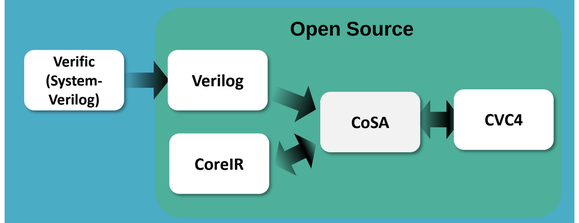
CoSA: Integrated Open-Source Formal Verification

Cristian Mattarei, Makai Mann, Aina Niemetz, Clark Barrett

Motivation

Agile Hardware Verification



- The Agile Hardware project has several verification needs
  - Functional correctness of the CGRA
  - Behavioral correctness of CoreIR designs after optimization passes
  - Correct mapping of applications to the CGRA
- CoreIR Symbolic Analyzer (CoSA) developed to address these challenges and provide open-source model checking
- Optional frontend Verific for full SystemVerilog support

Formal Representation

Symbolic Transition System (STS)

A *Symbolic Transition System* is a tuple $S = \langle V, I, T \rangle$ where V is a set of (input V_I, state V_S, and output V_O) variables, $I(V)$ is a formula representing the initial states, and $T(V, V')$ is a formula representing the transitions. A *state* of S is an assignment to the variables V.

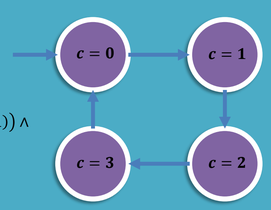
Synchronous Product of STSs

Given two Symbolic Transition Systems $S_1 := \langle V_1, I_1, T_1 \rangle$ and $S_2 := \langle V_2, I_2, T_2 \rangle$, the synchronous product S of S_1 and S_2, namely $S_1 \times S_2$, is defined as $S := \langle V_1 \cup V_2, I_1 \wedge I_2, T_1 \wedge T_2 \rangle$.

STS Example: Counter from 0 to 3

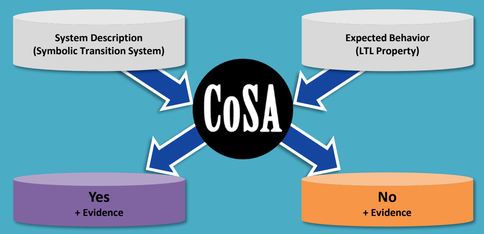
$\boldsymbol{S = \langle V, I, T \rangle}$ *where:*

- $\boldsymbol{V := \{c\}}$,
- $\boldsymbol{I(V) := c = 0}$,
- $\boldsymbol{T(V, V') := \big((c < 3) \rightarrow (c' = c + 1)\big) \wedge ((c = 3) \rightarrow (c' = 0))}$



Formal Analysis

Symbolic (LTL) Model Checking



Given a system model $\boldsymbol{M}$, and its expected behavior $\boldsymbol{\varphi}$, the CoSA model checker can solve the problem $\boldsymbol{M \models \varphi}$, showing if the system is compliant with the expected behavior

Linear Temporal Logic (LTL) Examples

Safety: $\boldsymbol{G\varphi}$, $\boldsymbol{\varphi}$ holds for every state

Finally: $\boldsymbol{F\varphi}$, $\boldsymbol{\varphi}$ finally holds

Liveness: $\boldsymbol{GF\varphi}$, $\boldsymbol{\varphi}$ holds infinite many times

Processor Verification

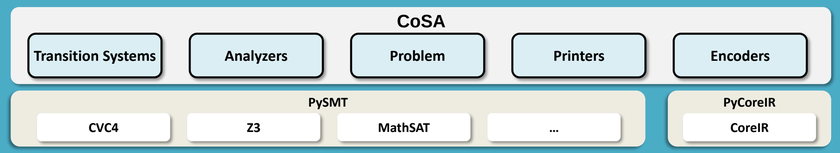
CoSA can be used as a back-end for the Symbolic QED family of techniques. [10]

Symbolic QED (SQED)
Verify a processor core using only the ISA (without a property). SQED checks for any inconsistencies in interleaved, repeated computation. This technique can find any logical bug consisting of a sequence of instructions, limited only by the underlying formal tool. Another approach can be used to find single-instruction bugs.

