Design Tools for Hardware

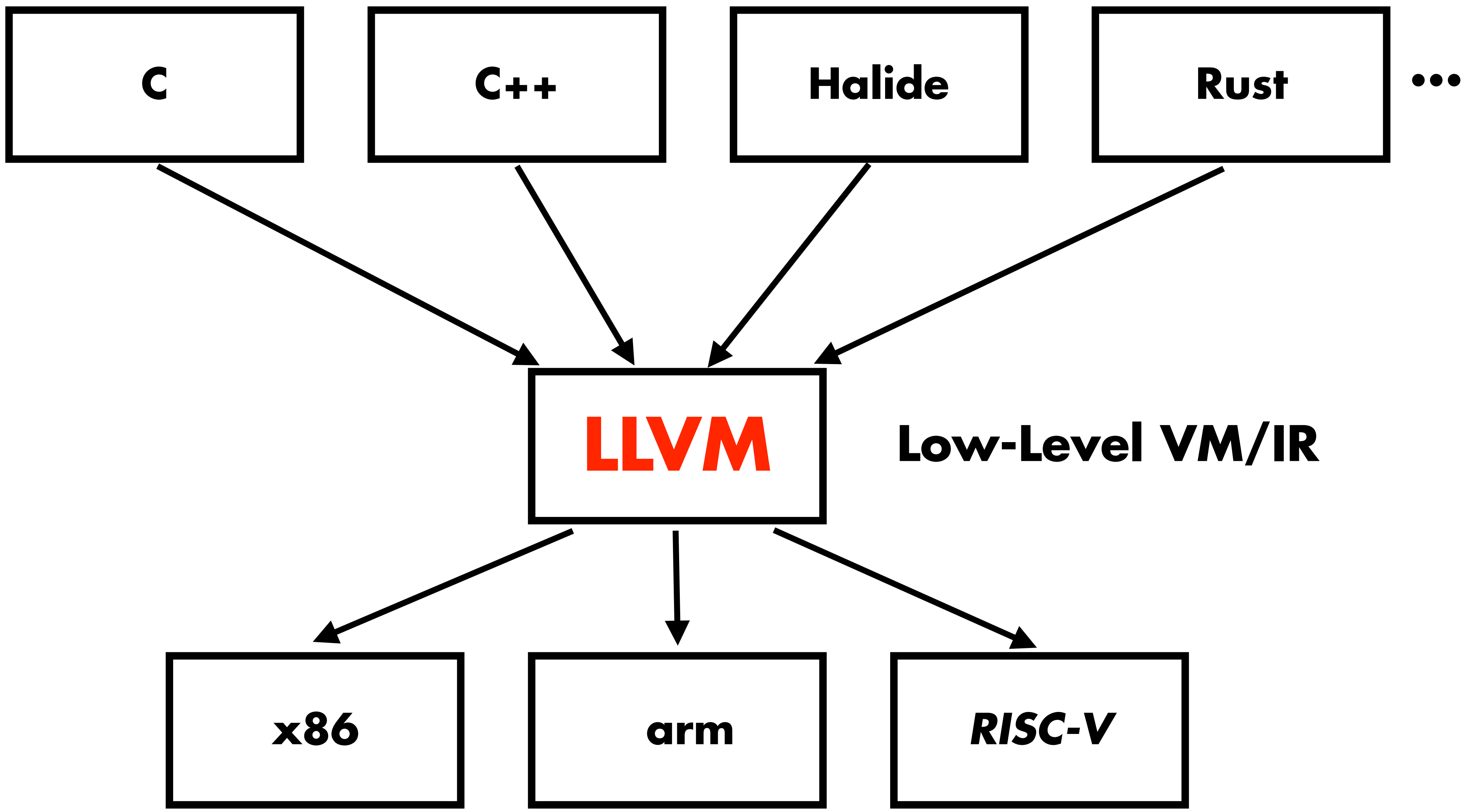
Pat Hanrahan

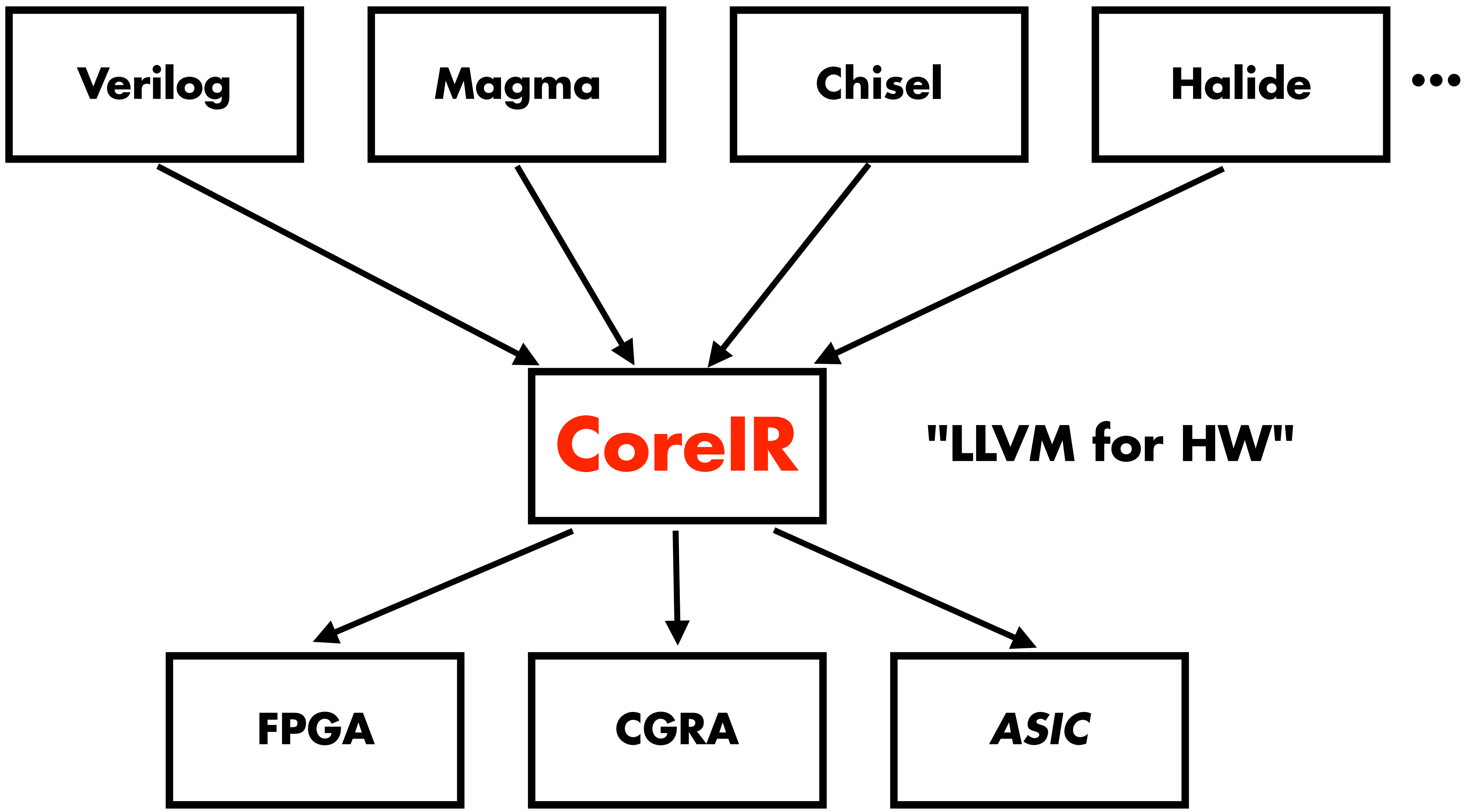
AHA Center Retreat

Asilomar

June 23, 2018

Hardware Compiler


C
C++
Halide
Rust
...
LLVM
Low-Level VM/IR
x86
arm
RISC-V


Verilog
Magma
Chisel
Halide
...
CoreIR
"LLVM for HW"
FPGA
CGRA
ASIC

Different Abstractions

LLVM - Processor + memory

- ALU, registers, memory

CoreIR - RTL

- Combination and sequential logic
