

Optimality Study of Logic Synthesis for LUT-Based FPGAs

Jason Cong and Kirill Minkovich
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095, USA

{cong, cory_m}@cs.ucla.edu

ABSTRACT

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 LEKO and 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 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.

Categories and Subject Descriptors

B.6.3 [Hardware]: Logic Design – Design Aids

General Terms

Algorithms, Experimentation, Performance

Keywords

Technology Mapping, Logic Synthesis, Boolean Logic, Optimization, FPGA Lookup Table

1. INTRODUCTION

Field programmable gate arrays (FPGAs) have been gaining momentum as an alternative to application-specific integrated circuits (ASICs). FPGAs consist of programmable logic, 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.

Given an RTL design, the typical FPGA synthesis process consists of RTL elaboration, logic synthesis, and the physical design (layout synthesis) [10]. 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, Figure 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 it just takes the logical AND of all of 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.

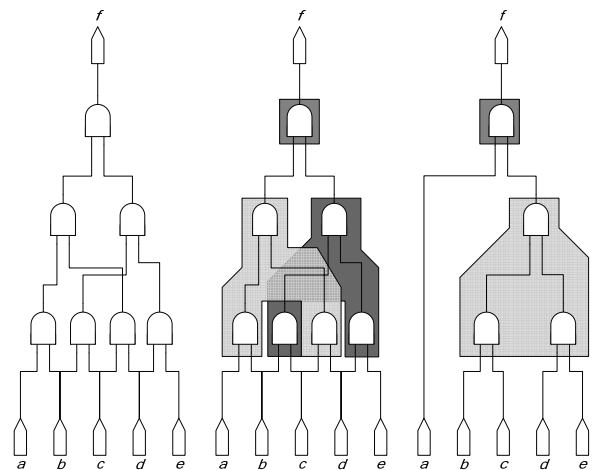


Figure 1. Possible area-minimal mapping solutions: (a) original circuit, (b) mapping solution without logic optimization, (c) mapping solution with logic optimization.

As the 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 Chortlecrf [19], MIS-pga [25], XMap [22], VisMap [30], TechMap [27],

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

FPGA'06, February 22–24, 2006, Monterey, California, USA.

Copyright 2006 ACM 1-59593-292-5/06/0002...\$5.00.

Praetor [13], DAG-map [8], EdgeMap [31], FlowMap [12], Zmap [15], Cutmap [11], BoolMap [23] and FlowSYN [9]. 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 [19][25][22][30][27][13], while others focused on area minimization possible under delay or depth constraints [27][8][31][12][15][11][23][9]. 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 [18], DAOmap [7] and the ABC mapper [1][32]. 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. However, 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 PEKO (Placement Examples with Known Optimal) examples, the study in [6] showed the surprising results that wirelengths produced by state-of-the-art placement tools at that time were 1.66 to 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., [17][16][14][21]), as well as novel placement algorithms (e.g., [26][20][29][5][4]). Within three years, the optimality gap on the PEKO examples was reduced to roughly 20% on average [4].

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 [24] presented a method which 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 see how much more the circuit area could be reduced by post-processing 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 LEKO and 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.

2. CONSTRUCTION OF BENCHMARKS

2.1 Construction of LEKO Examples

We present an algorithm for constructing a network G_n (with n inputs and outputs) of an arbitrarily large size that has a known optimal technology mapping solution. G 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 (Logic synthesis Examples with Known Optimal) examples. The building block of our construction is a “hard graph” named G_5 , with the following properties:

1. It has five inputs and five outputs.
2. Every output is a function of all five inputs.
3. Each internal node of G_5 has exactly two inputs.
4. There exists an optimal (in terms of area/depth) mapping of G_5 into a 4-LUT mapping solution, denoted M_5 , such that M_5 only has 4-LUTs (no 3-LUTs or 2-LUTs). For the G_5 shown in Figure 2, M_5 has exactly seven 4-LUTs.

