VTR 7.0: Next Generation Architecture and CAD System for FPGAs

JASON LUU, University of Toronto
JEFFREY GOEDERS, University of British Columbia
MICHAEL WAINBERG, University of Toronto
ANDREW SOMERVILLE, University of New Brunswick
THIEN YU, University of Toronto
KONSTANTIN NASARTSCHUK, University of New Brunswick
MIAD NASR, University of Toronto
SEN WANG, University of New Brunswick
TIM LIU and NOORUDDIN AHMED, University of Toronto
KENNETH B. KENT, University of New Brunswick
JASON ANDERSON, JONATHAN ROSE, and VAUGHN BETZ, University of Toronto

Exploring architectures for large, modern FPGAs requires sophisticated software that can model and target hypothetical devices. Furthermore, research into new CAD algorithms often requires a complete and open source baseline CAD flow. This article describes recent advances in the open source Verilog-to-Routing (VTR) CAD flow that enable further research in these areas. VTR now supports designs with multiple clocks in both timing analysis and optimization. Hard adder/carry logic can be included in an architecture in various ways and significantly improves the performance of arithmetic circuits. The flow now models energy consumption, an increasingly important concern. The speed and quality of the packing algorithms have been significantly improved. VTR can now generate a netlist of the final post-routed circuit which enables detailed simulation of a design for a variety of purposes. We also release new FPGA architecture files and models that are much closer to modern commercial architectures, enabling more realistic experiments. Finally, we show that while this version of VTR supports new and complex features, it has a 1.5× compile time speed-up for simple architectures and a 6× speed-up for complex architectures compared to the previous release, with no degradation to timing or wire-length quality.

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids—*Automatic synthesis*; B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and routing*; C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Adaptable architectures*

General Terms: Algorithms, Design

Additional Key Words and Phrases: FPGA, CAD, architecture modeling

ACM Reference Format:
Jason Luu, Jeffrey Goeders, Michael Wainberg, Andrew Somerville, Thien Yu, Konstantin Nasartschuk, Miad Nasr, Sen Wang, Tim Liu, Nooruddin Ahmed, Kenneth B. Kent, Jason Anderson, Jonathan Rose, and Vaughn Betz. 2014. VTR 7.0: Next generation architecture and CAD system for FPGAs. ACM Trans. Reconfig. Technol. Syst. 7, 2, Article 6 (June 2014), 30 pages.
DOI: http://dx.doi.org/10.1145/2617593

The authors gratefully acknowledge the funding support of NSERC, SRC, Altera, and Texas Instruments.
Author's address: V. Betz; email: vaughn@eecg.toronto.edu.

DOI: http://dx.doi.org/10.1145/2617593

1. INTRODUCTION AND CONTEXT

The increasing size and complexity of modern field-programmable gate arrays (FPGAs) continues to grow in line with Moore's law, and state-of-the-art devices now incorporate billions of transistors and myriad features. Programmable logic devices may serve as all or part of the platform for future systems on a single chip [Xilinx 2013b; Altera 2012b; Lattice 2012; Microsemi 2013]. This exponential growth and feature evolution brings with it deep questions on the architecture of device features, and gives rise to challenges across the broad spectrum of CAD tools required to optimize designs for the devices. Some questions can be explored with the commercial CAD tools that support specific devices [Altera 2013; Xilinx 2013a], but exploration of new architectural features beyond simple variants of commercial architectures and the investigation of enhanced algorithms for various stages of the CAD flow require open-source tools that can be modified by researchers.

Several at least partially open-source FPGA CAD flows exist. The RapidSmith tools [Lavin et al. 2011] allow interaction with Xilinx's commercial FPGA architectures and CAD tools by providing a suite of design and chip database interaction functions and visualization utilities; RapidSmith has been used to interface a hard macro placer and a router with the Xilinx tool flow. The Torc project [Steiner et al. 2011] utilitizes different APIs, but provides a similar ability to interact with and augment portions of Xilinx's commercial CAD flow. The VTR CAD flow we enhance in this work has a different design goal than Torc or RapidSmith: instead of augmenting the tool flow for a commercial FPGA, it allows the investigation of entirely new FPGA architectures by taking a concise, human-readable architecture description as one of its inputs. The fact that the entire flow is open-source also enables exploration of new CAD ideas that require modifications to multiple stages of the CAD flow. Independence [Sharma et al. 2005] is an open-source placement and routing engine for FPGAs; like VTR it can be targeted at different FPGA architectures. Independence is able to produce high-quality placements for an even wider variety of FPGA routing architectures than VTR, but does so at a very high CPU cost that limits its scalability to large designs.

The complexity and capability of FPGA software for commercial devices has grown along with the devices, but the open-source VPR/VTR software [Betz et al. 1999; Luu et al. 2009; Rose et al. 2012] has struggled to keep pace, and indeed has fallen behind on many fronts. If the open-source software cannot support known-good architectural features, or achieve quality of results commensurate with that of commercial flows, then the research results derived from the software could be significantly misleading. The purpose of this article is to describe and measure a major new version of the open-source VTR FPGA CAD flow that contains a series of important new features.

One of the key values to our research community is that the features described in this article are integrated into a single, working and tested version of software. While some of the new features may have been implemented in isolation in prior versions of the software, it is significantly more work to ensure that they operate alongside *other* new features so that the total sum can be used to make progress in the field. This level of effort is onerous: it has been noted in other branches of computer science and engineering that "Computer Science is now Big Science," reflecting the idea that it is difficult to make progress without pursuing a large-scale many-person effort. This is becoming true in our sub-discipline of FPGAs. And so, this article reports on the most recent advances within the large-scale effort to provide the community with a robust open-source framework that we believe will enable a variety of new research on FPGA CAD and architecture.

Our major contributions are enhancements to the VTR flow that will enable FPGA research and several new algorithmic ideas and results as detailed here.

(1) We provide an integrated and open-source tool flow with many new features which work together to enable a wider variety of CAD and architecture experiments. For example by adding CAD support for multi-clock circuits and high-speed arithmetic, we enable modern circuits to be run through the flow and yield realistic results. Prior versions of VTR handled such circuits poorly: the clocks in multiclock devices were shorted into a single clock, and the arithmetic was implemented using many levels of soft logic (i.e. look-up-tables). This could have led to inaccurate architecture conclusions.
(2) We include architecture files that describe complex, realistic FPGA architectures. As FPGAs evolve, the accurate capture and description of all of the capabilities and features of the FPGA is both more difficult and more important. Building on the work of [Luu et al. 2011], we enhance the description capability of the framework, and describe a series of useful and complete experimental architectures. These descriptions are a form of software themselves.
(3) We provide benchmark circuits that work with these tools and architectures. Both architecture and CAD research need a representative set of example circuits from which to draw their conclusions. Given the complexity of the architectures and toolflow, it is important to cultivate and release a set of relevant applications that are guaranteed to function with each software and architecture release.
(4) We detail a technique for the robust optimization of designs with multiple timing constraints, even when some constraints are impossible to meet.
(5) We present a new packing algorithm for logic blocks with complex internal interconnect which is 25× faster than prior work, while achieving higher result quality.
(6) We quantify the speed gain and routability impact of adding hard adders to an FPGA architecture.

This article is organized as follows. After describing the overall flow of the tools, we describe the new capabilities discussed earlier, and present measurements illustrating the utility of each feature. We then describe the new clustering approach and architecture files. We provide a measurement of the quality of results of the new release, compared to earlier versions of the software. This is important to be able to normalize the relative capabilities of the different versions and to ensure that adding features has not degraded the basic algorithms. The final sections conclude and point to future work on this project.

2. OVERALL FLOW

The overall VTR CAD flow is depicted in Figure 1. There are two inputs to the flow: the user-designed Verilog application circuit, and an FPGA architecture description file [Luu et al. 2011] which is a human-readable (and writable!) description of the physical logic, routing and I/O of a hypothetical FPGA. ODIN [Jamieson et al. 2010] elaborates the Verilog, translating and partially synthesizing the Verilog code into a netlist of primitives. This synthesis includes the division of a design's larger memories and multipliers into pieces that can be accommodated by the memories or multipliers available on the device, as described by the architecture file [Rose et al. 2012]. The soft logic in this netlist is then optimized by the ABC tool [Mishchenko et al. 2009], a slightly modified version of release 70930, and technology mapped into the appropriate-size lookup tables (LUTs) and flip-flops. VPR performs all physical optimization—clustering, placement and routing—of the logic primitives (LUTs, flip-flops, small multiplies, etc.) into the complex logic and other blocks described in the architecture file. VPR also performs all the analysis steps that measure result quality: timing analysis, area estimation and power estimation.

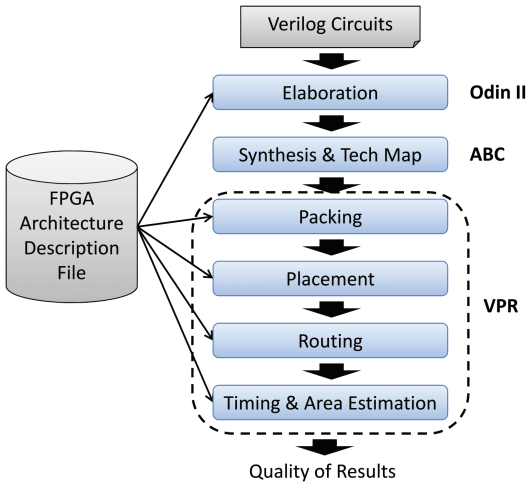


Fig. 1. Overall CAD flow.

The contributions of this work are in the ODIN-II and VPR stages of this flow. These stages directly read and adapt to the architecture file, so we will first describe the new architecture files released with VTR version 7.0, along with the benchmark circuits we use to measure the flow quality.

3. ARCHITECTURES AND FILES

As FPGA architectures have become increasingly complex, so too have their architectural descriptions, and hence the creation of a complete and realistic FPGA architecture can be difficult. To lower this barrier to FPGA research, we have created a set of carefully designed architecture description files to serve as templates. These architectures can be used unmodified as VTR targets for researchers investigating new CAD algorithms and can also be used as a baseline within which a new architecture change can be inserted and investigated by FPGA architects. In the following section we describe the various architecture files provided in the VTR 7.0 release, along with their recommended usage and the benchmark circuits which can be targeted to each architecture. See Section 11 for download instructions for the entire VTR 7.0 release, including these architecture and benchmark files and a manual that describes the architecture file syntax in detail.

3.1. Comprehensive Architecture File

The *Comprehensive Architecture* is the flagship architecture of the VTR release, and we suggest that new users of VTR begin with it. It describes an FPGA architecture with a number of modern features, including fracturable LUTs, carry-chains, fracturable multipliers, and configurable memories. Its architecture features are chosen to be in line with both the recommendations of prior research and current commercial practice in Xilinx's Virtex 7 and Altera's Stratix IV architectures. The routing architecture uses length-4 wire segments, single-driver routing, an Fc_{in} of 0.15, an Fc_{out} of 0.1, and an Fs of 3.

