FPGA Technology Mapping: A Study of Optimality

Andrew Ling
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Canada
aling@eecg.toronto.edu

Deshanand P. Singh
Altera Corporation Toronto Technology Centre
Toronto, Canada
dsingh@altera.com

Stephen D. Brown
Altera Corporation Toronto Technology Centre
Toronto, Canada
sbrown@altera.com

ABSTRACT

This paper attempts to quantify the optimality of FPGA technology mapping algorithms. We develop an algorithm, based on Boolean satisfiability (SAT), that is able to map a small subcircuit into the smallest possible number of lookup tables (LUTs) needed to realize its functionality. We iteratively apply this technique to small portions of circuits that have already been technology mapped by the best available mapping algorithms for FPGAs. In many cases, the optimal mapping of the subcircuit uses fewer LUTs than is obtained by the technology mapping algorithm. We show that for some circuits the total area improvement can be up to 67%.

Categories and Subject Descriptors

B.6.3 [**Hardware**]: Logic Design - *Design Aids*

General Terms

Algorithms, Experimentation, Performance

Keywords

Boolean Satisfiability, Resynthesis, Optimization, Cone, FPGA, Lookup Table

1. INTRODUCTION

FPGAs (Field Programmable Gate Arrays) are reconfigurable integrated circuits that are characterized by a sea of programmable logic blocks surrounded by a programmable routing structure. Most modern FPGA devices contain programmable logic blocks that are based on the K-input lookup table (K-LUT) where a K-LUT contains 2^K truth table configuration bits so it can implement any K-input function. Figure 1 illustrates the general structure of a 2-LUT. The number of LUTs needed to implement a given circuit determines the size and cost of the FPGA-based realization. Thus one of the most important phases of the FPGA CAD flow is the technology mapping step that maps an optimized circuit description into a LUT network present in the target FPGA architecture. The goal of the technology mapping step is to reduce area, delay, or a combination thereof in the network of programmable logic blocks that is produced. In this work, we assess state-of-the-art FPGA technology mapping algorithms in terms of *area*-optimality. Timing-driven technology mapping is not covered in this study.

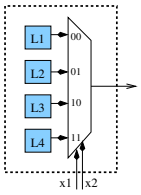


Figure 1: 2-input LUT

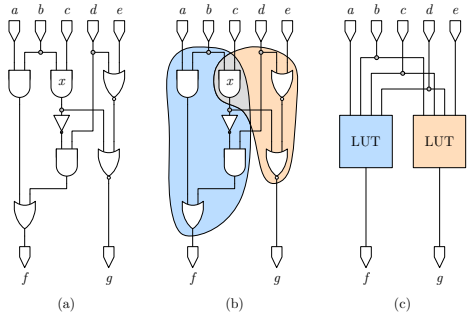


Figure 2: Technology mapping as a covering problem. (a) Original Netlist (b) Possible Covering (c) LUT Mapping from Covering

The process of technology mapping is often treated as a covering problem. For example, consider the process of mapping a circuit into LUTs as illustrated in Figure 2. Figure 2a illustrates the initial gate-level network, Figure 2b illustrates a possible covering of the initial network using 4-LUTs, and Figure 2c illustrates the LUT network produced by the covering. In the mapping given, the gate labeled x is covered by both LUTs and is said to be *duplicated*. Somewhat surprisingly, gate duplication is often necessary to minimize the area of LUT networks [6].

The fundamental question that we ask in this paper is: Given the LUT-level network created by a technology mapping algorithm, how much can its area be reduced? For small subcircuits, it is possible to answer this question in an optimal manner. Consider an arbitrary function $f(i_0, i_1, \ldots, i_n)$. Suppose that we seek to determine if it can be implemented in three or fewer 2-LUTs. This problem can be solved by

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, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DAC 2005, June 13–17, 2005, Anaheim, California, USA.
Copyright 2005 ACM 1-59593-058-2/05/0006 ...$5.00.

