A Comparison of Verilog Synthesis Frontends

Daniel Stokes*, Georgiy Krylov†, Jean-Philippe Legault†, Panos Patros*, Kenneth B. Kent†
*ORKA Cloud and Adaptive Systems Lab, Dept. of Software Engineering, University of Waikato, Aotearoa New Zealand
email: djns1@students.waikato.ac.nz, email: panos.patros@waikato.ac.nz
ORCiD: 0000-0002-1366-9411
†Centre for Advanced Studies-Atlantic, Faculty of Computer Science, University of New Brunswick, Fredericton, Canada
email: georgiy.krylov@unb.ca, email: jlegault@unb.ca, email: ken@unb.ca

***Abstract*—A crucial consideration in choosing a frontend synthesis tool is the quality of the synthesised result. This kind of benchmarking is critical to choosing a fit-for-purpose tool. However, to the best of the authors' knowledge, the only comparison of Odin II, the front-end of Verilog-to-Routing, and another synthesis tool was focused primarily on Odin II and Yosys' performance with respect to commercial counterparts in the Xilinx ISE tool. Further, such an evaluation is to improve confidence in research findings utilising these tools. The quality of results for a poorly optimised research tool may not reflect the performance of real-world applications, adding uncertainty to any findings and requiring extra work from the researcher to obtain valid results. We compare Odin II and Yosys targeting the Xilinx Artix-7 architecture provided by SymbiFlow.**

***Keywords*—Field Programmable Gate Arrays (FPGA); Computer Aided Design (CAD); Verilog.**

I. Introduction

To achieve performance and sustainability, projects incorporate specialised technologies for resource-intensive sections of their applications replacing software with hardware. Application Specific Integrated Circuits (ASICs) provide significant performance gains at the cost of a high upfront investment and a significant loss in flexibility of application. Few projects have sufficiently well-defined requirements and the funding required to manufacture their own ASICs; hence, Field Programmable Gate Arrays (FPGAs) are used as an alternative. FPGAs provide much of the performance of ASICs without the upfront cost, and are a better fit for prototyping and research [1] because they can be reprogrammed if the specifications change or if any errors in the design are found.

Applications for embedded devices are written in specialised Hardware Description Languages (HDLs), such as Verilog or VHDL. Designing and efficiently mapping HDL programs to hardware involves many steps, from parsing and optimising the HDL design to allocating logic elements to hardware resources. Verilog-to-Routing (VTR) [2] is an open source tool that aims to provide a highly efficient Computer Aided Design (CAD) flow, for mapping the Verilog 2005 [3] HDL onto an FPGA. VTR targets either commercial or experimental designs, making it a powerful tool for both circuit design and research into the next generation of FPGA devices and architectures. Other CAD flows, such as SymbiFlow [4], also exist. CAD flows share flow components with VTR, which enables comparing different frontend logic synthesis tools.

The work by Hung [5], provides a comparison of Odin II and Yosys front-end synthesis tools as part of VTR. Their experiment was conducted six years prior to this paper, which means many changes happened to both rapidly-growing projects, and the results of the comparison might differ. This paper presents the current challenges of conducting the comparison between the two tools, as well as aims to provide up-to-date information on performance differences.

II. Background

This section describes the principles of operation and the distinguishing features of the Symbiflow and VTR CAD flows.

A. Verilog To Routing (VTR)

VTR is an open source CAD flow for mapping a Verilog 2005 circuit design to an FPGA bitstream. It uses several interoperating tools for producing an FPGA program.

1) Odin II: The VTR flow consists of five primary stages, performed by three separate tools as shown in Figure 1. The first stage in this flow, Odin II [6] is responsible for parsing the Verilog circuit description into a flattened netlist—a process known as synthesis. This involves three distinct phases: parsing, elaboration and technology mapping. The parsing stage involves taking the provided Verilog files and parsing them into an Abstract Syntax Tree (AST). The AST encodes information about the structure and hierarchy of the program, where each node in the tree corresponds to a Verilog construct from the source files. It is necessary to unroll loops in Verilog at this stage, since these are infeasible to implement in hardware. Optimisations such as constant folding can also be done to reduce the number of nodes in the final netlist [7].