Figure 2 depicts the soft logic block which has 40 general input pins, 20 general output pins, one carry-in pin, one carry-out pin, and one clock input pin. In this architecture the general logic block inputs feed a full crossbar that allows them to drive the inputs of any Fracturable Logic Element (FLE). The crossbar also allows any FLE output to drive any FLE input. Each FLE consists of one fracturable LUT with two optionally

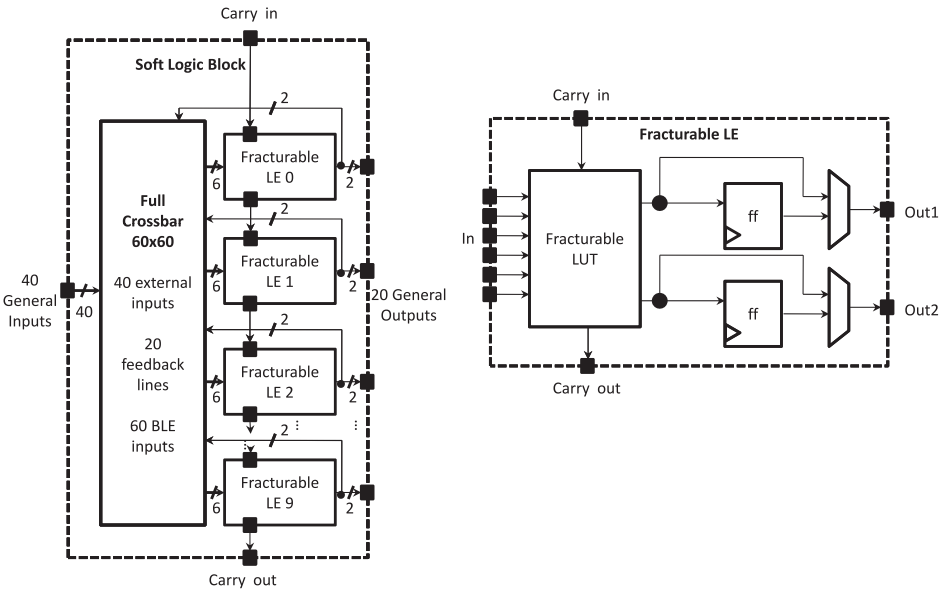


Fig. 2. Soft Logic Block in the Comprehensive Architecture.

registered outputs. As shown in Figure 3, each fracturable LUT can operate as one 6-LUT or as two 5-LUTs with all 5 inputs shared, in a manner similar to Xilinx's Virtex 7 architecture [Xilinx 2013b].

Each 5-LUT in this architecture can optionally operate in an arithmetic mode, as shown in Figure 4. As described later in Section 5, this release adds new support for FPGA architectures with carry chains throughout the CAD flow. In this arithmetic mode, each 5-LUT operates as two 4-LUTs that drive two bits of a hardened full adder. To avoid requiring additional input-selection multiplexers (which are costly), the two 4-LUTs share all their inputs. The sum output of the full adder is multiplexed with the output of the original 5-LUT, so either (but not both) of these signals can be the 5-LUT output in Figure 2. The carry in and carry out pins of the full adder are permanently connected (hardwired) to adjacent adders, such that all 20 adders in a logic block form a single chain. To implement adders larger than 20 bits, the carry chain connects the carry out of the last adder in a logic block to the carry in of the first adder in the logic block immediately below it using a dedicated inter-block routing path.

The multiplier logic block in the Comprehensive Architecture can operate as one 36×36 multiplier, or be fractured into two independent 18×18 multipliers. Additionally, each 18×18 multiplier can be further subdivided into two independent 9×9 multipliers. The height of the multiplier blocks is four times the height of a soft logic block, and hence it can access four horizontal routing channels. Overall, this multiplier architecture is similar to that of Altera's Stratix IV devices [Altera 2009]. Every eighth column of the Comprehensive FPGA consists of a column of multiplier blocks.

This Comprehensive Architecture contains a configurable 32 kilobit RAM block, which is also four times the height of a soft logic block and is instantiated in every eighth column. The block has a configurable aspect ratio ranging from 32K words deep x 1 bit wide (32Kx1) to 1Kx32, for both single and dual-port modes. In single-port mode, the block can additionally be configured as a 512x64 memory. The I/Os of this architecture are on the perimeter, with each I/O block holding 8 I/O pins, and each I/O pin can be configured to be either an input or an output.

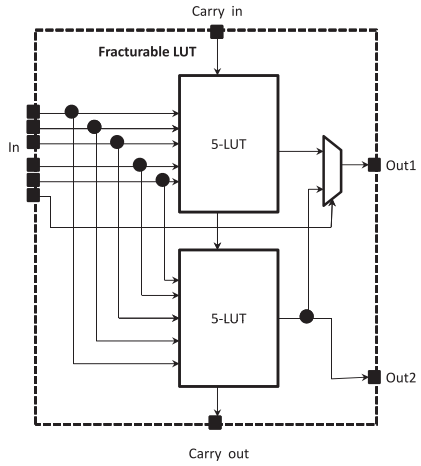


Fig. 3. Fracturable LUT in the Comprehensive Architecture.

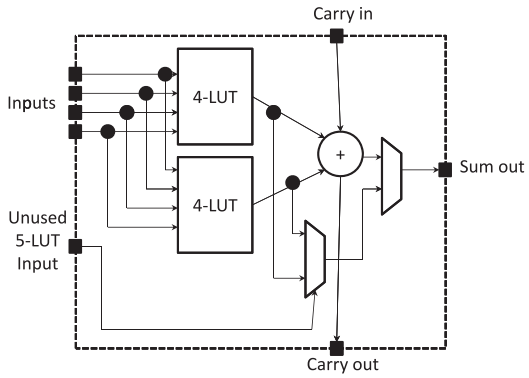


Fig. 4. Arithmetic Mode of Comprehensive Architecture: 5-LUT split into two 4-LUTs and Hard Adder.

The models that allow the toolset to compute the area, delay and energy consumption of any circuit implemented on the FPGA are a key part of each architcture file. Ideally, we would create area, delay and energy models of each block and routing structure from a detailed transistor-level design in a consistent process technology. However, the effort required to create such a transistor-level design for each structure on an FPGA is extremely high. We adopted an alternative approach, where we use models based on detailed transistor-level designs in a 45 *nm* Predictive Technology Model (PTM) [Cao 2008] process for key structures such as the FPGA routing and use data from commercial chips for other parts of the FPGA. All area, delay and power results are scaled to a 40/45 *nm* process operating at 0.9 V so they are self-consisent as described here.

—*Area.* The routing area is based on transistor sizings for a 45 *nm* PTM process FPGA from the iFAR repository [Kuon and Rose 2008]. The area of logic, RAM and multiplier blocks are taken from published Stratix III values [Wong et al. 2011], scaled to 40 *nm*, and adjusted to account for architecture differences (e.g. our use of 32 Kb RAM blocks instead of 9 Kb).

—*Delay.* All delays are taken from comparable resources in the fastest speed grade of Stratix IV, which is a 40 *nm* FPGA. The resistance and capacitance of the general

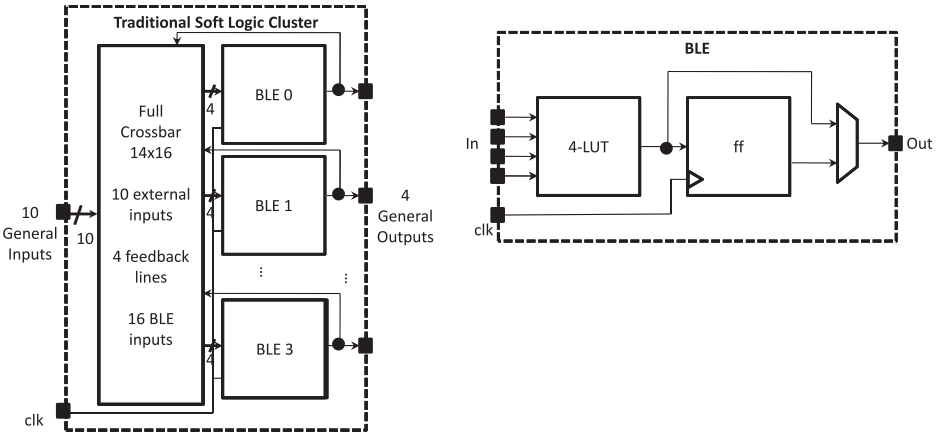


Fig. 5. Classical Architecture Soft Logic Block.

routing wires are based on the intermediate level metal from the International Technology Roadmap for Semiconductors [ITRS 2011] at 45 *nm*.

—*Energy.* The static and dynamic power of the logic block and general routing wires come from a detailed transistor-level design in a 45 *nm* PTM process at 0.9 V. The dynamic power for other blocks (e.g. multiplier blocks) is taken from the 0.9 V, 40 *nm* Stratix IV FPGA (adjusted for architecture differences where necessary), and the static power for these blocks is set to zero.

3.2. Classical Architecture File

While the Comprehensive Architecture described here represents a much closer modeling of modern FPGAs than has been typically used in the academic literature, we recognize that some types of research benefit from dealing with much simpler FPGAs – in particular those with only LUTs, flip-flops and I/Os. As such, we also provide in the release the *Classical Architecture*, so that researchers can compare against legacy work that may not support modern features such as multipiers, carry chains and memories. Figure 5 illustrates the classical soft logic block. It consists of N basic logic elements (BLEs), where each BLE is a LUT with an optionally registered output [Betz et al. 1999]. We provide a Classical Architecture with ten general inputs and four BLEs per cluster (N = 4), and each of the LUTs has four inputs. All routing wires are length 4, single-driver, with $Fc_{in} = 0.15$ and $Fc_{out} = 0.25$, and $Fs = 3$. There are eight I/O pins per I/O block. The area and delay models come from a 90*nm* transistor-optimized architecture from the iFAR repository [Kuon and Rose 2008].

3.3. Other Architecture Files

Several variants of the Comprehensive and Classical Architectures described earlier are included in the VTR 7.0 release as indicated in Table I. These include an architecture that uses a depopulated crossbar with the logic cluster to save area and simplified architectures without fracturable LUTs, without carry chains, and/or without hard logic. We also include an architecture with hard floating point units to show how to use VTR to model an unconventional logic block.

3.4. Benchmarks

This release includes three sets of benchmarks appropriate for use with different architecture files. The benchmarks are largely the same as those found in the VTR 1.0

Table I. Major Architecture Files

File name	Description
k6_frac_N10_frac_chain_mem32K_40nm	Comprehensive Arch.: ten fracturable 6-LUTs with carry chains, 32kb RAM and hard multipliers
k6_frac_N10_frac_chain_depop50_mem32K_40nm	Comprehensive Arch. with depopulated crossbar
k6_frac_N10_mem32K_40nm	Comprehensive Architecture without carry chains
k6_frac_N10_40nm	Comprehensive Architecture without carry chains or hard logic
k4_N4_90nm	Classical Architecture: four 4-LUTs per logic cluster and no hard blocks
hard_fpu_arch_timing	Classical Arch. with hardened floating point block

Table II. Heterogeneous Benchmark Data and Domain

Circuit	# 6LUTs	# Mult	# Mem Bits	# Add	Max Add Size	Domain
bgm	38,537	11	0	12,719	98	Finance
blob_merge	8067	0	0	4510	14	Image Proc
boundtop	3053	0	32,768	521	20	Ray Trace
ch_intrinsics	425	0	256	0	0	Memory Init
diffeq1	362	5	0	202	67	Math
diffeq2	272	5	0	204	68	Math
LU8PEEng	25,251	8	46,608	6083	47	Math
LU32PEEng	86,521	32	673,328	17,819	47	Math
mcml	107,784	30	5,210,112	29,611	130	Med Physics
mkDelayWorker32B	5588	0	532,916	1050	34	Packet Proc
mkPktMerge	239	0	7344	96	7	Packet Proc
mkSMAdapter4B	1960	0	4456	570	34	Packet Proc
or1200	3075	1	2048	896	121	Soft Proc
raygentop	1884	18	5376	831	33	Ray Trace
sha	2001	0	0	329	37	Cryptog
stereovision0	9567	0	0	5105	21	Comp Vision
stereovision1	9874	152	0	4286	20	Comp Vision
stereovision2	11,012	564	0	15,096	35	Comp Vision
stereovision3	174	0	0	31	12	Comp Vision

release. The most important set are the *heterogeneous* benchmarks, which come from a variety of real applications, including arithmetic-focused solvers, processors, medical physics simulations, and image processing. Table II gives the number of different types of resources each benchmark uses: the number of LUTs in the circuit when mapped to 6-LUTs, the number of multipliers, the total memory bits, the number of adders, and finally the number of bits (i.e. length) of the largest adder. It also gives the application domain of each benchmark. We encourage users to use the heterogeneous benchmark set (together with the Comprehensive Architecture file) in their research, as they better represent the architecture and use of modern FPGAs than simpler benchmark sets and architectures.

