VPR 5.0: FPGA CAD and Architecture Exploration Tools with Single-Driver Routing, Heterogeneity and Process Scaling

JASON LUU, IAN KUON, PETER JAMIESON, TED CAMPBELL, ANDY YE, and WEI MARK FANG, University of Toronto
KENNETH KENT, University of New Brunswick
JONATHAN ROSE, University of Toronto

The VPR toolset has been widely used in FPGA architecture and CAD research, but has not evolved over the past decade. This article describes and illustrates the use of a new version of the toolset that includes four new features: first, it supports a broad range of *single-driver* routing architectures, which have superior architectural and electrical properties over the prior multidriver approach (and which is now employed in the majority of FPGAs sold). Second, it can now model, for placement and routing a heterogeneous selection of hard logic blocks. This is a key (but not final) step toward the incluion of blocks such as memory and multipliers. Third, we provide optimized *electrical* models for a wide range of architectures in different process technologies, including a range of area-delay trade-offs for each single architecture. Finally, to maintain robustness and support future development the release includes a set of regression tests for the software.

To illustrate the use of the new features, we explore several architectural issues: the FPGA area efficiency versus logic block granularity, the effect of single-driver routing, and a simple use of the heterogeneity to explore the impact of hard multipliers on wiring track count.

Categories and Subject Descriptors: B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and routing*

General Terms: Algorithms, Design, Experimentation, Reliability

Additional Key Words and Phrases: FPGA, VPR, field-programmable gate arrays, placement, routing, heterogeneous, hard blocks

ACM Reference Format:
Luu, J., Kuon, I., Jamieson, P., Campbell, T., Ye, A., Fang, W. M., Kent, K., and Rose, J. 2011. VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling. ACM Trans. Reconfig. Technol. Syst. 4, 4, Article 32 (December 2011), 23 pages.
DOI = 10.1145/2068716.2068718 http://doi.acm.org/10.1145/2068716.2068718

1. INTRODUCTION

Field-Programmable Gate Array (FPGAs) have evolved dramatically over the past ten years, as they have taken advantage of new process technologies and architectural innovations. The VPR toolset [Betz and Rose 1997; Betz et al. 1999] has been widely used as the core CAD tool in FPGA architectural exploration and CAD algorithm research over that time. However, centralized development of the tool largely stopped in the year 2000 with the release of Version 4.30. Many research efforts have modified VPR to investigate various aspects of architecture, including alternative routing architectures

Authors' addresses: J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, and W. M. Fang, University of Toronto, Toronto, Canada; K. Kent, University of New Brunswick, Canada; J. Rose (corresponding author), University of Toronto, Toronto, Canada; email: jayar@eecg.utoronto.ca.

DOI 10.1145/2068716.2068718 http://doi.acm.org/10.1145/2068716.2068718

[Lemieux et al. 2004], hard-wired corners [Wang et al. 2006], power issues [Poon et al. 2005], and modifications to logic clusters [Lemieux and Lewis 2001] among many others. VPR also formed the basis for a commercial architecture exploration environment [Lewis et al. 2003]. These efforts all produced useful research results but the modified versions of VPR are either not publicly available or lacked the architectural flexibility or robustness required in a broadly useful research tool (with the notable exception of Poon et al. [2005] which was released as a simple-to-apply patch to the VPR 4.30 release). In this article, we describe four significant new features of VPACK/VPR that are major steps towards creating a modern research environment for FPGA CAD and architecture, and illustrate their use in a number of experiments. The four new features are as follows.

(1) *Single-Driver Routing Architectures*. The most common method for programmably interconnecting FPGA routing tracks is to employ a single driver for every distinct track [Young et al. 1999; Lewis et al. 2003; Xilinx 2008b; Altera 2008], as opposed to the multiple tristate or pass transistor drivers used in previous generations of FPGAs. The driver is itself driven by a multiplexer. This type of routing architecture requires a rather different routing architecture generator, which has to optimize for different kinds of programmable connectivity constraints. VPR 5.0 supports this method of routing while continuing to provide support for the legacy methods.
(2) *Heterogeneous Logic Blocks*. Modern FPGAs all contain blocks such as multipliers [Xilinx 2008b, 2008a; Altera 2008, 2007] and memories [Xilinx 2008b, 2008a; Altera 2008, 2007] that complement the conventional Look Up Table (LUT)-based logic blocks. VPR 5.0 can now support the placement and routing of an arbitrary number of different types of blocks combined on the FPGA. This capability is the first step that is needed to permit FPGA architects to explore one of the key questions in FPGAs architecture: to determine what special, hard circuit blocks to include on the FPGA as opposed to having the same function be implemented in the LUT-based *soft* logic.
(3) *Optimized Circuit Design in released Architecture Files*. To obtain meaningful results from VPR, accurate area and delay measurements of the target FPGA are required. When using prior versions of VPR, a user was required to design her own specific transistor-level circuits to generate her own area, delay, resistance, and capacitance parameters relating to an architecture. This can be a difficult and time-consuming process. The VPR 5.0 release includes a large suite of architecture files targeting a range of area and delay trade-offs. These different soft logic architectures are optimized transistor-level designs. These files *also* span across a wide range of process technologies, currently ranging from 180 nm CMOS down to 22 nm CMOS based on the predictive technology models from Zhao and Cao [2006].
(4) *Robustness*. VPR became an effective tool in part because of its broad applicability across many architectures, but also because it was a high-quality software implementation with an easy-to-understand software architecture. It compiles without difficulty on many platforms (and was even used as part of the SPEC CPU2000 benchmark). The aim in this new version is to maintain this high level of quality, both through careful software engineering and the inclusion of a suite of regression tests that permit short-turnaround software checking, and longer-turnaround software coverage and quality of result checking.

In addition, one particular issue has arisen in the field of FPGAs architecture, due to the lack of a CAD tool that can model heterogeneous hard blocks (item number 2 given before) such as memories and multipliers: the size and scope of the benchmark circuits used are too small to represent realistic modern design. This has happened because almost all large-scale modern designs contain significant quantities of hard block memories, the modern academic architect was limited to experimenting with very small (mid-1990's size) benchmark circuits that are increasingly unrepresentative of the current use of FPGAs. This limits the ability of academic research to explore the architectural issues facing the next generations of FPGAs. Our hope is that this new version of VPR will lead to the use of larger more realistic circuits in both CAD and architecture research.

The release of VPR, Version 5.0, is publicly available at http://www.eecg.utoronto.ca/vpr. This work was initially described in Luu et al. [2009]; this article contains a new section illustrating the use of the heterogeneous features of the tool. The remainder of this article is organized as follows: Section 2 provides background on FPGA architecture and VPR itself. The new features implemented in VPR and the issues encountered are described in Section 3. In Section 4 we illustrate the capabilities of the new VPR by exploring two architectural issues enabled by the new features: LUT size with fully optimized circuit design and single-driver routing, as well as the effect of process scaling on segment track length. We also run an experiment to measure the effect of using hard multipliers in an FPGA. Finally, conclusions and future work are summarized in Section 5.

2. BACKGROUND AND TERMINOLOGY

The VPR toolset was designed to enable research in CAD and architecture for FPGAs and one of its key features is that it can perform timing-driven packing, placement, and routing (and timing analysis) for a wide range of different FPGA architectures that are described in a human-readable architecture file. In this section, we review the architectural parameters from the previous version of VPR that are needed to understand the new features we present in this article. We also review the basic VPR-based CAD flow.

2.1. FPGA Architecture Parameters

VPR [Betz and Rose 1997; Betz et al. 1999] was designed to target island-style FPGAs such as that illustrated in Figure 1. These FPGAs contain I/O blocks and logic blocks surrounded by programmable routing. As the logic blocks are all assumed to be identical, a single logic block and its adjacent routing can be combined to form a *tile* that can be replicated to create the full FPGA. The basic logic block and routing architecture parameters are reviewed in the following sections.

2.1.1. Logic Block Architecture. VPR 4.30 modelled a homogeneous array of logic blocks in a two-level hierarchy; the first level consisted of a BLE that could implement a combinational logic function and a flip-flop. The combinational logic function has commonly been a LUT, but the tool can model any function. The BLE description in VPR is parameterizable to have different numbers of inputs, **K**. The second level of the hierarchy is formed by groups of **N** BLEs, known as a logic *cluster* (also known as, among other terms, as CLBs or LABs), as shown in Figure 2. Although there are a total of $K \cdot N$ inputs and N outputs inside the cluster, the cluster only presents I inputs and N outputs to the external-to-the-cluster routing, where $I \leq K \cdot N$. VPR 4.30 assumes that all I inputs to the cluster and N outputs can be routed internally to all of the $K \cdot N$ inputs of the BLEs in what is known as a *fully-connected* cluster. Although commercial FPGAs and Xilinx [2008b], Altera [2008], and Lemieux and Lewis [2001] use clusters that are less than fully connected, both VPR 4.3 and 5.0 continue to make the fully-connected assumption. This internal-to-the-logic-cluster connectivity is known as *intracluster* routing, to distinguish it from the connections between the clusters, called *intercluster* routing, which is described shortly.

