Area-Efficient FPGA Logic Elements: Architecture and Synthesis

Jason H. Anderson

Dept. of ECE, University of Toronto
Toronto, ON Canada
e-mail: janders@eecg.toronto.edu

Qiang Wang

Xilinx, Inc.
San Jose, CA USA
e-mail: qiangw@xilinx.com

Abstract— We consider architecture and synthesis techniques for FPGA logic elements (function generators) and show that the LUT-based logic elements in modern commercial FPGAs are over-engineered. Circuits mapped into traditional LUT-based logic elements have speeds that can be achieved by alternative logic elements that consume considerably less silicon area. We introduce the concept of a *trimming input* to a logic function, which is an input to a K-variable function about which Shannon decomposition produces a cofactor having fewer than $K-1$ variables. We show that trimming inputs occur frequently in circuits and we propose low-cost asymmetric FPGA logic element architectures that leverage the trimming input concept, as well as some other properties of a circuit's AND-inverter graph (AIG) functional representation. We describe synthesis techniques for the proposed architectures that combine a standard cut-based FPGA technology mapping algorithm with two straightforward procedures: 1) Shannon decomposition, and 2) finding non-inverting paths in the circuit's AIG. The proposed architectures exhibit improved logic density versus traditional LUT-based architectures with minimal impact on circuit speed.

I. Introduction

Look-up-tables (LUTs) have been the mainstay of FPGA logic blocks since the invention of FPGAs in the mid-1980s. A K-LUT is a single-output memory with K address lines that can implement *any* Boolean function that uses up to K variables. The earliest FPGAs used 4-LUTs, established as the best LUT size to maximize area-efficiency [1]. State-of-the-art FPGAs are oriented towards speed. Interconnect in FPGAs is slow as compared with custom ASICs, due to the presence of programmable routing switches and the overheads imposed by programmability. Modern FPGAs therefore use 6-LUTs, which lead to fewer levels of interconnect and thereby provide higher speed [2, 3].

Relative to custom ASICs, FPGAs consume up to 35× more silicon area for implementing a given function [4]. Higher logic density is a key goal for FPGAs in pursuit of closing the gap with custom ASICs. Higher density equates with lower cost, shorter wirelengths and lower power. A known issue with the use of 6-LUTs in logic blocks is under-utilization. Many LUT functions in mapped circuits simply do not require 6 inputs, potentially leading to low logic density. To counter this, the commercial vendors add extra outputs onto their LUTs – a straightforward modification due to the nature of a LUT's implementation in hardware, which is a tree structure. The LUTs in modern FPGAs are thus made *fracturable* into smaller LUTs. LUTs in the Xilinx Virtex-6 FPGA can implement any single 6-variable logic function, or any two functions that together use up to 5 distinct variables [2]. The 6-LUT in Altera's Stratix IV FPGA offers even more flexibility, including the ability to implement two separate 4-variable functions [3].

We explore new FPGA logic element architectures, which can offer improved density over the 6-LUTs present in modern commercial FPGAs. Our architectures represent a new way of improving logic density, as compared with the fracturable multi-output LUTs in today's commercial FPGAs. Our elements contain smaller LUTs and yet they deliver most if not all of the benefit afforded by larger LUTs, while retaining the area associated with the smaller LUTs.

The genesis of the proposed architectures is rooted in observations made about the synthesis netlist, which is an AND-inverter graph (AIG). In particular, we observe that logic functions in circuits represented using AIGs frequently have a *trimming input*: Shannon decomposition about the input produces a "small" cofactor (uses few variables). The trimming input property permits the use of low-cost logic elements that require less silicon area than LUTs. We present straightforward techniques to map circuits into the proposed architectures. While recent work on technology mapping and logic synthesis has focused on reducing the number of LUTs needed to implement circuits [5, 6], our work is unique in that we take an architectural approach wherein we use logic synthesis concepts to influence the design of the logic element architecture itself. Our experimental results demonstrate that with the proposed logic element architectures, silicon area can be reduced without any appreciable impact to circuit delay – a result we believe will keenly interest FPGA architects and vendors.

The remainder of this paper is organized as follows: Section II introduces relevant background material on FPGA logic elements, technology mapping and synthesis techniques for FPGAs. Our logic element architectures are described in Section III, as are the approaches used to map circuits into them. An experimental evaluation is given in Section IV. Conclusions appear in Section V.

II. Background

A. Traditional Logic Element: LUT

The thrust of our work is a new approach for reducing the silicon area consumed by FPGA logic elements, while at the same time maintaining functional flexibility and also circuit speed. The LUTs in today's FPGAs are quite costly, consuming silicon area exponentially proportional to the number of LUT inputs (K). A K-LUT is a hardware implementation of a truth table and contains 2^K SRAM cells (one SRAM cell for each of the 2^K minterms of K Boolean variables) and 2^{K-1} 2-to-1 multiplexers organized in a tree structure (to "select" the truth table row and steer an SRAM cell's content to the LUT output). A 6-LUT, such as those used in the Xilinx Virtex-6 [2] FPGA, contains 64 SRAM cells and 63 2-to-1 multiplexers. A 5-LUT, on the other hand, has only 32 SRAM cells – half the area of a 6-LUT.

