ROHD-HCL: Configurable, Reusable Hardware Components

Max Korbel
Intel Corporation
Santa Clara, California, USA

Desmond Kirkpatrick
Intel Corporation
Hillsboro, Oregon, USA

Yao Jing Quek
Intel Corporation
Penang, Malaysia

ABSTRACT

The ROHD Hardware Component Library (ROHD-HCL) is a free and open-source library of high-quality, configurable, well-documented design and verification components developed using the Rapid Open Hardware Development (ROHD) framework. The library aims to address the challenges of hardware reuse across the industry and provide examples and a foundation for hardware developers to build upon. The library also includes a configuration methodology and web-app to make components even to easier to use.

1 INTRODUCTION

One major difficulty in hardware design is the limited reuse of components. Existing open-source libraries (e.g. BaseJump STL [5]) often suffer from the typical challenges of industry-standard SystemVerilog methodologies. Others comprise mostly utilities or language extensions without full tests, documentation, nor illustrate how to add more reusable and composable components [1][2].

The hardware world needs high-quality, configurable, reusable, pre-validated, nicely-packaged, well-documented components that can be seamlessly dropped into designs. The ROHD Hardware Component Library (ROHD-HCL) provides this by leveraging the ROHD framework, a Dart-based hardware generator framework that enables modern hardware design and verification [3]. This separate library uniquely comprises a collection of reusable, configurable, composable, unit tested components for both design and verification that can be utilized in diverse designs, and also serves as a source of practical examples of ROHD hardware implementations. ROHD-HCL is offered as free and open-source software with a permissive license at https://github.com/intel/rohd-hcl. The library also includes a methodology for component configuration and a web-app which leverages it.

The library has over 70 components in categories such as:

- Encoders & Decoders
- Arbiters
- FIFOs & Queues
- Find, Count, & Detect
- Sort
- Rotate
- Arithmetic
- Error handling
- Data flow
- Memory
- Standard interfaces
- Models

2 HARDWARE COMPONENTS

Design component development in ROHD-HCL is guided by several principles. Components should be general, easily reusable, and as configurable as may be useful. They must be extensively tested and come with excellent documentation and examples. The first component in a category should be the simplest, and the focus should be on the breadth of component types before depth. Components should focus on correctness first, then optimization, which ensures that functionality is not compromised in the pursuit of performance. This approach is facilitated by having tests in place to protect functionality.

Because the library is built with ROHD, it is possible to use object-oriented programming (OOP) approaches to help make components consistent, succinct, and interchangeable. Error checking components in the library offer a good example. An abstract definition for error checking transmitters and receivers defines the API requirements and provides automation which can be reused for any implementation (ports, configuration, etc.). Implementations such as Hamming ECC and Parity offer different capabilities for error checking and correction with trade-offs on transmission size and hardware cost. A user of the library could specify that some communication between devices requires error checking in a generic way, and leave the implementation flexible. Thus, the same hardware design could flexibly support either parity checks, ECC, or any other error checking implementation which is compliant with the abstract API requirements. Some types of arbiters, encoders, and decoders have a similar structure.

Parallel prefix or "scan" trees are useful for efficient implementation of computations which involve associative operators such as encoding, or-reduction, and addition. By leveraging advanced programming idioms, like functors, allowing for passing of functions which generate prefix trees into an adder generator, a wide variety of adders are supported by the library. For example, tree patterns defined by ripple, Sklanksy, Kogge-Stone, and Brent-Kung are included which gives those four varieties of adders, but also the same four varieties of or-reduction and priority encoding.

A powerful feature of component libraries is composability, and ROHD-HCL expands upon wire-level connections from ROHD by including handshaking automation (e.g. via a ready-valid protocol) in a composable type-safe manner compatible with linear dataflow as well as fan-in and fan-out arbitration on a subset of components.

3 STANDARDIZED INTERFACES

Today's standardized interfaces are usually just specification documents. Implementations vary in naming and compatibility and often require additional mapping for interconnections. Furthermore, using advanced features of SystemVerilog, such as interfaces, modports, structs, unpacked arrays, etc., may not be supported by tools handling the connectivity. ROHD offers a solution to these challenges by supporting Interface abstractions which are simple to connect, can contain configuration information, automate uniquification, support hierarchical definitions, and convert to generated SystemVerilog that is simple, lint-clean, and easy for tools to read.

LATTE '24, April 28, 2024, San Diego, CA, USA

The ROHD-HCL library takes this a step further by providing centralized, spec-compliant implementations of standardized interfaces. These implementations support configurations from the specification and include automation and features from ROHD interfaces. ROHD-HCL also includes simulation-time interface compliance checkers, further enhancing the reliability of the interfaces.

4 VERIFICATION COMPONENTS

While much of the focus of hardware development often lies in the design of components, the importance of verification components cannot be overstated for ensuring functionality and reliability.

Although the components in the library are already tested, checkers for components in the library provide an additional layer of assurance by ensuring proper usage of the components. For example, a FIFO checker, easily attached to any instance, detects underflow or overflow scenarios which could indicate errors in the surrounding design.