- Distributed state / registers and memories
- Structural : DAG of operations (always synthesizable)
- Hierarchical modules
- Generators (parameterized modules)

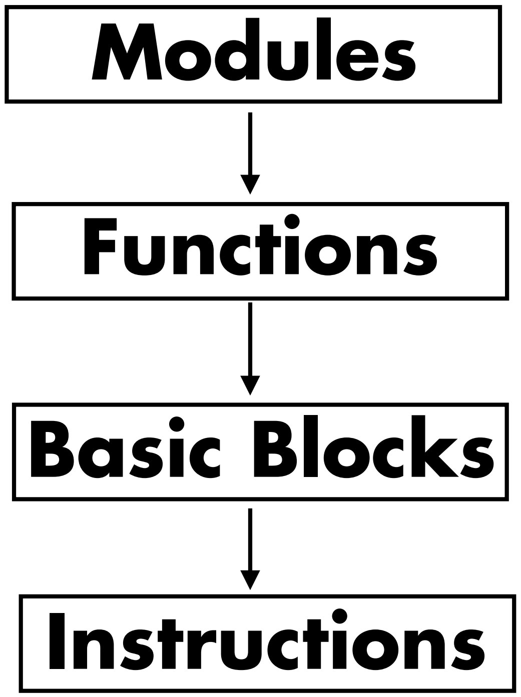
CoreIR Primitives

Bitwise	buf, not, and, or, xor, andr, orr, xorr, shl, lshr, ashr
Arithmetic	neg add, sub, mul udiv, urem, sdiv, srem, smod
Comp	eq, neq, slt, sgt, sle, sge, ult, ugt, ule, uge
Stateful	reg, regrst, mem
Other	mux, slice, concat, zext, sext