B. Technology Mapping

For the purpose of technology mapping, the combinational part of a logic circuit can be represented as a Boolean network, which is a directed acyclic graph (DAG), $G(V, E)$, where each node, $v \in V$, represents a logic function and an edge, $e \in E$, represents a dependency between logic functions. Primary inputs (PIs) of the Boolean

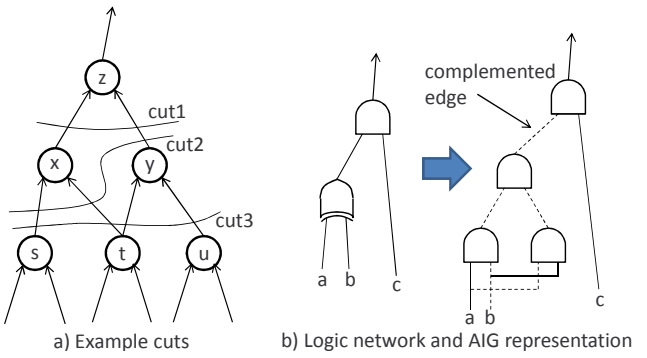

Fig. 1. Cuts and AND-inverter graph (AIG) representation.

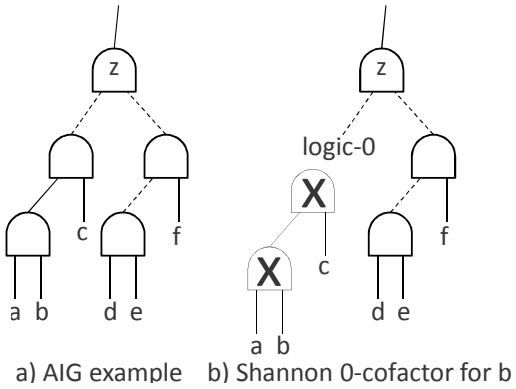

Fig. 2. Finding Shannon cofactors using AIGs.

network have an in-degree of 0 (no fanins); primary outputs (POs) have an out-degree of 0 (no fanouts). The *fanin cone* of a node v is the sub-graph of G comprising v and all of its predecessors (both immediate and transitive predecessors).

Modern FPGA technology mappers are based on the idea of finding the *K-feasible cuts* for the nodes of the Boolean network. A cut, C, for a node v is a partition, (V,$\overline{V}$), of the fanin cone of v such that $v \in \overline{V}$ and the restriction that no node in $\overline{V}$ lies in the fanin cone of any node in V. Fig. 1(a) illustrates three different 3-feasible cuts for a node z. Taking $cut3$ as an example, $\overline{V} = \{x, y, z\}$. For any cut C, we use $Inputs(C)$ to represent the set of nodes in V that drive a node in $\overline{V}$. For $cut3$ in Fig. 1(a), $Inputs(cut3) = \{s, t, u\}$. We say that C is K-feasible if $|Inputs(C)| \leq K$. Since a K-LUT can implement any K variable function, the logic functionality of $\overline{V}$ in a K-feasible cut, $C = (V, \overline{V})$, can be realized in a single LUT. There is a one-to-one correspondence between finding all of the K-feasible cuts for a node and finding all the K-LUT mappings for the node. Computationally efficient techniques to compute the set of all K-feasible cuts for all network nodes are well-known [7, 8].

Given that one can find all K-feasible cuts for each node in the Boolean network, the high-level technology mapping flow is as follows: 1) the network is traversed in topological order and a "best" cut is selected for each node. The best cut is normally selected using a cost function that ranks cuts according to physical metrics: area, delay, and power consumption. 2) After cut selection, the network is traversed in reverse topological order: LUTs in the mapped network are introduced corresponding to the best cut for each node.

C. ABC Framework and the AIG Representation

We conduct our research using the ABC synthesis framework, developed by Mishchenko at UC Berkeley [9]. In ABC, the Boolean network representation of the circuit contains only 2-input AND gates and inverters – a data structure known as an AND-inverter graph (AIG). Fig. 1(b) shows a logic circuit and its AIG representation. Observe that inverters are not represented explicitly in the AIG, but rather as attributes on edges between the AND gates. Despite the simplicity of AIGs, the current best published results on FPGA synthesis and mapping have been produced using the synthesis and mapping algorithms implemented within the ABC framework (e.g. [6, 10]).

The ABC framework incorporates a cut-based FPGA technology mapper that has been shown to produce results on par with any competing mapper [11]. The mapper uses the notion of *priority cuts* for each node of the AIG, where, instead of storing *all* K-feasible cuts for each node (as was done in prior mappers), only a limited set of highly ranked cuts are stored, saving memory and run-time. Despite the pruning of cuts, no appreciable quality degradation was observed [11]. We use the priority cuts mapper as a comparative baseline in this paper.

D. Shannon Decomposition

Our architecture and mapping approach are based on the well-known Shannon's decomposition theorem, which we briefly review here. Using Shannon's theorem, any Boolean function, g, of n variables, $g = f(x_1, x_2, ..., x_i, ...x_n)$, can be decomposed with respect to one of its variables, x_i, and written as the logical OR of two subfunctions:

$$g = x_i \cdot f(x_1, x_2, ..., 1, ..., x_n) + \overline{x_i} \cdot f(x_1, x_2, ..., 0, ..., x_n) \quad (1)$$

where $f(x_1, x_2, ..., 1, ..., x_n)$ is called the 1-cofactor of f with respect to variable x_i, and $f(x_1, x_2, ..., 0, ..., x_n)$ is called the 0-cofactor. A shorthand notation for the 1- and 0-cofactors is g_{x_i} and $g_{\overline{x_i}}$, respectively. In this paper, we use $|g_{x_i}|$ to represent the number of variables in a cofactor (in this case, the 1-cofactor). We refer to this as the *size* of the cofactor.

"Non-inverting paths" in the AIG can be used to compute Shannon cofactors of logic functions. Non-inverting paths are chains of AND gates in the AIG connected through uncomplemented (i.e. "true") edges. Fig. 2(a) gives an example AIG for a function: $z = \overline{(a \cdot b \cdot c)} \cdot (\overline{\overline{d \cdot e} \cdot f})$. Suppose that we wish to compute the 0-cofactor about input b. Setting b to logic-0 propagates up the AIG through non-inverting edges(s) resulting in AND gate outputs evaluating to logic-0. In essense, AND gates are "eliminated" from the AIG, namely, those marked with **x** in Fig. 2(b), producing a reduced AIG. The 0-cofactor with respect to b is: $z_{\overline{b}} = (\overline{0}) \cdot (\overline{\overline{d \cdot e} \cdot f}) = \overline{f} + d \cdot e$.