The specific G_5 we used to construct our LEKO and LEKU benchmarks can be seen in Figure 2, and the optimality of its technology mapping solution is stated in Theorem 1 below and verified in its proof.

Theorem 1: G_5 has an area-optimal technology mapping solution of seven 4-LUTs.

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 [28] using the command “xl_cover -h 0.” The binate solver in our case returned a seven 4-LUT solution. The reason 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.

■

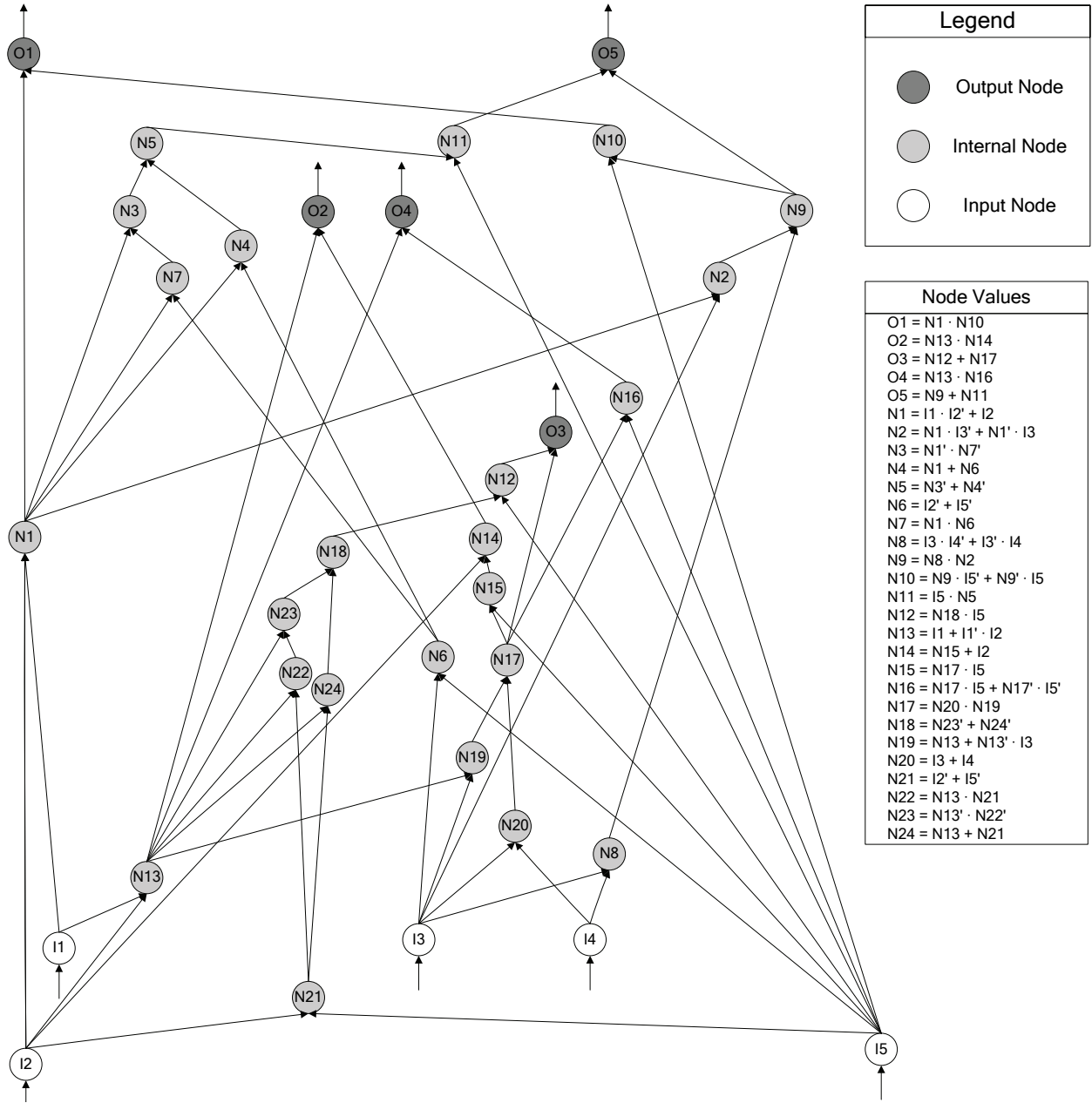


Figure 2. An example of G5

Using this newly created G5, a LEKO circuit G is created by stacking up G5s in such a way that from the outputs of G, there is only one way to traverse G5 to get to the inputs. The exact algorithm is presented in Figure 3, where *createLEKO(L)* creates a LEKO example with $L \cdot 5^{L-1}$ G5s in L layers. In the algorithm, the \cup (*union*) operator does not disturb the order of the inputs or

outputs. For example, when looking at the $A \cup B$ 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 $A \cup B$; 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.

```

algorithm createLEKO (L)
input: the number of layers L, output: network G
G =  $\bigcup^{5^{L-1}}$  copy(G5);
for i = 2,3,...,L do
  currLayer =  $\bigcup^{5^{L-i}}$  copy(G5);
  for j = 0,1,...,5L-i-2 do
    createEdge( Goutput[(5·j) mod (5L-1)] ,
