Escaping the Academic Sandbox: Realizing VPR Circuits on Xilinx Devices

Eddie Hung, Fatemeh Eslami, and Steven J. E. Wilton
Department of Electrical and Computer Engineering
University of British Columbia, Vancouver, Canada
{eddieh, feslami, stevew}@ece.ubc.ca

***Abstract*—This paper presents a new, open-source method for FPGA CAD researchers to realize their techniques on real Xilinx devices. Specifically, we extend the Verilog-To-Routing (VTR) suite, which includes the VPR place-and-route CAD tool on which many FPGA innovations have been based, to generate working Xilinx bitstreams via the Xilinx Design Language (XDL). Currently, we can faithfully translate VPR's heterogeneous packing and placement results into an exact Xilinx 'map' netlist, which is then routed by its 'par' tool. We showcase the utility of this new method with two compelling applications targeting a 40nm Virtex-6 device: a fair comparison of the area, delay, and CAD runtime of academia's state-of-the-art VTR flow with a commercial, closed-source equivalent, along with a CAD experiment evaluated using physical measurements of on-chip power consumption and die temperature, over time. This extended flow — VTR-to-Bitstream — is released to the community with the hope that it can enhance existing research projects as well as unlock new ones.**

I. Introduction

Ever since Versatile Place and Route (VPR) introduced to the academic community as an open-source project by Betz and Rose in 1997 [1], a large number of research groups across the world have since adopted it as the de-facto FPGA CAD tool on which to base their experiments. Given a pre-mapped logic netlist, VPR has enabled these researchers to explore two types of innovations: (1) new, theoretical FPGA architectures — for example, the structure of a new logic block, and (2) to experiment with the different packing, placement and routing algorithms that target these architectures. In this paper, we provide researchers with the ability to pursue (2) by allowing their CAD innovations to be evaluated on real commercial FPGAs with minimal effort, rather than just using theoretical models.

The latest evolution of VPR, version 6.0, forming part of the Verilog-To-Routing (VTR) project, unifies a Verilog synthesis (ODIN II) and tech-mapping (ABC) tool into a complete flow capable of taking a circuit described in Verilog all the way through to a place-and-routed FPGA [2]. Enabled by this new support, this paper extends the Verilog-To-Routing flow to allow the academic community to generate high-quality, working bitstreams that can be programmed onto commercially-available FPGAs. We believe there is tremendous value in escaping the academic sandbox — up until now, only closed, proprietary (and costly) CAD tools could be used to target physical FPGAs. This disconnect is highlighted in Figure 1, and has a number of disadvantages: CAD innovations, commonly built upon VPR, that are made by the research community cannot be applied to real industrial applications unless adopted by the tool vendors, and there exists no way to physically validate the models that exist in VPR given that fabricating a custom FPGA is beyond the means of most research groups.

Whilst the use-cases for this extended flow are many, in this paper we explore two of the more prominent questions: how do the current standard of academic CAD tools compare to their vendor equivalents when targeting an identical architecture, and is it possible to physically measure the effect of CAD innovations? In the first application, because we are targeting a near-identical architecture that is compatible with vendor tools, we are able to make much fairer, and more accurate, comparisons than have been done in the past. Secondly, given that we are able to realize our circuits on physical FPGAs, we can now make physical measurements — we showcase this with a figure-of-merit which is receiving an increasing amount of attention: power.

In this work, we will be targeting Xilinx devices exclusively through using RapidSmith [3]: a framework for creating human-readable netlists in the Xilinx Design Language (XDL) format that can subsequently be converted into the proprietary NCD representation native to the vendor toolchain. There exists no reason why this approach would not be extensible to devices from other FPGA vendors if the necessary information is released in the future. An additional, unique, feature of using Xilinx devices from the Virtex-5 family and above is the presence of a System Monitor block which allows on-chip measurements of voltage and current required for power consumption.

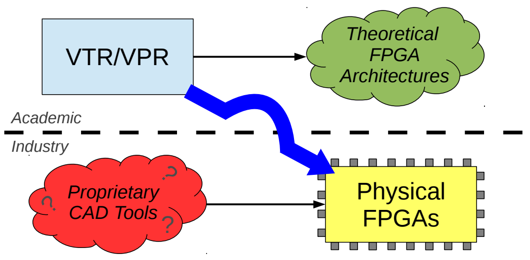


Figure 1: Escaping the Academic Sandbox

Our objective is to enable this flow whilst making the minimum set of modifications to the vanilla VTR project, with the intention of providing a pure platform from which researchers can extend their existing work onto real devices without requiring a significant effort to port or re-base their code. Furthermore, by providing a path to bitstream, we are hopeful that this would invite industry to consider using the VTR project in their own applications, and perhaps more importantly, push for and even contribute their own new features and improvements.

II. Related Work

Perhaps most related to this current work is the Altera Quartus II University Interface Program (QUIP) [4], which allows researchers to inject and extract circuits from various stages in the Quartus II CAD tool. Packed VPR circuits can be converted into the Verilog Quartus Module (VQM) format (though no tool currently exists to do so for the latest hetereogeneous architectures) and manual circuit placement is only possible through applying placement constraints and then re-running the Quartus II placer. Currently, to our knowledge no open-source tool is available to allow VPR 6.0 circuits to be automatically converted into an Altera bitstream in the way this work does for Xilinx netlists. A previous foray into opening up Xilinx Virtex-II devices (0.13um) for research was made by the JHDLBits project [5] — a combination of JHDL and JBits tools to enable both high-level parameterized access for Xilinx primitives, as well as low-level access directly into the FPGA bitstream. Besides being restricted to targeting an obsolete Virtex-II architecture, this work does not leverage the timing-driven VPR tool on which a significant amount of CAD research is based.

