Formal Verification of Hardware in the CIRCT Compiler

MSc. Thesis - Amelia Dobis (2nd October 2023 - 12th April 2024)

Advisors: Kevin Laeufer (UC Berkeley) & Prof. Zhendong Su (ETH Zürich)

Overview

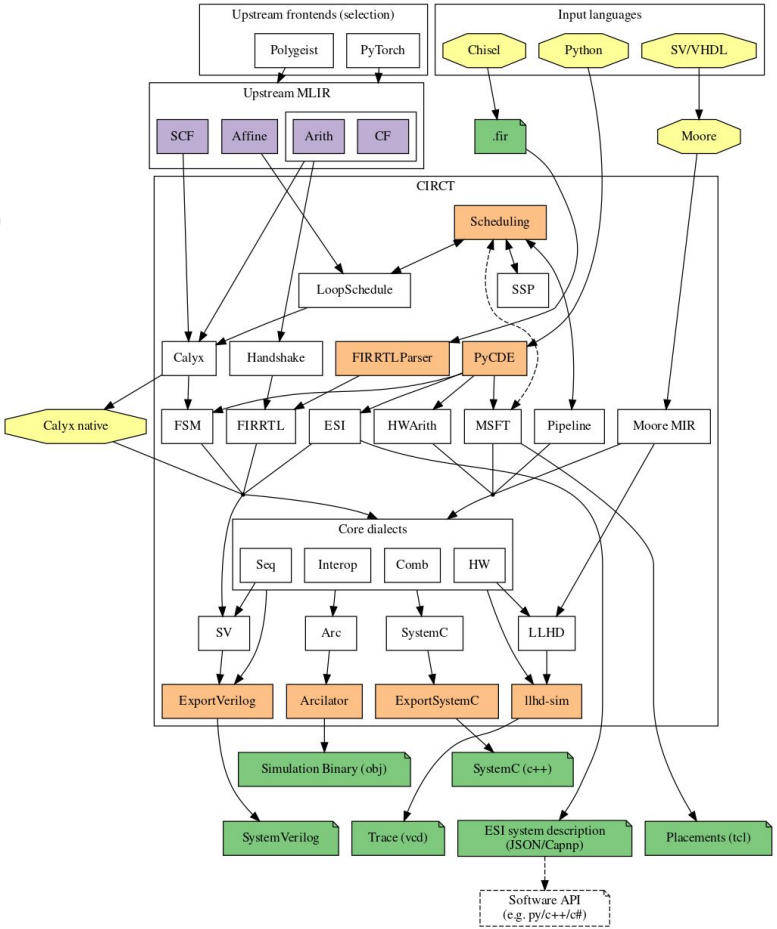
1)Background & Motivation

2)Creating a Formal Backend for CIRCT

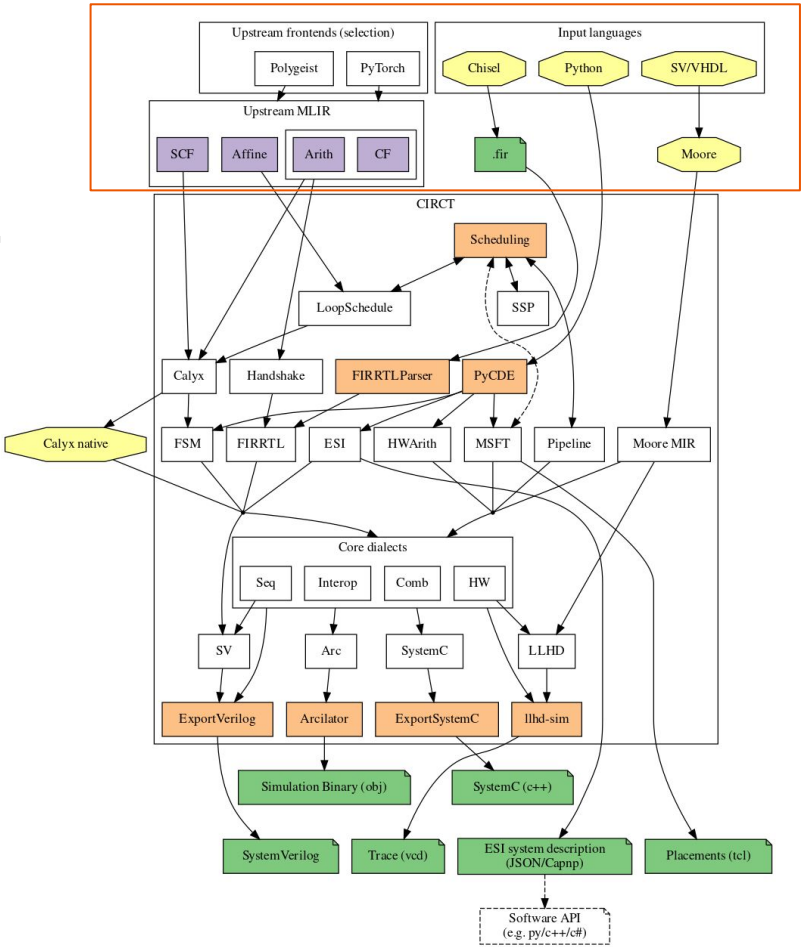
3)Enabling the use of SVA Sequences

4)Conclusion

Background

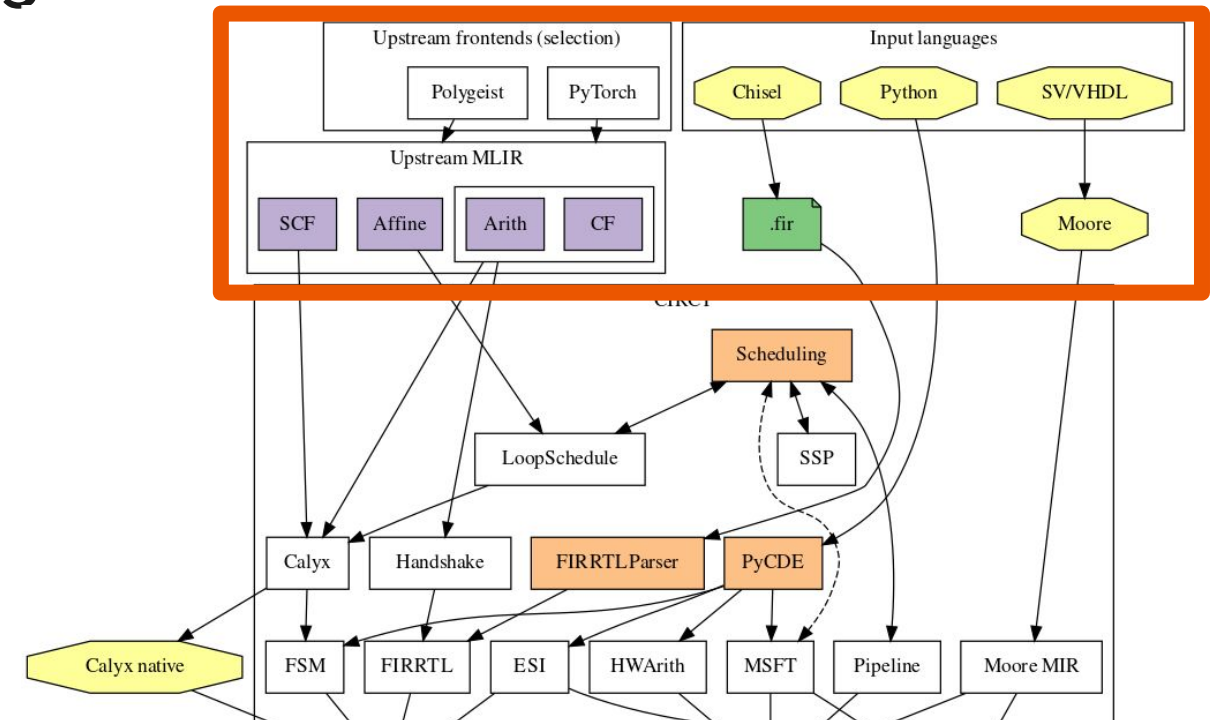

Upstream frontends (selection)
Polygeist
PyTorch
Input languages
Chisel
Python
SV/VHDL
Upstream MLIR
SCF
Affine
Arith
CF
.fir
Moore
CIRCT
Scheduling
LoopSchedule
SSP
Calyx
Handshake
FIRRTLParser
PyCDE
Calyx native
FSM
FIRRTL
ESI
HWArith
MSFT
Pipeline
Moore MIR
Core dialects
Seq
Interop
Comb
HW
SV
Arc
SystemC
LLHD
ExportVerilog
Arcilator
ExportSystemC
llhd-sim
Simulation Binary (obj)
SystemC (c++)
SystemVerilog
Trace (vcd)
ESI system description (JSON/Capnp)
Placements (tcl)
Software API (e.g. py/c++/c#)

