Raising FPGA Logic Density Through Synthesis-Inspired Architecture

Jason H. Anderson, *Member, IEEE,* Qiang Wang, *Member, IEEE,* and Chirag Ravishankar, *Student Member, IEEE*

***Abstract*—We leverage properties of the logic synthesis netlist to define both a new FPGA logic element (function generator) architecture and an associated technology mapping algorithm that together provide improved logic density. We demonstrate that an "extended" logic element with slightly modified K-input LUTs achieves much of the benefit of an architecture with K+1-input LUTs, while consuming silicon area close to a K-LUT (a K-LUT requires half the area of a K+1-LUT). We introduce the notion of "non-inverting paths" in a circuit's AND-inverter graph (AIG) and show their utility in mapping into the proposed logic element architectures. We propose a general family of logic element architectures, and present results showing that they offer a variety of area/performance trade-offs. One of our key results demonstrates that while circuits mapped to a traditional 5-LUT architecture need 15% more LUTs and have 14% more depth than a 6-LUT architecture, our extended 5-LUT architecture requires only 7% more LUTs and 5% more depth than 6-LUTs, on average. Nearly all of the depth reduction associated with moving from K-input to K+1-input LUTs can be achieved with considerably less area using extended K-LUTs. We further show that 6-LUT optimal mapping depths can be achieved with a small fraction of the LUTs in hardware being 6-LUTs and the remainder being extended 5-LUTs, suggesting that a heterogeneous logic block architecture may prove to be advantageous.**

***Index Terms*—Field-programmable gate arrays, FPGAs, architecture, logic synthesis, area, optimization.**

I. Introduction

FOR over twenty years, the logic blocks (function generators) in field-programmable gate arrays (FPGAs) from the two main commercial vendors have been based primarily on look-up-tables (LUTs), registers and carry logic. During the same time period, FPGA fabrication technology has scaled to the present 40/45 nm, hard IP blocks have been incorporated, and considerable innovations have appeared in FPGA routing architecture. Relatively little change has been made to the composition of core logic blocks in FPGAs, aside from the shift toward larger LUTs. We speculate that a reason for this may be a lack of research focus on synthesis techniques for easy targeting and evaluation of non-LUT-based logic block architectures. In this paper, we consider FPGA logic block architecture and propose logic elements with superior area-efficiency, as well as a simple mapping strategy for the proposed logic elements.

The tight interaction between architecture and computer-aided design (CAD) for FPGAs is well-established. The approach taken in most FPGA architecture research is hardware-driven in the sense that the hardware "idea" comes first into the architect's mind, and CAD tools are subsequently developed to target and gauge the benefit of the proposed hardware. A classic work by Rose et al. studied what LUT size provides the best area-efficiency in FPGAs [1]. Here, we re-visit the question of area-efficiency, however in our case, architecture research is turned upside down and the reverse approach is taken: the CAD tools themselves suggest a particularly natural architecture.

Raising logic density is the goal of this research and is one which we believe to be well-motivated. There has recently been a trend toward larger LUTs in commercial FPGAs. The LUTs in the Xilinx Virtex-5 FPGA and the Altera Stratix-III FPGA can realize 6-input logic functions [2], [3]. However, since customer designs contain only a limited number of large functions, LUTs in commercial chips are multi-output and *fracturable* into several smaller LUTs. For example, the 6-LUT in Virtex-5 can implement any 6-input function, or any two functions that together use up to 5 distinct inputs. LUTs in Stratix-III offer even more fracture-flexibility and can implement two independent 4-LUTs. From the vendor perspective, while the delay benefits of 6-LUTs are desireable, care has been taken to mitigate under-utilization and achieve high logic density [4]. In this paper, we re-examine the LUT structure and challenge the conventional wisdom that full K-input LUTs are necessary to implement K-variable logic functions in FPGA logic blocks. We show that a smaller K-input logic element that uses fewer transistors can be used in place of a K-LUT, with relatively little impact to circuit speed. Logic density is improved through the use of the proposed logic element. In this paper, logic density refers to amount of silicon area (based on transistor count) required to implement a given circuit.

We propose a new technology mapping approach and FPGA logic element architecture, both of which are motivated by properties of the logic synthesis netlist. Our mapping approach and architecture are relatively small variations on published techniques. Specifically, we use properties of logic synthesis netlist to identify "gating" inputs to LUTs, where a gating input is one that has a particular logic state (logic-0 or logic-1) that forces the LUT output to evaluate to a particular logic state (logic-0 or logic-1). We give a simple scheme for finding such gating inputs and show that they occur frequently in circuits.

J. Anderson is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4 Canada (e-mail: janders@eecg.toronto.edu).

Q. Wang is with Huawei Technologies (U.S.A.), Santa Clara, CA 95050 USA (email: Qiang.sc.Wang@huawei.com).

C. Ravishankar is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1 Canada (e-mail: cravisha@uwaterloo.ca).

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada.

We leverage the gating input concept in the design of new logic element architectures that offer considerably better logic density compared with those in today's commercial FPGAs. Our combined "architecture + CAD approach" to attack logic density is an entirely new direction and is in contrast to recent CAD work which used don't-cares to reduce LUT count in mapped circuit implementations [5].

A preliminary version of a portion of this article appeared in [6]. In this extended journal version, we generalize the mapping approach to discover additional gating input signals to LUTs. In particular, we offer an approach to identify inputs that cause the LUT function to evaluate to logic-1, in addition to the logic-0 case described in the conference version. The extended LUT architecture is altered accordingly to handle the scenario where a gating input forces a function to logic-1. We propose a generalized family of extended LUT architectures, each offering a different area/performance trade-off. Finally, we present more comprehensive experimental results, including results for an additional set of benchmark circuits.

The remainder of the paper is organized as follows: Background and related work appear in Section II. Our mapping approach and the proposed logic element architecture are described in Section III. An experimental study is presented in Section IV. Conclusions and suggestions for future work are offered in Section V.

II. Background

A. LUT Hardware Architecture

A K-input LUT (K-LUT) is a hardware implementation of a truth table that can implement *any* logic function that requires up to K variables. Central to our work is the property that the required silicon area to implement a LUT increases exponentially with the number of LUT inputs. Fig. 1 shows the hardware for 2 and 3-input LUTs. 3-input LUTs have 8 SRAM cells to hold the truth table of the logic function implemented by the LUT, and require a tree of seven 2-to-1 multiplexers. 2-LUTs have 4 SRAM cells and require three 2-to-1 multiplexers. In general, K-LUTs have 2^K SRAM cells and $2^K - 1$ multiplexers. 6-LUTs in today's commercial FPGAs have 64 SRAM cells. Adding additional inputs to a LUT is clearly a costly endeavor, and the thrust of our work is an approach for "getting away with" using smaller LUTs, while at the same time realizing the benefits of larger LUTs.

B. FPGA Technology Mapping

Research on technology mapping for FPGAs was active in the early 1990s with a wide range of algorithms proposed (e.g. [7], [8]). In comparison with technology mapping for ASIC standard cells, the FPGA mapping problem is simplified as a consequence of the target gate being a K-LUT that can implement any K-variable function. FPGA mappers need not focus on finding logic functions in circuits that match with gate functions in a target library; rather, FPGA mappers must cover a circuit with K-variable functions (*any* K-variable functions). Recent FPGA technology mappers are based on the notion of cuts [9], [10], which we review here. The combinational portion of a circuit can be represented by a directed acyclic graph (DAG) $G(V, E)$, where each node, $z \in V$, represents a single-output logic function and edges between nodes, $e \in E$, represent input/output dependencies among the corresponding logic functions. For a node z in the circuit DAG, let $Inputs(z)$ represent the set of nodes that are fanins of z.

Fig. 2(a) illustrates the notion of a cut for a node z. A cut for z is a partition, $(V, \overline{V})$, of the nodes in the subgraph rooted at z, such that $z \in \overline{V}$. For z's cut in Fig. 2(a), $\overline{V}$ consists of two nodes, x and z. A cut is called K-feasible if the number of nodes in V that drive nodes in $\overline{V}$ is $\leq K$. In the case of Fig. 2(a), there are 3 nodes that drive nodes in $\overline{V}$, and, the cut is 3-feasible. For a cut $C = (V, \overline{V})$, $Inputs(C)$ represents the nodes in V that drive a node in $\overline{V}$. For the cut in Fig. 2(a), $Inputs(cut) = \{a, d, c\}$. $Nodes(C)$ represents the set of nodes, $\overline{V}$.

For a K-feasible cut, $C = (V, \overline{V})$, the logic function of the subgraph of nodes $\overline{V}$ can be implemented by a single K-LUT (since the cut is K feasible and a K-LUT can implement any function of K variables). The key point to realize is that the problem of finding all of the possible K-LUTs that generate a node's logic function is equivalent to the problem of finding all K-feasible cuts for the node. Generally, there can be multiple K-feasible cuts for a node in the network, corresponding to multiple LUT implementations. $Cuts(z)$ represents the set of all feasible cuts for a node z.