More recently, a number of works such as GoAhead and Torc [6], [7] have been created to aid dynamic partial reconfiguration research into Xilinx devices using the XDL interface — allowing users to interact with those FPGAs in a way that is unsupported by the vendor tools. Torc provides a high-level interface that allows users to manipulate circuits at many different levels of the CAD flow — from an industry-standard EDIF netlist through to a Xilinx bitstream — but this does not include direct integration with VPR. An alternative approach to open-access to FPGAs is taken by the ZUMA project [8], which embeds an "FPGA-on-an-FPGA" to provide an abstraction layer capable of implementing the same bitstream on both Xilinx and Altera devices. Unfortunately, the minimum area overhead of this additional layer is 40X, impairing the utility of any physical on-chip measurements of power or temperature.

III. VTR-To-Bitstream

Presently, we utilize the VTR project (version 1.0) to generate a partial FPGA circuit implementation — a synthesized, packed, and placed netlist — before translating that result into a proprietary Xilinx format which is then

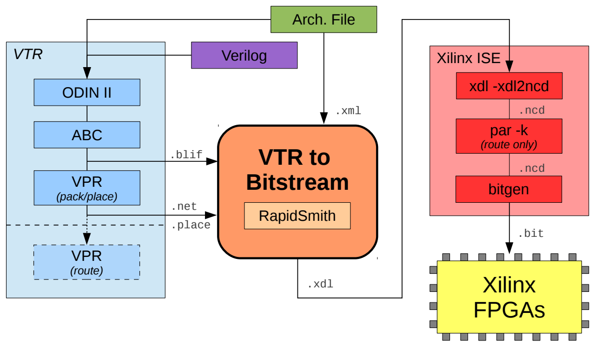


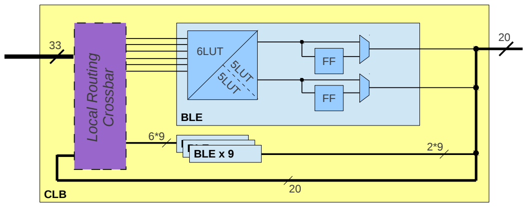
Figure 2: Proposed Toolchain

inserted back into the standard Xilinx ISE flow for routing, timing analysis and bitstream generation. Although VTR and VPR supports FPGA routing, due to the complex routing structures that exist in Xilinx FPGAs, we currently omit this feature and defer this task to the vendor tools. This flow is illustrated in Figure 2.

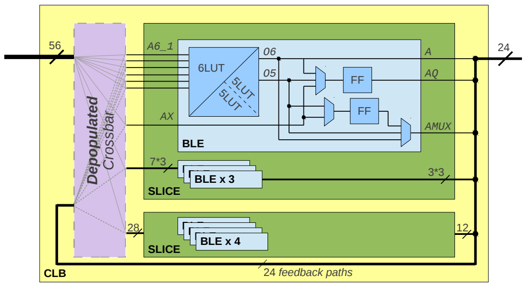
The VTR project takes as its inputs an architecture description (`.xml`), and a Verilog circuit description. The first stage, ODIN II, is responsible for parsing and elaborating the Verilog input into a technology-independent netlist, output into the BLIF format. This stage is also responsible for inferring hard multiplier blocks from the HDL '`*`' operator as blackbox primitives, as well as converting any instantiated RAM instances into single-bit primitives consistent with the provided architecture description.

Next, ABC performs logic optimizations and technology-mapping on the input BLIF netlist so that all soft-logic conforms to the maximum lookup-table size supported by the FPGA architecture. The output of ABC is also a BLIF file. In order to measure the state-of-the-art, we have checked out the latest version of ABC toolkit (revision 1291 from [9]) that supersedes the version packaged with VTR 1.0, and which enables more aggressive logic optimizations to be performed (in particular, the construction of functionally-reduced AIGs for large circuits). Our experiments have shown these to give a significant improvement to the area and delay of the final implementation.

The third and final stage of the VTR flow is VPR, which is responsible for packing each of the LUTs and FFs into a logic cluster, as well as multipliers and memories into their most appropriate blocks. During packing, the objective is to combine as many related logical elements into the same block as possible, as doing so will reduce the distance that each net will need to travel (its wirelength) and hence improve timing. This packed netlist is written to an XML file with a `.net` extension. Once all of this logic has been packed, VPR then places them into legal locations on the FPGA, again with the objective of reducing the total wirelength of all nets in the circuit. The final placement is recorded in a text-based `.place` format. The tech-mapped `.blif` netlist,



(a) Default VPR Model (fully-populated input crossbar)



(b) Virtex-6 CLB (full external inputs and depopulated feedback crossbar)

Figure 3: VPR Logic Cluster Models

`.net` packing and `.place` placement is then fed into our VTR-to-Bitstream tool. With the aid of RapidSmith [3], these results are then faithfully translated into a single text-based Xilinx XDL file which captures all LUT masks taken from the `.blif`, their packing and all net connections as described in the `.net`, as well as their exact placement sites on the FPGA from `.place`. To maintain compatibility with RapidSmith, our translator tool is built using the Java language.

