Formal Verification of Hardware using MLIR

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

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

Overview

Motivation

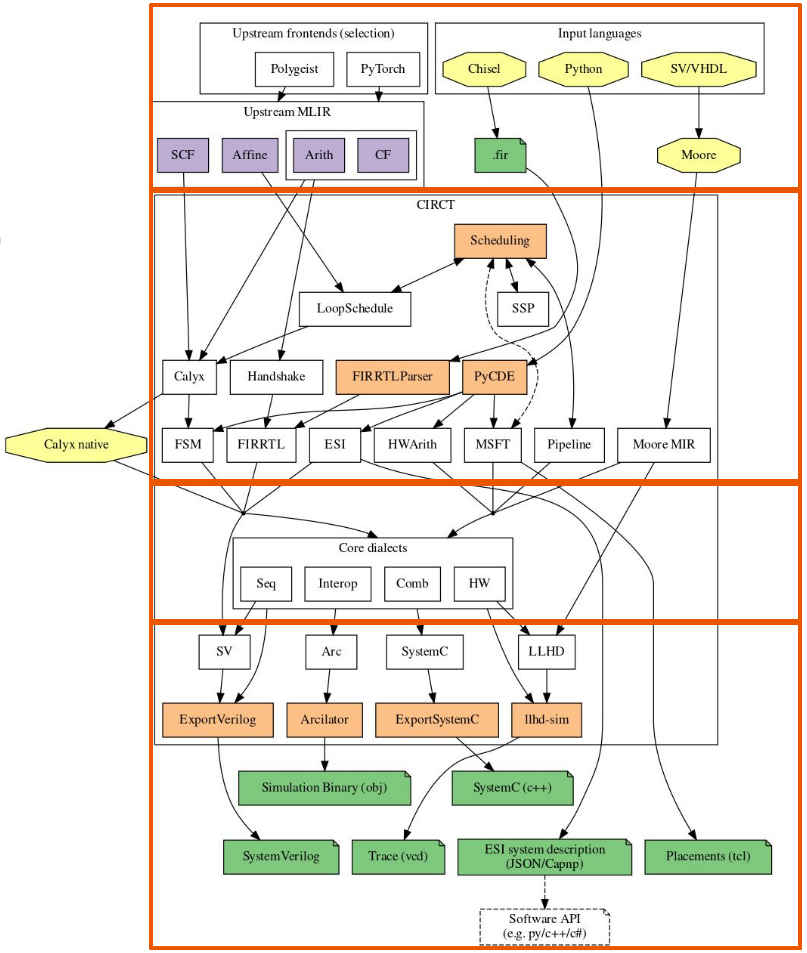
- Lack of **open-source** verification support for Linear Temporal Logic (LTL).
- Verification needs to go through **external commercial tools** for SystemVerilog.
- **SVA sequences** are required to encode complex specifications for verification.
- Need for an open-source solution that supports all of this directly out of CIRCT.

Background

Background: CIRCT

- Circuit IR Compilers and Tools.
- MLIR-based Hardware compilation framework.
- Supports many front-ends and backends (usually SystemVerilog).
- Current compiler backend for Chisel.

Background



Front-ends

Conversion / Optimization dialects

Core Dialects (generalized representation of hardware)

Targets

Background: Formal Verification

- Idea: Convert design into a mathematical model that we can use to prove assertions.
- Based around **SMT Solving:**
    - Find a possible assignment to variables in a formula to allow an assertion to hold
- For hardware: Bounded Model Checking

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)
```

$\Rightarrow$

```
// 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

My Work: Formal Verification with CIRCT

My Work: Formal Verification with CIRCT

- Goal: Enable Formal Verification in CIRCT front-ends using BMC and complex specifications.

