Title
Optimality study of logic synthesis for LUT-based FPGAs

Journal
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 26(2)

ISSN
0278-0070

Authors
Cong, J
Minkovich, K

Publication Date
2007-02-01

Peer reviewed

Optimality Study of Logic Synthesis for LUT-Based FPGAs

Jason Cong and Kirill Minkovich

***Abstract*—Field-programmable gate-array (FPGA) logic synthesis and technology mapping have been studied extensively over the past 15 years. However, progress within the last few years has slowed considerably (with some notable exceptions). It seems natural to then question whether the current logic-synthesis and technology-mapping algorithms for FPGA designs are producing near-optimal solutions. Although there are many empirical studies that compare different FPGA synthesis/mapping algorithms, little is known about how far these algorithms are from the optimal (recall that both logic-optimization and technology-mapping problems are NP-hard, if we consider area optimization in addition to delay/depth optimization). In this paper, we present a novel method for constructing arbitrarily large circuits that have known optimal solutions after technology mapping. Using these circuits and their derivatives (called Logic synthesis Examples with Known Optimal (LEKO) and Logic synthesis Examples with Known Upper bounds (LEKU), respectively), we show that although leading FPGA technology-mapping algorithms can produce close to optimal solutions, the results from the entire logic-synthesis flow (logic optimization + mapping) are far from optimal. The LEKU circuits were constructed to show where the logic synthesis flow can be improved, while the LEKO circuits specifically deal with the performance of the technology mapping. The best industrial and academic FPGA synthesis flows are around 70 times larger in terms of area on average and, in some cases, as much as 500 times larger on LEKU examples. These results clearly indicate that there is much room for further research and improvement in FPGA synthesis.**

***Index Terms*—Circuit optimization, circuit synthesis, design automation, field-programmable gate arrays (FPGAs), optimization methods.**

I. Introduction

FIELD-PROGRAMMABLE gate arrays (FPGAs) have been gaining momentum as an alternative to application-specific integrated circuits (ASICs). FPGAs consist of programmable logic, input-output (I/O), and routing elements, which can be programmed and reprogrammed in the field to customize an FPGA, enabling it to implement a given application in a matter of seconds or milliseconds. The most common type of programmable-logic element used in an FPGA is called a K-LUT, which is a K-input one-output lookup table (LUT), capable of implementing any K-input one-output Boolean function.

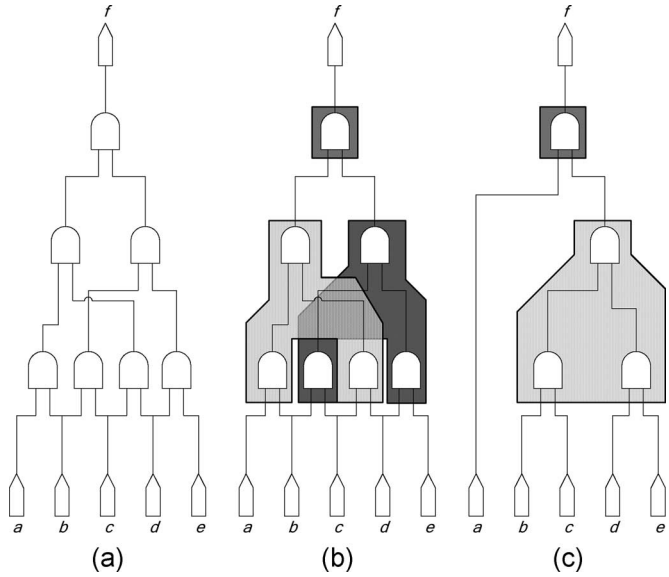


Fig. 1. Possible area-minimal mapping solutions. (a) Original circuit. (b) Mapping solution without logic optimization. (c) Mapping solution with logic optimization.

Given a register transfer level (RTL) design, the typical FPGA synthesis process consists of RTL elaboration, logic synthesis, and the physical design (layout synthesis) [11]. In this paper, we will focus on logic synthesis, which can be broken down into two main steps: logic optimization and technology mapping. Logic optimization transforms the current gate-level network into an equivalent gate-level network more suitable for technology mapping. Technology mapping transforms the gate-level network into a network of programmable cells (in our case, these cells are LUTs) by covering the network with these cells. Several algorithms perform logic optimization during technology mapping. As an example, Fig. 1 shows the difference between mapping algorithms that use logic optimization and those that do not. By examining the logic function of f, we can see that it just takes the logical AND of all its inputs; thus, by manipulating the circuit, we can reduce the mapping solution by one 4-LUT. Since the size of the circuit will be directly proportional to the price of an FPGA that can implement it, the logic-synthesis step will play an integral role in the design flow.

As FPGA technology gained popularity throughout the 1990s, a large amount of work was published that dealt with logic synthesis and/or technology mapping of FPGAs, including Chortle-crf [20], XMap [24], TechMap [31], DAG-map [9],

Manuscript received March 16, 2006; revised July 17, 2006 and September 13, 2006. This work was supported in part by the National Science Foundation under Grant CCF-0306682 and Grant CCF 0430077 and in part by Altera Corporation, Magma Design Automation Inc., and Xilinx Inc. under the California MICRO Program. This paper was recommended by Associate Editor K. Bazargan.

The authors are with the Computer Science Department, University of California, Los Angeles, CA 90095 USA (e-mail: cong@cs.ucla.edu; cory_m@cs.ucla.edu).

Digital Object Identifier 10.1109/TCAD.2006.887922

FlowMap [13], Zmap [16], Cutmap [12], BoolMap [27], and many others. More comprehensive surveys of FPGA synthesis and mapping algorithms are available from [8] and [11]. These mapping algorithms employ many different techniques to achieve their solutions, including dynamic programming, bin packing, BDD-based logic simplification, and cut enumeration, just to name a few. Some of these algorithms focused on delay minimization [14], [20], [24], [29], [31], [35], while others focused on area minimization possible under delay or depth constraints [9], [10], [12], [13], [16], [27], [31], [36]. All of these algorithms were developed over a ten-year period in the 1990s but, after this influx, the amount of new published work began to decrease steadily with only a few novel algorithms emerging in the past few years—such as IMAP [3], Hermes [19], DAOmap [7], and the ABC mapper [1], [38]. To many people, this signaled that FPGA synthesis algorithms had probably hit a plateau. It is natural to then question whether the current logic-synthesis and technology-mapping algorithms for FPGA designs are producing near-optimal solutions. Although there are many empirical studies that compare different FPGA synthesis/mapping algorithms, little is known about how far these algorithms are from the optimal (recall that both logic-optimization and technology-mapping problems are NP-hard if we consider area optimization in addition to delay/depth optimization).

In fact, a similar question was raised a few years ago when placement research slowed down. However, using a set of cleverly constructed examples, called placement examples with known optimal (PEKO), the study in [6] showed surprising results: Wirelengths produced by state-of-the-art placement tools at that time were 1.66–2.53 times the optimal solutions in the worst cases. These results generated a renewed interest in placement research; within two to three years, a large body of papers was published on placement optimality studies (e.g., [15], [17], [18], and [23]), as well as novel placement algorithms (e.g., [4], [5], [22], [30], and [34]). Within three years, the optimality gap on the PEKO examples was reduced to roughly 20% on average [4]. The actual improvement on the IBM (ISPD04) benchmarks was 24% by the mPL placer [33]. This leads one to believe that the improvement on the artificially constructed PEKO examples correlates to some extent the improvement on the "real" (more realistic) examples. This might be due to the fact that a small suboptimality of an algorithm on the real benchmarks often gets magnified into a significant optimality gap on the PEKO benchmarks.

