Application of Sketch Guided Synthesis to Runtime Reconfigurable FPGA Primitives

VISHAL CANUMALLA, University of Washington, USA

FPGA technology mapping is a critical step in hardware compilation; from high-level hardware design descriptions, toolchains must find an equivalent low-level implementation on the target FPGA's primitives. FPGAs are becoming increasingly heterogeneous via the addition of new, highly configurable primitives with complex behaviors. Effectively utilizing these primitives can produce orders of magnitude better performance, meaning robust technology mapping is more important than ever. Current automated approaches for technology mapping work for simple primitives, but fall short when tasked to map modern complex primitives. This is because configuring these primitives requires tools which can understand and utilize their complex behavior, such as pipelined arithmetic or runtime reconfigurability. Put simply: configuring modern, complex FPGA primitives is equivalent to synthesizing programs. Consequently, ongoing work—Lakeroad—applies program synthesis to address this mapping gap by using semantics extracted from Verilog descriptions of primitives. However, Lakeroad's compilation flow still requires adding support for new primitives to the tool, and it is unclear whether this approach will be extensible to increasingly diverse complex primitives.

In this paper, we identify the `CFGLUT5` as a primitive that thoroughly demonstrates the extensibility of Lakeroad while also highlighting the limitations of the approach as of now. `CFGLUT5` is a specialized primitive by Xilinx, that has behavior currently unrealized in Lakeroad's primitive library. The difficulty of mapping to the primitive is evident, as it is not supported by any compiler, including Xilinx's own proprietary toolchain Vivado. We show the promise of Lakeroad's approach by mapping a variety of common operations to `CFGLUT5`, and elaborate on ongoing work and limitations.

CCS Concepts: • **Hardware → Hardware description languages and compilation**; • **Theory of computation** → *Automated reasoning*.

Additional Key Words and Phrases: Program Synthesis, FPGA, Technology Mapping, High-level Synthesis

1 INTRODUCTION

Compiling a workload to an FPGA is an important and challenging endeavor. FPGAs offer a wide range of customization and specialization for consumers looking to accelerate workloads due to their varied set of *primitives*; small modules effective at a variety of tasks. If a workload is effectively mapped to an FPGA's primitives, users can see orders of magnitude improvement in performance compared to an un-tuned workload. For users to effectively customize an FPGA for their workload, FPGA manufacturers offer sophisticated hardware compilers (such as Xilinx's Vivado) [4] that systematically map portions of a hardware design to FPGA primitives, in a process called *technology mapping*. Technology mapping is traditionally built into these compilers as a mix of automated mapping workflows and manually-written syntactic pattern matching, where high-level designs match pre-written templates for primitive instantiation. Current technology mappers succeed in effectively mapping to simple primitives, such as the common lookup table (LUT) [3].

However, FPGA manufacturers have recently begun adding new, specialized, highly configurable primitives to their ecosystem. Such primitives have complex and programmable behavior, and current tools are incapable of automatically mapping program designs to these primitives. Thus, compiler engineers graft verbose, handwritten pattern matchers onto the tool-chains, a laborious process that must be repeated for any new primitive. Yet, even handwritten pattern matchers fall short for some complex primitives. For example, Xilinx offers the `CFGLUT5` primitive for their range of FPGAs, which has the ability to reconfigure itself at runtime. It has primarily been used in FIR filter designs due to its efficiency and reconfigurability [2, 6]. Despite being more than 10

Author's address: Vishal Canumalla, vishalc@cs.washington.edu, University of Washington, Seattle, Washington, USA.

years old, Vivado still **does not automatically support** CFGLUT5. Users must manually instantiate the primitive in their design, which requires thorough understanding of its complex behavior. It is clear that as more unique and complex primitives are introduced, current methods of support will not scale, and a more automated approach is needed.

Lakeroad is a new tool that addresses this issue. By utilizing sketch-guided program synthesis, Lakeroad automatically synthesizes technology mappings to hardware primitives given minimal user input. Lakeroad automatically extracts formal semantics for hardware primitives from vendor provided Verilog, and generates implementations of hardware designs from a short, user provided configuration file. Lakeroad currently supports a variety of simple primitives such as LUTs and carry chains, as well as more complex primitives such as DSPs. Automated technology mapping is an enticing feature of Lakeroad, but as more primitives are released by vendors, it is crucial that Lakeroad is robust enough to support new primitives with minimal effort.