The next phase of the Odin II flow is elaboration. It involves taking the high level Verilog abstractions encoded by the AST and converting them into an Odin II-specific data structure, called a *netlist*. It also involves instantiating Verilog modules (high-level language feature grouping programs by functionality) at the appropriate locations in the circuit.

After Odin II builds the netlist, it moves to the final phase: technology mapping. This process maps netlist nodes to specialised FPGA circuitry known as hard blocks. Often, FPGA designs include specialised hardware implementations of common operations, these are known as hard blocks or Intellectual Property (IP) cores on commercial devices. During the technology mapping phase, Odin II reads the architecture file and identifies the types of available hard blocks. Odin II can also analyse the circuit and architecture file to identify the number and physical location of these blocks allowing

 ISBN: 978-1-61208-921-8

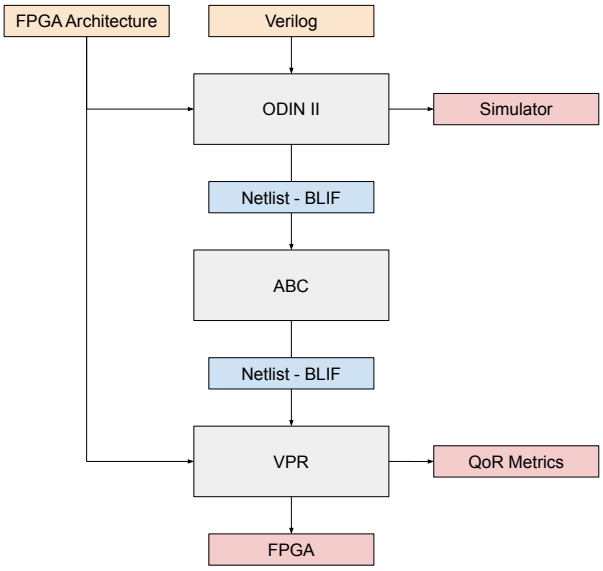

Figure 1. Stages of the VTR CAD flow

the technology mapping to account for placement and routing costs associated with each allocation [8]. Odin II also converts logic to be suitable for placement on hard blocks based on the specifications of the target FPGA [9].

The final stage of Odin II is its power-estimation capabilities even when hard blocks are present via simulating the circuit's behavior over a number of specified inputs [10].

While Odin II's technology mapping gives it a significant lead for academic and experimental applications, Odin II has shortfalls. One significant barrier to its adoption is incomplete coverage of the Verilog 2005 specification. At present Odin II only covers a subset of the language thereby reducing the selection of the programs that can be synthesized to completion.

2) ABC: The second stage in the flow is optimisation. VTR uses the ABC [11] tool to perform this step. ABC takes the BLIF file output by Odin II and performs a range of combinational and sequential logic optimisations on the netlist to reduce the overall circuit footprint. These optimisations are critical for obtaining quality synthesised results, which are measured via comprehensive Quality of Result (QoR) metrics about the synthesis output), such as device size, maximum frequency and estimate of placed wire length [12].

3) VPR: Versatile Place and Route (VPR) [13] runs the final three stages of the flow: packing, placement and routing. VPR uses a genetic algorithm to efficiently find a strong final result without searching for the best possible solution. The quality of the final result is captured in QoR metrics output by VPR once the flow completes. Researchers and designers can use these metrics to evaluate the quality of the individual phases in the flow and the impact of design decisions made at both the Verilog and FPGA architecture level.

B. SymbiFlow

A stage of the VTR flow interfaces with the next via a BLIF file, allowing for stages to be replaced or removed, as desired. Hung [5] used this characteristic to compare stages of the VTR flow to their commercial counterparts, e.g., SymbiFlow [4].

The SymbiFlow project [4] is an FPGA CAD flow targeting commercial architectures, with support for Xilinx Artix 7 [14] and Zynq SoC 7000 [15] families, and for the Lattice iCE40 [16] and ECP5 [17] devices. The synthesis flow of the project closely resembles that of VTR, with VPR being used to pack, place and route for both projects. Instead of Odin II, SymbiFlow uses Yosys [18] for synthesis, which in turn integrates ABC internally for optimisation. As with VTR, each stage of this flow is modular, so a range of different tools, including both open source and commercial, can be used.