For researchers interested in comparing with legacy work, we include the classic MCNC circuits. These benchmarks are compatible with all architecture files but we recommend using them with the Classical Architecture file. Lastly, we include small, "toy" benchmarks that contain floating-point blocks for use with the floating-point architecture file only.

Table III gives the data from a full VTR run (with default settings) of the heterogeneous benchmarks on the Comprehensive Architecture. The columns from left to right

Table III. Heterogeneous Benchmark Statistics after Full VTR Flow

Circuit	W_{min} (Tracks)	Wirelength (# CLBs crossed)	Crit Delay (ns)	Num CLB
bgm	82	607,288	30.81	3925
blob_merge	84	114,935	15.01	707
boundtop	54	29,302	6.30	256
ch_intrinsics	48	3,938	3.80	37
diffeq1	62	10,367	16.82	35
diffeq2	58	8,959	12.72	26
LU8PEEng	96	426,212	86.92	2566
LU32PEEng	142	1,777,082	80.48	8762
mcml	130	1,604,850	44.33	7676
mkDelayWorker32B	76	128,101	7.58	511
mkPktMerge	50	14,965	4.15	21
mkSMAdapter4B	60	26,001	5.76	201
or1200	82	55,672	9.00	289
raygentop	76	33,251	5.06	208
sha	62	18,243	9.99	202
stereovision0	64	118,420	4.47	1126
stereovision1	108	205,770	5.91	1085
stereovision2	124	730,344	16.20	2258
stereovision3	32	745	2.79	13

are: the circuit name, the minimum routable channel width (W_{min}), the total wirelength (measured in logic blocks traversed) and the critical path delay for a "low stress" routing where the channel width is W_{min} plus 30%, and finally, the number of soft logic blocks.

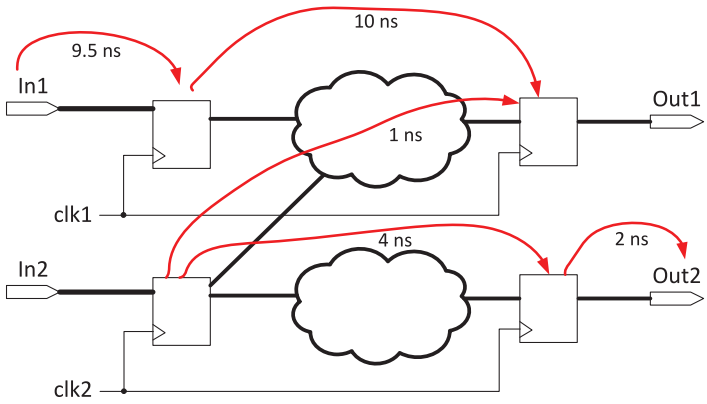
4. MULTIPLE CLOCK ANALYSIS AND OPTIMIZATION

Prior versions of VPR have supported only a single, simple form of timing analysis. All registers in the design were considered to be on one clock domain, and I/Os were implicitly assumed to be registered on this single clock domain. However, this assumption does not match the behaviour of modern designs, which commonly include many clocks [Hutton et al. 2005]. For example, a *single* DDR3 interface typically requires 5 to 7 clocks in an FPGA; designs with various memory and serial interfaces therefore commonly contain dozens of clocks. In addition, I/Os cannot always be modeled as simple registers – sometimes data is registered in the chip on the other end of the board trace producing a different variety of timing constraint. To allow experimentation with more complex designs and realistic benchmarks, VTR 7.0 allows one to specify, analyze and optimize for more complex timing constraints. Currently only the VPR stage of the VTR flow analyzes and optimizes these constraints as VPR has the most impact on design timing; possible future work would be to make the earlier parts of the flow also understand and optimize timing constraints.

4.1. Timing Constraint Specification

To allow specification of complex timing constraints in a flexible and standard way, we support a subset of the Synopsys Design Constraint (SDC) language [Gangadharan and Churiwala 2013], including:

—a (potentially different) target period for each clock in a design and a relative phase for each clock vs. other clocks;




```
create_clock -period 10 clk1
create_clock -period 4 -waveform {1 3} clk2
create_clock -period 4 -name output_clk
set_false_path -from [get_clocks{clk1}] -to [get_clocks{output_clk}]
set_input_delay -clock clk1 -max 0.5 [get_ports{in1}]
set_output_delay -clock output_clk -max 1 [get_ports{out*}]
```

Fig. 6. Example circuit with 3 timing constraints.

—specification of whether groups of clocks are "frequency-related" and hence paths between them should be analyzed synchronously, or whether transfers between the clocks use asynchronous techniques and therefore should not be analyzed;
—optional board-level delays to and from chip inputs and outputs, respectively.

VPR 7.0 can also be run without an SDC file, as a convenience to CAD researchers using simpler circuits. For single-clock circuits, VPR 7.0's "no-SDC" behaviour matches that of earlier VPR versions to allow easy quality comparisons; all I/Os are assumed to be registered on the single clock, and the frequency of the single clock is maximized. For multi-clock circuits with C clocks, VPR (without an SDC file) will assume there are no synchronous transfers between clocks or between the I/Os and clocks, and will attempt to simultaneously maximize the frequency of all C clock domains.

4.2. Timing Analysis

To compare CAD algorithms or FPGA architectures, we need both to *analyze* and report a design's timing and *optimize* all the timing constraints. As in prior versions of VPR, timing analysis utilizes a graph that represents all the timing dependencies between circuit elements that are connected [Betz et al. 1999]. VPR parses the SDC file and creates a data structure that stores the number of *timing constraints* that will require separate graph traversals and the source nodes, sink nodes, and timing constraint value (e.g., 10ns) for each constraint. Figure 6 shows a sample circuit and its SDC file. The first two lines constrain the two real clocks in the design, while the third line creates a "virtual" clock for the output I/Os. The clock waveform created by each of these three lines is also shown in Figure 6. The fourth line indicates that paths from *clk1* to the design outputs should not be analyzed, while the last two lines specify board-level delays to and from the I/Os. This SDC file leads to 5 timing constraints, and the curved arrows in the figure indicate the source and destination nodes of each constraint, while the value of the timing constraint is marked on the arrow.

For every timing constraint, we perform a pair of breadth-first traversals to compute slacks. First, we perform a forward breadth-first traversal from the source sequential elements (registers, I/Os and any embedded blocks like RAMs that contain sequential elements) to compute the arrival time at each node. Second, we perform a backward breadth-first traversal from sink sequential elements, using the value of the timing constraint as the destination required time, to compute required times at all nodes. The slack of a connection i for this timing constraint is then simply $T_{required}(i)$ - $T_{arrival}(i)$. A connection can be part of multiple paths with different timing constraints; for such connections *slack(i)* is the minimum slack value for connection i over all timing constraints. For a design with C timing constraints, $2 \cdot C$ breadth-first traversals are required to analyze all register-to-register paths. The circuit of Figure 6, for example, requires ten (five forward and five backward) breadth-first traversals to compute the slack of every connection across all constraints. For designs with several clocks, we have found timing analysis is still a small part of VPR's runtime; however, CPU time optimizations such as those of [Hutton et al. 2005] would be a useful future work to keep timing analysis fast for designs with many clocks or many different multicycle values.

4.3. Timing Optimization

The three major algorithms of VPR—clustering, placement and routing—all simultaneously optimize wirelength and circuit timing. To optimize circuit timing, the optimizers in earlier versions of VPR determine the timing importance, or *criticality*, of a connection c via the following criticality metric [Betz et al. 1999]:

$$Crit(c) = 1 - \frac{slack(c)}{D_{max}}, \tag{1}$$

where D_{max} is the delay of the longest path in the design. With multiple timing constraints, (1) is no longer sufficient, as the normalization by D_{max} does not result in a correct ranking of the criticalities of connections covered by different timing constraints. To see this, consider the circuit of Figure 6 and assume that the longest path in the *clk1* domain is 10ns, while the longest path in the *clk2* domain is 3 ns. Since all slacks in the *clk2* domain will be $\leq$ 4 ns, the criticality of *every* connection on a path between *clk2* registers will be ≥ 0.6, if we apply (1) with $D_{max} = 10$ ns. The criticality of connections in the *clk1* domain, on the other hand, can span the full range from 0 to 1; the net effect is over-optimization of the *clk2* domain at the expense of *clk1*. The correct normalization factor is now connection-specific; it is the value of the timing constraint being applied in the current timing graph traversal. That is:

$$Crit(c) = \forall_{constraint}\ \max\left[1 - \frac{slack_{constraint}(c)}{\max\left(T_{constraint}, D_{max\_constraint}\right)}\right], \tag{2}$$

where $T_{constraint}$ is the maximum permitted timing value (e.g. clock period) associated with a timing constraint and $D_{max\_constraint}$ is the maximum delay amongst the paths covered by that constraint. The *max* in the denominator of (2) ensures that we compute reasonable criticalities even if a timing constraint is impossible, and some paths have delays greater than $T_{constraint}$. Note that a connection can be part of multiple paths that contribute to multiple timing constraints; accordingly (2) computes the maximum criticality over all constraints for each connection c.

Section 3.4 introduced the *heterogeneous* benchmark circuits on which we test multi-timing constraint optimization. Unfortunately, 18 of these 19 benchmarks contain only one clock, showing they still fall below the complexity level of modern industrial designs. To test multi-timing constraint optimization, we therefore ran a series of experiments where we set different timing constraints for: (i) the (usually single

Table IV.
Relative timing and wirelength vs. timing constraints. All results are geometric averages from the heterogeneous benchmark circuits run on the Comprehensive Architecture without carry chains with a channel width of $W_{min} + 30\%$.

Clock constraint	IO constraint	Clock Period	I/O Delay	Wirelength
0.1 ns	0.1 ns	1.0	1.0	1.0
0.1 ns	1000 ns	0.99	1.31	0.99
1000 ns	0.1 ns	1.21	0.87	0.93
1000 ns	1000 ns	1.16	1.22	0.87

clock) register-to-register paths and (ii) paths between I/Os and registers for these benchmarks. Table IV summarizes the results and illustrates that VPR is capable of simultaneously optimizing multiple timing constraints well. For example, optimizing both clock period and I/O timing (by constraining both to an impossible-to-meet 0.1 ns constraint) improves I/O timing by 31% vs. optimizing only clock period (by setting an easy I/O timing constraint of 1000 ns). This comes at only a 1% cost in the clock period and a 1% increase in routed wirelength. Optimizing I/O timing alone can produce a further 13% improvement in I/O timing and can reduce wirelength by 7% vs. optimizing both clock and I/O timing, but it increases the clock period by 21%. Thus, VPR responds correctly to the constraints specified, trading off performance for wirelength accordingly.