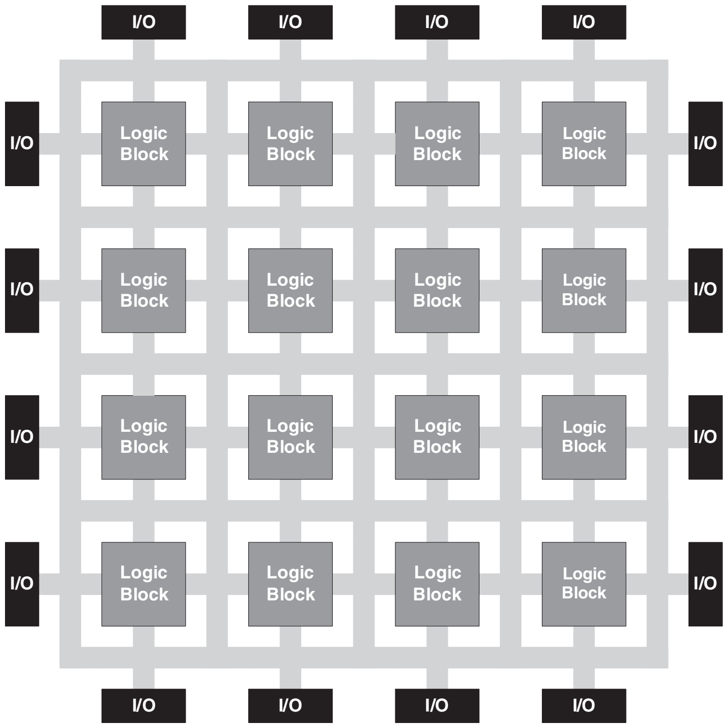

Fig. 1. Island Style FPGA.

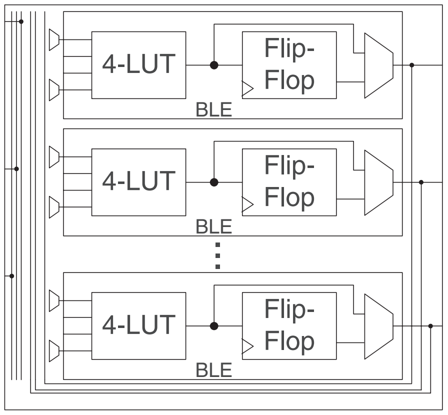

Fig. 2. Logic cluster.

2.1.2. Routing Architecture. The intercluster routing network employed in VPR consists of routing *channels* between the logic blocks as illustrated in the thick lines in Figure 1. Note that this figure gives a *logical* representation of the wiring, and that typical physical implementation of the wires and switches combine the logic block with the routing. Each channel is composed of individual routing *tracks* which consist of wires and programmable switches. The channel *width*, **W**, is the number of tracks in each channel. Programmable connections between routing tracks are made within *switch blocks* that can be found at the intersection of routing channels. The number of programmable connections incident upon a track in a switch block is typically defined as the flexibility of the switch block, and is referred to as the parameter $\mathbf{F_s}$. Connections between the clusters and routing channels are formed in connection blocks; the fraction of tracks that an input is connected to is referred to as $\mathbf{F_{c_{in}}}$, and for the outputs as $\mathbf{F_{c_{out}}}$.

A routing track typically consists of *segments* which travel a distance $\mathbf{L}$ in logic clusters before being interrupted by a programmable switch.

The routing tracks can also be programmably connected to logic blocks or I/O blocks and the topology of these connections and the connections between tracks has a significant effect on the FPGA's area and performance.

The previous VPR routing architecture generator assumes that each routing track can be driven by multiple possible sources, which we will refer to as *multidriver routing*. These multiple drivers are programmably connected to the track either through pass transistors or tristate buffers, only one of which is active during a given FPGA configuration.

2.2. Architecture File Description

In the previous version of VPR, a textual file description of the target FPGA is provided in a human-readable architecture file. That file provides all of the parameters described earlier, and several more. It also describes the delays, either using absolute delays or resistance and capacitance estimates, of the various components of the FPGA. These delays must be determined through knowledge of the IC process employed by the target FPGA, and through extensive circuit design to choose appropriate transistor sizes. While our research group had access to proprietary IC processes provided by the Canadian Microelectronics Corporation, we were unable to publish realistic architecture files with delays, resistances and capacitances without violating nondisclosure requirements. This meant that there have been no publicly available realistic architecture/electrical descriptions of VPR-based FPGAs. Furthermore, the amount of work required to design electrically realistic FPGA designs is extensive. The new release of VPR includes architecture files that have circumvented these issues.

2.3. VPR-Based CAD Flow

The experimental process used in typical VPR-based CAD and architecture research is illustrated in Figure 3. This flow takes an input circuit at the logic level and an FPGA architecture with electrical design specifications and "implements" the circuit on the specified FPGA. The timing analyzer provides a measure of the critical path delay of the implemented circuit, and area models associated with VPR provide an approximation of the area taken up by that circuit in that FPGA.

In the first step of the flow, the circuit is synthesized and technology mapped, typically using a combination of tools such as SIS [Sentovich et al. 1992], FlowMap [Cong and Ding 1994], or ABC [Mishchenko et al. 2007], all of which use BLIF as the input and output format.

The output after technology mapping must then be packed into the logic clusters available on the FPGA. This is done using T-VPack [Marquardt 1999; Betz et al. 1999], which performs a timing-driven packing. Once packing is complete, VPR is then used to perform placement and routing for the circuit. Finally, timing analysis is completed to determine the performance of the circuit on the FPGA.

A key step illustrated in the figure is called *architecture generation*, which takes the human-readable architecture file parameters, and generates the specific structures and connections for the entire FPGA. This is a difficult process that has to meet many simultaneous constraints as described in Betz et al. [1999].

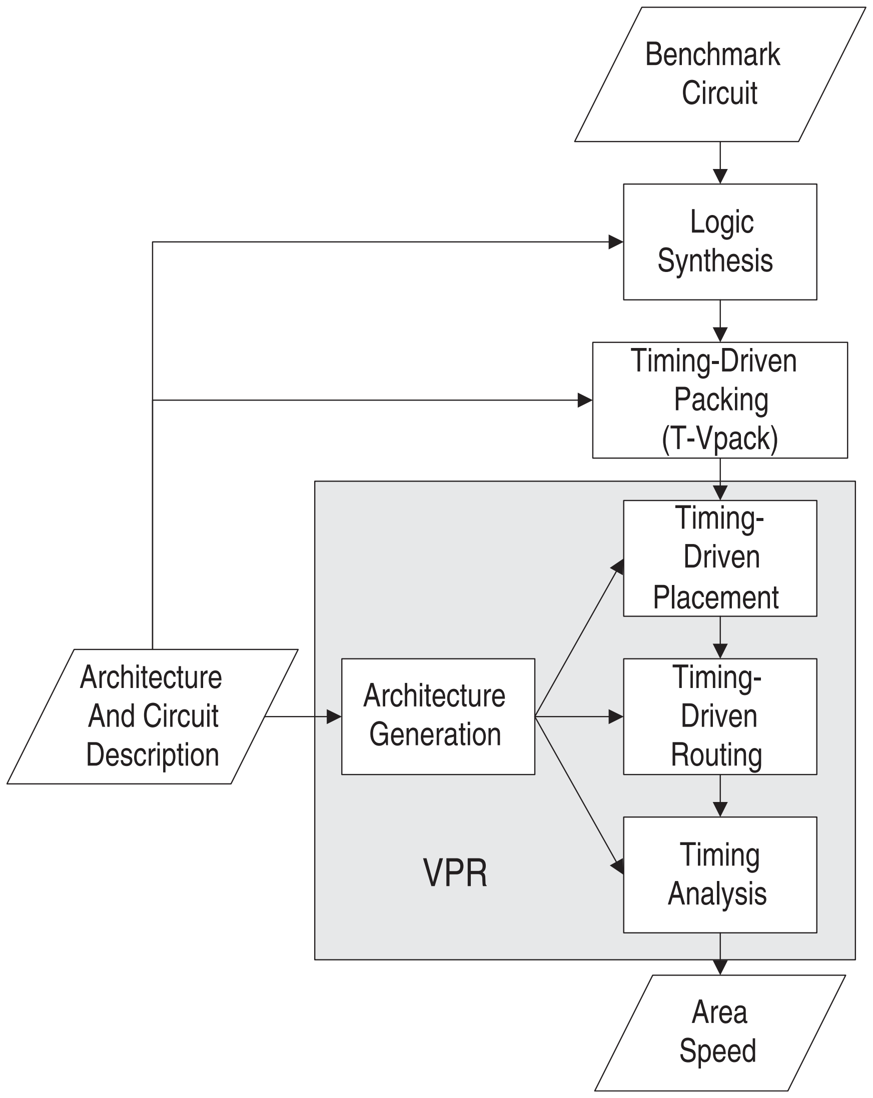

Fig. 3. VPR CAD flow.

The most difficult part of architecture generation is the creation of the *routing resource graph* which contains the precise details of every wire and switch in the architecture. The timing-driven placement algorithm uses the graph to extract timing information needed during placement, and the router directly uses the graph during routing both to determine the available connections and the delay of connections.

VPR was designed so that new ideas in routing architectures (those not presented as possibilities by the current architecture file description) can be explored by making changes only to the architecture generator without having to change the placement tool, the router, or the timing analyzer. However, the scope of the changes required for the features of this new version necessitated significant changes to the entire tool that we will describe in the following section.