Electrical QED (EQED)
Identify manufacturing errors in a taped-out chip. CoSA supports this approach by incorporating fault analysis, which looks for traces while allowing some elements to have faulty behavior. This is crucial for identifying the cause of failed assertions in silicon that passed in simulation.

CoreIR Symbolic Analyzer (CoSA)

Architecture



CoSA is written in Python and builds on top of PySMT [4], which provides a solver-agnostic Python library to interface with SMT solvers. The internal architecture of CoSA is divided into the following parts:

- **Transition Systems**: defines the internal representation of the model, which is based on a hierarchical set of Transition Systems;
- **Analyzers:** implements the logic responsible for solving a verification problem. This includes BMC engines and liveness checking;
- **Problems:** used to define and manage the status of a verification problem;
- **Printers:** provides support for trace printing (i.e., textual or VCD format), and model translation such as the generation of a Simple Symbolic Transition System;
- **Encoders:** responsible for encoding different model descriptions into the internal representation. This includes interpreting CoreIR models, and extracting additional information used to optimize the verification process.

Functionalities and Analyses

- **Safety Verification:** standard invariant verification with proving capabilities
- **LTL Verification:** support for all temporal operators, with specialized algorithms
- **Equivalence Checking:** synchronous product between systems under analysis and reduction to safety verification
- **Lemma Verification:** automated check if the model satisfies the lemmas, and integration with the verification part
- **Clock Abstraction:** skips neg-edge steps in case of system with only pos-edge registers
- **Synchronous Input Models:** multiple models are combined into a synchronous product in order to simplify the analysis
- **Automated Lemma Extraction:** interaction with CoreIR in order to improve the equivalence checking performance

	Bounded	Unbounded	Engines
Safety (e.g., $\boldsymbol{G\ \varphi}$)	✓	✓	BMC [2], K-Induction [19], Interpolation [5]
Finally (e.g., $\boldsymbol{F\ \varphi}$)	✓	✓	BMC [2], K-Liveness [9]
Liveness (e.g., $\boldsymbol{GF\ \varphi}$)	✓	✓	BMC [2], K-Liveness [9]
Nested Operators (e.g., $\boldsymbol{G(\varphi\ U\ \gamma)}$)	✓	✗	BMC [2]

Input Formats

- **Verilog:** Standard Verilog HDL input handled natively
- **SystemVerilog:** Translated to Verilog using Verific [6]
- **CoreIR:** Supported with CoreIR and PyCoreIR (python bindings) [3, 7]
- **BTOR2:** A minimal text format for bit-vectors and one-dimensional arrays [8]
- **Symbolic Transition System:** Formal language for describing hardware systems
- **Simple Symbolic Transition System:** Same as Symbolic Transition System, but flattened
- **Explicit Transition System:** Assign concrete values to variables over time

github.com/cristian-mattarei/CoSA

Links and References

[1] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. Cvc4. In Proc. of the 23rd International Conference on Computer Aided Verification, 2011.
[2] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In International conference on tools and algorithms for the construction and analysis of systems, 1999.
[3] R. Daly. CoreIR: A simple LLVM-style hardware compiler. https: //github.com/rdaly525/coreir, 2017.
[4] M. Gario and A. Micheli. Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In Proc. of the 13th International Workshop on Satisfiability Modulo Theories, 2015.
[5] K. L. McMillan. Interpolation and sat-based model checking. In International Conference on Computer Aided Verification, pages 1–13. Springer, 2003.
[6] Verific Design Automation. http://www.verific.com/
[7] L. Truong. PyCoreIR : Ultralight Python bindings for CoreIR. https://github.com/leonardt/pycoreir
[8] A. Niemetz, M. Preiner, C. Wolf, A. Biere. Btor2, BtorMC and Boolector 3.0. CAV 2018
[9] K. Claessen and N. Sörensson. A liveness checking algorithm that counts. In Formal Methods in Computer-Aided Design. 2012.
[10] D. Lin, E. Singh, C. Barrett, S. Mitra. A Structured Approach to Post-Silicon Validation and Debug Using Symbolic Quick Error Detection

ISTC Agile