Background

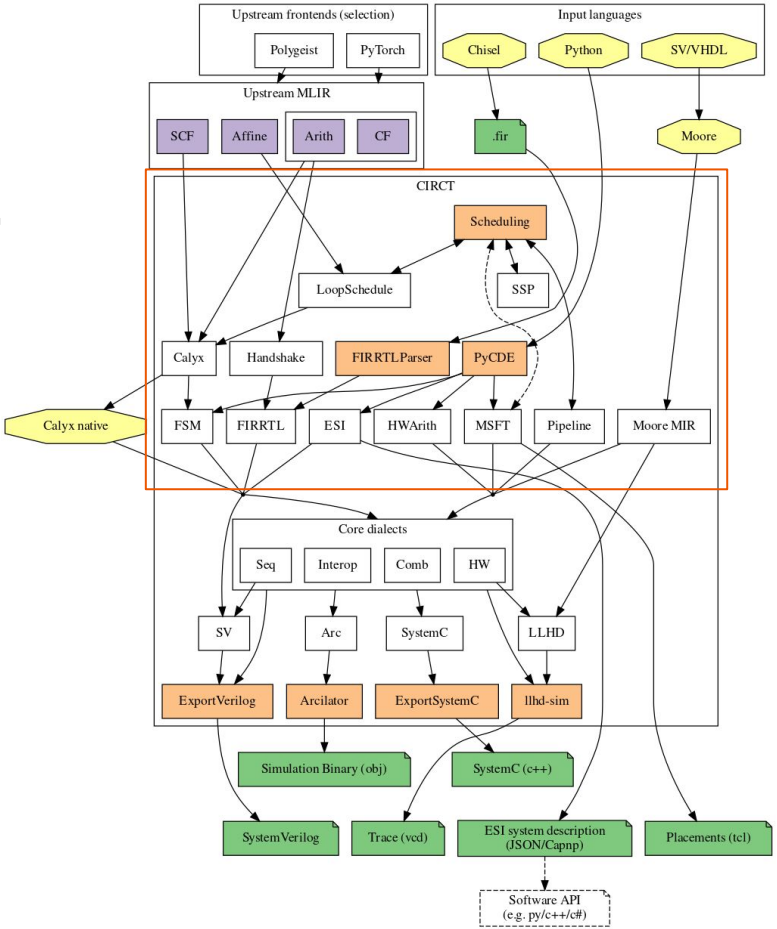

Upstream frontends (selection)
Polygeist
PyTorch
Input languages
Chisel
Python
SV/VHDL
Upstream MLIR
SCF
Affine
Arith
CF
.fir
Moore
CIRCT
Scheduling
LoopSchedule
SSP
Calyx
Handshake
FIRRTLParser
PyCDE
Calyx native
FSM
FIRRTL
ESI
HWArith
MSFT
Pipeline
Moore MIR
Core dialects
Seq
Interop
Comb
HW
SV
Arc
SystemC
LLHD
ExportVerilog
Arcilator
ExportSystemC
llhd-sim
Simulation Binary (obj)
SystemC (c++)
SystemVerilog
Trace (vcd)
ESI system description (JSON/Capnp)
Placements (tcl)
Software API (e.g. py/c++/c#)

