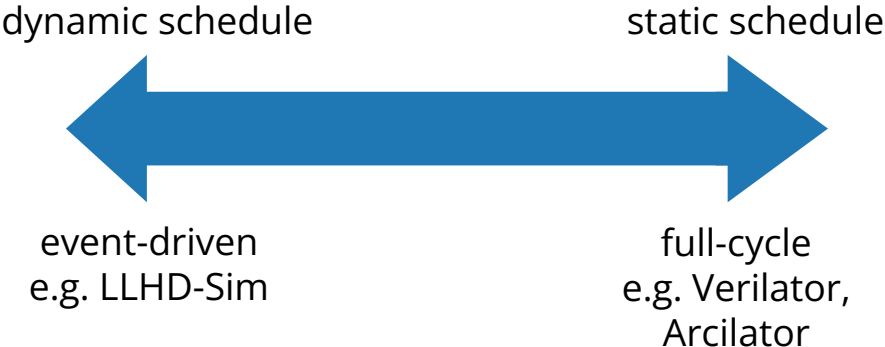
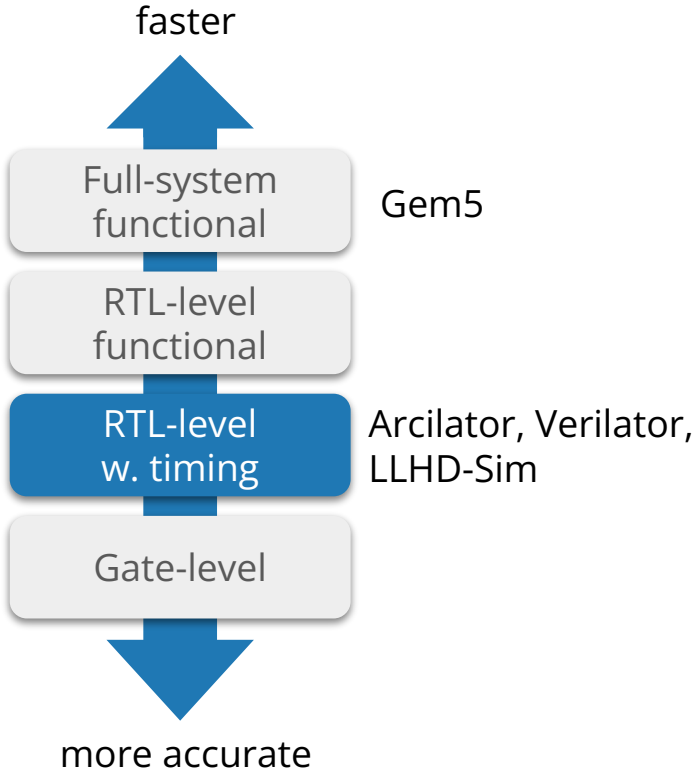
```
%2 = add %0, %1 : i32  
%3 = mul %0, %2 : i32
```

Hardware Simulation Overview



```
%2 = add %0, %1 : i32  
%3 = mul %0, %2 : i32
```

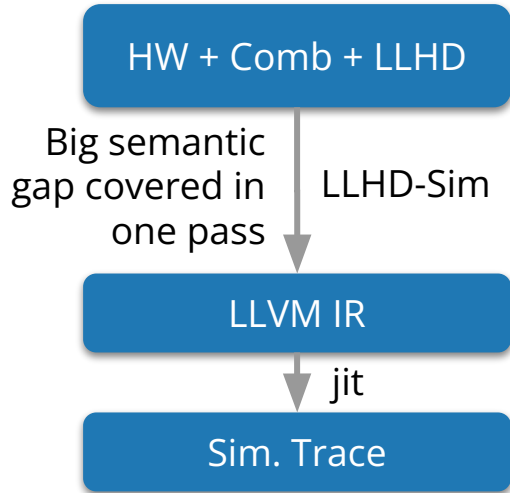
Hardware Simulation Overview



```
%2 = add %0, %1 : i32  
%3 = mul %0, %2 : i32
```

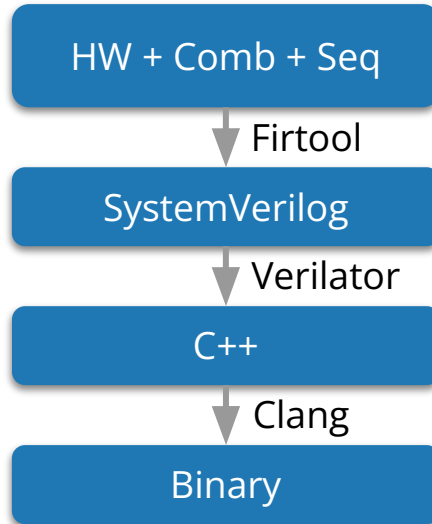
Hardware Simulator Overview

LLHD-Sim



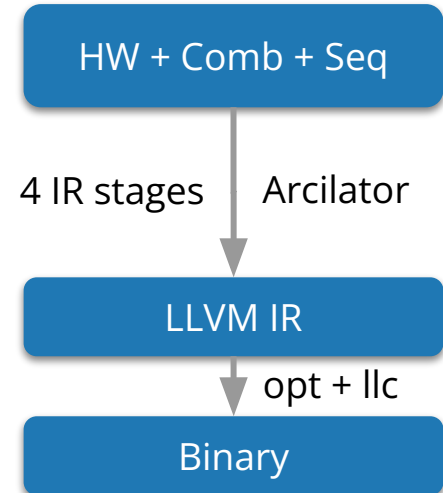
Aims for full SV-spec. support

Verilator



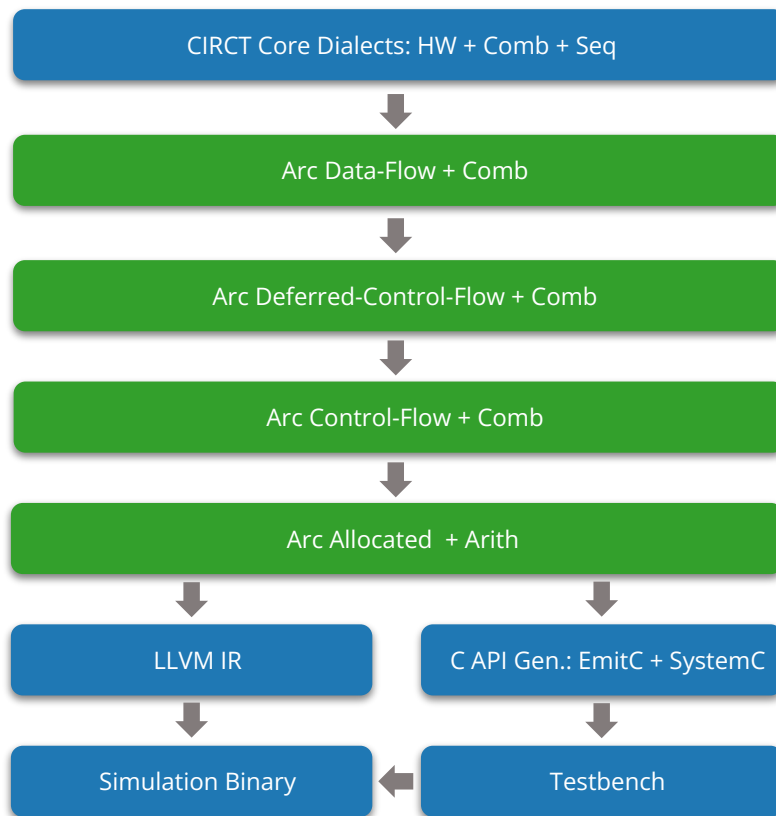
Two complicated, high-level languages as IRs