              currLayerinput[j]);
  end-for
  createEdge( Goutput [ 5L-1 ],
            currLayerinput[ 5L-1 ]);
  G = G  $\bigcup$  currLayer;
end-for
output G;

```

Figure 3. createLEKO algorithm

Logically, the *createLEKO* algorithm works as follows. It builds up the graph using layer upon layer of G5s in order to get a LEKO example of L layers. It first creates the bottom layer of 5^{L-1} G5s, then for each additional layer it makes 5^{L-1} copies of G5 and proceeds to connect the outputs of the graph G to the inputs of the newly created layer. It spreads out the connections in such a way that for an arbitrary G5 at the top layer, there exists a path to it from every G5 at the bottom layer (i.e., every G5 at the top level is connected to every G5 at the bottom level). Thus, using this algorithm and any number n , one can create a LEKO circuit having more than n nodes and a known optimal technology mapping solution whose optimality is proven in Theorem 2. By using this method, we were able to construct G_{25} (seen in Figure 4) by calling *createLEKO* with two layers. We similarly constructed G_{125} (Figure 5) 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 G5 in the circuit is mapped optimally without overlapping any other G5.

Proof: See Appendix.

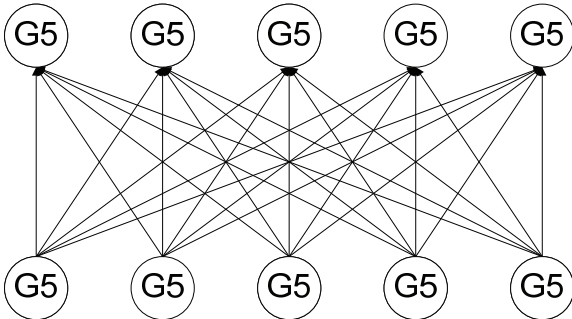


Figure 4. LEKO(G_{25})

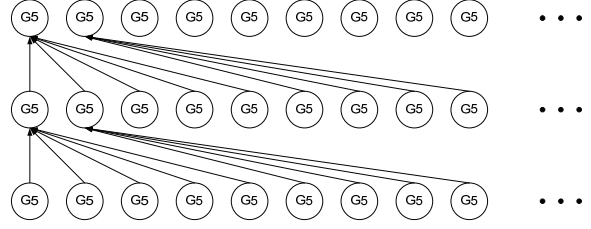


Figure 5. Partial view of LEKO(G_{125})

