Mind The (Synthesis) Gap: Examining Where Academic FPGA Tools Lag Behind Industry

Eddie Hung
Department of Computing
Imperial College London, England
e.hung@imperial.ac.uk

***Abstract*—Firstly, we present VTR-to-Bitstream v2.0, the latest version of our open-source toolchain that takes Verilog input and produces a packed, placed — and now routed — solution that can be programmed onto the Xilinx commercial FPGA architecture. Secondly, we apply this updated tool to measure the gap between academic and industrial FPGA tools by examining the quality of results at each of the three main compilation stages: synthesis, packing & placement, routing. Our findings indicate that the delay gap (according to Xilinx static timing analysis) for academic tools breaks down into a 31% degradation with synthesis, 10% with packing & placement, and 15% with routing. This leads us to believe that opportunities for improvement exist not only within VPR, but also in the front-end tools that lie upstream.**

I. INTRODUCTION

FPGA research into architecture design, and on the computer-aided design (CAD) algorithms to map circuit descriptions onto these architectures, has typically been evaluated using the VPR academic tool that is now packaged with the Verilog-To-Routing (VTR) toolflow [1]. The open-source VTR flow consists of three main tools: Odin II for synthesising circuits designed in Verilog into generic lookup table (LUT) resources, ABC for technology-mapping those resources into architecture-specific LUTs, and VPR for packing, placement, and routing. For architecture design, a key strength of VPR is that it allows researchers to easily experiment with new theoretical FPGA architectures — for example, in deciding the optimal number of inputs for a lookup table, or the best number of LUTs to group into a logic cluster. For CAD research, however, any new innovations can only be evaluated on those same theoretical architectures as opposed to commercially available devices due to the closed-source nature of vendor tools.

First contribution: is to present VTR-to-Bitstream (VTB) v2.0, an open-source extension to the VTR toolchain that provides a path not just from Verilog to a packed, placed, and routed netlist on a realistic FPGA architecture, but also onto a valid bitstream that can be programmed onto a physical Xilinx device. In contrast to the original version of VTB that only supported packing and placement [2], timing-driven routing is now supported, along with carry-chains and the integration of a new Verilog synthesis tool. This extension is provided as a patch available from http://eddiehung.github.io.

Second contribution: is to use the extended toolflow to make a fair and robust comparison between the quality of results (QoR) gained by academic and industrial offerings. By applying a varying combination of both flows to compile bitstreams for the same FPGA device, we are able to infer the gap between the individual tools responsible for synthesis, packing & placement, and routing.

Related work: Relating to our first contribution of providing open-source CAD for commercial FPGAs is RapidSmith [3] and Torc [4], which not only provide abstraction layers for manipulating Xilinx netlists but also contain limited, non timing-driven CAD functionality (e.g. a basic router with no conflict resolution). GoAhead [5] also exists to provide additional support for dynamic partial reconfiguration tasks that are not available through the vendor toolchain.

Bridging both of our contributions is Titan [6], a similar flow but for Altera devices. By leveraging the vendor's (closed-source) Quartus II tool for front-end synthesis but continuing to use VPR for back-end compilation, their hybrid flow enables experimentation with large, complex, and mixed-language netlists that are not supported by the open-source Odin II [7]. Their flow is also used to make comparisons between academic and commercial tools, finding that the VPR back-end alone produces circuits that are 53% slower than Quartus II. Crucially, since Titan continues to route for an approximate model of the commercial architecture, this also means their result is not backed by an industrial static timing analysis tool, nor realisable on a physical FPGA device.

The original version 1.0 of VTR-to-Bitstream [2] presented a Verilog to packed-and-placed netlist flow for Xilinx devices, used vendor tools to route the circuit, and found that the total critical-path delay gap across the entire flow was 110%. As an initial step towards closing this gap, in our second contribution we seek to identify how this total gap breaks down across the individual CAD stages.

II. XILINX ARCHITECTURAL MODEL

The Xilinx architectural model used in this paper is an extension of that in prior work [2], with a number of changes to support hardened adder functionality and carry-chains. The augmented cluster model is shown in Fig. 1.