Background

CIRCT Frontends

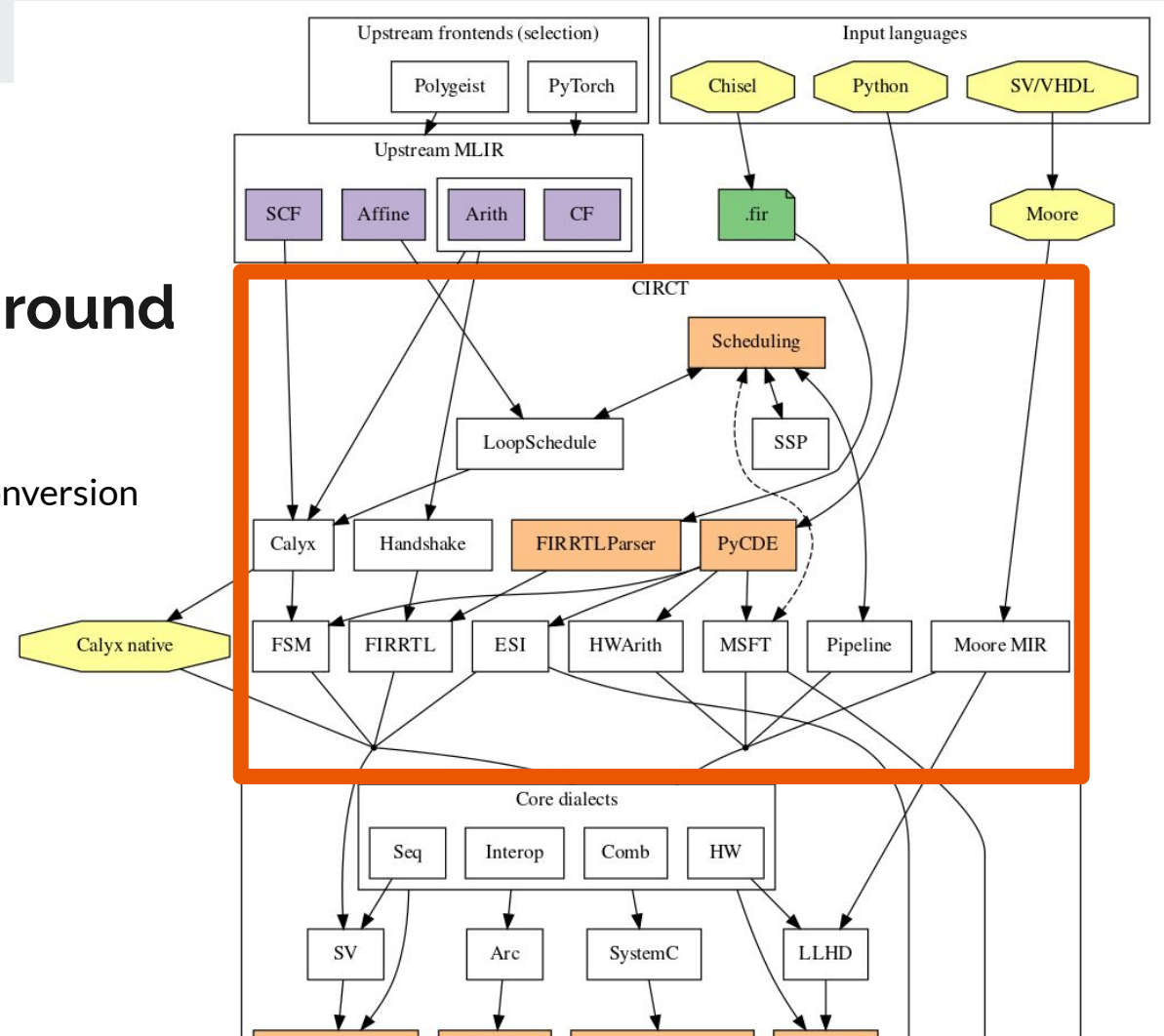


Background

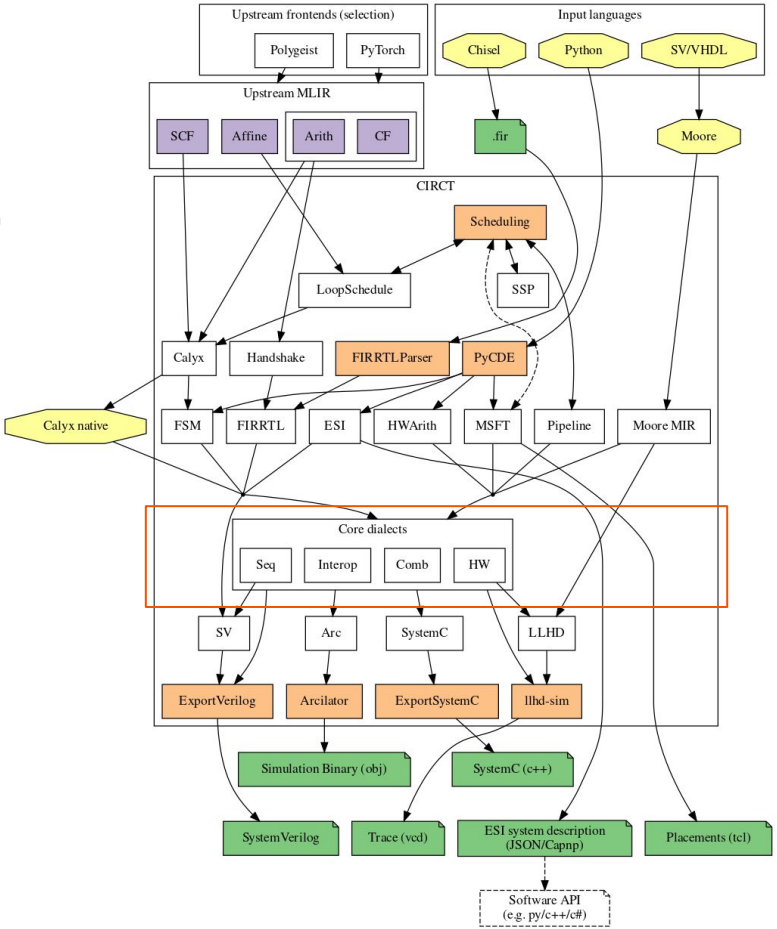

Upstream frontends (selection)
Polygeist
PyTorch
Input languages
Chisel
Python
SV/VHDL
Upstream MLIR
SCF
Affine
Arith
CF
.fir
Moore
CIRCT
Scheduling
LoopSchedule
SSP
Calyx
Handshake
FIRRTLParser
PyCDE
Calyx native
FSM
FIRRTL
ESI
HWArith
MSFT
Pipeline
Moore MIR
Core dialects
Seq
Interop
Comb
HW
SV
Arc
SystemC
LLHD
ExportVerilog
Arcilator
ExportSystemC
llhd-sim
Simulation Binary (obj)
SystemC (c++)
SystemVerilog
Trace (vcd)
ESI system description (JSON/Capnp)
Placements (tcl)
Software API (e.g. py/c++/c#)