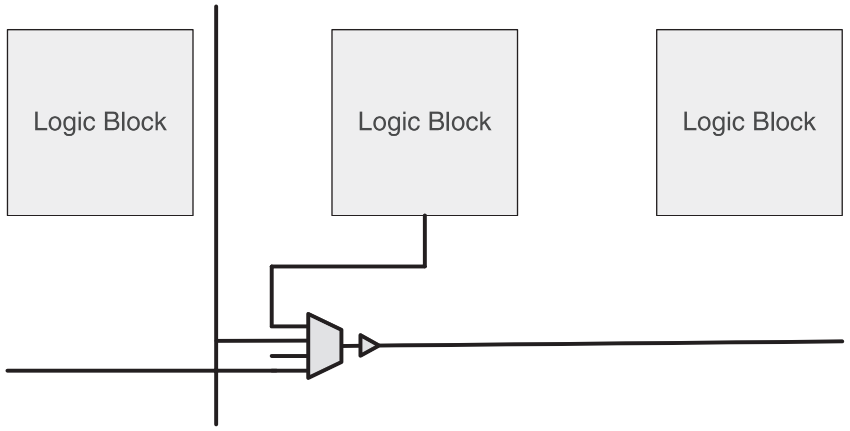

Fig. 4. Single-driver routing.

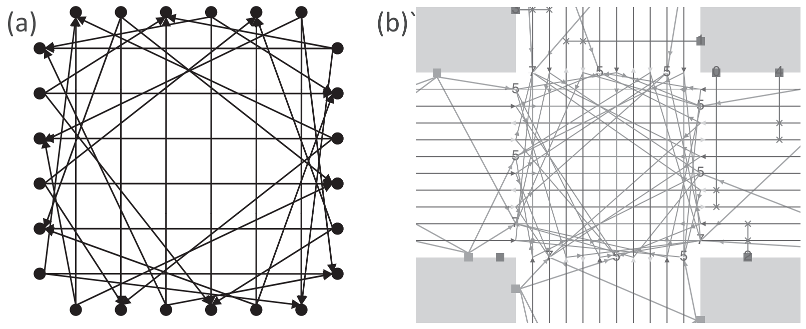

Fig. 5. Example single-driver switch block. (a) track-to-track connections of terminal points (b) complete switch block.

3. NEW FEATURES

In this section we describe the four key new features of VPR: single-driver routing, heterogeneity modeling, a wide set of electrically optimized architecture files, and a regression test suite.

3.1. Single-Driver Routing Architecture

In *single-driver routing* each routing track has only one physical driver and programmability is achieved by using a multiplexer on the input of the driver as illustrated in Figure 4. The details of the logic block and routing track connections to this multiplexer will be described later in Figure 5.

The work in Lewis et al. [2003] and Lemieux et al. [2004] has shown that the single-driver approach first employed by Young et al. [1999] *dominates* the multidriver approach, in the architectural sense, because it provides superior performance with less area. All of the FPGAs from the two largest FPGA vendors now use single-driver routing architectures exclusively [Altera 2008; Xilinx 2008b]. VPR 5.0 now supports a wide range of single-driver routing architectures (many more than described in Lemieux et al. [2004]) in addition to the multidriver routing provided by VPR 4.30.

This restriction to a single driver means that logic block outputs can only connect to the routing tracks whose drivers are adjacent to the logic block, to avoid overly long wires to neighboring logic clusters. Typical single-driver routing architectures place the driver at one end of a wire rather than, say, the middle, which would likely add unusable loading to the wire. This implies that each wire can only send signals in a specific direction (right, left, up, or down), and so there must be two kinds of tracks in each channel: left (or up)-driving and right (or down)-driving, a concept also known as *unidirectional* routing. Despite this restriction, prior work [Lewis et al. 2003; Lemieux et al. 2004] has shown that the total number of tracks required per channel is only slightly more than the number needed in bidirectional routing.

The routing architecture generator produces single-driver tracks in pairs, one in each direction. The architecture generator is faced with the following constrained graph generation problem: it must create a detailed routing architecture with W tracks per channel, where a fraction of the channel F_i should consist of tracks of length L_i (where length is measured in number of clusters traversed without switching), each track can connect to F_s other tracks, each logic cluster input pin can connect to $F_{c_{in}}$ tracks, and each cluster output can connect to $F_{c_{out}}$ tracks. Note that we have retained the ability to create tracks with different length of segments. These programmable connections must all be implemented by feeding the pins and tracks to an appropriate multiplexer connected to the single drivers. An additional issue that must be faced is that each of the multiplexers should be approximately the same size, a constraint we call the *mux balancing* criterion. We constrain the multiplexer sizes to differ by no more than 2 inputs. In addition, the start point of each wire in a track is staggered in the usual way [Betz et al. 1999]. The generation problem is somewhat more constrained than the multidriver case because there are fewer drivers for tracks and output pins to connect to at each (X,Y) grid location. All of this, combined with the goal of correctly generating an architecture for a wide range of routing architecture parameters, makes this generation problem difficult.

The generation algorithm roughly proceeds in the following way: connections from the logic block outputs to the drivers are made first, followed by the track-to-track connections. The logic block output pins connections are created in pairs (one for each direction in the unidirectional system) connecting to a starting wire of each of the two adjacent switch blocks, which drive in opposite directions. This proceeds in a round-robin fashion until either the $F_{c_{out}}$ specification is met or all the available starting wires are used by that pin. Each subsequent output pin continues the round robin from the tracks used by the previous output pin. Next, the track-to-track connections are made to the single drivers, using the Wilton switch block pattern [Wilton 1997] for any starting and ending wires. All other wires are connected to the multiplexers in a round-robin fashion.

Figure 5 illustrates a single-driver routing switch block produced by VPR 5.0, with $F_s = 3$, $F_{c_{in}} = 0.25$, and $F_{c_{out}} =$ full (where full means connect to all possible local drivers; this is typically less than all of the adjacent tracks). Part (a) of the figure shows the Wilton switch block pattern for single drivers applied to tracks that terminate at the switch block. Part (b) shows the complete switch block. Numbers within the switch block of the figure show the number of connections entering the multiplexer at those locations. The numbers on the logic block pins in the figure are labels for the input pins of the logic blocks. The arrows originating from the logic block show the output pin connections into the driver. Lines with "X" marks are input connection block connections. Lastly, arrows that originate at the switch block are track-to-track connections.

The routing algorithm used in VPR 5.0 to route a circuit on the single-driver detailed routing architecture is the same as that used in VPR 4.30, and is a timing-driven advancement on the PathFinder algorithm [McMurchie and Ebeling 1995].

As a demonstration of the capabilities of the single-driver and multidriver routing architecture generation capabilities, Figure 6 gives the single-driver and multidriver

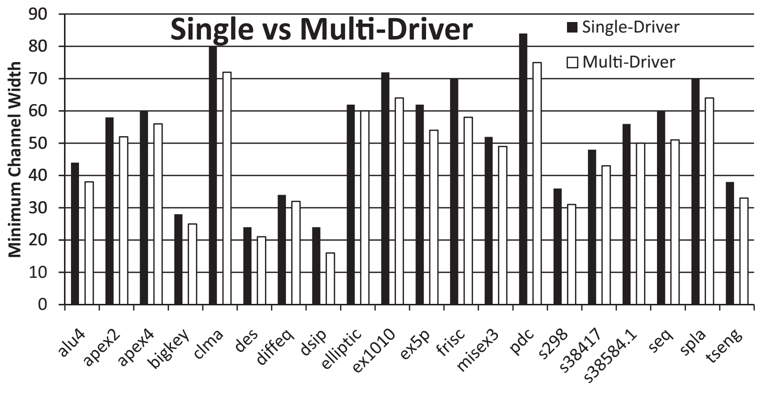

Fig. 6. Single- vs. multiple-driver track count.

minimum routable track counts (W) for a set of standard benchmark circuits implemented on an FPGA with clusters of 10 4-input LUTs, with $F_{c_{in}} = 0.25$, and $F_{c_{out}} =$ full, and segments of length L = 1. The average increase in track count of the single-driver architecture over the multiple-driver is 14%, consistent with the results published in Lewis et al. [2003]. Note that, because the area per track is lower with single-driver tracks (compared to multidriver tracks), that the net routing area is typically lower.

3.2. Heterogeneity

It is now common for there to be a heterogeneous set of logic blocks on an FPGA. In addition to the basic soft logic cluster, the additional blocks are often "hard" circuit structures that are designed to perform specific operations and, for this reason, these heterogeneous blocks are commonly called *hard blocks*. The differentiated blocks could also be a different kind of soft logic cluster that gives better performance or saves area [He and Rose 1993; Cong and Xu 1998].

Commercial FPGAs now commonly include hard memories [Altera 1995] and multipliers [Xilinx 2001]. There are many other possibilities that could be considered for inclusion as hard blocks, such as crossbars [Jamieson and Rose 2007] and floating-point multipliers [Beauchamp et al. 2006]. The specific selection of which blocks to make hard and include in an FPGA is one of the central questions in FPGA architecture. These blocks can offer significant benefits when used but, if unused, they are wasted. VPR 5.0 supports the placement and routing of monolithic heterogeneous blocks. As described later, full support of heterogeneity requires a clustering (also known as packing) tool that permits monolithic blocks to be fractured into smaller pieces. In addition, it requires a method of instantiating logical pieces of the hard block at the HDL level.