- Two main parts:
  1) Formal Backend for CIRCT
  2) Enable Linear Temporal Logic (LTL) in the formal backend

1 - Formal Backend for CIRCT

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 **state transition system**
  - Serialize into a .btor2 file and feed it into btormc

Formal Backend for CIRCT

Chisel:

```
class Counter extends Module {
    val en = IO(Input(Bool()))
    val count = RegInit(0.U(32.W))
    when(en && count =/= 22.U) { count := count + 1.U }
    assert(count =/= 10.U)
}
```

(simplified) btor2:

```
0 sort bitvector 1
1 sort bitvector 32
2 input 0 en
3 constd 1 22
4 constd 1 1
5 state 1 count
6 neq 0 5 3         ;count != 22
7 and 0 2 6         ;en && (count != 22)
8 add 1 5 4         ;count + 1
9 ite 1 7 8 5       ;`7` ? count+1 : count
10 next 1 5 9       ;count := `9`
11 constd 1 10
12 neq 0 5 11       ;count != 10
13 not 0 12         ;!(count != 10)
14 bad 13           ;solve(!count != 10)
```

Formal Backend for CIRCT

- Result: Core to BTOR2 conversion pass was added to CIRCT.
  - Use: `circt-opt --convert-hw-to-btor2 <name>.mlir`
  - Converts any HW mlir description to a btor2 file.
  - Btor2 file can be given to btormc for bounded model checking.

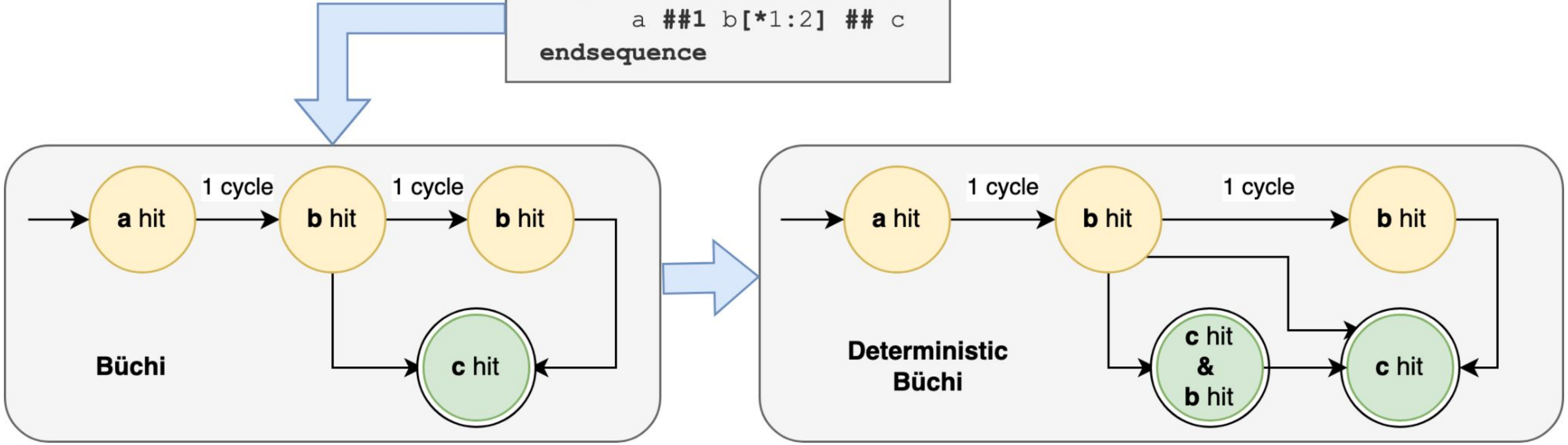
2 - SVA sequences with CIRCT

SVA sequences with CIRCT

- Goal: Enable SVA sequences to be expressed in CIRCT's formal backend.
- How:
  - Conversion of **LTL** dialect into a **Büchi automaton** form.
  - Optimization and node collapse in automaton form.
  - Conversion from Büchi automaton to **FSM**.
  - Conversion from FSM to **synthesizable hardware.**
- **This part is still in progress.**

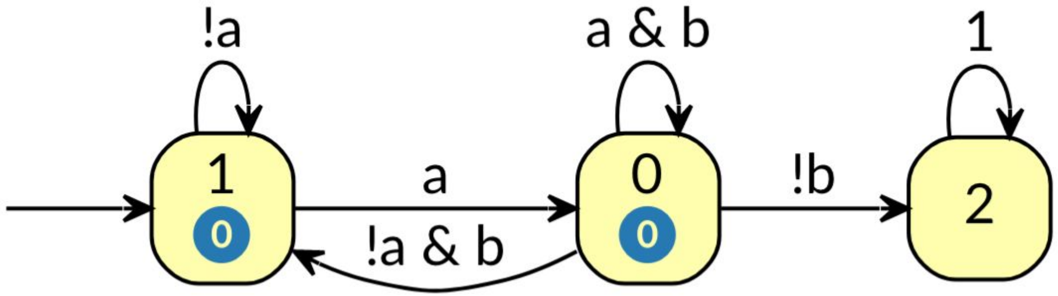
Theory: SVA Sequence to Automata Conversion

```
sequence a_seq_b12_seq_c
      a ##1 b[*1:2] ## c
endsequence
```