Traversing the circuit DAG in topological order, the cuts for a node z are generated by merging cuts from its fanin nodes, using the method described in [10], [9] and outlined here. Consider generating the K-feasible cuts for a node z with two fanin nodes, x and c. The list of K-feasible cuts for x and c have already been computed, due to the graph traversal order. Assume that node x has one K-feasible cut, C_X, and node c has one K-feasible cut, C_C, as shown in Figure 2(b). We can merge C_X and C_C to create a cut, C_Z, for node z, such that $Inputs(C_Z) = Inputs(C_X) \cup Inputs(C_C)$ and $Nodes(C_Z) = z \cup Nodes(C_X) \cup Nodes(C_C)$ (see Figure 2(b)). If $|Inputs(C_Z)| > K$, the resulting cut is not K-feasible, and is discarded. In this example, input nodes x and c have only one cut each, however, if instead they had multiple

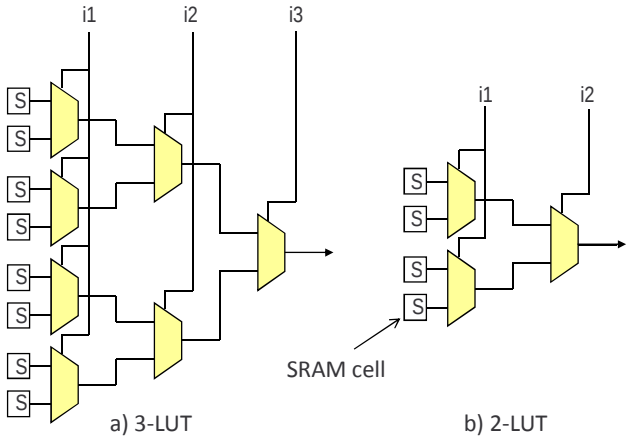

Fig. 1. 2 and 3-input LUTs.

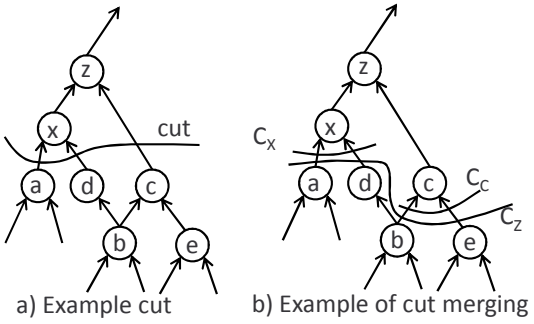

Fig. 2. Illustration of K-feasible cuts.

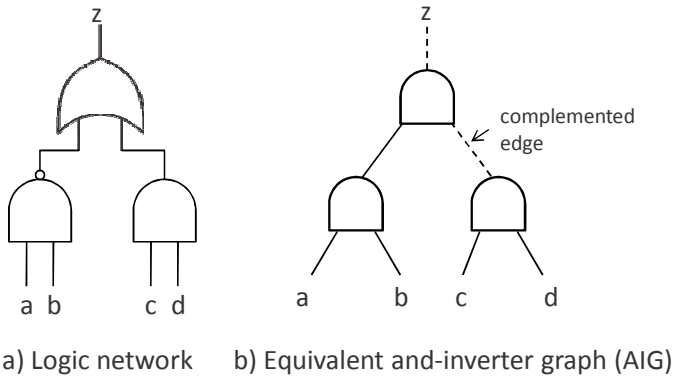

Fig. 3. AIG example.

cuts, all possible cut merges would be attempted to form the complete cut set for z. This provides a general picture of how the cut generation procedure works, however, there are several special cases to consider and the reader is referred to [9] for details.

Having computed the set of K-feasible cuts for each node in the DAG, the graph is traversed in topological order again. During this second traversal, a "best cut" is chosen for each node. The best cut may be chosen based on any criteria, whether it be area, power, delay, routability or a combination of these. The best cuts define the LUTs in the final mapped solution.

C. The ABC Synthesis Framework

Work on logic synthesis has been reinvigorated by the introduction of the ABC system developed at UC Berkeley [11]. In ABC, the circuit DAGs are AND-inverter graphs (AIGs), that is, logic functionality is represented as a network of 2-input AND gates connected by invertible edges. An example of an AND-inverter graph is shown in Fig. 3. The use of AIGs eases the implementation of many useful logic synthesis transformations (e.g., [12], [13]).

Among other advantages, AIGs have proven to be effective for cut-based LUT mapping. In [14], Mischenko et al. introduced the notion of *priority cuts*, where instead of storing all possible cuts for each node, only a subset of "priority cuts" is stored, based on a cost function. When generating the cut set for a node, only the priority cuts of its fanin nodes are considered for merging. Despite that many cuts are pruned with this technique, little quality degradation is observed in practice, and results are comparable to any competing mapper. Mapping quality is not compromised by using AIGs compared with other network representations [14].

We conduct our research using AIGs within the ABC framework and we propose new logic element architectures and a mapping approach. Our logic elements contain structures beyond LUTs and experimental results demonstrate the area and performance benefits of the proposed logic elements. It is worth mentioning that a few recent works also studied technology mapping into non-LUT structures. Ling et al. used SAT-based techniques for mapping into blocks with LUTs and gates [15]. Recent work from Actel used cut-based techniques to map into a logic block architecture with gates, and then applied Boolean matching to filter cuts that could not be legally mapped to the target block [16]. Both of these prior works considered the mapping problem in isolation, and not from the architecture evaluation perspective.

III. Logic Element Architecture and Mapping

Our architecture and technology mapper take advantage of the AIG representation of logic functions. In particular, the proposed logic element architecture relies on the property that *only* AND gates and inverters can appear in the graph. We introduce the proposed architecture using the example AIG shown in Fig. 4(a). Two 4-input cuts are shown: $cut0$ and $cut1$, corresponding to LUTs implementing the functions $z = q \cdot \overline{i2 \cdot i1 \cdot \overline{i0}}$ and $z = i4 \cdot \overline{i3} \cdot \overline{i2 \cdot m}$, respectively.

Both $cut0$ and $cut1$ are 4-feasible cuts. However, a key observation can be made regarding $cut0$ and $cut1$ in Fig. 4(a). Looking first at $cut0$, observe that one of the inputs to the cut is the output signal of gate q, and that signal is also a direct input of gate z (the root node). Since gate z is an AND gate, we know that when the output of q is logic-0, then the output of z *must* necessarily be logic-0. Conversely, when the output of q is logic-1, the output of z is the output of gate l (complemented), which in this case is only a 3-input logic function. Hence, for the case of $cut0$, even though the cut is 4-feasible and represents a logic function of 4 variables, we do not need the full flexibility of a 4-LUT in hardware to realize the function. In fact, we can realize the function using the logic element shown in Fig 5(a), comprising a 3-LUT and a single AND gate – an *extended* LUT. Since the AIG subject graph contains only 2-input AND gates with optional edge complementation, we need not be concerned with gates other than AND appearing in the input circuit graph. In essence, we are using a property of the synthesis graph to inspire our logic element architecture.

Turning now to $cut1$ in Fig. 4(a), one can see that the same observation also applies: if either $i4$ or $\overline{i3}$ is logic-0, then the output z is also logic-0. In this case, however, none of the cut inputs are also inputs to the root AND gate. Yet again, we do not need the full power of a 4-LUT to express the function of $cut1$. Regarding $cut1$, observe that the "gating" property does not hold for all of the inputs to the cut, for example, if input m is logic-0, we cannot determine whether z will be logic-0 or logic-1, as it will depend on the values of the other cut inputs.

It is worth mentioning the relationship between gating inputs to a function and *unate* inputs, which are well-described in the literature [17]. Consider a function f with an input variable x. x is said to be a unate input to f if and only if f's sum-of-products (SOP) representation contains *either* x or $\overline{x}$, but not both. If x is a unate input and f's SOP representation contains x (in true form), then a transition on x can only cause a transition on f in the *same* direction. On the other hand, if f's SOP represention contains only $\overline{x}$, then a transition on x can only cause a transition on f in the opposite direction. Certainly, gating inputs to a function are unate inputs; however, the converse is not necessarily true: a unate input may not necessarily be a gating input.

A. Mapping Approach: Non-Inverting Paths in the AIG

The core of our approach is to restrict cuts with K inputs to those that resemble the cuts in Fig. 4(a). The defining feature of such cuts is the presence of a *non-inverting path* from at least one of the cut inputs to the root of the cut. Some examples are shown in Fig. 4(b). In this case, when any of inputs $i1$, $\overline{i2}$, or $i5$ is logic-0, root node r's output must be logic-0. Observe that the edge crossing the cut may be a complemented edge, as is the case for $(i2, l)$ in the figure. However, edges along the path from the cut "frontier" nodes[1] to the root must be non-inverting. It is a straightforward process during cut generation to traverse the graph downward from the root and determine whether K-input cuts have at least one non-inverting path to a cut input. Fig. 4(c) gives an example cut with no non-inverting path from any of its inputs.

Restricting cuts with K inputs to be those that contain non-inverting paths will produce mappings that can be accommodated in an architecture with *extended* K-1-LUTs, which require about half the silicon area of K-LUTs. The extension to which we refer is the presence of an AND gate on the LUT output, as shown in Fig. 5(b). The other input to the AND gate can be programmably connected to either the true or complemented form of an input signal, i. The optional inversion is needed to handle the case of complemented edges crossing the cut, such as $(i2, l)$ in Fig. 4(b). The restriction that cuts have non-inverting paths is only imposed for cuts with K inputs; cuts that use less than K inputs remain unrestricted. When the logic element in Fig. 5(b) is used to implement functions that require less than K inputs, we assume that the input i is tied to either VCC or ground and that the multiplexer select SRAM cell is set such that the AND gate is bypassed. We believe this to be a reasonable assumption, as unused logic block inputs are common in FPGA designs and commercial FPGAs contain circuitry to tie unused inputs to a known logic state.

The obvious question that arises is: What is the impact of restricting the cuts of size K from the # of logic elements and speed (depth) perspectives? Surprisingly, as we will demonstrate in our experimental study, our mapping approach and logic element achieve much of the benefits of K-LUTs, while consuming much less area.

[1] AND gates driven by cut edges.