5. CARRY CHAINS

One of the key architectural decisions made in FPGAs is to determine which, if any, logic functions 'deserve' to be built in *hard*, specific logic rather than be implemented in the more generic *soft* logic. Early in the history of FPGAs, special hardened circuitry was created for arithmetic circuits that require carry-like logic [Hsieh et al. 1990; Woo 1995] to improve the performance of addition, subtraction and any other operation that used these functions. Since the long chain of carry computations (1 per bit of addition/subtraction) was often the critical path of circuit, it was deemed worthwhile to harden some or all of the logic, and also to harden the routing links between successive bits so as to make the chain as fast as possible. Also, since adders are both common and of variable length, the typical approach has been to locate the dedicated carry logic within the soft logic basic logic element and to provide high-speed links both within the CLB, and between CLBs. The notion of specialized logic within the soft logic block, connected by high-speed routing can be implemented in quite a different number of ways, so our goal in the VTR toolset is to support a variety of architectural options. The hardening of the routing, and the specificity of the carry logic requires that the entire tool flow be modified in significant ways to make successful use of the "carry chain" structures. In the following sections we describe how each step in the VTR flow is required to change, and some details of how the change is managed.

5.1. Architecture File Description of Carry Logic and Routing

We added functionality to the architecture description file to allow specification of hardened adders, intra-CLB fast carry links, and inter-CLB fast carry-links.

As described in the next section, the ODIN II Verilog elaboration tool must be made aware of the existence of special hardened adder units. This is done in the architecture file snippet (in the ODIN II models section of the file) shown in Figure 7. The key word 'adder' directs ODIN II to synthesize to a hard adder primitive instead of emitting logic equations for the adders that would be implemented by later synthesis steps in soft logic. As shown, the architecture file lists the input and output ports of the adder.

```
<model name="adder">
  <input_ports>
    <port name="a"/>
    <port name="b"/>
    <port name="cin"/>
  </input_ports>
  <output_ports>
    <port name="cout"/>
    <port name="sumout"/>
  </output_ports>
</model>
```

Fig. 7. Example declaration of adders for ODIN II.

```
<mode name="arithmetic">
  <!-- Adder instantiation -->
  <pb_type name="adder" blif_model=".adder" num_pb="1">
    <input name="a" num_pins="4"/>
    <input name="b" num_pins="4"/>
    <input name="cin" num_pins="1"/>
    <output name="sumout" num_pins="4"/>
    <output name="cout" num_pins="1"/>
    <!-- Delay information omitted for brevity -->
  </pb_type>
  <interconnect>
    <direct name="cin" input="fle.cin" output="adder.cin">
      <pack_pattern name="chain" in_port="fle.cin" out_port="adder.cin"/>
    </direct>
    <direct name="cout" input="adder.cout" output="fle.cout">
      <pack_pattern name="chain" in_port="adder.cout" out_port="fle.cout"/>
    </direct>
    <!-- Rest of interconnect information omitted for brevity -->
  </interconnect>
</mode>
```

Fig. 8. The adder mode in a CLB, and intra-CLB carry links.

Next, the physical adder itself is defined as a *mode* in the CLB section of the architecture file.[1] Figure 8 shows the code that defines a 4-bit adder and the links between the individual bits within the CLB. In this case, for example, the CLB can be used in an "arithmetic mode" as a 4-bit adder, or alternately as standard soft logic with LUTs and flip-flops (this mode is not shown for brevity). The architect can specify an adder of any size, though this is of course constrained by the number of input/output pins on the CLB and its internal routing connectivity. Multiple small adders can be chained together within a CLB with fast, dedicated-routing carry links; the Comprehesive Architecture, for example, uses this approach to combine twenty 1-bit hardened adders within one CLB. The *pack_pattern* construct labels the fast carry links inside a CLB so that a new, "prepack" phase of the packing algorithm can exploit these links, as described in Section 5.4.

Some additions may be too large to fit in one logic block and accordingly we allow the specification of fast, dedicated physical connections between the adders of adjacent CLBs. Figure 9 illustrates an example, vertical adder carry chain between CLBs, while

[1]The mode construct specifies mutually-exclusive functionality within a complex block [Luu et al. 2011].

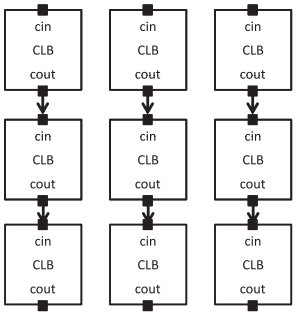


Fig. 9. Example of a vertical carry chain that propagates downwards.

```
<directlist>
  <direct name="adder_carry" from_pin="clb.cout" to_pin="clb.cin"
          x_offset="0" y_offset="-1" z_offset="0"/>
</directlist>
```

Fig. 10. Example definition of inter-CLB carry links.

Figure 10 shows its specification. The *direct* XML tag specifies these links in a relative fashion: the *from_pin* and *to_pin* attributes specify which pins are connected (`cout` $\rightarrow$ `cin`), and the offset attributes specify which logic blocks connect. Here, the `x` offset is 0 and `y` offset is -1 to specify chains that drive the adjacent block to the south.

5.2. Elaboration of Arithmetic in ODIN II

ODIN II infers addition/subtraction from the either the plus/minus sign in the Verilog HDL code or from an explicit instantiation of an adder macro. If the inferred adder (i.e. the logical adder) is larger than the size of the physical adders, ODIN II implements the logical adder using a composition of smaller adders of the actual physical size. In the absence of hardened carry logic, addition and subtraction is synthesized into soft logic as in prior versions of VTR. Subtraction is handled by ODIN II in standard 2's complement fashion: one of the inputs as negated and the carry-in is asserted.

Figure 11 shows example Verilog code for the explicit and implicit instantiation of a hard adder. An addition is done via instantiation, while a subtraction is achieved via inference. In both cases, ODIN II will generate instantiations of a hard-adder primitive for the downstream tools to process. As well, ODIN's multiplier synthesis code has been modified to make use of hard addition/carry logic (if present in the architecture) when combining results from hard multiplier blocks. Figure 12 illustrates the synthesis of a 6-bit adder (my_adder in Figure 11) into an architecture in which the physical adder is 4 bits wide, as in Figure 8. Here, two 4-bit hard adders are required to fully implement the six bits, and the last two bits of the second adder are wasted.

We have found that for very small logical adders, the implementation of arithmetic is more efficient and faster in soft logic. As such, a configuration option allows a user to specify the minimum size addition (in terms of the number of bits) required for ODIN to make use of hard adders. We assume that the carry lines within a hard adder block and the carry-chain that joins hard adders together cannot be directly accessed by other interconnect (i.e. these are not routed into general-purpose interconnect). Consequently, if the carry-out bit of an adder/subtractor is used elsewhere in a user circuit, ODIN II pads the adder to force carry-out data to a sum bit (which can drive general interconnect).

```
module exampleInstance (in1, in2, sum, diff);
input [5:0] in1, in2;
wire cin = 1'b0;
wire cout;
output [6:0] sum, diff;

adder my_adder (in1, in2, cin, cout, sum);
assign diff = in1 - in2;

endmodule
```

Fig. 11. Example explicit and implicit instantiation of hard adders.

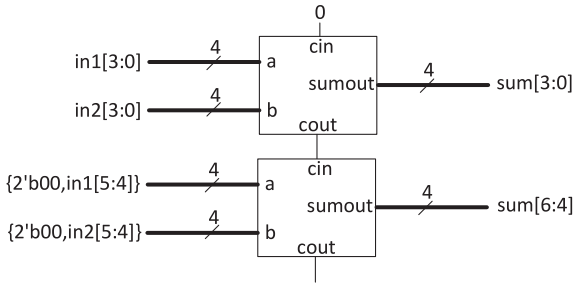


Fig. 12. Example 6-bit logical adder mapped to two 4-bit adders.

5.3. Logic Synthesis in ABC

The logic synthesis tool needs to, at the very least, pass the hard adder logic identified during elaboration down to the later stages. An ideal logic synthesis tool would be able to make intelligent decisions about when to keep adder logic, when to synthesize non-adder logic into adders, and when it makes sense to combine adder logic with the regular logic of a circuit. Our logic synthesis tool, ABC, treats adders as black boxes, passing them through the flow without any logic optimizations. The inputs and outputs of the adders are treated as primary outputs and inputs by ABC, preventing many optimizations across them – for example, sweeping away unused inputs or leveraging constant inputs for optimization.

5.4. Packing in VPR

We have added a "pre-packing" stage in VPR that assembles the adders elaborated by ODIN II into groups, based on the number of adders inside a logic block. The main packing algorithm keeps these pre-packed groups together to ensure that the adders use the fast carry links within and between CLBs.

One complication with carry chains is that when a carry chain is packed into a logic block, that logic block will often still have additional LUTs and flip-flops available for non-adder logic. To utilize these resources, the packer fills these LUTs and flip-flops using the same greedy packing algorithm used for other logic blocks. For example, in the case of a registered addition, the flip-flops are packed in the same CLB as the carry chain. Currently the prepacker requires that the architect label all carry chain links in the architecture file, using the *pack_pattern* construct shown in Figure 8.

5.5. Placement - VPR

If a logical adder spans more than one CLB, the dedicated inter-CLB carry links must be utilized to maintain high performance. These links are "hardwired" (not programmable)

so the CLBs comprising a carry chain must be placed in a specific relative orientation to make use of them: for example vertically adjacent in the architecture of Figure 9. We have modified the placement engine in VPR to support the relative placement of CLBs in any orientation. The architecture description file, as described earlier, specifies dedicated carry connections between CLBs, and the specific pins on the CLBs used for this purpose. When such pins are used by the design netlist, VPR groups the relevant CLBs together into a "placement macro" which maintains the relative placements of individual logic blocks within it. In VPR 7.0, the macro notion is not limited solely to vertical carry chains; instead any shapes and sizes can be defined.

Each placement macro is placed as if it were a single entity. For example, if a carry chain spans three logic blocks in a 3x1 column, then that entire 3x1 block of CLBs moves as a unit. Thus, the simulated annealing-based placement engine must swap it with 3 other vertically adjacent logic blocks (or vacant slots).

5.6. Interconnect Generation – VPR

Carry-chains require direct connections between logic block pins that do not pass through general interconnect wires. Accordingly, we have modified the inter-CLB routing architecture generator in VPR [Betz and Rose 2000] to add edges directly between the routing resource nodes that represent the appropriate pins as required by the new *directlist* construct of Figure 10. The carry pins in commercial FPGAs *must* use these dedicated links; they have no connection to the general routing. To model this restriction, we now allow each pin on a logic block to specify a different fraction of routing wires to which it should connect, F_c. For the carry pins, an architect would set F_c to 0. When a design netlist uses a pin that cannot connect to the general routing, but which can make a *directlist* connection to a pin on an adjacent block, VPR automatically creates a placement macro containing the relevant blocks to ensure the placement will be routable.

In keeping with the architectural flexibility goal of VPR, these new constructs can be used to model more than just carry chains. For example, many commercial FPGAs incorporate nearest-neighbour direct connections that allow "regular" logic block outputs to connect to the inputs of nearby blocks without use of general routing wires. An architect can now experiment with such connections by specifying appropriate *directlist* connections; by specifying a non-zero F_c value for the block pins involved these direct connections become optional, rather than mandatory, routing choices.