With this `.xdl` netlist, the circuit can now enter the standard Xilinx ISE flow by first converting this text-based file into the native binary `.ncd` format by using the Xilinx `xdl` tool with the `-xdl2ncd` switch. The resultant file can then be provided to the Xilinx `par` router with the `-k` switch to indicate that only routing is to be performed. It is also worth pointing out here that for Virtex-5 and above architectures, packing and placement is performed in a combined fashion at the `map` stage of the ISE flow, and can no longer be placed by `par`. The routed output is then written to a different `.ncd` file, which can then be provided to the `bitgen` tool to generate a bitstream that can be programmed onto the device as normal.

A. *Modelling the Logic Cluster (CLB)*

In this work, we target the Virtex-6 architecture though no fundamental reason exists why this technique would not extend to past and future device families; XDL support exists for the very latest Virtex-7 parts.

Key to the success of the flow illustrated in Fig. 2 is a detailed VTR architecture description which captures as many of the advanced features available in the physical device as possible. Figure 3a shows the default VPR model for a representative FPGA logic cluster containing N=10 BLEs (Basic Logic Elements), each of which supports a K=6 -input lookup-table which can be fractured into two 5-input LUTs, each connected to its own dedicated flip-flop with a multiplexed output. There are I=33 external inputs to each logic cluster (derived from the rule-of-thumb: $I = \frac{K}{2}(N+1)$), which pass through a fully-populated local-routing crossbar that allows any crossbar input to connect to any crossbar output. This input set is also supplemented by a feedback set made up of all 20 BLE outputs.

In contrast, our own VPR model for the Virtex-6 logic cluster, which Xilinx calls a CLB, is shown in Figure 3b and was derived from its user guide [10]. Here, each CLB is actually divided into two SLICEs, each of which contain four BLEs. In Xilinx architectures, each SLICE can be a SLICEM or SLICEL type, with the former capable of also implementing distributed memory, which are not currently supported by VTR; hence we treat all SLICEMs as SLICELs and use them as logic resources only. Each BLE in the Virtex-6 contains a fracturable 6LUT (with `O6` and `O5` outputs) and two FFs, but contains a much more flexible interconnect to feeds its three outputs: a combinational only output (`A`), a sequential output (`AQ`) and one dual-purpose output (`AMUX`). In addition, an extra bypass BLE input (`AX`) exists to allow either of the two FFs to be utilized independently of the LUT. We feel that it is a true testament to the flexibility of the VTR's architecture description language that such a complex structure can be captured without any modifications.

The intra-cluster local routing in our Virtex-6 model is no longer fully-populated: firstly, the number of external inputs I=56 — meaning that each BLE input pin can be reached directly from the global interconnect — doing away with an input crossbar altogether. Secondly, through careful study and experimentation, we have been able to discover that each BLE output can only connect to 6 LUT inputs (and conversely, each LUT input can only be reached by 3 BLE outputs) — the exact pattern found on the device is also modelled in our architecture file. This matches previous research, which found that fully-populated crossbars are not preferred due to their large area costs [11]. Lastly, this architecture description is populated with accurate (worst-case) LUT, CLB, IOB, DSP and BRAM delays as reported in the Xilinx datasheet [12].

B. *Modelling the FPGA Layout*

Another area where a disconnect lies between the accepted academic FPGA model and commercial devices is when considering the FPGA layout. In VPR, it is assumed that the I/O blocks of an FPGA device follows its perimeter; but whilst the number of I/O signals that can be supported at each site — its capacity — can be adjusted from the architecture file, its locations cannot be modified. In contrast, Xilinx devices do not contain any horizontal rows of I/O blocks along the top and bottom perimeters of the FPGA, they can however contain multiple columns of I/Os which are not just constrained to the sides. In the device that we chose to target, three I/O columns exist: one at the left perimeter, and two columns in the middle of the device. No I/Os exists on the right side of the device (instead, the right perimeter is occupied by PCIe and Ethernet cores) as seen in Figure 4. We have successfully modified VPR to extend support for describing I/O columns in the same manner as heterogeneous blocks (RAM, DSP, etc.).

Besides possessing I/O blocks in different columns across a device, in the Virtex-6 architecture these blocks can support two I/O signals per site, but exist only in every-other (odd) row. Furthermore, not all I/O sites may be bonded — or connected — for every chip package: as an example, the LX240T device that we target in this work is available in the FF784 (with 400 user-accessible pins), FF1156 (600) and FF1759 (720) package formats. For the FF1156, all I/Os at and above row 200 are unbonded and hence unavailable for designer use. In addition, large embedded hard-blocks — such as CPUs or high-speed I/O cores — may also exist as exceptions to the uniform device fabric. In our device, an embedded CPU occupies a 5x80 tile region in the centre of our device which can be seen in Fig 4b, on top of the irregular I/O arrangement mentioned earlier. To support these exceptions, it was necessary to modify the VTR architecture file so that such regions of the device can be marked as unavailable during placement.