In this work, we evaluate the extensibility of Lakeroad's approach to semantics extraction and sketch-guided synthesis, by adding support for the CFGLUT5 primitive. CFGLUT5 was chosen primarily for its runtime reconfigurability, which no primitive in Lakeroad exhibits. Through adding support for CFGLUT5, we demonstrate the exciting promise of sketch-guided synthesis as an extensible approach to supporting diverse primitives, while also discussing limitations and difficulties that can be addressed to make the technique even more powerful and automated.

2 APPROACH

Lakeroad applies *sketch-guided* program synthesis to the technology mapping problem. In short, program synthesis takes in a formal specification, and outputs a program that abides by the assertions set in the specification. Sketch guided synthesis takes in formal semantics of the working language, a specification, and sketches with "holes": empty parts of the program that the synthesis tool fills in. In Lakeroad, output programs correspond to an FPGA mapping configuration, while the specification corresponds to the high-level HDL code. A key insight of Lakeroad is to utilize vendor-provided primitive specifications as the formal semantics to apply synthesis.

From Xilinx's provided HDL description of CFGLUT5, we ran the Lakeroad importer to obtain Racket semantics for the primitive, which we then used to synthesize design mappings. CFGLUT5 demonstrated the current limitations of Lakeroad in the import process, which is discussed in detail in section 3. From the semantics, we wrote sketches for basic bitwise operations on CFGLUT5. These bitwise operations such as AND, NOT, and OR serve as a litmus test for the validity of the imported semantics, as we can easily utilize key functional behavior of CFGLUT5. For example, though AND, NOT, and OR can be trivially implemented in a simple LUT by configuring the memory accordingly, we can use CFGLUT5's dynamic reconfigurability to program CFGLUT5's memory at runtime.

Finally, for full testing of Lakeroad's synthesis, we attempted to map parts of a design for one of the most common use cases for CFGLUT5, the reconfigurable FIR filter. In particular, prior work [2] has found a cluster of CFGLUT5s to be effective at implementing the following function for two vectors $\vec{c}$, $\vec{x}$ of size N.

$$f(\tilde{x}_b^N) = \sum_{n=0}^{N-1} c_n x_{n,b} \tag{1}$$

where n is the nth component of a vector, and $x_{n,b}$ is the bth bit of x_n.

Table 1. Comparison of mapping abilities of mapping tools. A ✓ means the the operation (OP) could be mapped to CFGLUT5 using the specified tool chain. An ✗ means the tool could not map the operation to CFGLUT5. **IP** means the process is ongoing, and most likely possible.

OP	CFGLUT5 via LR	CFGLUT5 via Vivado
NOT	✓	✗
2AND	✓	✗
3AND	✓	✗
2OR	✓	✗
3OR	✓	✗
$f(\tilde{x}_b^N)$	**IP**	✗

3 PRELIMINARY RESULTS

In our initial efforts to support CFGLUT5, we found two points of interest to evaluate Lakeroad's effectiveness. **First,** the amount of manual effort required in semantics extraction, and **secondly**, the manual effort required in synthesizing a mapping.

Extracting the semantics of the CFGLUT5 Verilog specification to Lakeroad required manually adding 36 lines of Verilog to produce complete semantics. Furthermore, the imported semantics had to be modified to be fully usable, which required adding < 5 lines of Racket code. Overall, the amount of manual effort beyond what Lakeroad expects from the user was minimal, but critical for full support. Future work for Lakeroad to avoid this could be building a more complete importer, rather than solely relying on underlying technlogies like Yosys [11].

