TITAN: ENABLING LARGE AND COMPLEX BENCHMARKS IN ACADEMIC CAD

Kevin E. Murray, Scott Whitty, Suya Liu, Jason Luu, Vaughn Betz

Department of Electrical and Computer Engineering, University of Toronto, Ontario, Canada
email: {kmurray,jluu,vaughn}@eecg.utoronto.ca, {scott.whitty,suya.liu}@mail.utoronto.ca

ABSTRACT

Benchmarks play a key role in FPGA architecture and CAD research, enabling the quantitative comparison of tools and architectures. It is important that these benchmarks reflect modern designs which are large scale systems that make use of heterogeneous resources; however, most current FPGA benchmarks are both small and simple. In this paper we present Titan, a hybrid CAD flow that addresses these issues. The flow uses Altera's Quartus II FPGA CAD software to perform HDL synthesis and a conversion tool to translate the result into the academic BLIF format. Using this flow we created the Titan23 benchmark set, which consists of 23 large (90K-1.8M block) benchmark circuits covering a wide range of application domains. Using the Titan23 benchmarks and a detailed model of Altera's Stratix IV architecture we compared the performance and quality of VPR and Quartus II targeting the same architecture. We found that VPR is at least 2.7× slower, uses 5.1× more memory and 2.6× more wire compared to Quartus II. Finally, we identified that VPR's focus on achieving a dense packing is responsible for a large portion of the wire length gap.

1. INTRODUCTION

Open-source CAD flows, such as the VTR project [1], are crucial to FPGA research as open-source tools allow the FPGA architecture and CAD algorithms to be easily modified. To obtain accurate CAD or architecture results however, we need more than an open-source CAD flow. It is essential that the benchmark designs used to exercise a new algorithm or architecture represent the current, and ideally the future, usage of FPGAs. Unfortunately, the most commonly used FPGA benchmark suites are currently composed of designs that are both much smaller than the largest commercial FPGAs, and much simpler than current industrial designs. The MCNC20 benchmark suite [2], for example, has an average size of only 2960 blocks, while the latest commercial FPGAs [3, 4] contain up to *2 million* logic cells. Furthermore, half of the MCNC benchmarks are purely combinational, and none of the designs contain hard blocks such as memories or multipliers. The recently released VTR benchmark suite [1] is an improvement, but it still consists of designs with an average size of only 23,400 blocks, which would fill only 1% of the largest FPGAs. Only 10 of the 19 VTR designs contain any memory blocks and at most 10 memories are used in any design. In comparison, Stratix V and Virtex 7 devices contain up to 2,660 and 3,760 memory blocks respectively. Without larger benchmarks, key issues such as CAD tool scalability for very large designs cannot be investigated, and without more up-to-date benchmarks the validity of architecture studies is questionable.

There are many barriers to the use of state-of-the-art benchmark circuits with open-source tool flows. First, obtaining large benchmarks can be difficult, as many are proprietary. Second, purely open-source flows have limited HDL language coverage. The VTR flow, for example, uses the ODIN-II Verilog parser which can process only a subset of the Verilog HDL – any design containing System Verilog, VHDL or a range of unsupported Verilog constructs cannot be used without a substantial re-write. As well, if part of a design was created with a higher-level synthesis tool, the output HDL is not only likely to contain constructs unsupported by ODIN-II, but is also likely to be very hard to read and rewrite with only supported constructs. Third, modern designs make extensive use of IP cores, ranging from low-level functions such as floating-point multiply and accumulate units to higher-level functions like FFT cores and off-chip memory controllers. Since current open-source flows lack IP, all these functions must be removed or rewritten; this is not only a large effort, it also raises the question of whether the modified benchmark still accurately represents the original design, as IP cores are often a large portion of the design.

In order to avoid many of these pitfalls, we have created Titan, a hybrid flow that utilizes a commercial tool, Altera's Quartus II design software, for HDL elaboration and synthesis, followed by a format conversion tool to translate the results into a form open-source tools can process. The Titan flow has excellent language coverage, and can use any unencrypted IP that works in Altera's commercial CAD flow, making it much easier to handle large and complex benchmarks. We output the design early in the Quartus II flow, which means we can change the target FPGA architecture and use open-source synthesis, placement and routing engines to complete the design implementation. Consequently we believe we have achieved a good balance between enabling realistic designs, while still permitting a high degree of CAD and architecture experimentation. Our contributions include:

- Titan, a hybrid CAD flow that enables the use of larger and more complex benchmarks with academic CAD tools.
- The Titan23 benchmark suite. This suite of 23 designs has an average size of 421,000 blocks, and most designs are highly heterogeneous with thousands of RAM and/or multiplier blocks.
- A comparison of the quality and run time of the academic VPR placement and routing engine to the commercial Quartus II tool. This comparison helps identify how academic tool quality compares to commercial tools, and highlights several areas for potential improvement in VPR.

2. THE TITAN FLOW

The basic steps of the Titan flow are shown in **Fig. 1**. Quartus II performs elaboration and synthesis (`quartus_map`) which generates a Verilog Quartus Map (VQM) file. The VQM file is a technology mapped netlist, consisting of the basic primitives in the target architecture; see **Table 3** for primitives in the Stratix IV architecture. The VQM file is then converted to the standard Berkeley Logic Interchange Format (BLIF), which can be passed on to conventional open-source tools such as ABC and VPR [5, 6].

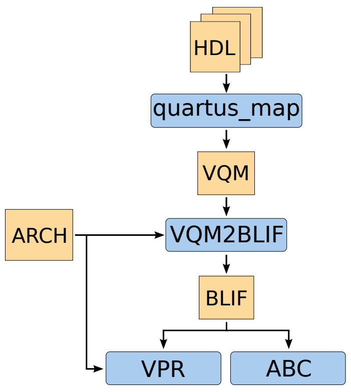


Fig. 1: The Titan Flow.