Background

Optimization/Conversion Dialects

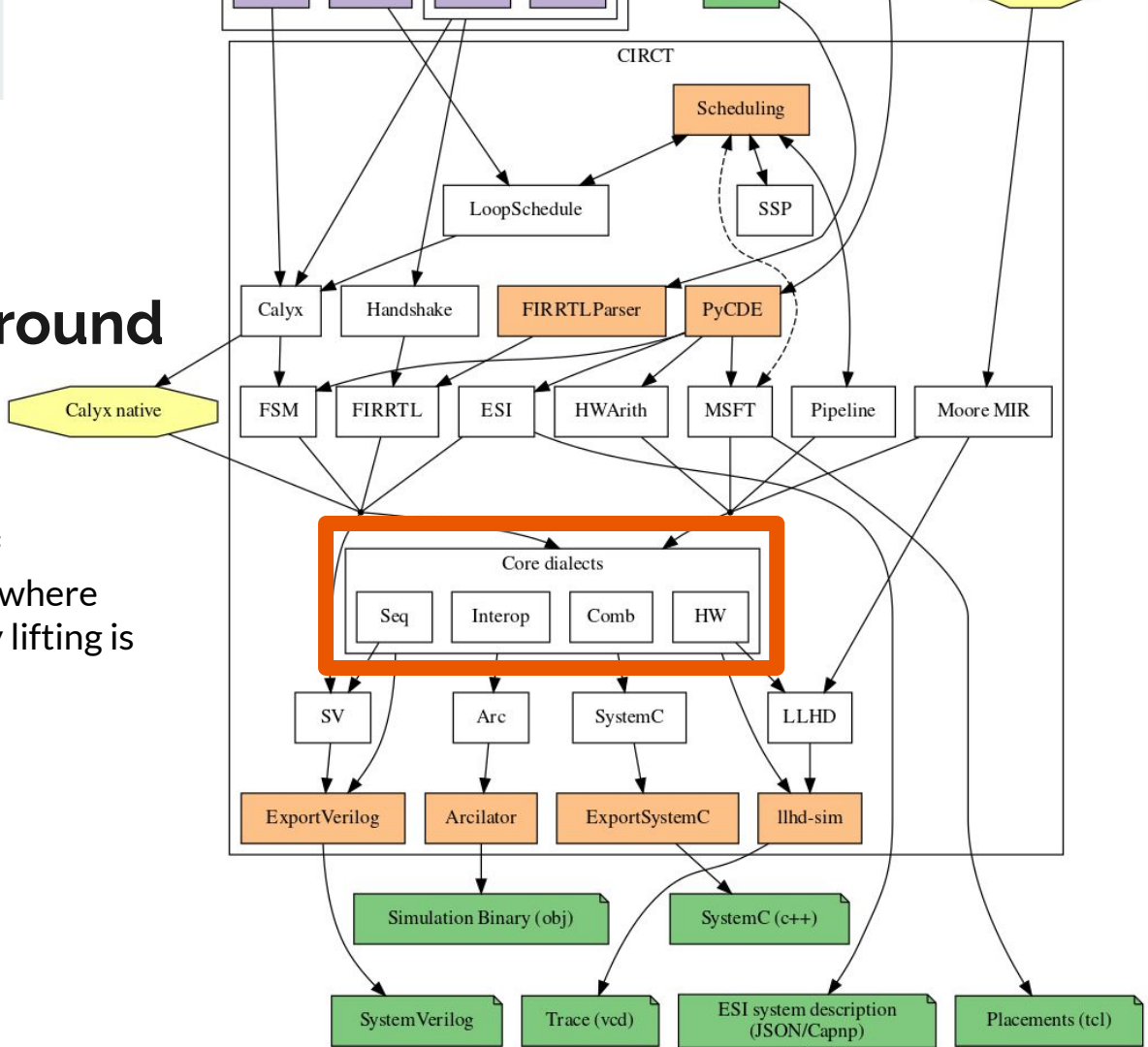


Background

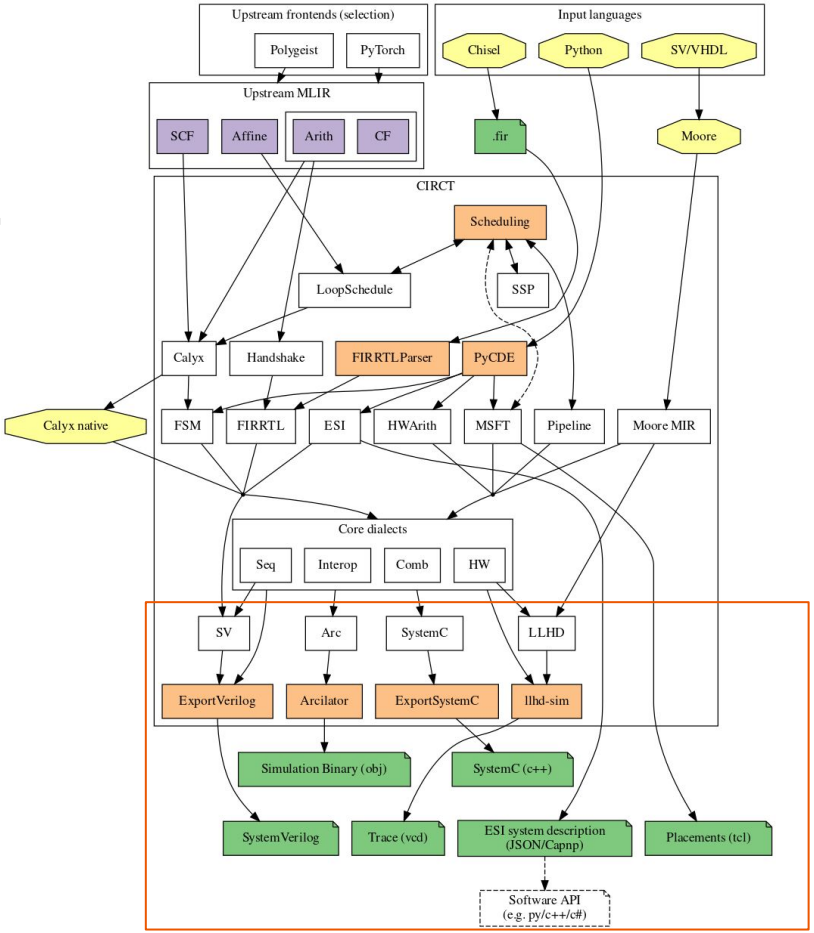

Upstream frontends (selection)
Polygeist
PyTorch
Input languages
Chisel
Python
SV/VHDL
Upstream MLIR
SCF
Affine
Arith
CF
.fir
Moore
CIRCT
Scheduling
LoopSchedule
SSP
Calyx
Handshake
FIRRTLParser
PyCDE
Calyx native
FSM
FIRRTL
ESI
HWArith
MSFT
Pipeline
Moore MIR
Core dialects
Seq
Interop
Comb
HW
SV
Arc
SystemC
LLHD
ExportVerilog
Arcilator
ExportSystemC
llhd-sim
Simulation Binary (obj)
SystemC (c++)
SystemVerilog
Trace (vcd)
ESI system description (JSON/Capnp)
Placements (tcl)
Software API (e.g. py/c++/c#)

Background

Generalized representation of hardware. This is where most of the heavy lifting is done.