To accommodate heterogeneity, and appropriate logical coordinate system needs to be created. The X-Y coordinate system in VPR 5.0 is dictated by the soft logic cluster tile: one unit (called a "grid") in the X and Y directions is designated as the space taken up by the basic soft logic cluster. All other blocks must be multiples of this grid size. In addition, we have chosen to restrict hard blocks to be in one grid in width, so that they form a column. We further restrict a column to be composed of only one type of block. Although these restrictions prevent a more general cross-column approach, it appears sufficient for all but the extremely large hard blocks. Each hard block may be broken in a different number of subblocks, not unlike the logic elements in a cluster. Each type of block may have different timing characteristics, routing connectivity, and height. The height of a block must be an integral number of grid units. In the event that a block's height does not divide evenly into the height of the core, some grid locations are left empty.

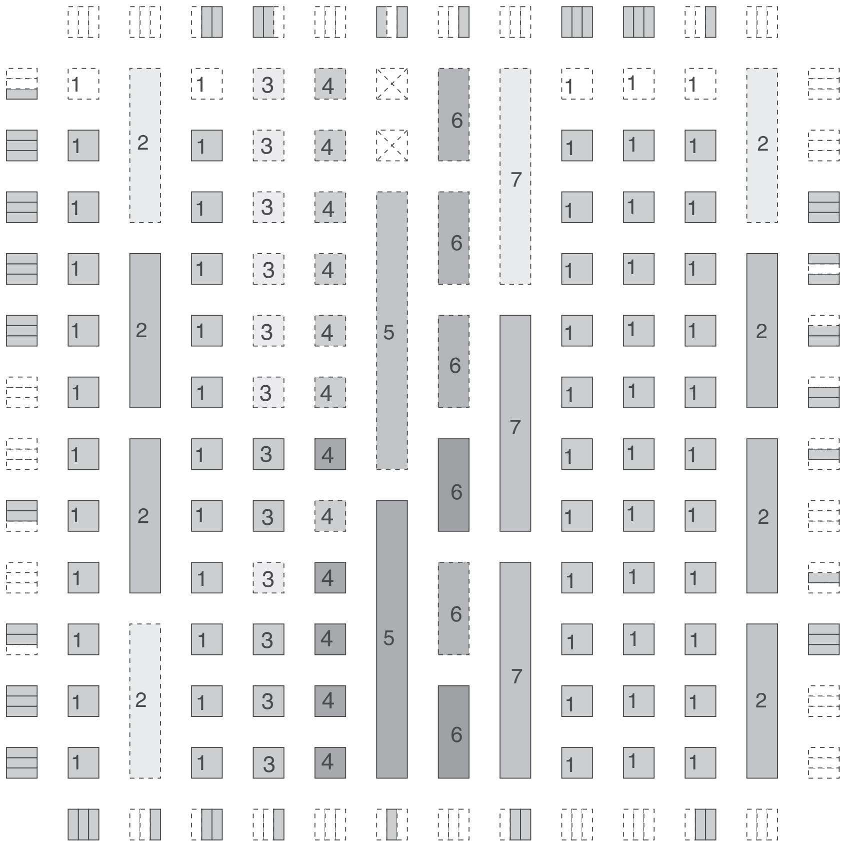

Fig. 7. A heterogeneous FPGA in VPR 5.0.

We also assume that the routing tracks crossing a multigrid height block are *transparent*; meaning that if a hard block spans multiple rows, the horizontal routing tracks pass through the hard block at every grid location. The alternative, to physically and logically interrupt the channel, would likely cause a severe disruption to the routing fabric. It is also more difficult to generate a routing architecture with such interruptions. There are no input or output pins allowed in the portion of the channel that passes through the heterogeneous block.

Note that we make a key assumption about the physical size of the embedded blocks: that they are roughly the same width as the soft logic block, so that the electrical parameters of the wires crossing through the block in this way are the same as the wires crossing the soft logic block.

VPR 5.0 models logic blocks, heterogeneous blocks, and I/Os using the same data structure. This differs from the previous version which modeled logic blocks and I/Os with separate data structures. This new way permits an arbitrary number of different types of blocks to be modeled, and actually simplified the code.

To make it easier to identify different blocks, the graphical display output of VPR employs different colors to identify the different types of blocks in the core of the FPGA for up to 6 different colors. Figure 7 illustrates (without the use of color) a heterogeneous FPGA generated by VPR 5.0 with seven different kinds of core logic blocks labeled 1 through 7.

The timing parameters of all blocks are now specified using a *timing matrix*. The timing in a subblock is modeled as a complete set of all possible delays from each input to each output of the subblock. This is an improvement on the previous version which allowed different delays for different subblocks but each subblock can only have one single delay. Heterogeneous subblocks can have purely combinational or registered output.

VPR 5.0 uses an XML-based architecture file format so as to leverage the convenient modeling hierarchy in XML [World Wide Web Consortium 2008]. All together, these additions give VPR the capability to implement circuits with hard blocks.

3.3. Electrical Optimized Architecture Files

In order to provide accurate area and delay results for an architecture, VPR requires that the delay, resistance, capacitance, and area of various circuit elements of that architecture be specified in an architecture file. When an industrial FPGA is designed, circuit designers will spend months carefully tuning and trading off aspects of the circuit design. It is clearly impractical to expend such manual effort for every potential architecture file. Furthermore, for any given *logical* architecture as specified by the list of parameters given before, there could be many different electrical designs with different area and speed trade-offs. A major feature of this new release of VPR is the inclusion of a large suite of logical architecture files, with optimized circuit design targeting a range of area and delay trade-offs for each logical architecture. The electrical design of these logical architecture files was produced by the automated transistor sizing tool and methodology described in Kuon and Rose [2008b]. In addition to this, we have leveraged the predictive technology models described in Zhao and Cao [2006] to produce architecture files describing optimized FPGAs in a wide range of IC process technologies, from 180 nm CMOS down to 22 nm CMOS. Architecture files were not released in the past since they would reveal information obtained under nondisclosure agreements but, with these predicted models, such issues are avoided.

There are three primary inputs to the automated electrical design tool. The first input is the logical architecture which describes the high-level structure of the FPGA including parameters such as cluster size, LUT size, and routing flexibility. The tool can only handle FPGAs built using cluster-based logic blocks (delays and areas for hard blocks must be estimated separately by the user). The second input must describe the process technology to determine the characteristics of the transistors that will be used to implement the architecture. Finally, an optimization objective must be supplied to determine to what extent the transistors will be sized for area or delay.

The sizing tool uses these three inputs to produce a transistor sizing specific for the given architecture and IC process, optimized with respect to the optimization objective. From the final electrical design determined by the tool, the area and delay measurements needed for the architecture file can be generated. In the past, optimizing the FPGA transistor sizes for each architecture was not feasible and, instead, fixed designs were used for a broad range of parameters. With the more thorough optimization that is possible with this new optimizer, more realistic area and delay measurements can be made which improves the quality of the final area and delay measurements from VPR. The architecture files are available for download from the VPR Web site, and can be found directly at: http://www.eecg.utoronto.ca/vpr/architectures. Table I gives a listing of the range of logical architectures, IC processes, and optimization objective[1] that can be found in that Web site.

[1]Note that in general Area^{10}Delay is used in place of pure area optimization since it improves the performance of a minimum area design with only a trivial increase in area. Typically, we found that a modest 2% area increase achieved a 4-5 $\times$ improvement in delay. This makes for more realistic designs.

Table I. Range of Parameters Included in Architecture Listing

Parameter	Range/Values Considered
Cluster Sizes	2–12
LUT Sizes	2–7
Track Length	1–8
Channel Width	Up to 3 Values
$F_{c_{in}}$	Up to 3 Values
$F_{c_{out}}$	Up to 3 Values
Cluster Inputs	Up to 3 Values
IC Processes	22 nm–180 nm
Optimization Objectives	Area^{10}Delay, AreaDelay, Delay

Table II. Regression Test Suites

Test	Test Description
checkin reg tests	Fast tests and high code coverage
arch sweep	Randomly generated architectures
mult exp	Designs with multipliers
N K sweep	Comprehensive LUT and cluster sweep
options sweep	All VPR options
QoR	Quality of results

3.4. Robustness and Regression Tests

A key goal of the new release is to maintain two key properties of robustness from the original VPR: to be able to work reliably across a wide range of logical architectures, and to be high-quality software. We created a system and suite of regression tests that gives high architecture coverage and quality coverage. We note that while many researchers have modified VPR for point experiments, it is far more difficult to keep these two properties intact. The regression tests will also aid future developers modifying VPR 5.0 by enabling them to test their implementations against known results. One major challenge with developing a regression test system is that the exact results of the algorithms used in VPR are often sensitive to small perturbations in the inputs or order of computation. This is due to the nondeterministic behavior of the placement algorithm, and the routing algorithm. A naive approach of comparing numerical metrics for equality results in many failed tests due to the small variations that occur. To overcome this problem, the regression tests specify a range, that the user may adjust, for each quantity and for each test in order to judge if a difference is due to an error or experimental variation. A second major challenge with the regression tests is the large test space to cover. The goal was to test critical points in this test space in as little time as possible. To achieve this goal, different test suites were created to allow the user to choose the coverage and runtime trade-offs to test. Table II gives an overview of the test suites available in the regression test system.