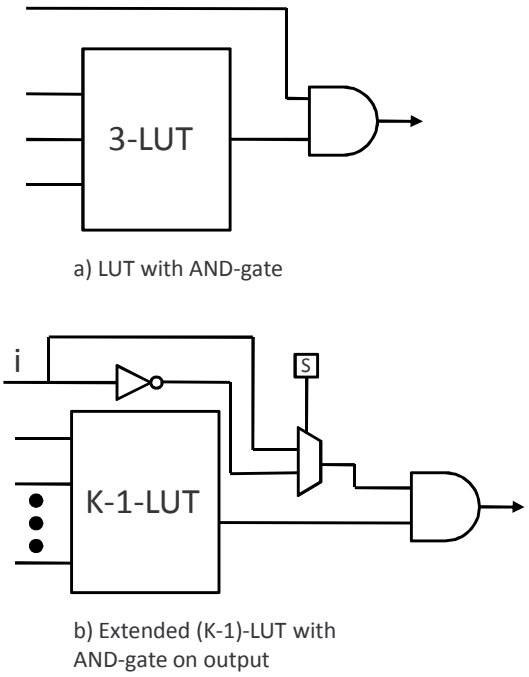

Fig. 5. Extended LUT with additional AND gate.

To define our approach formally, let $SCuts(z)$ be the set of cuts for a root node z that use less than K inputs, as computed using the standard merging procedure described in Section II:

$$SCuts(z) = \{C \in Cuts(z)\ s.t.\ |Inputs(C)| < K\} \quad (1)$$

and let $LCuts(z)$ be the set of K-input cuts of z that contain a non-inverting path to one of the cut inputs:

$$\begin{aligned} LCuts(z) \quad = \quad & \{C \in Cuts(z)\ s.t. \\ & |Inputs(C)| = K \wedge \\ & (\exists\ \pi_{i,z} \subset AIG\ s.t.\ i \in Inputs(C) \wedge \\ & \pi_{i,z}\ is\ non-inverting)\} \end{aligned} \quad (2)$$

where $\pi_{i,z}$ is a path in the AIG from a cut input i to the cut root z. If i directly drives z, then the path is a single edge and is a non-inverting path. Otherwise, there must be k intermediate nodes on the path from i to z and without loss of generality, we can represent $\pi_{i,z}$ as a sequence of AIG edges:

$$\pi_{i,z} = (i, n_1), (n_1, n_2), ..., (n_k, z) \quad (3)$$

As described in Section III-A, for path $\pi_{i,z}$ to be called non-inverting in (2), all of the edges on the partial path from n_1 to z must be uncomplemented, i.e. the edges $(n_1, n_2), ..., (n_k, z)$ must be true edges. The edge crossing the cut, (i, n_1), may be true or complemented.

Finally, the set of filtered cuts that will be considered for a node z in our technology mapper is:

$$FCuts(z) = SCuts(z)\ \cup\ LCuts(z) \quad (4)$$

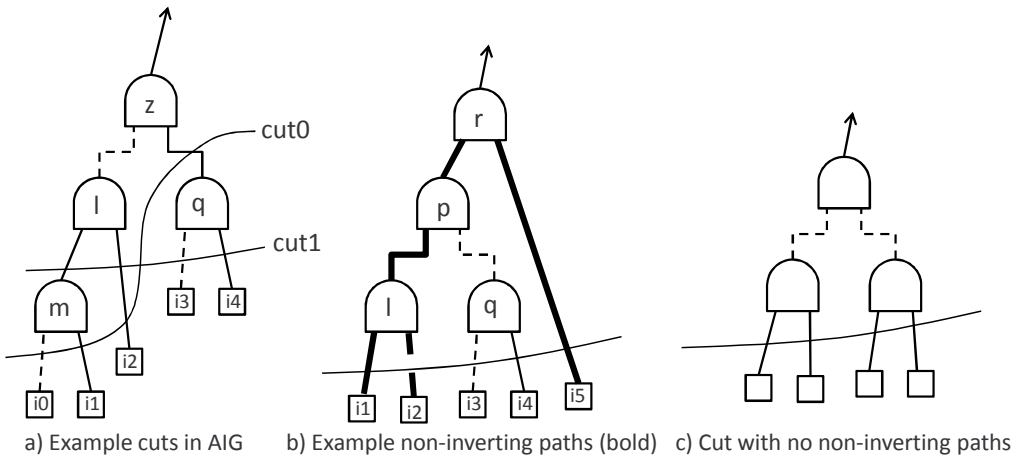

Fig. 4. Cut examples.

B. Identifying Additional Gating Inputs

While the discussion above centers on identifying a LUT input that cause the LUT's function to evaluate to logic-0, there may also exist easily identifyable LUT inputs that cause a function to evaluate to logic-1. An example case is illustrated in Fig. 6, which shows a cut from one of the benchmark circuits used in our experimental study (`alu4`). The logic function implemented by the cut is: $f = \overline{\overline{i1} \cdot i2 \cdot \overline{i3} \cdot i4 \cdot \overline{i4 \cdot i5 \cdot i6}}$. An inspection of the AIG reveals that no input to the cut has a non-inverting path to the root – no single input can cause the function to evaluate to logic-0. Applying De Morgan's law to the two clauses in the cut's Boolean function, we attain the function in conjunctive normal form: $f = (i1 + \overline{i2} + i3 + \overline{i4}) \cdot (\overline{i4} + \overline{i5} + \overline{i6})$. In this form, we see by inspection that input $i4$ is a gating input to the cut: When $i4$ is logic-0, function f evaluates to logic-1. Observe in the AIG that there are reconvergent paths from input $i4$ to the cut root f. Though the cut in Fig. 6 does not contain an non-inverting path, it does indeed have a gating input. We again do not need the full power of a 6-LUT to implement the function.

The block architecture shown in Fig. 7 is capable of handling both cases where a gating input causes the function to evaluate *either* logic-0 or logic-1. It has approximately the same silicon area as the block in Fig. 5(b)[2], with the key change being that the 2-input `AND` gate in Fig. 5(b) is replaced with a 2-to-1 multiplexer, $MUX2$, in Fig. 7. One of $MUX2$'s data inputs is received from the LUT; its second data input is received from an SRAM configuration cell. The SRAM configuration cell is configured according to whether the gating input causes the function to evaluate to logic-0 or logic-1. As before, multiplexer $MUX1$ permits the input's gating state to be either logic-0 or logic-1. The architecture in Fig. 7 is also referred to as an extended LUT, however, in this case, the LUT is extended with a `MUX` instead of an `AND`.

A straightforward extension of the mapping approach outlined above can be used to identify cut inputs that cause a function to evaluate to logic-1. Let r be the root gate of the

[2]The block in Fig. 7 uses an extra SRAM configuration cell.

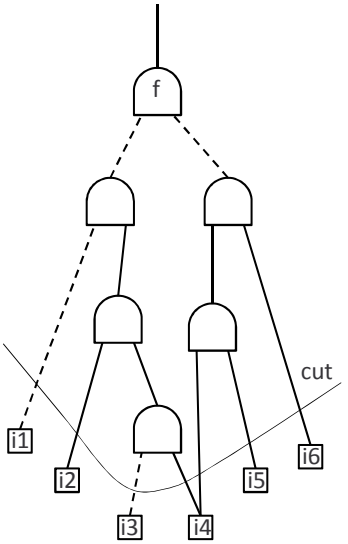

Fig. 6. Example AIG cut from benchmark circuit `alu4` with a controlling input *i4* that causes the function to evaluate to logic-1.

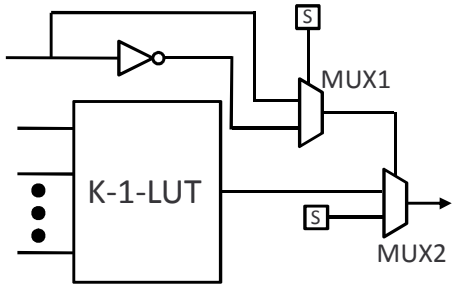

Fig. 7. Extended LUT with additional 2-to-1 multiplexer.

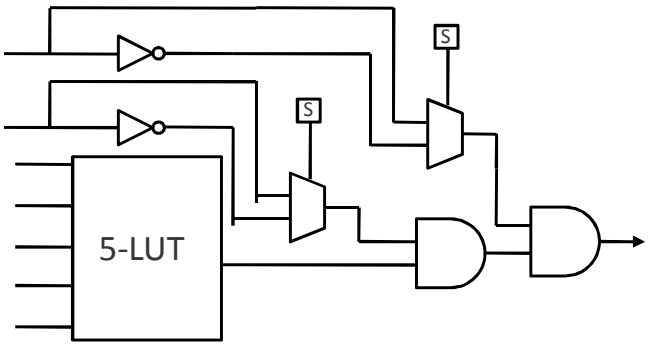

Fig. 8. 5-LUT with two cascaded AND gates.

cut under consideration; let a and b represent r's fanins; and, let i be the cut input we wish to analyze. Input i is a gating input that causes the cut function to evaluate to logic-1 if the following conditions are met:

- The fanin edges of r are inverted.
- There are non-inverting paths from i to a, and from i to b. The non-inverting paths from i cause a and b to evaluate to logic-0 when i is in a particular logic state (either logic-0 or logic-1).

In essence, we seek *partial* non-inverting paths from a cut input to the fanin nodes of the cut's root node, with the added requirement that the root's fanin edges be complemented. Note that the logic element in Fig. 7 can also accommodate cases where a gating cut input causes the cut root to evaluate to logic-0. Such cases can be discovered using the approach outlined in Section III-A above, namely, finding a single non-inverting path from an input to the root.