Unfortunately, there is no simple way to extend the ideas of testing placement optimality to logic synthesis because of the inherent differences in the two problems. Therefore, little progress has been made on testing the optimality of logic-synthesis algorithms. The research in [28] presented a method that could only create very small structureless test cases, and they were used to test very primitive mappers. Another method, described in [2], used a SAT solver as an exact logic-synthesis tool for LUT-based FPGAs to determine how much more the circuit area could be reduced by postprocessing the mapping solutions produced by existing mappers. But, the results suggested that current mappers could not be easily improved. This is largely due to the highly localized search algorithm used in this approach (SAT-based optimal logic optimization is applied to logic cones of up to ten inputs).

In this paper, we present a novel method for constructing arbitrarily large circuits that have known optimal solutions after technology mapping, or known upper bound solutions after logic optimization and technology mapping for LUT-based FPGAs. Using these circuits [called Logic synthesis Examples with Known Optimal (LEKO) and Logic synthesis Examples with Known Upper bounds (LEKU)], we show that although leading FPGA technology-mapping algorithms can produce close to optimal solutions, the results from the entire logic-synthesis flow (logic optimization + mapping) are far from optimal. The best industrial and academic FPGA synthesis flows are around 70 times larger in terms of area on average and, in some cases, as much as 500 times larger on LEKU examples. These results clearly indicate that there is much room for further research and improvement in FPGA synthesis.

II. Construction of Benchmarks

A. Construction of LEKO Examples

We present an algorithm for constructing a network $\mathrm{G_n}$ (with n inputs and outputs) of an arbitrarily large size that has a known optimal technology mapping solution. $\mathrm{G_n}$ is constructed in a special way by replicating a small circuit with a known optimal mapping solution into a circuit of any size that also has a known optimal mapping solution. These circuits are called LEKO examples. The building block of our construction is a "core graph" named $\mathrm{C_n}$, with the following properties.

1) It has n inputs and n outputs.
2) Every output is a function of all n inputs.
3) Each internal node of $\mathrm{C_n}$ has exactly two inputs.
4) There exists an optimal (in terms of area/depth) mapping of $\mathrm{C_n}$ into a 4-LUT mapping solution, denoted as $\mathrm{M_n}$, such that $\mathrm{M_n}$ only has 4-LUTs (no 3-LUTs or 2-LUTs). For the $\mathrm{C_5}$ shown in Fig. 2, $\mathrm{M_5}$ has exactly seven 4-LUTs.

This general method can be used to create core graphs of arbitrary size, although in this paper, we will only present how $\mathrm{C_5}$ and $\mathrm{C_6}$ are constructed. These core graphs target the 4-LUT architecture, because it is the simplest of those currently in use. But, the ideas behind our construction can be extended to Altera's Adaptive Logic Module or any sized LUT architecture by only adapting the fourth property to reflect an optimal mapping in that architecture.

The specific $\mathrm{C_5}$ we used to construct our LEKO and LEKU benchmarks is shown in Fig. 2, and the optimality of its technology mapping solution is stated in Theorem 1a and verified in its proof. The core graph $\mathrm{C_6}$ was made similarly to $\mathrm{C_5}$ by following the four properties and slowly modifying the structure until it was hard for the structural-based mappers to map [ABC and DAOmap]; then, the logic was modified to make it difficult for the logic-synthesis tools, like ISE and Quartus, to map. The optimality of $\mathrm{C_6}$'s depth is extracted from the DAOmap result, and the optimality of the ten 4-LUTs needed to map it is proven in Theorem 1b.

Theorem 1a: $\mathrm{C_5}$ has an area-optimal technology mapping solution of seven 4-LUTs.

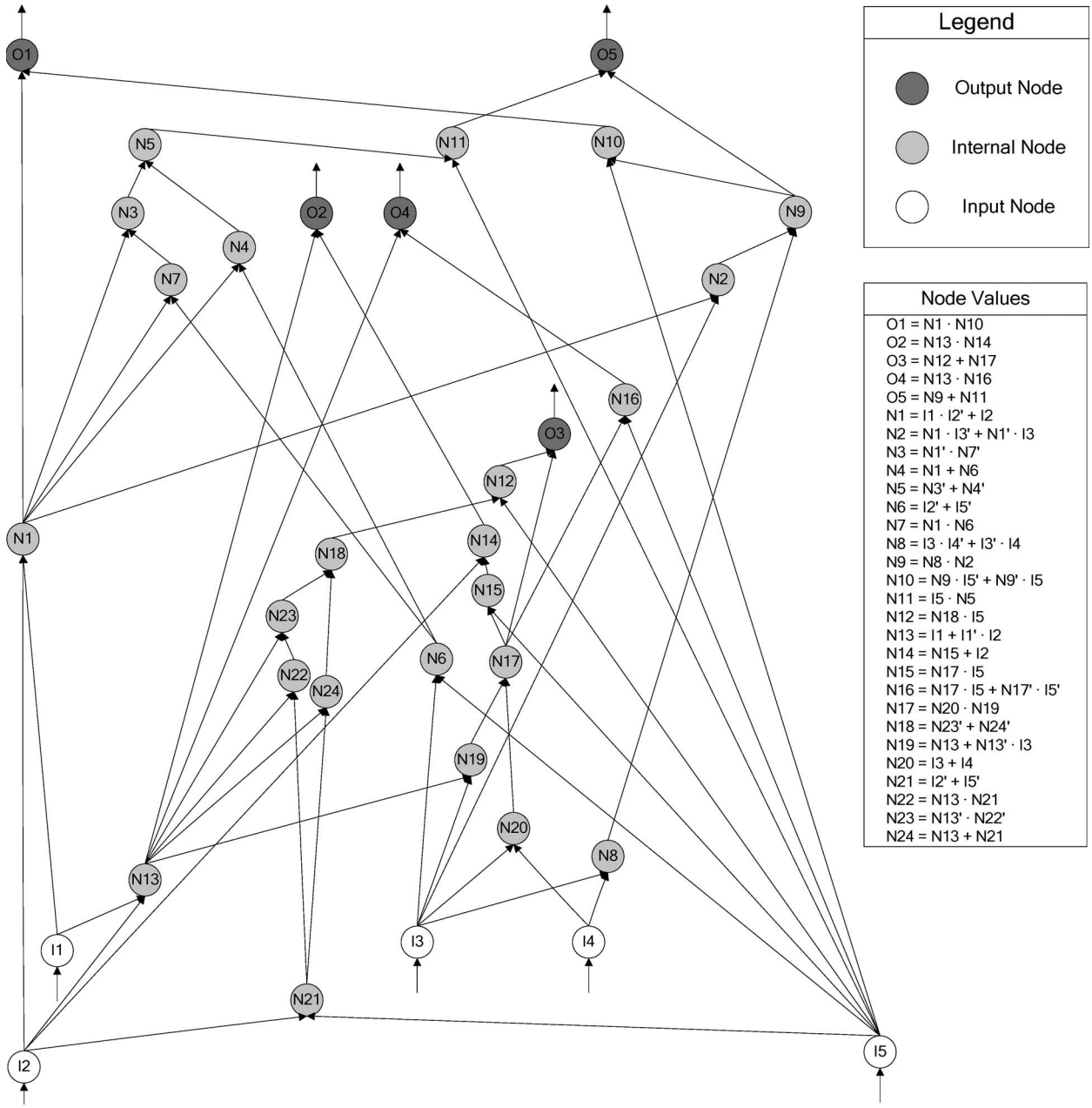


Fig. 2. Example of C_5.