In contrast to the default VTR architecture which possess a dedicated full-adder, devices from the Xilinx Virtex-6 family have only a 2:1 carry multiplexer (referred to as MUXCY in Xilinx literature) and a 2-input XOR sum gate (XORCY) in hard-logic, requiring the remaining functionality for a full adder (another 2-input XOR gate for deciding whether the carry-in should be propagated, and an AND gate for generating a new carry) to be implemented in soft-logic [8].

The cluster (CLB) model adopted in this work is made up of two logic slices, shown in Figure 1. Within the Virtex-6 architecture, a logic slice holds four basic logic elements (BLEs) each of which contains a 6-input LUT that can be fractured into two 5-input LUTs (with all inputs shared), followed by two flip-flops. A bypass input (AX) can be used to reach either flip-flop, or to feed the XADDER carry-in directly without passing through the LUT. Three outputs exist for each BLE: a combinational output (A) from the primary LUT output O6,

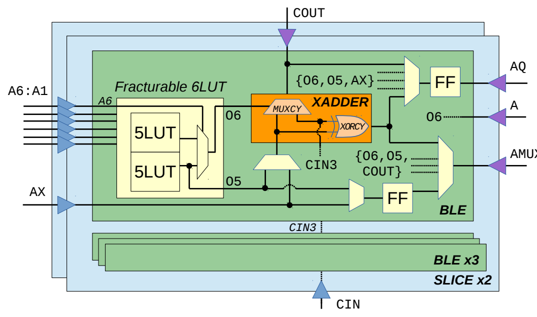


Fig. 1: Virtex-6 logic cluster model adopted in this work. (XADDER represents our new hardened adder black-box).

TABLE I: VTR-to-Bitstream's Virtex-6 architectural support.

Supported:

- Basic logic elements with 6-input LUTs, fracturable into two 5-input LUTs followed by two flip-flops and a carry-chain
- Block RAMs fracturable between a 36K or 18K memory, simple or true dual-port, and full aspect ratio support: from 1b×32K – 72b×512
- DSPs only as a 25x18 combinational multiplier
- Multiple global clock buffers (BUFG) for low skew clock distribution

Not yet supported:

- Logic slice clock enable, set/reset, nor wide multiplexers employing the MUXF7/F8 hardened resources
- Block RAMs as hardened FIFOs, nor distributed (LUT) memory available in SLICEM resources
- All other DSP48E1 functionality (such as 48-bit accumulation)

a sequential output (AQ) and a multi-purpose output (AMUX) shared between the secondary flip-flop, both LUT outputs O6 & O5, and the carry out or sum from the adder. Each slice also contains one carry input and one output (CIN and COUT) connected as a chain through all four BLEs.

Table I details the main Virtex-6 features that our flow supports through leveraging VTR. Given that we are constrained by that which is supported by VTR, in future should any unsupported features be inferred into black-boxes, it would be straightforward to add support for those too. Although we target a Virtex-6 device in this work, with the correct specifications our approach can be extended to any other device from any vendor — currently, we are constrained to those architectures supported by the XDL functionality inside the ISE toolflow; unfortunately, Xilinx has since deprecated XDL in their latest Vivado toolflow (targeting the UltraScale family and beyond).

III. Extending VPR To Route For Xilinx Devices

The challenge with extending VPR to route for an exact Xilinx architecture is that VPR's model can only generate relatively simple routing networks: supporting only horizontal and vertical (but not diagonal) wires, regular and symmetric across both channels, throughout the device. This limitation has been an obstacle in prior comparisons with industrial tools [6]. In the context of the commercial architecture that we are targeting, we find that the actual Xilinx Virtex-6 routing network contains a number of features — in particular diagonal and L-shaped bent wires — that VPR cannot currently model. Through querying Torc [4], we have inferred that the connectivity of each tile appears to be that shown in Figure 2.