In our experimental study, we consider both AND-extended LUTs (Fig. 5(b)) as well as MUX-extended LUTs (Fig. 7), and we show that the added flexibility afforded by the MUX-extended LUT provides modestly better performance and area results[3].

C. Generalized Architectural Families: Extended LUTs

Having considered two classes of gating inputs to LUTs, we now broaden the scope to consider cases wherein there are *multiple* gating input signals. The AND and MUX-extended LUT logic element architectures described above (and shown in Figs. 5(b) and 7) can be viewed as members of a *family* of logic element architectures, each containing an L-input LUT with M cascaded gates on its output. For example, Fig. 8 shows a 5-LUT with two cascaded AND gates. We characterize such logic element architectures in a general form as $\{L,M\}$-**AND-extended LUTs** and $\{L,M\}$-**MUX-extended LUTs**. For example, a $\{4,2\}$-AND-extended LUT contains a 4-input lookup table, followed by two cascaded AND gates. The generalized forms are depicted in Fig. 9. Our experimental study considers a wide range of logic element architectures that fall into these generalized logic element families.

[3]The conference version of this paper considered only AND-extended LUTs [6].

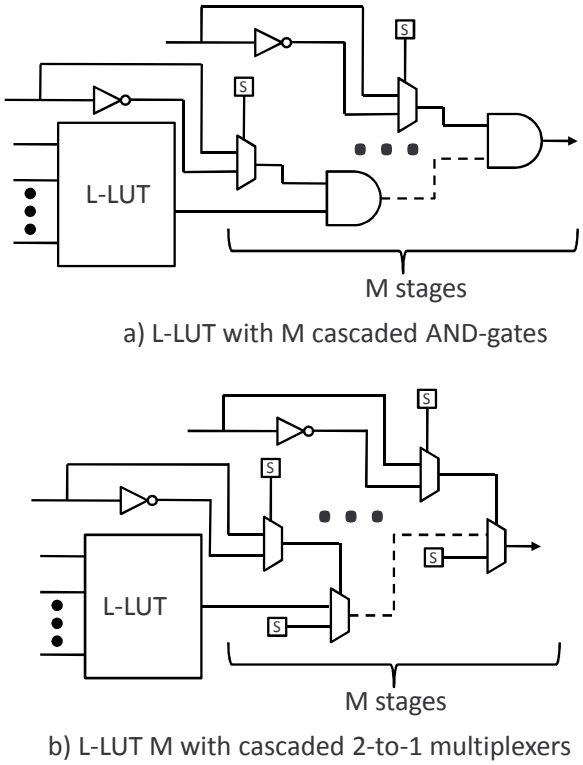

Fig. 9. L-LUT with cascade of M gates.

We envision that LUTs extended with other types of gates, for example an exclusive-OR-extended LUT, may also prove useful, however, mapping circuits into such architectures is not straightforward and cannot be achieved through a simple traversal of the AIG representation.

D. Overall Architecture

Fig. 10 gives an abstract view of an FPGA and illustrates the proposed architectural change. The FPGA itself is a two-dimensional array of tiles with programmable logic and routing resources. The figure shows that in addition to LUTs, tiles contain other logic, for example fast carry chain arithmetic logic and storage elements (programmable flip-flops). We propose to replace the K-input LUTs with alternative logic elements that *also* use K inputs, namely the *extended* LUTs that use considerably fewer transistors (and less area) than K-LUTs (illustrated on the lower-right of Fig. 10). The arithmetic and other logic surrounding the LUTs can remain unchanged in the proposed architecture.

Regarding carry logic, in Xilinx FPGA families such as Virtex-5 and Virtex-6, coupled with each 6-LUT is a 2-to-1 *carry chain multiplexer* driven by the LUT output. The multiplexer realizes the carry generate/propagate functionally in carry look-ahead addition. Specifically, for the addition of two bits A and B, the 6-LUT is used in dual-output mode, where one of the two outputs produces the propagate function ($A \cdot B$) and the second output produces the generate function ($A \oplus B$). Two functions of two common inputs can be

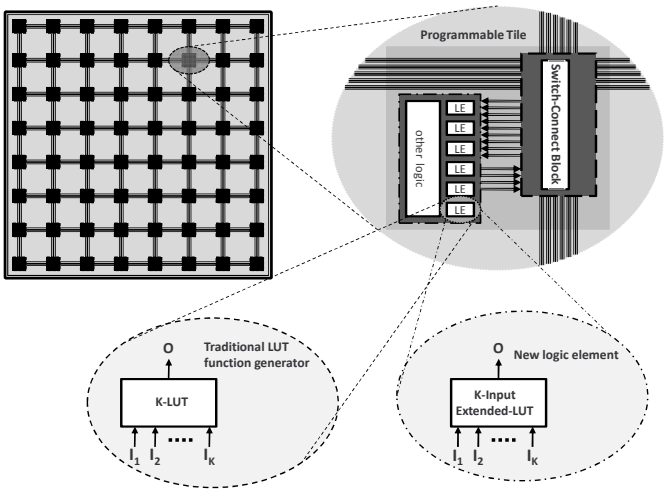

Fig. 10. Illustration of proposed architectural change.

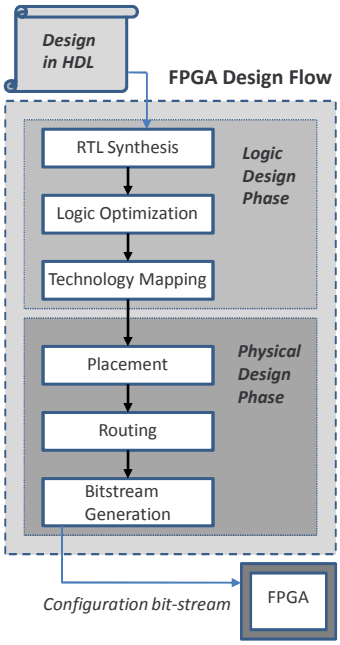

Fig. 11. Standard FPGA CAD flow.

realized in a dual-output 3-LUT and consequently, as long as the proposed extended LUT contains at least a 3-LUT, carry arithmetic can be handled identically to today's commercial chips. Note also that in some commercial FPGAs, the SRAM cells in the LUTs can be used to implement small memories and/or shift registers. Such functionality is also possible using the proposed extended LUTs, albeit with few SRAM bits available.

Since the original K-LUTs and the proposed elements use the same number of inputs, it is expected that they will exert a similar demand on the FPGA's programmable routing fabric. The equivalent pin demand implies that the new logic elements can be interchanged with the orignal LUTs, while the programmable routing fabric remains constant. In other words, the new logic elements do not necessitate a change in the FPGA's routing fabric – a property that makes them fairly straightforward to incorporate into an existing commercial architecture.

E. CAD Implementation

Fig. 11 illustrates the typical FPGA design flow, comprising HDL and logic synthesis, technology mapping, placement, routing and finally bitstream generation. Only the technology mapping phase needs to be specialized for the proposed architectural change. No changes are necessary for the other phases of the flow, e.g. changes to placement and routing. Supporting the proposed architecture is relatively low-cost from the tools' standpoint.

Mapping circuits into logic elements with multiple cascaded gates ($L > 1$) can be achieved through repeated application of the techniques described above, with the added requirement that finding more than one non-inverting path in the AIG may be necessary. For example,to map a 7-input logic function into the architecture of Fig. 8, we must identify two gating inputs. Mapping a function into a logic element with a cascade of `MUX` gates is also straightforward. For example, we wish to evaluate whether a 6-variable function, f, can map into a logic element with a 4-LUT and two cascaded `MUX` gates. We first identify a gating input, i, to the function f using the approach described in Section III-B. If that is successful, we are left with a 5-variable function, g, that is a factor of f (variable i is factored out)[4]. We then use the same procedure to search for a gating input to g. If such an input to g can be identified, then function f can be realized in the logic element.

Note that the structure of cascaded gates in Fig. 8 is for illustration/clarity only – two cascaded `AND` gates can be implemented more efficiently in CMOS as single larger `AND` gate, rather than multiple serially connected small gates – a 2-input `AND` gate requires 6 transistors (a 2-`NAND` followed by an inverter); a 3-input `AND` gate requires 8 transistors (a 3-`NAND` followed by an inverter). Note that the CAD and architectural perspectives of the proposed logic elements can be separated from the circuit-level details: From the point of view of the technology mapping, knowing the K and L parameters along with the logic element style (`AND` or `MUX`) are sufficient to produce a legal mapping.

In cut-based FPGA technology mapping to K-LUTs, for a node v with n nodes in its transitive fanin cone, there are at most $O(n^K)$ potential cuts [18]. For each such cut, performing the check for non-inverting paths requires, in the worst case, traversing the entire fanin cone – $O(n)$ time. Hence the overall complexity for mapping is $O(n \cdot n^K) = O(n^{K+1})$, which is polynomial time as K is a fixed constant. In general, we did not observe any appreciable increase in mapper run-time for targeting the proposed architectures versus targeting a traditional LUT-based architecture. The placement and routing steps are, by far, the most compute-intensive phases of the FPGA CAD flow.

We map circuits into the proposed architectures using a

[4]In fact, g is either the 0-cofactor or 1-cofactor of f after Shannon decomposition with respect to input i.

modified version of the ABC technology mapper based on priority cuts [14]. Our modified mapper can operate in one of two ways:

1) **Hard:** When the technology mapper is generating the set of K-feasible cuts for a node, we ignore all cuts that cannot be accommodated in the architecture being targeted. The mapping solution produced is therefore *guaranteed* to contain only logic element instances that fit into the target logic element architecture.
2) **Soft:** We do not ignore any of the K-feasible cuts generated for any node. Rather, we change the way cuts are ranked by the priority cuts mapping algorithm. Specifically, we use the mapped depth as the primary criterion for ranking cuts, and as a secondary criterion, we prefer to choose cuts that legally fit into the target logic element architecture.