2.2 Construction of LEKU Examples

A LEKU (Logic synthesis Examples with Known Upper-bounds) 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 the paper we focus on constructing LEKU(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 G_{25} , we tried different decomposition algorithms—LEKU-CD(G_{25}) was constructed by first collapsing LEKU(G_{25}) into a two-level network, then decomposing the result into an equivalent two-bounded simple-gate network (using SIS [28] commands *collapse* and *tech_decomp*), while LEKU-CB(G_{25}) was constructed by first collapsing the network, then balancing was done using the *collapse* and *balance* commands of the ABC system [32]. From the circuit size profile shown in Table 1, 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(G_{25}), we broke LEKU-CD(G_{25}) up into a collection of non-overlapping circuits, one circuit for each primary output. The resulting collection of circuits is clearly equivalent to LEKU-CD(G_{25}), and denoted as LEKU-CD(G_{25})’.

3. RESULTS

The results of our study will be presented in two parts: we first discuss the LEKO circuits created by *createLEKO* (G_{25} , G_{125} and G_{625}), and we then discuss the LEKU examples which are functionally equivalent circuits of LEKO(G_{25})—LEKU-CD(G_{25}), LEKU-CD(G_{25})’ and LEKU-CB(G_{25}). The details of these circuits are shown in Table 1. Using these examples, we present the results of running state-of-art academic FPGA mappers DAOMap [7], ABC [32] and the leading-edge FPGA synthesis systems from Altera [33] and Xilinx [34] on each of the circuits. DAOMap from the SIS [28] and RASP [15] environment was used with options allowing only the use of LUTs with four or less inputs. Berkeley’s ABC mapper was used from the ABC [32] environment also for mapping into 4-LUTs. Note that DAOMap produces a depth-optimal mapping solution as FlowMap [12] 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 [33] using Stratix device EP1S80F150817 and the option for area optimization. Xilinx’s logic synthesis tool was run from Xilinx ISE 7.1i [34] using

Virtex device xcv3200e and also the option for area optimization. For the interests of this study, 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.

Table 1. LEKO and LEKU examples used for optimality study

Circuits		# Nodes	Depth	#I/O	Optimal	
					# Nodes	Depth
LEKO	G ₂₅	305	13	50	70	4
	G ₁₂₅	2350	20	225	525	6
	G ₆₂₅	15,875	27	1250	3,500	8
LEKU-CD(G ₂₅)		1,166,655	19	50	70	4
LEKU-CB(G ₂₅)		814	16	50	70	4

3.1 Synthesis Results on LEKO Examples

The 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 2 show, each mapper and logic synthesis tool does a fairly good job mapping the benchmarks. The average gap from optimal varies from 5% (by Quartus) to 23% (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 get 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.

Table 2. 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	4,435	4,072	3,737	3,974	3,500
	Ratio	1.27	1.16	1.07	1.14	1.00
Average Ratio		1.23	1.16	1.05	1.13	1.00

3.2 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 3, we see that all four synthesis flows perform poorly and produce synthesis results with area ranging from 71X to 504X larger than the known upper bounds (the mapping results of the equivalent LEKO examples), averaging 172X 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. 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 maybe significant opportunity for improvement of 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.

Table 3. 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	*	1.00
LEKU-CD(G ₂₅)'	Area	25,247	35,271	5,005	9,717	70
	Ratio	361	504	72	139	1.00
LEKU-CB(G ₂₅)	Area	322	191	239	280	70
	Ratio	4.60	2.73	3.41	4.00	1.00
Average Ratio		230	314	74	71	1.00

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

4. CONCLUSIONS

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 (Logic synthesis Examples with Known Optimals) and LEKU (Logic synthesis Examples with Known Upper bounds) 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 show 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.

We hope that these surprising results and examples will stimulate the logic synthesis community as did the PEKO examples in 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 on density and cost of future integrated circuits. It is possible that the artificial examples constructed by our algorithm may not appear in every real-life circuit. However, these examples will help to

identify deficiencies in the current logic synthesis algorithms and improve their quality.