We add routing support by preprocessing the Xilinx routing graph and importing it directly into VPR. This approach is made possible because VPR's router is not tied to operating only with routing graphs generated internally: analysis of the

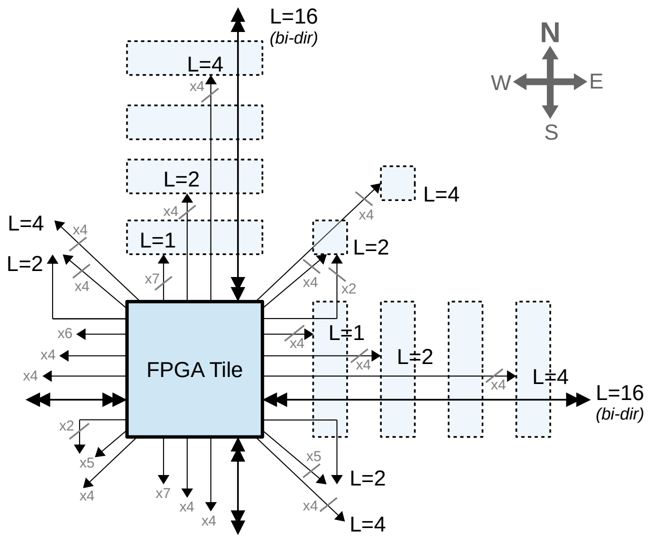


Fig. 2: Inferred routing connectivity of a Virtex-6 tile. L is the manhattan distance that the wire traverses. Diagonal wires with $L = 2$ represents a span from (x,y) to (x±1,y±1).

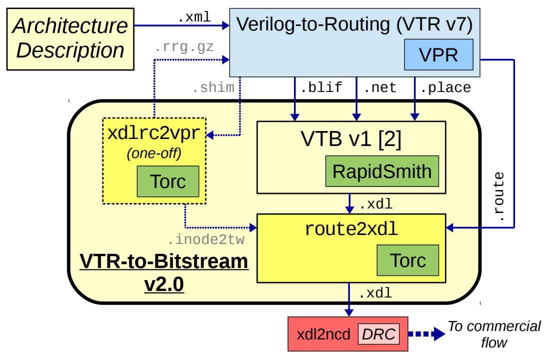


Fig. 3: Components of the VTR-to-Bitstream v2 flow.

source code shows that the only requirement is a directed graph where each node represents a resource containing its starting and ending coordinates, its type (e.g. a source, output pin, horizontal wire, etc.) as well as its timing cost.

We generate this new routing graph by stitching the graph generated by VPR's model to one extracted from the Xilinx device database (inclusive of LUT route-throughs) acquired from Torc. Our tool, `xdlrc2vpr` performs this task only once per architecture, where specifically it stitches each Xilinx output pin to be the only fanout from its corresponding VPR output pin. The equivalent task is also performed on all input pins, and all other resources are transformed into the VPR coordinate space and assigned the appropriate timing cost. By stitching, we preserve the interface of the routing graph through which VPR interacts, whilst modifying its internals to represent the true graph that exists on silicon. Later, when VPR has finished routing the circuit, we invoke our second tool `route2xdl` to translate the `.route` output from VPR back into the programmable routing switches (PIPs) used by the Xilinx netlist, described in the Xilinx Design Language (XDL) format.

Figure 3 illustrates how these two tools integrate into the entire VTR-to-Bitstream v2.0 flow. This flow is an extension of prior work: VTR-to-Bitstream v1.0 [2] (which itself uses

