IMPROVING LOGIC DENSITY THROUGH SYNTHESIS-INSPIRED ARCHITECTURE

Jason H. Anderson

Dept. of ECE, Univ. of Toronto
Toronto, ON Canada
email: janders@eecg.toronto.edu

Qiang Wang

Xilinx, Inc.
San Jose, CA USA
email: qiangw@xilinx.com

ABSTRACT

We leverage properties of the logic synthesis netlist to define both a logic element 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. Results show that while circuits mapped to a traditional 5-LUT architecture need 14% more LUTs and have 12% more depth than a 6-LUT architecture, our extended 5-LUT architecture requires only 7% more LUTs and 2.5% 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.

1. INTRODUCTION

For over twenty years, the logic blocks 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 a logic element with superior area-efficiency, as well as a simple mapping strategy for the proposed element.

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].

We propose a new technology mapping scheme and 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, yet they achieve considerably higher logic density as compared with traditional logic blocks. The remainder of the paper is organized as follows: Background and related work appear in Section 2. Our mapping approach and the proposed element architecture are described in Section 3. An experimental study is presented in Section 4. Conclusions and suggestions for future work are offered in Section 5.

2. BACKGROUND

2.1. FPGA Technology Mapping

Research on technology mapping for FPGAs was active in the early 1990s with a wide range of algorithms proposed

978-1-4244-3892-1/09/$25.00 ©2009 IEEE

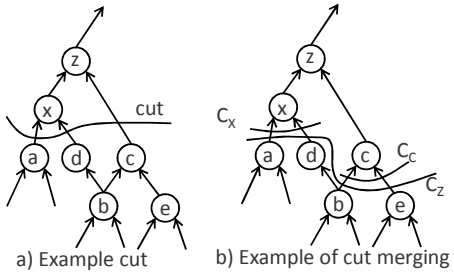

Fig. 1. Illustration of K-feasible cuts.

(e.g. [5, 6]). Recent technology mappers are based on the notion of cuts [7, 8], 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. 1(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. 1(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. 1(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. 1(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 [8, 7] 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. Say that node x has one K-feasible cut, C_X, and node c has one K-feasible cut, C_C, as shown in Figure 1(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 1(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 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 [7] 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.

2.2. The ABC Synthesis Framework

Work on logic synthesis has been reinvigorated by the introduction of the ABC system developed at UC Berkeley [9]. 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. 2. The use of AIGs eases the implementation of many useful logic synthesis transformations (e.g., [10, 11]).

Among other advantages, AIGs have proven to be effective for cut-based LUT mapping. In [12], 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.

We conduct our research using AIGs within the ABC framework and we propose a new logic element architecture and mapping approach. Our logic element contains structures beyond LUTs and experimental results demonstrate the area and performance benefits of the proposed element. It is worth mentioning that Ling et al. considered using SAT-based techniques for mapping into blocks with LUTs and gates [13]. 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 [14]. Both of these prior works considered the mapping problem in isolation, and not from the architecture evaluation perspective.

2.3. LUT Hardware Architecture

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. 3 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,

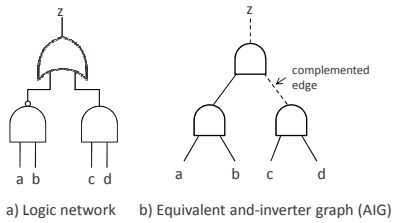

Fig. 2. AIG example.

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.

3. ELEMENT ARCHITECTURE AND MAPPING

Our mapper and architecture leverage a property of the AIG. We illustrate this using the example AIG shown in Fig. 4(a). Two 4-input cuts are shown: $cut0$ and $cut1$, corresponding to LUTs implementing the functions $z = q \cdot i2 \cdot i1 \cdot \overline{i0}$ 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. 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.

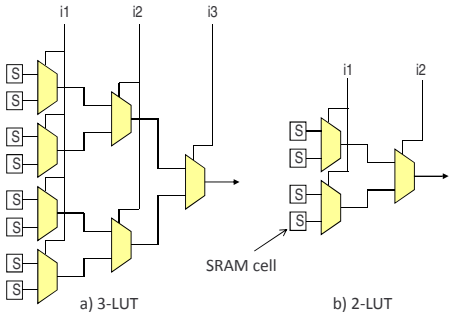

Fig. 3. 2 and 3-input LUTs.