Theory: Less trivial conversions

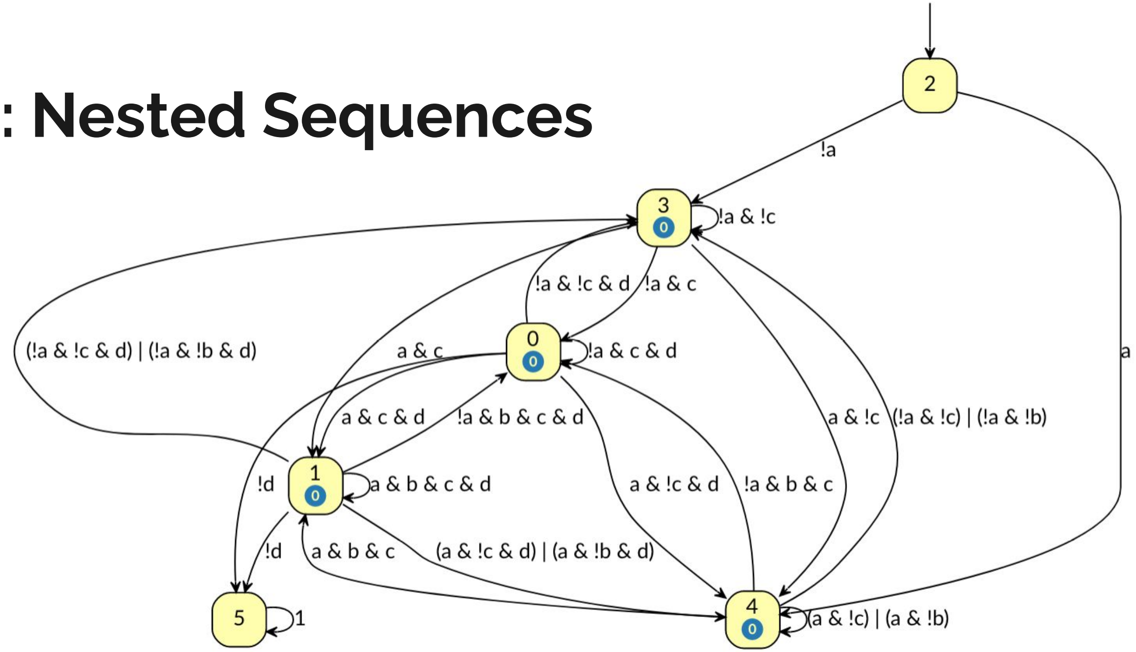
- Encoding delays as automata is simple (basically a shift register).
- How do we encode more complex relations?
    - Ex: non-overlapping implications: `a |=> b`
        - *a implies that b holds 1 cycle later* `a ##1 b | ~a`



Theory: Nested Sequences

- Problem: Sequences can be nested
  - What if both a and b are complex expressions
    - Ex: `(a |=> b) |=> (c |=> d)`

Theory: Nested Sequences



- *Generated with SPOT (https://spot.lre.epita.fr/app/)*

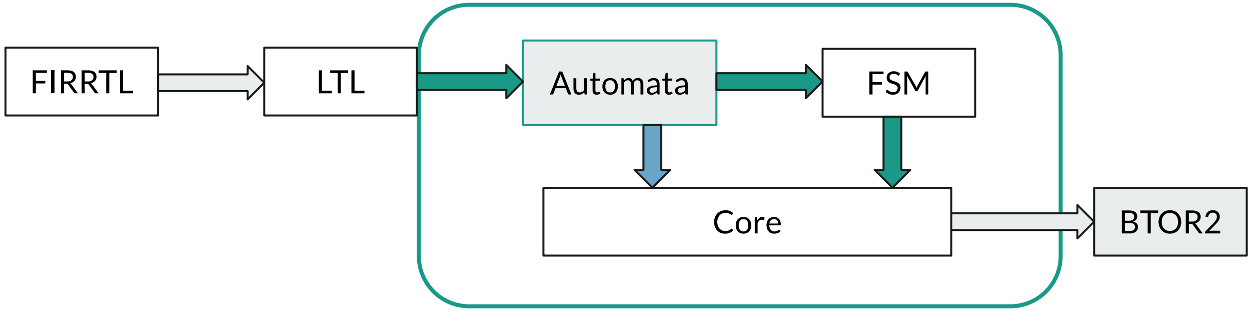
In Practice: General Translation of LTL to Automata