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 G5 so that it remains to be a “hard core” basis of constructing larger hard examples.

The LEKO and LEKU examples are available online at [35].

5. Acknowledgements

This research was supported in part by Altera Corporation, Magma Design Automation Inc., and Xilinx Inc. under the California MICRO Program.

We would like to thank Dr. Deming Chen for providing the DAOmap implementation, Prof. Robert Brayton and Dr. Alan Mishchenko for providing the ABC mapper, and Joey Lin for his helpful feedback on the paper.

6. REFERENCES

- [1] R. Brayton, S. Chatterjee, M. Ciesielski, and A. Mishchenko, “An Integrated Technology Mapping Environment,” *Proc. International Workshop on Logic and Synthesis*, pp. 383-390, 2005.
- [2] S. Brown, A. Ling, and D. Singh, “FPGA Technology Mapping: A Study of Optimality,” *Proc. Design Automation Conference*, pp. 427-432, 2005.
- [3] S. Brown, V. Manohararajah, and Z. Vranesic, “Heuristics for Area Minimization in LUT Based FPGA Technology Mapping,” *International Workshop on Logic and Synthesis*, 2004.
- [4] T. Chan, J. Cong, and K. Sze, “Multilevel Generalized Force-directed Method for Circuit Placement,” *Proc. of the International Symposium on Physical Design*, San Francisco, CA, April, 2005.
- [5] T. Chan, J. Cong, T. Kong, J. R. Shinnerl, and K. Sze, “An Enhanced Multilevel Algorithm for Circuit Placement,” *Proc. International Conference on Computer Aided Design*, 2003.
- [6] C. Chang, J. Cong, and M. Xie, “Optimality and Scalability Study of Existing Placement Algorithms,” *IEEE Trans. on Computer-Aided Design*, pp. 621-627, 2003.
- [7] D. Chen and J. Cong, “DAOmap: A Depth-Optimal Area Optimization Mapping Algorithm for FPGA Designs,” *IEEE Trans. Computer-Aided Design*, pp. 752-759, 2004.
- [8] K. Chen, et al., “DAG-Map: Graph-based FPGA Technology Mapping for Delay Optimization,” *IEEE Design and Test of Computers*, vol. 9, no. 3, pp. 7-20, Sep. 1992.
- [9] J. Cong. and Y. Ding, “Beyond the Combinatorial Limit in Depth Minimization for LUT-Based FPGA Designs,” *Proc. International Conference on Computer Aided Design*, Nov. 1993.
- [10] J. Cong and Y. Ding, “Combinational Logic Synthesis for LUT Based Field Programmable Gate Arrays,” *ACM Trans. Design Automation of Electronic Systems*, Vol. 1, No. 2, pp. 145-204, April 1996.
- [11] J. Cong and Y. Hwang, “Simultaneous Depth and Area Minimization in LUT-Based FPGA Mapping,” *International Symposium on Field-Programmable Gate Arrays*, Feb. 1995.
- [12] J. Cong and Y. Ding, “FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs,” *IEEE Trans. Computer-Aided Design*, pp. 1-12, 1994.
- [13] J. Cong, C. Wu, and E. Ding, “Cut Ranking and Pruning: Enabling A General and Efficient FPGA Mapping Solution,” *International Symposium on Field-Programmable Gate Arrays*, Feb. 1999.
- [14] J. Cong, G. Nataneli, M. Romesis, and J. Shinnerl, “An Area-Optimality Study of Floorplanning,” *Proc. of the International Symposium on Physical Design*, pp. 78-83, 2004.
- [15] J. Cong, J. Peck, and Y. Ding, “RASP: A General Logic Synthesis System for SRAM-Based FPGAs,” *International Symposium on Field-Programmable Gate Arrays*, pp. 137-143, 1996.
- [16] J. Cong, M. Romesis, and M. Xie “Optimality and Stability Study of Timing-Driven Placement Algorithms,” *Proc. International Conference on Computer Aided Design*, pp. 472-478, 2003.
- [17] J. Cong, M. Romesis, and M. Xie, “Optimality, Scalability and Stability Study of Partitioning and Placement Algorithms,” *Proc. of the International Symposium on Physical Design*, pp. 88-94, 2003.
- [18] E. Dubrova and M. Teslenko, “Hermes: LUT FPGA Technology Mapping Algorithm for Area Minimization with Optimum Depth,” *Proc. International Conference on Computer Aided Design*, pp 748-751, 2004.
- [19] R. Francis, J. Rose, and Z. Vranesic, “Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs,” *Design Automation Conference*, 1991.
- [20] A. Kahng and Q. Wang, “Implementation and Extensibility of an Analytic Placer,” *Proc. of the International Symposium on Physical Design*, pp. 18-25, 2004.
- [21] A. Kahng and S. Reda, “Evaluation of Placer Suboptimality Via ZeroChange Netlist Transformations,” *Proc. of the International Symposium on Physical Design*, pp. 208-215, 2005.
- [22] K. Karplus, “Xmap: A Technology Mapper for Table-lookup Field-Programmable Gate Arrays,” *Design Automation Conference*, 1991.
- [23] C. Legl, B. Wurth, and K. Eckl, “A Boolean Approach to Performance-Directed Technology Mapping for LUT-Based FPGA Designs,” *Design Automation Conference*, June 1996.
- [24] I. Markov and J. A. Roy, “On Sub-optimality and Scalability of Logic Synthesis Tools,” *International Workshop on Logic and Synthesis*, May 2003.
- [25] R. Murgai, et al., “Improved Logic Synthesis Algorithms for Table Look Up Architectures,” *Proc. International Conference on Computer Aided Design*, Nov. 1991.
- [26] J. Roy, D. Papa, S. Adya, H. Chan, A. Ng, J. Lu, and I. Markov, “Capo: Robust and Scalable Open-Source Min-Cut Floorplacer,” *Proc. of the International Symposium on Physical Design*, pp. 224-226, 2005.
- [27] P. Sawkar and D. Thomas, “Technology Mapping for Table-Look-Up Based Field Programmable Gate Arrays,” *ACM/SIGDA Workshop on Field Programmable Gate Arrays*, Feb. 1992.