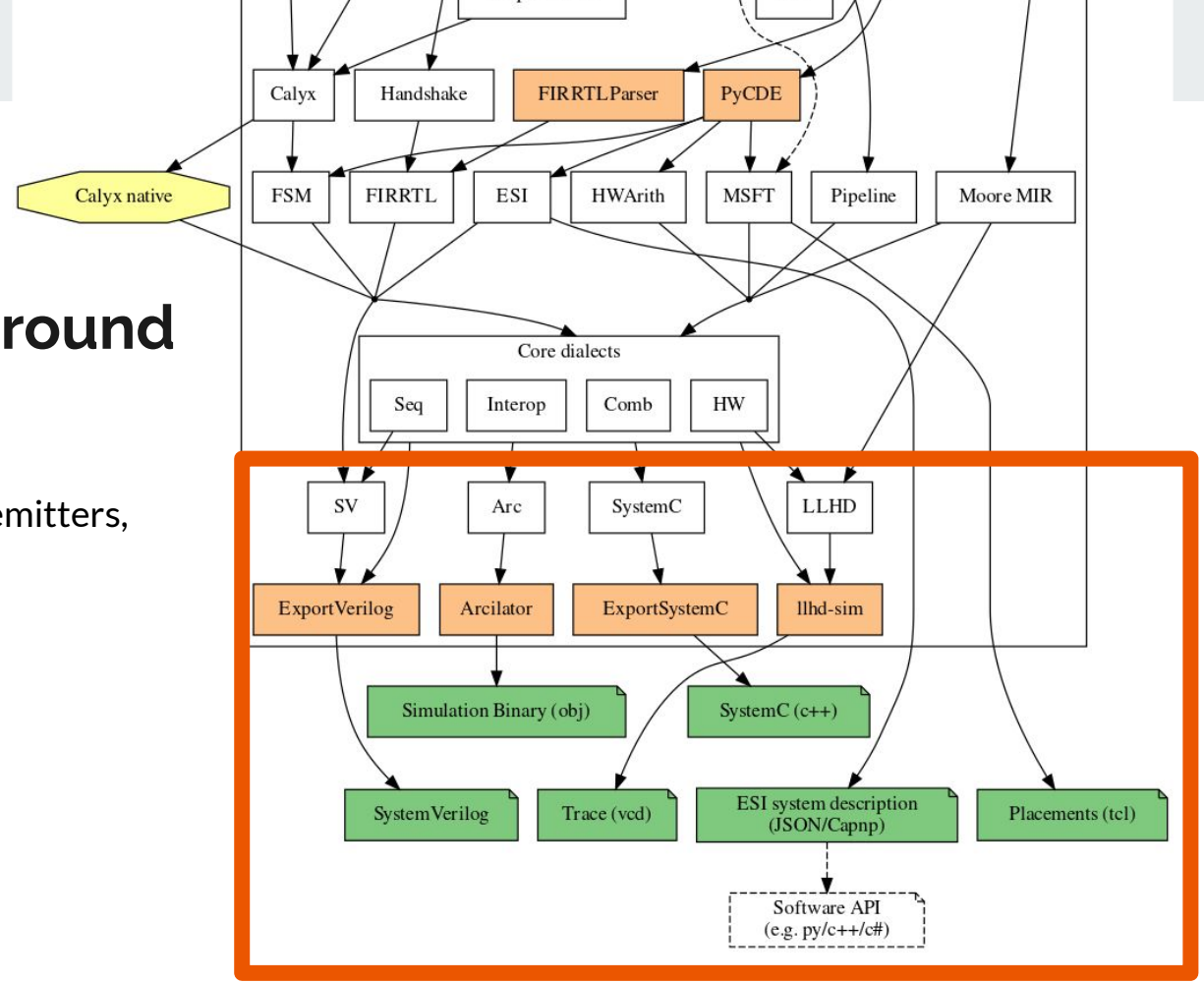


Background


Upstream frontends (selection)
Polygeist
PyTorch
Input languages
Chisel
Python
SV/VHDL
Upstream MLIR
SCF
Affine
Arith
CF
.fir
Moore
CIRCT
Scheduling
LoopSchedule
SSP
Calyx
Handshake
FIRRTLParser
PyCDE
Calyx native
FSM
FIRRTL
ESI
HWArith
MSFT
Pipeline
Moore MIR
Core dialects
Seq
Interop
Comb
HW
SV
Arc
SystemC
LLHD
ExportVerilog
Arcilator
ExportSystemC
llhd-sim
Simulation Binary (obj)
SystemC (c++)
SystemVerilog
Trace (vcd)
ESI system description (JSON/Capnp)
Placements (tcl)
Software API (e.g. py/c++/c#)

Background

Target Dialects, emitters, and targets.



Background

Example: Chisel compilation flow

⇒ Show `lower_to_hw` demo

Motivation

- Lack of **verification support** in CIRCT.
- Chisel had access to some formal verification before, but it's now deprecated.
- **Formal verification tools** are lacking for high-level hardware designs.
- Powerful tools, such as SVA sequences, are required to encode complex specifications for verification.
- Modern Verification engineers should also have access to **modern hardware verification tools**.

Formal Backend for CIRCT

- Goal: Convert CIRCT's front-ends into a format that can be used to perform Bounded Model Checking.
- How:
  - Convert CIRCT's core dialects into a BTOR2 form.
  - Serialize into a .btor2 file and feed it into btormc
- Difficulties:
  - Btor2 circuits are encoded as state transition machines
  - Btor2 format requires a particular handling of ports, assertions, and assumptions

Understanding Bounded Model Checking (BMC)

BMC: Convert circuit into a **state-transition system**, where each state is a combinatorial unrolling of the circuit for a given set of values assigned to its registers.