5.7. Hard Adder Logic Experiments

We now illustrate the use of hard adders and their impact on circuit speed. We make use of the Comprehensive Architecture described in Section 3, which includes multipliers, memories, a hard adder and a standard soft logic and routing architecture.

In the first experiment, we measure the impact of hard adder size on critical path delay for designs that consist solely of an adder of varying size, with registered inputs and outputs. The following four cases are used: 1) The VTR flow with hard adders (each CLB has 20 one-bit adders, 2 adders for every fracturable 6-LUT), 2) the VTR flow without dedicated adders, 3) the Altera Quartus II 12.0 flow with dedicated adders on an Altera Stratix IV EP4SGX70HF35C2 device, 4) the same as #3 but with carry chain use disabled.

Figure 13 plots the critical path delay in nanoseconds versus the size of the adder in bits for all four scenarios. In all cases, the delay increases linearly as the number of adder bits increases, as expected. We also observe the expected performance improvement when using hardened adders for large user adders. For example, a hard 128-bit adder is 5.6× faster than a soft 128-bit adder in our architecture and 3.4× faster on a Stratix IV architecture. We notice a difference of approximately 0.6 ns between the

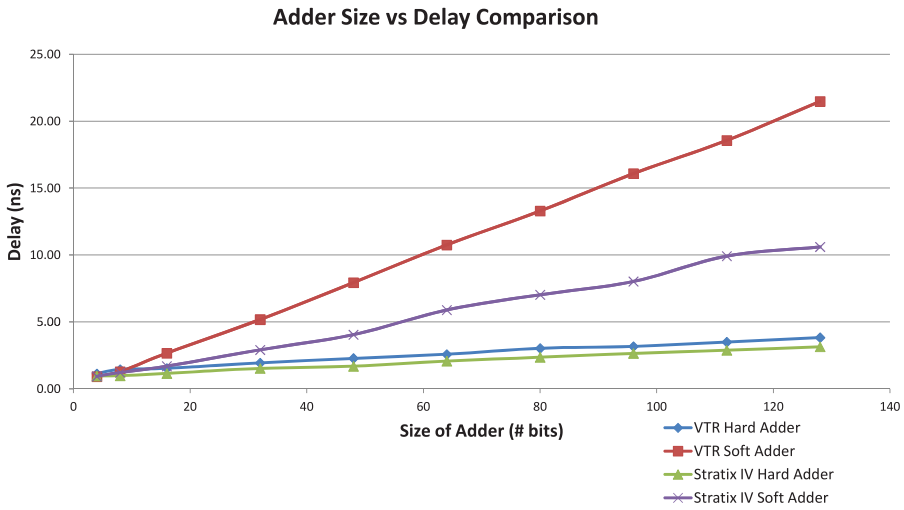


Fig. 13. Impact of adder size on critical path delay.

critical path delay of our carry-chain FPGA compared to that of Stratix IV. Quartus II utilizes differences in LUT input delays to optimize a critical path, and Stratix IV also has a nearest-neighbour interconnect which lets the registers feeding the carry chain bypass the general routing. VTR with the Comprehensive Architecture does not support either of these features, leading to the extra 600 ps delay to reach a carry chain. We also observe that soft adders are faster than hard adders for small logical adders. For the VTR case, the crossover point occurs at around 12 bits, whereas for Stratix IV the crossover point occurs at around 4 bits.

A second experiment measures the impact of our carry-chain architecture on the 19 heterogeneous benchmarks described in Section 3.4. Each circuit was implemented in an FPGA with and without hard adders, and Table V shows the results. The first column gives the benchmark name. The next five columns are all ratios of the result achieved in an architecture with hard adders to those achieved in architecture without hard adders; they are the minimum routable channel width (W_{min}), the critical path delay of the circuit routed at 1.3 $\times W_{min}$, the number of nets between logic blocks, the number of soft logic blocks, and the total number of LUT + adder primitives in the circuit. The final column gives the length (in bits) of the largest adder chain in the circuit. We observe that, as expected, circuits with large adders tend to have an improvement to critical path delay while circuits without large adders have little if any impact. We also observe an increase in the number of CLBs needed with hard adders, which was unexpected. This is due to the black-boxing of the hard adders in the ABC tool preventing optimization across adder boundaries.

6. ENERGY MODEL

Energy consumption has become a major factor in the design of new FPGA architectures. An FPGA's energy consumption is a function of its architecture, circuit design, and the CAD algorithms that synthesize the user design into the device. It is important to integrate energy modeling into the CAD flow, both to enable researchers to evaluate the energy consumption characteristics of new FPGA architectures and to develop new CAD algorithms that can optimize energy.

Power models have been created for previous versions of VPR [Poon et al. 2005; Li and He 2005; Jamieson et al. 2009], but each of these efforts focused only on the classical

Table V. Hard versus Soft Adders. All Values Except Longest Chain are the Hard Adder Architecture Result Divided by the Soft Adder Result

Circuit	W_{min}	Delay	Num Nets	Num CLB	LUTs + Adders	Longest Chain
bgm	0.71	1.16	1.32	1.34	1.58	98
blob_merge	1.14	1.45	2.13	1.30	2.09	14
boundtop	0.90	0.98	1.08	1.10	1.17	20
ch_intrinsics	0.96	0.96	0.97	1.00	1.00	0
diffeq1	1.19	0.75	1.11	0.97	1.16	67
diffeq2	1.12	0.77	1.30	0.96	1.48	68
LU8PEEng	0.84	0.75	1.23	1.22	1.38	47
LU32PEEng	0.82	0.70	1.21	1.23	1.37	47
mcml	1.25	0.56	1.55	1.16	1.35	130
mkDelayWorker32B	1.00	1.04	1.15	1.14	1.19	34
mkPktMerge	1.09	0.91	1.05	1.40	1.44	7
mkSMAdapter4B	1.07	1.02	1.24	1.22	1.28	34
or1200	1.11	0.67	1.26	1.12	1.30	121
raygentop	1.12	1.00	1.29	1.20	1.26	33
sha	1.24	0.73	0.92	0.97	1.02	37
stereovision0	1.07	1.03	1.17	1.24	1.28	21
stereovision1	1.04	1.03	1.22	1.22	1.38	20
stereovision2	0.81	0.93	1.06	0.94	0.87	35
stereovision3	0.94	1.04	1.01	1.00	1.14	12
geomean	1.01	0.90	1.20	1.14	1.28	
stdev	0.15	0.21	0.26	0.14	0.26	

LUT-based soft logic cluster, and did not support any kind of hard heterogeneous block. None of these enhancements were integrated into the trunk of the software and hence quickly became obsolete with new developments. In particular, those models do not support the broader architecture description language created in VPR 6.0. The new VTR 7.0 supports a much wider range of architectures, where architects can describe arbitrarily complex blocks using the architecture description language.

In this section we present an overview of the energy model integrated into VTR 7.0, which provides the ability to model the underlying FPGA at two levels of abstraction - one quite detailed (at the transistor and layout level), and one relatively simple and macroscopic. We feel these two methods are necessary as the architect may not have the resources to complete a detailed design of each block and may be satisfied with less accurate approximation. We provide both static and dynamic energy estimates.

6.1. Detailed Energy Estimation

The detailed energy estimation method described in Goeders and Wilton [2012] will provide the most accurate capture of an architecture's energy consumption in the VTR flow. It works at the transistor level, but much of the transistor-level design for known primitives (LUTs, flip-flops, and multiplexers) is provided by VPR 7.0 itself. In the detailed method, the physical structure of the logic block described in the architecture file is translated into inverters, simple multiplexers, and wires. The architecture generation engine computes wire lengths by estimating overall area and approximating the resistance and capacitance of the wires. It then determines transistor sizes from the capacitance driven, by applying the method of logical effort [Weste and Harris 2010] which seeks to generally minimize the delay. Alternatively, users can override automatic buffer sizes by explicitly defining them in the architecture file.

Once these structures are generated, energy estimation is performed for each inverter, multiplexer, and wire. These estimates depend on both the behavior of the

input signals and the properties of the target technology. This information is generated using two additional tools that are included in the release. The signal behaviors (the switching activity and static probability) are computed using the ACE 2.0 [Lamoureux and Wilton 2006] activity estimation tool, which the user must operate together with known input vectors or random stimulation vectors. The CMOS technology information must be provided in the form of Spice-level models. These are used by a script that performs multiple SPICE simulations to extract a set of specific process characteristics, including: transistor capacitances, leakage currents, P-to-N inverter ratio, and multiplexer output voltages. Since this only needs to be done once per technology, these results are stored in a separate VTR 7.0 energy technology file. The VTR 7.0 release provides energy technology files for the PTM [Cao 2008] 22 nm, 45 nm, and 130 nm technologies, to save users effort.

As described in Goeders and Wilton [2012] both the inter-CLB routing circuitry and the global clocking structure are always modeled using this procedure, and the logic block can also make use of it. However, if the logic block is too complex or different in its internal structure the architect may need to resort to modelling at a more abstract level, which is described in the next section. Note that the energy modelling infrastructure does not yet support multiple clocks.

6.2. Macro-Model Energy Estimation

An architect may not have the resources to engage at the given detailed level for certain logic blocks, and other "hard" blocks like RAM and DSP blocks are not currently energy-modeled at this detailed level in VPR, so it is important to provide higher-level, faster-to-develop, abstractions. We support three simpler power estimation methods. In the first, the architect models a block's energy consumption by specifying the amount of energy consumed per toggle of each input pin to the block. In the second method, the user provides the equivalent internal capacitance of the block, and the activity is averaged across all input pins to calculate the dynamic power. In the third method, the user provides the absolute value of the dynamic power of the block. For each method, the static power must be provided as an absolute value. The estimation method for each block is specified within the architecture file. Data for the more abstract levels of the energy model can typically be found by perusing the spread-sheet-based early power estimators provided by FPGA vendors [Altera 2012a; Xilinx 2011].

6.3. Results

In this section we illustrate some of the measurements possible with the new energy models. We performed power estimation on the heterogeneous benchmark circuit set using the Comprehensive Architecture without carry chains. This architecture contains a fairly standard soft logic architecture, together with hard memories and fracturable multipliers. In this architecture, the power usage of routing, CLBs, and the clock network are modelled using the detailed transistor-level approach. Power dissipation of memories is modelled as an energy per toggle of the clock pin when the RAM is enabled, and multipliers are modelled as energy per toggle of each input pin. The static power of memories and multipliers are ignored. In the experiments, the FPGA is auto-sized to fit the benchmark circuit, the channel width is the minimum channel width for the circuit plus 30%, and the clock frequency set to the F_{max} reported by timing analysis for each circuit.

