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. Similar mapped circuit speed can be realized by blocks 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 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 ∈ V, represents a logic function and an edge, e ∈ E, represents a dependency between logic functions. Primary inputs (PIs) of the Boolean 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, ∂v), of the fanin cone of v such that v ∈ ∂v and the restriction that no node in ∂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, $\overrightarrow{V} = \{x, y, z\}$. For any cut $C$, we use $\text{Inputs}(C)$ to represent the set of nodes in $V$ that drive a node in $\overrightarrow{V}$. For cut3 in Fig. 1(a), $\text{Inputs}(\text{cut3}) = \{s, t, u\}$. We say that $C$ is $K$-feasible if $|\text{Inputs}(C)| \leq K$. Since a $K$-LUT can implement any $K$ variable function, the logic functionality of $\overrightarrow{V}$ in a $K$-feasible cut, $C = (\overrightarrow{V}, \overrightarrow{V})$, 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 the 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 ABC (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, reviewed here. Using Shannon’s theorem, any Boolean function, $g$ of $n$ variables, $g = f(x_1, x_2, \ldots, x_n)$, can be decomposed with respect to one of its variables, $x_i$, and written as the logical OR of two subfunctions:

$$g = x_i \cdot f(x_1, x_2, \ldots, 1, \ldots, x_n) + \overline{x_i} \cdot f(x_1, x_2, \ldots, 0, \ldots, x_n)$$  \hspace{1cm} (1)$$

where $f(x_1, x_2, \ldots, 1, \ldots, x_n)$ is called the 1-cofactor of $f$ with respect to variable $x_i$, and $f(x_1, x_2, \ldots, 0, \ldots, x_n)$ is called the 0-cofactor. A shorthand notation for the 1- and 0-cofactors is $g_x$ and $g_{\overline{x}}$ respectively. In this paper, we use $[g_x]$ 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: $z = (a \cdot b \cdot c) \cdot (d \cdot e \cdot f)$. 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 essence, 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: $z_{\overline{b}} = (0) \cdot (d \cdot e \cdot f) = \overline{z} + d \cdot e$.

III. LOGIC ELEMENT ARCHITECTURES AND MAPPING

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: $z_e = (a \cdot b \cdot c) \cdot (d \cdot \overline{f})$ and $z_{\overline{e}} = (a \cdot b \cdot c) \cdot \overline{f}$. 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, 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 $z$ 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, $g = f(x_1, \ldots, x_k)$, 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 $\sim 23\%$ area savings. We refer to the element in Fig. 3(a) as the 5+4-LUT logic element architecture.
produces either a 0- or 1-cofactor having zero variables, i.e., either
Fig. 3(b). The element is more restrictive than the
fine the concept of a
LUTs in circuits, suggesting it may indeed be possible to reduce logic
computational power of a 6-LUT is not needed for the majority of
inputs. The circuit name is shown on the horizontal axis; the per-
the extent to which 6-variable logic functions in LUTs have trimming
LUTs that implement 6-variable logic functions. Fig. 5(a) illustrates
ing the priority cuts mapper described in [11]. We then analyzed the
prevalence of trimming inputs to logic functions in real circuits, we mapped 5 representative circuits into 6-LUTs us-
ing the priority cuts mapper described in [11]. To understand this element, con-
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 \( |\overline{g_{x_i}}| = 0 \). 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 presence 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
configuration 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: \( y = (a \cdot b \cdot c \cdot d \cdot e \cdot \overline{f}) \).
Applying De Morgan’s theorem to the clauses yields:
\( y = (\overline{\overline{a} + \overline{b} + \overline{c} + d \cdot e \cdot \overline{f}}) \). The literal \( \overline{f} \) 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: \( |g_c| = 1 \) and therefore: \( |\overline{g_c}| = 0 \). 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 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 range of low-cost logic element architectures that are an alternative 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),

![Fig. 3. Asymmetric 5+4-LUT and extended 5-LUT logic element architectures.](image)

![Fig. 4. AIGs with gating inputs.](image)

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

Despite the complexity and range 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 Figs. 3 and 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)| ≤ 5, C is qualified.
   (b) If |Inputs(C)| = 6, C is qualified iff ∃ i ∈ Inputs(C) such that |gi| ≤ 4 or |gi| = 0.

2. Extended 5-LUT [Fig. 3(b)]:
   (a) If |Inputs(C)| ≤ 5, C is qualified.
   (b) If |Inputs(C)| = 6, C is qualified iff ∃ i ∈ Inputs(C) such that i is a gating input (i.e. |gi| = 0 or |gi| = 0).

3. Two extended 4-LUTs [Fig. 6(a)]:
   (a) If |Inputs(C)| ≤ 5, C is qualified.
   (b) If |Inputs(C)| = 6, C is qualified iff ∃ i ∈ Inputs(C) such that: 1) the number of distinct variables shared by both g_i and g_j is ≤ 4; or, 2) ∃ j ∈ Inputs(C), where j ≠ i and j is a gating input to functions g_i and g_j.

4. Extended 4+3+LUT [Fig. 6(b)]:
   (a) If |Inputs(C)| ≤ 4, C is qualified.
   (b) If |Inputs(C)| = 5, C is qualified iff ∃ i ∈ Inputs(C) such that |gi| ≤ 3 or |gi| ≤ 3.
   (c) If |Inputs(C)| = 6, C is qualified iff ∃ i ∈ Inputs(C) such that: 1) the number of distinct variables shared by both g_i and g_j is ≤ 4 and where |gi| ≤ 3 or |gi| ≤ 3; or, 2) ∃ j ∈ Inputs(C), where j ≠ i and j is a gating input to both g_i and g_j, and where |gi| ≤ 4 or |gi| ≤ 4.

5. Extended 4+3-LUT [Fig. 6(c)]:
   (a) If |Inputs(C)| ≤ 4, C is qualified.
   (b) If |Inputs(C)| = 5, C is qualified iff ∃ i ∈ Inputs(C) such that |gi| ≤ 3 or |gi| ≤ 3.
   (c) If |Inputs(C)| = 6, C is qualified iff ∃ i ∈ Inputs(C) such that: 1) the number of distinct variables shared by both g_i and g_j is ≤ 4, and where |gi| ≤ 3 or |gi| ≤ 3; or, 2) i is a gating input to g_i, and if h represents the non-constant cofactor of g about i, then ∃ j ∈ Inputs(C), where j ≠ i, and where |h_j| ≤ 3 or |h_j| ≤ 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. 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?

\[ h = g_i \text{ if } |g_i| > 0, \text{ else } h = g_j \]

\[ \text{In fact, FPGA tile die size and routing wire lengths (and delay) can be reduced through the use of logic elements with lower silicon area.} \]
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 criteria. 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. Moreover, the data 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 configuration cells (see Table I)3. We combine the SRAM cell count with the

3Estimating area using the number of 2-to-1 multiplexers in each logic element architecture produces similar results.
element count data in Table II. We estimate that in a modern architecture, such as Xilinx Virtex-6, LUTs consume 25% of the core 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 unit², the relative tile area of any 6-input architecture can be computed as: \( f = (75\% + 25\% \times \text{ratio}) \), 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 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-LUT 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 smaller cofactor with fewer variables. The trimming input property of logic functions can improve 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 49 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