C. Xilinx Architecture Support

Currently, our extended VTR-to-Bitstream flow supports:
Architecture

1) 6-input Lookup Tables fracturable into two 5-input LUTs
2) Two flip-flops per Basic Logic Element with LUT bypass
3) Global net (BUFG) support for clock distribution

Block RAM

4) Individually fracturable between a 36K or two 18Ks
5) Simple or True dual-port configuration
6) Configurable width/depth, from 1bit*32K to 72bits*512

DSP

7) 25x18 combinational multiplier

Missing from this list is support for carry-chains as well as distributed LUT-RAM. Primarily, the key barrier to supporting these advanced features lies with the upstream VTR project, which currently do not infer these structures from the HDL circuit description. Similarly, support for utilizing Block RAM resources as FIFOs, for other DSP functionality such as Multiply-Accumulate (MAC), barrel-shifting and pattern matching, as well as high-speed I/O interfaces are also missing. We believe that if new support for any of the features discussed were to be added in a future release of VTR, it would be trivial to extend them to the bitstream level.

IV. Framework

By default, VPR will perform packing and placement (and routing) with the objective of minimizing area and delay — in terms of logic blocks, placement wirelength, and routing channel width — doing so allows designers to benchmark the quality of their CAD algorithms in terms of these abstract metrics. However, when targeting an existing device, there is no flexibility to change the FPGA array size nor the number of routing tracks that exist.

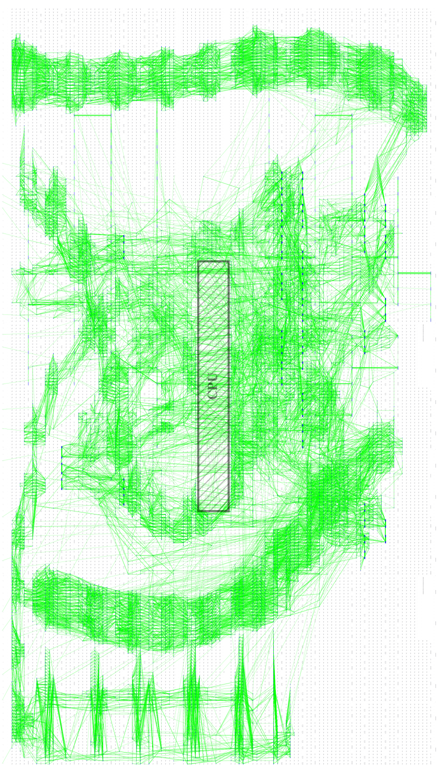
In this work, we will be targeting the Xilinx Virtex-6 `xc6vlx240t-1ffg1156` FPGA device, which is included on the ML605 Evaluation Kit. This device provides 38,000 slices (152,000 LUTs), 416 (15Mbits) Block RAM resources and 768 DSP blocks, across a 100x240 tile array. Conveniently, the VTR architecture description allows for a fixed FPGA size to be specified for the device (rather than an auto-sizing aspect ratio) which was utilized. Targeting a fixed FPGA size, as opposed to a minimum-sized FPGA, means that it was not necessary to pack as densely as possible — as long as the final implementation fits onto the device. For this reason, when running VPR we have disabled the default `--allow_unrelated_clustering` feature, which otherwise packs each CLB as aggressively as possible with unrelated logic solely to minimize area, with no apparent benefits to timing.

Our experiments have shown that whilst keeping this option enabled reduced the average number of logic slices used by 18% when packing each CLB with the full 8 LUTs, it also doubled the placement (bounding box) cost. More importantly, we discovered that this resulted in only the smallest benchmark circuit (bgm) being routeable on our Virtex-6 device (which also does not have the luxury of adjusting its routing channel width). Hence, we believe that fully packing each logic cluster in this manner is unrealistic — this is further reflected in the number of LUTs per slice (presented later) that the vendor tools themselves return.

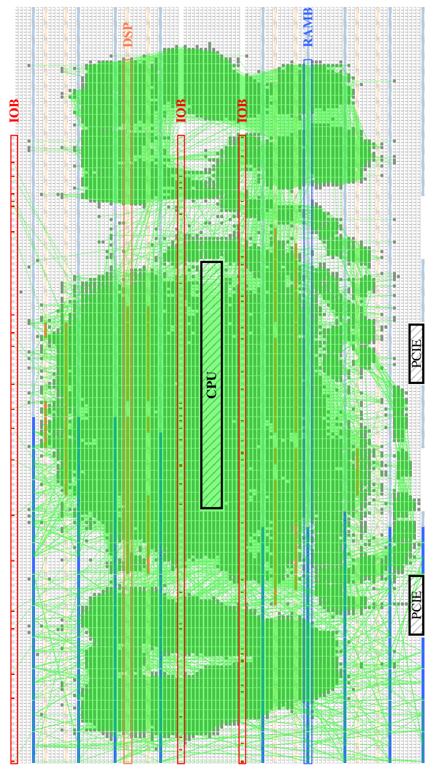
For each circuit, we randomly selected a pin location for each I/O signal and constrained both the VTR and ISE flows to use the same assignment, to emulate a realistic, but worst-case, usage scenario. The only signal which was not randomly placed in this manner was the clock pin, which was fixed to the "`J9`" (56,119,1) location containing dedicated paths to global clock buffer resources (BUFG) [13]. This was necessary as currently, VPR does not have any understanding as to where these clock-buffers are positioned on the FPGA, and so is unable to optimize for this pad location. For our ISE experiments, we also constrain the circuit to a clock period of 1 ns in order to force the tools to work as hard as possible, matching VTR behaviour.