The purpose of the "soft" flow is to evaluate, for a K-LUT-based logic element architecture, how many LUTs in the mapping solution need to be full K-LUTs if optimal mapping depth is to be achieved, with the remainder being accommodated by extended LUTs.

While the AIG circuit representation was the inspiration for our proposed architectures, its use is not required to identify gating inputs, nor required to target the proposed logic element architectures. Consider, for example, the standard sum-of-products (SOP) and product-of-sums (POS) representations of logic functions (as an alternative to AIGs). Any input in a function's SOP representation that is present in all of its product terms in one polarity (either true or complemented) is a gating input that can force the function to logic-`0`. Likewise, any input in a function's POS representation that is present in each of its sum clauses in one polarity is a gating input that can force the function to logic-`1`. Hence, while AIGs offer a convenient way to find gating inputs – through non-inverting paths – it is also easy to identify gating inputs for functions SOP/POS form. We believe it to be straightforward to modify any cut-based FPGA technology mapper to target the proposed extended LUTs, by filtering candidate cuts that do not meet the gating requirements.

IV. Experimental Study

A. Methodology

We use the mapper in [14] as the baseline mapper to which we compare. The baseline mapper was executed in depth mode, which achieves the minimum depth mapping and then performs area-driven post-passes based on the area-flow concept [19]. The technology-independent transformations applied to circuits prior to technology mapping have considerable impact on mapping results. Multiple technology-independent transformation scripts are included with the ABC package. Prior to technology mapping, we applied the `resyn2` script. We also investigated using the `compress2` script, but found it produced slightly worse depth results, on average. For mapping into extended K-LUTs, we altered the area-driven post-passes in ABC to ensure that they produced mappings compliant with the logic element architectures targeted.

We borrow the approach of [5] and use two different sets of benchmark circuits in our experimental study: 1) The 20 combinational and sequential circuits commonly used in academic FPGA CAD and architecture research, and 2) the 13 largest circuits from the widely used VPR 5.0 circuit set [20]. We used Altera's Quartus 9.1 tool to synthesize the VPR 5.0 circuits from Verilog to BLIF. Altera's QUIP (Quartus University Interface Program) flow [21] was used to produce BLIF for each circuit following HDL elaboration and technology independent synthesis. For all architectures and circuits considered, technology independent optimization (using `resyn2`) and technology mapping was executed 6 times and the best result achieved is reported. A similar multi-pass methodology was applied in [22].

B. Results

We present two sets of results. We first present results for 6-input logic element architectures. Such logic elements could be directly interchanged with the 6-LUTs in a modern commercial FPGA, such as the Xilinx Virtex-5. Changes to the interconnection fabric would not be required, as the fabric is already designed to handle the routing demand imposed by 6-input logic elements. Subsequently, we present results for 7-input logic element architectures. Early commercial FPGAs (in the 1980s and 1990s) used 4-LUTs and the recent trend has been towards larger LUTs. In the future, we may well see commercial architectures with 7-input elements, and consequently, it is desirable to evaluate area/performance trade-offs for 7-input logic elements.

Table I gives results for mapping circuits into 6-LUTs (the baseline), 5-LUTs, and six different 6-input logic element architectures: {5,1}-`AND`, {5,1}-`MUX`, {4,2}-`AND`, {4,2}-`MUX`, {3,3}-`AND`, {3,3}-`MUX`. Recall that {L,M}-`AND` and {L,M}-`MUX` architectures contain an L-LUT followed by a cascade of M gates (`AND` or `MUX`). Hence, the architectures presented in the table use progressively less silicon area as the table is read from left to right. With the exception of the 5-LUTs, all architectures in the table require 6 inputs and hence, they could all be embedded into similar FPGA routing fabric that consumes a fixed amount of silicon area. For each architecture and circuit considered, both the depth of the mapped network (labeled "DEP") and the number of logic elements are given (labeled "#LEs"). The top half of the table shows results for the 20 circuits most commonly used in FPGA research; the bottom half of the table shows results for the VPR 5.0 circuits. We observed markedley different results for the two different benchmark circuit sets and therefore, we decided to give geometric mean results for each circuit set, as well as for all of the circuits together (last rows of the table).

We first consider results for 5-LUTs versus 6-LUTs (see the "6-LUTs" and "5-LUTs" columns of Table I). For the 20 standard benchmarks, 5-LUT mapping solutions are 12% deeper and use 15% more LUTs than 6-LUT mapping solutions. For the VPR 5.0 circuits, 5-LUT mappings are 18% deeper and use 13% more LUTs than 6-LUT mappings. The VPR 5.0 circuits are more sensitive to changes in the number of LUT inputs versus the 20 standard circuits commonly used in FPGA

TABLE I
RESULTS FOR 6-INPUT LOGIC ELEMENT ARCHITECTURES AND 5-LUTS.

Circuit	6-LUTs		5-LUTs		5,1-AND		5,1-MUX		4,2-AND		4,2-MUX		3,3-AND		3,3-MUX	
	DEP	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**
alu4	5	774	6	866	6	778	5	804	6	823	6	768	6	876	6	760
apex2	6	868	7	990	6	887	6	906	6	955	6	913	7	972	6	930
apex4	5	752	6	840	5	780	5	761	5	824	5	805	6	810	5	788
bigkey	3	579	3	806	3	691	3	691	3	695	3	696	3	1254	3	811
clma	10	2795	12	3179	10	2985	10	2917	12	3079	10	3278	13	3354	11	2922
des	4	691	5	948	5	906	5	839	5	1011	5	928	6	1290	5	1010
diffeq	8	636	9	757	9	732	9	721	11	806	10	773	15	1144	13	819
dsip	3	689	3	692	3	693	3	691	3	696	3	694	3	1369	3	697
elliptic	10	1797	12	1873	11	1855	11	1828	14	1880	12	1903	19	2950	18	2100
ex1010	6	2452	7	2755	6	2483	6	2505	7	2350	6	2533	7	2613	6	2573
ex5p	5	504	5	594	5	534	5	516	5	562	5	532	5	606	5	541
frisc	13	1735	14	1842	13	1823	13	1831	15	1924	13	1908	21	2878	20	2027
misex3	5	723	6	811	5	770	5	745	5	818	5	786	6	838	5	771
pdc	7	1948	7	2495	7	2107	7	2057	7	2320	7	2067	7	2392	7	2070
s298	8	641	9	731	8	677	8	642	9	710	8	702	10	743	9	641
s38417	7	2567	8	3068	7	3002	7	2985	8	3165	7	3221	10	3695	9	3562
s38584.1	6	2287	7	2688	6	2586	6	2529	6	3267	6	3083	8	3810	7	3228
seq	5	780	5	908	5	795	5	808	5	845	5	827	5	876	5	812
spla	6	1670	7	1906	7	1673	7	1694	7	1919	7	1712	7	2023	7	1716
tseng	8	647	9	692	8	685	8	680	10	721	9	704	12	941	11	729
GEOMEAN	6.08	1075.16	6.82	1241.43	6.32	1154.75	6.26	1140.92	6.78	1228.35	6.42	1195.25	7.69	1485.95	7.06	1220.59
RATIO VS 6-LUTs:			1.12	1.15	1.04	1.07	1.03	1.06	1.11	1.14	1.06	1.11	1.26	1.38	1.16	1.14

Circuit	6-LUTs		5-LUTs		5,1-AND		5,1-MUX		4,2-AND		4,2-MUX		3,3-AND		3,3-MUX	
	DEP	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**
cf_cordic_v_18_18_18	9	3822	11	4461	10	4205	10	4202	12	5444	11	5387	18	7124	17	6467
cf_fir_24_16_16	18	10929	18	11668	18	11843	18	11718	24	14796	19	15620	36	17327	36	17152
des_perf	3	3264	4	4959	4	4337	4	4213	4	5606	4	4751	6	9430	6	7753
mac1	17	1959	21	2215	18	2277	18	2143	22	2847	19	2854	29	3697	20	2890
mac2	31	6906	39	7491	32	7601	32	7361	39	9798	33	9841	52	12535	34	9885
oc54	22	2393	24	2726	22	2601	22	2598	28	3253	23	3310	39	3694	38	3297
paj_boundtop_hierarchy	4	1294	6	1376	4	1373	4	1371	5	1407	4	1406	5	1571	4	1509
paj_raygentop_hierarchy	16	6314	17	6677	17	6696	17	6647	22	7790	18	7912	30	9445	30	9078
paj_top_hierarchy	33	32967	34	35016	34	35196	34	35119	44	40882	34	43536	65	49804	65	48300
rs_decoder_2	11	1649	14	2265	13	2076	13	1885	16	2454	16	2263	22	3361	18	2697
sv_chip0_hierarchy	5	12615	6	12955	6	12970	6	12970	8	13693	7	13950	11	14374	10	14253
sv_chip1_hierarchy	8	25473	9	26882	9	27107	9	26376	11	31162	10	32929	15	39652	15	35776
sv_chip2_hierarchy	18	46440	19	50104	19	50215	19	49973	24	59437	22	58675	33	60000	33	60000
GEOMEAN	11.88	6384.02	13.98	7233.55	12.93	7094.34	12.93	6946.43	16.01	8463.19	13.91	8437.15	21.90	10413.09	19.67	9391.64
RATIO VS 6-LUTs:			1.18	1.13	1.09	1.11	1.09	1.09	1.35	1.33	1.17	1.32	1.84	1.63	1.66	1.47
ALL CIRCUITS GEOMEAN	7.92	2168.87	9.05	2485.73	8.38	2360.91	8.33	2324.36	9.51	2627.41	8.70	2581.12	11.61	3199.70	10.57	2726.89
RATIO VS 6-LUTs:			1.14	1.15	1.06	1.09	1.05	1.07	1.20	1.21	1.10	1.19	1.47	1.48	1.34	1.26

research. In general, while more 5-LUTs are needed than 6-LUTs to implement a circuit, a 5-LUT requires just half the silicon area of a 6-LUT. On the other hand, 6-LUTs deliver a considerable depth reduction over 5-LUTs (14% across all circuits in both benchmark sets) which is the prime reason that commercial FPGA vendors have trended to 6-LUTs recently in their high performance product lines.