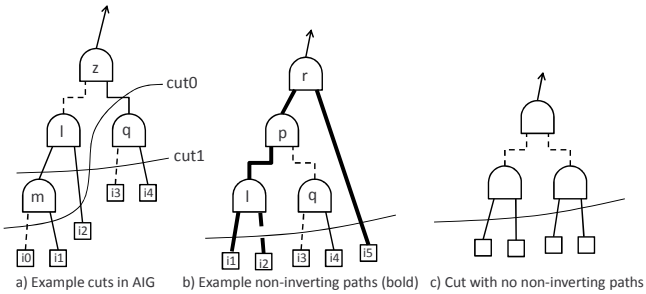

Fig. 4. Cut examples.

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.

Mapping Scheme: 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 accomodated in an architecture with "extended" K-1-LUTs, which require about half the 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 one of three signals. It can be tied to logic-1 for the case of implementing functions that require less than K variables, or it can be tied to the true or complemented form of an input signal. The optional inversion is needed to handle the case of complemented edges crossing the cut, such as $(i2, l)$ in Fig. 4(b).

[1] AND gates driven by cut edges.

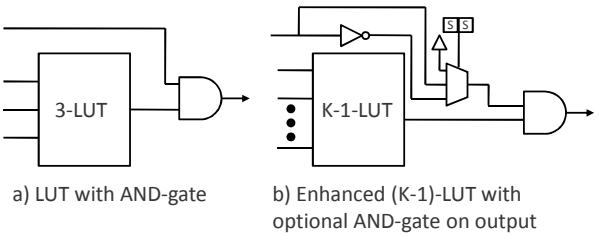

Fig. 5. Extended LUT with additional AND gate.

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.

The obvious question that arises is: What is the impact of restricting the cuts of size K from the area (# of LUTs) 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. Furthermore, aside from establishing a hard restriction on the K-input cuts, we also investigated the impact of using a *soft* preference for K-input cuts having non-inverting paths to study the fraction of LUTs in circuits that need to be mapped onto true K-LUTs, with the remainder being accomodated by extended K-1-LUTs.

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 2:

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

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:

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

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 3, 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:

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

We built a technology mapper based on priority cuts [12] that can operate in one of two ways:

1. Select only from $FCuts(z)$ when choosing the cut for node z during technology mapping – this represents a hard restriction.

2. Prefer to choose cuts from $FCuts(z)$ when choosing the cut for node z only as a secondary criterion, with depth being the primary criterion.

4. EXPERIMENTAL STUDY

We use the mapper in [12] as the baseline mapper to which we compare. It is worth reinforcing that recent work has shown that mapping quality is not compromised by using AIGs compared with other network representations [12]. 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 [15]. We use 20 large combinational and sequential circuits commonly used in academic research. 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. In this work, we applied the `resyn2` script twice to each circuit prior to tech mapping. 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 element in Fig. 5(b).

4.1. Mapping Results

Table 1 shows the mapping results for three cases: 1) mapping into 6-LUTs (the baseline), 2) mapping into extended 5-LUTs, and 3) mapping into 5-LUTs. The ratio row at the bottom of the table compares the ratios of the geometric means to the baseline case. Looking first at the results for 5-LUTs (right-hand columns), observe that on average, depth increases 12%, as compared with 6-LUTs, and 14% more LUTs are needed. While more LUTs are needed, these are 5-LUTs, which are half the die size of 6-LUTs. The considerable depth reduction (12%) is one of the prime reasons commercial vendors have trended to 6-LUTs recently in their high performance product lines.

Turning to the results for extended 5-LUTs (see Table 1), observe that compared to 6-LUTs, depth is increased by only 2.5% and 7.3% more LUTs are needed, on average. For 16

Table 1. Mapping results.

	Baseline (6-LUTs)		Extended 5-LUTs		5-LUTs	
Circuit	**LUTs**	**Levels**	**LUTs**	**Levels**	**LUTs**	**Levels**
alu4	840	5	825	6	949	6
apex2	968	6	1005	6	1095	7
apex4	779	5	821	5	886	6
bigkey	579	3	691	3	804	3
clma	3111	10	3171	11	3490	12
des	820	5	938	5	983	5
diffeq	665	8	772	9	790	9
dsip	689	3	691	3	692	3
elliptic	1884	10	1963	11	1972	12
ex1010	2603	6	2649	6	2900	7
ex5p	540	5	585	5	647	5
frisc	1745	13	1889	13	1901	15
misex3	776	5	831	5	860	6
pdc	2143	7	2295	7	2397	8
s298	660	8	706	8	722	10
s38417	2559	7	2965	7	3042	8
s38584	2270	6	2598	6	2683	7
seq	856	5	878	5	1006	5
spla	1759	7	1865	7	2054	7
tseng	678	8	722	8	729	9
Geomean:	1136.37	6.20	1219.11	6.35	1296.00	6.93
Ratio:			1.073	1.025	1.140	1.119

Table 2. Mapping results: extended 6-LUTs, 7-LUTs.