Arcilator

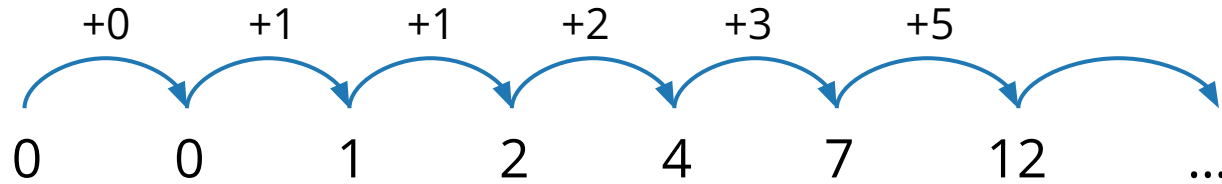


Only support CIRCT Core dialects, aim for performance

Four new intermediate levels



Running Example - Sum of Fibonacci Numbers



$$\begin{aligned}F_0 &= 0 \\F_1 &= 1 \\F_n &= F_{n-2} + F_{n-1}\end{aligned}$$

$$\sum_{i=0}^{n-1} F_i$$

CIRCT Core Representation

```
hw.module @SumOfFibonacci(  
  in %clock : !seq.clock,  
  in %rst : i1,  
  in %en : i1,  
  out out : i32)  
{  
  %c0_i32 = hw.constant 0 : i32  
  %c12_i32 = hw.constant 12 : i32  
  %0 = comb.mux %rst, %c0_i32, %4 : i32  
  %reg = seq.compreg %0, %clock : i32  
  %fib.out, %fib.count = hw.instance "fib" @Fibonacci(  
    clock: %clock: !seq.clock, rst: %rst: i1, en: %2: i1)  
    -> (out: i32, count: i32)  
  %1 = comb.icmp ult %fib.count, %c12_i32 : i32  
  %2 = comb.and %en, %1 : i1  
  %3 = comb.add %reg, %fib.out : i32  
  %4 = comb.mux %2, %3, %reg : i32  
  hw.output %reg : i32  
}
```

Instantiation

Combinational
logic

```
hw.module private @Fibonacci(  
  in %clock : !seq.clock,  
  in %rst : i1,  
  in %en : i1,  
  out out : i32,  
  out count : i32)  
{  
  %c1_i32 = hw.constant 1 : i32  
  %c0_i32 = hw.constant 0 : i32  
  %0 = comb.mux %rst, %c0_i32, %5 : i32  
  %reg_0 = seq.compreg %0, %clock : i32  
  %1 = comb.mux %rst, %c1_i32, %3 : i32  
  %reg_1 = seq.compreg %1, %clock : i32  
  %2 = comb.mux %rst, %c0_i32, %7 : i32  
  %counter = seq.compreg %2, %clock : i32  
  %3 = comb.mux %en, %reg_0, %reg_1 : i32  
  %4 = comb.add %reg_0, %reg_1 : i32  
  %5 = comb.mux %en, %4, %reg_0 : i32  
  %6 = comb.add %counter, %c1_i32 : i32  
  %7 = comb.mux %en, %6, %counter : i32  
  hw.output %reg_1, %counter : i32, i32  
}
```