In addition to the standard CAD flow, SymbiFlow also includes peripheral functionality. In particular, an important feature for this work is the architecture database. SymbiFlow includes several sub-projects, each targeting a different device family. This work focuses on the Artix-7 XC7A200T architecture generated by Project X-Ray [19]. The database contains architecture files that describe the CLBs and hard blocks the device provides, which can be compiled into a VPR architecture file. It also provides Verilog files that are used during the technology mapping phase in Yosys.

While SymbiFlow has many similarities with VTR, it does not use Odin II for synthesis, instead opting to use Yosys. There are several reasons for this, the most obvious is more comprehensive language support. Another powerful feature of Yosys is its scripting interface that gives the user more control over the exact behaviour of the tool. This is achieved by breaking the synthesis flow into three distinct types of operation: Frontends, Passes and Backends.

A Yosys Frontend parses an HDL (not necessarily Verilog) into a common netlist representation. Input files can be processed by different Frontends and added to the internal model as needed. The internal model constructed by the Frontends can be processed by Passes. Yosys Passes can perform a variety of operations including technology mapping, optimisation through ABC and manipulating the module hierarchy. Finally, the processed netlist can be output in many different formats via the corresponding Yosys Backend.

One benefit of this modularity is that it allows users to add custom Passes for their application. Effective utilisation of hard blocks can vastly improve the quality of the generated circuit by reducing the number of LUTs, DFFs and wires. Hard blocks can also provide more performant implementations than would otherwise be implemented via soft logic.

Yosys [18] provides hardcoded technology mapping flows for a range of supported architectures, but requires the user to generate custom Verilog descriptions of the FPGA architecture to target a new device—even when used in combination with VPR. In contrast, Odin II can perform this stage at no extra cost by utilising the architecture file used for VPR. This gives Odin II a significant advantage when evaluating novel and experimental FPGA designs, as designers do not need a supplemental architecture definition for technology mapping.

 ISBN: 978-1-61208-921-8

III. Experimental Comparison of Odin II and Yosys

Comparing the performance of Odin II and Yosys requires accounting for a wide range of factors. The first decision to make is which architecture(s) to use for the comparison. Since both Odin II and Yosys will be used with VPR, the architecture must have a VPR XML architecture definition available. While Odin II can use the same architecture file as VPR, Yosys primarily relies on hardcoded technology mapping passes specific to a given architecture.

SymbiFlow provides XML architecture files for the Artix-7 family, and the corresponding technology mapping support required by Yosys. In addition, SymbiFlow supports placement and routing of these devices using VPR, thereby allowing the final QoR of the synthesized circuits to be compared with the VTR flow. This allows running both tools against a concrete commercial architecture, providing a real-world benchmark of each flow's performance. In particular, this work focuses on the largest in the Artix-7 family—the XC7A200T architecture. The largest device was chosen to ensure all benchmarks could successfully route, as several of the larger benchmarks tested require too many blocks for the smaller architectures.

A further consideration when choosing an architecture relates to hard block inference. Odin II only recognises hard blocks that match a specific signature, while Yosys can map to any hard block in its technology mapping database. While Odin II's approach is good when targeting experimental architectures where the hard blocks can be written to match the expected format, for a commercial device Yosys has support for hard block inference. In particular, Yosys is capable of inferring RAM blocks, which can greatly reduce circuit footprint because multiple CLBs are required to emulate memory.To ensure a meaningful comparison, both Yosys and Odin II were configured and run appropriately, such that both tools could recognize the same potential hard blocks.

To isolate the behaviour of just Odin II and Yosys, it is necessary to ensure that the rest of the flow is consistent across all runs. To control for this, both flows use the ABC and VPR versions built by the SymbiFlow project. However, to provide a more comprehensive analysis, this work also tests Odin II against the version of ABC packaged with VTR. This makes it possible to identify any behaviours that Odin II depends upon that are not present in the SymbiFlow version of ABC.

Since VPR uses a non-deterministic genetic algorithm based approach to performing packing, placement and routing the final result depends on the initial seed. A representative sample of average performance must be obtained via averaging the performance across multiple runs of VPR. Since Yosys, ABC and Odin II are deterministic they are only run once per benchmark, while VPR is run with 10 seeds on the BLIF file produced by each flow. Once the synthesis flow completes for each benchmark, the result is copied according to the number of iterations needed by VPR, which is then run on each copy with a different seed. The full flow appears in Figure 2.