V. Appl. 1: How does VPR compare with ISE?

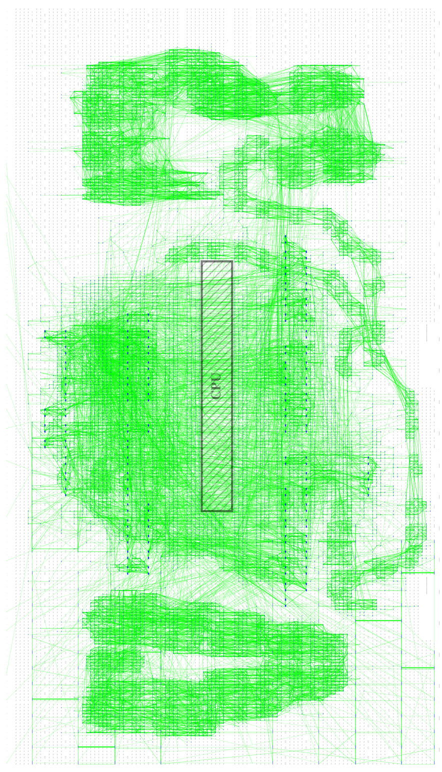
The first application that we chose to explore using VTR-to-Bitstream was to answer the question: how does the current standard of academic CAD tools compare with their commercial equivalents? Rather than intend this to be an apples-to-apples competition, we see this as an opportunity for the academic community to learn from robust industrial-



(a) Xilinx ISE placement viewed from FPGA Editor



(b) VPR Packing/Placement from GUI



(c) Translated VPR circuit, viewed from FPGA Editor

Figure 4: Placement visualization for benchmark circuit `mcml` (best viewed in colour)

strength tools in order to identify broad areas on which future improvement and research efforts should be focused.

For the initial step, we visualized how the placement of a Verilog circuit input into the VTR flow differed from the same Verilog fed into the Xilinx flow. Figure 4a shows the placement generated by Xilinx ISE, when exported from Xilinx FPGA Editor, whilst Fig. 4b visualizes the VPR placement as displayed in its GUI. Lastly, Figure 4c shows the same VPR result after it has been translated into XDL and then `.ncd`, as viewed again from Xilinx FPGA Editor. In all cases, we have only re-coloured, re-ordered and/or changed the opacity of these figures to enhance their clarity.

Interestingly, the VPR placement looks vastly different to the ISE placement both in terms of the number of blocks utilized (this is quantified as being 15% more in the results that follow) as well as in routing complexity (34% more routing resources). At first glance, this result suggests that ISE employs a more efficient high-level synthesis engine (which is able to reduce the number of global nets) as well as a more effective placement algorithm, which is able to produce a more compact rat's nest visualization. ISE's disjointed placement result also supports the assertion that analytical placement techniques are used [14], which differ from the simulated annealing approach taken by VPR.

Next, we compared key metrics between the two flows: Table I shows the circuit quality between VTR-to-Bitstream and Xilinx ISE for the four largest circuits supplied in the VTR project, which occupies between 14–51% of our LX240T FPGA device. Each circuit is represented by four columns: "ISE" which represents a run of ISE at its default settings, "ISE--" which represents an intentionally crippled run where advanced features were disabled in order to make for a fairer comparison, "VTR" for the VTR-to-Bitstream flow, and a final "%" column which provides some context for how much of the total device resources were consumed by VTR. For "ISE--", the tool was configured to match the VTR flow as closely as possible: optimize for minimum area, disable inference for carry-chains, shift-registers and FSM extraction, as well as resource-sharing, global net promotion and multiplexer-extraction. Comparison between VTR and ISE is made in the final two columns, where the geometric mean of their ratios are presented.

A number of key trends are visible in these results. Firstly, in three of four cases, VTR uses more logic than default ISE (on average 1.3X more slices and 1.6X more LUTs) and in two cases more than ISE-- too (1.01X slices, 1.4X LUTs). We believe that this is due to a more mature HDL front-end that can optimize logic more efficiently, as well as original ISE's support for advanced architectural features. The increase in

	bgm				stereovision2				mcml				LU32PEEng				VTR/	VTR/
Metric	ISE	ISE--	VTR	%	ISE	ISE--	VTR	%	ISE	ISE--	VTR	%	ISE	ISE--	VTR	%	ISE	ISE--
Slices	6245	6922	5338	*14*	2400	3513	6631	*17*	13761	15388	15838	*42*	19585	27430	19406	*51*	1.28	1.01
LUTs	15406	19149	18234	*12*	5358	4867	18452	*12*	46324	52603	59583	*39*	50639	72417	62591	*41*	1.60	1.37
+ O5 and O6	1510	2241	3521		731	1517	8707		19592	9716	34333		10719	10013	14058		2.83	2.59
LUTs per Slice	2.5	2.8	3.4		2.2	1.4	2.8		3.4	3.4	3.8		2.6	2.6	3.2		1.24	1.35
Registers	6362	6319	3705	*1*	9420	11130	16764	*5*	18724	50058	50177	*16*	14777	16509	19656	*6*	1.39	1.01
RAMB36	-	-	-	-	-	-	-	-	159	159	124	*29*	147	147	-	-	1	1
RAMB18	-	-	-	-	-	-	-	-	-	-	70	*8*	6	6	300	*36*		
DSP48	22	22	22	*2*	480	474	564	*73*	106	106	176	*22*	64	64	64	*8*	1.18	1.19
BUFG	16	1	1		1	1	1		3	1	1		1	1	1		0.38	1
IOB		289		*48*		331		*55*		69		*11*		216		*31*	-	-
map time (s)	317	294	-		454	475	-		1417	1914	-		1317	1826	-		-	-
map mem (MB)	947	970	-		1373	1394	-		1939	2262	-		1862	2168	-		-	-
VPR time (s)	-	-	1728		-	-	2570		-	-	25856		-	-	15757		9.06	7.80
VPR mem (MB)	-	-	2558		-	-	2758		-	-	3717		-	-	3882		2.16	1.98
par time (s)	307	384	225		410	450	244		640	856	1075		740	854	787		0.94	0.78
par mem (MB)	1027	1060	1024		1479	1488	1199		1875	2039	2159		1833	2110	1806		0.98	0.92
Routing PIPs	391K	481K	401K		306K	314K	601K		1012K	1257K	1359K		1403K	1921K	1591K		1.32	1.09
Logic depth	14	22	25		6	15	13		52	113	133		84	190	133		1.99	0.95
T_crit (ns)	7.75	9.80	19.61		15.4	15.81	16.42		30.08	57.96	101.58		48.37	85.87	101.07		2.09	1.44
% Routing Delay	74.4	78.1	71.4		7.6	27.8	87.0		55.8	80.1	80.2		70.7	81.2	87.0		2.10	1.32