- [28] E. Sentovitch, K. Singh, et al., "SIS: A System for Sequential Circuit Synthesis," Technical Report, UCB/ERL M92/41, U.C. Berkeley, May 1992.
- [29] N. Viswanathan and C. Chu, "FastPlace: Efficient Analytical Placement Using Cell Shifting, Iterative Local Refinement and a Hybrid Net Model," *Proc. of the International Symposium on Physical Design*, pp. 26-33, 2004.
- [30] N. Woo, "A Heuristic Method for FPGA Technology Mapping Based on the Edge Visibility," Design Automation Conference, 1991.
- [31] Yang H. and D. F. Wong, "Edge-map: Optimal Performance Driven Technology Mapping for Iterative LUT Based FPGA Designs," *Proc. International Conference on Computer Aided Design*, Nov. 1994.
- [32] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification," <http://www.eecs.berkeley.edu/~alanmi/abc/>
- [33] Altera Inc., Quartus 5.0, <http://www.altera.com/>.
- [34] Xilinx Corporation, ISE Logic Design Tools 7.1i, <http://www.xilinx.com/>.
- [35] UCLA Optimality Study Project, <http://cadlab.cs.ucla.edu/~pubbench/>

7. APPENDIX

Summary of the Proof of Theorem 2

Now that we have the ability to construct arbitrarily sized LEKO circuits, we can show that this construction actually creates a circuit G with a known optimal binate cover, which we prove in Theorem 2. 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 $G5$ s. Claim 2 and Claim 3 will help prove the base case, while Claim 4 which, working with Claim 2 and Claim 3, helps prove the inductive step.