	Baseline (6-LUTs)		Extended 6-LUTs		7-LUTs	
Circuit	**LUTs**	**Levels**	**LUTs**	**Levels**	**LUTs**	**Levels**
alu4	840	5	754	5	701	5
apex2	968	6	886	6	948	5
apex4	779	5	726	5	690	5
bigkey	579	3	467	2	467	2
clma	3111	10	2909	9	2731	9
des	820	5	817	4	707	4
diffeq	665	8	654	7	623	7
dsip	689	3	911	2	911	2
elliptic	1884	10	1867	9	1831	9
ex1010	2603	6	2378	6	2318	6
ex5p	540	5	498	5	470	4
frisc	1745	13	1662	12	1629	11
misex3	776	5	726	5	682	5
pdc	2143	7	2106	6	1904	6
s298	660	8	618	7	579	7
s38417	2559	7	2532	6	2407	6
s38584	2270	6	2216	6	2052	5
seq	856	5	786	5	748	5
spla	1759	7	1722	6	1650	5
tseng	678	8	687	7	659	7
Geomean:	1136.37	6.20	1091.96	5.55	1042.60	5.32
Ratio:			0.961	0.896	0.917	0.859

of 20 circuits, identical depth was achieved in comparison with 6-LUT mappings. Nearly all of the depth benefit of using 6-LUTs is realized with the proposed logic element. In essence, the proposed logic element delivers the benefit of 6-LUTs, while its area corresponds to that of 5-LUTs. Moreover, not as many extended 5-LUTs as pure 5-LUTs are needed to implement the circuits.

Table 2 shows the results when the baseline mapping remains 6-LUTs, and the comparison is with extended 6-LUTs and pure 7-LUTs. Moving from 6-LUTs to extended 6-LUTs reduces depth by 10% and reduces LUT count by 4%. The data suggests that if vendors added an AND gate to their 6-LUT outputs and then mapped to such extended 6-LUTs, depth would be cut by about 10%, on average. In so doing, the 6-LUT-based blocks would need additional logic block inputs to provide a signal to the AND 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 AND gate. The right columns of the table show that pure 7-LUTs provide a depth benefit of 14% over pure 6-LUTs and reduce LUT count by 8%. Extended 6-LUTs provide nearly all of the depth benefit of 7-LUTs.

We recognize that LUTs in commercial FPGAs are sometimes used for purposes other than implementing logic functions – they are also used for implementing small RAMs and shift registers. We propose that such functions can equally be realized using (smaller) extended LUTs, chained together, as needed.

4.2. Die Area and Delay Impact

Using an extended 5-LUT instead of a 6-LUT will reduce the tile area needed for a logic block. Smaller tiles will reduce wirelengths, interconnect capacitance and delay. As shown in Fig. 6(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. 6(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. This implies that if the original tile area were 1 unit^2, as in Fig. 6(a), the new tile area would be 0.85 units^2. Results in Table 1 demonstrate that 7.3% more extended 5-LUTs are needed vs. 6-LUTs to implement circuits. Consequently, logic density in silicon will scale by $1.073 \times 0.85 = 0.91$, which is roughly a 9% improvement in logic density vs. 6-LUTs.

Assuming a square tile layout, the tile dimensions are reduced from 1×1 to 0.92×0.92, as shown in Fig. 6(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 [16] to pack, place and route circuits into logic blocks containing eight 6-LUTs and flip-flops. The cluster size of eight

Table 3. Mapping results with 6-LUT baseline and alternate cost functions preferring extended 5-LUTs.

	6-LUTs (ABC mapping)	6-LUTs (alterate cost-1)	6-LUTs (alterate cost-2)
Circuit	% of LUTs needing full 6-LUTs	% of LUTs needing full 6-LUTs	% of LUTs needing full 6-LUTs
alu4	20.24	12.53	12.53
apex2	14.57	6.64	0
apex4	39.28	8.66	0
bigkey	39.4	39.4	0
clma	16.84	9.18	9.18
des	21.59	15.62	0
diffeq	42.29	32.81	32.81
dsip	0.29	0.29	0
elliptic	29.91	10.4	10.4
ex1010	43.91	8.63	0
ex5p	33.95	14.63	0
frisc	31	8.3	0
misex3	19.72	8.97	0
pdc	39.2	12.34	0
s298	18.53	11.15	0
s38417	35.43	26.92	0
s38584	32.65	20.6	0
seq	15.67	7.36	0
spla	41.56	13.08	0
tseng	26.03	9.34	0
Average:	28.10	13.84	3.25

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. 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.

4.3. 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? The results of this analysis are shown in Table 3.