Cross-validate implementations: python, C++, verilog

https://github.com/StanfordAHA/Primitives/blob/master/coreirprims.csv

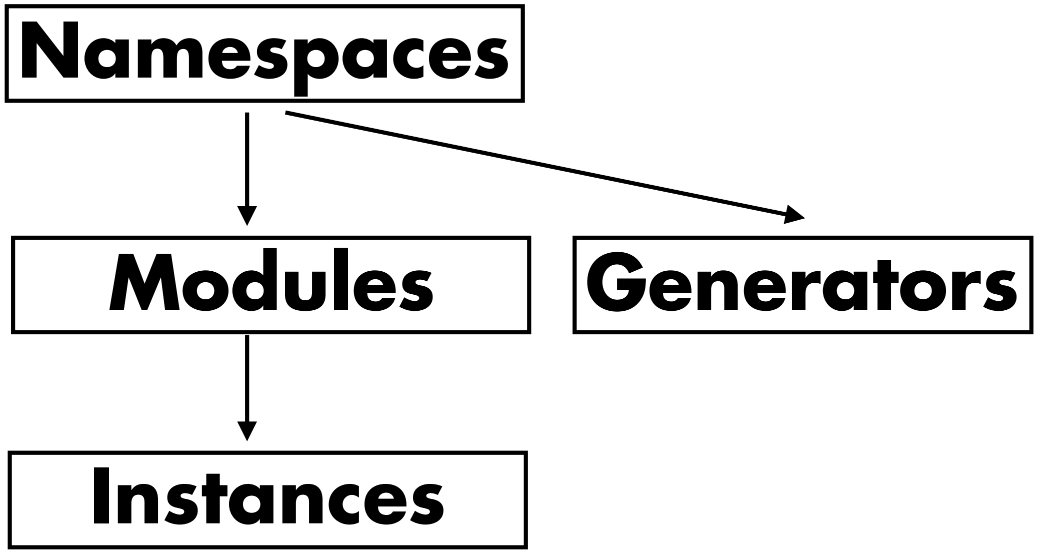
LLVM Pass Types



IR passes
- ModulePass
- CallGraphPass
- CallGraphSCCPass
- FunctionPass
- LoopPass
- RegionPass
- BasicBlockPass
- InstructionPass