Additionally, each synthesis flow was run 10 times to allow for a comparison of run-time metrics such as total time taken and memory usage. The *time* command is used to report the *user time* and *maximum resident set size (max. RSS)* metrics for each run. An important note is that ABC does not have integrated support for optimising across clock domains. To handle this case, VTR runs ABC once for each clock domain present. In these experiments, each run of ABC generated separate run-time metrics. The *user time* metric was obtained by summing across all runs, while the *max. RSS* was obtained by taking the maximum value across all runs.

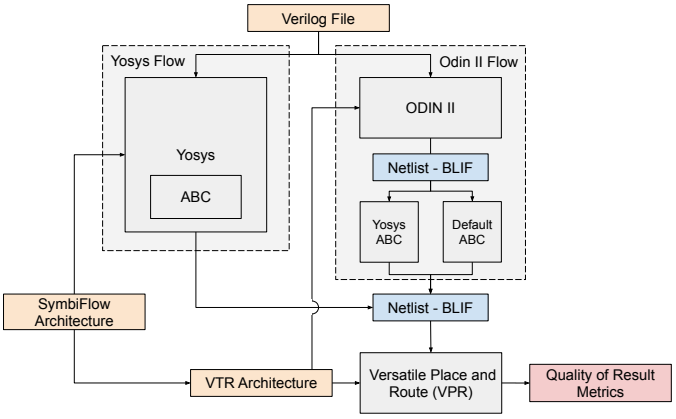

Figure 2. Full Benchmark Flow

All results were obtained on a Ubuntu 18.04 (kernel 4.15.0-117) with an Intel® Core™ i5-8400 CPU @ 2.80GHz, configured to run in performance mode. The machine was configured with 4x16GiB DDR4 RAM running at 2133 MT/s.

A. Benchmarks

To compare Yosys and Odin II, this work focuses on a subset of the Verilog benchmarks from the VTR project that can be synthesized to completion with both flows. Several well-known benchmarks were considered for evaluation, but they either failed to synthesize through Odin II or did not have an open source Verilog implementation available. While these VTR benchmarks can be correctly synthesised by Odin II on VTR's comprehensive architecture, many use hardware RAM blocks that are incompatible with the Xilinx architecture used in the experimental flow. To support these RAM blocks, this work utilises VTR's ability to leverage generic hard blocks to target the Xilinx Block RAM module included in the SymbiFlow architecture definition. In particular, this architecture defines the *RAMB18E1_VPR* hard block, which corresponds to the *RAMB18E1* Xilinx IP core. This enables the *mkPktMerge* benchmark, which otherwise could not have been included in this evaluation. However, modifying all such benchmarks is infeasible and several others can not easily be converted while maintaining support for Odin II. In total, fourteen benchmarks were modified to support the new flow.

B. Hypotheses

Hung showed that the Yosys flow produced a better (i.e., lower) critical path delay and similar logic area utilisation [5]. For this comparison, it is expected that Yosys will further

 ISBN: 978-1-61208-921-8

TABLE I
BLOCKS AND NETS IN NETLIST PRIOR TO PACKING IN VPR

	Blocks		Nets	
Test Name	Odin	Yosys	Odin	Yosys
and latch	6	8	5	7
bgm	54624	69412	54592	69380
blob merge	8030	8378	7930	8278
diffeq1	4907	6030	4811	5934
mkPktMerge	347	402	191	291
multiclock output and latch	14	20	13	19
multiclock read write	16	42	16	41
sha	3569	3143	3533	3107
single ff	4	6	3	5
single wire	2	2	1	1
stereovision0	21431	34493	21234	34296
stereovision1	42579	58469	42434	58324
stereovision2	60233	91063	60051	90881
stereovision3	310	477	281	447

extend this gap, as it has received significant tuning against the XC7A200T target architecture. In contrast, Odin II aims to work out-of-the-box for a wide range of devices limiting the number of architecture specific optimisations possible.

In addition to QoR, run-time statistics are also an important aspect of both flows. Hung's work does not show either flow as having conclusively better total flow time, and does not report on the time taken by each phase of the flow, only reporting the total flow time. Hung's work also does not include a report on the memory requirements of the flows. It is hoped this evaluation will be able to provide more insight into the expected memory consumption and total time for each flow.