The conversion from VQM to BLIF is performed using our VQM2BLIF tool. At a high level, this tool performs a one-to-one mapping between VQM primitives and BLIF .subckt, .names, and .latch structures. To convert a VQM primitive to BLIF, the VQM2BLIF tool requires a description of the primitive's input and output pins. VPR also requires this information to parse the resulting BLIF; we store it in the architecture file for use by both tools.

VQM2BLIF can output different BLIF netlists to match a variety of use cases. Circuit primitives such as arithmetic, multipliers, RAM, Flip-Flops, and LUTs are usually modelled using BLIF's .subckt structure, which represents these primitives as black boxes. While this is usually sufficient for physical design tools like VPR, some primitives like LUTs and Flip-Flops can also be converted to the standard BLIF .names and .latch primitives respectively. This allows the circuit functionality to be understood by logic synthesis tools such as ABC. VQM2BLIF also supports more detailed conversions of VQM primitives, depending on their operation mode. This allows downstream tools, for instance, to differentiate between RAM blocks operating in single or dual port modes.

Some benchmarks make use of bidirectional pins, which cannot be modelled in BLIF. Therefore VQM2BLIF splits any bidirectional pins into separate input and output pins, and makes the appropriate changes to netlist connectivity.

It is also important to note that the sizes of benchmarks created with the Titan flow are not limited by the capacity of the targeted FPGA family. Quartus II's synthesis engine does not check whether the design will fit onto the target device, allowing VQM files to be generated for designs larger than any current commercial FPGA. The VQM2BLIF tool also runs quickly, taking less than 4 minutes to convert our largest benchmark.

The VQM2BLIF tool, detailed documentation, scripts to run the Titan flow and the complete benchmark set are available from: `http://www.eecg.toronto.edu/~vaughn/software.html`

3. FLOW COMPARISON

Using a commercial tool like Quartus II as a "front-end" brings several advantages that are hard to replicate in open-source flows. It supports several HDLs including Verilog, VHDL and SystemVerilog, and also supports higher level synthesis tools like Altera's QSYS, SOPC Builder, DSP Builder and OpenCL compiler. It also brings support for Altera's IP catalogue, with the exception of some encrypted IP blocks.

These factors significantly ease the process of creating large benchmark circuits for open-source CAD tools. For example, converting an LU factorization benchmark [7] for use in the VTR flow [1] involved roughly one month of work removing vendor IP and re-coding the floating point units to account for limited Verilog language support. Using the Titan flow, this task was completed within a day, as it only required the removal of one encrypted IP block from the original HDL which accounted for less than 1% of the design. In addition, since over 68% of the design logic was used by the floating point units, the Titan flow better preserves the original design characteristics.

Table 1: Comparison of architecture experiments supported by the VTR and Titan flows.

Experiment Modification	VTR	Titan	Titan Flow Method
Device Floorplan	Yes	Yes	Architecture file
Inter-cluster Routing	Yes	Yes	Architecture file
Clustered Block Size / Configuration	Yes	Yes	Architecture file
Intra-cluster Routing	Yes	Yes	Architecture file
Logic Element Structure	Yes	Yes	Architecture file
LUT size / Combinational Logic	Yes	Yes	ABC re-synthesis
New RAM Block	Yes	Yes	Architecture file (up to 16K depth)
New DSP Block	Yes	Yes	Architecture file (up to 36 bit width)
New Primitive Type	Yes	No	No method to pass black box through Quartus II

A concern in using a commercial tool to perform elaboration and synthesis is that the results may be too device or vendor specific to allow architecture experimentation. However this is not necessarily the case. The Titan flow still allows a wide range of experiments to be conducted as shown in **Table 1**. The ability to use tools like ABC to re-synthesize the netlist ensures experiments with different LUT sizes, and even totally different logic structures such as AICs [8], can still occur. RAM is represented as device independent "RAM slices" which are typically one bit wide, and up to 14 address bits deep. These RAM slices are packed into larger physical RAM blocks by VPR, and hence arbitrary RAM architectures can be investigated. Similarly, multiplier primitives (up to 36x36 bits) are packed into DSP blocks by VPR, allowing a variety of experiments. A simple remapping tool could also resize the multiplier primitives if desired. The structure of a logic element (connectivity, number of Flip-Flops, etc.) can also be modified without having to re-synthesize the design, and inter-block routing architecture and electrical design can both be arbitrarily modified.

Compared to VTR, the largest limitation is the inability to add support for new primitive types.

Another use of Titan is to test and evaluate CAD tool quality. Both post-technology mapping tools and logic re-synthesis tools can be plugged into the flow.

Titan provides a front-end interface between commercial and academic CAD flows which is complimentary to the back-end VPR to bitstream interface presented in [9].

Overall, the Titan flow enables a wide range of FPGA architecture experiments, and can be used to evaluate new CAD algorithms on realistic architectures with realistic benchmark circuits, and allows for more extensive scalability testing with larger benchmarks.

4. BENCHMARK SUITE

A wide range of benchmark designs were run through the Titan flow, with the goal of creating a set of large benchmarks representative of modern FPGA usage. Of the 46 benchmarks converted, the 23 largest from a diverse set of application domains were chosen to create the Titan23 benchmark suite described in **Section 4.2**. While the rest of this paper reports results primarily on the Titan23 benchmark suite, we are releasing the full set of 46 converted benchmarks. We believe that a range of benchmark sizes can be useful during the development stages of new FPGA CAD tools and architectures.

4.1. Benchmark Conversion Methodology

To convert a benchmark from HDL to BLIF, the design was first synthesized in Quartus II. For most designs this required no HDL modification, but some required replacing vendor/technology specific IP (e.g. PLLs, explicitly instantiated RAM blocks) with an equivalent Altera implementation, or working around obscure language features. Once the design was synthesized successfully, the resulting VQM file could be passed to VQM2BLIF.