Moving onto the proposed {5,1}-AND and {5,1}-MUX architectures, observe that across all circuits (bottom rows of Table I), relative to 6-LUTs, the {5,1}-AND architecture increases depth by 6% and the {5,1}-MUX architecture increases depth by 5%. As compared with 6-LUTs, the number of logic elements is increased by 9 and 7% for the {5,1}-AND and {5,1}-MUX architectures, respectively. Both of the proposed architectures require silicon area close to that of a 5-LUT, yet they both deliver most of the depth benefit of 6-LUTs. Moreover, not as many of the extended 5-LUTs are needed to implement circuits versus pure 5-LUTs. On the depth and logic element count axes, the added flexibility offered by the MUX architecture provides slightly better results than the AND architecture.

It is worthwhile to examine the dependence of the results on the benchmark set. For the {5,1}-MUX architecture, mapping depth is 3% higher than 6-LUTs, on average, for the standard 20 benchmarks. However, for the VPR 5.0 benchmarks, mapping depth is 9% higher than 6-LUTs, on average. The results demonstrate that the choice of benchmark set can have a significant impact on both architectural conclusions as well as the perceived efficacy of CAD algorithms. While it remains unclear which of the two benchmark sets is more representative of the universe of all circuits, the VPR 5.0 circuits appear to carry a higher "richness" in their logic functions and exact a higher demand on the underlying logic element architecture.

The {4,2}-AND and {4,2}-MUX in Table I contain a 4-LUT with 2 cascaded gates. The gap in mapped depth between these two architectures is wider than in the {5,1} case. Relative to 6-LUTs, the {4,2}-AND architecture increases depth by 20% and logic element count by 21%, whereas the {4,2}-MUX architecture increases depth by just 10% and logic element count by 19%. The MUX architectures can accommodate a wider range of logic functions. Fig. 12 illustrates a cut that can be implemented in a {4,2}-MUX architecture yet cannot be implemented in a {4,2}-AND architecture. In the example, input $i6$ has a non-inverting path to the root. With $i6$ factored out, the remaining function is: $f = (i1+\overline{i2}+\overline{i3})\cdot(\overline{i3}+\overline{i4}+\overline{i5})$, in which $i3$ is a gating input whose logic-0 state causes f to evaluate to logic-1. While $i3$ is not a gating input to the overall function g, it is indeed a gating input to function f, revealed only after $i6$ is selected as a first gating input to g. The right-side of Fig. 12 shows how the signals map to pins of the logic element architecture. A last observation in relation to the {4,2} architectures is that the {4,2}-MUX architecture actually produces mapping solutions having smaller depth than 5-LUTs, despite the fact that the {4,2} solution uses half of the area of a 5-LUT.

The data in the right-most columns of Table I for the {3,3}-AND and {3,3}-MUX architectures is included for completeness. Such architectures increase depth by over 34% percent versus 6-LUTs, on average. We do not believe such a performance loss would be acceptable in a future commercial architecture.

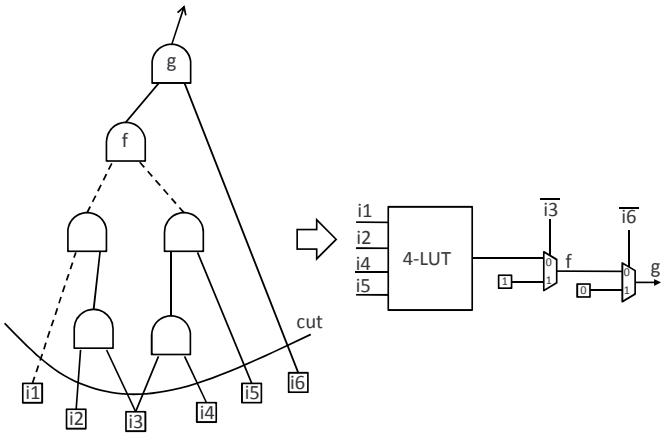

Fig. 12. Cut mapping into {4,2}-MUX architecture.

Table II shows the results for 7-LUTs (baseline), 6-LUTs, and a variety of 7-input logic element architectures. Looking at the last rows of the "6-LUTs" columns, we see that 6-LUT mappings are 22% deeper than 7-LUT mappings and require 9% more LUTs, on average, across all circuits. For both circuit sets, the depth advantage of moving to 7-LUTs from 6-LUTs is larger than that observed for moving to 6-LUTs from 5-LUTs. We expect that some modern commercial designs may be highly pipelined and therefore more "shallow" than the circuits considered here. Highly pipelined circuits may exhibit less dependence on LUT size.

The architectural trends observed in Table II are similar to those in Table I. As before, we observe that most of the depth benefit of moving from 6-LUTs to 7-LUTs can be achived with the {6,1}-AND and {6,1}-MUX architectures, with the MUX architecture providing slightly better results. On average, across all circuits, the {6,1}-MUX architecture offer mappings that are 8% deeper than 7-LUTs and use 7% more logic elements. This can be compared with 6-LUTs, which have roughly the same silicon area, yet whose mapping depth is 22% higher than 7-LUTs. As was the case in Table I, we observe a pronounced difference between the two circuit sets. In general, the standard 20 benchmarks are considerably less sensitive to the target element architecture.

The data in Table II suggests that if vendors added an MUX gate to their 6-LUT outputs and then mapped to such extended 6-LUTs, depth would be cut by about 14%, on average. In so doing, the 6-LUT-based blocks would need additional logic block inputs to provide a signal to the MUX input, possibly impacting routing demand. However, the Xilinx Virtex-5, for example, already has extra inputs on its logic blocks (e.g. the bypass inputs), which could perhaps be made dual-usage for driving the MUX gate.

Table III gives the approximate hardware cost of the key logic element architectures considered in this paper, accounting for the cost of the cascaded gates. For each architecture, we list the # of SRAM configuration cells (including cells

TABLE II
RESULTS FOR 7-INPUT LOGIC ELEMENT ARCHITECTURES AND 6-LUTS.

Circuit	7-LUTs		6-LUTs		6,1-AND		6,1-MUX		5,2-AND		5,2-MUX	
	DEP	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**
alu4	5	667	5	774	5	680	5	671	5	753	5	690
apex2	5	822	6	868	5	875	5	861	6	848	6	815
apex4	5	657	5	752	5	689	5	675	5	715	5	696
bigkey	2	467	3	579	2	467	2	467	3	691	2	464
clma	9	2408	10	2795	9	2562	9	2559	10	2873	10	2662
des	4	611	4	691	4	747	4	705	5	875	4	805
diffeq	6	585	8	636	7	615	7	605	9	743	8	718
dsip	2	911	3	689	2	911	2	911	2	915	2	913
elliptic	9	1754	10	1797	9	1775	9	1775	11	1854	10	1857
ex1010	6	2125	6	2452	6	2250	6	2215	6	2297	6	2307
ex5p	4	418	5	504	4	513	4	503	4	564	4	514
frisc	10	1636	13	1735	11	1656	11	1646	13	1793	12	1783
misex3	5	612	5	723	5	665	5	635	5	711	5	657
pdc	5	1806	7	1948	6	1923	6	1867	6	2089	6	1950
s298	7	536	8	641	7	606	7	565	8	633	8	582
s38417	6	2425	7	2567	6	2561	6	2547	7	2954	7	2921
s38584.1	5	2063	6	2287	6	2230	5	2242	6	2545	6	2474
seq	4	731	5	780	5	706	4	775	5	746	5	715
spla	5	1419	6	1670	6	1560	5	1586	6	1659	5	1751
tseng	7	616	8	647	7	647	7	635	8	680	8	670
GEOMEAN:	5.15	977.68	6.08	1075.16	5.42	1040.13	5.26	1028.66	5.97	1140.28	5.65	1078.43
RATIO VS 7-LUTs:			1.18	1.10	1.05	1.06	1.02	1.05	1.16	1.17	1.10	1.10
Circuit	**7-LUTs**		**6-LUTs**		**6,1-AND**		**6,1-MUX**		**5,2-AND**		**5,2-MUX**	
	DEP	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**
cf_cordic_v_18_18_18	7	3598	9	3822	8	3878	8	3877	10	4214	10	4166
cf_fir_24_16_16	12	10699	18	10929	15	11885	15	11810	18	11843	18	11893
des_perf	2	2752	3	3264	3	3264	3	3264	4	4320	4	4058
mac1	15	1817	17	1959	15	2018	15	2000	18	2261	16	2102
mac2	26	6551	31	6906	27	7113	27	7101	32	7600	28	7279
oc54	16	2021	22	2393	18	2323	18	2326	22	2586	21	2528
paj_boundtop_hierarchy	4	1262	4	1294	4	1289	4	1289	4	1349	4	1346
paj_raygentop_hierarchy	12	5870	16	6314	14	6588	14	6528	17	6685	17	6643
paj_top_hierarchy	23	31329	33	32967	27	34096	27	34041	34	35152	34	35073
rs_decoder_2	10	1482	11	1649	11	1583	11	1553	14	2064	13	1833
sv_chip0_hierarchy	4	12360	5	12615	5	12656	5	12656	6	12968	6	12923
sv_chip1_hierarchy	6	24125	8	25473	8	26471	8	26448	9	27088	9	26523
sv_chip2_hierarchy	15	41515	18	46440	17	46708	17	46564	19	50446	18	50253
GEOMEAN:	9.35	5915.06	11.88	6384.02	10.87	6486.38	10.87	6461.58	13.00	7073.98	12.58	6877.38
RATIO VS 7-LUTs:			1.27	1.08	1.16	1.10	1.16	1.09	1.39	1.20	1.35	1.16
ALL CIRCUITS GEOMEAN:	6.51	1986.84	7.92	2168.87	7.13	2139.12	7.00	2121.59	8.12	2340.29	7.75	2237.52
RATIO VS 7-LUTs:			1.22	1.09	1.09	1.08	1.08	1.07	1.25	1.18	1.19	1.13