Verifying Combinatorial Circuits: Core of BMC, convert circuit and assertion into an SMT formula.

```
val a = IO(Input(UInt(32.W)))
val b = a + 1.U
assert(b > a)
```

```
// does there exist an assignment to a such that the assertion _fails_
solve(and(
    equal(b, add(a, lit(1))), // define b
    not(gt(b, a))             // can the assertion be violated?
))
```

IF CORRECT: UNSAT
ELSE: SAT + Counterexample

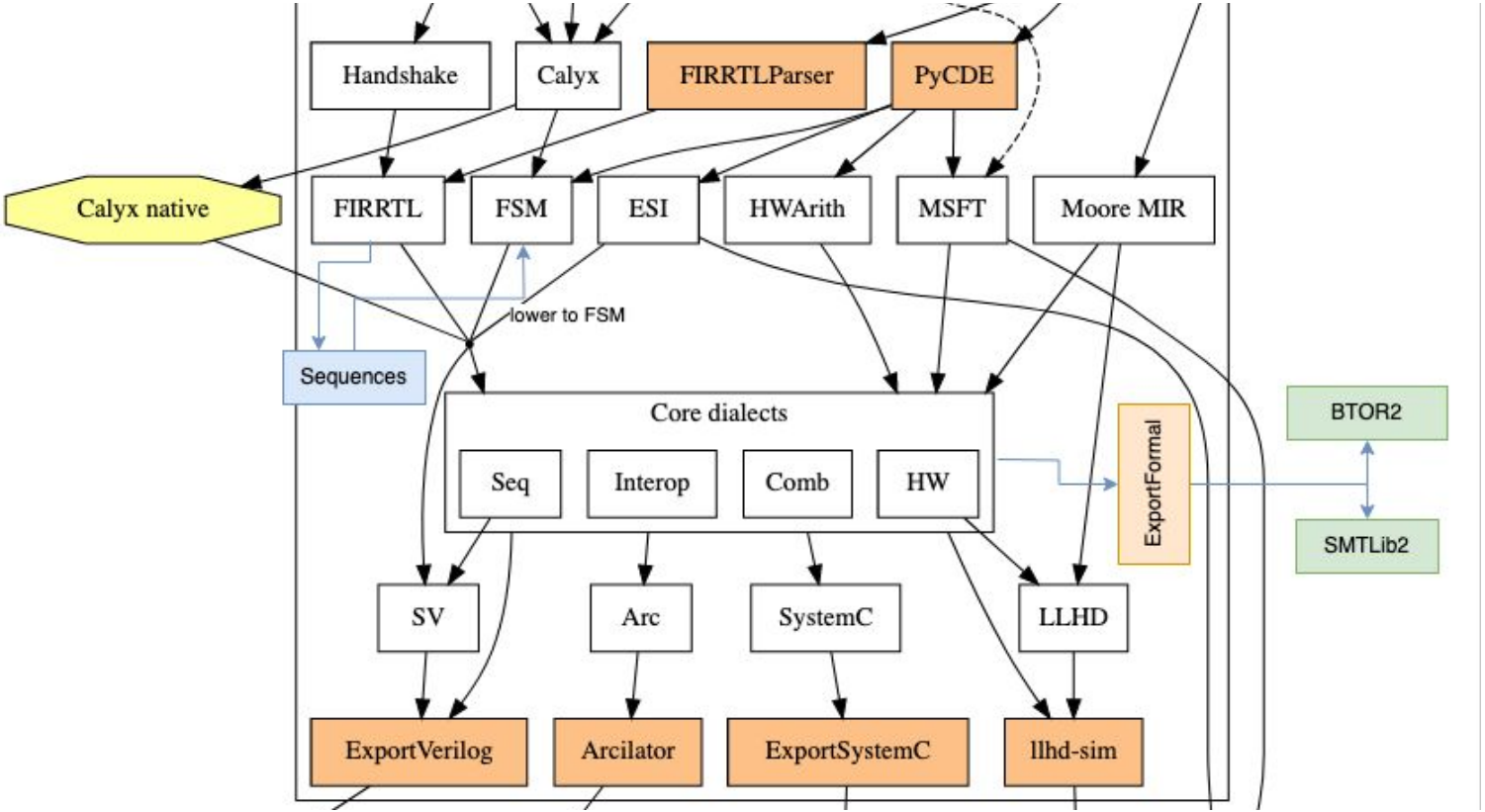
Demo: Verifying a real circuit through CIRCT

Let's see what a conversion of a combinatorial circuit would look through CIRCT down to btor2.