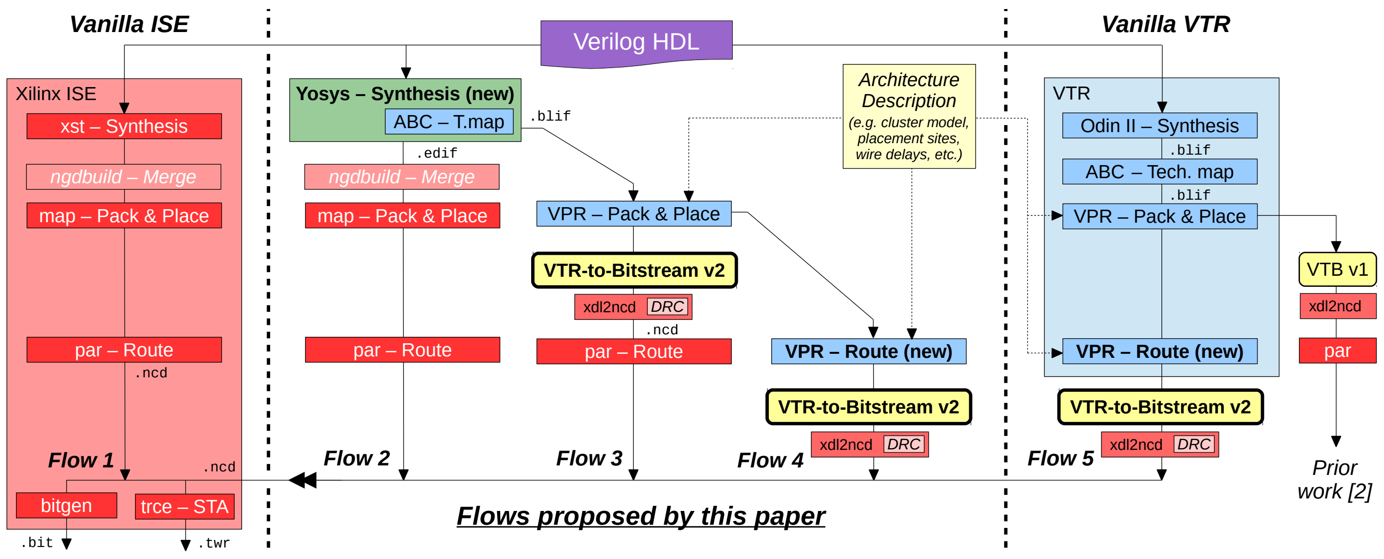


Fig. 4: Industrial comparison experiments (where flows 2–5 contain new synthesis/routing support over prior work).

RapidSmith [3] to interact with the XDL format) that is responsible for converting the `.blif` netlist, the `.net` packing result, and the `.place` result into an equivalent XDL netlist. This netlist would be similiar to the output from ISE's `map` tool. `route2xdl` then adds routing PIPs to this placed netlist, generating an output similar to ISE's `par`.

IV. Methodology

Experiments were conducted using the five different flows shown in Figure 4. Flow 1 describes vanilla ISE, in which a Verilog benchmark is compiled as normal entirely using industrial tools, primarily: `xst` for front-end synthesis, `map` for packing and placement, and `par` for routing. Once a legal circuit has been generated, static timing analysis and bitstream generation can occur.

Flows 2 to 4 describe an alternative flow for generating Xilinx circuits, with various tool stages replaced by academic equivalents. We apply the same ABC script used by VTR across flows 2–5: "`resyn; resyn2; if;`". For flow 2 only, we also enable the `-u` switch in `map` which prevents unused logic from being recursively removed to allow for a fairer comparison with VPR, which has no such ability.

In flows 3 through 5, we use our VTR-to-Bitstream tool to translate VTR's various output files into a fully placed and/or routed XDL netlist. During conversion of the XDL format into Xilinx's proprietary NCD format (necessary for timing analysis and bitstream generation), the netlist must pass Xilinx's design rule check (DRC) — which checks that all resources are legally placed, full connectivity from all net sources to all net and component sinks exists, and that there are no routing violations — thus providing us with confidence that we are creating valid netlists. For these same flows involving VPR, we apply the "`--allow_unrelated_clustering off`" option to prevent maximum density packing, which was previously found to lead to unrouteable circuits [2].

We target a mid-range Virtex-6 device (xc6vlx240t) found on the ML605 evaluation kit, using Xilinx ISE v14.4. This FPGA device is built on 40nm technology, and contains approximately 150K LUTs arranged into 37K logic slices. This logic, along with RAM and DSP resources, are spread over a 102x240 tile grid. In all experiments using the Xilinx tools, we set an aggressive 1ns clock constraint to encourage maximum CAD effort, matching behaviour in the academic flows. We conduct experiments across multiple placement seeds using the Imperial College HPC service [9], on servers with two Intel Xeon X5650 processors and 48GB RAM. However, to ensure accurate runtime metrics, we ran each experiment on its own exclusive server and measured using `/usr/bin/time`.