III. Logic Element Architectures and Mapping

In this section, we first introduce the proposed logic element architectures, and then describe technology mapping techniques used to map circuits into them.

A. Architectures

We introduce our first logic element architecture using the example function z of Fig. 2(a). Considering the Shannon decomposition with respect to variable e, we have cofactors: $z_e = \overline{(a \cdot b \cdot c)} \cdot (\overline{\overline{d} \cdot f})$ and $z_{\overline{e}} = \overline{(a \cdot b \cdot c)} \cdot \overline{f}$. Observe that z_e is a function of 5 variables, whereas, $z_{\overline{e}}$ is a function of only 4 variables, i.e. $|z_{\overline{e}}| = 4$. In such cases, we do not need a full 6-input LUT to implement the logic function. We can use a considerably smaller block. In particular, instead of using a 6-LUT, we can implement the logic function in the logic element shown in Fig. 3(a), containing a 5-LUT, a 4-LUT and a 2-to-1 multiplexer. Carrying on with the example, input $i6$ of the element in Fig. 3(a) can be tied to variable e and the 5-LUT can be used to implement z_e, while the 4-LUT implements $z_{\overline{e}}$. We refer to function z

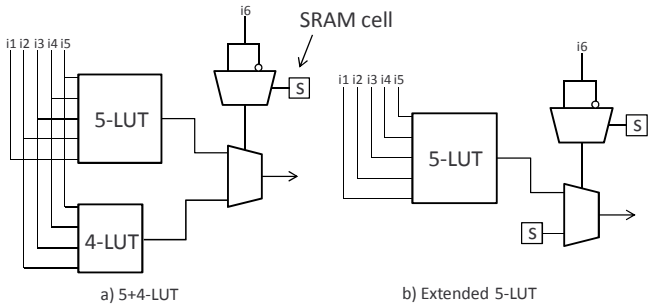

Fig. 3. Asymmetric 5+4-LUT and extended 5-LUT logic element architectures.

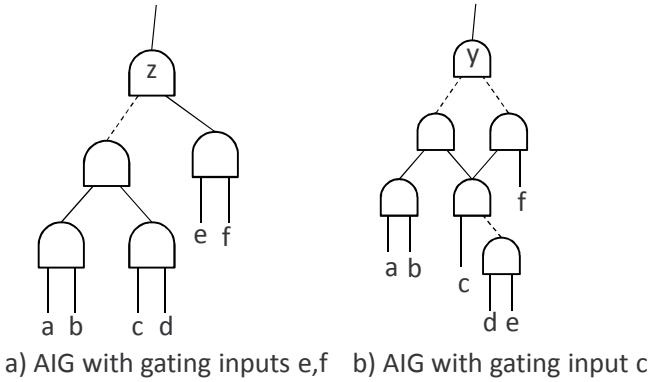

Fig. 4. AIGs with gating inputs.

as having a *trimming input* (e) due the presence of the small cofactor. Shannon decomposition about e produces a cofactor where some input variables are "trimmed" away. We formally characterize a trimming input as follows:

Definition: A k-variable logic function, $g = f(x_1, ..., x_k)$, has a trimming input x_i if Shannon decomposition about x_i produces a 0- or 1-cofactor having fewer than $k-1$ variables, i.e., $|g_{x_i}| < k-1$ and/or $|g_{\overline{x_i}}| < k-1$.

Observe that the select input to the multiplexer in Fig. 3(a) has programmable inversion controlled by an SRAM configuration cell. The programmable inversion permits handling of cases where *either* the 0-cofactor or the 1-cofactor is the smaller cofactor. The structure in Fig. 3(a) uses $32 + 16 + 1 = 49$ SRAM configuration cells, as compared with 64 cells for a 6-LUTs – a ∼23% area savings. We refer to the element in Fig. 3(a) as the *5+4-LUT* logic element architecture.

To understand the prevalence of trimming inputs to logic functions in real circuits, we mapped 5 representative circuits into 6-LUTs using the priority cuts mapper described in [11]. We then analyzed the LUTs in the mapping solutions that used all 6 of their inputs; i.e., the LUTs that implement 6-variable logic functions. Fig. 5(a) illustrates the extent to which 6-variable logic functions in LUTs have trimming inputs. The circuit name is shown on the horizontal axis; the percentage of 6-LUTs with a trimming input is shown on the vertical axis. The data shows that overwhelmingly, 6-variable logic functions in LUT-mapped circuits have at least one trimming input. The full computational power of a 6-LUT is not needed for the majority of LUTs in circuits, suggesting it may indeed be possible to reduce logic element size without compromising mapped depth.

Before introducing the second logic element architecture, we define the concept of a *gating input* to a logic function. A gating input to a logic function is an input whose logic-0 or logic-1 state forces the

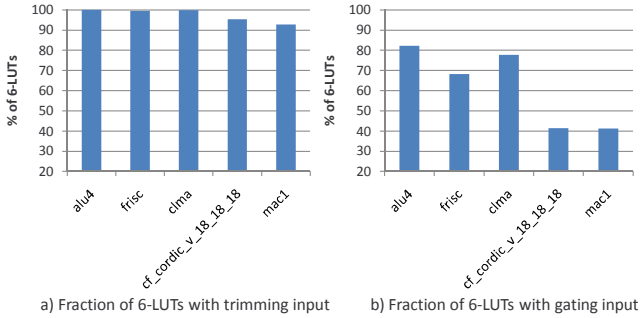

Fig. 5. Fraction of 6-input LUTs in mapped designs that a) have a trimming input, or b) have a gating input.

function to evaluate to either logic-0 or logic-1. That is, one of the two logic states of a gating input has a controlling effect on the overall function – when the gating input is in a particular logic state, the remaining inputs are *don't care*. Gating inputs can be identified through Shannon decomposition:

Definition: A k-variable logic function, $g = f(x_1, ..., x_k)$, has a gating input if Shannon decomposition about one of its inputs, x_i, produces either a 0- or 1-cofactor having zero variables, i.e., either $|g_{x_i}| = 0$ or $|g_{\overline{x_i}}| = 0$. Input x_i is called a gating input to g.