C. Results

This evaluation used the 14 selected VTR benchmarks against the SymbiFlow XC7A200T architecture, each run 10 times through VPR with distinct seeds. Likewise, each synthesis tool was run 10 times against each benchmark to gather run-time statistics. Where applicable, the geometric mean (geomean) is used to combine the results from the 10 VPR or synthesis runs. These benchmarks vary in size significantly, from the smallest *single wire* benchmark, with 2 blocks and 1 net under Odin II, to the largest *stereovision2*, with 60,233 blocks and 60,051 nets under Odin II. Table I shows the number of netlist primitives in each benchmark, when synthesized with Odin II or Yosys and optimized by ABC. This table also shows that Odin II produces fewer netlist primitives across all benchmarks except *sha*.

The primary QoR metrics of interest are the critical path delay and the logic block area metrics. The critical path delay in seconds (s) is the inverse of the maximum frequency (Hz), at which the circuit can be run. The logic block area, measured in Minimum Width Transistor Areas (MWTAs), determines the size of the FPGA required and influences the total circuit power usage. Figure 3 shows the geomean critical path delay of Odin II and Yosys, normalised such that the critical path delay for Odin II is 1.0. Note that both ABC versions tested with the Odin II flow produced identical results, so only one value is reported. Likewise, Figure 4 shows geomean logic area used by Odin II and Yosys, normalised such that the logic area used by Odin II is 1.0. Hung [5] ran Yosys and Odin II

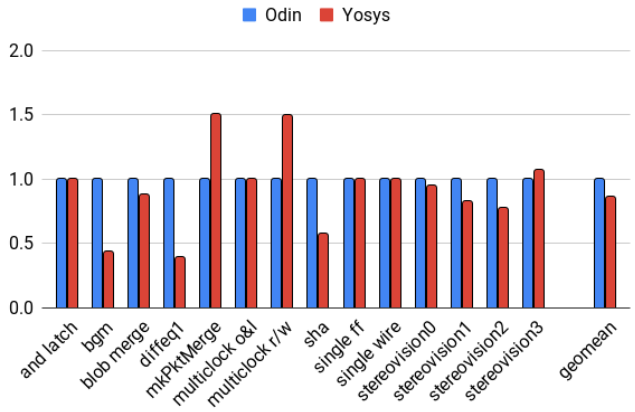

Figure 3. Normalised critical path delay

against the *stereovision2* benchmark, targeting a Xilinx Virtex-6 family device. That work found that both flows produced a similar critical path delay, while Odin II used approximately half as many logic slices as Yosys. This contrasts with the findings in our work where Yosys has a 14% lower critical path delay, while also using 11% less area. There is a similar improvement in critical path delay for the *bgm* benchmark, which was also included in Hung's comparison, suggesting that Yosys is producing much higher quality circuits for this evaluation than in Hung's work. This matches expectations as outlined in Section III-B, as Yosys has had several years of development since the publication of Hung's work, with significant focus to supporting the Xilinx Artix-7 line used in this evaluation. For the *mkPktMerge* benchmark, Odin II uses a 6.8x larger logic block area, but still yields a 1.5x shorter critical path delay. This is a significant difference in performance, suggesting that both tools have specific applications in which they excel, however, there are general trends that can be observed. In particular, we see that Yosys yields shorter critical path delays—producing a geomean critical path delay across all benchmarks of 8.95888 ns, a 14% improvement over Odin II's 10.373 ns. When considering logic block area, we see Yosys tends to yield smaller areas—producing a geomean logic area used across all benchmarks of 1,925,560 MWTAs, an 11% improvement over Odin II's 2,168,990 MWTAs.

An important consideration here is the size of the benchmarks. While small benchmarks can give insight into specific behaviours, larger benchmarks are more representative of real world applications. Considering just the seven benchmarks that generate over 1000 blocks when synthesised with Odin II, we see that Yosys produces a lower critical path delay in every benchmark. Overall, Yosys produces a geomean critical path delay of 23.0714 ns compared to Odin II's 35.1330 ns, a 36% improvement. Similarly, the geomean logic block area used for Yosys is 32,277,700 MWTAs compared to Odin II's 35,538,200 MWTAs, a 9.2% improvement. This suggests that Yosys is better optimised for larger benchmarks, where its targeted technology mapping can produce a better QoR.