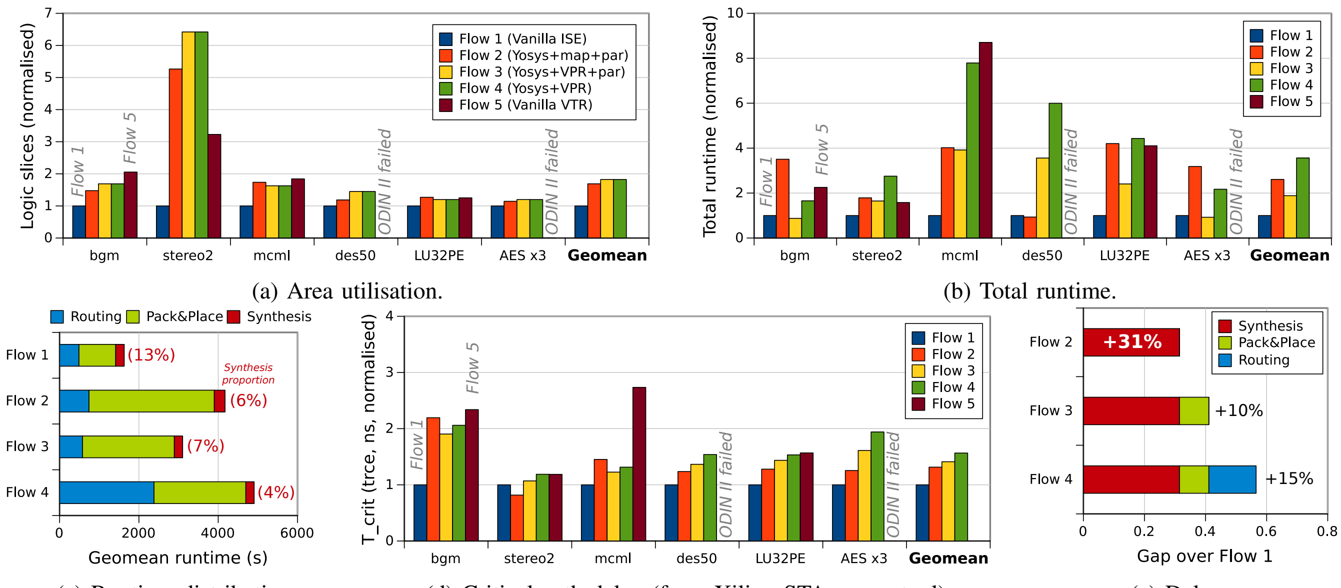
For the academic flow, we use Verilog-to-Routing v7.0 [1], along with Yosys v0.5 [10]. We evaluate across six benchmarks, consisting of the four largest benchmarks provided as part of VTR that fitted onto our FPGA (revisiting those reported in [2]) as well as des50, a smaller variant of the des90 benchmark from Titan [6] that does fit on our target device, and our own AES x3 benchmark composed of a chain of three 128-bit AES encoders [11] followed by three decoders.

Yosys front-end: Currently, a weakness of the academic VTR flow lies with its Verilog elaboration capabilities — the task of converting circuits described in this language into logic gates, flip-flops, and other resources — that is currently performed by Odin II [7], which does not fully support all language constructs such as those found in the des50 and AES x3 benchmarks. In flows 2–5, we replace Odin II with Yosys [10], which has gained traction as part of Qflow: an open-source digital synthesis flow for ASIC design. Yosys has extensive support for the synthesisable features of Verilog-2005, and is capable of outputting identical netlists in BLIF (for use in academic tools) as well as industry-standard EDIF. Internally, Yosys calls upon the same ABC tool as VTR for technology-mapping.

Timing model: Although Xilinx provide the complete routing resource graph in their XDLRC, no timing costs are given. By analysing static timing analysis reports (which lists only the total net delay), we were able to infer an approximate figure for each group of resources (e.g. all A6 LUT inputs, or all length-2 wires, etc.) through linear regression.

V. Industrial Comparison Results

Figure 5 compares the area utilisation, total tool runtime, and critical path delay of all six benchmark circuits (and their geometric mean) across flows 1–5, when operating from the



(a) Area utilisation.

(b) Total runtime.

(c) Runtime distribution.

(d) Critical-path delay (from Xilinx STA `trce` tool).

(e) Delay gap.

Fig. 5: Quality-of-Results summary (geomean across 10 seeds, normalised to flow 1: vanilla ISE).

same Verilog input. Unsurprisingly, the commercial flow (#1) outperforms the academic flow (#4) in all three figures of merit. In looking more closely at the area, such as the number of logic slices occupied by a circuit, it can be seen that the fully open-source flow 4 produces circuits that consume an extra 19–540% (geomean 82%) area over the vendor's flow 1 (which uses `xst` as its front-end synthesis tool). Although flow 2, and flows 3 & 4, are provided with the same logical netlist (either as EDIF or BLIF formats respectively) different packing decisions made by `map` and VPR result in different slice utilisations.