4. EXPERIMENTS USING NEW FEATURES

This new version of VPR will enable the exploration of a number of architectural issues, two of which will be examined in this section. The first will be to revisit the effect of the logic block functionality on the performance and area of an FPGA, with the use of single-driver routing and electrically optimized circuits. The second will be to explore the effect of process technology scaling on FPGAs with different routing segment lengths (L) and LUT size, again with single-driver routing.

4.1. Experimental Methodology

We will employ the standard empirical methodology in which benchmark circuits are implemented on each target FPGA (using the CAD flow described in Section 2.3) and the area and performance of each benchmark is measured.

A number of architectural parameters will be varied in the following experiments. In all cases, the number of inputs to the logic cluster (I) was set based on the LUT size (K) and cluster size (N) using the formula $I = K/2 * (N + 1)$ from Ahmed and Rose [2004]. In that paper and its predecessor [Betz and Rose 1998], this number of inputs was necessary to achieve 98% utilization of the BLEs, which was at roughly the knee of the utilization versus number of inputs curve. The routing channel width (W) will be set to be 20% larger than the minimum channel width required to route each of the benchmark circuits. For those experiments that focus only on soft logic clustered logic block architectures (Sections 4.2 and 4.3) we will use the 20 largest MCNC benchmark circuits. Each of these circuits was technology mapped by SIS with FlowMap [Cong and Ding 1994], clustered with T-VPack [Marquardt 1999], and then placed and routed with the new version of VPR. Placement and routing is repeated with ten different placement seeds and only the results from the fastest design will be used.

We have used the new architecture files which include timing and area information required to produce accurate area and performance results. For the experiments described here, we will restrict our attention to FPGAs electrically optimized for minimum area-delay product. These are the architecture files labeled under the “objective” column in the archive (described in Section 3.3). as $area \cdot delay$.

The area metric used here is the usual sum of minimum-width transistor count. This model is described in Kuon and Rose [2008a], and is based on the one described in Betz et al. [1999]; this model carefully counts every transistor (inflating the count based on the size of the transistor needed to achieve the drive) in the logic and routing of the FPGA. The critical path delay of each circuit is measured by VPR’s timing analysis engine, which is based on an Elmore delay analysis of the critical path of the circuit.

The final area and delay measurements reported for each FPGA design will be the ensemble results across all 20 benchmarks. The area is reported as the total area required for the twenty circuits and delay is reported as the geometric mean critical path delay for the twenty benchmark circuits.

4.2. Effect of LUT Size with Single-Driver Routing and Fully Optimized Circuits

In this section, we revisit (for the third time) the effect of FPGA logic block LUT size on area and performance, as was done in Ahmed and Rose [2004], but now in the context of single-driver routing and electrically optimized circuits. Although Ahmed and Rose [2004] optimized routing buffer sizes (to account for track length differences with different logic blocks) our new architecture files optimize *every* transistor, including logic and intracluster routing transistors, which the previous work did not. We look at FPGA architectures with LUT size ranging from 2 to 7 and clusters size ranging 2 to 12. For this work, single-driver length four (L = 4) routing segments were used and the target process technology was 90 nm CMOS. The methodology described earlier was used with the benchmark circuits and the resulting area measurements for each architecture are plotted in Figure 8. In this figure, each cluster size is plotted as a different curve while the LUT size is varied along the x-axis. The results for the different cluster sizes are all similar which mirrors past observations [Ahmed and Rose 2004] for multidriver routing.

As well, the same general trend seen in Ahmed and Rose [2004] of increasing area required for LUTs larger than size four can be seen in the figure. However, unlike in Ahmed and Rose [2004], the FPGA area *decreases* as the LUT size is reduced below four, whereas Ahmed and Rose [2004] has it increasing. Since area is measured as the number of clusters required multiplied by the area of these clusters (including the routing), this means that the reduced area for clusters with smaller LUT sizes more than offsets the increase in the number of clusters required as the LUT size decreases.

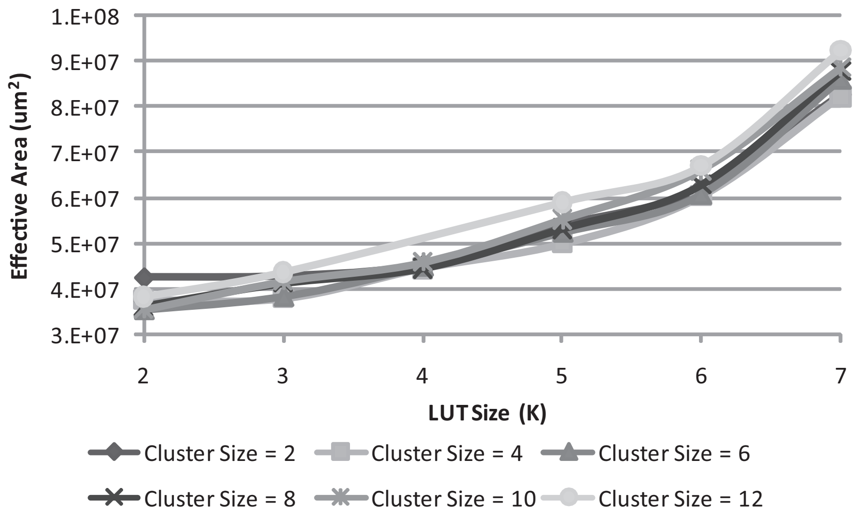

Fig. 8. Area vs. cluster and LUT size.

In the prior work, the opposite behavior was observed as the increased number of clusters required dominated the area results.

There are a number of potential reasons for these differences and the use of single-driver routing may be one contributing factor. The overall area was observed to increase for the smallest LUT sizes in past studies due to the increased area required for intercluster routing [Ahmed and Rose 2004]. We observe this increase as well but the increase is not as large. In Ahmed and Rose [2004], as the LUT size increased from 2 to 7 the total routing area decreased by over 40% but, for the same change in LUT size, we observe a decrease in routing area of only 20 to 25%. This smaller change in the routing area means the penalty of the smaller LUT sizes is not as significant and the area savings within the cluster of the smaller LUT sizes is able to compensate. Since single-driver routing reduces the area required for routing tracks, this may be one reason for the difference. However, it is also possible that the fully optimized (for area-delay product) electrical designs, which may have reduced transistor sizes for the smaller LUT sizes, also contributed to this difference.

The average delay results are plotted for these architectures in Figure 9. It can be seen that increasing the LUT size improves the performance of the FPGA but the improvements begin to diminish beyond LUTs of size four. It appears that increasing the cluster size typically offers better performance. Both these trends match the general trends seen in multidriver routing [Ahmed and Rose 2004].

The delay results can be combined with the area results and the resulting area-delay product is plotted in Figure 10. In this figure, one can see the 4-LUT architectures yield the lowest area-delay designs but, in general, the results are very similar for 3-, 4-, and 5-LUTs.

These results indicate that changes to the logic block architecture are not necessary to take full advantage of single-driver routing architecture since many of the conclusions reached in past works continue to apply. In the next section, we examine whether process technology scaling should lead to any architectural changes.

4.3. Process Technology Scaling

The increased integration enabled by shrinking process technologies enabled FPGAs to leverage the dramatic scaling of Moore's law. However, the shift to smaller technologies

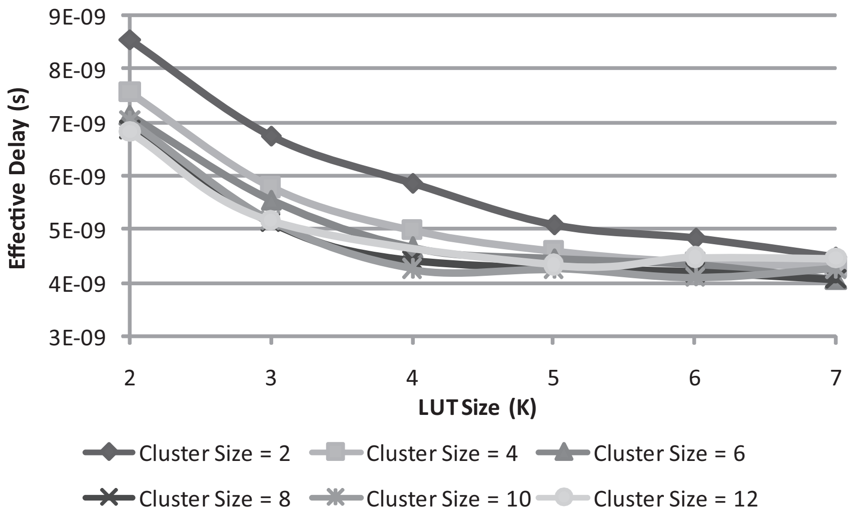

Fig. 9. Delay vs. cluster and LUT size.