In addition to the QoR, run-time metrics, such as total

 ISBN: 978-1-61208-921-8

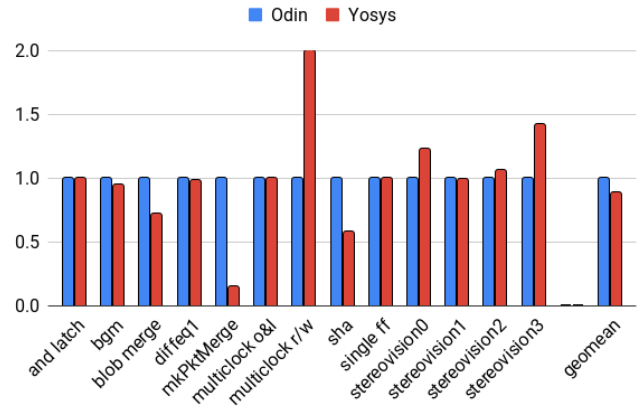

Figure 4. Normalised logic area used

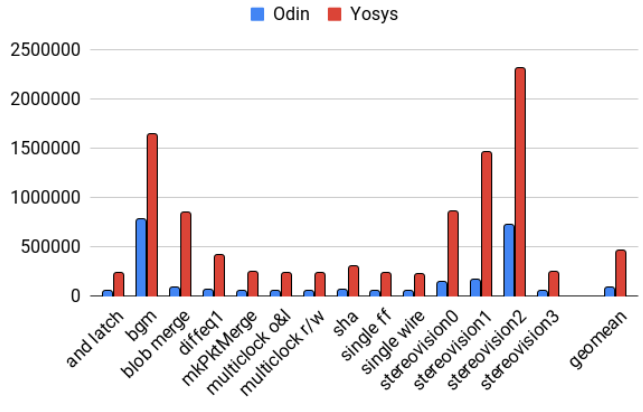

Figure 5. Geomean max. RSS (kilobytes) for synthesis flow

time taken and memory usage, are also important. Figure 5 shows the geomean max. RSS for each of the synthesis tools tested: Yosys required more memory than Odin for every benchmark. This shows that Yosys used between 2.1x and 10.3x more memory than Odin II across all benchmarks, with a geomean across all benchmarks of 460,209 kilobytes, 5.1x larger than Odin II's 89,760.2 kilobytes. While this is a significant difference, which may impact some workloads, it is unlikely to be a deciding factor for most designs, since the lowest *max. RSS* value reported by VPR was 13,227,032 kilobytes, 5.7x larger than the largest *max. RSS* reported by Yosys. A more important difference is the total user time of each flow. The small benchmarks with under 1000 primitives have limited use since the total user time for Odin II is under one second and so the reported timings are dominated by both the launch overhead and the VPR time taken (minimum of 38.62 s for the smallest benchmark), so we only consider the benchmarks with over 1000 primitives. Figure 6 shows the user time taken for the benchmarks with over 1000 primitives, noting that times include ABC for both flows. This shows that the Yosys synthesis flow runs between 1.1x and 3.2x faster than Odin II in five of the seven cases. In the case of *sha*, Yosys runs 78.8x faster than the default Odin II flow, while for *diffeq1* Yosys takes 4.1x longer at 16.12 s compared to Odin II's 3.96 s. It is worth noting that Odin II only consumes between 0.13% and 8% of the total pre-VPR flow time, so the Odin II flow's poor performance is likely due to a combination of a lower quality initial netlist requiring more work from ABC and running more demanding ABC optimisations. Across all cases, the VTR version of ABC is between 1.7% and 5.7% faster than the Yosys version of ABC. Both versions produced functionally identical results for all benchmarks tested, but the output BLIFs had small cosmetic differences suggesting a version difference, in favour of the VTR version.