In some cases, benchmark designs required more I/Os than were available on actual Stratix IV devices, preventing the designs from fitting in Quartus II. In these scenarios, the additional I/Os were connected to shift registers whose input/output was connected to a device pin. This is similar to the methodology described in [10].

Some IP blocks, such as the `sld_mux` in some of Altera's JTAG controllers and older DDR memory controllers are encrypted. These IP blocks were removed from the original HDL to avoid generating an encrypted VQM file. If possible, an equivalent unencrypted IP block was substituted. This was the case for some DDR controllers, since new Altera DDR controllers are not encrypted. Once encrypted IP was removed in the HDL, the design was re-synthesized and the new VQM file passed to VQM2BLIF. In general, only a small portion of the design logic had to be modified or removed.

4.2. Titan23 Benchmark Suite

The Titan23 benchmark suite consists of 23 designs ranging in size from 90K-1.8M blocks, with the smallest utilizing 40% of a Stratix IV EP4SGX180 device, and the largest designs unable to fit on the largest Stratix IV device. The designs represent a wide range of real world applications and are listed in **Table 2**. All benchmarks make use of some or all of the different heterogeneous blocks available on modern FPGAs, such as DSP and RAM blocks.

4.3. Comparison to Other Benchmark Suites

The characteristics outlined above make the Titan23 benchmark suite quite different from the popular MCNC20 benchmarks [2], which consist of primarily combinational circuits and make no use of heterogeneous blocks. Furthermore, the MCNC designs are extremely small. The largest (`clma`) uses less than 4% of a Stratix IV EP4SGX180 device, making it one to two orders of magnitude smaller than modern FPGAs.

Another benchmark suite of interest is the collection of 19 benchmarks included with the VTR design flow. These benchmarks are larger than the MCNC benchmarks, with the largest (`mcml`) reported to use 99.7K 6-LUTs [1]. Interestingly, when this circuit was run through the Titan flow, it uses only 11.7K Stratix IV ALUTs (6-LUTs) after synthesis, indicating the differences between ODINII+ABC and Quartus II's integrated synthesis. Additionally, only 10 of the VTR circuits make use of heterogeneous resources, and none use dedicated carry arithmetic. The Titan23 benchmark suite provides substantially larger benchmark circuits that make more extensive use of heterogeneous resources.

Several non-FPGA specific benchmark suites also exist. The various ISPD benchmarks [11] are commonly used to evaluate ASIC tools, but are only available in gate-level netlist formats. This makes them unsuitable for use as FPGA benchmarks, since they are not mapped to the appropriate FPGA primitives. The IWLS 2005 benchmarks [12] are available in HDL format, and the Titan flow enables them to be used with FPGA CAD tools. However, the largest design consists of only 36K blocks after running through the Titan flow – too small to be included in the Titan23.

5. STRATIX IV ARCHITECTURE CAPTURE

Recall that to use the Titan flow (without re-synthesis), the architecture file must use the VQM primitives as its fundamental building blocks. The architecture file can describe an architecture built out of these primitives, which can be combined into arbitrary complex blocks with arbitrary routing. We chose to align the architecture closely with Stratix IV. This allowed us to compare computational requirements and result quality between VPR and Quartus II, and identify possible areas for improvement.

To enable this comparison, a detailed VPR-compatible FPGA architecture description was created for Altera's Stratix IV family of 40nm FPGAs [13]. The Stratix IV device family was selected over the larger, more recent Stratix V family because of the architecture documentation available as part of Altera's QUIP [14]. As detailed below, this process also identified some limitations in VPR's architecture modelling capabilities. Some of the modelled Stratix IV primitives are shown in **Table 3**.

5.1. Floorplan

Stratix IV is an island style FPGA architecture, where the core of the chip is divided into rows and columns of blocks, and each column is built from a single type of block (LAB, DSP, etc.). The device aspect ratio and average spacing between blocks was determined by viewing an EP4SE820 device, the largest in the Stratix IV family, in the Quartus II floorplanner. An example floorplan is shown in **Fig. 2**.

5.2. Global (Inter-Block) Routing

The global or inter-block routing in Stratix IV uses wires 4 and 20 LABs long in the horizontal routing channels, and wires 4 and 12 LABs long in the vertical routing channels.

Table 2: Titan23 Benchmark Suite.

Name	Total Blocks	Clocks	ALUTs	REGs	DSP 18x18s	RAM Slices	RAM Bits	Application
gaussianblur	1,859,501	1	805,079	1,054,068	16	334	1,702	Image Processing
bitcoin_miner	1,061,829	2	455,263	546,597	0	59,968	297,664	SHA Hashing
directrf	934,482	2	471,194	447,032	960	40,029	20,307,968	Communications/DSP
sparcT1_chip2	814,799	98	368,381	430,976	24	14,355	1,585,435	Multi-core μP
LU_Network	630,212	58	194,376	399,782	896	41,647	9,390,528	Matrix Decomposition
LU230	567,993	2	208,997	293,177	924	64,664	10,112,704	Matrix Decomposition
mes_noc	548,047	9	273,323	248,988	0	25,728	399,872	On Chip Network
gsm_switch	487,454	5	154,996	296,681	0	35,776	6,254,592	Communication Switch
denoise	343,263	1	322,385	8,811	192	11,827	1,135,775	Image Processing
sparcT2_core	287,839	1	169,332	109,624	0	8,883	371,917	μP Core
cholesky_bdti	257,750	1	78,490	173,385	1,027	4,920	4,280,448	Matrix Decomposition
minres	252,600	2	108,117	126,105	614	17,608	8,933,267	Control Systems
stap_qrd	237,193	1	72,259	161,822	579	9,474	2,548,957	Radar Processing
openCV	212,616	1	108,094	86,460	740	16,993	9,412,305	Computer Vision
dart	202,414	1	103,844	87,386	0	11,184	955,072	On Chip Network Simulator
bitonic_mesh	192,648	1	110,617	49,570	676	31,616	1,078,272	Sorting
segmentation	174,072	1	161,723	6,561	104	5,658	3,166,997	Computer Vision
SLAM_spheric	124,648	1	113,660	6,874	296	3,744	11,488	Control Systems
des90	109,962	1	63,022	30,244	352	16,256	560,640	Multi μP system
cholesky_mc	108,239	1	29,264	74,051	452	5,123	4,444,096	Matrix Decomposition
stereo_vision	92,662	3	38,829	49,049	152	4,287	203,777	Image Processing
sparcT1_core	91,235	1	41,935	45,013	8	4,277	337,451	μP Core
neuron	90,779	1	24,760	61,477	565	3,799	638,825	Neural Network

Table 3: Important Stratix IV primitives.

