CoSA: Integrated Open-Source Formal Verification

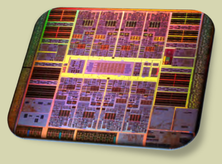
Clark Barrett (Stanford)

Designs Thrust — Posh Open Source Hardware (POSH)

Agile Hardware Design

Verification Challenge

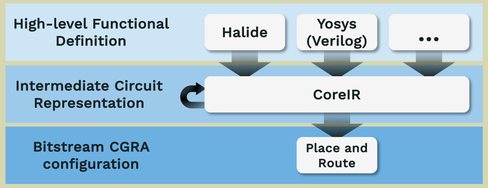
Systems on a Chip (SoCs)
- Growing in size and complexity
- Single chip contains multiple cores, caches, accelerators



SoC Verification
- SoCs used in critical applications
- Correctness is essential
- Failures can be extremely costly (Intel FDIV: $500M)
- Verification dominates design

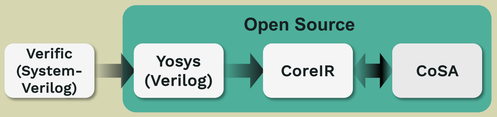
Formal Verification has potential to revolutionize verification
- Better than testing: covers all possible cases
- Already used extensively in industry
- But there are many challenges

Agile Hardware Flow



- Improve hardware design productivity with ideas from software
  - Reduce development time
  - Make process more 'agile'
  - Improve debugging tools
- Tailored for the image processing domain
- Start with a high-level description language for image processing, e.g., **Halide [18]**
- Targets a Coarse-Grained Reconfigurable Array (**CGRA**)
  - Similar to an FPGA but operates at the word level
  - Compose specialized heterogeneous tiles containing memories and dedicated processing elements (essentially ALUs).
- Uses **CoreIR** as a representation throughout the flow
- Allows for multiple front-ends such as **Yosys [24]**
- Inspire design re-use and innovation by providing open-source toolchain

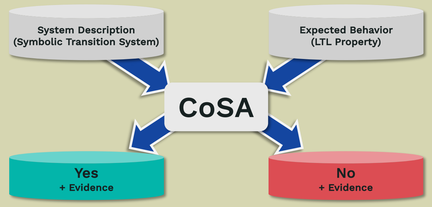
Agile Hardware Verification



- 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

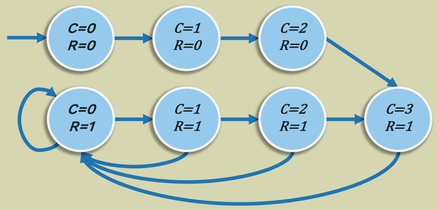
Formal Analysis

Symbolic (LTL) Model Checking



Given a (formal) system description, and its expected behavior (Linear Temporal Logic), the CoSA model checker can prove, or disprove, if the system is compliant with the expected behavior

Example of a Counter with Reset



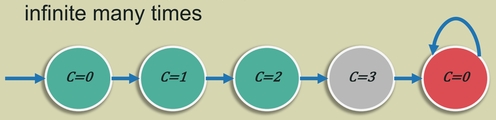
The counter counts from 0 to 3, and it can resets to 0 only when it has reached the value 3. The variable C represents the count value, while R is the enabling of the reset signal

Formal Verification Examples (LTL)

Safety: $G(R = 0)$, predicate $R = 0$ should hold for every state

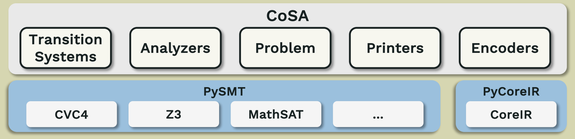
Finally: $F(c = 2)$, predicate $c = 2$ should finally hold (at any point in time)

Liveness: $GF(c = 2)$, predicate $c = 2$ should hold infinite many times



CoSA

Architecture



CoSA is written in Python and builds on top of PySMT [12], 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 an SMV file [8];
- **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., $G\ \varphi$)	✓	✓	BMC [5], K-Induction [19], Interpolation [15]
Finally (e.g., $F\ \varphi$)	✓	✓	BMC [5], K-Liveness [9]
Liveness (e.g., $GF\ \varphi$)	✓	✓	BMC [5], K-Liveness [9]
Nested Operators (e.g., $G(\varphi\ U\ \gamma)$)	✓	✗	BMC [5]

Additional Input Formats

```
VAR # Variables Definition
en_clr: BV(1);

INIT # Initial states
en_clr = 0_1;

TRANS # Transition Relation
(en_clr = 1_1) -> (next(en_clr) = 1_1);
(en_clr = 0_1) -> ((self.out > 5_16) <-> (next(en_clr) = 1_1));
```

Symbolic Transition System

Example of constraining the clear enable (en_clr) signal to be inactive until the output reaches 5.

```
# States
I:  reset = 0_1, reset_done = 0_1
S1: reset = 1_1, reset_done = 0_1
S2: reset = 0_1, reset_done = 0_1
S3: reset_done = 1_1

# Transitions
I -> S1
S1 -> S2
S2 -> S3
S3 -> S3
```

Explicit Transition System

Example of a reset procedure starting from the initial states and covering both pos-edge and neg-edge active registers. The signal reset_done informs for the correct execution of the procedure.