The second logic element architecture we consider is shown in Fig. 3(b). The element is more restrictive than the *5+4-LUT* element as it requires the presense of a gating input to implement a 6-variable function. We refer to the element as an *extended 5-LUT* – a term we introduced in our prior work [12]. To understand this element, consider the AIG example of Fig. 4(a). Observe that AIG inputs e and f are gating inputs which have non-inverting paths to the root node. When either e or f is logic-0, the function is forced to logic-0. The 6-variable function of Fig. 4(a) can be realized in an extended 5-LUT, where either variable e or f is attached to input $i6$ and the SRAM configuraton cell feeding the `MUX` data input is set to logic-0. The programmable inversion on the `MUX` select input allows either state of a gating input signal to be the "forcing" state.

A different style of gating input example is given in Fig. 4(b). In this case, the logic function of the AIG is: $y = \overline{(a \cdot b \cdot c \cdot \overline{d \cdot e})} \cdot \overline{(c \cdot \overline{d \cdot e} \cdot f)}$. Applying De Morgan's theorem to the clauses yields: $y = (\overline{a} + \overline{b} + \overline{c} + d \cdot e) \cdot (\overline{c} + d \cdot e + \overline{f})$. The literal $\overline{c}$ appears in both clauses and therefore, input c is a gating input that causes the function to evaluate to logic-1 (when c is logic-0). The 0-cofactor with respect to c is: $y_{\overline{c}} = 1$ and therefore: $|y_{\overline{c}}| = 0$. Note that the extended 5-LUT here has broader application versus that described in our prior work [12]; the prior work only considered gating inputs that force a function to logic-0.

We analyzed the pervasiveness of gating inputs in LUT-mapped circuits. Taking the same approach as above, examining 6-input LUTs in mapping solutions produced by [11], Fig. 5(b) shows the fraction of 6-LUTs that have a gating input for the same circuits considered in Fig. 5(a). For two of the five circuits, about 80% of 6-LUTs have a gating input. Comparing Fig. 5(a) with Fig. 5(b), we see that as expected, LUTs with gating inputs are not as common as LUTs with trimming inputs. While LUTs with gating inputs must by definition have a trimming input, the converse is not true. LUTs with a trimming input do not necessarily have a gating input. Nevertheless, Fig. 5(b) illustrates that 6-input LUTs implementing logic functions with gating inputs are common in some circuits.

With Shannon decomposition in hand as a tool to easily identify logic functions with trimming inputs and gating inputs in AIGs, we can target a variety of low-cost logic element architectures that are

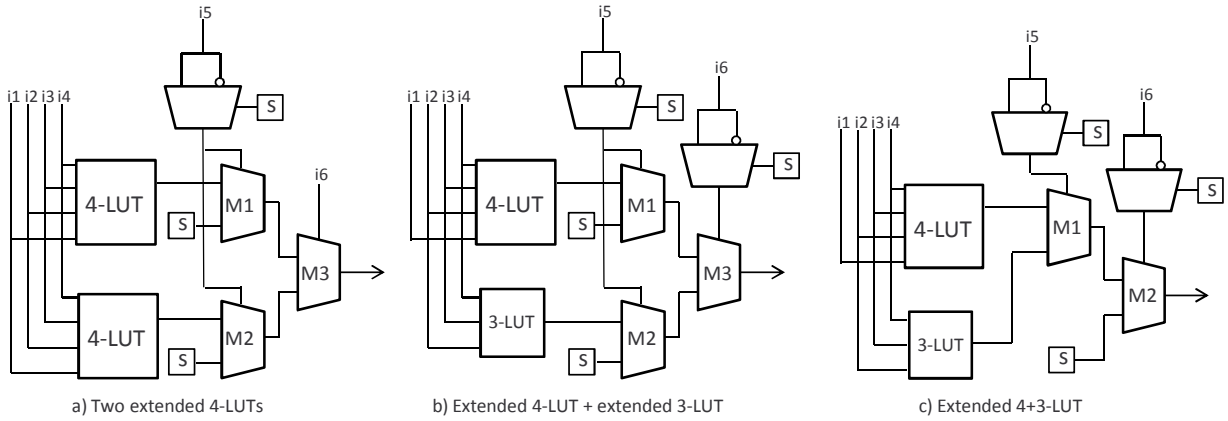

Fig. 6. Low-cost logic element architectures.

TABLE I
NUMBER OF SRAM CONFIG. CELLS IN LOGIC ELEMENT ARCHITECTURES.

Architecture	SRAM cells	Ratio vs. 6-LUTs
6-LUT	64	1.00
5-LUT	32	0.50
5+4-LUT	49	0.77
Extended 5-LUT	34	0.53
Two extended 4-LUTs	35	0.55
Extended 4-LUT + extended 3-LUT	28	0.44
Extended 4+3-LUT	27	0.42

alternatives to 6-LUTs. Fig. 3 and Fig. 6 show the logic element architectures considered in this paper. All of the elements use 6 inputs, making them interchangeable in a fixed FPGA routing architecture. The elements in Fig. 3 were explained above. Fig. 6(a) shows a third element we call *two extended 4-LUTs*, which comprises two extended 4-LUTs feeding the data inputs of a 2-to-1 multiplexer (M3). Programmable inversion is not needed on the select input, $i6$, to the multiplexer since its data inputs are fed by symmetric logic structures.