Table VI provides the total estimated power usage for each of the benchmark circuits (described in Section 3.4) as well as a breakdown between the major components of the FPGA. The total power consumption ranges from 5 mW to 481 mW. On average, 51% of the total power is static power due to subthreshold leakage; this high percentage is partially due to the low operating frequency of some of the circuits. Overall, global

Table VI. Power Usage, and Breakdown by Circuit (45 nm)

	Circuit Properties			Architecture		Power		Power Breakdown					
Benchmark Circuit	# LUTS (k=6)	# Memories	# Multipliers	FPGA size (# CLBs)	Channel Width	Freq. (MHz) (f_{max})	**Total Power (mW)**	% Dynamic (vs Static)	% Routing	% CLBs	% Memories	% Multipliers	% Clock
bgm	32275	0	11	3969	150	37.8	**204.7**	33	53	44	0.0	1.6	1.9
blob_merge	6019	0	0	784	94	96.8	**28.9**	32	40	53	0.0	0.0	6.4
boundtop	3009	1	0	324	78	152.3	**16.1**	56	44	47	0.8	0.0	7.8
ch_intrinsics	401	1	0	64	66	249.3	**5.0**	73	44	41	4.5	0.0	10.4
diffeq1	484	0	5	196	78	46.5	**8.9**	60	39	28	0.0	30.0	3.0
diffeq2	320	0	5	196	62	60.8	**9.0**	66	34	26	0.0	36.0	3.7
LU8PEEng	22601	45	8	2809	148	8.7	**106.7**	9	49	49	0.7	0.2	0.8
LU32PEEng	76153	168	32	9216	232	8.9	**438.9**	10	58	40	0.6	0.2	0.7
mcml	101487	159	27	9025	144	12.7	**301.4**	11	49	48	1.2	0.2	1.2
mkDelayW.	5214	43	0	2304	102	129.5	**105.3**	60	50	34	9.6	0.0	6.4
mkPktMerge	231	15	0	676	70	213.7	**37.2**	76	43	32	16.4	0.0	8.9
mkSMAdap.	1947	5	0	324	72	178.5	**17.8**	64	43	40	8.6	0.0	8.1
or1200	3041	2	1	625	88	75.0	**23.8**	42	47	47	1.1	0.0	4.7
raygentop	2142	1	7	289	94	202.6	**29.4**	76	46	31	0.6	17.4	5.0
sha	2273	0	0	289	66	69.9	**10.5**	36	37	58	0.0	0.0	4.8
stereovision0	11289	0	0	1225	78	229.1	**75.3**	62	41	51	0.0	0.0	8.0
stereovision1	10277	0	38	1444	136	177.1	**127.4**	69	58	35	0.0	2.7	4.9
stereovision2	29196	0	213	7396	208	55.2	**480.5**	51	63	25	0.0	9.4	2.4
Average	**17131**	**24**	**19**	**2286**	**109**	**111.4**	**112.6**	**49**	**47**	**41**	**2.4**	**5.4**	**4.9**

routing is responsible for 49% of the total power, CLBs (including local routing) for 41%, and the clock network for 5%. Memories and multipliers also contribute a few percentage points on average; however, since their static power is ignored, this directly depends on the number of memories and multipliers instantiated in the user circuit.

As expected, the results show that generally the power consumption grows with the size of the benchmark circuit. However, other factors also play a large role. For example, the second largest circuit (*LU32PEEng*) consumes 46% more power than the largest circuit (*mcml*), primarily due to the much larger channel width (232 versus 144). This results in 58% of the total power of *LU32PEEng* being from routing, versus 49% for the larger *mcml* circuit. Signal activities and operating frequency also play a very significant role. For example, 51% of the power of the third largest circuit, *stereovision2* is dynamic power, versus only 10% and 11% for the largest two circuits. This behaviour is also evident in the smaller *raygentop* and *sha* benchmarks. Although they use identical sized FPGAs, the *raygentop* circuit uses nearly three times the power, of which 76% is dynamic, versus 36% for the *sha* circuit.

7. CLUSTERING ADVANCES

The packing stage of the CAD flow maps "atoms" in a technology-mapped user netlist into the "primitives" present in the FPGA logic blocks. VPR 6.0 incorporated a preliminary implementation of a packer that could target complex logic blocks organized with internal hiearchy, modes of operation, heterogeneity, and arbitrary interconnect

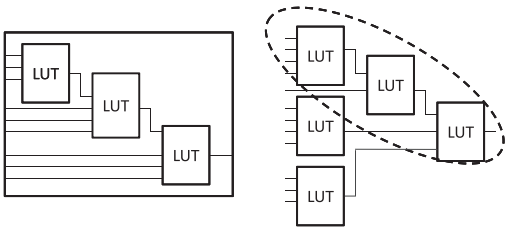


Fig. 14. Architecture with hard-wired LUTs (left) and how parts of a netlist may be clustered together to utilize it (right).

between the primitives. This enabled researchers to explore heterogeneous FPGAs containing configurable multipliers, memories, and other unique logic blocks. However, that packer exhibited poor run-time on architectures that were relatively simple (full crossbar interconnect, for example), and there were examples where it was not using the blocks as efficiently as it should have. Our long-term goal is to develop algorithms that function according to the inherent packing difficulty of an architecture – applying more computation effort only when needed. The packer in the VTR 7.0 takes several steps in that direction.

The overall goal of the packing stage is to find a legal mapping of atoms to primitives such that all logic blocks respect the constraints (routing and otherwise) imposed upon them by the architect, while at the same time optimizing for some combination of area and speed. As a result of the arbitary interconnect permitted within a block, the position of an atom in the block matters, giving rise to a placement problem within the block. There is also an associated routing problem. For example, in a logic block with four flip-flops connected as a shift register, there may exist only one correct order for packing the flip-flops, owing to limited routing flexibility. We will briefly describe a series of enhancements to improve the run-time, architecture-adapability and quality of results for the packer.

Best-Fit Enhancement. The previous version of the packer (VPR 6.0) used a first-fit packing algorithm that depended on the order that primitives were specified in the architecture description. It is far better for the algorithm itself to choose the within-block placement independent of the order in the file, and so in VPR 7.0, we use a best-fit algorithm. In the new approach, each potential primitive is considered as a target for the atom under consideration, and the one with the fewest pins is selected; in essence, we prefer to pack atoms into the least-flexible primitives that can accept them.

Grouping Highly-Constrained Atoms into Molecules. There are a number of intra-logic-block structures that have been proposed over the years that use highly constrained routing from one primitive to another. A straightforward example is the typical connection between a lookup table and flip-flop – once the LUT is selected to be a specific atom, only a connected flip-flop in the netlist may be chosen for the connected flip-flop primitive slot. Another example of this limited flexibility are the carry-chain links within a block discussed in Section 5. Such fixed connections imply *groups* of atoms that must be considered as a single unit during packing – a notion we refer to as a *molecule*. Simply put, a molecule is a collection of atoms, connected in a specific way, that must be handled as a single unit during packing. A more complex example of this is the concept of "hard-wired LUTs" proposed in [Chung and Rose 1992] and illustrated in Figure 14. That architecture contains inflexible connections between a cascade of three LUTs with no intermediate fanout. The LUT atoms in the technology-mapped netlist must match this pattern before they may be assigned to this special structure. In VPR 7.0 the architect can annotate interconnect structures in the architecture file to indicate to the packer the need to pre-group structures in the netlist. During

packing, these larger structures (molecules) are matched exactly to the fixed interconnect among the primitives.

Approximate Routing with Pin Counting. As described earlier, the basic legality check in the packer of VPR 6.0 routes every candidate atom-to-primitive mapping during packing. For architectures with full connectivity between all internal primitives this is unnecessary and it suffices to ensure that the external (to the block) pin counts are not exceeded. In VPR 6.0 such counts were used to rapidly invalidate atom-to-primitive mappings that were provably unroutable due to pin-count restrictions.

VPR 7.0 incorporates more sophisticated pin counting, based on the concept of *pin classes*. Roughly speaking, any two pins (internal or external to the block) are part of the same class if they are part of a common interconnect network. By performing pin counting at the finer-grained pin-class level, the packer can reject bad candidate mappings more often thus saving significant runtime of the intra-logic-block router.

Speculative Packing. In VPR 6.0, over 90% of runtime was spent in routing during packing – the router was invoked every time an atom was mapped to a primitive as a conservative check for legality. In VPR 7.0, the advancements in pin classes and molecules mean that in most cases for straightforward architectures the selected atoms will succeed in routing. For this reason, we speculatively pack a logic block and assume that it will route. The router is not invoked for a logic block until the packer has finished packing atoms to that logic block, after which a final route is performed. If that route check passes, as it usually does, speculative packing has succeeded so the packer moves on to pack the next logic block. If the router fails, then the algorithm will revert back to the atom-by-atom routing for that block. Currently, the packer always attempts speculative packing irrespective of the difficulty of the target logic block.

7.1. Results: Old vs. New Packer

To measure the improvements from these enhancements, we ran an experiment to compare the packer in VPR 7.0 with that in 6.0, using the benchmarks described in Section 3.4. Each circuit is run through the VTR 7.0 flow from elaboration with ODIN II to logic synthesis with ABC to packing with one of AAPack 7.0 or AAPack 6.0, and then placement and routing with VPR 7.0. All VPR settings are left at default values except placement inner_num which is set to 10. The architecture being investigated is the Comprehensive Architecture described in Section 3.1, but without carry-chains, as VPR 6.0 cannot target architectures with carry chains.

The results are shown in Table VII. The leftmost column shows the names of the benchmarks, and the other columns show the VPR 7 result divided by the VPR 6 result. The second column is the minimum channel width necessary to route the circuit, followed by the critical path delay of the circuit at $1.3 \times W_{min}$, the total time needed to perform packing, the number of inter-CLB nets and the number of CLBs after packing. The final two rows give the geometric mean and standard of deviation for the values of that column. On average, the packer is sped up 12-fold in VPR 7.0 versus VPR 6.0. This shows that our techniques use substantially less computational effort than the packer in VPR 6.0 for architectures with simple interconnect. With respect to quality of results, on average, the number of external nets is reduced by 10% which results in a 20% reduction in mininum channel width; the critical path delay is reduced by 6%, and the number of soft logic blocks is the same, showing that the new packer is substantially improved.

8. POST-ROUTING NETLIST FOR TIMING AND POWER SIMULATION

Commercial FPGA vendors provide tools to generate a post-routing simulation netlist and an associated SDF (standard delay format) file with delay information. These enable simulation of the implemented design with industry-standard simulators, such

Table VII. Packing Results VPR7/VPR6

Circuit	W_{min}	Delay	Pack Time	Num Nets	Num CLB
bgm	0.71	0.83	0.45	0.89	1.00
blob_merge	0.78	1.00	0.16	0.82	1.00
boundtop	0.78	0.85	0.03	0.86	0.95
ch_intrinsics	0.89	1.01	0.04	0.92	0.93
diffeq1	1.00	0.96	0.10	0.94	0.92
diffeq2	1.08	0.93	0.07	0.91	1.00
LU8PEEng	0.77	0.95	0.18	0.98	1.02
LU32PEEng	0.72	0.97	0.18	0.98	1.02
mcml	0.48	0.97	0.08	0.89	0.97
mkDelayWorker32B	0.80	0.92	0.07	0.91	1.01
mkPktMerge	1.00	1.06	0.14	0.99	1.00
mkSMAdapter4B	0.77	0.89	0.02	0.91	0.97
or1200	0.86	0.98	0.06	0.87	0.97
raygentop	0.92	1.00	0.11	0.90	1.01
sha	0.74	1.00	0.04	0.71	0.99
stereovision0	0.66	0.78	0.06	0.92	1.09
stereovision1	0.82	0.92	0.06	0.92	1.09
stereovision2	0.71	0.92	0.08	0.94	1.06
stereovision3	1.00	0.90	0.05	0.89	1.00
geomean	0.80	0.94	0.08	0.90	1.00
stdev	0.17	0.07	0.12	0.05	0.04

as ModelSim, with accurate timing information extracted from the physical layout. The post-routing netlist is typically structural-level HDL, comprising interconnected primitives (e.g., LUTs) that closely align with those in the FPGA architecture. Such simulations serve two main purposes: 1) verifying functionality with timing considerations, and 2) to facilitate more accurate power analysis and optimization through the use of signal switching activities that model glitches.