Claim 1: Tree-like structure of $G5$ s (No reconverging paths of $G5$ s)

Given an arbitrary $G5$, x , on the top layer and G , starting at any $G5$ at the bottom layer there is only one way to traverse the $G5$ s to get to x .

Proof

Assume we start at an arbitrary $G5$, call it x , on the top layer, and from the construction it should be obvious that a path exists from any $G5$ at the bottom layer to x (i.e., x is connected to every $G5$ on the bottom layer). Now let us consider the maximum number of $G5$ s we are connected to after one layer, which is 5 (since x has 5 inputs). Similarly, after two layers the maximum number of $G5$ s that are connected to x is 5^2 (since x has 5 inputs and the $G5$ s that feed its inputs also have 5 inputs), and the maximum number of $G5$ s that can reach x at layer L (after $L-1$ layers) is 5^{L-1} . Now, if there were any reconverging paths connecting x to the rest of the $G5$ s, there would be strictly less than 5^{L-1} $G5$ s at the bottom layer that can reach x . By construction it should be obvious that every $G5$ at the bottom layer can reach x , therefore there are no reconverging paths.

Claim 2: Mapping upward (layer i)

Mapping the nodes in $G5$ 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 $G5$ on layer i are already LUTs, we know that the *optimal* mapping of each $G5$ 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 $G5$) has to split into two separate LUTs, thereby creating one additional LUT. This is because every output of a particular $G5$ in layer i will never combine with another output from that $G5$ 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 $G5$ 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 $G5$ 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 $G5$ 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 $G5$ on layer $i+1$ using $N-1$ 4-LUTs and one 3-LUT? Consider Figure 6 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 $G5$.

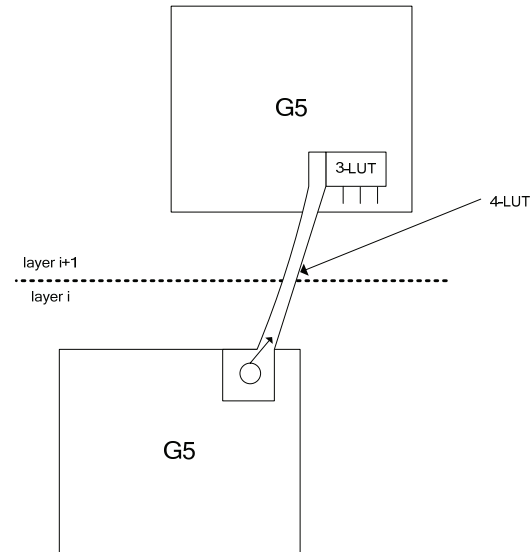


Figure 6. A LUT spanning two layers

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 $G5$ at layer i are already LUTs. We know that in the optimal mapping of each $G5$ 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 $G5$ 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 above 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 G5 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 G5 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 a: Mapping exactly all the nodes of layer 1 and not mapping any nodes of layer 2.

Since there is no overlap between the G5s (thus trying to pack nodes from different G5s into one LUT cannot possibly reduce area) and we know the optimal mapping of G5, the optimal mapping of this layer will result in mapping each G5 separately.

Case b: 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 b cannot happen and case a must happen. From case a we can see that P(1) must hold, thus the base case is proved.

Inductive step

$P(m-1) \Rightarrow P(m)$.

Proof

Recall that P(m) is saying that the optimal mapping for all nodes up to layer m is the optimal mapping of each G5 separately. Since P(m-1) is assumed, we know that the optimal mapping for all nodes up to layer m-1 is the optimal mapping of each G5 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 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 m+1. And 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 G5 separately, and the inductive step is proven. Therefore, by induction, the optimal mapping of G is that which maps every G5 optimally and separately.