2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
OCTOBER 26-27, 2021

Netlist Paths (extended abstract)

A tool for front-end netlist analysis

Jamie Hanlon, Samuel Kong, Graphcore Ltd, Bristol, United Kingom

Keywords—Python; SystemVerilog; Netlist; RTL; Front-End

I. Introduction

For an RTL design engineer, there is an abundance of EDA tools that assist and aid in the development of ASIC designs, including for example, simulation, linting, power analysis and synthesis. Typically, these EDA tools read in designs in the form of a Hardware Description Language (HDL) and elaborate them into internal, proprietary netlist models, representing the components and connectivity of the design. The internal representation is often made accessible to users for the purposes of the tool via a TCL Application Programming Interface (API) and/or graphical user interface (GUI). There is, however, an absence of a simple, generic tool that provides RTL design engineers programmatic access to a source-level netlist model, allowing them to inspect types, constants, variables and structure of a design.

In this paper, we shall discuss Netlist Paths, an open-source library and tool that enables the inspection of a netlist model of an RTL design using a Python API or command-line interface (CLI). We shall explore the benefits of using Netlist Paths and the applications it has in the chip design cycle, based on our experience of ASIC chip design at Graphcore. Applications including investigation of critical timing paths, quality-of-result (QoR) analysis and generation of documentation. First, we shall review existing, related works and how Netlist Paths builds upon, complements or differs from them.

II. Related Work

To analyse an RTL design, an engineer's first choice of tool will likely be one of the standard EDA simulators such as Synopsys VCS, Mentor Questa or Cadence Xcelium. These simulators provide powerful facilities to inspect the design for debugging conventionally with a GUI, which does not easily allow integration into custom tooling. Beyond these tools, an engineer may turn to synthesis tools such as Synopsys Fusion Compiler or Cadence Genus to analyse a design, but this is a more involved route with longer run times.

Addressing the issue of custom tooling, Synopsys offers a Native Programming Interface (NPI) to their Verdi platform, which provides TCL and C APIs to access the database produced during simulation [1]. NPI has interfaces to different aspects of the database, but in particular, a netlist model, which is the result of technology-independent, non-optimized synthesis. With this model, the structure of the design is accessible as low-level objects such as gates and memories connected via ports and nets. A drawback of using the NPI Netlist Model for source-level tooling is that correspondence to the original RTL source code is not directly maintained, and synthesis of RTL into combinatorial logic cells further complicates that correspondence.

There are several notable open-source tools that provide facilities for parsing and elaborating SystemVerilog: Verible (a project from Google) [2], Slang (SystemVerilog language services) [3] and Surelog [4][5] are all libraries intended to be used as a front-end for EDA tooling. Currently, their respective projects only include simple tools, e.g. for linting. Perhaps the best-known open-source tools are Verilator [6] and Yosys [7][8]. Verilator is a mature Verilog simulator, that is becoming increasingly widely used, particularly in the burgeoning open-source hardware community. Yosys is a Verilog synthesis framework geared towards FPGA development, but only supports Verilog 2005. The Netlists Paths tool described in this paper builds upon Verilator, chosen primarily because of its maturity as a simulation platform and known operability with Graphcore's SystemVerilog codebase, and uses it to parse and elaborate the design. However, the alternative open-source language frontends listed may have equally been used by Netlist Paths or could be in the future.

III. NETLIST PATHS

Netlist Paths reads an Extensible Markup Language (XML) representation of a design produced by Verilator, which is a single module with all tasks, functions and sub module instances inlined. The XML is used to build up a directed graph data-structure representing the design netlist, where nodes are logic statements and variables, and edges are data dependencies between them. Python is becoming an increasingly common part of the infrastructure for ASIC design and verification flows, so the library is written in C++ with a Python interface, making a good trade-off between performance and ease of use/integration.

The original motivation for developing Netlist Paths was to tackle the problem of corresponding critical paths identified during timing analysis in the Physical Design flow back to the RTL, since timing-critical paths become heavily optimised and often only retain their start and end point names. Having the ability to rapidly explore paths between the matching start and end points bridges the information gap created by synthesis, making it easy to locate exactly how a critical path traverses through design RTL. Once a critical path has been located, potential fixes to shorten or break it can be quickly validated, rather than re-running the full physical build cycle.

A related issue with the interaction between the Logical and Physical Design teams is in the specification of parts of a design for QoR reporting, such as metrics on groups of registers and paths. Historically, many ad-hoc patterns are maintained in Physical Design scripts for identifying these groups to drive their QoR reporting, but as a design changes these patterns inevitably become stale and cause the reporting to break, often in a non-obvious way. A better arrangement is to maintain the patterns with the RTL, so that they can be updated as the design changes. Netlist Paths provides a simple way to check QoR group patterns against the design, for example by asserting the existence of registers and particular timing paths, and that the specified patterns provide good coverage of the design.

In the Graphcore Front-End environment, we have been able to use Netlist Paths to perform checking on Python specifications of design components that are used for code generation and documentation, to ensure that the specifications match the implementations and vice versa. These tests are run as part of our Continuous Integration (CI) test-suite with minimal impact on the overall run-time. By making use of the Python API, Netlist Paths has provided versatility and ease-of-use that cannot be found in existing EDA tools and as a result, has improved RTL code quality and engineering productivity.

IV. CONCLUSIONS

We have presented Netlist Paths, a lightweight, open-source tool that provides RTL design engineers easy access to a source-level netlist model of their design through a Python API and CLI. The combination of these features enables the straightforward creation of custom tooling for Front End workflows, without any of the constraints of EDA licensing. We have illustrated how we have leveraged this ability at Graphcore to improve the interface between the Logical and Physical design teams for fixing critical timing paths and specifying QoR groups, and within the Logical Design team to support Python specification of design modules for code generation and documentation.

REFERENCES

[1] Synopsys VC Apps: Native Programming Interface (NPI), Version Q-2020.03, March 2020.

[2] Google Verible, https://google.github.io/verible

[3] Slang, https://sv-lang.com

[4] Alain Dargelas, Henner Zeller. Universal Hardware Data Model, Workshop on Open-Source EDA Technology (WOSET) 2020.

[5] Surelog, https://github.com/alainmarcel/Surelog

[6] Verilator, https://www.veripool.org/verilator

[7] Clifford Wolf, Johann Glaser. Yosys - A Free Verilog Synthesis Suite. In *Proceedings of Austrochip 2013.*

[8] Yosys, http://www.clifford.at/yosys