Fig. 6(b) depicts an even lower-cost architecture: an *extended 4-LUT + extended 3-LUT*. It is similar to the architecture in Fig. 6(a), with the main change being that one of the extended 4-LUTs has been exchanged with an extended 3-LUT. In Fig. 6(b), programmable inversion is required for input $i6$, as it drives the select input of a `MUX` (M3) fed by asymmetric logic structures. Observe that in Figs. 6(a) and 6(b) either the true or inverted form of input signal $i5$ is used to drive the select inputs of *both* multiplexers: M1 and M2. We evaluated the utility of independent polarity control for the select signals on M1 and M2 (which requires an extra `MUX` and SRAM configuration cell), however, the extra flexibility did not affect the results significantly.

Finally, Fig. 6(c) shows the *extended 4+3-LUT* logic element architecture. In this architecture, a 4-LUT and a 3-LUT drive a `MUX` (M1) whose select input is received from $i5$, which in turn drives a second (deeper) `MUX` (M2) whose select input is received from $i6$. One of the data inputs of the deep `MUX` is fed by an SRAM configuration cell. Table I shows the number of SRAM configuration cells in the logic element architectures considered, as well as the ratio of the number of SRAM cells versus 6-LUTs. The number of SRAM configuration cells is a reasonable proxy for the silicon area cost of each architecture. The smallest logic element architecture uses nearly 60% fewer SRAM cells than a 6-LUT.

B. Technology Mapper Implementation

Despite the complexity and variety of logic element architectures, it is straightforward to technology map circuits into them using the concepts above, namely, Shannon decomposition and gating inputs. Consider a K-feasible cut C and let g represent the Boolean function corresponding to the cut. Depending on which of the logic element architectures in Fig. 3 and Fig. 6 is being targeted, we qualify C using the requirements below. A cut C that does not meet the requirements for the target architecture is discarded.

1. **5+4-LUT [Fig. 3(a)]:**
   (a) If $|Inputs(C)| \leq 5$, C is qualified.
   (b) If $|Inputs(C)| = 6$, C is qualified iff $\exists\ i \in Inputs(C)$ such that $|g_i| \leq 4$ or $|g_{\overline{i}}| \leq 4$.
2. **Extended 5-LUT [Fig. 3(b)]:**
   (a) If $|Inputs(C)| \leq 5$, C is qualified.
   (b) If $|Inputs(C)| = 6$, C is qualified iff $\exists\ i \in Inputs(C)$ such that i is a gating input (i.e. $|g_i| = 0$ or $|g_{\overline{i}}| = 0$).
3. **Two extended 4-LUTs [Fig. 6(a)]:**
   (a) If $|Inputs(C)| \leq 5$, C is qualified.
   (b) If $|Inputs(C)| = 6$, C is qualified iff $\exists\ i \in Inputs(C)$ such that: 1) the number of distinct variables shared by both g_i and $g_{\overline{i}}$ is ≤ 4; or, 2) $\exists\ j \in Inputs(C)$, where $j \neq i$ and j is a gating input to functions g_i and $g_{\overline{i}}$.
4. **Extended 4-LUT + extended 3-LUT [Fig. 6(b)]:**
   (a) If $|Inputs(C)| \leq 4$, C is qualified.
   (b) If $|Inputs(C)| = 5$, C is qualified iff $\exists\ i \in Inputs(C)$ such that $|g_i| \leq 3$ or $|g_{\overline{i}}| \leq 3$.
   (c) If $|Inputs(C)| = 6$, C is qualified iff $\exists\ i \in Inputs(C)$ such that: 1) the number of distinct variables shared by both g_i and $g_{\overline{i}}$ is ≤ 4 and where $|g_i| \leq 3$ or $|g_{\overline{i}}| \leq 3$; or, 2) $\exists\ j \in Inputs(C)$, where $j \neq i$ and j is a gating input to both g_i and $g_{\overline{i}}$, and where $|g_i| \leq 4$ or $|g_{\overline{i}}| \leq 4$.
5. **Extended 4+3-LUT [Fig. 6(c)]:**
   (a) If $|Inputs(C)| \leq 4$, C is qualified.
   (b) If $|Inputs(C)| = 5$, C is qualified iff $\exists\ i \in Inputs(C)$ such that $|g_i| \leq 3$ or $|g_{\overline{i}}| \leq 3$.

(c) If $|Inputs(C)| = 6$, C is qualified iff $\exists\ i \in Inputs(C)$ such that: 1) the number of distinct variables shared by both g_i and $g_{\overline{i}}$ is ≤ 4, and where $|g_i| \leq 3$ or $|g_{\overline{i}}| \leq 3$; or, 2) i is a gating input to g, and if h represents the non-constant cofactor of g about i[1], then $\exists\ j \in Inputs(C)$, where $j \neq i$, and where $|h_j| \leq 3$ or $|h_{\overline{j}}| \leq 3$.

We altered the priority cuts mapping algorithm in ABC to honor the requirements above when targeting a circuit into the corresponding architecture. Illegal cuts were discarded during the cut generation phase. ABC also provides area-reducing post-mapping routines based on the area-flow concept [13], which we customized to target the new architectures.

The logic element architectures in Figs. 3 and 6 have 6-inputs. Such elements could be directly interchanged with the 6-LUTs in a modern commercial architecture (such as Xilinx Virtex-6), without any changes to the routing architecture[2]. Hence, by analyzing these architectures, we aim to answer the question: Can logic density be improved through a change to the logic element architecture within an existing routing fabric (i.e. a fabric designed to handle 6-input elements)?