Table I: VTR and Xilinx ISE comparison for Virtex-6 — "ISE--" column indicates a crippled instance for fairer comparison

the number of LUTs used is less than the number of slices due to a large proportion of the logic generated by VTR being combined into fracturable LUTs: 2.8X and 2.6X more LUTs with both inputs (O5 and O6 entry) utilized for ISE and ISE--. Equally interesting is that on average, VTR packs more logic into each slice than either of the two vendor flows — 1.2X and 1.4X — even though ISE-- is also optimizing for minimum area. In terms of heterogeneous blocks, VTR infers the same number of RAMB blocks (when considering that each RAMB36 component can be fractured into two RAMB18s) but approximately 1.2X more DSPs are used than for ISE.

The following row of results compare the CPU and memory requirements of the three flows. We have omitted the smaller runtimes for the front-end of each flow (xst, ngdbuild for ISE, and ODIN II and ABC for VTR) and concentrated on comparing Xilinx's map tool with VPR — both of which are responsible for packing and placing the circuit. We translate the VPR outputs into the `.ncd` format, after which both placed results are then routed using Xilinx's par tool. We provide the CPU runtime and peak memory usage for all three tools (measured using the `/usr/bin/time` tool, when run on an Intel Core 2 Quad 2.8 GHz workstation with 8 GB of RAM. Also reported are the number of routing PIPs (resources) that each routed circuit mapping uses — we present this as a very rough estimate for routing resource utilization. These results show that VPR is up to 9X slower than map, and consumes ~2X more memory. Interestingly, when the subsequent par routing stage is considered, VPR's placement result is perhaps even slightly easier to route than for the Xilinx solutions. Furthermore, the number of PIP routing resources required to implement the VPR result is only a fraction more than for the vendor tool; however, a strong correlation with logic utilization exists.

Lastly, the critical-path logic depth and delay are presented. Here, we see a clear difference in the quality of the two flows: VTR is on average 2.1X slower than ISE, and 1.4X slower than ISE--. When compared to full ISE, VTR circuits have twice the critical-path logic depth, which we believe can be attributed to advanced architectural features such as carry-chains. A fairer comparison can be made with ISE-- where many of these features are disabled; here, even though the logic depth is approximately the same, the circuits are still much slower. Given that the logic/routing proportion of delay is similar between the ISE-- and VTR flows, we can conclude that the packing and placement algorithms in VPR are not as efficient as those in commercial tools, which we expect to be highly-tuned and possess more complete information for their particular architecture.

VI. Appl. 2: Measuring Physical Power

A second application that being able to program real FPGAs unlocks is the ability to measure physical power draw of circuits. Xilinx devices from the Virtex-5 family and above contain a System Monitor block, built around an Analog-to-Digital Converter, which allows physical on-chip operating parameters, such as power supply voltages and die temperature, to be acquired [15]. Given that VTR-to-Bitstream allows the CAD tools which target physical FPGAs to be modified, we can now accurately evaluate the effect of power-optimizations, as opposed to just using models.

Typically, CAD researchers will use a flexible power model (such as one described in [16]) to evaluate the effects of their algorithmic innovations [17]. However, such vector-less power models can be inaccurate — reference [16] reported over-estimating the power of LUTs and under-estimating switch-boxes, which may result in misleading conclusions. In these situations, it would be desirable to