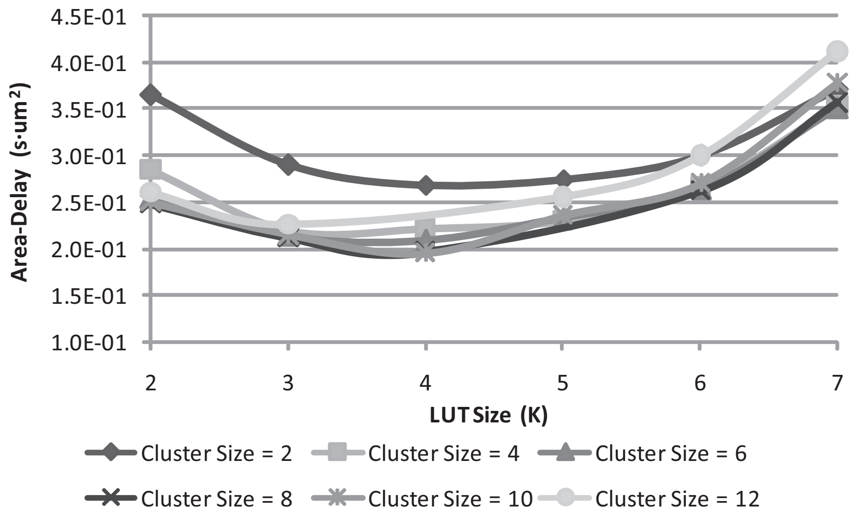

Fig. 10. Area-delay vs. cluster and LUT size.

is also accompanied by new challenges, including those related to interconnect scaling. In smaller process technologies, the delay of a constant-length wire increases. This increase is partially offset by the ability to use shorter wires, since logic also shrinks due to smaller transistors and, therefore, a shorter wire can still reach the same number of devices. However, looking at the predictions from the International Technology Roadmap for Semiconductors (ITRS), the delay of local and intermediate-length wires is expected to increase [ITRS 2007]. (This has long been recognized as a problem for global wires as their dimensions often do not shrink with each process node [Ho et al. 2001].) This has potential implications for FPGAs as the routing, including the wiring, is a significant fraction of the delay and area of an FPGA. The delay of the routing may increase as a result; alternatively, area needed for routing may increase as buffers are added and sized to ameliorate the delay issues.

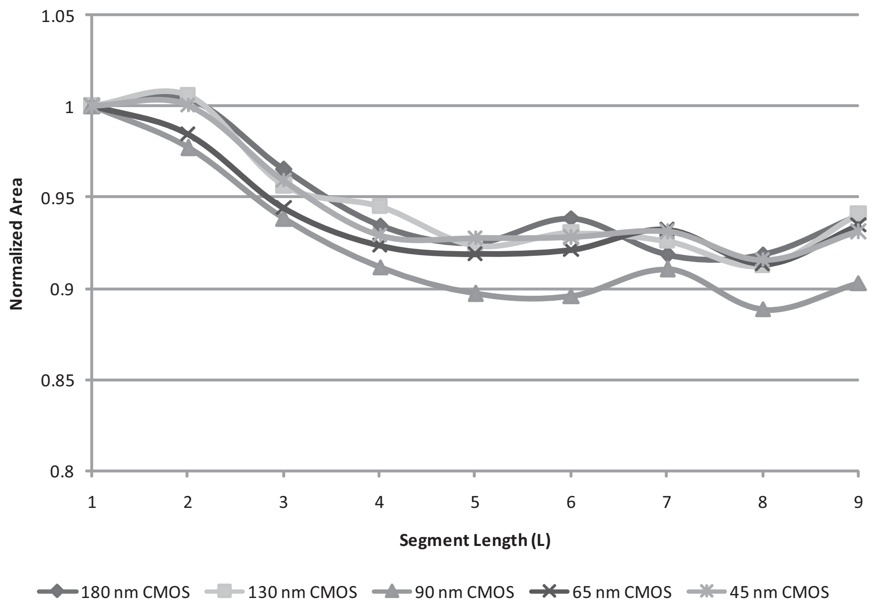

Fig. 11. Area vs. segment length/technology.

Given the potential for wiring delay to alter the delay and area of the routing, we sought to investigate the potential impact of process scaling on a number of architectural parameters. We explored the effect of new process technologies on routing segment length and LUT size. Routing segment length is interesting to consider because increased wire delays should have the greatest effect on these parameters. LUT size is worth investigating as it has the strongest influence on the cluster areas and shifts in sizing may alter the best LUT size for a logic block.

The effect on segment length was first explored by considering segment lengths of between 1 and 9 and also clusters for technologies ranging from 180 nm down to 45 nm CMOS. In all cases, 100% single-driver routing was assumed and 4-LUT logic clusters of size 10 were used for the logic blocks. When measuring delays, the wires in the inter-cluster routing and intra-cluster routing are modeled as having resistance and capacitances that are based on their physical length (which is set by the wire's logical length and the area required for the logic cluster and routing) and process technology. These wires are assumed to be implemented in an intermediate layer of metal and the properties of these wires were set based on the predictions from the ITRS [2007]. In the delay simulations, the transistors are modeled using the predictive technology models from Zhao and Cao [2006], as represented in the newly released architecture files.

The area and delay for each architecture was then determined using the standard experimental process described before. Figure 11 gives the area (normalized to the architecture with segment length 1) versus segment length with each curve showing a different process technology. One can observe that the same general trends hold in all technologies. Segments of length 1 and 2 have significant area overhead; but for longer segment lengths, the area decreases until approximately length 9 where area begins to increase again. These results are consistent with past observations made for multidriver architectures [Betz et al. 1999].

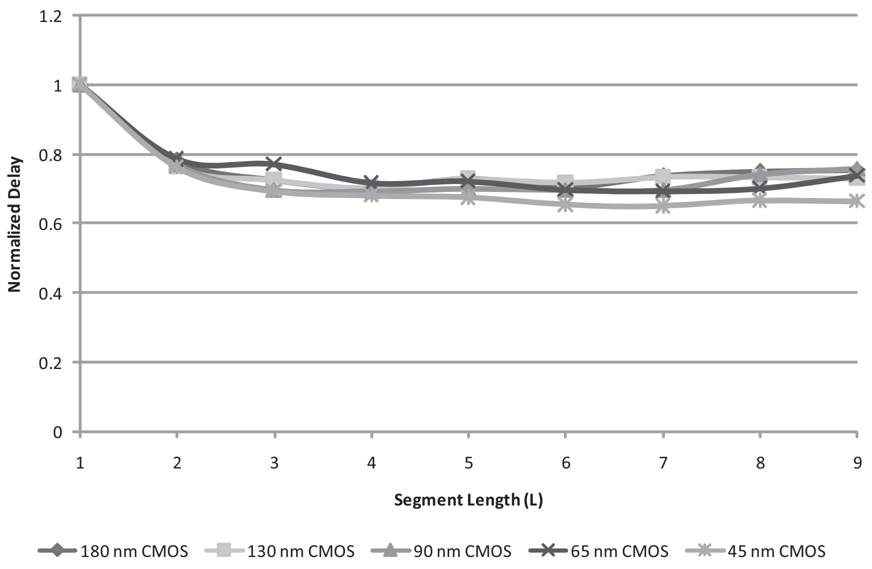

Fig. 12. Delay vs. segment length/technology.

Figure 12 gives the delay (normalized in each technology to the segment length 1 architecture) versus segment length with each curve showing a different process technology. This experiment offers finer resolution on segment length than that in Betz et al. [1999] and shows that there is a wide region in which segment length does not affect FPGA speed when segments are greater than length three. Most interestingly, the results do not appear to be significantly affected by the different process technologies.

The impact of process scaling on LUT size was also examined. For this investigation, single-driver length 4 routing segments were assumed and logic clusters of size 10 were used. LUTs ranging in size from 2 to 7 were examined in technologies ranging from 180 nm to 45 nm CMOS using the predictive technology models [Zhao and Cao 2006]. The area and delay results for these architectures are plotted in Figures 13 and 14 respectively. In both figures, the different processes are plotted as different curves as indicated in the legend. For each technology, the results are normalized to the area or delay of the 2-LUT architecture.

The area results are all consistent between the technologies with increasing LUT size leading to increased area. Similarly, for the delay, the trends are similar for all technologies and the delay improves with increased LUT size until 6-LUTs beyond which there is only a minimal change to the FPGA's performance. Clearly, it appears that despite the challenges of new process technologies, their effect on architectural parameters is minimal. It is likely, however, that if other issues such as power consumption are considered then architectural adjustments may be needed.

These results appear to run counter to the trend observed in commercial high-performance FPGAs which have made use of increased LUT sizes in recent technologies. However, as described previously, the results summarized in this section are for designs optimized to minimize the area-delay product of each design. High-performance FPGAs are likely more aggressive in obtaining delay improvements while lower-cost FPGAs families such as the Cyclone [Altera 2007] and Spartan [Xilinx 2008a] families may be have optimization objectives more similar to ours. These lower-cost FPGAs made consistent use of 4-LUTs for many generations which is the same trend that would be expected based on these results.