- Idea: Split nested sequence into simple sub-sequences and merge their automata together (mix of Yosys's technique and *[1]*).
    - Create basic sequences out of a nested sequence.
    - Convert the basic sequences into automata.
    - Define **merge operations** for operations connecting two automata.
    - Iteratively merge all sub-sequences in a DFS order.
    - Collapse duplicate nodes and transitions.

[1] Javier Esparza, Jan Křetínský, and Salomon Sickert. 2020. A Unified Translation of Linear Temporal Logic to ω-Automata. J. ACM 67, 6, Article 33 (December 2020), 61 pages. https://doi.org/10.1145/3417995

In Practice: Implementation in CIRCT

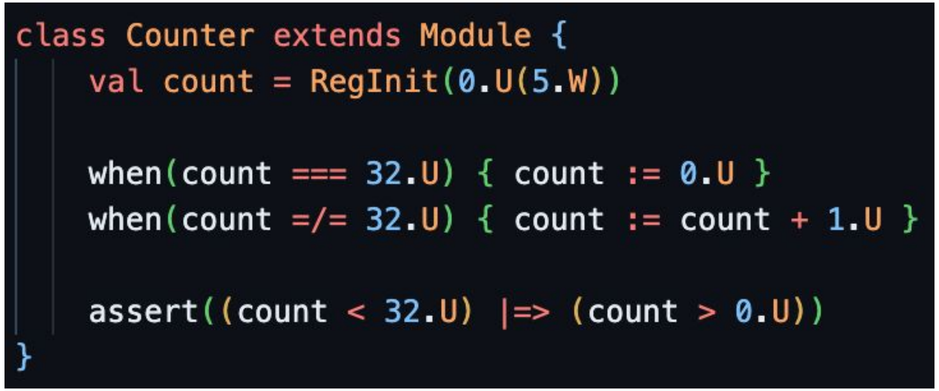
- Introduce **new dialect** to encode arbitrary automata.
- Add **conversion from LTL to Automata** (see previous slide).
- Add **lowering from Automata to FSM** (make deterministic).
- Add **convert FSM to Core** (generate registers).
- (*maybe*) direct conversion from Automata to Core.



Conclusion

Conclusion: What is this for?

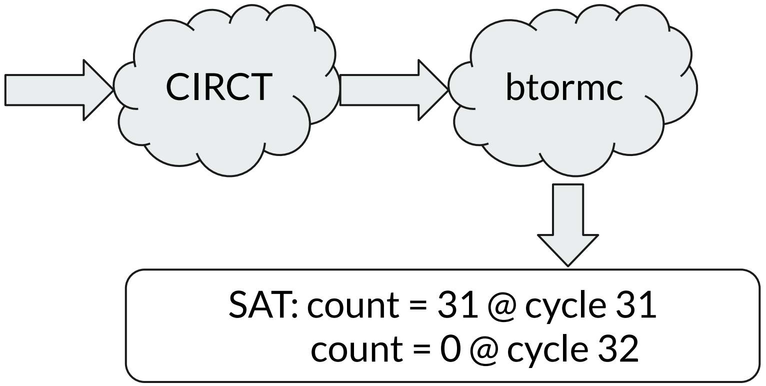
- Enables Complex specification-based verification to be done in high level hardware languages like Chisel.



```
class Counter extends Module {
    val count = RegInit(0.U(5.W))

    when(count === 32.U) { count := 0.U }
    when(count =/= 32.U) { count := count + 1.U }

    assert((count < 32.U) |=> (count > 0.U))
}
```



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/
- SPOT:
  - https://spot.lre.epita.fr/app/
- Unified Translation of Linear Temporal Logic to ω-automata
  - https://dl.acm.org/doi/abs/10.1145/3417995

Any Questions ?