in LUTs, cells to control optional input inversion, and cells to feed data inputs on multiplexers), the # of 2-to-1 multiplexers [5], and the number of inputs to the logic element. We have assumed that LUTs are implemented using a tree of 2-to-1 multiplexers, as shown in Fig. 1. The right-most columns of Table III give the ratio of the # of SRAM cells and # of multiplexers to baseline 6-LUTs. Such ratios represent the approximate hardware area cost of each architecture versus 6-LUTs. We stress that the ratios are *approximate* as we have not, for example, included any buffer costs and expect that large LUTs implemented as multiplexer trees contain repeaters at intermediate tree nodes. Likewise, we have not included transistor sizings, which we expect to be vendor and device specific. In general, the data in Table III support the observation logic element logic area is dominated by LUT area, with the LUT dominance decreasing as gates are successively added in cascade to the LUT output. For example the {5,1}-MUX architecture is estimated to consume 52% of a 6-LUT's area.

[5]We assume a 2-input AND is roughly the same size as a 2-to-1 multiplexer.

C. Die Area and Delay Impact

Using the data in Table III and in the results Table I, we can make a coarse estimate of the overall improvement in logic density. Using an extended 5-LUT, such as the {5,1}-MUX architecture, instead of a 6-LUT will reduce the tile area needed for a logic element by roughly 50%. Smaller tiles will reduce wirelengths, interconnect capacitance and delay. As shown in Fig. 13(a), we estimate that in a CLB, such as the Xilinx Virtex-5 FPGA, the interconnection fabric (and its configuration circuitry and SRAM cells) consumes 50% of the tile layout area; the eight 6-LUTs in a Virtex-5 CLB (and their SRAM cells) consume 30% of the tile; and flip-flops and other circuitry comprise 20% of the tile.

Fig. 13(b) gives an estimate of the tile area when the eight 6-LUTs are replaced with eight extended 5-LUTs. We assume LUT area is halved, and therefore total tile area is reduced by 15% and LUTs now comprise about 17.5% of the tile.

TABLE III
HARDWARE COST OF LOGIC ELEMENT ARCHITECTURES

Architecture	# SRAM cells	# 2-to-1 MUX	# inputs	SRAM ratio vs. 6-LUTs	MUX ratio vs. 6-LUTs
5-LUT	32	31	5	0.50	0.49
6-LUT	64	63	6	1.00	1.00
5,1-AND	33	33	6	0.52	0.52
5,1-MUX	34	33	6	0.53	0.52
4,2-AND	18	19	6	0.28	0.30
4,2-MUX	20	19	6	0.31	0.30
7-LUT	128	127	7	2.00	2.02
6,1-AND	65	65	7	1.02	1.03
6,1-MUX	66	65	7	1.03	1.03
5,2-AND	34	35	7	0.53	0.56
5,2-MUX	36	35	7	0.56	0.56

This implies that if the original tile area were 1 unit^2, as in Fig. 13(a), the new tile area would be 0.85 units^2. Results in Table I demonstrate that 7% more extended 5-LUTs are needed vs. 6-LUTs to implement circuits. Consequently, logic density in silicon will scale by $1.07 \times 0.85 = 0.91$, which is roughly a 9% improvement in logic density vs. 6-LUTs. In other words, a given logic circuit would require 9% less silicon area if the proposed architecture is used.

Assuming a square tile layout, the tile dimensions are reduced from 1×1 to 0.92×0.92, as shown in Fig. 13(b) ($\sqrt{0.85} = 0.92$). Thus, the x-dimension and y-dimension have each been reduced by about 8%. Metal wire capacitance would be reduced accordingly, mitigating the higher logic depth associated with extended 5-LUTs. Recognize that a fraction of interconnection capacitance is metal capacitance and fraction is switch capacitance (capacitive load due to routing switches attached to metal wire segments). Switch capacitance is unaffected, so we cannot assume that interconnect delay will be reduced by 8%. Nevertheless, the tile size reduction bodes well for the practicality of the proposed logic block.

To further validate our results, we used VPR 5.0 [20] to pack, place and route circuits into logic blocks containing eight 6-LUTs and flip-flops. The cluster size of eight matches closely with Virtex-5 and Stratix-III FPGAs, whose logic blocks contain eight and ten 6-LUTs, respectively. A simple routing architecture with unidirectional length-4 wire segments was used. The circuits mapped into pure 6-LUTs were placed and routed and the minimum number of tracks per channel, W_{MIN}, needed to route each circuit was determined.

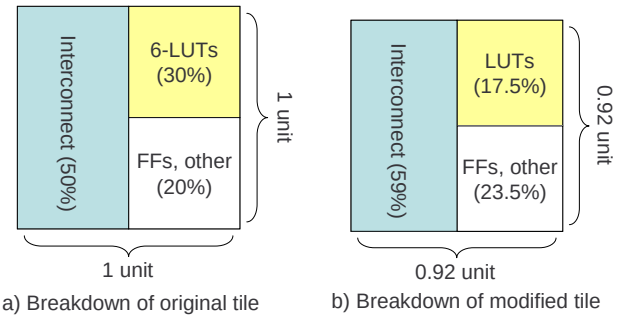

Fig. 13. Estimated tile area impact.

Then, both the baseline and experimental (enhanced 5-LUT) mapping solutions were packed, placed and routed into an architecture with $1.2 \cdot W_{MIN}$ tracks per channel. That is, routing architecture was held invariant between the baseline and experimental routing solutions. Each circuit was placed and routed 3 times with different placement seeds and the minimum critical path delay across the 3 runs was determined for each circuit. On average, critical path delay was 6% worse with the extended 5-LUTs, which concurs reasonably with the depth results given above. Note that 6% is a conservative upper bound on the performance hit, as it does not include the benefit of smaller tiles and reduced capacitance provided by the extended 5-LUTs.

D. Architectural Analysis

Finally, we did a preliminary architectural investigation of the value of heterogeneous logic blocks. We posed the question: If 6-LUT optimal depth must be achieved, how many of the LUTs need to have the full functionality of a 6-LUT vs. how many can be implemented using extended 5-LUTs, i.e. the $\{5,1\}$-`MUX` architecture? The results of this analysis are shown in Table IV. The left side of the table shows results for the standard 20 benchmark circuits; the right side of the table gives results for the VPR 5.0 circuits. For each circuit, two percentages are given. The first percentage, in the "ABC mapping" column shows the fraction of LUTs in mapping solutions produced by the baseline mapper [14] that require the full functionality of a 6-LUT (and could not be implemented using an extended 5-LUT). The second percentage, in the "Alternate mapping" column, gives results for the the mapping approach described in Section III-E that prefers to use extended 5-LUTs, but but does not impose hard restrictions and will not use extended 5-LUTs if mapping depth is compromised. These mapping solutions have the same optimal-depth as the mapping solutions of the baseline 6-LUT mapper.

The results in Table IV show that even using the baseline mapper, only 12% of LUTs need the full functionality of a 6-LUT to achieve optimal depth. Note that for this work, we used a more recent version of ABC than was used in [6]. The mapper in the new version of ABC incorporates the WireMap algorithm described in [23], which tends to produce fewer LUTs that use all 6 inputs. With the alternative mapping, we

TABLE IV
FRACTION OF LUTS IN MAPPING SOLUTIONS THAT NEED FULL 6-LUTS TO ACHIEVE OPTIMAL MAPPING DEPTH (VERSUS THAT COULD BE ACCOMMODATED IN A {5,1}-EXTENDED MUX ARCHITECTURE.

Circuit	ABC mapping	Alternate mapping	Circuit	ABC mapping	Alternate mapping
alu4	8.1%	1.7%	cf_cordic_v_18_18_18	10.5%	0.9%
apex2	8.4%	0.9%	cf_fir_24_16_16	18.1%	0.2%
apex4	8.0%	2.6%	des_perf	12.3%	12.3%
bigkey	38.9%	0.0%	mac1	10.5%	0.2%
clma	8.6%	2.7%	mac2	5.6%	0.1%
des	18.2%	3.9%	oc54	15.0%	0.3%
diffeq	20.3%	1.5%	paj_boundtop_hierarchy_no_mem	6.8%	0.0%
dsip	0.1%	0.0%	paj_raygentop_hierarchy_no_mem	12.5%	0.4%
elliptic	8.0%	3.0%	paj_top_hierarchy_no_mem	16.6%	0.1%
ex1010	8.7%	5.7%	rs_decoder_2	25.1%	3.0%
ex5p	6.2%	1.7%	sv_chip0_hierarchy_no_mem	4.9%	0.3%
frisc	7.4%	0.2%	sv_chip1_hierarchy_no_mem	9.3%	2.4%
misex3	6.5%	0.6%	sv_chip2_hierarchy_no_mem	10.5%	0.7%
pdc	11.2%	1.3%			
s298	7.1%	0.7%	**Average:**	12.1%	1.6%
s38417	22.3%	3.1%			
s38584.1	17.0%	0.2%			
seq	6.5%	0.7%			
spla	12.8%	0.3%			
tseng	8.8%	0.6%			
Average:	11.7%	1.6%			

observe that only 1.6% of LUTs need to be full 6-LUTs to achieve optimal depth. The data in Table I revealed that in most cases, optimal depth can be achieved without *any* pure 6-LUTs. Yet, observe that no circuit has a value of 0 in the "Alternative mapping" column of Table IV. This is due to our cost function that only *prefers* to use extended 5-LUTs, and is therefore heuristic.