Registers

CIRCT Core Dialects

Arc Data-Flow
Comb

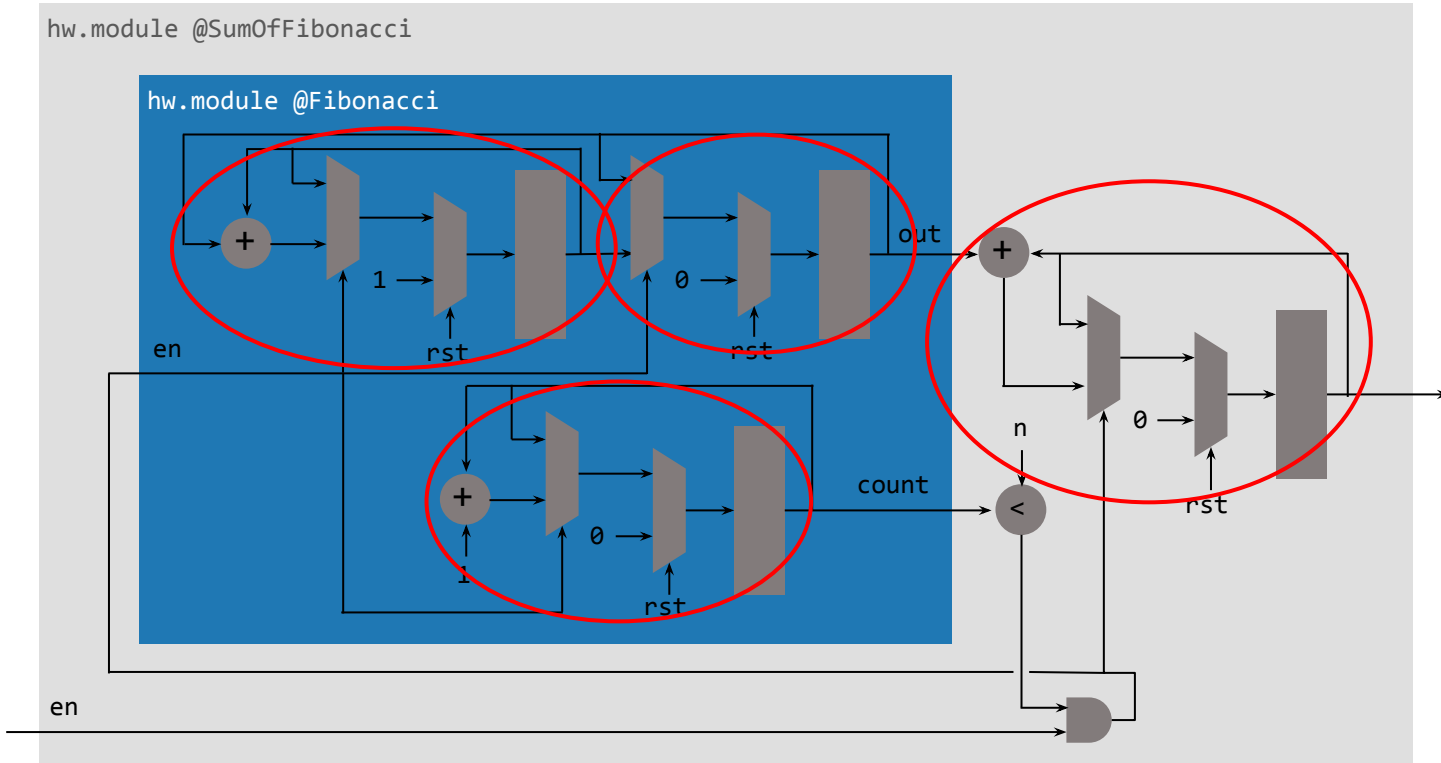
Arc Deferred-CF
Comb

Arc Control-Flow
Comb

Arc Allocated
Arith

LLVM IR

Block Diagram of CIRCT Core Representation



CIRCT Core Dialects

Arc Data-Flow Comb

Arc Deferred-CF Comb

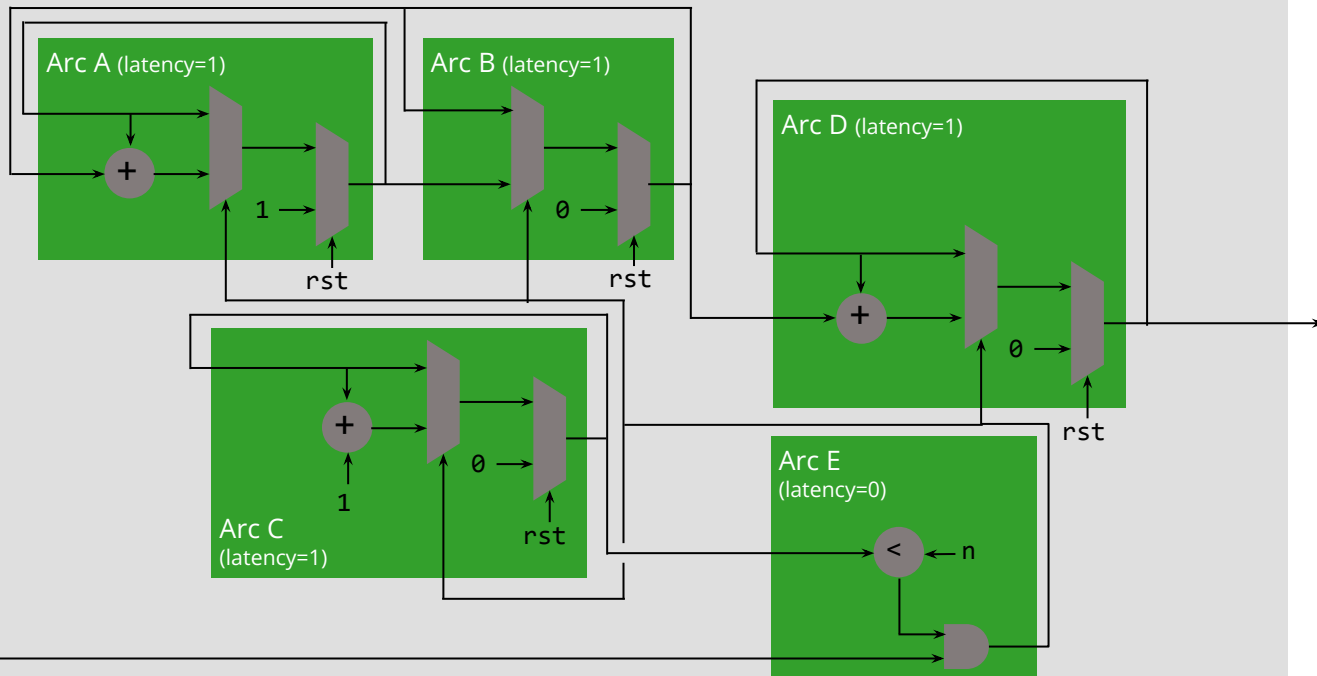
Arc Control-Flow Comb

Arc Allocated Arith

LLVM IR

Arc Data-Flow representation

hw.module @SumOfFibonacci



CIRCT Core Dialects