Netlist Primitive	**Description**	**Model Quality**
`lcell_comb`	LUT and adder	Good
`dffeas`	Register	Good
`mlab_cell`	LAB LUTRAM	Good
`mac_mult`	Multiplier	Good
`mac_out`	Accumulator	Good
`ram_block`	RAM slice	Good
`io_{i,o}buf`	I/O Buffer	Moderate
`ddio_{in,out}`	DDR I/O	Moderate
`pll`	Phase Locked Loop	Poor

There are approximately 70% more horizontal wires than vertical wires. In Stratix IV the long wires are only accessible from the short wires and not from block pins. Additionally, Stratix IV allows LABs in adjacent columns to directly drive each other's inputs.

While VPR can model a mixture of long and short wires, it assumes the same configuration in both the horizontal and vertical routing channels. Additionally, VPR cannot model Stratix IV's short to long wire connectivity, or the direct-link interconnect between LABs in adjacent columns. As a result, the inter-block routing was modelled as length 4 and 16 wires (the average lengths), with both long and short wires accessible from logic block output pins. Unidirectional routing was used and the channel width was set to 300 wires.

5.3. Logic Array Block (LAB)

In Stratix IV, each LAB consists of 10 ALMs with 52 inputs from the global routing, and 20 feedback connections from the ALM outputs. Stratix IV uses a half-populated crossbar at the ALM inputs to select from the 72 possible input signals [15, 16]. The LAB has 40 outputs to global routing driven directly by the ALMs. Since no detailed information is available on the exact switch patterns used for the half-populated ALM input crossbars, they were modelled as shown in **Fig. 3**, otherwise the capture is accurate.

The Stratix IV LAB also includes three chain like structures (Carry Chain, Share Chain, Register Chain), however VPR does not currently support dedicated routing chain structures within logic blocks. Extra flexibility was added to allow each intermediate segment of a chain to drive a LAB output to ensure these structures were routable in VPR's packer.

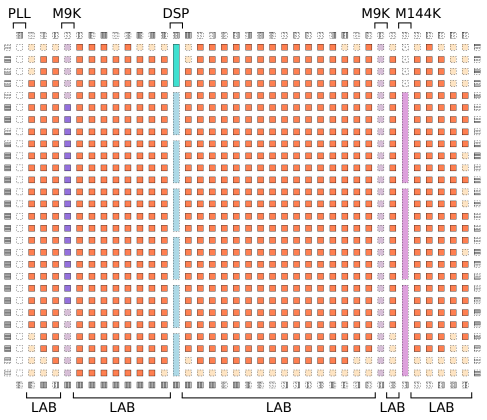


Fig. 2: Final placement of the `leon2` benchmark using the simplified architecture. Column block types are annotated, and I/Os are located around the perimeter.

Half of the LABs in a Stratix IV device can also be configured as small RAMs, referred to as Memory LABs (MLABs), which were also modelled. The F_{Cin} and F_{Cout} values were set to 0.055 and 0.100 respectively, to match the global routing connectivity in Stratix IV.

5.4. Adaptive Logic Module (ALM)

The ALM was modelled as two `lcell_comb` primitives, each representing a 6-LUT and full adder, along with two `dffeas` primitives representing flip-flops. The modelled ALM connectivity is shown in **Fig. 3**. The Stratix IV ALM contains 64-bits of LUT mask, less than what is required by two dedicated 6-LUTs. VPR cannot model this restriction and assumes two 64-bit LUT masks; however this extra flexibility is expected to have minimal impact on results, since few pairs of 6-LUTs can pack together in one ALM due to the limited number of inputs (8).

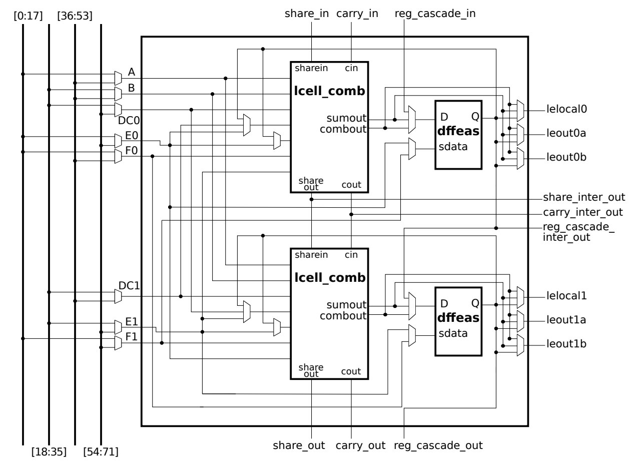


Fig. 3: Stratix IV ALM and half-populated input crossbar as captured in the detailed architecture model.

5.5. DSP Block

The Stratix IV DSP blocks are composed of eight `mac_mults` (18×18 multipliers) and two `mac_outs` (accumulator, rounding, etc.). In the Stratix IV architecture a `mac_out`'s inputs are only driven by `mac_mults`. However, similarly to the LAB/ALM chain structures, the `mac_out`'s inputs must also be accessible from the global routing to pack successfully in VPR.