Beyond 6-input logic element architectures, we also study *analogous* 7-input architectures. For example, the 7-input logic element architecture analogous to the *5+4-LUT* architecture above is a *6+5-LUT* architecture. 7-LUTs will certainly deliver improved depth vs. 6-LUTs, at a higher area cost. With the 7-input logic element architecture investigation, we seek to answer the question: Can the performance of 7-LUTs be achieved using logic element architectures that require the silicon area of 6-LUTs or even less?

IV. Experimental Study

We evaluate logic element architectures according to two metrics: 1) the number of logic elements needed to implement a circuit, and 2) the depth of the mapping (number of logic elements on the longest combinational path). Metric #1 can be combined with the data in Table I to estimate the relative logic density of each architecture. Metric #2 is a proxy for circuit speed.

We use two sets of benchmark circuits. The first set is the "standard" 20 benchmarks that are widely used in FPGA CAD and architecture research. The second set are the 13 largest circuits from the popular VPR 5.0 FPGA placement, routing, and architecture evaluation framework [14]. The VPR 5.0 circuits were synthesized from Verilog to BLIF using Altera's Quartus 9.1 tool. Altera's QUIP (Quartus University Interface Program) flow was used to produce BLIF after HDL elaboration and technology independent optimization.

We use ABC's `resyn2` script for technology independent optimization prior to technology mapping. We also experimented with other technology independent optimization scripts that come packaged with ABC, however, we found they produced slightly worse depth results, on average. For each circuit, `resyn2` followed by technology mapping was run 6 times, and the best result achieved across all runs was taken as the data point for the circuit, where mapped depth was the primary ranking criteria and the number of logic elements was the secondary critera. The priority cuts mapper [11] used as the baseline was executed in depth mode; it optimizes area (number of logic elements) on non-critical paths as a secondary criteria.

Table II gives results for 6-input architectures and 5-LUTs (for comparison). The top-half of the table presents results for the standard 20 benchmarks; the bottom-half of the table gives results for the VPR 5.0 circuits. The bottom rows of the table give the average data across all circuits, and the ratios relative to 6-LUTs. Let us begin with the right-most column of the table, for 5-LUTs, we see that depth is 14.3% higher than 6-LUTs, on average, and element count is 15% higher. The depth gap between 5 and 6-LUTs is considerable, which explains the vendors' motivation for moving to 6-LUTs. Observe that the depth gap is wider for the VPR 5.0 circuits (18%), than for the standard 20 circuits (12%). We observed such differences between the two circuit sets for all architectures considered.

Moving on to the proposed *5+4-LUT* architecture (column labeled *5+4-LUT*), results show that both depth and element count are virtually identical to 6-LUTs. Element count is less than 1% higher when *5+4-LUT* elements are used vs. 6-LUTs. The *5+4-LUT* architecture uses only 49 SRAM cells vs. 64 cells in a 6-LUT – a considerable reduction in silicon area. The data in Table II suggests that such area savings can be realized without significant increase to depth or element count. Prior work has shown that when circuits are mapped to 6-LUTs, less than 40% of the LUTs in the mapping solutions use all 6 inputs [15]; the balance are small LUTs that are straightforward to map into the proposed *5+4-LUT* architecture. The data in Table II shows that the full functional flexibility (and silicon cost) of LUTs is not required for the majority of functions in logic circuits – LUTs appear to be over-engineered for their intended purpose.

Continuing from left-to-right in Table II, we next consider the *extended 5-LUT* architecture. On average, across all circuits, depth and element count are increased by 6% and 8% vs. 6-LUTs, respectively. *Extended 5-LUTs* have roughly the same silicon area as a 5-LUT, and yet they deliver most of the depth benefit of 6-LUTs. The next architecture, *two extended 4-LUTs*, appears to be superior to the *extended 5-LUT* architecture and the two use roughly the same silicon area. With the *two extended 4-LUTs* architecture, depth and element count are within ∼5-6% of 6-LUTs. The *two extended 4-LUTs* architecture offers reasonably good depth, while using only about half the silicon area of a 6-LUT.

Examining the results for the two low-cost logic element architectures that combine a 4-LUT and a 3-LUT, the two element architectures appear to be similar from both the depth and logic element count perspectives. The *extended 4+3-LUT* requires just 42% of the SRAM cells of a 6-LUT, yet circuits mapped into the architecture have just 6% higher depth than a 6-LUT. We believe the logic element architecture may be especially useful in low-cost product lines, where silicon area and cost is a primary factor, and a slight performance degradation is tolerable.

Table III gives a summary of results for 7-input logic element architectures and 6-LUTs (for comparison). The individual circuit-by-circuit results could not be included, due to page limitations. Table III gives, for each architecture, the normalized geometric mean of depth and element count, across all benchmark circuits. The right-most column gives the number of SRAM configuration cells in each logic element architecture. Normalization is with respect to 7-LUTs. In general, the architectural trends are the same as those observed with 6-input architectures. Note, however, that the difference in mapped depth is more pronounced between 6 and 7-LUTs versus 5 and 6-LUTs. 6-LUT mappings have 22% more depth than 7-LUT mappings; whereas, 5-LUT mappings have 14% more depth than 6-LUT mappings. Also observe that the *6+5-LUT* architecture results in 5% more depth vs. 7-LUTs; however, the analogous 6-input architectures were equivalent from the depth angle. Broadly, we observe that the majority of the depth benefit associated with a move to 7-LUTs can be achieved using a logic element architecture such as *extended 5-LUT + extended 4-LUT* that is in fact smaller than the 6-LUTs deployed in FPGAs today.