Arc Data-Flow Comb

Arc Deferred-CF Comb

Arc Control-Flow Comb

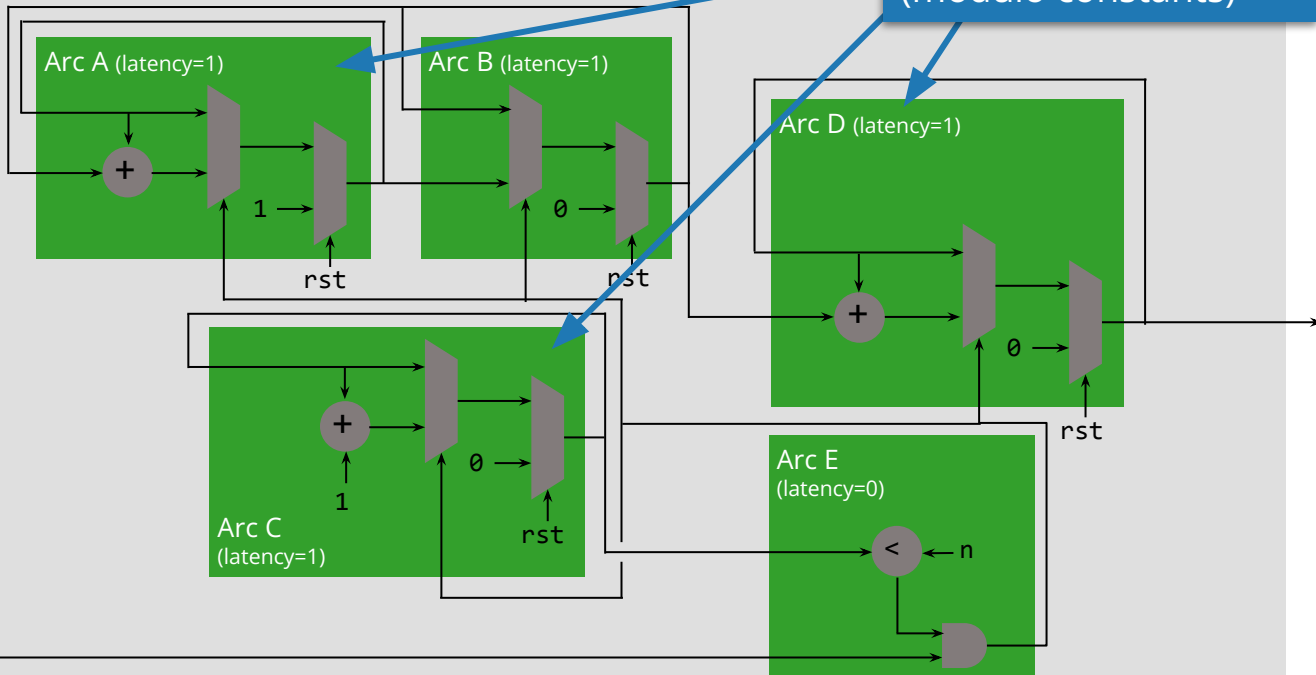
Arc Allocated Arith

LLVM IR

Arc Deduplication

hw.module @SumOfFibonacci

Equivalent
(modulo constants)



CIRCT Core Dialects

Arc Data-Flow
Comb

Arc Deferred-CF
Comb

Arc Control-Flow
Comb

Arc Allocated
Arith

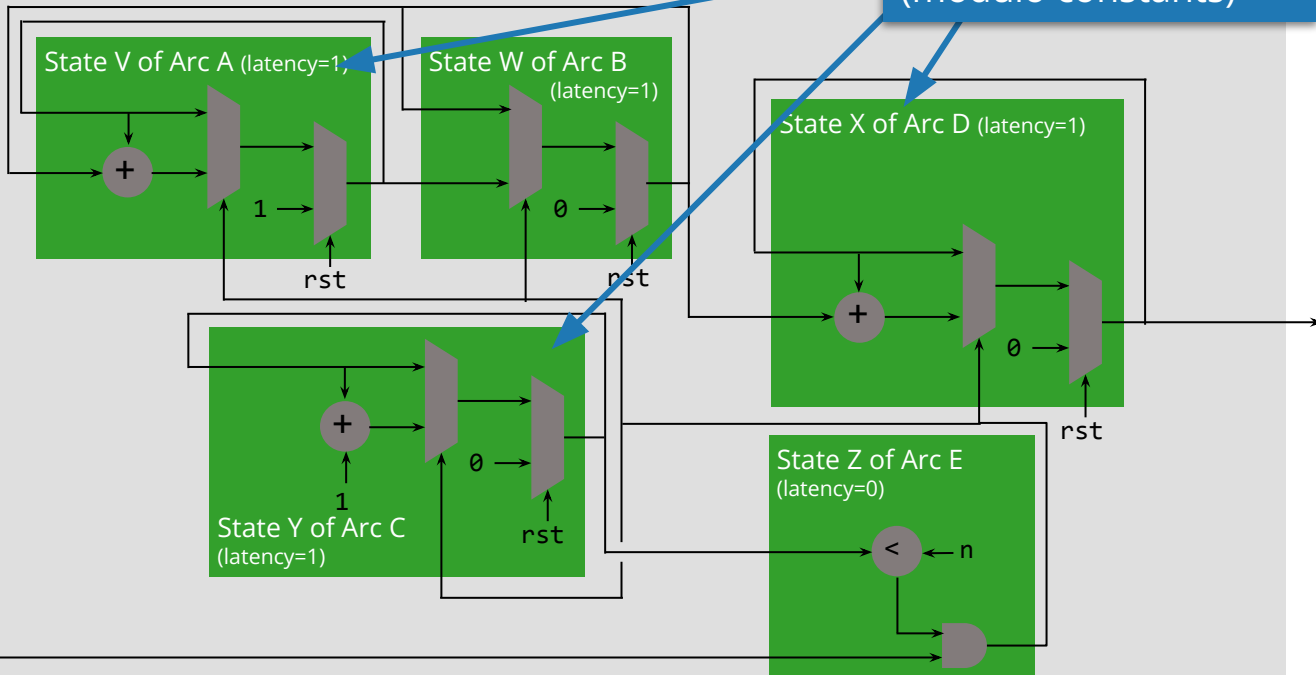
LLVM IR

en

Arc Deduplication

hw.module @SumOfFibonacci

Equivalent
(modulo constants)