Pass manager
- Executes passes in order
- Caches intermediates

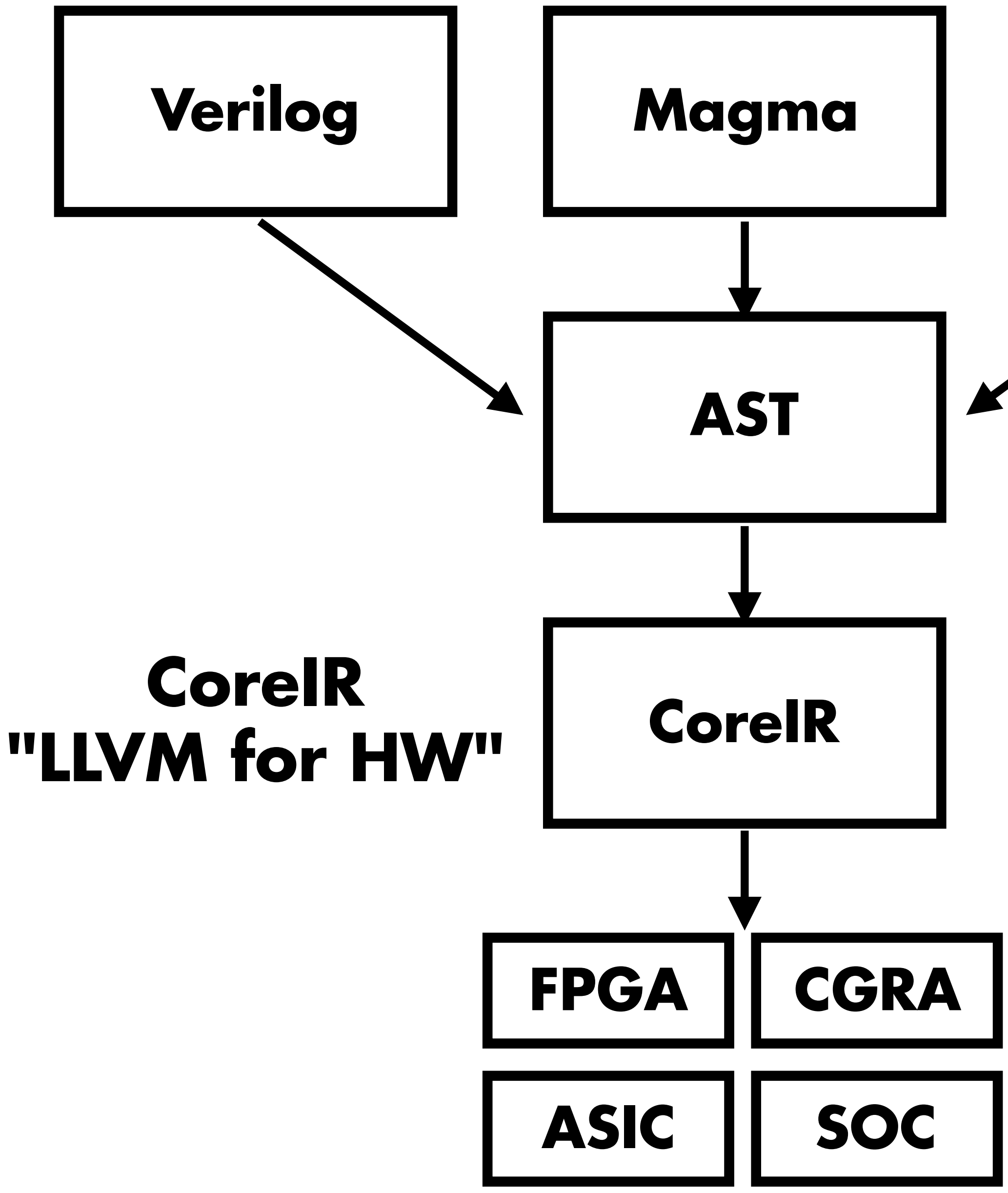
CoreIR Pass Types



Direct Pass Types
- NamespacePass
- ModulePass
- InstancePass

Other Pass Types
- InstanceGraphPass
- InstanceVisitorPass

CoreIR Passes



CoreIR passes

- Constant folding
- Graph rewrites
- Dead circuit elimination
- Generators
- Elaboration
- Debugging support
- Verification / model ch.
- Technology mapping
- Linking