Proof: This is proved using the binate-cover technique, which is able to compute the minimum-area technology mapping solution. In particular, we use the binate-covering solver in SIS [32] using the command "xl_cover - h 0." The binate solver in our case returned a seven 4-LUT solution. The reason that this method cannot be used to prove the optimality of the larger LEKO circuits is because this tool is computationally infeasible for returning an optimal binate-covering solution for any graph with more than 100 logic gates. ■

Theorem 1b: C_6 has an area-optimal technology mapping solution of ten 4-LUTs.

Proof: This can also be proved using the binate cover provided in SIS. But for testing any core graphs with more than six inputs, the binate-cover algorithm would have to be implemented using a current SAT solver. ■

Using this newly created C_n, a LEKO circuit G is created by stacking up C_ns in such a way that from the outputs of G, there is only one way to traverse C_n to get to the inputs. The exact algorithm is presented in Fig. 3, where createLEKO (L) creates a LEKO example with $L \cdot n^{L-1}$ C_ns in L layers. In the algorithm, the $\bigcup$ (union) operator does not disturb the order of the inputs or outputs. For example, when looking at the $A \bigcup B$, we can think of the inputs and outputs as an array of nodes. Then, the index of every input node from A appears before any input node from B in $A \bigcup B$; likewise, the property holds for the output nodes. The Copy operator creates a copy of the network renaming all the nodes, createEdge*(x, y)* just creates an edge from x to y, and $G^{output}[i]$ is the i^{th} output of G (Fig. 3).

Logically, the createLEKO algorithm works as follows. It builds up the graph using layer upon layer of C_ns in order to

```
algorithm createLEKO (L)
input: the number of layers L, output: network G
G = U_{n^{L-1}} copy(C_n) ;
for i = 2,3,…,L do
    currLayer = U_{n^{L-1}} copy(C_n) ;
    for j = 0,1,…,n^L-2 do
        createEdge( G^output[(n · j) mod (n^L -1)] ],
                    currLayer^input[j]);
    end-for
    createEdge( G^output [ n^L -1 ],
                currLayer^input[ n^L -1 ]);
    G = G ⋃ currLayer;
end-for
output G;
```

Fig. 3. createLEKO algorithm.

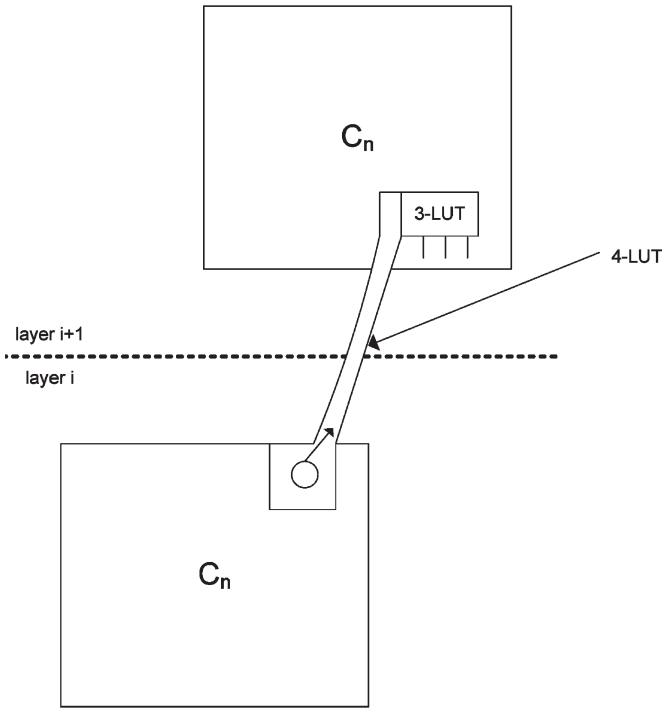


Fig. 4. LUT spanning two layers.

get a LEKO example of L layers. It first creates the bottom layer of $n^{\mathrm{L}-1}$ $\mathrm{C_n}$s, then for each additional layer, it makes $n^{\mathrm{L}-1}$ copies of $\mathrm{C_n}$ and proceeds to connect the outputs of graph G to the inputs of the newly created layer. It spreads out the connections in such a way that for an arbitrary $\mathrm{C_n}$ at the top layer, there exists a path to it from every $\mathrm{C_n}$ at the bottom layer (i.e., every $\mathrm{C_n}$ at the top level is connected to every $\mathrm{C_n}$ at the bottom level). Thus, using this algorithm and any number m, one can create a LEKO circuit having more than m nodes and a known optimal technology mapping solution whose optimality is proved in Theorem 2. By using this method, we were able to construct G_{25} (Fig. 5) by calling createLEKO with two layers and a C_5. G_{36} (Fig. 6) was constructed the same way but used

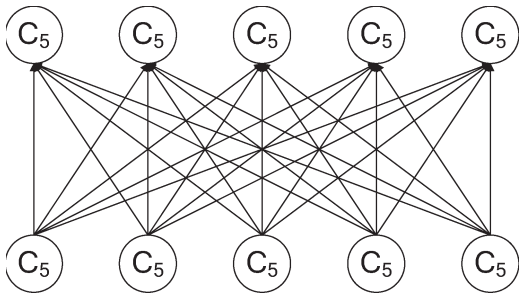


Fig. 5. LEKO(G_{25}).

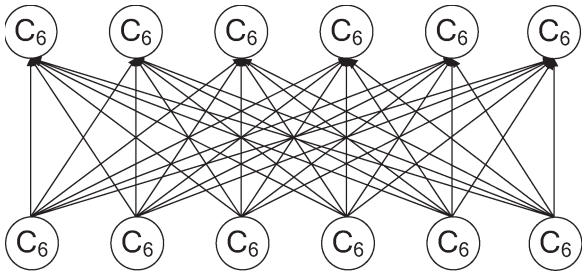


Fig. 6. LEKO(G_{36}).

C_6 instead of C_5. We similarly constructed G_{125} (Fig. 7) with three layers, and we constructed G_{625} with four layers.

Theorem 2: The optimal mapping solution of an arbitrarily sized LEKO circuit without logic optimization is achieved when every $\mathrm{C_n}$ in the circuit is mapped optimally without overlapping any other $\mathrm{C_n}$.

Proof: Now that we have the ability to construct arbitrarily sized LEKO circuits, we can show that this construction creates a circuit G with a known optimal binate cover, which we proved in Theorem 1. Assuming we have an arbitrary LEKO circuit G with L layers, we prove Theorem 2 by induction over the layers of G. Claim 1 will be used in almost all of the other claims as it proves that there are no reconverging paths of $\mathrm{C_n}$s. Claims 2 and 3 will help prove the base case, while Claim 4, working with Claims 2 and 3, helps prove the inductive step. ■

Claim 1: Treelike structure of $\mathrm{C_n}$s (no reconverging paths of $\mathrm{C_n}$s).

Given an arbitrary $\mathrm{C_n}$, x, on the top layer and a LEKO G, starting at any $\mathrm{C_n}$ at the bottom layer, there is only one way to traverse the $\mathrm{C_n}$s to get to x.