5.6. RAM Block

Stratix IV supports two types of dedicated RAM blocks, the M9K and the M144K, each with different maximum depth and width limitations, and supporting ROM, Single Port, Dual Port and Bidirectional Dual Port operating modes. VPR supports non-mixed width RAMs using the *memory_class* directive, but does not provide native support for mixed-width RAMs, such as a rate conversion FIFO configured with a 1K×8 write port and 512×16 read port. To work around this limitation, mixed-width RAMs were modelled by elaborating all supported operating modes in the architecture file.

6. SIMPLIFIED ARCHITECTURE

As will be shown in **Section 7.3**, the detailed architecture capture described above performs very poorly in VPR's packer. As a result, a simplified architecture was created which makes some additional approximations. In the LAB, the half-populated crossbar used for the ALM inputs was replaced with a full crossbar. To avoid limitations related to placing LABs at MLAB locations, it is assumed that all LABs can be used as MLABs. For RAM blocks operating in mixed-width mode, the exact depth and width constraints were relaxed. While these relaxed constraints can potentially allow more RAM slices to pack into a RAM block than is architecturally possible, the RAM block will typically run out of pins before this occurs.

7. BENCHMARK RESULTS

In this section we use the Titan23 benchmark suite described in **Section 4**, in conjunction with the Stratix IV architecture capture described in **Section 5**. This allows us to compare the popular academic VPR tool with Altera's commercial Quartus II software. Using the Stratix IV architecture capture, VPR was able to target an architecture similar to the one targeted by Quartus II, allowing a coarse comparison of CAD tool quality.

7.1. Benchmarking Configuration

In all experiments, version 12.0 (no service packs) of Quartus II was used, while an early revision of VPR 7.0 (r1499) was used. The newer version of VPR was selected over VPR 6.0 since it provides substantial improvements to packing performance. During all experiments a hard limit of 48 hours run time was imposed; any designs exceeding this time were considered to have failed to fit. Most benchmarks were run on systems using Xeon E5540 (45nm, 2.56GHz) processors with either 16GB or 32GB of memory. For some benchmarks, systems using Xeon E7330 (65nm, 2.40GHz) and 128GB of memory were used; performance data collected on these machines is not directly comparable to the 16/32GB systems. For each benchmark, both tools were run on the same class of machines.

Since our Stratix IV architecture capture does not yet include timing information, both tools were run in a non-timing-driven mode. In VPR, this meant providing the `-timing_analysis off` command line option, while in Quartus II the `Optimize Timing` fitter option was set to `off`.

To ensure both tools were operating at comparable effort levels, VPR placement was also run with the `-fast` (inner_num = 1.0) option which reduces placement effort to a level similar to Quartus II's `STANDARD_FIT` mode. Quartus II supports multithreading, but was restricted to use a single thread to remain comparable with VPR.

Quartus II targets actual FPGA devices that are available only in discrete sizes. In contrast VPR allows the size of the FPGA to vary based on the design size. While it is possible to fix VPR's die size, we allowed it to vary, so that differences in block usage after packing would not prevent a circuit from fitting.

7.2. Quality of Results Metrics

Several key metrics were measured and used to evaluate the different tools. They fall into two broad categories.

The first category focuses on tool computational needs, which we quantify by looking at wall clock execution time for each major stage of the design flow (Packing, Placement, Routing), and also the peak memory consumption.

The second category of metrics focus on the Quality of Results (QoR). We measure the number of physical blocks generated by VPR's packer, and the total number of physical blocks used by Quartus II. Another key QoR metric is wire length (WL). Unlike VPR, Quartus II reports only the routed WL and does not provide an estimate of WL after placement. If a circuit fails to route in VPR, we estimate its required routed WL by scaling VPR's placement WL estimate by the average gap between placement estimated and final routed WL (~40%).

7.3. Detailed and Simplified Architecture Comparison

Initial attempts to use the detailed architecture capture (**Section 5**) resulted in most circuits taking over 48 hours to pack in VPR. This prompted the creation of a simplified architecture (**Section 6**). To quantify the impact of these modifications, VPR's relative performance and QoR on these two versions of the Stratix IV architecture was investigated.

Of the 46 total benchmarks converted, only the smallest benchmark with 7867 blocks (and none of the Titan23 benchmarks in **Table 2**) was able to pack, taking over 47 hours. This was over 400× slower than when using the simplified architecture. The design also used 46% more LABs and had 19% higher estimated WL after placement.

The key component of VPR's poor packing performance on the detailed architecture is the partially depleted crossbar feeding the ALM inputs, which leads to many failures of the packer's routability check.

7.4. Performance Comparison with Quartus II

Table 5 shows both the absolute run time and peak memory of VPR, and the relative values compared to Quartus II on the Titan23 benchmark suite, using the simplified architecture of **Section 6**. Quartus II's absolute run time and peak memory across the same benchmarks, while targeting Stratix IV, is shown in **Table 6**. VPR's run time is dominated by the packing step, which takes on average ~78% of the total run time on benchmarks that completed. In contrast, Quartus II has a more even run time distribution with placement taking the largest amount of time (49%), and with a significant amount of time (22%) spent on miscellaneous actions. For both tools, run time can be quite substantial on the larger benchmarks, taking up to 36.5 hours with Quartus II, and in excess of 48 hours with VPR.[1] It is also clear that Quartus II's run time and memory usage is more consistent than VPR's, generally scaling with the design size.

Looking at the relative run time and peak memory of the two tools in **Table 5**, we can draw some further insights. One of the most obvious is that VPR's packer is substantially slower (13.3×) than Quartus II's. Furthermore, the packer's run time is quite volatile, ranging from 4.0× slower in the best case to 34.0× slower in the worst case. A portion of this difference can be attributed to the increased flexibility of VPR's packer, which can target a broad range of architectures, while Quartus II's packer targets a finite set.