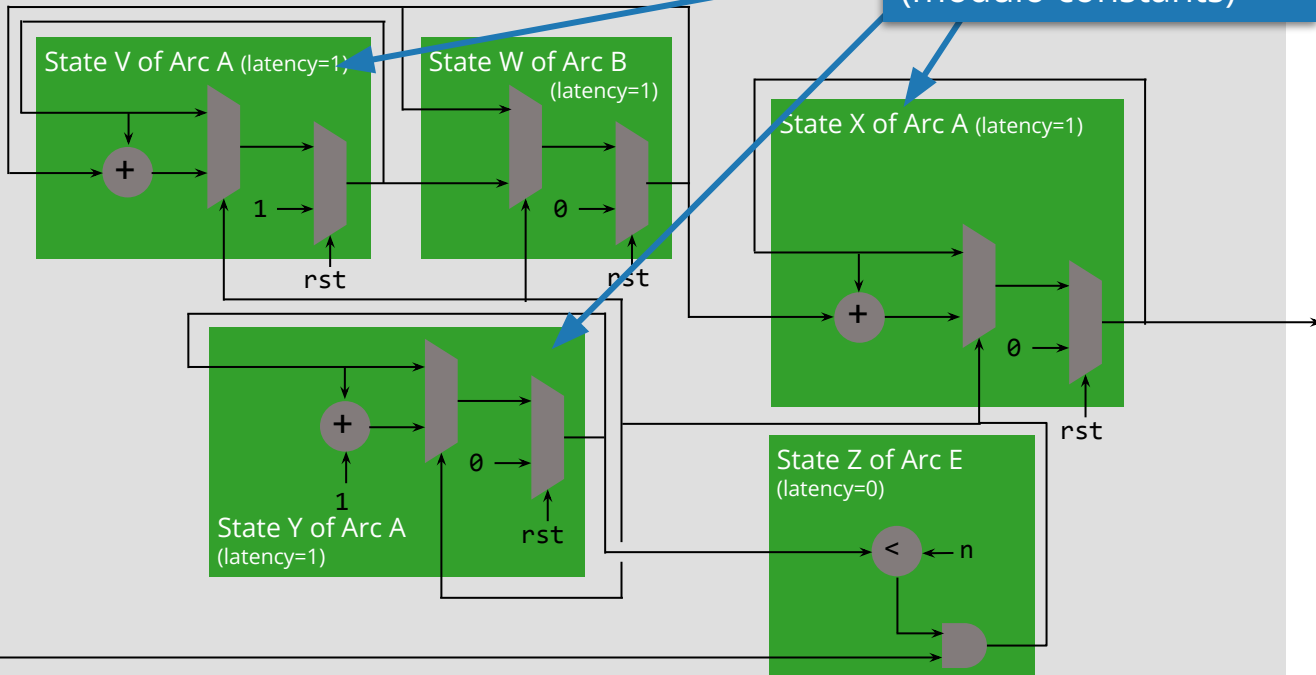
en

- CIRCT Core Dialects
- Arc Data-Flow Comb
- Arc Deferred-CF Comb
- Arc Control-Flow Comb
- Arc Allocated Arith
- LLVM IR

Arc Deduplication

hw.module @SumOfFibonacci

Equivalent
(modulo constants)



en

- CIRCT Core Dialects
- Arc Data-Flow Comb
- Arc Deferred-CF Comb
- Arc Control-Flow Comb
- Arc Allocated Arith
- LLVM IR

Arc Data-Flow Representation

```
arc.define @LFSR_arc(%arg0: i32, %arg1: i32, %arg2: i1, %arg3: i1) -> i32 {
  %c0_i32 = hw.constant 0 : i32
  %0 = comb.add %arg0, %arg1 : i32
  %1 = comb.mux %arg2, %0, %arg0 : i32
  %2 = comb.mux %arg3, %c0_i32, %1 : i32
  arc.output %2 : i32
}
arc.define @LFSR_arc_0(%arg0: i1, %arg1: i32, %arg2: i32, %arg3: i1) -> i32 {
  %c1_i32 = hw.constant 1 : i32
  %0 = comb.mux %arg0, %arg1, %arg2 : i32
  %1 = comb.mux %arg3, %c1_i32, %0 : i32
  arc.output %1 : i32
}
arc.define @SumOfFibonacci_arc_0(%arg0: i32, %arg1: i1) -> i1 {
  %c12_i32 = hw.constant 12 : i32
  %0 = comb.icmp ult %arg0, %c12_i32 : i32
  %1 = comb.and %arg1, %0 : i1
  arc.output %1 : i1
}
}
hw.module @SumOfFibonacci(in %clock : !seq.clock, in %reset : i1, in %enabled : i1, out out : i32) {
  %c1_i32 = hw.constant 1 : i32
  %0 = arc.state @LFSR_arc(%0, %1, %4, %reset) clock %clock latency 1 : (i32, i32, i1, i1) -> i32
  %1 = arc.state @LFSR_arc_0(%4, %0, %1, %reset) clock %clock latency 1 : (i1, i32, i32, i1) -> i32
  %2 = arc.state @LFSR_arc(%2, %c1_i32, %4, %reset) clock %clock latency 1 : (i32, i32, i1, i1) -> i32
  %3 = arc.state @LFSR_arc(%3, %1, %4, %reset) clock %clock latency 1 : (i32, i32, i1, i1) -> i32
  %4 = arc.state @SumOfFibonacci_arc_0(%2, %enabled) latency 0 : (i32, i1) -> i1
  hw.output %3 : i32
}
}
```

CIRCT Core Dialects

Arc Data-Flow
Comb

Arc Deferred-CF
Comb

Arc Control-Flow
Comb

Arc Allocated
Arith

LLVM IR

Arc Data-Flow Optimizations

Dedup



CIRCT Core Dialects

**Arc Data-Flow
Comb**

Arc Deferred-CF
Comb

Arc Control-Flow
Comb

Arc Allocated
Arith

LLVM IR

Arc Data-Flow Optimizations

Dedup

Lookup tables

```
%res = comb.and %0, %1 : i1
```



```
%idx = comb.concat %c0_i2, %0, %1 : i2, i1, i1  
%tbl = hw.constant 0b1000 : i4  
%res = comb.shr %tbl, %idx : i4
```

CIRCT Core Dialects

Arc Data-Flow
Comb

Arc Deferred-CF
Comb

Arc Control-Flow
Comb

Arc Allocated
Arith

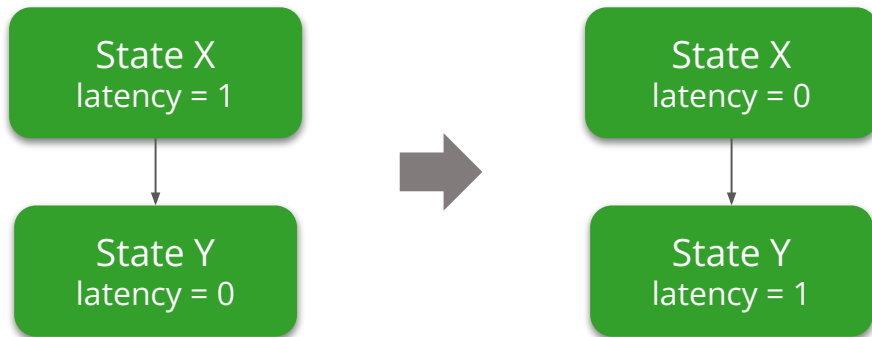
LLVM IR

Arc Data-Flow Optimizations

Dedup

Lookup tables

Register/latency retiming



CIRCT Core Dialects

Arc Data-Flow
Comb

Arc Deferred-CF
Comb

Arc Control-Flow
Comb

Arc Allocated
Arith

LLVM IR

Arc Data-Flow Optimizations

Dedup

Lookup tables

Register/latency retiming

Loop re-rolling

```
%1 = comb.mul %0, 2
%2 = comb.mul %1, 2
%3 = comb.mul %2, 2
%4 = comb.mul %3, 2
```



```
%1 = scf.for %i = %c0 to %c4 step %c1 iter_args(%arg0 = %0) {
  %2 = comb.mul %arg0, 2
  scf.yield %2
}
```