Trackers are another essential verification component for logging transactions across interfaces. However, developing trackers can be a time-consuming process and is often overlooked in all but the most crucial interfaces. ROHD-HCL includes monitors and trackers for standard interfaces and some components.

Bus Functional Models (BFM) are another crucial verification component. They are indispensable for testing hardware devices, providing a high-level abstraction of a device's behavior and a well-defined API for driving stimulus. ROHD-HCL includes models for standard interfaces and common use cases which are typically only available from vendors and only compatible with costly tools.

Configuration information can be collected from ROHD Interface objects, so configuration alignment between verification and design components becomes automatic. Additionally, just like the design components, these verification components are high quality, heavily tested, and well-documented.

5 CONFIGURATION

Hardware components in ROHD-HCL each come with a Configurator, an object with a standard software API for enabling common interactions with components. A Configurator implementation is responsible for defining a set of configurable "knobs" which control generation of a ROHD module instance.

In exchange for these implementations, the base Configurator class adds functionality to generate SystemVerilog as well as save or load configuration in JSON format. A Configurator enables applications to generically support interactions with components. For example, Configurators are used for the ROHD-HCL web-app shown in Figure 1. A component registry, which enumerates all the ROHD-HCL Configurators, is queried to generate a list of components that can be generated within the web-app. The web-app enables even non-ROHD developers to easily configure and obtain SystemVerilog component implementations. The web-app and Configurator classes have stand-alone definitions, enabling applications whereby non-library components could be configured and generated as well.

Configurators and the registry are also used to generate SystemVerilog for the purpose of generating synthesized schematic web pages (leveraging Yosys [6] and d3-hwschematic [4]), which

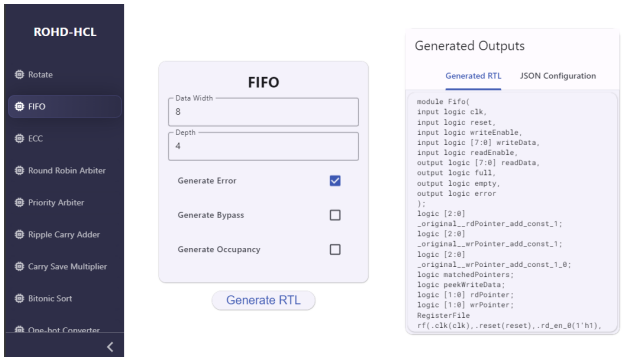

Figure 1: The ROHD-HCL Configuration Web-App for configuring and generating components interactively.

are automatically built and hosted as part of the continuous deployment (CD) flow on GitHub.

6 FUTURE WORK & CONCLUSION

Expansion of the library is a primary area of focus. This includes increasing the number of components for both design and verification, as well as diversifying the types and categories of components. Performance optimization for area, power, performance, clock speed, and other factors as well as improvements in flexibility and configurability are other areas of future work. Existing unit tests will ensure that improvements do not compromise functionality.

Interactive component development and inspection is another area of future work, building upon the web-app. Flutter offers a low-effort way to launch the app while developing components, giving developers and contributors a way to "see" their component as they work on it (e.g. dynamic hierarchy, schematic, and simulation waveform generation). Other in-progress development for debug tooling and waveform viewing in ROHD will also aid in these areas.

Finally, improving documentation is a continuous process and crucial for easy usage and increased adoption. The goal is to make the ROHD-HCL library as user-friendly and accessible as possible. Attracting both increased usage and contributions is crucial for the accelerated growth of the library. More users will inevitably improve the quality and breadth of the library, attract more contributions, and build trust in the library in a positive feedback loop.

ROHD-HCL has established a solid framework for building families of reusable, and validated, configurable components using the strong capabilities of the Dart language and ROHD. The main metric of success for ROHD-HCL is successful usage of the library for real designs. Within Intel, multiple real projects are leveraging ROHD-HCL. ROHD-HCL is hosted on the public pub.dev Dart package manager, where it is in the top two-thirds of all Dart and Flutter packages downloaded worldwide. Informal announcement posts for ROHD-HCL on social media have garnered tens of thousands of impressions, hundreds of reactions, and dozens of comments and reposts.

ROHD-HCL is a step towards a world where anyone can freely leverage high-quality components for their hardware development and focus on adding new value.

REFERENCES

[1] [n.d.]. Amaranth. https://github.com/amaranth-lang/amaranth
[2] [n.d.]. SpinalHDL. https://github.com/SpinalHDL/SpinalHDL
[3] Max Korbel. 2022. Rapid open hardware development framework. In *Proc. Workshop Open-Source EDA Technol.*
[4] Nic30. [n.d.]. https://github.com/Nic30/d3-hwschematic
[5] Michael Bedford Taylor. 2018. Basejump STL: Systemverilog needs a standard template library for hardware design. In *Proceedings of the 55th Annual Design Automation Conference*. 1–6.
[6] Claire Wolf. [n.d.]. Yosys Open SYnthesis Suite. https://yosyshq.net/yosys/.