A "back-of-the-envelope" approach can be used to estimate the overall logic density of the 6-input architectures. We approximate the area of each element architecture by its number of SRAM configura-

[1] $h = g_i$ if $|g_i| > 0$, else $h = g_{\overline{i}}$.

[2] In fact, FPGA tile die size and routing wire lengths (and delay) can be reduced through the use of logic elements with lower silicon area.

TABLE II
LOGIC ELEMENT COUNT AND DEPTH RESULTS FOR 6-INPUT LOGIC ELEMENT ARCHITECTURES AND 5-LUTS.

	6-LUTS		5+4-LUT		Ext. 5-LUT		Two ext. 4-LUTs		Ext. 4 + Ext. 3		Ext. 4+3-LUT		5-LUTs	
CIRCUIT	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**	**DEP**	**#LEs**
alu4	5	774	5	774	6	751	6	760	6	734	6	751	6	866
apex2	6	868	6	868	6	906	6	906	6	901	6	906	7	990
apex4	5	752	5	752	5	766	5	769	5	762	5	766	6	840
bigkey	3	579	3	579	3	691	3	689	3	691	3	692	3	806
clma	10	2795	10	2767	10	3231	10	2917	10	3044	10	3237	12	3179
des	4	691	4	725	5	848	5	819	5	867	5	845	5	948
diffeq	8	636	8	643	9	722	9	723	9	733	9	730	9	757
dsip	3	689	3	691	3	691	3	691	3	694	3	692	3	692
elliptic	10	1797	10	1797	11	1828	11	1832	11	1843	11	1842	12	1873
ex1010	6	2452	6	2456	6	2505	6	2517	6	2500	6	2505	7	2755
ex5p	5	504	5	468	5	497	5	518	5	512	5	517	5	594
frisc	13	1735	13	1740	13	1832	13	1834	13	1835	13	1835	14	1842
misex3	5	723	5	723	5	745	5	746	5	741	5	751	6	811
pdc	7	1948	7	1980	7	2025	7	2071	7	2035	7	2025	7	2495
s298	8	641	8	641	8	665	8	648	8	663	8	666	9	731
s38417	7	2567	7	2551	7	2998	7	2917	7	3076	7	3024	8	3068
s38584.1	6	2287	6	2287	6	2553	6	2431	6	2509	6	2554	7	2688
seq	5	780	5	786	5	813	5	812	5	809	5	805	5	908
spla	6	1670	6	1698	7	1671	6	1804	7	1689	6	1800	7	1906
tseng	8	647	8	651	8	680	8	679	8	703	8	703	9	692
GEOMEAN (STD 20 CIRCUITS):	6.08	1075.16	6.08	1076.38	6.32	1143.29	6.27	1138.85	6.32	1144.99	6.27	1153.47	6.82	1241.43
RATIO VS 6-LUTS:			1.000	1.001	1.039	1.063	1.031	1.059	1.039	1.065	1.031	1.073	1.123	1.155
cf_cordic_v_18_18_18	9	3822	9	3866	10	4203	10	4114	10	4593	10	4585	11	4461
cf_fir_24_16_16	18	10929	18	10452	18	11843	18	11793	18	13605	18	13630	18	11668
des_perf	3	3264	3	4019	4	4314	4	4081	4	5044	4	4881	4	4959
mac1	17	1959	17	2019	18	2166	18	2108	18	2487	18	2481	21	2215
mac2	31	6906	31	7116	32	7396	32	7267	32	8406	32	8415	39	7491
oc54	22	2393	22	2385	22	2600	22	2523	22	2761	22	2780	24	2726
paj_boundtop_hierarchy_no_mem	4	1294	4	1294	4	1371	4	1327	4	1371	4	1371	6	1376
paj_raygentop_hierarchy_no_mem	16	6314	16	6193	17	6693	17	6518	17	7276	17	7281	17	6677
paj_top_hierarchy_no_mem	33	32967	33	31614	34	35196	34	33758	34	39470	34	39440	34	35016
rs_decoder_2	11	1649	12	1751	13	1919	12	1870	14	2043	14	1989	14	2265
sv_chip0_hierarchy_no_mem	5	12615	5	12684	6	12970	6	12909	6	13728	6	13727	6	12955
sv_chip1_hierarchy_no_mem	8	25473	8	24831	9	26984	9	26425	9	29564	9	29564	9	26882
sv_chip2_hierarchy_no_mem	18	46440	19	48099	19	50062	19	50070	21	55333	21	55443	19	50104
GEOMEAN (VPR 5.0 CIRCUITS):	11.88	6384.02	12.01	6505.71	12.93	7001.29	12.85	6837.28	13.10	7689.56	13.10	7658.75	13.98	7233.55
RATIO VS 6-LUTS:			1.011	1.019	1.088	1.097	1.081	1.071	1.102	1.205	1.102	1.200	1.176	1.133
GEOMEAN (ALL CIRCUITS):	7.92	2168.87	7.95	2186.57	8.38	2334.51	8.32	2307.36	8.42	2424.54	8.38	2431.56	9.05	2485.73
RATIO VS 6-LUTS:			1.004	1.008	1.058	1.076	1.051	1.064	1.064	1.118	1.059	1.121	1.143	1.146

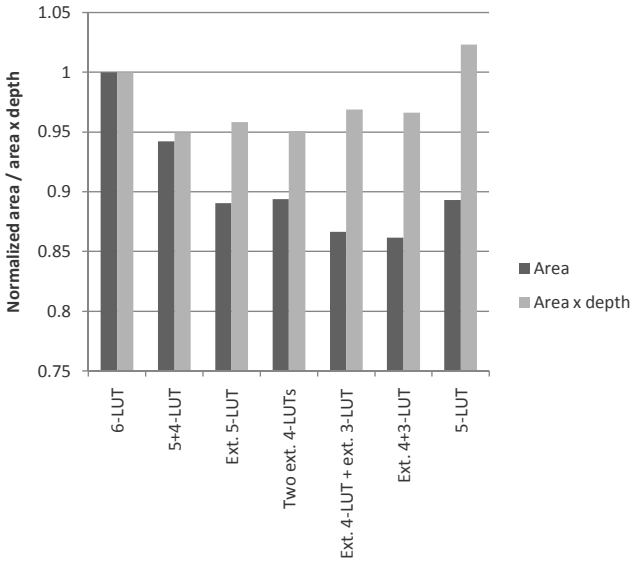

Fig. 7. Approximate area and area-depth product for 6-input logic element architectures.