Incorporating Physical Design

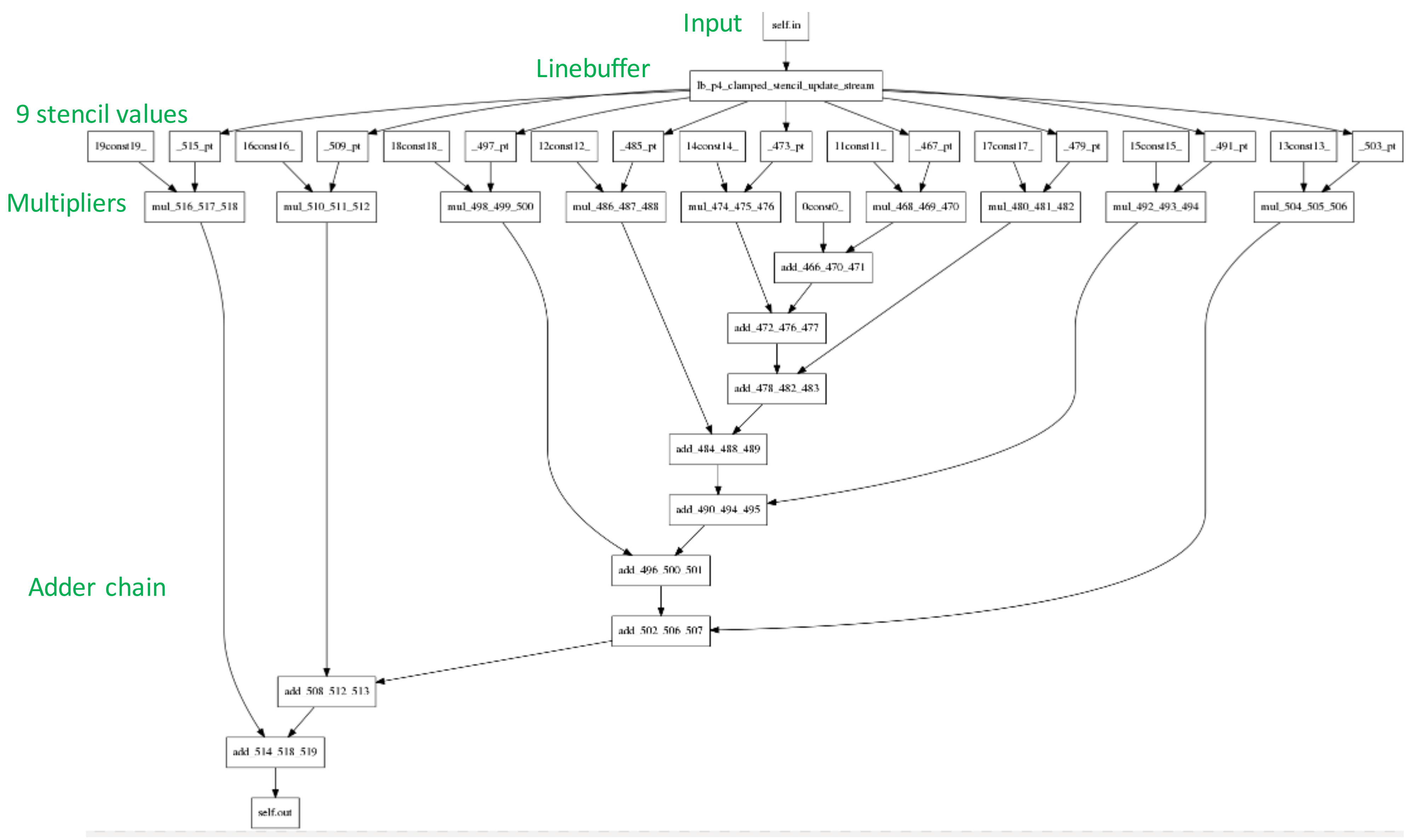
Physical design

- Floor plans
- Clock domains
- Power domains
- Global signals

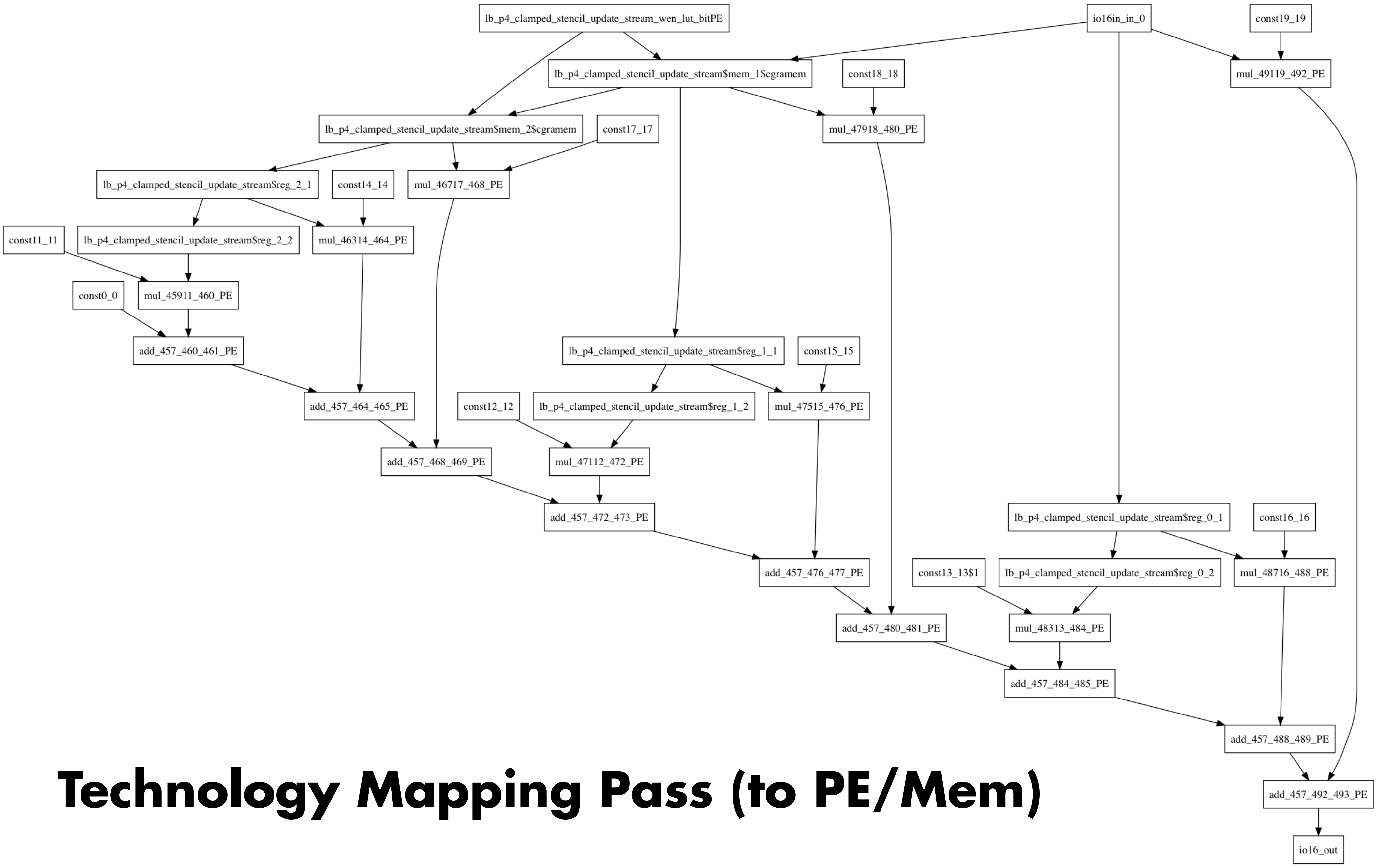
Leverage

- Types instead of names
- Meta-data annotations
- CoreIR passes

Convolution in CoreIR



CoreIR Primitives to CGRA Primitives



Technology Mapping Pass (to PE/Mem)