CIRCT Core Dialects

Arc Data-Flow
Comb

Arc Deferred-CF
Comb

Arc Control-Flow
Comb

Arc Allocated
Arith

LLVM IR

Arc Data-Flow Optimizations

Dedup

Lookup tables

Register/latency retiming

Loop re-rolling

Arc vectorization

CIRCT Core Dialects

**Arc Data-Flow
Comb**

Arc Deferred-CF
Comb

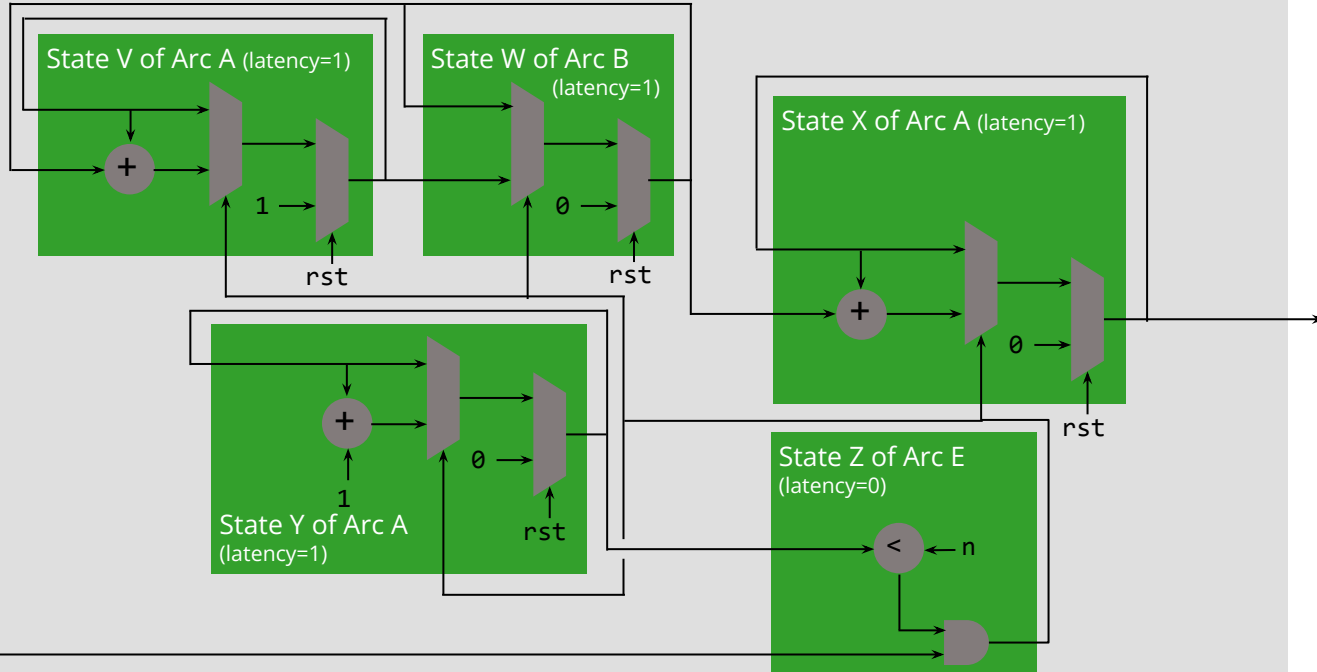
Arc Control-Flow
Comb

Arc Allocated
Arith

LLVM IR

Arc Data-Flow Representation

hw.module @SumOfFibonacci



CIRCT Core Dialects

Arc Data-Flow Comb

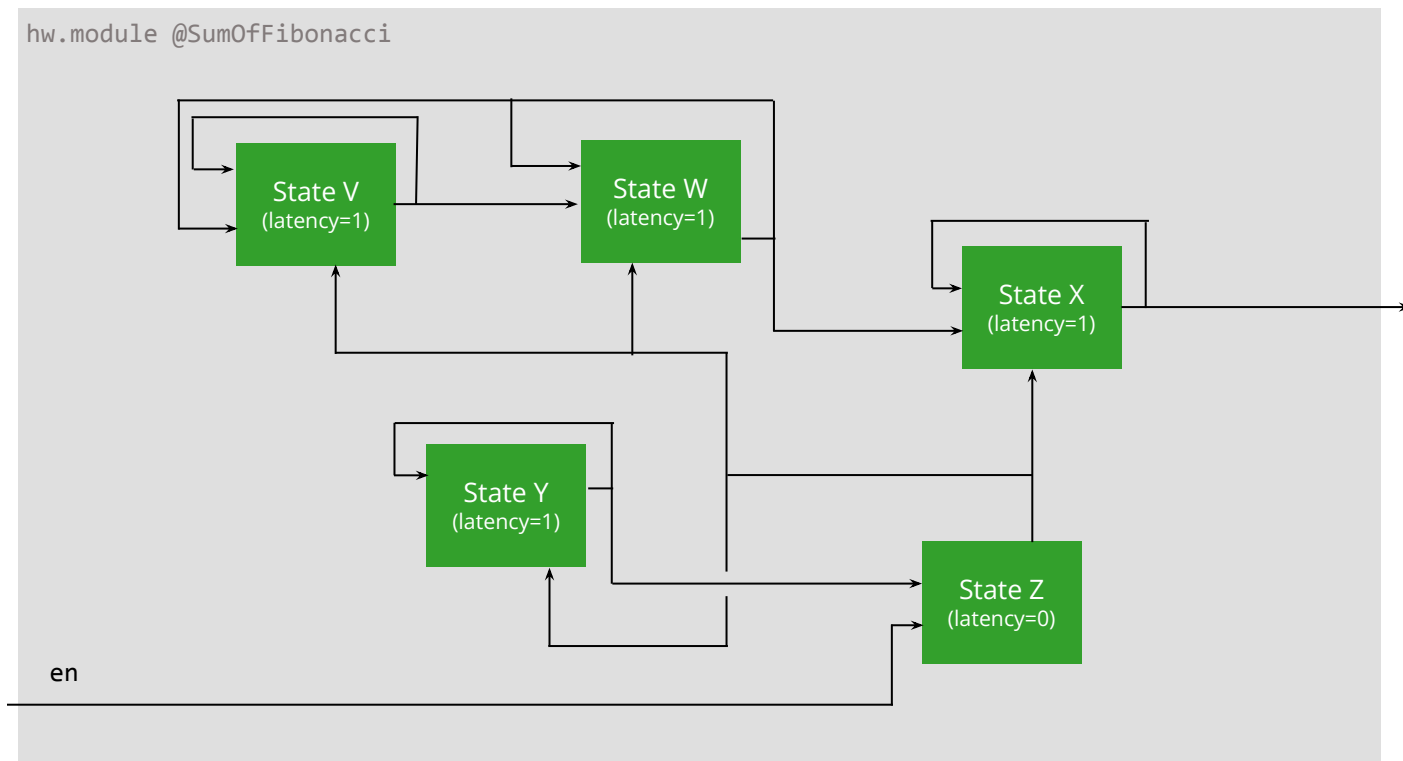
Arc Deferred-CF Comb

Arc Control-Flow Comb

Arc Allocated Arith

LLVM IR

Arc Data-Flow Representation



CIRCT Core Dialects

Arc Data-Flow Comb

Arc Deferred-CF Comb

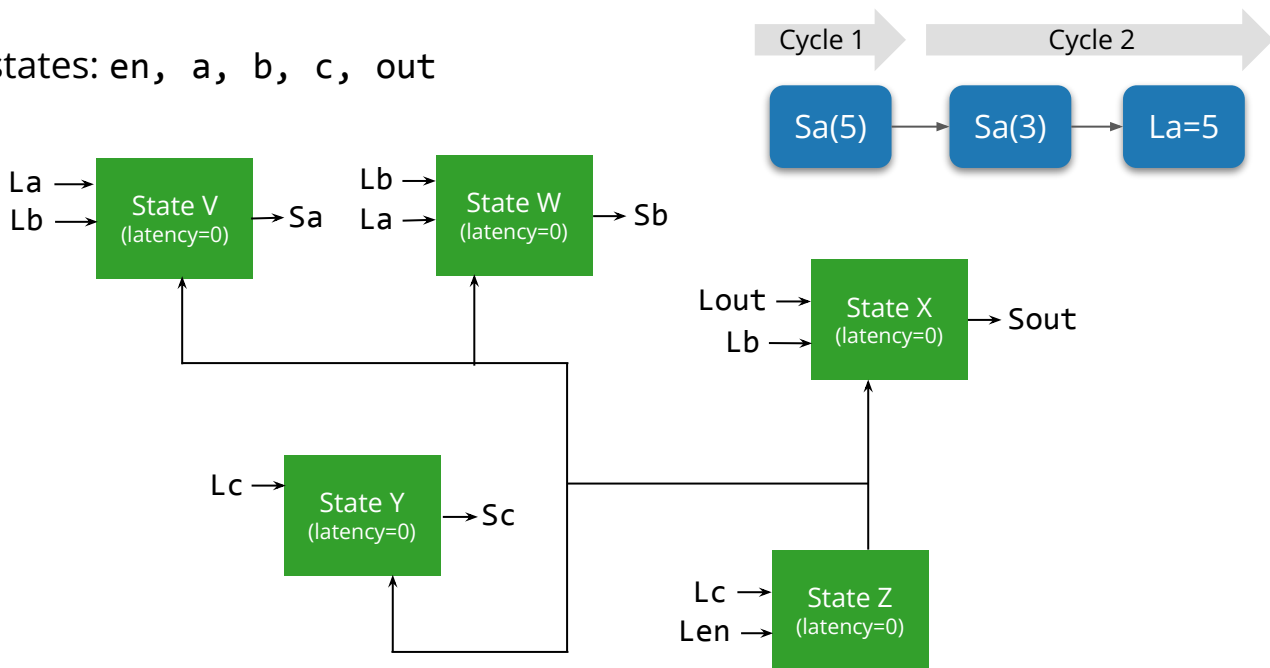
Arc Control-Flow Comb

Arc Allocated Arith

LLVM IR

Arc Deferred-Control-Flow Representation

Allocate states: en, a, b, c, out



La...Load from state a
Sa...Store to state a

- CIRCT Core Dialects
- Arc Data-Flow Comb
- Arc Deferred-CF Comb**
- Arc Control-Flow Comb
- Arc Allocated Arith
- LLVM IR

Arc Deferred-Control-Flow Optimizations

Convert muxes to conditional branches

```
%then = comb.and ...  
%else = comb.add ...  
%0 = comb.mux %cond, %then, %else
```



```
%0 = scf.if %cond {  
  %then = comb.and ...  
  scf.yield %then  
} else {  
  %else = comb.add ...  
  scf.yield %else  
}
```

CIRCT Core Dialects

Arc Data-Flow
Comb

Arc Deferred-CF
Comb

Arc Control-Flow
Comb

Arc Allocated
Arith

LLVM IR

Arc Deferred-Control-Flow Optimizations

Convert muxes to conditional branches

Group resets and enables

```
scf.if %enable {  
  store %a  
  scf.yield  
}  
scf.if %enable {  
  store %b  
  scf.yield  
}
```



```
scf.if %enable {  
  store %a  
  store %b  
  scf.yield  
}
```

