CircuitGraph: A Python package for Boolean circuits

Joseph Sweeney[1], **Ruben Purdy**[1], **Ronald D Blanton**[1], **and Lawrence Pileggi**[1]

1 Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213

DOI: 10.21105/joss.02646

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @skadio
- @prw99r
- @r-a-hoggarth

Submitted: 30 August 2020
Published: 04 December 2020

Summary

A Boolean circuit is a fundamental mathematical model ubiquitous in the design of modern computers. The model consists of a directed graph wherein nodes are logic gates with corresponding Boolean functions and edges are wires which determine the composition of the gates. `CircuitGraph` is a open-source Python library for manipulating and analyzing Boolean circuits.

Statement of need

Analysis, manipulation, and generation of Boolean circuits is fundamental to many aspects of digital hardware design, cryptography, constraint solving, and other areas. Highly optimized software for processing Boolean circuits exists. Unfortunately it generally is proprietary, with expensive license fees. Furthermore, these options suffer from poor documentation, are closed source, and typically rely on Tool control language (Tcl). While simple, Tcl is slow, has limited libraries and supporting community, and is unnecessarily verbose. These reasons motivate the development of our open source solution. While this software will directly benefit our lab as a research platform, it certainly has application in other environments such as the classroom.

Functionality

The functionality of `CircuitGraph` has been tailored to our research needs, however, the library is easily extensible to many other applications of Boolean circuits. In the following sub sections, we highlight some of the library's key features.

The core of the library is the `Circuit` class, which internally uses a `networkx.DiGraph` data structure from Hagberg et al. (2008). The class implements key Boolean circuit functionalities on top of the graph as we describe below.

Interfaces

Compatibility with existing systems is a primary goal for our library. Towards this end, we have built interfaces for a subset of Verilog, the most commonly used Boolean circuit format. This library supports generic stuctural Verilog which is the typical output of synthesis tools. Specifically, the library can parse combinational gates in the following forms. We also provide an interface to parse sequential elements.

```
assign a = b|(c^d);
xor(e,f,g);
```

Additionally, we have provided a library of generic and benchmark circuits that can be quickly instantiated.

```
import circuitgraph as cg
c0 = cg.from_file('path/circuit.v')
c1 = cg.from_file('path/circuit.bench')
c2 = cg.from_lib('c17')
```

Composition

A common issue found in similar tools is the poor expressivity of circuit construction primitives. We aim to build a simple, but powerful syntax for creating and connecting nodes in a circuit. The ease of our syntax is enabled by the Python language. An example of this syntax is below.

```
# add an OR gate named 'a'
c0.add('a','or')

# create an AND gate with circuit inputs in a single line. Input connections to th
c0.add('g','and',fanin=[c.add(f'in_{i}','input') for i in range(4)])
```

Synthesis

We provide an interface to common synthesis tools including yosys from Wolf (n.d.) and Cadence Genus. This allows the user to run basic synthesis routines on circuits from within Python. Specifically, we support the generic multi-level synthesis routines of both tools.

```
# synthesize circuit with yosys
c_syn = cg.syn(c0, "Yosys")
```

Satisfiability

Satisfiability is an essential problem related to Boolean circuits. Surprisingly, commercial synthesis tools do not directly support its use (although the open source tools yosys does). We add satisfiability to our library which in turn enables a wide array of analysis including sensitization, sensitivity, and influence. Our implementation utilizes pysat from Ignatiev et al. (2018). The main interface is simple allowing the user to determine satisfiability of a circuit under a set of variable assignments. To develop more complex routines, the user can also access the underlying pysat.solver instance. In conjunction with satisfiability, we provide interfaces to approximate and exact model count algorithms.

```
# check satisfiability assuming 'a' is False
cg.sat(c0,{'a':False})

# get number of solutions to circuit with 'a' False
cg.model_count(c0,{'a':False})
```

Future Work

We plan on adding support for the BLIF and Bench formats. Additionally, we may expand the compatibility with Verilog standards if a need is shown. Support for timing-based synthesis may be useful in some scenarios. Other improvements could interfaces to open source simulation and Automatic Test Pattern Generation (ATPG) tools.

Requirements

As previously mentioned, `CircuitGraph` relies on the `networkx` and `pysat` libraries. Additionally, it uses `pyverilog` to parse verilog netlists.

References

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), *Proceedings of the 7th Python in Science Conference (SciPy2008)* (pp. 11–15).

Ignatiev, A., Morgado, A., & Marques-Silva, J. (2018). PySAT: A Python toolkit for prototyping with SAT oracles. *SAT*, 428–437. https://doi.org/10.1007/978-3-319-94144-8_26

Wolf, C. (n.d.). *Yosys manual*.