Quartus II spends a large portion of its run time during placement, and this is reflected when looking at the relative placement run time of the two tools. Here, VPR's placement engine is faster than Quartus II's, taking 49% less time. There could be several reasons behind this. VPR typically uses fewer LABs than Quartus II (see **Section 7.5**), which decreases the size of VPR's placement problem. Quartus II also enforces stricter placement legality constraints and uses more intelligent directed moves [17].

The tools also show a fairly large gap in routing run time, with VPR taking 3.4× longer than Quartus II. In six of the benchmarks, VPR's router was either unable to route the design or total time exceeded 48 hours. One caveat on these results is that VPR is routing high fan-out nets like clocks, while Quartus II is not, since it places these on dedicated clock networks.

As to overall run time, for benchmarks it successfully fit, VPR takes 2.7× longer that Quartus II. However, it should be noted that this result is skewed in VPR's favour, since it does not account for benchmarks which did not complete.

Peak memory consumption is also much higher (5.1×) in VPR. This is quite significant and will often limit the design sizes VPR can handle. It is interesting to note that the largest benchmark that Quartus II will fit (`bitcoin_miner`), uses approximately the same memory in Quartus II as the smallest Titan23 benchmark (`neuron`) uses in VPR.

[1]In contrast, the largest MCNC20 circuit took 61s in VPR and 96s in Quartus II, highlighting the importance of using large benchmarks to evaluate CAD tools.

7.5. Quality of Results Comparison with Quartus II

The relative QoR results for the Titan23 benchmark suite are shown in **Table 4**. These results show several trends. First, VPR uses fewer LABs (0.8×) than Quartus II. While this reduced LAB utilization may initially seem a benefit (since a smaller FPGA could be used), this comes at the cost of WL as will be discussed in **Section 7.6**.

Table 4: VPR 7/Quartus II Quality of Result Ratios.

Name	LAB	DSP	M9K	M144K	WL	Note
gaussianblur						
bitcoin_miner						
directrf						
sparcT1_chip2	1.01	3.00	0.88		1.17	*
LU_Network	1.39	3.00	1.26		3.67	*
LU230	0.53	2.99	3.54	23.75	4.61	*
mes_noc	0.81		0.91		2.18	*
gsm_switch	0.66		1.02		1.83	
denoise	0.73	2.12	2.65		1.56	
sparcT2_core	0.92		0.91		1.68	*
cholesky_bdti	1.08	2.98	1.00		4.47	*
minres	0.64	2.74	1.28		3.71	
stap_qrd	1.74	2.01	1.00		3.27	
openCV	0.81	2.77	1.17	0.83	4.42	
dart	0.76		0.93		2.57	
bitonic_mesh	0.71	2.13	1.03		2.54	
segmentation	0.75	2.22	1.54	4.50	1.58	
SLAM_spheric	0.63	1.95			1.82	
des90	0.73	1.52	1.00		3.07	
cholesky_mc	0.82	1.98	1.19	0.80	3.60	
stereo_vision	0.73	4.00	1.11		3.05	
sparcT1_core	0.89	1.00	0.93		1.40	
neuron	0.67	1.77	1.65		6.17	
Geomean	0.82	2.27	1.21	2.91	2.64	

* WL scaled from placement estimate.

Looking at the other block types, VPR uses 2.3× as many DSP blocks and 1.2× as many M9K blocks as Quartus II, showing that Quartus II is better able to utilize these hard block resources. Since only four circuits use M144K blocks in both tools, it is difficult to draw meaningful conclusions.

Routed WL is our best metric for comparing the overall quality of the VPR and Quartus II physical design tools. Somewhat surprisingly, the wire length gap is quite large, with VPR using 2.6× more wire than Quartus II.[2] Without access to Quartus II's internal packing and placement statistics, it is difficult to identify which step(s) of the design flow are responsible for this difference. However, in **Table 4** it appears that circuits where VPR uses substantially fewer LABs than Quartus II often have the largest difference in WL.

7.6. Modified Quartus II Comparison

To further investigate this correlation between packing density and WL, we re-ran the benchmarks through Quartus II using several different combinations of packing and placement settings. The impact of these settings on the relative QoR between VPR and Quartus II are shown in **Table 7**.

We investigated the effect of telling Quartus II to always pack densely, and the effect of disabling placement finalization. In default mode Quartus II varies packing density based on the expected utilization of the targeted FPGA, spreading out the design if there is sufficient space. Also by default, Quartus II performs placement finalization, where it breaks apart clusters by moving individual LUTs and Flip-Flops.

[2]The WL gap is quite different (0.6×) on the largest MCNC20 circuit, emphasizing how modern benchmarks can greatly impact CAD tool QoR.

Table 5: VPR 7 run time in minutes and memory in GB. Relative speed to Quartus II (VPR/Q2) is shown in parentheses.