CIRCT Core Dialects

Arc Data-Flow
Comb

Arc Deferred-CF
Comb

Arc Control-Flow
Comb

Arc Allocated
Arith

LLVM IR

Arc Deferred-Control-Flow Optimizations

Convert muxes to conditional branches

Group resets and enables

Arc Execution Graph scheduling

CIRCT Core Dialects

Arc Data-Flow
Comb

**Arc Deferred-CF
Comb**

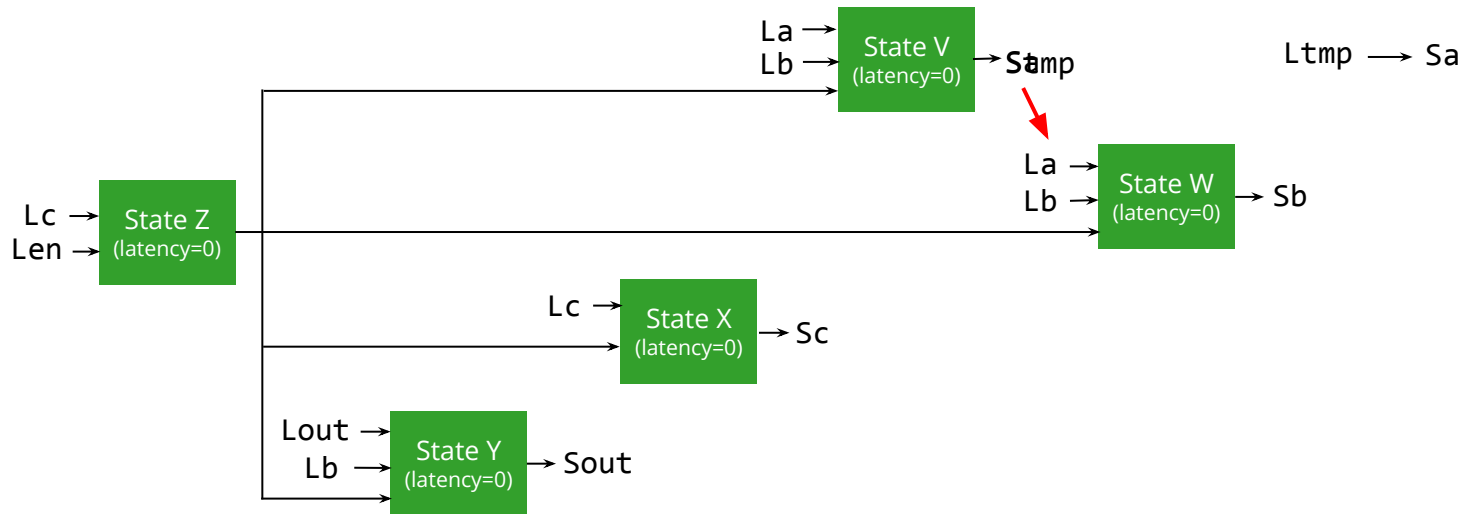
Arc Control-Flow
Comb

Arc Allocated
Arith

LLVM IR

Arc Control-Flow representation

Allocate states: en, a, b, c, out, tmp



CIRCT Core Dialects

Arc Data-Flow Comb

Arc Deferred-CF Comb

Arc Control-Flow Comb

Arc Allocated Arith

LLVM IR

Arc Control-Flow Optimizations

Memory layout



CIRCT Core Dialects

Arc Data-Flow
Comb

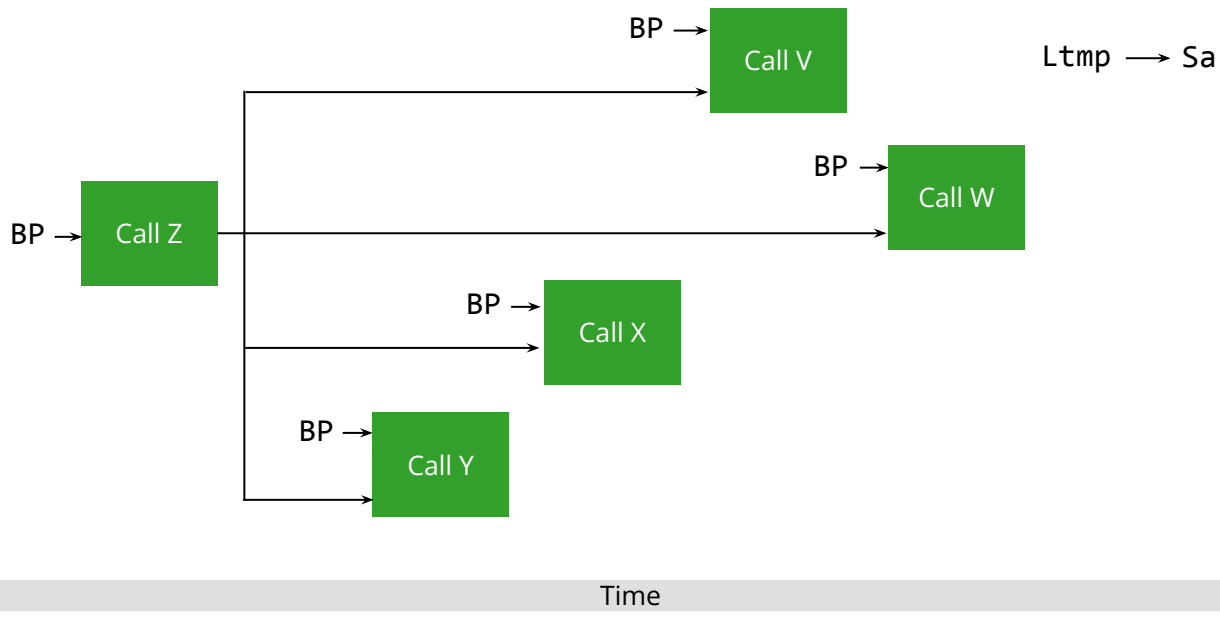
Arc Deferred-CF
Comb

**Arc Control-Flow
Comb**

Arc Allocated
Arith

LLVM IR

Arc Allocated



CIRCT Core Dialects

Arc Data-Flow Comb

Arc Deferred-CF Comb

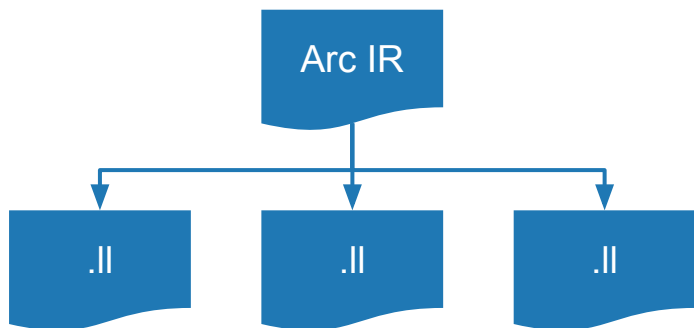
Arc Control-Flow Comb

Arc Allocated Arith

LLVM IR

Arc Allocated Optimizations

Split into multiple files for opt+llc parallelism



CIRCT Core Dialects

Arc Data-Flow
Comb

Arc Deferred-CF
Comb

Arc Control-Flow
Comb

**Arc Allocated
Arith**

LLVM IR

LLVM IR

```
define void @SumOfFibonacci_passthrough(ptr %0) {  
  %2 = getelementptr i8, ptr %0, i32 24  
  %3 = load i32, ptr %2, align 4  
  %4 = getelementptr i8, ptr %0, i32 4  
  store i32 %3, ptr %4, align 4  
  ret void  
}
```

```
define void @SumOfFibonacci_clock(ptr %0) {  
  %2 = getelementptr i8, ptr %0, i32 20  
  %3 = load i32, ptr %2, align 4  
  %4 = getelementptr i8, ptr %0, i32 2  
  %5 = load i1, ptr %4, align 1  
  %6 = icmp ult i32 %3, 12  
  %7 = and i1 %5, %6  
  %8 = getelementptr i8, ptr %0, i32 12  
  %9 = load i32, ptr %8, align 4  
  %10 = getelementptr i8, ptr %0, i32 16  
  %11 = load i32, ptr %10, align 4  
  %12 = getelementptr i8, ptr %0, i32 1  
  %13 = load i1, ptr %12, align 1  
  %14 = add i32 %9, %11  
  %15 = select i1 %7, i32 %14, i32 %9  
}
```

```
%16 = select i1 %13, i32 0, i32 %15  
%17 = getelementptr i8, ptr %0, i32 32  
store i32 %9, ptr %17, align 4  
store i32 %16, ptr %8, align 4  
%18 = load i32, ptr %17, align 4  
%19 = select i1 %7, i32 %18, i32 %11  
%20 = select i1 %13, i32 1, i32 %19  
%21 = load i32, ptr %10, align 4  
%22 = getelementptr i8, ptr %0, i32 36  
store i32 %21, ptr %22, align 4  
store i32 %20, ptr %10, align 4  
%23 = add i32 %3, 1  
%24 = select i1 %7, i32 %23, i32 %3  
%25 = select i1 %13, i32 0, i32 %24  
store i32 %25, ptr %2, align 4  
%26 = getelementptr i8, ptr %0, i32 24  
%27 = load i32, ptr %26, align 4  
%28 = load i32, ptr %22, align 4  
%29 = add i32 %27, %28  
%30 = select i1 %7, i32 %29, i32 %27  
%31 = select i1 %13, i32 0, i32 %30  
store i32 %31, ptr %26, align 4  
ret void  
}
```

CIRCT Core Dialects

Arc Data-Flow
Comb

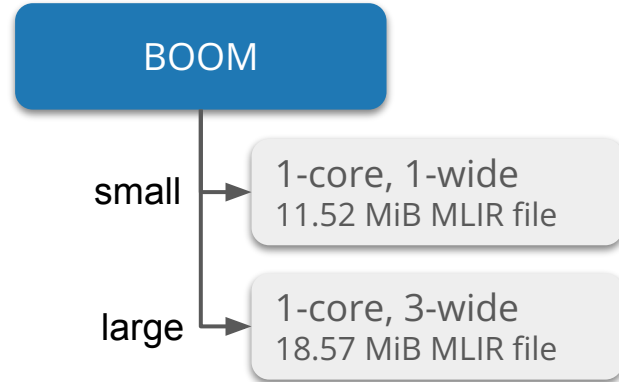
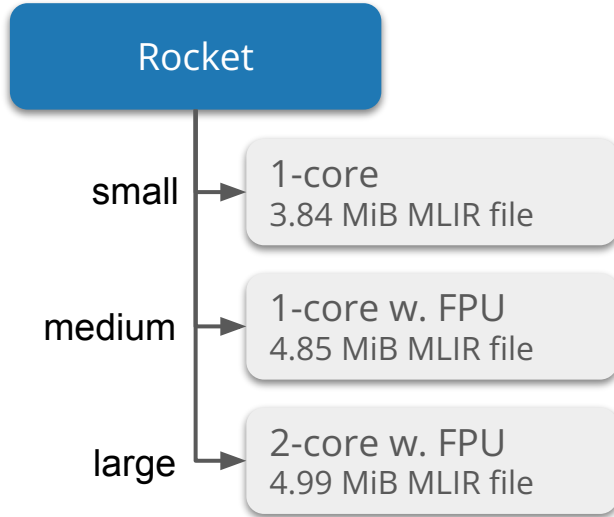
Arc Deferred-CF
Comb