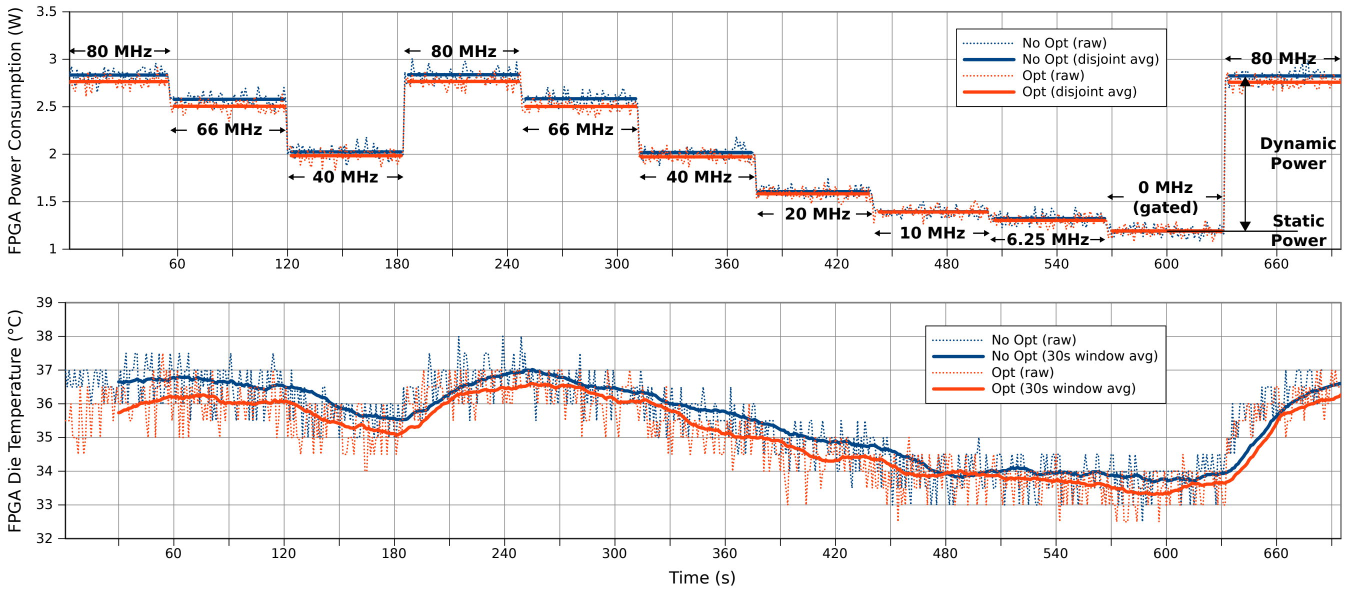


Figure 5: FPGA physical on-chip measurements — (above) power consumption, (below) die temperature

validate any conclusions found with experimentally derived physical measurements; Wilton et al. do so by experimentally quantifying the impact of pipelining on FPGA power in [18].

In this work, we seek to measure the power profile, over time, of the large LU32PEEng benchmark circuit both before and after applying CAD optimizations when mapped onto the Xilinx Virtex-6 device found on the ML605. In the absence of a large, highly-pipelined benchmark, like reference [18] we forcibly pipeline a circuit by inserting a flip-flop at the output to every lookup-table (thereby changing the functionality of the circuit, but maintaining its logic characteristics somewhat) in order to boost its maximum clock frequency, as estimated by static timing analysis, from 9 MHz to over 80 MHz (where routing delay now dominates) and also stimulate the data inputs of the circuit using vectors driven by a linear feedback shift-register, which were verified to activate the circuit in simulation. Due to these low-level circuit modifications in VPR, we are unable to extract the *identically* pipelined circuit to re-insert into ISE for a fair comparison; this remains a topic for future work. In contrast to [18], due to architectural support we are now able to report true device-level static and dynamic power measurements, as opposed to board-level values. Additionally, we are evaluating a modern, high-end 40nm FPGA, for which the balance of static and dynamic power has shifted since the 0.13um device used previously.

Figure 5 shows on-chip measurements for power consumption and die temperature, as acquired using the built-in System Monitor block and logged using Xilinx ChipScope software. The exact same pipelined LU32PEEng (BLIF) circuit was used, but with and without a simplistic power-guided CAD optimization. These results show a profile of both metrics over time, during which we dynamically alter the circuit's clock frequency by using the on-chip MMCM clock manager and dedicated clock multiplexing resources to re-synthesize a 200 MHz reference, thereby emulating a processing unit adapting to a varying workload. In this work, we seek only to showcase the utility of being able to evaluate such applications when real bitstreams can be generated, therefore our CAD optimization is simplistic in that it merely considers the activity of each affected net (as estimated using ACE 2.0 [16]) during clustering and placement costing by giving high activity blocks more criticality. The results show a small, but noticeable, 3% power saving at 80 MHz, which correlates with the die temperature. We anticipate that, with more advanced CAD optimizations, further power savings can be measured.

VII. Future Work and Challenges Ahead

The current status of this VTR-to-Bitstream project is that, due to its complexity, routing is deferred to proprietary Xilinx tools. Supporting this final stage would allow researchers to test new routing algorithms and perform experiments across the entire FPGA CAD flow. One of the key challenges with this task is that VPR is unable to model the intricate global routing architecture present on Xilinx devices. For example, Virtex-6 switch boxes contain a diverse mix of short, medium and long routing tracks, some of which are allowed to connect directly to CLB inputs, whilst others can only be used for turning. In addition, flexible “bounce” points exist to allow routing tracks, as well as CLB input and feedback connections, to make indirect connections. These cannot be modelled by the scalable, but homogeneous, routing architecture supported currently. We believe this would be possible if newer versions of VPR allowed this routing architecture to be described with the same flexibility as its logic blocks. This would then enable applications which require low-level access to routing resources: [19] proposes that circuits be rapidly re-compiled using incremental techniques for debug insertion.

In this paper, we used the `xdl` tool to perform "xdl2ncd" translation — however, a reverse path "ncd2xdl" also exists. This feature would allow users to decode binary Xilinx netlists in order leverage ISE's front-end, such as its HDL synthesis engine, or its packing or placement algorithms for further research. Lastly, whilst we have been able to verify that the VTR-to-Bitstream flow functions correctly from end-to-end for small circuits (for example, a state machine which drives an LED display according to the switches on our ML605 board) we would like to extensively verify much larger circuits. This would be possible by using post-synthesis (gate-level) simulation, or by using formal verification techniques, both of which can be enabled by the `netgen` tool that is provided by Xilinx ISE.