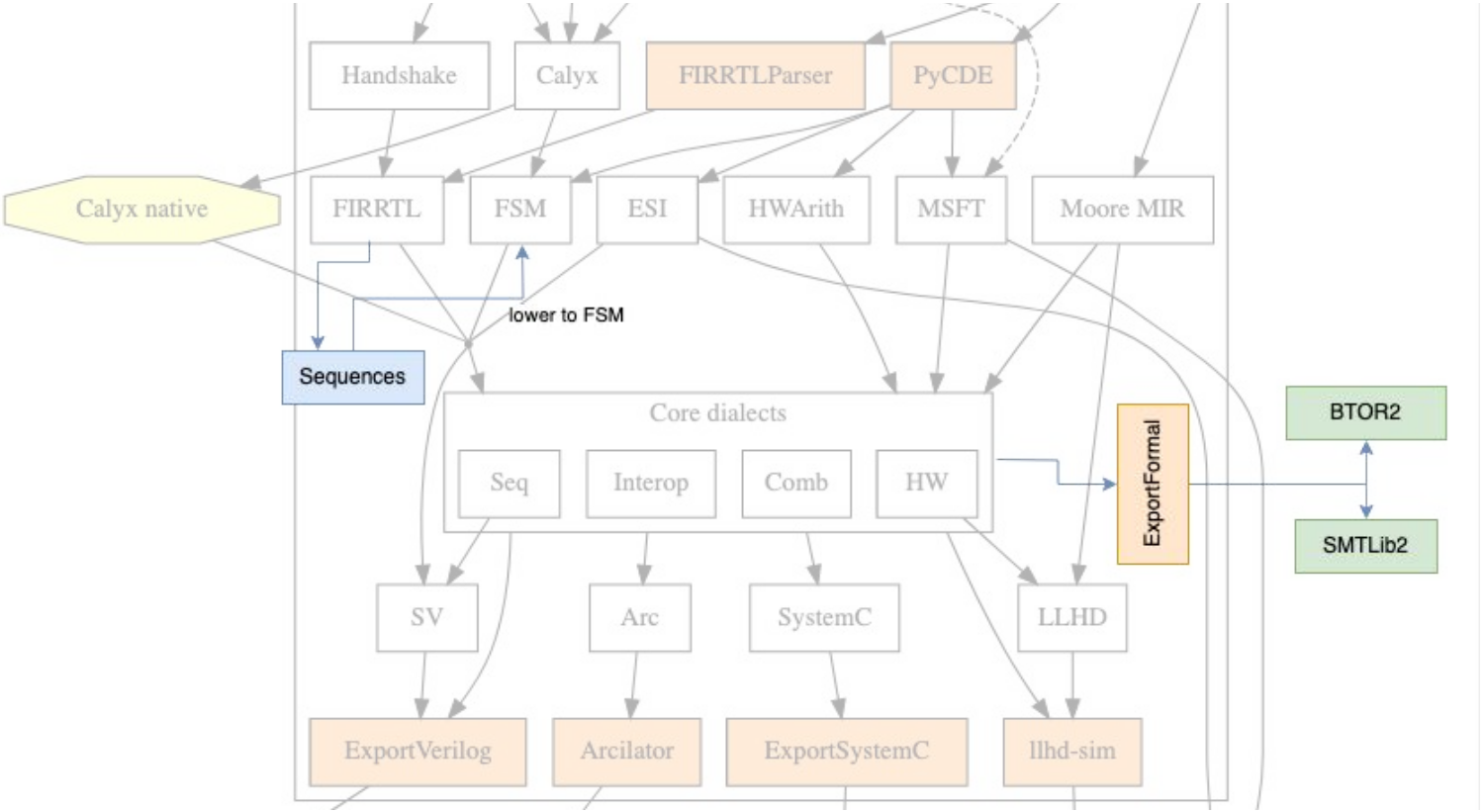
⇒ Show `lower_to_btor2` demo

Future Work: Temporal Assertions in CIRCT

- Goal: Enable the use of temporal assertions using a structure taken from SystemVerilog Assertion (SVA) sequences.
- How:
  - Create new dialect (**Sequences, or extend the recent LTL and Verif dialects**) to support encoding SVA-style sequences in CIRCT front-ends.
  - Translate sequences into an automata format, interpretable as a formal transition system.
  - Incorporate formal transition system into Formal backend.
  - Export to SystemVerilog directly or SMTLib2 and BTOR2 using Formal Backend.



Sequences Dialect in CIRCT



Sequences Dialect in CIRCT

Existing Tools

- SPOT is an existing tool for conversion from LTL to an automata format, which can be used for the sequence lowering to FSM.

- Recent work on LTL and Verif dialects, which implement the base of what I was thinking of for the first part of SVA. This could be greatly extended to implement more complex LTL features and lower to our Formal backend.

What has been done so far

1. **BTOR2** conversion has been implemented as a lowering pass from the **hw** dialect.
    a. This fully supports “all” operations for **combinatorial circuits** (only tested on firrtl).
    b. Supports the use of formal **assume** and **assert** statements (preconditions and postconditions of the circuit).
    c. Tested and works with **btormc**.
2. Work is ongoing to support sequential circuits, but state-transition system conversion is still incomplete.

Conclusion

- The goal is to enable verification within CIRCT.
- This would allows for bounded model checking to be done on circuits implemented in any of CIRCT's front-ends (including Chisel).
- This project is taking place from **October 2023 to April 2024**.
- Feel free to email me (adobis@berkeley.edu) if you want to learn more about this topic !

Resources

- Kevin Laeufer's Guest lecture on Formal Verification in Chisel:
  - https://github.com/agile-hw/lectures/blob/main/22-formal/lec22-formal.ipynb
  - Recording : https://www.youtube.com/watch?v=ssAbq5tdh8Y
- BTOR2 Format:
  - https://link.springer.com/chapter/10.1007/978-3-319-96145-3_32
- CIRCT:
  - https://circt.llvm.org/docs/GettingStarted/

Any Questions ?