In summary, we suggest that a heterogeneous architecture with a fraction of pure 6-LUTs and a fraction of extended 5-LUTs may be viable. Very few pure 6-LUTs are needed in the architecture, perhaps 5% at most.

V. CONCLUSIONS AND FUTURE WORK

We proposed a family of FPGA logic element architectures inspired by the AIG network representation used in modern logic synthesis research. The logic element is an extended LUT, which contains a L-LUT along with M cascaded AND or MUX gates on its output. Results show that that a $\{5,1\}$-MUX extended LUT provides performance close to a 6-LUT, yet has silicon area close to that of a 5-LUT. We believe our work should keenly interest commercial vendors whose logic blocks are based on 6-LUTs. Higher logic density can be achieved by exchanging some or all of the 6-LUTs with extended 5-LUTs, with little negative impact on circuit delay.

It is worth recalling an early work published in 1992 by Chung and Rose that considered mapping circuits into multiple LUTs that were hard-wired together in specific configurations [24]. One sample architecture considered in that work was two cascaded 4-LUTs – the output of one LUT hard-wired to an input of a second LUT. The observation that modern FPGAs do not incorporate such hard-wired LUTs is perhaps reflective of the difficulty in mapping to such architectures. In our work, the logic element architecture is driven by the netlist representation which greatly simplifies mapping.

Finally, in this work, mapping was performed directly on netlists produced by technology independent transformation scripts. Future work will involve exploration of technology independent transformations to encourage creation of netlist topologies that can be accom,odated by the extended LUT element.

ACKNOWLEDGEMENTS

The authors thank Alan Mishchenko at UC Berkeley for providing the source code for the most recent ABC framework.

REFERENCES

[1] J. Rose, R. Francis, D. Lewis, and P. Chow, "Architecture of field-programmable gate arrays: the effect of logic block functionality on area efficiency," *IEEE JSSC*, vol. 25, no. 5, pp. 1217–1225, Oct 1990.
[2] *Virtex-5 FPGA Data Sheet*, Xilinx, Inc., San Jose, CA, 2007.
[3] *Stratix-III FPGA Family Data Sheet*, Altera, Corp., San Jose, CA, 2008.
[4] T. Ahmed, P. Kundarewich, J. Anderson, B. Taylor, and R. Aggarwal, "Architecture-specific packing for Virtex-5 FPGAs," in *ACM/SIGDA Int'l Symposium on FPGAs*, Monterey, CA, 2008, pp. 5–13.
[5] A. Mishchenko, R. Brayton, J. Jiang, and S. Jang, "Scalable don't care based logic optimization and resynthesis," in *ACM Int'l Symposium on Field Programmable Gate Arrays*, Monterey, CA, 2009, pp. 151–160.
[6] J. Anderson and Q. Wang, "Improving logic density through synthesis-inspired architecture," in *IEEE International Conference on Field Programmable Logic and Applications*, Prague, Czech Republic, 2009, pp. 105 – 111.
[7] R. Francis, J. Rose, and K. Chung, "Chortle: A technology mapping program for lookup table-based field programmable gate arrays," in *ACM/IEEE DAC*, 1990, pp. 613–619.
[8] J. Cong and Y. Ding, "Flowmap: An optimal technology mapping algorithm for delay optimization in look-up-table based FPGA designs," *IEEE Transactions on CAD*, vol. 13, no. 1, pp. 1–12, 1994.
[9] M. Schlag, J. Kong, and P. Chan, "Routability-driven technology mapping for lookup table-based FPGAs," *IEEE Transactions on CAD*, vol. 13, no. 1, pp. 13–26, 1994.
[10] J. Cong, C. Wu, and E. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," in *ACM/SIGDA Int'l Symposium on FPGAs*, 1999, pp. 29–35.
[11] "ABC – a system for sequential synthesis and verification," *http://www.eecs.berkeley.edu/∼alanmi/abc/*, 2009.

[12] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis," in *ACM/IEEE DAC*, 2006, pp. 532–536.
[13] A. C. Ling, J. Zhu, and S. D. Brown, "Delay driven AIG restructuring using slack budget management," in *ACM/IEEE Great Lakes Symposium on VLSI*, 2008, pp. 163–166.
[14] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *IEEE/ACM Int'l Con. on CAD*, 2007.
[15] A. Ling, D. Singh, and S. Brown, "FPGA PLB architecture evaluation and area optimization techniques using boolean satisfiability," *IEEE Trans. on CAD*, vol. 26, no. 7, pp. 1196–1210, July 2007.
[16] A. Kennings, K. Vorwerk, A. Kundu, V. Pevzner, and A. Fox, "FPGA technology mapping with encoded libraries and staged priority cuts," in *ACM/SIGDA Int'l Symp. on FPGAs*, 2009, pp. 143–150.
[17] J. Jacob and A. Mishchenko, "Unate decomposition of Boolean functions," in *Int'l Workshop on Logic Synthesis*, 2001, pp. 66–71.
[18] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: enabling a general and efficient FPGA mapping solution," in *ACM Int'l Symposium on FPGAs*, 1999, pp. 29–35.
[19] V. Manohararajah, S. Brown, and Z. Vranesic, "Heuristics for area minimization in LUT-based FPGAs," in *International Workshop on Logic and Synthesis*, 2004, pp. 14–21.
[20] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, M. Fang, and J. Rose, "VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling," in *ACM/SIGDA Int'l Symp. on FPGAs*, 2009, pp. 133–142.
[21] Altera Corp., "Quartus university interface program," *http://www.altera.com/education/univ/research/unv-quip.html*, 2009.
[22] A. Mishchenko, R. Brayton, and S. Jang, "Global delay optimization using structural choices," in *ACM/SIGDA International Symposium on FPGAs*, Monterey, CA, 2010, pp. 181–184.
[23] S. Jang, B. Chan, K. Chung, and A. Mishchenko, "WireMap: FPGA technology mapping for improved routability," in *ACM Int'l Symp. on FPGAs*, 2008, pp. 47–55.
[24] K. Chung and J. Rose, "TEMPT: Technology mapping for the exploration of FPGA architectures with hard-wired connections," in *IEEE/ACM DAC*, Anaheim, CA, 1992, pp. 361–367.

Jason H. Anderson (S'96-M'05) received the B.Sc. degree in computer engineering from the University of Manitoba, Winnipeg, MB, Canada, in 1995 and the Ph.D. and M.A.Sc. degrees in electrical and computer engineering from the University of Toronto (U of T), Toronto, ON, Canada, in 2005 and 1997, respectively. He is an Assistant Professor with the Department of Electrical and Computer Engineering (ECE), U of T. In 1997, he joined the field-programmable gate array (FPGA) implementation tools group at Xilinx, Inc., San Jose, CA. From 2005 to 2008, he managed groups at Xilinx focused on strategic research and development projects. He became a Principal Engineer at Xilinx in 2007. He joined the ECE Department at U of T in 2008. His research interests include all aspects of computer-aided design (CAD) and architecture for FPGAs.

Dr. Anderson was a recipient of the Ross Freeman Award for Technical Innovation, the highest innovation award given by Xilinx, for his contributions to the Xilinx placer technology in 2000. Since joining the U of T faculty, he has twice received awards for excellence in undergraduate teaching, in 2009 and 2010. He has authored numerous papers in refereed conferences and journals, and holds over twenty issued U.S. patents. He serves on the technical program committees of various conferences, including the ACM International Symposium on Field Programmable Gate Arrays and the IEEE International Conference on Field Programmable Technology.

Qiang Wang (S'91-M'99) received the M.A.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Toronto (U of T), Toronto, ON, Canada, in 1993 and 1999, respectively.

Dr. Wang is currently a Principal Engineer in the Wireless R & D Department at Huawei Technologies (U.S.A.), Santa Clara, CA, where he is working on baseband ESL designs. From 2002 to 2010, he was with the field-programmable gate array (FPGA) implementation tools group at Xilinx Inc., San Jose, CA, where he developed placement and other physical design tools for Virtex and Spartan series FPGAs and was involved in the development of new FPGA architectures. From 1999 to 2002, he was a member of the FPGA core group at Lattice Semiconductor Corp., San Jose, CA, where he participated in the development of Lattices first FPGA family.

Dr. Wang has served on the technical program committee of the ACM International Symposium on Field Programmable Gate Arrays. He has authored a number of papers and currently holds six issued U.S. patents.

Chirag Ravishankar (S'11) received the B.A.Sc. degree from the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada in 2010. He is currently pursuing the M.A.Sc. degree at the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada.

His research interests include field-programmable gate array architectures and CAD algorithms. Specifically, he is interested in logic synthesis and technology mapping algorithms from the perspective of area and power reduction. He is also interested in parallel implementations to reduce the run-time of CAD tools.

Mr. Ravishankar was the recipient of the Undergraduate Student Research Award (USRA) from the Natural Sciences and Engineering Research Council (NSERC) of Canada in 2009.