Proof: Assume we start at an arbitrary $\mathrm{C_n}$, call it x, on the top layer. From the construction it should be obvious that a path exists from any $\mathrm{C_n}$ at the bottom layer to x (i.e., x is connected to every $\mathrm{C_n}$ on the bottom layer). Now, let us consider the maximum number of $\mathrm{C_n}$s we are connected to after one layer, which is n (since x has n inputs). Similarly, after two layers the maximum number of $\mathrm{C_n}$s that are connected to x is n^2 (since x has n inputs and the $n\mathrm{C_n}$s that feed x's inputs also have n inputs), and the maximum number of $\mathrm{C_n}$s that can reach x at layer L (after L-1 layers) is $n^{\mathrm{L}-1}$. Now, if there were any reconverging paths connecting x to the rest of the $\mathrm{C_n}$s, there would be strictly less than $n^{\mathrm{L}-1}$ $\mathrm{C_n}$s at the bottom layer that

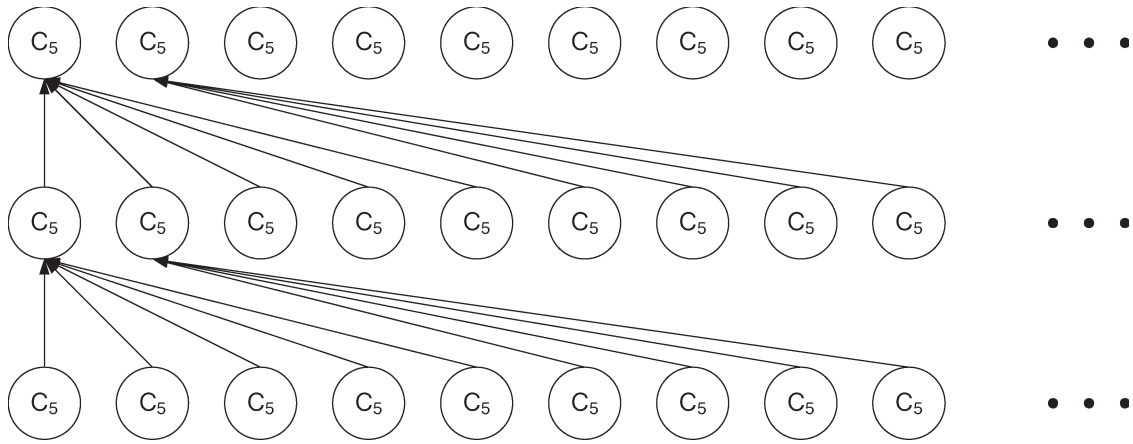


Fig. 7. Partial view of LEKO(G_{125}).

can reach x. By construction, it should be obvious that every C_n at the bottom layer can reach x, therefore, there are no reconverging paths. ■

Claim 2: Mapping upward (layer i).

Mapping the nodes in C_n at layer i so that the resulting LUT takes nodes from layer i and $i+1$ (i.e., mapping upward across a layer) requires one more LUT than mapping within the layers.

Proof: Assuming that the inputs to C_n on layer i are already LUTs, we know that the optimal mapping of each C_n has everything packed as tightly as possible (it only uses 4-LUTs), so in order to extend into layer $i+1$, one of the LUTs (in the optimal mapping of a C_n) has to split into two separate LUTs, thereby creating one additional LUT. This is because every output of a particular C_n in layer i will never be combined with another output from that C_n in any layer above i (Claim 1). ■

Claim 3: Mapping upward (layer $i+1$).

Any extensions of LUTs from layer i into layer $i+1$ will not result in layer $i+1$ being mapped with fewer LUTs than mapping within the layers.

Proof: Assume that the number of LUTs to map a C_n optimally is N, and the LUT that spans layer i and layer $i+1$ is called x. Since LUT x is partially in layer i and partially in layer $i+1$, the LUT has at most three inputs in layer $i+1$. Since this LUT in layer $i+1$ has only three inputs to choose from, and we know the optimal mapping for C_n in layer $i+1$ is strictly made up of LUTs with exactly four inputs. This will result, in the best case, in a mapping for the C_n in layer $i+1$ with N LUTs plus one spanning the two layers. Another way to look at this is to consider the question: Can you map the C_n on layer $i+1$ using $N-1$ 4-LUTs and one 3-LUT? Consider Fig. 4 for a pictorial representation of the question proposed. The answer to this question is clearly no because of the optimality of the N LUTs needed to map C_n. ■

Claim 4: Mapping downward (layer i).

Any extensions of LUTs from layer i into layer $i-1$ will not result in layer i being mapped with fewer LUTs.

Proof: Assume that the inputs to C_n at layer i are already LUTs. We know that in the optimal mapping of each C_n everything is already packed as tightly as possible (it only uses 4-LUTs), so extending into layer $i-1$ will not be possible unless there are reconverging paths at some layer below i. However, this is impossible. Due to the tree structure of G, every input into every C_n at layer i will never meet again; this was proven in Claim 1. ■

Proof of Theorem 2: Let G be an arbitrary LEKO circuit with L layers constructed using the previous construction.

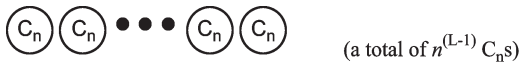
Let us define a property that we will use in the proof.

Property P(n): Let P(n) mean that the optimal mapping for all nodes up to layer n is the optimal mapping of each C_n separately.

It is then enough to show that P(1) is true, and if $P(m-1) \Rightarrow P(m)$, where $2 \leq m \leq L$ (which will show by induction that the optimal mapping of our arbitrary G is just the optimal mapping of each C_n separately). ■

Base Case: P(1) is true.

Proof: Before we begin the proof, this is what "all nodes up to layer 1" looks like:



Now, that we have an understanding of what this looks like, let us consider all the possible ways to map all nodes up to layer 1.

Case 1) Mapping exactly all the nodes of layer 1 and not mapping any nodes of layer 2.

Since there is no overlap between the C_ns (thus trying to pack nodes from different C_ns into one LUT cannot possibly reduce the area) and we know the optimal mapping of C_n, the optimal mapping of this layer will result in mapping each C_n separately.

Case 2) Mapping exactly all the nodes of layer 1 and possibly mapping some nodes of layer 2.

Now, we have to consider the case where the optimally mapped 4-LUT solution for layer 1 maps some nodes in layer 2. But from Claim 2 (its assumption holds since we are at the lowest layer and all the inputs are primary inputs) and Claim 3, we see that it will not help mapping if LUTs span across layer 1 and into layer 2; thus, Case 2) cannot happen and Case 1) must happen. From Case 1), we can see that P(1) must hold; thus, the base case is proved. ■

Inductive Step: $\mathrm{P(m-1)} \Rightarrow \mathrm{P(m)}$.