VTR 7.0 now includes a post-routing netlist and SDF file generator. In addition to the two purposes noted earlier, we can use this capability to verify the VTR tool flow itself. In particular, one can simulate the front-end Verilog RTL and directly compare its functionality with the post-routing Verilog netlist, providing a measure of confidence that the toolchain is indeed error free. The post-routing generated structural Verilog instantiates primitives drawn from a `primitives.v` library file in the VTR 7.0 distribution; LUTs, flip-flops, combinational multipliers, and RAM blocks are currently supported. FPGA architects wishing to perform timing simulations for architectures containing other primitive types need only describe the functionality of the new blocks in the primitives library file.

To demonstrate the new functionality, we performed a glitch analysis on a set of benchmarks. Glitches are unnecessary spurious toggles on logic signals that arise as a result of unequal path delays to a signal's driving gate. While glitches do not present a functionality problem, they do result in additional dynamic power consumption; in the designs studied in Shum and Anderson [2011], for example, glitches consumed an average of 20% of total power. We implemented the benchmarks in an architecture with ten 6-LUTs per logic block and length-4 routing segments and simulated with randomly toggling primary inputs. Each circuit was simulated twice with the same vector set: once without timing information (functional simulation) and once considering circuit

Table VIII. Average Switching Activities without and with Consideration of Glitches

Circuit	Avg. Toggle Rate (func sim.)	Avg. Toggle Rate (timing sim.)	Circuit	Avg. Toggle Rate (funct sim.)	Avg. Toggle Rate (timing sim.)
6-bit ripple adder	48.5%	57.9%	pdc	22.3%	23.7%
alu4	35.9%	38.3%	dsip	48.2%	53.6%
apex2	31.5%	33.4%	ex5p	28.7%	32.5%
apex4	24.7%	26.0%	s38417	13.7%	14.8%
bigkey	42.1%	51.1%	misex3	32.7%	35.1%

delays (timing simulation). Table VIII shows toggle rate results for the circuits, which include a 6-bit ripple adder (that uses carry chains) and representative combinational and sequential MCNC circuits. Toggle rates always increase with glitch-aware simulation; the increase ranges from 5% to 21%, with an average of 10%. We expect the new simulation netlist functionality will enable a variety of research on FPGA power estimation and optimization, as well as research on how the underlying FPGA architecture influences signal toggle rates. It will also be useful for those studying repair and redundancy methods for FPGAs.

9. NEW FEATURES OF ODIN II

Several new features of the frontend elaboration tool, ODIN II, are included in release 7.0.

Memory Inference. ODIN II will now infer both single and dual-port memories from 2-dimensional arrays of bits [Somerville and Kent 2012], rather than requiring explicit instantitation. ODIN can now also implement a logical memory out of either hard block RAMs or soft logic and flip-flops. The user can control the soft vs. hard implementation choice via a threshold parameter – if the logical memory is larger than the threshold it is built out of hard blocks; otherwise it is built out of soft logic.

Simulation. To ease and automate checking the correctness of the flow we have added a simulator into ODIN that can directly simulate either Verilog or any form of BLIF emitted by other stages of the flow [Libby et al. 2011]. We use this feature to test the correct functionality of every stage of the flow by emitting BLIF from it and simulating it against the input Verilog. The benchmarks distributed with the VTR flow are accompanied by a set of input and matching output test vectors to aid with automatic checking of results.

Visualization. ODIN now includes a tool to visualize the gate-level structure of the design early in the flow; users can interactively highlight blocks or paths and show logic values generated by the ODIN II simulator [Nasartschuk et al. 2012]. This tool helps FPGA architects understand the design structure and helps ODIN II developers test and debug new elaboration features.

10. RESULTS AND COMPARISONS

In this section, we compare the VTR 7.0 release with prior versions. The first comparison covers all of the releases of only the VPR stage on an older, simple architecture. The second comparison is between the two most recent versions of the full VTR flow on a modern heterogenous architecture.

10.1. Comparison of VPR Versions on a Homogeneous Multidriver Architecture

We compare VPR versions 4.3, 5.0, 6.0 and 7.0 to determine their quality of results (QoR) in speed and other metrics. VPR 4.3 is used as the baseline measurement. The architecture used for this experiment is one which all versions of VPR are capable of targeting (and as such, is different from the architectures presented in Section 3.4)

Table IX. VPR Cross-Version Comparison, Using Simple Architecture

	VPR 4.3	VPR 5.0	VPR 6.0	VPR 7.0	5.0/4.3	6.0/4.3	7.0/4.3
Post-pack Nets	1625	1625	1554	1553	1.00	0.95	0.95
Post-pack Clusters	731	731	730	730	1.00	1.00	1.00
Placement Est. WL	21129	21309	20273	19723	1.01	0.96	0.93
W_{min}	28.4	28.8	28.3	28.4	1.01	1.00	1.00
Total Routing WL	31166	31606	31112	31199	1.01	1.00	1.00
High Stress Crit. Path Delay (ns)	25.9	26.8	26.5	25.4	1.03	1.02	0.98
Low Stress Crit. Path Delay (ns)	19.7	19.7	19.6	19.8	1.00	1.00	1.01
Packing CPU time (s)	0.03	0.05	0.71	0.81	1.73	26.54	30.19
Placement CPU time (s)	12.2	13.3	29.5	17.9	1.09	2.43	1.47
Routing CPU time (s)	2.13	2.32	2.65	2.75	1.09	1.24	1.29
Total CPU time (s)	14.5	15.9	33.3	22.0	1.10	2.30	1.52

with the following attributes: it employs clusters with 4-input LUTs (K = 4) and four LUTs in each cluster (N = 4). The internal cluster delay is set to the same value for all versions. In the routing architecture, all wiring segments are set to length 4 and employ tri-state (bidirectional) buffers, as single-driver interconnect is not available in VPR 4.3. The routing flexibility parameters are set to $Fc_{in} = 0.25$ and $Fc_{out} = 1.00$, and $Fs = 3$. The MCNC 20 benchmark circuits are used as VPR 4.3 cannot target the more complex heterogeneous circuits in other benchmark sets. The experiment was run on an Intel i5-2500 machine (four 3.30GHz CPUs) with 16GB of main memory and a 64-bit Ubuntu Linux environment (v12.04).

The flow of the experiment is as follows: each circuit starts as a technology mapped netlist of LUTs, flip-flops and I/Os, and goes through the VPR flow to determine W_{min}. The routing is repeated at a 'low stress' channel width set to $W_{min} + 30\%$ to obtain the post-routed circuit delay and wirelength. In the table, we refer to the results obtained during the minimum channel width run as 'high stress'. Results shown are geometric means across all 20 circuits over 3 placement seeds to minimize CAD noise.

One of most significant results from Table IX is the packing CPU time from the recent two VPR versions, which is much larger than the two oldest versions. This is due to the complex legality checks inside the packer, which were discussed in Section 7. Unexpectedly, the placement in VPR 6.0 is also significantly slower than that in VPR 4.3; this has been largely remedied in version 7.0 by making the computation of net bounding boxes incremental in placement as described in Betz et al. [1999]. The quality of results from the packer are very close across all four versions. Both the VPR 6.0 and 7.0 packers produce packing solutions with ∼5% fewer post-packed nets. The placement in versions 6.0 and 7.0 also achieves ∼4-to-7% lower estimated wirelength, which is likely due to the smaller number of external nets produced by the packer.

Overall all four releases of VPR produce similar quality on this simple architecture, but CPU time grew by 2.30× in VPR 6.0 vs. VPR 4.30, as VPR added support for more complex features. In VPR 7.0 we have improved the situation by reducing CPU time by 33% vs. VPR 6.0, despite adding several new features. Further CPU time improvements remain an important future work, given the many-hour to multi-day compile time of both commercial and academic tools for the largest (500,000+ LUTs), FPGA circuits [Murray et al. 2013].

10.2. Comparison on Comprehensive Architecture without Carry Chains

We now compare the two most recent versions of VPR on a modern heterogenous architecture and circuits: the Comprehensive Architecture without carry chains described in Section 3. Carry chains cannot be used, as VPR 6.0 did not support them. This experiment was run on the same compute platform as the previous experiment. The

Table X. Comparison of VPR 6.0 and 7.0 on Comprehensive without Carry Chains Architecture

Statistic	VPR 6.0	VPR 7.0	7.0/6.0
Post-pack Nets	3848	3468	0.90
Post-pack Clusters	861	867	1.01
Placement Est. WL	50059	38484	0.77
Min W	89.7	72.7	0.81
Routing Final WL	73126	57108	0.78
High Stress Crit. Path Delay (ns)	13.31	12.56	0.94
Low Stress Crit. Path Delay (ns)	13.12	12.36	0.94
Packing CPU time (s)	308.1	11.1	0.04
Placement CPU time (s)	58.5	44.7	0.76
Routing CPU time (s)	4.6	4.4	0.96
Total CPU time (s)	383.5	64.1	0.17

front-end of the flow was performed using the versions of ODIN II and ABC in VTR 7.0 while the back-end was either VPR 6.0 or VPR 7.0. Critical path delay results were obtained by running the circuits with W_{min} + 30% tracks. All results are geometric means across the 19 heterogeneous circuits over 3 different placement seeds.

As shown in Table X, the packing CPU time in VPR 7.0 has been greatly improved (about 25× speedup) versus VPR 6.0, as is expected from Section 7. The packer in 7.0 also absorbs 10% more nets than the version 6.0 packer on this more complex architecture. With better packing quality and an improved placement move generator, the minimum channel width required is reduced by 19% on average and that leads to a 22% reduction in final routing wirelength. As for the placer, use of an incremental bounding box calculation [Betz et al. 1999] reduced placement CPU time by 24%. Overall the new VPR release has a 6× compilation speedup with better wirelength and no increase in critical path delay.

11. SOFTWARE ENGINEERING OF OPEN SOURCE PROJECT AND RELEASE

All of the software described in this article is provided under an open-source MIT license and is available for download from `http://code.google.com/p/vtr-verilog-to-routing/`.

In this release, we have improved our software engineering and release test environment. First, there is a automated build environment that rebuilds the software whenever new source code is uploaded to the active 'trunk' under development. The automated build is triggered by every new commit to the archive. Each such build is automatically tested with a small set of runs through the flow, which are checked both for completion and quality of results (QoR). The results for compile time, area, number of logic blocks and critical path delay are checked to be within range of stored "golden" results. A second, longer quality test is triggered after the first one, which uses more and larger circuits with the same types of checking. Subsequently, there is a nightly build and a weekly build, each of which also tests QoR using progressively more and larger circuits. This ability to track the QoR as the software changes is important to the stability of the flow and tools.

The following website shows the 'waterfall' of all of these builds and tests: `http://canucks.eecg.toronto.edu:8080/waterfall`.

Links in the waterfall display give the quality of results for each type of build/test in a table. We have also begun to establish uniform quality guidelines across the software submitted by all collaborators to the project. This is an effort to both keep the memory footprint and execution time to a minimum, as well as keeping the code comprehensible for new people engaged in the project. A key part of this has been the reviews of code

submitted to the archive, and the revision to meet a coding standard. Without this, the software as an extensible platform will become unmangeable.