Name	Total Blocks	Pack		Place		Route		Total		Mem.		Note
gaussianblur	1,859,501											*†
bitcoin_miner	1,061,829											*†
directrf	934,482											*†
sparcT1_chip2	814,799	1,940.5	(3.96×)	159.9	(0.37×)					31.0	(3.54×)	*†
LU_Network	630,212	1,083.4	(7.48×)	137.2	(0.70×)							*†
LU230	567,993	1,557.5	(21.25×)	98.1	(0.56×)					81.0	(10.99×)	*†
mes_noc	548,047	474.6	(23.13×)	50.3	(0.80×)					26.0	(4.29×)	‡
gsm_switch	487,454	468.1	(18.29×)	57.7	(0.52×)	54.1	(3.65×)	579.9	(3.17×)			*
denoise	343,263	72.6	(6.14×)	30.6	(0.68×)	7.1	(1.07×)	110.3	(1.43×)	17.0	(4.03×)	
sparcT2_core	287,839	108.6	(10.29×)	17.2	(0.47×)					12.0	(3.46×)	‡
cholesky_bdti	257,750	199.6	(18.59×)	12.6	(0.50×)					15.0	(4.73×)	‡
minres	252,600	197.9	(18.27×)	23.3	(0.48×)	26.8	(5.11×)	248.0	(3.09×)	27.0	(7.62×)	*
stap_qrd	237,193	250.3	(11.63×)	34.2	(0.85×)	22.6	(4.90×)	307.1	(3.89×)	18.0	(5.94×)	*
openCV	212,616	116.3	(21.67×)	8.2	(0.43×)	17.8	(5.20×)	142.2	(4.00×)	23.0	(8.05×)	
dart	202,414	65.7	(9.50×)	6.6	(0.38×)	6.6	(1.79×)	78.9	(1.96×)	11.0	(4.15×)	
bitonic_mesh	192,648	214.0	(33.97×)	25.7	(0.82×)	33.2	(5.28×)	272.8	(4.46×)	30.0	(9.01×)	*
segmentation	174,072	40.8	(8.36×)	9.1	(0.51×)	3.7	(1.07×)	53.6	(1.62×)	12.0	(4.84×)	
SLAM_spheric	124,648	37.7	(11.03×)	5.9	(0.36×)	13.0	(4.26×)	56.6	(1.96×)	7.9	(3.61×)	
des90	109,962	63.9	(28.62×)	4.8	(0.45×)	7.4	(3.90×)	76.2	(3.70×)	15.0	(6.73×)	
cholesky_mc	108,239	28.8	(11.53×)	3.8	(0.49×)	5.9	(5.09×)	38.6	(2.50×)	11.0	(5.66×)	
stereo_vision	92,662	19.2	(8.85×)	1.9	(0.29×)	4.3	(6.22×)	25.4	(2.05×)	6.5	(4.09×)	
sparcT1_core	91,235	22.5	(9.69×)	2.6	(0.52×)	2.2	(0.97×)	27.2	(2.11×)	4.7	(2.79×)	
neuron	90,779	47.9	(26.38×)	2.5	(0.40×)	9.5	(9.95×)	59.9	(4.80×)	9.5	(5.44×)	
Geomean		137.0	(13.34×)	15.4	(0.51×)	10.4	(3.39×)	95.8	(2.71×)	15.7	(5.14×)	

* Run on 128GB machine. †Exceeded 48 hours run time. ‡Un-routable.

Table 6: Quartus II run time in minutes and memory in GB.

Name	Total Blocks	Pack	Place	Route	Misc.	Total	Mem.	Note
gaussianblur	1,859,501							*§
bitcoin_miner	1,061,829	381.3	981.3	180.8	647.4	2,190.8	9.1	*
directrf	934,482							*§
sparcT1_chip2	814,799	490.5	434.9	229.3	446.6	1,601.2	8.8	*
LU_Network	630,212	144.9	196.4	15.8	41.6	398.6	6.8	*
LU230	567,993	73.3	174.2	34.2	146.0	427.7	7.4	*
mes_noc	548,047	20.5	63.2	11.4	39.1	134.2	6.1	
gsm_switch	487,454	25.6	111.8	14.8	30.5	182.7	5.5	*
denoise	343,263	11.8	44.8	6.6	13.8	77.0	4.2	
sparcT2_core	287,839	10.6	36.7	10.2	10.2	67.7	3.5	
cholesky_bdti	257,750	10.7	25.0	4.0	8.2	47.9	3.2	
minres	252,600	10.8	48.1	5.3	16.0	80.2	3.5	*
stap_qrd	237,193	21.5	40.0	4.6	12.9	79.0	3.0	*
openCV	212,616	5.4	18.8	3.4	7.9	35.6	2.9	
dart	202,414	6.9	17.5	3.7	12.3	40.4	2.7	
bitonic_mesh	192,648	6.3	31.3	6.3	17.3	61.2	3.3	*
segmentation	174,072	4.9	17.8	3.4	7.1	33.2	2.5	
SLAM_spheric	124,648	3.4	16.5	3.1	6.0	29.0	2.2	
des90	109,962	2.2	10.8	1.9	5.7	20.6	2.2	
cholesky_mc	108,239	2.5	7.7	1.2	4.1	15.4	1.9	
stereo_vision	92,662	2.2	6.7	0.7	2.9	12.4	1.6	
sparcT1_core	91,235	2.3	5.0	2.3	3.4	12.9	1.7	
neuron	90,779	1.8	6.2	1.0	3.5	12.5	1.8	
Geomean		12.2	35.8	6.3	16.3	72.7	3.4	

* Run on 128GB machine. †Exceeded 48 hours run time. ‡Un-routable. §Exceeded size of largest Stratix IV.

Table 7: QoR ratios for different Quartus II packing density and placement finalization settings.

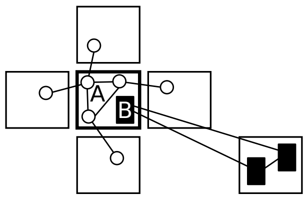
Q2 Settings	Q2:Q2 Def. LAB	Q2:Q2 Def. WL	VPR:Q2 LAB	VPR:Q2 WL
Default	1.00	1.00	0.82	2.46
No Final.	1.03	1.13	0.79	2.18
Dense	0.88	1.37	0.92	1.80
Dense No Final.	0.78	1.76	1.05	1.40

Note: the default VPR:Q2 WL is different from **Table 4** because some benchmarks would not fit for some Quartus II settings combinations.

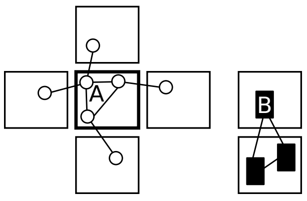
Disabling placement finalization resulted in a moderate increase in Quartus II's WL. Forcing Quartus II to pack densely significantly reduced the number of LABs used. This resulted in a large increase in Quartus II's WL, narrowing the WL gap between VPR and Quartus II. Simultaneously disabling finalization and forcing dense packing further reduced the number of LABs used, and further increased Quartus II's WL. With these settings the WL gap between VPR and Quartus II reduced to 1.4× from the original 2.5×.