Magma

https://github.com/phanrahan/magma

Generators

Generators widely used in hardware design

- `Verilog` **parameters,** `generate` **statement**
- `Genesis2`, **…**
- `coregen, rocketship, …`
- **…**

Hardware design is meta-programming

- **Hardware is a spatial "program"**
- **Hardware design language (HDL) is a program that generates a (HW) program**

Chisel

```scala
//A n-bit adder with carry in and carry out
class Adder(val n:Int) extends Module {
  val io = IO(new Bundle {
    val A    = Input(UInt(n.W))
    val B    = Input(UInt(n.W))
    val Cin  = Input(UInt(1.W))
    val Sum  = Output(UInt(n.W))
    val Cout = Output(UInt(1.W))
  })
  //create an Array of FullAdders
  val FAs   = Array.fill(n)(Module(new FullAdder()).io)
  val carry = Wire(Vec(n+1, UInt(1.W)))
  val sum   = Wire(Vec(n, Bool()))

  //first carry is the top level carry in
  carry(0) := io.Cin

  //wire up the ports of the full adders
  for (i <- 0 until n) {
    FAs(i).a := io.A(i)
    FAs(i).b := io.B(i)
    FAs(i).cin := carry(i)
    carry(i+1) := FAs(i).cout
    sum(i) := FAs(i).sum.toBool()
  }
  io.Sum := sum.asUInt
  io.Cout := carry(n)
}
```

Magma

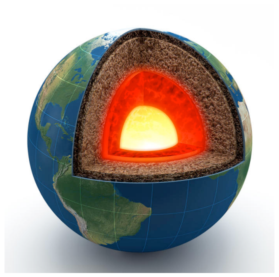
```python
class FullAdder(Circuit):
    IO = ["a", In(Bit), "b", In(Bit), "cin", In(Bit),
          "out", Out(Bit), "cout", Out(Bit)]
    @classmethod
    def definition(io):
        _sum = io.a ^ io.b ^ io.cin
        wire(_sum, io.out)
        carry = io.a & io.b | io.b & io.cin | io.a & io.cin
        wire(carry, io.cout)

def DefineAdder(N):
    T = UInt(N)
    class Adder(Circuit):
        name = f"Adder{N}"
        IO = ["a", In(T), "b", In(T), cin", In(Bit),
              "out", Out(T), "cout", Out(Bit)]
        @classmethod
        def definition(io):
            adders = col(FullAdder, N)
            circ = fold(adders, {"cin":"cout"})
            wire(io.a, circ.a); wire(io.b, circ.b)
            wire(io.cin, circ.cin)
            wire(io.cout, circ.cout)
            wire(io.out, circ.out)
    return Adder

Adder4 = DefineAdder(4)
adder = Adder4()
wire(main.I0, adder.a)
wire(main.I1, adder.b)
```

Python

- Dynamically-typed, polymorphic
- Popular, with extensive libraries



Magma - "Chisel in Python"

- Static product data types
- Circuits, instances, and wiring

Python-Magma system

- Python meta-programs Magma
- Enables embedded DSLs

Linking with Generators

Linking

- Encourages modularity and reuse
- Incremental builds

CoreIR "object" files contain generator instances

- Required to interoperate with verilog designs

Issues

- Type checking generators
  - Type signature of a module interface could be any function of the parameters
- Expanding generators
  - Run generator code while linking
  - Supporting generators in different languages, different execution contexts