Proof: Recall that P(m) is saying that the optimal mapping for all nodes up to layer m is the optimal mapping of each $\mathrm{C_n}$ separately. Since $\mathrm{P(m-1)}$ is assumed, we know that the optimal mapping for all nodes up to layer $\mathrm{m-1}$ is the optimal mapping of each $\mathrm{C_n}$ separately. Now, all we need to know to prove P(m) is that any LUTs spanning two separate layers will not result in a better mapping solution (Claim 4). With Claim 2, which uses the inductive hypothesis $\mathrm{P(m-1)}$ to uphold the assumption that all the inputs to layer m are already LUTs, and Claim 3, we know that the mapping of nodes up to layer m will not intrude on layer $\mathrm{m+1}$. Moreover with Claim 4, we know that the mapping will not create LUTs that intrude into any layer below m. Thus, the optimal mapping for layer m is to map each $\mathrm{C_n}$ separately, and the inductive step is proven. Therefore, by induction, the optimal mapping of G is that which maps every $\mathrm{C_n}$ optimally and separately. ■

B. Construction of LEKU Examples

A LEKU example LEKU(G) is derived from the LEKO example G after collapsing and gate decomposition of G. Clearly, the optimal optimization + technology mapping solution of G provides an upper bound on the area of the corresponding LEKU(G) example due to the functional equivalence of G and LEKU(G). In this paper, we focus on constructing LEKU($\mathrm{G_{25}}$), which may already result in over 1 million gates after collapsing and decomposition. It is not reasonable to require the existing FPGA synthesis tools to handle larger examples beyond such sizes.

In fact, after collapsing of $\mathrm{G_{25}}$, we tried different decomposition algorithms. LEKU-CD($\mathrm{G_{25}}$) was constructed by first collapsing LEKO($\mathrm{G_{25}}$) into a two-level network, then decomposing the result into an equivalent two-bounded simple-gate network (using SIS [32] commands *collapse* and *tech_decomp*); LEKU-CB(G) was constructed by first collapsing the network, then balancing was done using the *collapse* and *balance* commands of the ABC system [38]. From the circuit-size profile shown in Table II, one can see that ABC's internal canonical AND–INV representation leads to the removal of a large number of functionally equivalent gates.

Since Xilinx's mapper could not accept a circuit as large as LEKU-CD($\mathrm{G_{25}}$), we broke LEKU-CD($\mathrm{G_{25}}$) up into a collection of nonoverlapping circuits; one circuit for each primary output. The resulting collection of circuits is clearly equivalent to LEKU-CD($\mathrm{G_{25}}$) and denoted as LEKU-CD($\mathrm{G_{25}}$).

C. LEKO and LEKU Properties

The LEKO circuit can exhibit a multitude of properties based on which core graph is used to construct it. In this paper, we constructed our core graph $\mathrm{C_5}$ to be as hard as possible for the industrial tools (Quartus and ISE) to map, while $\mathrm{C_6}$ was made with special properties that made it difficult for the academic tools (ABC and DAOMap) to map. Both of these core graphs were constructed by hand to show that the heuristics used in all of these tools can be easily subverted.

It is interesting to note that core graphs can be constructed from pre-existing benchmarks or complex logic blocks like adders, multipliers, and shifters. To construct a core graph from a pre-existing benchmark, all one has to do is extract a piece of logic that has an equal number of input and output and is hopefully hard to map. The problem is that it can be quite difficult to trick the tools into performing the wrong thing with a simple circuit. For example, every mapper managed to optimally map, in terms of depth and area, an 8-bit adder and an 8×8 multiplier.

Since these circuits were constructed by hand, it is interesting to see how similar they are to existing benchmarks. We will show this by comparing the LEKO and LEKU to Microelectronics Center of North Carolina (MCNC) benchmarks in terms of their Rent's exponent, I/O to node count, and maximum fanout free cone (MFFC) size. The MCNC benchmarks were processed through standard simplification scripts including decomposition into two input gates (for a fair comparison to the LEKO and LEKU circuits) but remained unmapped. Rent's rule [26] shows the relationship between the number of external I/O connections to a logic block and the number of logic gates in the logic block. It is estimated to be the slope of the regression line comparing the size versus number of I/O on the log–log scale. Higher Rent's exponent values correspond to a higher topological complexity, with a Rent's exponent value of zero corresponding to a simple logic chain and a value of one corresponding to a clique. In this paper, we estimate Rent's exponent using a top–down partitioning with hMetis [25] similar to the method described in [37]. The Rent's exponents of the MCNC benchmarks in Table I(b) range from 0.51 to 0.87 with an average of 0.74. When comparing our LEKO and LEKU circuits to the MCNC benchmarks, we can see that the range of the Rent's exponent falls perfectly in line with the MCNC benchmarks with an average exponent of 0.7. From this view, there is little structural difference between the two sets of benchmarks.

Another way to examine these circuits is by analyzing their MFFC sizes. Using the SIS platform, we were able to calculate the average MFFC size [Table I(a)] of the MCNC circuits to be six. Once again, the LEKO and most of the LEKU circuits match the MCNC benchmarks very well in terms of the average MFFC sizes. The LEKU-CD($\mathrm{G_{25}}$) circuit is the exception; it has an extremely high average MFFC size which makes it quite different from the MCNC benchmarks but not entirely unrealistic (a similar MFFC size shows up when examining an error checking circuit with only one output). But instead of trying to reflect a real circuit, it can be used as a tool to evaluate how well your algorithm finds duplication.

III. Results

The results of this paper will be presented in two parts. We first discuss the LEKO circuits created by create*LEKO* ($\mathrm{G_{25}}$, $\mathrm{G_{125}}$, and $\mathrm{G_{625}}$ were created by using $\mathrm{C_5}$, and $\mathrm{G_{36}}$, $\mathrm{G_{216}}$, and $\mathrm{G_{1296}}$ were created using $\mathrm{C_6}$), and we then discuss the LEKU examples which are functionally equivalent circuits of LEKU($\mathrm{G_{25}}$)—LEKU-CD($\mathrm{G_{25}}$), LEKU-CD($\mathrm{G_{25}}$), and LEKU-CB($\mathrm{G_{25}}$). The details of these circuits are shown in Table II. Using these examples, we present the results of running state-of-the-art academic FPGA mappers DAOmap [7], ABC [38],

TABLE I
(a) RENT'S EXPONENT AND MFFC ANALYSIS OF THE MCNC BENCHMARKS. (b) RENT'S EXPONENT AND MFFC ANALYSIS OF THE LEKO/LEKU EXAMPLES

Circuits	Rent's Exponent	# MFFC	Average MFFC Size
alu4	0.77	272	8
apex2	0.61	229	8
apex4	0.78	462	4
bigkey	0.87	588	5
clma	0.51	537	17
des	0.77	634	5
diffeq	0.71	578	4
dsip	0.83	250	10
elliptic	0.84	1253	4
ex1010	0.67	888	5
ex5p	0.80	438	3
frisc	0.77	1286	4
misex3	0.72	385	5
pdc	0.67	873	4
s298	0.78	230	8
s38417	0.65	2019	4
s38584	0.72	1981	4
seq	0.75	500	5
spla	0.70	704	6
tseng	0.81	519	3
Average	0.74	731	6

(a)

Circuits		Rent's Exponent	# MFFC	Average MFFC Size
LEKO	**G_{25}**	0.66	95	3
	G_{125}	0.79	700	3
	G_{625}	0.86	4625	3
	G_{36}	0.64	150	5
	G_{216}	0.79	1332	5
	G_{1296}	0.83	10584	5
LEKU	**CD(G_{25})**	0.54	25	46682
	CB(G_{25})	0.58	208	4
	CB(G_{36})	0.61	245	3

(b)

and the leading-edge FPGA synthesis systems from Altera [39] and Xilinx [40] on each of the circuits. DAOmap from the SIS [32] and RASP [16] environments was used with options allowing only the use of LUTs with four or less inputs (DAOmap—k 4). Berkeley's ABC mapper [38] was also used for mapping into 4-LUTs (using the "FPGA" command which targets 4-LUTs). Note that DAOmap produces a depth-optimal mapping solution as FlowMap [13] but uses 29% less LUTs on average as calculated from [7]. ABC mapper also produces depth-optimal mapping solutions, but uses 7% less LUTs than DAOmap on average, as reported in [1]. Altera's logic-synthesis tool was run from Quartus 5.0 [39] using Stratix