For this release, a snapshot of the trunk code was captured and tested more extensively to form the 7.0 public release. It is this code that is exercised in the tables in Section 10 and for which extensive documentation is provided with the release distribution. The trunk version will continue to evolve as it is the code under development.

For this release, we have also decided to unify the numbering scheme of all of the component parts of VTR: ODIN, ABC, VPR, architecture files, scripts and benchmark circuits. Each one of these will have the release number 7.0 associated with them. This means that ODIN will jump many numbers ahead; and, we are attaching our own release number to ABC since we have modified the source code in several ways to make it work in our flow. The key idea is that, if users have the same version number of all three stages of the flow, then our testing indicates all the software will work together.

12. FUTURE WORK

While this article describes significant increases in the capability of the VTR toolsuite, there are still many enhancements that would increase its utility as an FPGA architecture and CAD exploration tool, including the following.

(1) *Combined Intra-Block and Inter-Block Routing*. Modern commercial FPGAs contain logic blocks with complex internal connectivity (e.g. depopulated crossbars) that require internal-to-the-logic-block routing. In VPR 7.0, this routing is performed during packing, and the final router completes only inter-block connections. Separating within-block and between-block routing in this way prevents the final router from permuting which signal connects to which input pin of a block, except for the special case of full crossbar connectivity within a block. By upgrading the final router to perform both inter- and intra-block routing in one step we believe we will improve timing and wiring quality for architectures with complex intra-block connectivity, possibly at the cost of a CPU time increase.
(2) *Area Modeling.* For complete architecture results, we require modeling of the area taken up by all elements of the heterogenous architecture. This should include not only proper transistor sizing and area estimation of the soft logic and routing by incorporating techniques like those of Chiasson and Betz [2013] but also efficient methods to estimate the area of memories and other hard blocks.
(3) *Multi-Clock Energy Analysis and Post-Routing Netlist Generation.* The energy analysis tool and the post-routing netlist generator assume there is only one clock in the design; these features should be augmented to handle the new multi-clock capability.
(4) *Improved Synthesis Optimization with Hard Blocks.* Our work on carry chain support has shown that once we make use of hard blocks (such as adders) at the ODIN II stage, we prevent any downstream logic optimization across these hard blocks in ABC. Accordingly, we will enhance ODIN II to perform key optimizations such as comon sub-expression elimination and constant propagation in a "hard block aware" way; we believe this will reduce both adder and general logic usage. We will also experiment with improvements to ABC's "white box" feature to make the functionality inside hard blocks visible to ABC, allowing optimization across the boundaries they form without destroying the optimized hard structure.
(5) *Relationally Placed Macros.* The carry chain work has added support to VPR for relationally placed macros; this feature could be expanded to enable direct user specification of such macros, which would enable experiments with floorplanning-like flows.

(6) *ODIN II language support.* We plan to continously improve the Verilog language coverage in ODIN II, including for loops, functions, casez, casex and generate statements, as well as combinational overriding and Verilog-2001 module syntax.

We also expect many new studies to be enabled by the features in this release of VTR. Some of our own future work will use this release to investigate different ways to incorporate arithmetic into FPGAs, new flows that leverage floorplanning to reduce compile time, and comparisons vs. industrial CAD tools on very large circuits. The new multi-clock support enables studies into how to most efficiently design the pre-fabricated clocking networks within an FPGA. Studies of FPGA block and routing architecture can now be performed with more complex circuits, more up-to-date process technology and with power trade-offs considered alongside area and delay. Other investigations into more radically different FPGA architectures such as 3D FPGAs [Ababei et al. 2006] or NoC-enabled FPGAs [Abdelfattah and Betz 2013] will require VTR code changes, but these investigations will also benefit from the more complete baseline VTR software.

13. CONCLUSION

This article has presented the latest release of the Verilog-To-Routing CAD flow for FPGAs. The release incorporates many enhancements, including multiple clock timing analysis and optimization, carry chain support, energy modeling and clustering improvements. We have also presented measurements showing that while this version of VTR is more feature rich, it is faster and has higher optimization quality than the prior release. The full release includes architecture files that are more representative of modern FPGAs than those typically used in academic work and which include delay, area and power modes upon which researchers can base further architecture enhancements. The VTR 7.0 release represents the combined efforts of a large number of researchers and groups around the world, and we believe it will enable the research community to pursue new directions in FPGA algorithms, architectures and applications.

ACKNOWLEDGMENTS

The authors would like to thank the numerous researchers who have helped create and enhance the VPR, ODIN II and ABC CAD tools over many years. Without their work, this new release and its enhancements would not be possible.

REFERENCES

C. Ababei, H. Mogal, and K. Barzargan. 2006. Three-dimensional place and route for FPGAs. *IEEE Trans. CAD*. 1132–1140.

M. Abdelfattah and V. Betz. 2013. The power of communication: energy-efficient NoCs for FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 1–8.

Altera. 2009. Stratix IV Device Family Overview. http://www.altera.com/literature/hb/stratix-iv/stx4_siv51001.pdf.

Altera. 2012a. *PowerPlay Early Power Estimator: User Guide.* http://www.altera.com/support/devices/estimator/pow-powerplay.jsp.

Altera. 2012b. Stratix V Device family overview. http://www.altera.com/devices/fpga/stratix-fpgas/stratix-v/stxv-index.jsp.

Altera. 2013. *Quartus II Version 13.0 Handbook.* http://www.altera.com/.

V. Betz and J. Rose. 2000. Automatic generation of FPGA routing architectures from high-level descriptions. In *Proceedings of the ACM International Symposium on FPGAs*. 175–184.

V. Betz, J. Rose, and A. Marquardt. 1999. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, Norwell, MA.

Y. Cao. 2008. Berkeley predictive technology model. http://ptm.asu.edu/.

C. Chiasson and V. Betz. 2013. COFFE: Fully-automated transistor sizing for FPGAs. In *Proceedings of the IEEE International Conference on Field-Programmable Technology*.

K. Chung and J. Rose. 1992. TEMPT: Technology mapping for the exploration of FPGA architectures with hard-wired connections. In *Proceedings of the ACM/IEEE Design Automation Conference*. 361–367.

S. Gangadharan and S. Churiwala. 2013. *Constraining Designs for Synthesis and Timing Analysis: A Practical Guide to Synopsys Design Constraints*. Springer.

J. Goeders and S. Wilton. 2012. VersaPower: Power estimation for diverse FPGA architectures. In *Proceedings of the IEEE International Conference on Field Programmable Technology*. 229–234.

H.-C. Hsieh, W. Carter, J. Ja, E. Cheung, S. Schreifels, C. Erickson, P. Freidin, L. Tinkey, and R. Kanazawa. 1990. Third-generation architecture boosts speed and density of field-programmable gate arrays. In *Proceedings of the IEEE Custom Integrated Circuits Conference*. 31.2/1–31.2/7.

M. Hutton, D. Karchmer, B. Archell, and J. Govig. 2005. Efficient static timing analysis and applications using edge masks. In *Proceedings of the ACM International Symposium on FPGAs*. 175–183.

ITRS. 2011. Interconnect chapter. http://www.itrs.net/Links/2011ITRS/2011Chapters/2011Interconnect.pdf.

P. Jamieson, K. Kent, F. Gharibian, and L. Shannon. 2010. Odin II: An open-source Verilog HDL synthesis tool for CAD research. In *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines*. 149–156.

P. Jamieson, W. Luk, S. J. Wilton, and G. A. Constantinides. 2009. An energy and power consumption analysis of FPGA routing architectures. In *Proceedings of the IEEE International Conference on Field Programmable Technology*. 324–327.

I. Kuon and J. Rose. 2008. Automated transistor sizing for fpga architecture exploration. In *Proceedings of the 45th Annual Design Automation Conference (DAC'08)*. ACM, New York, 792–795.

J. Lamoureux and S. J. Wilton. 2006. Activity estimation for field-programmable gate arrays. In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 1–8.

LATTICE. 2012. ECP3 family. http://www.latticesemi.com/products/fpga/ecp3/index.cfm.

C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings. 2011. Rapid-Smith: Do-It-yourself CAD tools for Xilinx FPGAs. In *Proceedings of the International Conference on Field-Programable Logic and Applications*. 349–355.

F. Li and L. He. 2005. Power Modeling and Characteristics of Field Programmable Gate Arrays. *IEEE Trans. CAD* 24, 11, 1712–1724.

J. Libby, A. Furrow, P. O'Brien, and K. Kent. 2011. A framework for verifying functional correctness in Odin II. In *Proceedings of the IEEE International Conference on Field-Programmable Technology*. 1–6.

J. Luu, J. Anderson, and J. Rose. 2011. Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect. In *Proceedings of the ACM International Symposium on FPGAs*. 227–236.

J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, and J. Rose. 2009. VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling. In *Proceedings of the ACM International Symposium on FPGAs*. 133–142.

Microsemi. 2013. Smartfusion2 SoC FPGAs. http://www.microsemi.com/fpga-soc/soc-fpga/smartfusion2.

A. Mishchenko. 2009. ABC: A system for sequential synthesis and verification. http://www.eecs.berkeley.edu/alanmi/abc.

K. Murray, S. Whitty, J. Luu, S. Liu, and V. Betz. 2013. Titan: Enabling large and realistic benchmarks for FPGAs. In *Proceedings of the IEEE International Conference on Field-Programmable Logic and Applications*. 1–8.

K. Nasartschuk, R. Herpers, and K. Kent. 2012. Visualization support for FPGA architecture exploration. In *Proceedings of the IEEE International Symposiumon Rapid System Prototyping*. 128–134.

K. K. W. Poon, S. J. E. Wilton, and A. Yan. 2005. A detailed power model for field-programmable gate arrays. *ACM Trans. Des. Autom. Electron. Syst.* 10, 2, 279–302.

J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson. 2012. The VTR project: Architecture and CAD for FPGAs from Verilog to routing. In *Proceedings of the ACM International Symposium on FPGAs*. 77–86.

A. Sharma, C. Ebeling, and S. Hauck. 2005. Architecture-adaptive routability-driven placement for FPGAs. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*. 427–432.

W. W.-K. Shum and J. H. Anderson. 2011. FPGA glitch power analysis and reduction. In *Proceedings of the IEEE/ACM International Symposium on Low-Power Electronics and Design*. 27–32.

A. Somerville and K. Kent. 2012. Improving memory support in the VTR flow. In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 197–202.

N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, and M. French. 2011. Torc: Towards an open-source tool flow. In *Proceedings of the ACM International Symposium on FPGAs*. 41–44.

N. Weste and D. Harris. 2010. *CMOS VLSI Design: A Circuits and Systems Perspective* 4th Ed. Addison Wesley.

H. Wong, V. Betz, and J. Rose. 2011. Comparing FPGA vs. custom CMOS and the impact on processor microarchitecture. In *Proceedings of the ACM International Symposium on FPGAs*. 5–14.

N.-S. Woo. 1995. Revisiting the cascade circuit in logic cells of lookup table based FPGAs. In *Proceedings of the ACM International Symposium on FPGAs*. 90–96.

Xilinx. 2011. *Xilinx Power Estimator User Guide.*

Xilinx. 2013a. Vivado design software. http://www.xilinx.com/.

Xilinx. 2013b. Xilinx Virtex-7 family overview. http://www.xilinx.com/products/silicon-devices/fpga/virtex-7/index.htm.

Received December 2013; revised February 2014; accepted March 2014