Next-Gen Generators Plans

Evolve designs

Genesis2

- Existing CGRA Design

Genesis3

- Genesis2 in python

Magma

- From text processing to circuits / wiring

DSLs

- PE, Memory, Interconnect, SOC

DSLs

DSLs

Image processing - halide

Hardware

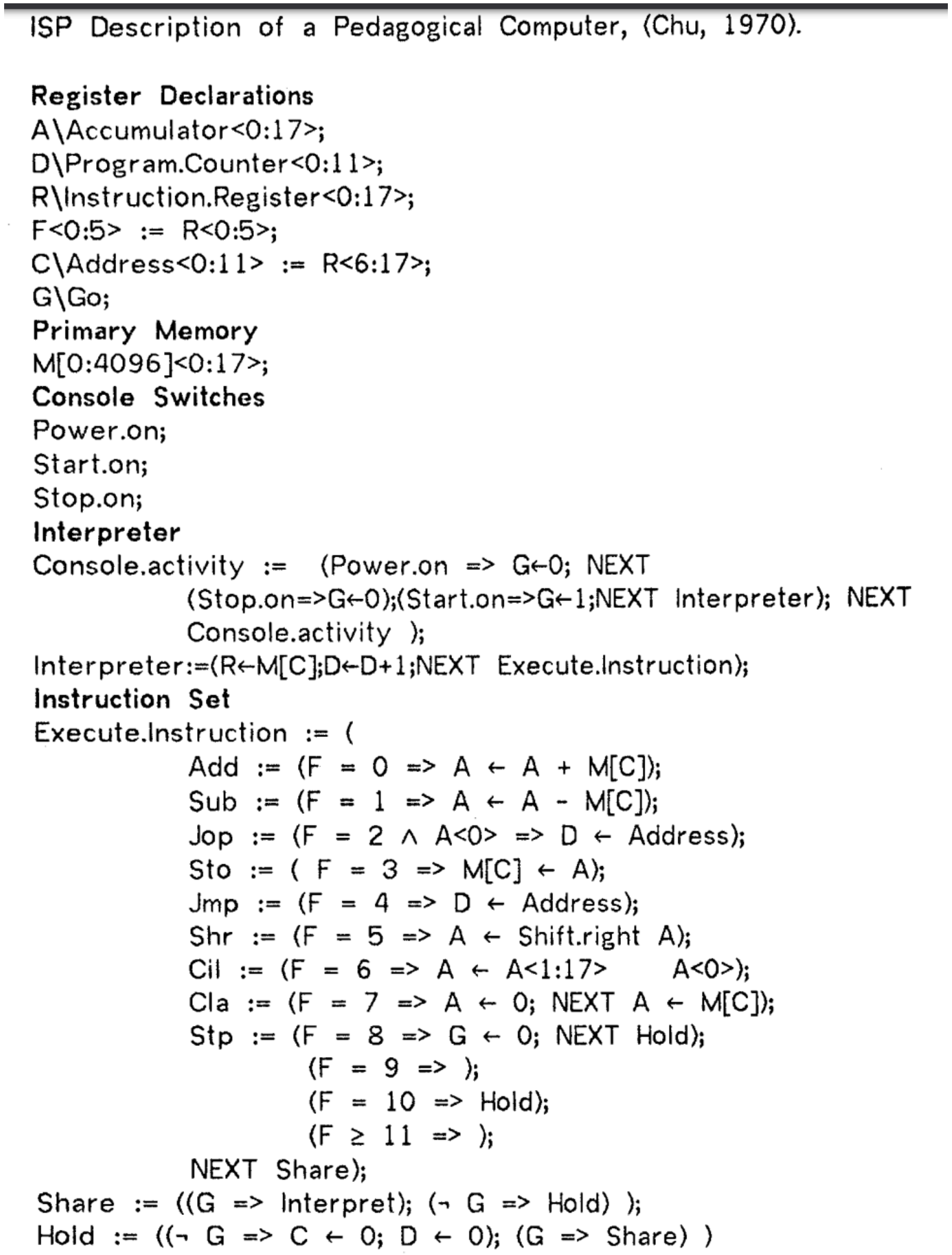
- Processing element - PE
- Memory
- Interconnect
- SOC

Finite state machines - silica (co-routines)