TABLE II
LEKO AND LEKU EXAMPLES USED FOR OPTIMALITY STUDY

Circuits		Core Graph	# Nodes	Depth	#I/O	# Nodes	Depth
			LEKO			Optimal Mapping Result	
LEKO	**G_{25}**	**C_5**	305	13	50	70	4
	G_{125}		2350	20	225	525	6
	G_{625}		15,875	27	1250	3,500	8
	G_{36}	**C_6**	768	27	72	120	6
	G_{216}		7,020	41	432	1,080	9
	G_{1296}		56,592	55	2592	8,640	12
			LEKU			Upper Bound on Optimal Synthesis Result	
LEKU	**CD(G_{25})**	**C_5**	1,166,655	19	50	70	4
	CB(G_{25})		814	16	50	70	4
	CB(G_{36})	**C_6**	824	14	72	120	6

TABLE III
MAPPING RESULTS ON LEKO EXAMPLES

Circuits		DAOmap	ABC	Quartus	ISE	Optimal
LEKO(G_{25})	Area	83	80	72	80	70
	Ratio	1.19	1.14	1.03	1.14	1.00
LEKO(G_{125})	Area	650	609	561	588	525
	Ratio	1.24	1.16	1.07	1.12	1.00
LEKO(G_{625})	Area	4435	4072	3737	3974	3500
	Ratio	1.27	1.16	1.07	1.14	1.00
Average Ratio (using C_5)		1.23	1.16	1.05	1.13	1.00
LEKO(G_{36})	Area	139	149	121	158	120
	Ratio	1.16	1.24	1.01	1.32	1.00
LEKO(G_{216})	Area	1301	1336	1082	1078	1080
	Ratio	1.20	1.24	1.00	1.00	1.00
LEKO(G_{1296})	Area	10695	10650	8645	8626	8640
	Ratio	1.24	1.23	1.00	1.00	1.00
Average Ratio (using C_6)		1.20	1.24	1.00	1.10	1.00
Average Ratio		1.22	1.20	1.03	1.12	1.00

device EP1S80F1508I7 and the option for area optimization. Xilinx's logic-synthesis tool was run from Xilinx ISE 7.1i [40] using Virtex device xcv3200e and also the option for area optimization. All the tools were run with the default settings unless otherwise stated. For this paper, we only performed the logic-synthesis steps of these tools and did not go through final placement and routing. The depths of the mapped LEKO circuits are not reported here for two reasons: Xilinx and Altera optimize for delay instead of depth, and the final logic element in the Xilinx device is not a 4-LUT but a slice that combines two 4-LUTs.

A. *Mapping Results on LEKO Examples*

This section will illustrate how well mappers perform in achieving the optimal mapping solution, if they do not have to carry out logic optimization. We tested this by running each one of the LEKO circuits on each one of the mappers. As the results in Table III show, each mapper and logic-synthesis tool does a fairly good job mapping the benchmarks. The average gap from

TABLE IV
LOGIC-SYNTHESIS RESULTS ON LEKU EXAMPLES

Circuits		DAOmap	ABC	Quartus	ISE	Upper Bounds
LEKU-CD(G_{25})	Area	22,717	30,511	10,381	*	70
	Ratio	325	436	148	*	1
LEKU-CD(G_{25})'	Area	25,247	35,271	5,005	9,717	70
	Ratio	361	504	72	139	1
Average LEKU-CD Ratio		343	470	110	139	1
LEKU-CB(G_{25})	Area	322	191	239	280	70
	Ratio	4.6	2.7	3.4	4.0	1
LEKU-CB(G_{36})	Area	356	339	206	290	120
	Ratio	3.0	2.8	1.7	2.4	1
Average LEKU-CB Ratio		3.8	2.8	2.6	3.2	1
Average Ratio		173	236	56	71	1

(Note: *The Xilinx mapper was not able to accept a circuit of this size)

optimal varies from 3% (by Quartus) to 22% (by DAOMap), with an average of 15%. This shows that the current LUT-based FPGA mappers or synthesis tools perform quite well on circuits, where logic optimization is not needed to obtain the optimal solutions. Note that Quartus and ISE perform both logic optimization and technology mapping, while DAOMap performs technology mapping only and ABC performs some logic optimization during mapping which allows the removal of a large amount of logic. It is interesting to see that by using different core graphs, we can show that DAOmap can outperform ABC. At first, it would seem that the smaller core graph C_5 produced results further away from the optimal, but this is only the case for the tools that performed logic optimization. C_6 has a difficult structure for mapping but some of its functionality can be simplified, since it is very hard to create a C_6 that is difficult for the whole logic-synthesis process.

B. Synthesis Results on LEKU Examples

This section will illustrate how poorly most of the best available FPGA logic-synthesis flows perform when logic restructuring and/or optimizing is needed to achieve the optimal mapping solution. The academic mappers presented in this section are allowed to use standard preprocessing tools (script.algebraic for DAOmap and resyn2 for ABC mapper) for technology-independent logic optimization, since the LEKU examples require logic restructuring/optimization to achieve the optimal mapping solutions. From the results in Table IV, when examining the largest test case, we see that all four synthesis flows perform poorly and produce synthesis results with area ranging from 72X to 504X larger than the known upper bounds (the mapping results of the equivalent LEKO examples), averaging 256X larger. We believe Quartus produced a better solution on LEKU-CD than LEKU-CD, because it could perform more optimizations on each one of the circuits of LEKU-CD due to their smaller size. It is also interesting to note that in the correlation between the performance of commercial tools on LEKU-CB(G_{36}) and LEKU-CB(G_{25}), the simpler logical structure of C_6 allowed for more logic optimizations. One of the main reasons that every one of these algorithms performed so poorly is because they were not able to reconstruct the original structure of the circuit. The fact that the same logic-synthesis flows perform so much worse on the LEKU examples than the equivalent LEKO examples suggests that the existing logic-optimization algorithms are not capable of reproducing the initial circuit structure of the LEKO examples. This suggests that there may be significant opportunity for improvement in the existing logic-synthesis algorithms. For example, we believe that in order for logic-synthesis algorithms to perform well on the LEKU examples, they must have a more global view of resynthesis—including duplication removal, logic identification, and many other heuristics that examine the circuit globally. Without such global heuristics, algorithms do not perform well on LEKU examples and may produce poor results on large real-world circuits as well.

C. Applications of Results

Regarding the LEKO benchmarks, we feel that the ones we presented here not only have the known optimal solutions but also have many structural similarities to the widely used MCNC examples, based on the MFFC and Rent's exponent analysis. The problem with the MCNC benchmarks is that almost every logic-synthesis tool is specifically tuned to perform well on these benchmarks. By presenting a set of alternative-design examples, we hope to better understand where and how these algorithms differ. Another advantage of the LEKO construction is that it enables a designer to combine multiple "hard to map circuit cores" into one design with a known optimal. Knowing the optimal solution, the designer can see exactly where the algorithm made the incorrect choice and why.