To understand the effects of VTR's non-deterministic algorithm on BLIF files generated by different frontends, this work also looks at the run-time metrics for VPR, starting with VPR's max. RSS for each flow. We found that VPR's memory consumption is consistent for a given architecture and does

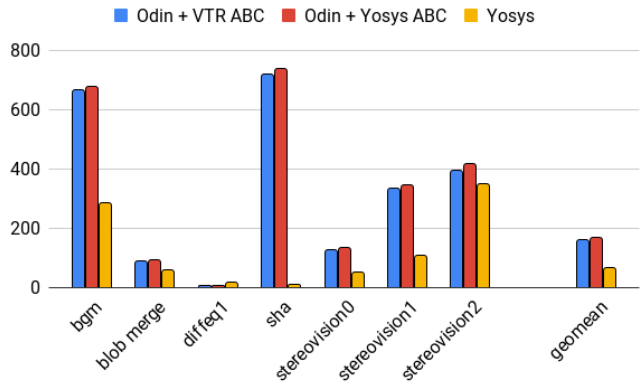

Figure 6. Geomean user time (s) for pre-VPR flow

not vary with the size of the benchmarks, measuring at 10.2-11.3 GiB for all benchmarks. Figure 7 shows the geomean VPR user time for each flow. This shows that for the larger benchmarks, the BLIF produced by Yosys takes longer to place and route than the corresponding BLIF produced by Odin II. When combined with the timings for the synthesis flow from Figure 6, we see that the full Yosys flow completed faster in five of the seven large benchmarks, while the full Odin II flow completed faster for *diffeq1* and *stereovision2*.

D. Threats to Validity

The comparison between Odin II and Yosys targeted a single architecture and a limited set of benchmarks. These benchmarks are those used by the VTR flow to validate performance, so Odin II's behaviour is likely to be optimised for performance against these benchmarks. It is thus possible, that the performance of each tool may deviate from the findings of this comparison when targeting a different architecture or with different sets of benchmarks. Further, the benchmarks used are smaller than modern benchmarks, such as the Titan Benchmark suite. Moreover, a number of core features were disabled for these benchmarks.

While the XC7A200T architecture provides a valuable reference for a commercial design, it is constantly being improved

 ISBN: 978-1-61208-921-8

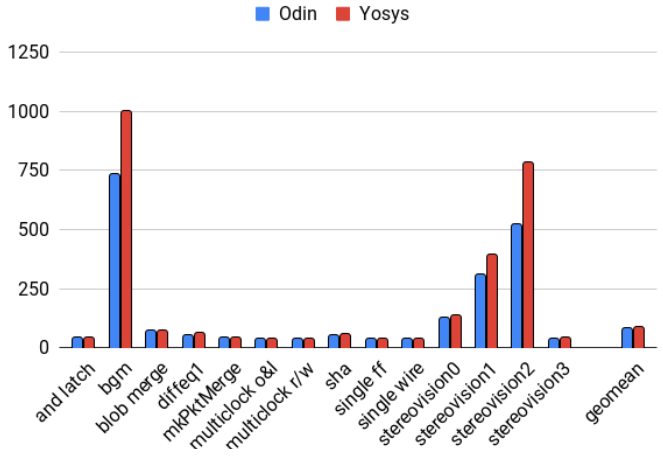

Figure 7. Geomean user time (s) for VPR

and so does not perfectly represent the true commercial device. Moreover, the developers of SymbiFlow have specifically tuned Yosys to run against this architecture, while many of Odin II's features cannot target this architecture. As such, a comparison against an architecture that can fully leverage Odin II's features may show different results.

IV. Conclusion

To answer how Odin II performs relative to other open source synthesis tools, this work undertook a comparison of Odin II with another open source synthesis tool Yosys, targeting the XC7A200T architecture.

The findings show that Yosys produced 14% lower geomean across all benchmarks for critical path delay, increasing to 34% lower geomean for circuits with over 1000 blocks. Additionally, Yosys also yielded 11% lower geomean across all benchmarks for logic area used, decreasing to 9.2% lower geomean for circuits with over 1000 blocks. This shows that Yosys consistently produces better QoR for large circuit designs when targeting the XC7A200T architecture. The comparison also looked at the run time metrics of the two tools where Yosys used between 2.1x and 10.3x more memory than Odin II, but completed the synthesis faster than Odin II in six out of seven large benchmarks. Further, the full Yosys CAD flow had equivalent memory requirements and completed faster than the full Odin II CAD flow in five of seven large benchmarks. As part of this evaluation, this work presented a framework for comparing the Odin II and Yosys synthesis flows. This framework allows the two flows to be compared against a range of architectures and benchmarks, enabling greater insight into the behaviour of these tools.