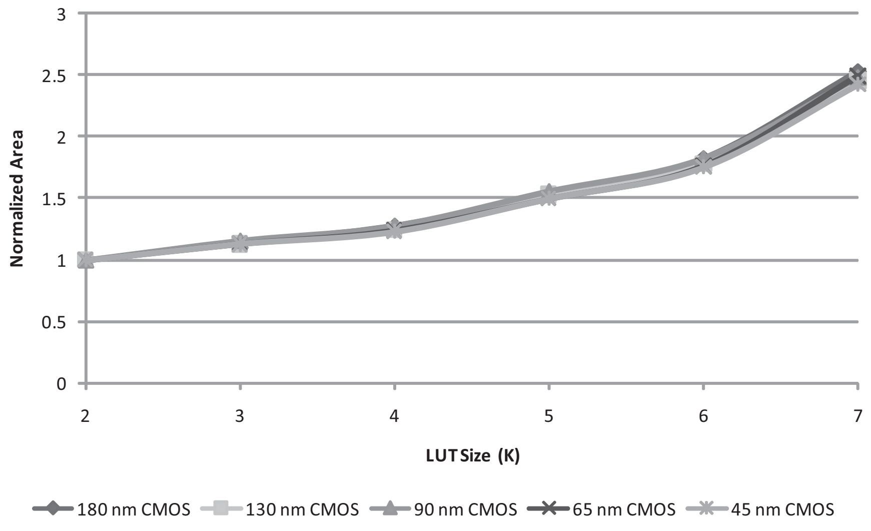

Fig. 13. Area vs. LUT size/technology.

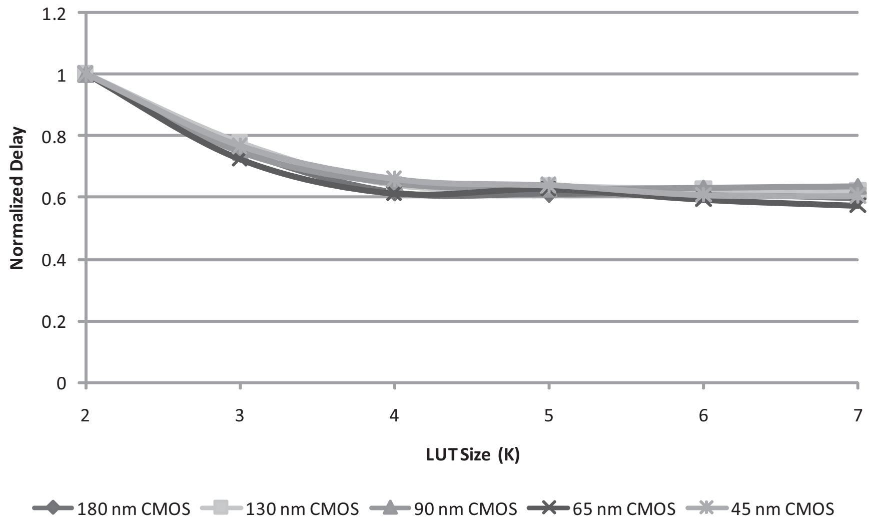

Fig. 14. Delay vs. LUT size/technology.

4.4. Illustration of Heterogeneity: Implementing Multipliers in Hard vs. Soft Logic

The new heterogeneity/hard block feature of VPR is perhaps one of the most important as it is the first step towards enabling the exploration of a much broader range of architectures than previously. Note that this is only the first step in the infrastructure required to really support heterogeneity; VPR 5.0 supports the placement, routing, and routing architecture generation of a monolithic heterogeneous block. To be useful, the block should be fracturable in the manner that large multipliers on modern FPGAs can be useful as a set of small multipliers. These smaller pieces must be packed or clustered into the larger monolith, necessitating a more general packing tool, which is the subject of future work. Also, an HDL language elaborator must provide the ability to instantiate the logical entities that can be packed into the hard block, as well as

Table III. Multiplier Circuits

Circuit	Number of Logical Multipliers	Smallest Multiplier	Largest Multiplier
cf_fir_3_8_8	4	8 × 8	8 × 8
iir	5	8 × 16	16 × 16
iir1	5	7 × 7	7 × 11
paj_raygentop_hierarchy_no_mem	18	8 × 7	16 × 16
rs_decoder_1	13	5 × 5	5 × 5
rs_decoder_2	9	9 × 9	9 × 9
sv_chip1_hierarchy_no_mem	152	8 × 8	8 × 8
sv_chip2_hierarchy_no_mem	564	8 × 3	16 × 8

breaking up logical entities that are too large to fit in one monolith. To enable the illustration of the monolithic placement and routing capability in this article, we used a new tool under development, ODIN II, as a Verilog front-end capable of instantiating multipliers. ODIN II is a new version of the ODIN Verilog elaborator tool [Jamieson and Rose 2005]. In this section we compared an FPGA with hard 18 × 18 multipliers to one that only contains pure soft logic.

In this experiment, we use 8 circuits that contain multipliers, as listed in Table III. The table gives the name of each circuit, the raw number of *logical multipliers* (a logical multiplier is a multiplier specified by the circuit, which can be of any size, as opposed to a *physical* multiplier which refers to the set of fixed sizes of multiplier available on the FPGA), and then gives the smallest and largest size of the multipliers in each circuit. While most of the circuits have just a few different multiplier sizes, the circuit sv_chip2_hierarchy_no_mem has many different sizes across the range given in the table.

Each circuit was synthesized by ODIN II in two ways: first, ODIN II was set to generate hard multipliers for each logical multiplier present in the circuit (for multipliers larger than 3 × 3 only; smaller multipliers are more efficiently implemented as soft multipliers). In all cases these hard multipliers use a single 18 × 18 multiplier that we have chosen to architect into one hypothetical FPGA. (Again, we do not yet have the ability to fracture the hard multiplier into smaller multipliers). The second method was to set ODIN II to generate all multipliers to be implemented as soft multipliers in the LUT-based logic.

The hard multiplier circuits were then synthesized through ABC (with the multipliers treated as black boxes) and then clustered (the soft logic only) with T-Vpack, and then placed and routed with VPR. The hard multiplier FPGA architecture specification is as follows: it employed a soft logic cluster consisting of 10 4-input lookup tables, with 22 inputs and 10 outputs. The routing architecture was of the single-driver type, with segments of length 4, Fcin = 0.25, and Fcout = 0.25 (for the both soft logic clusters and the hard multiplier blocks).

The 18 × 18 multiplier was set to span two rows in height, in order to match the pin demand of the hard block to the soft logic cluster; the multiplier has 36 inputs and 36 outputs which is roughly equivalent to the pin demand of two soft logic clusters, each of which has 22 total inputs and 10 outputs.

The soft multiplier-only circuits were also synthesized through ABC, T-VPACK, and VPR as well. The soft logic architecture was the same as the one used in the hard-multiplier FPGA.

The resulting minimum number of tracks, W, required to route the circuits was determined by setting VPR into a binary search mode, which continuously attempts to route the circuit with different numbers of tracks until it finds the smallest number that succeeds.

Table IV. Hard vs. Soft Multiplier: Logic Counts

Circuit	Number of Logical Multipliers	Soft Logic w Hard Mults	Soft Logic w/o Hard Mults	Soft/Hard Ratio
cf_fir_3_8_8	4	23	81	3.52
iir	5	43	303	7.05
iir1	5	56	126	2.25
paj_raygentop_hierarchy_no_mem	18	302	1020	3.38
rs_decoder_1	13	129	154	1.19
rs_decoder_2	9	274	340	1.24
sv_chip1_hierarchy_no_mem	152	1544	3862	2.50
sv_chip2_hierarchy_no_mem	564	4292	6224	1.45
geometric average				2.4

Table V. Hard vs. Soft Multiplier: Track Counts

Circuit	Number of Logical Multipliers	Track Count w Hard Mults	Track Count w/o Hard Mults	Soft/Hard Ratio
cf_fir_3_8_8	4	34	36	1.06
iir	5	42	42	1.00
iir1	5	28	36	1.29
paj_raygentop_hierarchy_no_mem	18	44	52	1.19
rs_decoder_1	13	36	36	1.00
rs_decoder_2	9	58	60	1.03
sv_chip1_hierarchy_no_mem	152	68	82	1.21
sv_chip2_hierarchy_no_mem	564	100	74	0.74
geometric average				1.05

Table IV gives the results of the experiment in terms of the number of logic and hard blocks used. The first column gives the name of the circuit, the second column lists the number of logical multipliers, the third gives the number of clusters (of 10 4 LUTs) required to implement the circuit in an FPGA with hard 18×18 multipliers, and then the number of clusters when there are no hard multipliers. The last column gives the ratio of these two cluster counts (the number without hard multipliers divided by the number with hard multipliers). Table IV clearly shows the advantage of the hard multipliers because the number of soft logic clusters required to implement each circuit dramatically increases the total number of clusters in the circuit, from a low of 1.52 times to more than 7 times. Clearly, large multipliers have the most effect here.

Table V gives the results of the experiment with respect to track count. The first column gives the name of the circuit and the second column lists the number of logical multipliers, as earlier. The third column gives the number of tracks required to route the circuits with hard multipliers, and the fourth is the number of tracks without the hard multipliers. The data suggests, with one exception, that the track count in the FPGA without hard multipliers is the same or larger, up to 30% larger. However the exception is also the largest circuit with the most multipliers. We looked more closely at the exception, the circuit sv_chip2_hierarchy_no_mem. It has a large number of multiplications by constants, which, when implemented in soft logic cause much of the soft logic to be eliminated through (presumably) constant propagation. This can also be observed in Table IV: despite the large number of multipliers, the soft logic version of sv_chip2_hierarchy_no_mem is only 45% larger than the hard logic version.