The LEKU designs, on the other hand, can be used to test how different logic-synthesis algorithms handle the design with inefficiency or redundancy. For example, one can take a LEKO design and start "introducing inefficiency or redundancy," which may include duplicate logic or signals that can eventually be factored away. Using these "flawed" designs, we can test various logic-synthesis tools to see the degree to which they handle such design flaws. This type of test is very important, since popularity of hardware languages enables people with little or no hardware background to design hardware. Sometimes people end up writing Verilog or very-high-speed integrated-circuit hardware-description language (VHDL) in the same way they write C, which results in designs with a high degree of inefficiency or redundancy. The LEKU circuits are meant to test how the existing logic-synthesis algorithms perform on such designs and how much room is left for improvement when handling each type of inefficiency and/or redundancy in the design.

A good mapper should not only employ a single heuristic but should be able to combine multiple heuristics so that it can perform well on different types of benchmarks. We presented a platform to create new benchmarks that can test every part of a logic-synthesis tool. For example, to check that all the heuristics of a logic-synthesis tool are working in harmony, one only has to create a set of C_ns representing difficult circuits and

use them to create a LEKO circuit. Also, to test the capabilities and strengths of various logic optimization phases of the tools, all one has to do is introduce certain types of inefficiency or redundancy (called targeted perturbation) in the underlying LEKO circuit (thus creating a LEKU circuit) and check to see if the tools can correct them.

The advantage of the LEKU circuit is that it establishes a clear bound on how much minimization one can do. Since there is no known lower bound of the MCNC benchmarks, there is no clear way to tell if an improvement on a previous algorithm is significant. For example, an improvement of some algorithm X on another algorithm Y by 10% might be insignificant if both algorithms are over 3x away from optimal. On the other hand, when an algorithm does really well on certain LEKU circuits, it means that the algorithm was able to find and correct all the targeted perturbations used to create it. Such a controlled experiment is very useful when one has a collection of LEKU circuits to use for testing how the synthesis tools may perform on common inefficiencies or redundancies that exist in some HDL designs. For example, one can inject targeted perturbations such as do-not-care signals and duplicated logic and see how well an algorithm can detect and correct them. Since these circuits cannot be minimized past a certain point, one knows that the reason the logic synthesis under test did well is precisely because it found the targeted perturbations that one injects and not because of anything else. Without a tight bound on the area, one cannot simply tell why or how the tool produced a better solution.

IV. Conclusion

In this paper, we presented an algorithm for creating synthetic benchmarks with known optimal technology mapping solutions for LUT-based FPGA designs. Using these, LEKO and LEKU benchmarks of sizes ranging from a few hundred nodes to over one million nodes, we experimented on four state-of-the-art FPGA logic-synthesis flows. We showed that although leading FPGA technology mapping algorithms can produce close to optimal solutions with an average gap of 15% on the LEKO examples, the results from the entire logic-synthesis flows (logic optimization + mapping) are far from optimal. The best industrial and academic FPGA synthesis flows are around 70 times larger in terms of area on average and, in some cases, as much as 500 times larger on LEKU examples.

It is important to understand that just because an algorithm performs poorly on a set of artificial benchmarks, it does not mean the algorithm will perform badly on real world circuits. Since logic synthesis is NP-hard and all exiting synthesis algorithms are heuristics in nature, one would naturally expect there are examples where the heuristics perform poorly. Nevertheless, it is important to have a quantitative measurement of the optimality gap. We would also like to emphasize that the performance of different heuristics on LEKO and LEKU may not reflect their performance on other examples. We refer the reader to [21, Ch. 6] on a discussion about the worst case versus average case performance of heuristics.

We hope that the rather surprising results on LEKO and LEKU examples will stimulate the logic-synthesis community as did the PEKO examples to the physical-design community. Needless to say, the potential of large-scale-area reduction is of great interest to the IC and EDA industries. If realized, it leads to significant improvement in density and cost of future integrated circuits. It is not clear if and how often these artificial examples constructed by our algorithm appear in real-life circuits. However, these examples will help to identify deficiencies in the current logic-synthesis algorithms and improve their quality. It is our hope that these benchmarks are not only used to determine that logic-synthesis tools catch commonly made mistakes (redundant logic and unused logic) that tool designers expect, but also create an online community that leads to the sharing of LEKO circuits that will test every possible design flaw.

Although our optimality study is done for LUT-based FPGAs, we think that the same technique can be easily extended to cell-based IC designs, where one needs to map to a library of different logic cells. In this case, we need to modify the construction of C_n so that it remains a "hard core" basis of constructing larger hard examples.

The LEKO and LEKU examples are available online [41].

Acknowledgment

The authors would like thank Dr. D. Chen for providing the DAOmap implementation, Prof. R. Brayton and Dr. A. Mishchenko for providing the ABC mapper, and J. Lin for his helpful feedback on this paper.

References

[1] R. Brayton, S. Chatterjee, M. Ciesielski, and A. Mishchenko, "An integrated technology mapping environment," in *Proc. Int. Workshop Logic and Synthesis*, 2005, pp. 383–390.

[2] S. Brown, A. Ling, and D. Singh, "FPGA technology mapping: A study of optimality," in *Proc. Des. Autom. Conf.*, 2005, pp. 427–432.

[3] S. Brown, V. Manohararajah, and Z. Vranesic, "Heuristics for area minimization in LUT based FPGA technology mapping," in *Proc. Int. Workshop Logic and Synthesis*, 2004, pp. 14–21.

[4] T. Chan, J. Cong, and K. Sze, "Multilevel generalized force-directed method for circuit placement," in *Proc. Int. Symp. Phys. Des.*, San Francisco, CA, Apr. 2005, pp. 185–192.

[5] T. Chan, J. Cong, T. Kong, J. R. Shinnerl, and K. Sze, "An enhanced multilevel algorithm for circuit placement," in *Proc. Int. Conf. Comput.-Aided Des.*, 2003, pp. 299–306.

[6] C. Chang, J. Cong, and M. Xie, "Optimality and scalability study of existing placement algorithms," in *Proc. ASP-DAC*, 2003, pp. 621–627.

[7] D. Chen and J. Cong, "DAOmap: A depth-optimal area optimization mapping algorithm for FPGA designs," in *Proc. IEEE/ACM ICCAD*, 2004, pp. 752–759.

[8] D. Chen, J. Cong, and P. Pan, "A decade of advances in FPGA design automation," *Foundations and Trends Electron. Des. Autom.*, vol. 1, no. 2, 2006, to be published.

[9] K. Chen *et al.*, "DAG-map: Graph-based FPGA technology mapping for delay optimization," *IEEE Des. Test Comput.*, vol. 9, no. 3, pp. 7–20, Sep. 1992.

[10] J. Cong and Y. Ding, "Beyond the combinatorial limit in depth minimization for LUT-based FPGA designs," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 1993, pp. 110–114.

[11] ——, "Combinational logic synthesis for LUT based field programmable gate arrays," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 1, no. 2, pp. 145–204, Apr. 1996.

[12] J. Cong and Y. Hwang, "Simultaneous depth and area minimization in LUT-based FPGA mapping," in *Proc. Int. Symp. Field-Programmable Gate Arrays*, Feb. 1995, pp. 68–74.