The framework described in this work enables Odin II and Yosys to be compared against many different benchmarks and architectures. The Titan Benchmark suite [20] is a much larger set of benchmarks that have been applied to evaluating the performance of VPR. However, due to Odin II's limited language support, the Titan flow makes use of Intel's Quartus II tool to perform synthesis. Future work is encouraged to revisit this suite once Odin II achieves sufficient language coverage.

Acknowledgments

The authors thank the University of New Brunswick, the University of Waikato, Nyriad Ltd., and ORKA Lab for the software and hardware resources to complete this work.

References

[1] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *ACM Comput. Surv.*, vol. 34, no. 2, p. 171–210, Jun. 2002. [Online]. Available: https://doi.org/10.1145/508352.508353

[2] K. E. Murray *et al.*, "Vtr 8: High-performance cad and customizable fpga architecture modelling," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 2, pp. 1–55, May 2020. [Online]. Available: https://doi.org/10.1145/3388617

[3] IEEE, "Ieee standard for verilog hardware description language," *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, pp. 1–590, 2006.

[4] K. E. Murray *et al.*, "Symbiflow and vpr: An open-source design flow for commercial and novel fpgas," *IEEE Micro*, vol. 40, no. 4, pp. 49–57, 2020.

[5] E. Hung, "Mind the (synthesis) gap: Examining where academic fpga tools lag behind industry," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–4.

[6] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon, "Odin ii - an open-source verilog hdl synthesis tool for cad research," in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, 2010, pp. 149–156.

[7] B. Yan and K. B. Kent, "Hard block reduction and synthesis improvements in odin ii," in *2015 International Symposium on Rapid System Prototyping (RSP)*, 2015, pp. 126–132.

[8] G. Krylov, J. P. Legault, and K. B. Kent, "Hard and soft logic trade-offs for multipliers in vtr," in *2020 23rd Euromicro Conference on Digital System Design (DSD)*, Aug 2020, pp. 40–43.

[9] J.-P. Legault, P. Patros, and K. B. Kent, "Towards trainable synthesis for optimized circuit deployment on fpga," in *2018 International Symposium on Rapid System Prototyping (RSP)*. IEEE, 2018, pp. 90–96.

[10] S. Seeley, V. Sankaranaryanan, Z. Deveau, P. Patros, and K. B. Kent, "Simulation-based circuit-activity estimation for fpgas containing hard blocks," in *2017 International Symposium on Rapid System Prototyping (RSP)*. IEEE, 2017, pp. 36–42.

[11] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification*, T. Touili, B. Cook, and P. Jackson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 24–40.

[12] VTR Developers, "Verilog-to-routing documentation," https://docs.verilogtorouting.org, 2021, [Online; Retrieved: Oct 2021].

[13] V. Betz and J. Rose, "Vpr: a new packing, placement and routing tool for fpga research," in *Field-Programmable Logic and Applications*, W. Luk, P. Y. K. Cheung, and M. Glesner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 213–222.

[14] Xilinx Inc., "Artix-7 fpga family," https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html, [Online; Retrieved: Oct 2021].

[15] ——, "Zynq-7000 soc," https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html, [Online; Retrieved: Oct 2021].

[16] Lattice Semiconductor Corporation, "ice40 lp/hx/lm - low-power, high-performance fpga," http://www.latticesemi.com/en/Products/FPGAandCPLD/iCE40, [Online; Retrieved: Oct 2021].

[17] ——, "Ecp5 / ecp5-5g," https://www.latticesemi.com/Products/FPGAandCPLD/ECP5, [Online; Retrieved: Oct 2021].

[18] C. Wolf, "Yosys open synthesis suite," http://www.clifford.at/yosys/, [Online; Retrieved: Oct 2021].

[19] "Project x-ray - xilinx series 7 bitstream documentation," https://symbiflow.github.io/prjxray-db/, [Online; Retrieved: Oct 2021].

[20] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Timing-driven titan: Enabling large benchmarks and exploring the gap between academic and commercial cad," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 2, pp. 1–18, Mar. 2015. [Online]. Available: https://doi.org/10.1145/2629579

 ISBN: 978-1-61208-921-8