This indicates that a significant portion of VPR's higher WL is likely due to packing effects, and principally due to a focus on achieving high packing density. We suspect that VPR's packer is sometimes packing largely unrelated logic together to minimize the number of clusters. This appears to be counter productive from a WL perspective.

For example, consider a LAB that is mostly filled with related logic A, but which can accommodate an extra unrelated register B. During placement, the cost of moving this LAB will be dominated by the connectivity to the related logic A. This could result in a final position that is good for A but may be very poor for the extra register B (i.e. far from its related logic), as shown in **Fig. 4a**. If this is a common occurrence it could lead to increased WL.



(a) Dense Packing



(b) Less Dense Packing

Fig. 4: Packing density and wire length example.

A better solution to the above scenario would have been to utilize additional clusters (pack less densely) to avoid packing unrelated logic together, as shown in **Fig. 4b**. Alternately, if the placement engine was able to recognize the competing connectivity requirements inside a cluster, it could break it apart, much like Quartus II's placement finalization.

8. CONCLUSION AND FUTURE WORK

First, we have presented Titan, a new hybrid flow that enables the creation of large benchmark circuits for use in academic CAD tools, supporting a wide variety of HDLs and a wide range of IP blocks. Second, we have presented the Titan23 benchmark suite built using the Titan flow. Titan23 significantly improves the state of open-source FPGA benchmarks by providing benchmarks across a wide range of application domains, which are much closer in both size and design style to modern FPGA usage. Third, we have presented a reasonable architecture capture of Altera's Stratix IV family, a modern high performance FPGA architecture. Finally, we have used this benchmark suite and architecture capture to compare the popular academic CAD tool VPR with a state-of-the-art commercial CAD tool, Altera's Quartus II. The results show that VPR is at least 2.7× slower, consumes 5.1× more memory and uses 2.6× more wire than Quartus II. Additional investigation identified VPR's focus on achieving high packing density to be an important factor in the WL difference.

The most obvious limitation of the current comparison between VPR and Quartus II is that both tools were run without timing optimization. In the future, we plan to add timing information to the Stratix IV architecture capture and evaluate the quality of these tools under real timing constraints.

The Titan23 benchmark suite represents a first step forward, but will need to be continually updated to keep pace with increasing FPGA design size and complexity. Therefore we would welcome additional benchmark contributions to cover larger design sizes and a wider range of applications.

Finally, given the substantial gap between VPR and commercial FPGA CAD tools, it is clear that there remains significant room for improvement in the run time, memory usage, and result quality of this academic CAD tool.

9. ACKNOWLEDGEMENTS

This work was funded by NSERC, Altera, Texas Instruments and the SRC. Computations were performed on the GPC supercomputer at the SciNet HPC Consortium.

We are grateful to the following organizations and individuals for providing benchmark circuits: Altera, Braiden Brousseau, Deming Chen, Jason Cong, George Constinides, Zefu Dai, Joseph Garvey, IWLS2005 [12], Mark Jervis, LegUP [18], Simon Moore, OpenCores.org, OpenSparc.net, Kalin Ovtcharov, Alex Rodionov, Russ Tessier, Danyao Wang, Wei Zhang, and Jianwen Zhu. We also thank David Lewis, Jonathan Rose and Tim Liu for useful discussions.

REFERENCES

[1] J. Rose, *et al.*, "The VTR project: Architecture and CAD for FPGAs from verilog to routing," in *FPGA*, 2012, pp. 77–86.

[2] S. Yang, "Logic synthesis and optimization benchmarks user guide 3.0," MCNC, Tech. Rep., 1991.

[3] *Stratix V Device Overview*, http://www.altera.com, Altera Corporation, December 2012.

[4] *7 Series FPGAs Overview*, http://www.xilinx.com, Xilinx Incorporated, November 2012.

[5] "ABC: A system for sequential synthesis and verification," http://www.eecs.berkeley.edu/~alanmi/abc/, Berkeley Logic Synthesis and Verification Group, 2013.

[6] V. Betz *et al.*, "VPR: A new packing, placement and routing tool for FPGA research," in *FPL*, 1997, pp. 213–222.

[7] W. Zhang, *et al.*, "Portable and scalable FPGA-based acceleration of a direct linear system solver," *ACM TRETS*, vol. 5, no. 1, pp. 6:1–6:26, Mar. 2012.

[8] H. Parandeh-Afshar, *et al.*, "Rethinking FPGAs: elude the flexibility excess of LUTs with and-inverter cones," in *FPGA*, 2012, pp. 119–128.

[9] E. Hung, *et al.*, "Escaping the academic sandbox: Realizing VPR circuits on xilinx devices," in *FCCM*, 2013.

[10] *OpenCore Stamping and Benchmarking Methodology*, http://www.altera.com, Altera Corporation, May 2008.

[11] N. Viswanathan, *et al.*, "The ISPD-2011 routability-driven placement contest and benchmark suite," in *ISPD*.

[12] *2005 Benchmarks*, http://iwls.org/iwls2005/benchmarks.html, IWLS, 2005.

[13] *Stratix IV Device Handbook*, http://www.altera.com, Altera Corporation, September 2012.

[14] *Quartus II University Interface Program*, http://www.altera.com, Altera Corporation, Febuary 2009.

[15] D. Lewis, *et al.*, "Architectural enhancements in Stratix-III and Stratix-IV," in *FPGA*, 2009, pp. 33–42.

[16] D. Lewis *et al.*, "The Stratix II logic and routing architecture," in *FPGA*, 2005, pp. 14–20.

[17] A. Ludwin *et al.*, "Efficient and deterministic parallel placement for FPGAs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 16, no. 3, pp. 22:1–22:23, June 2011.

[18] A. Canis, *et al.*, "LegUp: High-level synthesis for FPGA-based processor/accelerator systems," in *FPGA*, 2011, pp. 33–36.