Arc Control-Flow
Comb

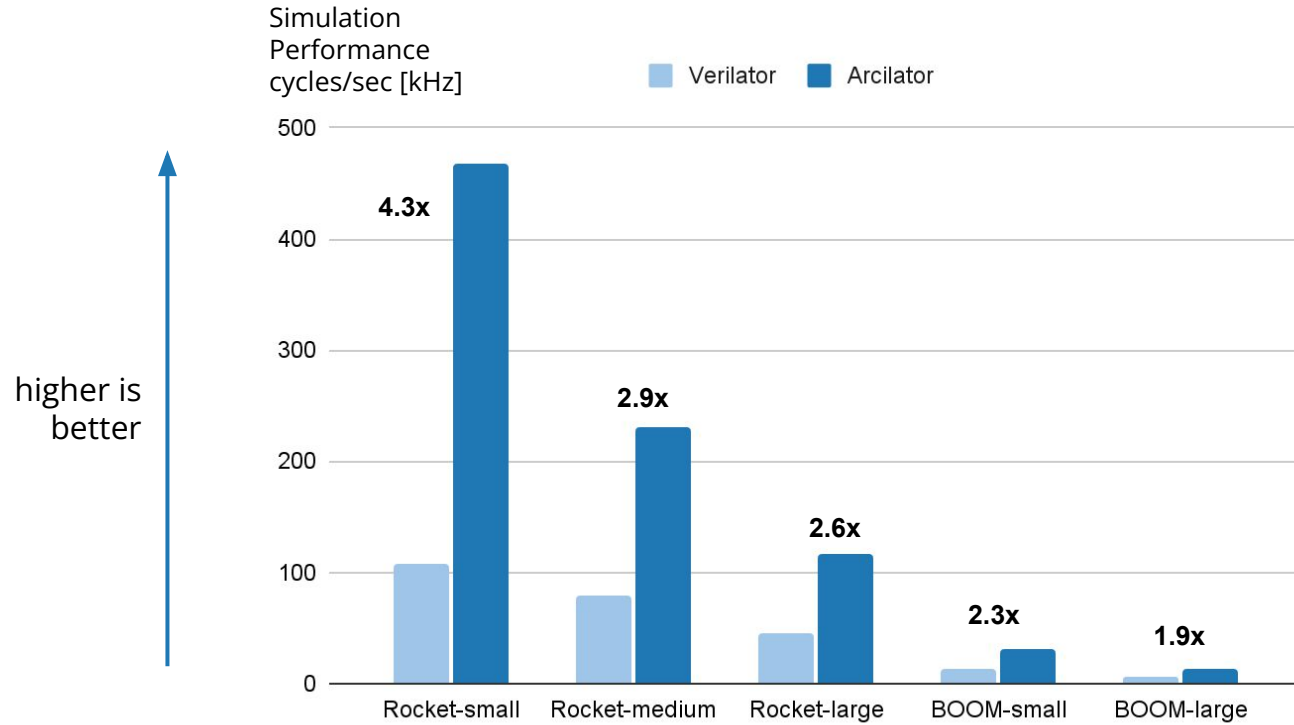
Arc Allocated
Arith

LLVM IR

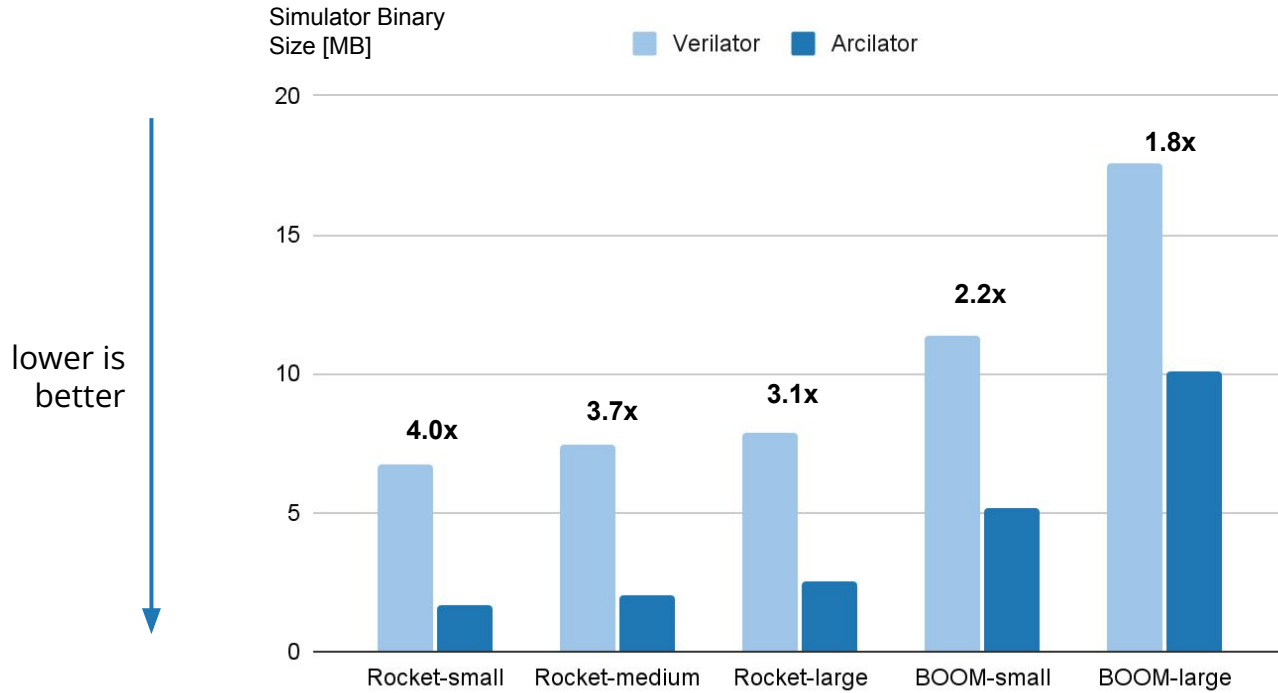
Benchmarked CPU Designs



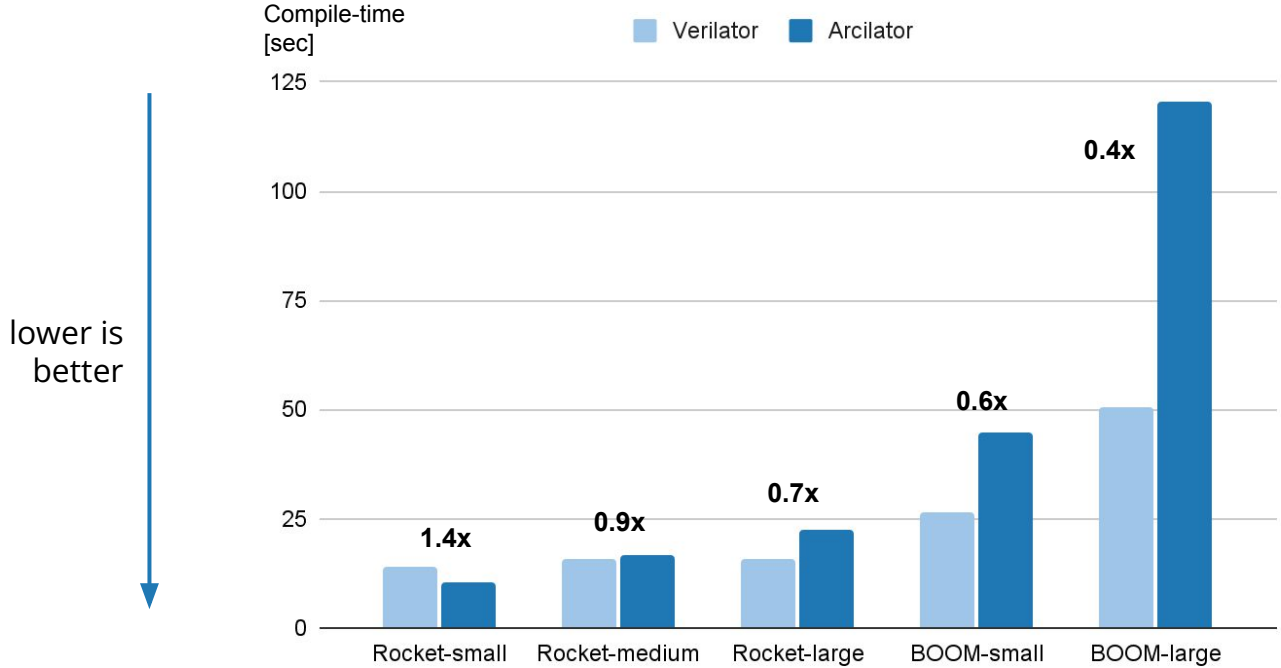
Arcilator simulates fast



Arcilator produces small binaries



Reasonable compile-time, more to come



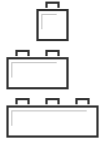
Many tools are needed to produce hardware

Synthesis

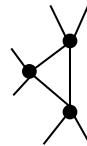
Simulation

Formal Verification

Conclusion



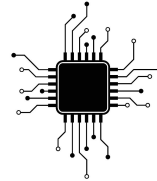
Library-based, modular design



Cyclic hardware
→ DAG of register arcs



≥ 2x faster than Verilator*



Simulate full RISC-V cores