For evaluating the extensibility of the mapping, we began supporting multiple operations, shown in Table 1. It was straightforward to map simple operations to CFGLUT5 using extracted semantics. We wrote both simple sketches where CFGLUT5 functioned as a normal LUT, as well as sequential sketches that configured CFGLUT5 over multiple clock cycles. In both cases, Lakeroad was able to synthesize a mapping. As per Xilinx's documentation for Vivado, the user must instantiate instances of CFGLUT5 in their design, so by default, no general design can map to CFGLUT5 via Vivado. However, Vivado can map these simple operations to simpler primitives, such as general LUTs, encouraging us to begin looking at more complex operations. For simple operations where a small sketch is sufficient, Lakeroad effectively and accurately maps to a new primitive where Xilinx is unable.

For larger and more complex operations such as that in eq. (1), creating a sketch that Lakeroad can use to synthesize a mapping was a complex endeavor. Work to synthesize a mapping for this operation is ongoing. As shown in prior work [2], a correct mapping exists, so a corresponding sketch template would enable Lakeroad to map this design to CFGLUT5s.

4 FUTURE DIRECTIONS

There are several avenues to further improve Lakeroad's extensibility. One possibility entails using a different representation for our HDL specifications, which could improve semantics extraction and rewriting opportunities. Past work on functional hardware languages has shown that there is promise in a pure representation [7, 9], and could be adapted to the purpose of Lakeroad.

Related work to Lakeroad has shown the promise of SMT in this space [1], but an issue arises for Lakeroad when scaling to large designs. Future work could involve using more specialized SMT solvers, such as Bitwuzla, which specializes in reasoning about bitvectors [5], or applying more sophisticated compiler techniques such as equality saturation [8, 10] to reduce search-space and identify new mappings.

REFERENCES

[1] Ross Daly, Caleb Donovick, Jackson Melchert, Rajsekhar Setaluri, Nestan Tsiskaridze Bullock, Priyanka Raina, Clark Barrett, and Pat Hanrahan. 2022. Synthesizing Instruction Selection Rewrite Rules from RTL using SMT. In *CONFERENCE ON FORMAL METHODS IN COMPUTER-AIDED DESIGN–FMCAD 2022*. 139.

[2] Martin Kumm, Konrad Möller, and Peter Zipf. 2013. Reconfigurable FIR filter using distributed arithmetic on FPGAs. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2058–2061.

[3] Alan Mishchenko, Satrajit Chatterjee, and Robert K. Brayton. 2007. Improvements to Technology Mapping for LUT-Based FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, 2 (2007), 240–253. https://doi.org/10.1109/TCAD.2006.887925

[4] Romina Soledad Molina, Veronica Gil-Costa, María Liz Crespo, and Giovanni Ramponi. 2022. High-Level Synthesis Hardware Design for FPGA-Based Accelerators: Models, Methodologies, and Frameworks. *IEEE Access* 10 (2022), 90429–90455. https://doi.org/10.1109/ACCESS.2022.3201107

[5] Aina Niemetz and Mathias Preiner. 2020. Bitwuzla at the SMT-COMP 2020. *CoRR* abs/2006.01621 (2020). arXiv:2006.01621 https://arxiv.org/abs/2006.01621

[6] Debapriya Basu Roy, Shivam Bhasin, Sylvain Guilley, Jean-Luc Danger, Debdeep Mukhopadhyay, Xuan Thuy Ngo, and Zakaria Najm. 2015. Reconfigurable LUT: A double edged sword for security-critical applications. In *Security, Privacy, and Applied Cryptography Engineering: 5th International Conference, SPACE 2015, Jaipur, India, October 3-7, 2015, Proceedings 5*. Springer, 248–268.

[7] Mary Sheeran. 1984. muFP, a language for VLSI design. In *Proceedings of the 1984 ACM Symposium on LISP and functional programming*. 104–112.

[8] Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. 2009. Equality saturation: a new approach to optimization. In *Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 264–276.

[9] Lenny Truong and Pat Hanrahan. 2019. A golden age of hardware description languages: Applying programming language techniques to improve design productivity. In *3rd Summit on Advances in Programming Languages (SNAPL 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[10] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. 2021. Egg: Fast and extensible equality saturation. *Proceedings of the ACM on Programming Languages* 5, POPL (2021), 1–29.

[11] Clifford Wolf, Johann Glaser, and Johannes Kepler. 2013. Yosys-a free Verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*. 97.