VIII. Conclusion

FPGA CAD research has typically been conducted using open-source academic tools, such as VPR and VTR, which allow designers to apply their innovations to theoretical FPGA architectures. This work proposes an alternative: rather than evaluating these improvements using only the models present in those tools, the VTR-to-Bitstream extension presented in this paper allows researchers to validate their work on real, commercial devices on which physical measurements can be taken. Unlike previous work which allows a path to bitstream, our work is built on top of the de-facto VTR flow with only a minimal set of modifications, lowering the barrier to entry. Currently, the project is capable of exporting a packed and placed (but unrouted) circuit from the Verilog input fed into VTR, which can then be imported into Xilinx ISE flow for routing, timing analysis, and bitstream generation.

We have showcased the utility of this project with two compelling applications: firstly, a comparison of the existing VTR flow with its commercial equivalent, Xilinx ISE. Results showed that VTR produced circuits that were competitive with ISE when comparing their area utilization, but was ~2X slower for delay. Analysis indicates that this discrepancy is due to several unsupported architecture features, such as carry-chains, as well as more efficient (and highly-tuned) CAD algorithms. Our second example application was to physically measure the on-chip power consumption and die temperature of the FPGA, for the same circuit with and without a simplistic CAD power optimization added to VTR; measured results showed that when enabled, a 3% power and 0.5°C temperature reduction was achieved.

We hope that the research community will find this project useful in their own work, and encourage interested parties to download the VPR-XDL translator tool, the VPR patch and the Xilinx architecture file used in this work from: http://ece.ubc.ca/~eddieh.

References

[1] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, ser. FPL '97, 1997, pp. 213–222.

[2] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson, "The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing," in *Proceedings of the 20th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA'12, Feb. 2012, pp. 77–86.

[3] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "RapidSmith: Do-It-Yourself CAD Tools for Xilinx FPGAs," in *Proceedings of the 21st International Workshop on Field-Programmable Logic and Applications (FPL'11)*, September 2011, pp. 349–355.

[4] Altera, "Benchmark Designs For The Quartus University Interface Program (QUIP) Version 1.1," http://www.altera.com/support/software/download/altera_design/quip/quip-download.jsp, Feb 2008.

[5] A. Poetter, J. Hunter, C. Patterson, P. Athanas, B. Nelson, and N. Steiner, "JHDLBits: The Merging of Two Worlds," in *Field Programmable Logic and Application*, ser. Lecture Notes in Computer Science, 2004, vol. 3203, pp. 414–423.

[6] C. Beckhoff, D. Koch, and J. Torresen, "GoAhead: A Partial Reconfiguration Framework," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, April 2012, pp. 37–44.

[7] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, and M. French, "Torc: Towards an Open-Source Tool Flow," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '11, 2011, pp. 41–44.

[8] A. Brant and G. G. Lemieux, "ZUMA: An Open FPGA Overlay Architecture," *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*, pp. 93–96, 2012.

[9] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification, Release 70930," https://bitbucket.org/alanmi/abc, November 2012.

[10] Xilinx, "Virtex-6 FPGA Configurable Logic Block User Guide," http://www.xilinx.com/support/documentation/user_guides/ug364.pdf, February 2012.

[11] G. Lemieux and D. Lewis, "Using Sparse Crossbars within LUT Clusters," in *Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '01, 2001, pp. 59–68.

[12] Xilinx, "Virtex-6 FPGA Data Sheet: DC and Switching Characteristics," http://www.xilinx.com/support/documentation/data_sheets/ds152.pdf, January 2012.

[13] ——, "Virtex-6 FPGA Clocking Resources User Guide," http://www.xilinx.com/support/documentation/user_guides/ug362.pdf, May 2012.

[14] S. Gupta, J. Anderson, L. Farragher, and Q. Wang, "CAD Ttechniques for Power Optimization in Virtex-5 FPGAs," in *Custom Integrated Circuits Conference, 2007. CICC'07. IEEE*, 2007, pp. 85–88.

[15] Xilinx, "Virtex-6 FPGA System Monitor User Guide," http://www.xilinx.com/support/documentation/user_guides/ug370.pdf, June 2010.

[16] J. B. Goeders and S. J. E. Wilton, "VersaPower: Power Estimation for Diverse FPGA Architectures," in *FPT 2012: Procedings of the 11th International Conference on Field-Programmable Technology; Seoul, South Korea*, December 2012, pp. 229–234.

[17] J. H. Anderson and C. Ravishankar, "FPGA Power Reduction by Guarded Evaluation," in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '10, 2010, pp. 157–166.

[18] S. J. E. Wilton, S. Ang, and W. Luk, "The Impact of Pipelining on Energy Per Operation in Field-Programmable Gate Arrays," in *In Field-Programmable Logic and Applications. Proceedings of the 13th International Workshop, FPL 2004*, 2004, pp. 719–728.

[19] E. Hung and S. J. E. Wilton, "Limitations of Incremental Signal-Tracing for FPGA Debug," in *FPL 2012, International Conference on Field Programmable Logic and Applications; Oslo, Norway*, August 2012, pp. 49–56.