Case Studies

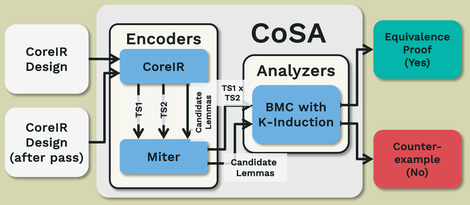
1. Hardware: Global Controller Verification

- Global controller handles configuration and debugging for a Coarse Grained Reconfigurable Array
- Implemented in Verilog
- Translated to CoreIR through the open-source VerilogToCoreIR plugin pass for Yosys **[13]**

Property	Result
Always return to **ready** state assuming counter delay < 10	T
When not in **ready** state, the counter always decreases	T
No underflow in counter	F
Read signal high implies the global controller is in **read** state	T
Write signal high implies the global controller is in **write** state	F

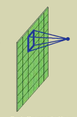
2. Software: Design Equivalence Checking

- Verify equivalence between CoreIR designs before and after optimization passes
- Fold-constants replaces any subgraph that has a constant value with a constant
- Difficult for traditional equivalence checking because the number of state elements can change
- Traditional Approach
  - Sequential Equivalence Checking
- Our approach
  - Leverage additional information from CoreIR
  - The pass generates candidate lemmas



3. Firmware: Verifying Applications

- 4x4 CGRA
  - 78,924 variables, including arrays and Bitvectors
  - Total of 443,376 bits and 4 arrays
- Application
  - 2x1 Convolution
  - Mapped to CGRA Primitives
  - Place and Route produces bitstream
- Memories encoded using the SMT Theory of Arrays
- CGRA implements linebuffer with 2 memories and complex logic
- Load bitstream into CGRA through CoSA using an Explicit State Synchronous System
- Verify Sequential Equivalence between original CoreIR application file and the configured CGRA
  - Equivalent in all executions up to 20 cycles (10 cycles of valid output)
- Barriers to inductive proof
  - Equivalence is not inductive
  - Cannot strengthen property with Array equivalence
    - CGRA uses 2 9-bit addressed memories for linebuffer
    - CoreIR application file implements linebuffer with a single 5-bit addressed memory



2x1 Convolution Example

Links and References

github.com/cristian-mattarei/CoSA

[1] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanovi'c, T. King, A. Reynolds, and C. Tinelli. Cvc4. In Proc. of the 23rd International Conference on Computer Aided Verification, 2011.
[2] C. Barrett, A. Stump, C. Tinelli, et al. The smt-lib standard: Version 2.0. In Proceedings of the 8th International Workshop on Satisfiability Modulo, 2010.
[3] C. W. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, et al. Satisfiability modulo theories. Handbook of satisfiability, 2009.
[4] 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.
[9] K. Claessen and N. So rensson. A liveness checking algorithm that counts. In Formal Methods in Computer-Aided Design. 2012.
[10] E. Clarke, K. McMillan, S. Campos, and V. Hartonas-Garmhausen. Symbolic model checking. In International Conference on Computer Aided Verification, 1996.
[11] R. Daly. CoreIR: A simple LLVM-style hardware compiler. https: //github.com/rdaly525/coreir, 2017.
[12] 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.
[13] D. Huff. Verilog to CoreIR translator. https://github.com/dillonhuff/ VerilogToCoreIR, 2018.
[15] K. L. McMillan. Interpolation and sat-based model checking. In International Conference on Computer Aided Verification, pages 1–13. Springer, 2003.
[17] J. Parkhurst, M. Horowitz, P. Hanrahan, and C. Barrett. AHA Agile Hardware Project. https://aha.stanford.edu/, 2018.
[18] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. ACM SIGPLAN Notices, 48(6):519–530, 2013.
[19] M. Sheeran, S. Singh, and G. Sta ̊lmarck. Checking safety properties using induction and a sat-solver. In International conference on formal methods in computer-aided design, 2000.
[21] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In Logics for concurrency, pages 238–266. Springer, 1996.
[22] A. Vasilyev, N. Bhagdikar, A. Pedram, S. Richardson, S. Kvatinsky, and M. Horowitz. Evaluating programmable architectures for imaging and vision applications. In 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 1–13, Oct 2016.
[24] C. Wolf, J. Glaser, and J. Kepler. Yosys-a free Verilog synthesis suite. In Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip), 2013.

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

THE ELECTRONICS RESURGENCE INITIATIVE