Testing - fault

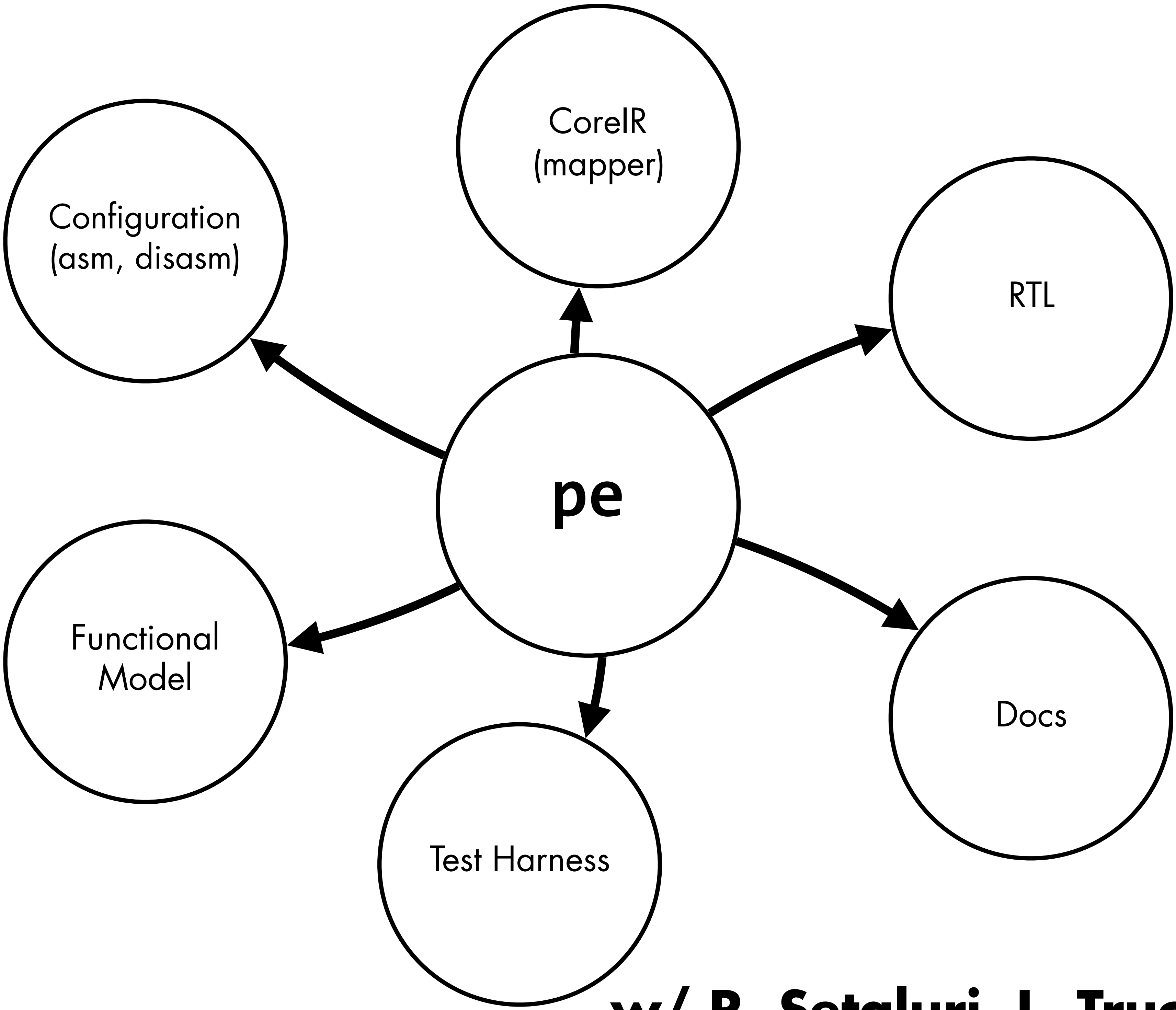
COMPUTER STRUCTURES: PRINCIPLES AND EXAMPLES
DANIEL P. SIEWIOREK / C. GORDON BELL / ALLEN NEWELL
COMPUTER SCIENCE SERIES

ISP
Also Tensilica TIE



ISP Description of a Pedagogical Computer, (Chu, 1970).

```
Register Declarations
A\Accumulator<0:17>;
D\Program.Counter<0:11>;
R\Instruction.Register<0:17>;
F<0:5> := R<0:5>;
C\Address<0:11> := R<6:17>;
G\Go;
Primary Memory
M[0:4096]<0:17>;
Console Switches
Power.on;
Start.on;
Stop.on;
Interpreter
Console.activity :=  (Power.on => G←0; NEXT
          (Stop.on=>G←0);(Start.on=>G←1;NEXT Interpreter); NEXT
          Console.activity );
Interpreter:=(R←M[C];D←D+1;NEXT Execute.Instruction);
Instruction Set
Execute.Instruction := (
          Add := (F = 0 => A ← A + M[C]);
          Sub := (F = 1 => A ← A - M[C]);
          Jop := (F = 2 ∧ A<0> => D ← Address);
          Sto := ( F = 3 => M[C] ← A);
          Jmp := (F = 4 => D ← Address);
          Shr := (F = 5 => A ← Shift.right A);
          Cil := (F = 6 => A ← A<1:17>    A<0>);
          Cla := (F = 7 => A ← 0; NEXT A ← M[C]);
          Stp := (F = 8 => G ← 0; NEXT Hold);
                 (F = 9 => );
                 (F = 10 => Hold);
                 (F ≥ 11 => );
          NEXT Share);
Share := ((G => Interpret); (¬ G => Hold) );
Hold := ((¬ G => C ← 0; D ← 0); (G => Share) )
```



w/ R. Setaluri, L. Truong

Hardware Design Tools

Magma - Chisel in python

CoreIR optimizing compiler for RTL

Cross-language linking of generators

Complete OSS tool-chain for CGRA

Next-gen generator in python

Parameterized SOC generator from DSLs

Also

- Debugger and breakpoints
- Testing
- PNR

Collaborative Projects: Get Involved!

HLS front-end to CoreIR?

Reusable components?

Participate in hackathons?

New applications?

Thank You