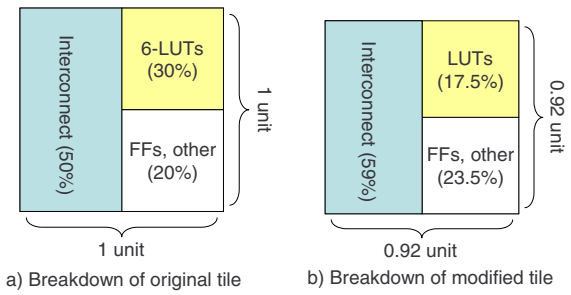

Fig. 6. Estimated tile area impact.

First examine the second column of Table 3 which shows results for the baseline 6-LUT mapping solution [12]. The values in the column give the percentage of *all* LUTs in the mapping solution that require the full functional flexibility of a 6-LUT, and could not be realized with an extended 5-LUT. On average, about 28% of LUTs in the mapping solutions need full 6-LUTs for the baseline mapper.

The third column of the table gives the results for our mapping scheme (described in Section 3) that prefers to use extended 5-LUTs, 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. Observe that, on average, about 13.8% of LUTs in the mapping solutions need full 6-LUTs.

The data in Table 1 revealed that in many cases (16 of 20 circuits), optimal depth can be achieved without *any* pure 6-LUTs. Yet, observe that no circuit has a value of 0 in the third column of Table 3. This is due to our cost function that only *prefers* to use extended 5-LUTs. The fourth column of the table combines the preferred and hard-constrained mappings: if optimal depth can be achieved for a circuit with the hard constraint (i.e. no pure 6-LUTs), a 0 appears for the circuit in the column. Otherwise, the soft-constrained mapping solution (column 3) is taken and some pure 6-LUTs are needed to attain optimal depth. Using this approach, across all circuits, only 3% of LUTs need to be pure 6-LUTs, on average. This constitutes a practical flow, comprised of mapping each circuit twice, first with the hard constraint and then with the soft constraint, and selecting the hard-constrained mapping solution if its depth is optimal.

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 10-20% at most.

5. CONCLUSIONS AND FUTURE WORK

We proposed a logic element architecture and mapper inspired by the AIG network representation used in modern logic synthesis research. The logic element is an extended K-LUT, which has the area of a K-LUT but provides the performance of a K+1-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.

We are extending the research to consider the benefit of having the LUT output drive a chain of `AND` gates, rather than just a single `AND` gate. An example of this is shown in Fig. 7, depicting a 5-LUT with two `AND` gates on its output.

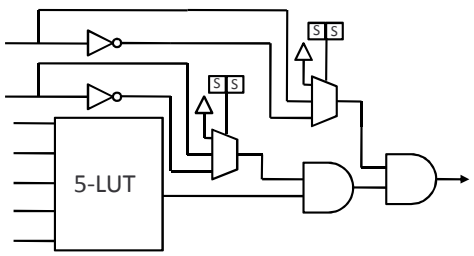

Fig. 7. 5-LUT with AND gate cascade.

Mapping into such an element is a straightforward extension of the approach described in Section 3, except 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. 7, we must identify two non-inverting paths in the AIG. Generalizing this idea, one can consider a family of logic element architectures with a K-LUT followed by m serially-connected AND gates, capable of accomodating a $K + m$-input function, provided the function's AIG representation contains m non-inverting paths. Our initial results for such architectures are promising and show that further improvements to logic density are possible.

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 [17]. 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. As with the "non-inverting paths" concept, perhaps other AIG properties can be discovered to permit easy mapping into non-AND structures, e.g. exclusive-OR, OR, or MUX structures.

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 accomodated by the extended LUT element.

6. 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] 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.

[6] J. Cong and Y. Ding, "Flowmap: An optimal technology mapping algorithm for delay optimization in look-up-table based FPGA designs," *IEEE Trans. on CAD*, vol. 13, no. 1, pp. 1–12, 1994.

[7] M. Schlag, J. Kong, and P. Chan, "Routability-driven technology mapping for lookup table-based FPGAs," *IEEE Trans. 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/SIGDA Int'l Symposium on FPGAs*, 1999, pp. 29–35.

[9] "ABC – a system for sequential synthesis and verification," *http://www.eecs.berkeley.edu/∼alanmi/abc/*, 2009.

[10] 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.

[11] 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.

[12] 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, pp. 227-233.

[13] 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.

[14] 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. 153–150.

[15] 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.

[16] 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.

[17] K. Chung, and J. Rose, "TEMPT: technology mapping for the exploration of FPGA architectures with hard-wired connections," in *ACM/IEEE DAC*, 1992, pp. 361–367.