There can be a number of forces driving the tracks counts in different directions, as discussed in Smith et al. [2009]. First, if the multipliers are only a small portion of the circuit, then they are unlikely to cause a large difference on the track count, accounting for those circuits that end up with the same track count. If the implementation of the soft multipliers spreads out the circuit a significant amount, this will increase the average wire length which will in turn increase the track count. It is also possible, in the case of the hard multiplier FPGA, that if there are many small multipliers then this will lower the average pin demand emanating from a grid point, and thereby will also lower the track count. While Smith et al. [2009] provides analytic expressions for these effects, they are only approximate, and it will likely be necessary to run experiments to determine the dominant effect for a given set of circuits.

5. CONCLUSIONS AND FUTURE WORK

This article has described four new features of the VPR tool suite for FPGA CAD and architecture research. The first two are the ability to model single-driver routing architectures, and heterogeneous logic structures. Third, we have developed and provided electrically optimized circuit and architecture files that permit exploration across a far wider range of area-delay trade-offs and IC processes. Finally we have provided software regression tests to enhance the robustness of development in the future. We have illustrated the new tool's use by performing a number of experiments showing each of these features.

VPR 5.0 and the associated architecture files can be downloaded at http://www.eecg.utoronto.ca/vpr.

5.1. Future Work

While this work is an important step in the development of publicly available Computer-Aided Design (CAD) tools that can support modern FPGA architectures, there is much work that remains. A significant limitation currently is the lack of support for heterogeneous blocks in the upstream tools in the CAD flow, from Register Transfer Level (RTL) design to packing. Our research group is currently working on such an upstream tool (now called ODIN II) which will use the *same* architecture file as VPR to permit the centralized description of hard block structures. The synthesis, technology mapping, and clustering support for hard blocks has been limited to treating any hard functionality as black boxes. A new version of ABC [Jang et al. 2009], which should be integrated into this flow, now includes a powerful *white box* capability that will markedly enhance the synthesis of hard circuits.

Future improved upstream tools must be flexible enough to permit a wide range of heterogeneous blocks; we plan to evolve a flow in which all tools connect to the same architecture description file for consistency of specification. As well, given the close relationship between clustering and placement, we plan to integrate the currently separate tools that perform these tasks (T-VPack and VPR). It will also be important to include power measurement and optimization capabilities, such as those provided by Poon et al. [2005] in previous versions. Finally, additional work is needed to add support for features such as arithmetic carry-chains and depopulation of the intra-cluster routing that are now common in commercial FPGAs. The challenge is to add this functionality with sufficient flexibility to enable architectural exploration of these features.

ACKNOWLEDGMENTS

This work has benefited from the contributions of many others. Daniele Paladino [Paladino 2008], Russ Tessier, Andrew Ling and Mark Jarvin, and Steve Wilton were involved in the development of versions of VPR that supported heterogeneous blocks and their experiences provided guidance for the implementation used in VPR 5.0. The authors are also grateful to Vaughn Betz, the original author of VPR.

REFERENCES

AHMED, E. AND ROSE, J. 2004. The effect of LUT and cluster size on deep-submicron FPGA performance and density. *IEEE Trans. VLSI Syst. 12,* 3, 288–298.

Altera. 1995. Datasheet: Flex 10k embedded programmable logic family. http://web.mit.edu/6.111/www/s2004/LABS/dsflok.pdf.

Altera. 2007. Cyclone III device handbook. ver. CIII5V1-1.2 http://www.altera.com/literature/hb/cyc3/cyclone3_handbook.pdf.

Altera. 2008. Stratix IV device handbook version SIV5V1-1.1. http://www.altera.com/literature/hb/stratix-iv/stratix4_handbook.pdf.

Beauchamp, M. J., Hauck, S., Underwood, K. D., and Hammert, K. S. 2006. Embedded floating-point units in FPGAs. In *Proceedings of the 14th International ACM/SIGDA Symposium on Field Programmable Gate Arrays*. ACM, New York, 12–20.

Betz, V. and Rose, J. 1997. VPR: A new packing, placement and routing tool for FPGA research. In *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*.

Betz, V. and Rose, J. 1998. How much logic should go in an FPGA logic block? *IEEE Des. Test Mag. 15,* 1, 10–15.

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

Cong, J. and Ding, Y. 1994. FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 13,* 1, 1–12.

Cong, J. and Xu, S. 1998. Delay-Optimal technology mapping for FPGAs with heterogeneous LUTs. In *Proceedings of the Design Automation Conference*. 704–707.

He, J. and Rose, J. 1993. Advantages of heterogeneous logic block architectures for FPGAs. In *Proceedings of the IEEE Custom Integrated Circuits Conference*. 7.4.1–7.4.5.

Ho, R., Mai, K., and Horowitz, M. 2001. The future of wires. *Proc. IEEE 89,* 4, 490–504.

ITRS. 2007. International Technology Roadmap for Semiconductors 2007 Ed. http://www.itrs.net/reports.html.

Jamieson, P. and Rose, J. 2005. A verilog RTL synthesis tool for heterogeneous FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 305–310.

Jamieson, P. and Rose, J. 2007. Architecting hard crossbars on FPGAs and increasing their area efficiency with shadow clusters. In *Proceedings of the International Conference on Field-Programmable Technology*. 57–64.

Jang, S., Wu, D., Jarvin, M., Chan, B., Chung, K., Mishchenko, A., and Brayton, R. 2009. Smartopt: An industrial strength framework for logic synthesis. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'09)*. ACM, New York, 237–240.

Kuon, I. and Rose, J. 2008a. Area and delay trade-offs in the circuit and architecture design of FPGAs. In *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays (FPGA'08)*. ACM, New York, 149–158.

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

Lemieux, G., Lee, E., Tom, M., and Yu, A. 2004. Directional and single-driver wires in FPGA interconnect. In *Proceedings of the IEEE International Conference on Field-Programmable Technology*. 41–48.

Lemieux, G. and Lewis, D. 2001. Using sparse crossbars within LUT clusters. In *Proceedings of the ACM/SIGDA 9th International Symposium on Field Programmable Gate Arrays (FPGA'01)*. ACM, New York, 59–68.

Lewis, D. et al. 2003. The Stratix^{TM} routing and logic architecture. In *Proceedings of the ACM/SIGDA 11th International Symposium on Field Programmable Gate Arrays (FPGA'03)*. ACM Press, 12–20.

Luu, J., Kuon, I., Jamieson, P., Campbell, T., Ye, A., Fang, W. M., and Rose, J. 2009. Vpr 5.0: Fpga cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'09)*. ACM, New York, 133–142.

Marquardt, A. R. 1999. Cluster-Based architecture, timing-driven packing and timing-driven placement for FPGAs. M.S. thesis, University of Toronto.

McMurchie, L. and Ebeling, C. 1995. Pathfinder: A negotiation-based performance-driven router for FPGAs. In *Proceedings of the ACM SIGDA International Symposium on Field Program Mable Gate Arrays (FPGA)*. 111–117.

Mishchenko, A., Chatterjee, S., and Brayton, R. K. 2007. Improvements to technology mapping for LUT-based FPGAs. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 26,* 2, 240–253.

Paladino, D. 2008. Academic clustering and placement tools for modern field-programmable gate array architectures. M.S. thesis, University of Toronto. https://tspace.library.utoronto.ca/handle/1807/11159.

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

Sentovich, E. M. et al. 1992. SIS: A system for sequential circuit synthesis. Tech. rep. UCB/ERL M92/41, Electronics Research Lab, University of California, Berkeley, CA.

Smith, A. M., Wilton, S. J., and Das, J. 2009. Wirelength modeling for homogeneous and heterogeneous fpga architectural development. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'09)*. ACM, New York, 181–190.

Wang, G., Sivaswamy, S., Ababei, C., Bazargan, K., Kastner, R., and Bozorgzadeh, E. 2006. Statistical analysis and design of HARP routing pattern FPGAs. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 25,* 10, 2088–2102.

Wilton, S. J. E. 1997. Architectures and algorithms for field-programmable gate arrays with embedded memories. Ph.D. thesis, University of Toronto.

World Wide Web Consortium. 2008. Extensible markup language (xml). http://www.w3.org/XML/.

Xilinx. 2001. Datasheet: Virtex 2.5 v field programmable gate arrays. http://www.gb.nrao.edu/gbt/MC/GBTprojects/pulsarSupport/PulsarSplgotCard/FPGA.pdf.

Xilinx. 2008a. Spartan-3A FPGA family: Data sheet. Ver. 1.0 http://www.xilinx.com/support/documentation/data_sheets/ds529.pdf.

Xilinx. 2008b. Virtex-5 user guide. UG190 (v4.0) http://www.xilinx.com/support/documentation/user_guides/ug190.pdf.

Young, S. P., Bauer, T. J., Chaudhary, K., and Krishnamurthy, S. 1999. FPGA repeatable interconnect structure with bidirectional and unidirectional interconnect lines. US Patent 5,942,913.

Zhao, W. and Cao, Y. 2006. New generation of predictive technology model for sub-45 nm early design exploration. *IEEE Trans. Electron. Dev. 53,* 11, 2816–2823. (Transistor models downloaded from http://www.eas.asu.edu/ ptm/.)

Received May 2009; revised August 2009; accepted September 2010