TABLE III
SUMMARY OF RESULTS FOR 7-INPUT ARCHITECTURES AND 6-LUTS.
GEOMEAN RESULTS ACROSS ALL CIRCUITS, NORMALIZED TO 7-LUTS.

ARCHITECTURE	DEPTH	#LEs	SRAM cells
7-LUTs	1.00	1.00	128
6+5-LUT	1.05	1.03	97
Extended 6-LUT	1.07	1.07	66
Two extended 5-LUTs	1.07	1.06	67
Extended 5-LUT + extended 4-LUT	1.08	1.09	52
Extended 5+4-LUT	1.08	1.08	51
6-LUTs	1.22	1.09	64

tion cells (see Table I)[3]. We combine the SRAM cell count with the element count data in Table II. We estimate that in a modern architecture, such as Xilinx Virtex-6, LUTs consume 25% of the core tile area, with the balance being interconnect (∼50%), registers and other logic (∼25%). If we take a baseline 6-LUT architecture to have a tile area of 1 unit[2], the relative tile area of any 6-input architecture can be computed as: $f \times (75\% + 25\% \times s)$, where f is the ratio of the number of logic elements required to implement circuits vs. 6-LUTs (from Table II), and s is the ratio of the number of SRAM configuration cells in the logic element architecture vs. a 6-LUT (column 3 of Table I). Fig. 7 shows normalized area and area-depth product for 6-input logic

[3]Estimating area using the number of 2-to-1 multiplexers in each logic element architecture produces similar results.

element architectures. Considering area alone, the smallest architectures (containing a 4-LUT and a 3-LUT) provide 14% higher logic density vs. 6-LUTs. The *5+4-LUT* and *two extended 4-LUTs* logic element architectures offer the best area-depth product – an ∼5% win over the baseline 6-LUT architecture.

V. Conclusions and Future Work

Silicon area-efficiency, speed, and power are three metrics where there remains a significant gap between FPGAs and ASICs. In this paper, we described new FPGA logic element architectures, and associated synthesis methods, that deliver improved area-efficiency relative to the LUTs present in commercial FPGAs today, with minimal impact on speed, and most likely a positive impact on power. A key contribution of this work is the observation that the logic functions implemented by LUTs in circuits frequently have *trimming* inputs – Shannon decomposition about such an input produces a small cofactor with fewer variables. The trimming input property of logic functions was used to inspire the design of new logic element architectures, many of which provide superior area-efficiency to the LUTs in today's commercial FPGAs. As an example, while a 6-LUT uses 64 SRAM configuration cells, one of our architectures (*5+4-LUT*) uses only 49 SRAM cells, and yet, produces mapping solutions of equal depth to 6-LUTs, and also equal element count.

A direction for future work is to combine our architecture/CAD techniques with the recent work on *don't care*-based FPGA technology mapping [6]. Leveraging don't cares in mapping has been proven to offer considerable reductions in the number of LUTs needed to implement circuits. It is unknown whether LUTs produced by the don't care-based mapping methods also exhibit the trimming input property we used in designing our area-efficient logic element architectures. It is worth exploring the extent to which the area-efficiency gains produced by the two different approaches are additive.

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-6 FPGA Data Sheet*, Xilinx, Inc., San Jose, CA, 2010.

[3] *Stratix-IV FPGA Family Data Sheet*, Altera, Corp., San Jose, CA, 2010.

[4] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Trans. On Computer Aided Design*, vol. 26, no. 2, pp. 203–215, Feb. 2007.

[5] D. Chen and J. Cong, "DaoMap: a depth-optimal area optimization mapping algorithm for FPGA designs," in *IEEE Int'l Conf. on Computer Aided Design*, 2004, pp. 752–759.

[6] 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.

[7] 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.

[8] J. Cong, C. Wu, and E. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," in *ACM Int'l Symposium on FPGAs*, 1999, pp. 29–35.

[9] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis," in *ACM DAC*, 2006, pp. 532–536.

[10] A. Mishchenko, R. Brayton, and S. Jang, "Global delay optimization using structural choices," in *ACM/SIGDA Int'l Symp. on FPGAs*, Monterey, CA, 2010, pp. 181–184.

[11] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *IEEE Int'l Conf. on Computer Aided Design*, 2007, pp. 354–361.

[12] J. Anderson, and Q. Wang, "Improving logic density through synthesis-inspired architecture," in *IEEE Int'l Conf. on Field Programmable Logic and Applications*, 2009, pp. 105–111.

[13] V. Manohararajah, S. Brown, and Z. Vranesic, "Heuristics for area minimization in LUT-based FPGAs," in *Int'l Workshop on Logic and Synthesis*, 2004, pp. 14–21.

[14] 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 Int'l Symp. on Field Programmable Gate Arrays*, 2009, pp. 133–142.

[15] S. Jang, B. Chan, K. Chung, and A. Mishchenko, "WireMap: FPGA technology mapping for improved routability," in *AMC Int'l Symp. on Field Programmable Gate Arrays*, 2008, pp. 47–55.