For the runtime metrics, we find that flow 4 can be between 1.6–7.8 times slower than flow 1 (geomean 3.5X). This is not surprising, since runtime is often a secondary concern for academic tools. However, it is worth noting that over-constraining the Xilinx tools to 1ns is not recommended, and can cause excessive runtimes that do not reflect typical industrial usage. We find that for mcml, a significant gap exists between flows 2 & 3, and flows 4 & 5, where the latter uses VPR for routing. Further investigation revealed that this benchmark contained two high fanout nets that each have around 40,000 sinks, requiring up to 200 seconds to route, consuming the majority (70%) of the total routing runtime.

Figure 5c shows the runtime distribution over the three main CAD stages. Immediately obvious is that synthesis consumes a reducing proportion of the total runtime budget (as annotated) yet the following results also show that it contributes to the largest gap in circuit performance. In addition, there is a significant difference in runtime between the industrial and academic (flow 4) router, though this can perhaps be attributed to the irregular nature of the imported Xilinx routing graph.

Lastly, Figure 5d provides a view of the mean circuit delay (as reported by Xilinx's static timing analysis tool `trce`) whilst Figure 5e illustrates the incremental delay gap between flows 2–4 over the baseline flow 1. The biggest gap (31%) exists when moving from flows 1 to 2, where only the synthesis tool is changed, as opposed to 10% and 15%, for packing & placement and routing respectively. This result lends credence to our claim: 'mind the (synthesis) gap'.

VI. Conclusion

In this work, we presented an open-source extension to the well-known VTR flow that has an improved Verilog front-end, as well as back-end support for creating valid bitstreams for a Xilinx Virtex-6 FPGA. This contribution is provided as a patch for download at http://eddiehung.github.io. Our second contribution was to make a rigorous, piecewise comparison between academic and industrial tools across the three main stages of the compilation flow, finding that the delay gap (as backed by a commercial STA tool) is 31% at the synthesis stage, 10% for packing & placement, and 15% for routing. The finding that the gap is largest at the front-end (which also consumes the least amount of runtime across the three stages) indicates that not only should researchers focus their efforts on improving back-end tools such as VPR, but that opportunities also exist at the upper levels, too.

Acknowledgements: Grateful for the support from the UK EPSRC (grants EP/I012036/1 & EP/I020357/1) and Xilinx.

References

[1] J. Luu *et al.*, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," *ACM TRETS*, vol. 7, no. 2, pp. 6:1–6:30, Jul. 2014.
[2] E. Hung *et al.*, "Escaping the Academic Sandbox: Realizing VPR Circuits on Xilinx Devices," in *IEEE FCCM'13*, Apr 2013, pp. 45–52.
[3] C. Lavin *et al.*, "RapidSmith: Do-It-Yourself CAD Tools for Xilinx FPGAs," in *IEEE FPL'11*, Sep 2011, pp. 349–355.
[4] N. Steiner *et al.*, "Torc: Towards an Open-Source Tool Flow," in *ACM FPGA'11*, Feb 2011, pp. 41–44.
[5] C. Beckhoff *et al.*, "GoAhead: A Partial Reconfiguration Framework," in *IEEE FCCM'12*, Apr 2012, pp. 37–44.
[6] K. E. Murray *et al.*, "Timing Driven Titan: Enabling Large Benchmarks and Exploring the Gap Between Academic and Commercial CAD," *ACM TRETS*, vol. 8, no. 2, pp. 10:1–10:18, Mar 2015.
[7] P. Jamieson *et al.*, "Odin II - An Open-source Verilog HDL Synthesis tool for CAD Research," in *IEEE FCCM'10*, 2010, pp. 149–156.
[8] Xilinx, "Virtex-6 FPGA Configurable Logic Block User Guide (UG364 v1.2)," Feb 2012.
[9] Imperial College, "High Performance Computing Service," http://www.imperial.ac.uk/ict/services/teachingandresearchservices/highperformancecomputing.
[10] C. Wolf, "Yosys Open SYnthesis Suite," http://www.clifford.at/yosys/.
[11] Altera, "Advanced Synthesis Cookbook," http://www.altera.co.uk/literature/manual/stx_cookbook.pdf, July 2011.