[13] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs,"

IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 13, no. 1, pp. 1–12, Jan. 1994.
[14] J. Cong, C. Wu, and E. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," in *Proc. Int. Symp. Field-Programmable Gate Arrays*, Feb. 1999, pp. 29–35.
[15] J. Cong, G. Nataneli, M. Romesis, and J. Shinnerl, "An area-optimality study of floorplanning," in *Proc. Int. Symp. Phys. Des.*, 2004, pp. 78–83.
[16] J. Cong, J. Peck, and Y. Ding, "RASP: A general logic synthesis system for SRAM-based FPGAs," in *Proc. Int. Symp. Field-Programmable Gate Arrays*, 1996, pp. 137–143.
[17] J. Cong, M. Romesis, and M. Xie, "Optimality and stability study of timing-driven placement algorithms," in *Proc. Int. Conf. Comput.-Aided Des.*, 2003, pp. 472–478.
[18] ——, "Optimality, scalability and stability study of partitioning and placement algorithms," in *Proc. Int. Symp. Phys. Des.*, 2003, pp. 88–94.
[19] E. Dubrova and M. Teslenko, "Hermes: LUT FPGA technology mapping algorithm for area minimization with optimum depth," in *Proc. Int. Conf. Comput.-Aided Des.*, 2004, pp. 748–751.
[20] R. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast technology mapping for lookup table-based FPGAs," in *Proc. Des. Autom. Conf.*, 1991, pp. 227–233.
[21] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.
[22] A. Kahng and Q. Wang, "Implementation and extensibility of an analytic placer," in *Proc. Int. Symp. Phys. Des.*, 2004, pp. 18–25.
[23] A. Kahng and S. Reda, "Evaluation of placer suboptimality via zero-change netlist transformations," in *Proc. Int. Symp. Phys. Des.*, 2005, pp. 208–215.
[24] K. Karplus, "Xmap: A technology mapper for table-lookup field-programmable gate arrays," in *Proc. Des. Autom. Conf.*, 1991, pp. 240–243.
[25] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in VLSI domain," in *Proc. Des. Autom. Conf.*, 1997, pp. 526–529.
[26] B. Landman and R. Russo, "On a pin versus block relationship for partitions of logic graphs," *IEEE Trans. Comput.*, vol. C-20, no. 12, pp. 1469–1479, 1971.
[27] C. Legl, B. Wurth, and K. Eckl, "A Boolean approach to performance-directed technology mapping for LUT-based FPGA designs," in *Proc. Des. Autom. Conf.*, Jun. 1996, pp. 730–733.
[28] I. Markov and J. A. Roy, "On sub-optimality and scalability of logic synthesis tools," in *Proc. Int. Workshop Logic and Synthesis*, May 2003.
[29] R. Murgai *et al.*, "Improved logic synthesis algorithms for table look up architectures," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 1991, pp. 564–567.
[30] J. Roy, D. Papa, S. Adya, H. Chan, A. Ng, J. Lu, and I. Markov, "Capo: Robust and scalable open-source min-cut floorplacer," in *Proc. Int. Symp. Phys. Des.*, 2005, pp. 224–226.
[31] P. Sawkar and D. Thomas, "Technology mapping for table-look-up based field programmable gate arrays," in *Proc. ACM/SIGDA Workshop Field Programmable Gate Arrays*, Feb. 1992, pp. 83–88.
[32] E. Sentovitch, K. Singh *et al.*, "SIS: A System for sequential circuit synthesis," Univ. California, Berkeley, Tech. Rep., UCB/ERL M92/41, May 1992.
[33] K. Sze, "Multilevel optimization for VLSI circuit placement," Ph.D. dissertation, Univ. California, Los Angeles, 2006.
[34] N. Viswanathan and C. Chu, "FastPlace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model," in *Proc. Int. Symp. Phys. Des.*, 2004, pp. 26–33.
[35] N. Woo, "A heuristic method for FPGA technology mapping based on the edge visibility," in *Proc. Des. Autom. Conf.*, 1991, pp. 248–251.
[36] H. Yang and D. Wong, "Edge-map: Optimal performance driven technology mapping for iterative LUT based FPGA designs," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 1994, pp. 150–155.
[37] X. Yang, E. Bozorgzadeh, and M. Sarrafzadeh, "Wirelength estimation based on rent exponents of partitioning and placement," in *Proc. Int. Workshop Syst.-Level Interconnect Prediction*, 2001, pp. 25–32.
[38] Berkeley Logic Synthesis and Verification Group, *ABC: A System for Sequential Synthesis and Verification*. [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc/
[39] Altera Inc., *Quartus 5.0*. [Online]. Available: http://www.altera.com/
[40] Xilinx Corporation, *ISE Logic Design Tools 7.1i*. [Online]. Available: http://www.xilinx.com/
[41] *UCLA Optimality Study Project*. [Online]. Available: http://cadlab.cs.ucla.edu/~pubbench/

Jason Cong received the B.S. degree from Peking University, Beijing, China, in 1985 and the M.S. and Ph.D. degrees from the University of Illinois, Urbana–Champaign (UIUC), in 1987 and 1990, respectively, all in computer science.

He is currently a Professor and Chairman of the Computer Science Department of University of California, Los Angeles (UCLA). He is also a Codirector of the VLSI CAD Laboratory. He was the Founder and President of Aplus Design Technologies, Inc., until it was acquired by Magma Design Automation, in 2003. He currently serves as the Chief Technology Advisor of Magma and AutoESL Design Technologies, Inc. Additionally, he has been a Guest Professor at Peking University, since 2000. His research interests include computer-aided design of very large-scale integration (VSLI) circuits and systems, design and synthesis of system-on-a-chip, programmable systems, novel computer architectures, nanosystems, and highly scalable algorithms. He has published over 230 research papers. He has graduated 17 Ph.D. students.

Dr. Cong is the recipient of a number of awards and recognition, including the Best Graduate Award from Peking University, in 1985, and the Ross J. Martin Award for Excellence in Research from the University of Illinois, in 1989, the NSF Young Investigator Award, in 1993, the Northrop Outstanding Junior Faculty Research Award from UCLA, in 1993, the ACM/SIGDA Meritorious Service Award, in 1998, and the SRC Technical Excellence Award, in 2000. He is also the recipient of the three best paper awards—the 1995 IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN Best Paper Award, the 2005 International Symposium on Physical Design Best Paper Award, and the 2005 ACM TRANSACTIONS ON DESIGN AUTOMATION OF ELECTRONIC SYSTEMS Best Paper Award. He served on the technical program committees and executive committees of many conferences, such as Asia and South Pacific Design Automation Conference (ASPDAC), Design Automation Conference (DAC), ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), International Conference on Computer-Aided Design (ICCAD), International Symposium on Circuits And Systems (ISCAS), International Symposium on Physical Design (ISPD), and International Symposium on Low Power Electronics and Design (ISLEPD), and several editorial boards, including the IEEE TRANSACTIONS ON VERY LARGE-SCALE INTEGRATION SYSTEMS and the ACM TRANSACTIONS ON DESIGN AUTOMATION OF ELECTRONIC SYSTEMS. He served on the ACM SIGDA Advisory Board, the Board of Governors of the IEEE Circuits and Systems Society, and the Technical Advisory Board of a number of EDA and silicon IP companies, including Atrenta, eASIC, Get2Chip, Magma Design Automation, and Ultima Interconnect Technologies.

Kirill Minkovich received the B.S. degree from the University of California, Berkeley, in 2003, and the M.S. degree from the University of California, Los Angeles, in 2006, where he is currently working toward a Ph.D. degree, all in computer science.

His current research interests include logic synthesis for LUT-based FPGAs, evaluation of synthesis algorithms, and synthesis for nanoscale architectures.