considering the Configurable Virtual Network (CVN) shown in Figure 3. The CVN consists of input lines connected to the variables $i_0 \ldots i_n$ and three 2-LUTs. A crossbar allows the LUT inputs to select any of the input lines or outputs from other LUTs. Each "switch" on the crossbar is configured by a virtual configuration bit. A **0** indicates that the crosspoint intersection is unconnected, while a **1** indicates a connection at the crosspoint. Clearly, it is possible to enumerate every possible circuit configuration involving three 2-LUTs by manipulating the virtual configuration bits for the crossbar as well as the truth table configuration bits for each of the 2-LUTs. In fact, as detailed in Section 3, we can express the output of the network f_{net} as a Boolean formula involving the variables $i_0 \ldots i_n$, the virtual configuration bits $V_1 \ldots V_m$ and the truth table configuration bits $L_1 \ldots L_o$. Given this formula, we ask the question: is there an assignment of $V_1 \ldots V_m$ and $L_1 \ldots L_o$ that will cause $f_{net}(i_0, i_1, \ldots, i_n, V_1 \ldots V_m, L_1 \ldots L_o)$ to be identical to $f(i_0, i_1, \ldots, i_n)$ for all values of $i_0 \ldots i_n$? This question can be answered exactly with Boolean satisfiability (SAT): Given a Boolean expression in Conjunctive-Normal-Form (CNF), where the expression consists of a product of clauses and each clause consists of a sum of literals, seek an assignment of variables so that each clause has at least one literal set to true. The solution space of this problem grows exponentially with respect to the input size, n, of the virtual network. However, we show in this paper that modern SAT solvers can be used to exactly resynthesize small circuits.

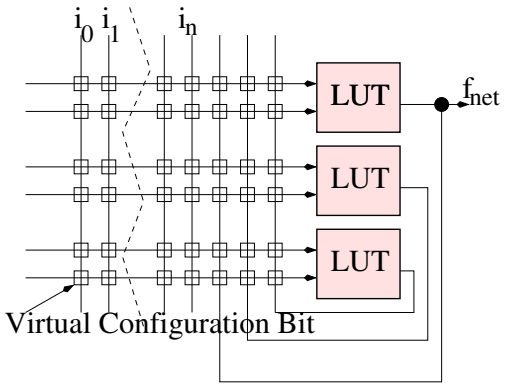


Figure 3: Configurable Virtual Network

Given this optimal resynthesis method for small subcircuits, we simply iteratively apply this technique to small portions of a larger circuit in a sliding window fashion until no additional improvement can be achieved. This approach does not guarantee the mapping optimality of the large circuit, but it does give us some indication of the area "left on the table" by the original technology mapping solution.

The remainder of this paper is organized as follows. First we review some of the key literature on area-driven LUT mapping. We then describe our optimal resynthesis approach based on SAT. Next, we describe the application of our optimal resynthesis approach to the 4-LUT networks produced by the best known FPGA technology mapper and provide a set of results. Finally, we provide concluding remarks and directions for future work.

2. BACKGROUND

Due to the popularity of LUT-based FPGA architectures [1, 18], a large body of existing work has studied area-driven LUT mapping algorithms. We review some of the key literature here.

The area minimization problem was shown to be NP-hard for LUTs of input size four and greater [15, 10]. Thus, heuristics are necessary to solve the area minimization problem in a reasonable amount of computation time. Early work considered the decomposition of circuits into a set of trees which were then mapped for area [13, 11]. The area minimization problem for trees is much simpler and can be solved optimally using dynamic programming. However, real circuits are rarely structured as trees and tree decomposition prevents much of the optimization that can take place across tree boundaries.

In a *duplication-free* mapping, each gate in the initial circuit is covered by a single LUT in the mapped circuit. The area minimization problem in duplication-free mapping can be solved optimally by decomposing the circuit into a set of maximum fanout free cones (MFFCs) which are then mapped for area [4]. Although the area minimal duplication-free mapping is significantly smaller than the area minimal tree mapping, the controlled use of duplication can lead to further area savings. In [6], heuristics are used to mark a set of gates as *duplicable*. Area optimization is then considered within an extended fanout free cone (EFFC) where an EFFC is an MFFC that has been extended to include duplicable gates.

There is also a new class of algorithms that attempt to reduce the area of a circuit that has already been technology mapped using Sets of Pairs of Functions to be Differentiated SPFDs ([12, 19]). SPFDs involve using "don't care" information to express alternative functional permissibilities for the truth table of each LUT in a particular design. In many cases, it allows us to explore alternate and equivalent functional representations of the circuit which may reduce area.

2.1 Terminology

The remainder of this paper uses standard nomenclature for describing technology mapping. This is as follows: The combinational portion of a LUT network can be represented as a directed acyclic graph (DAG). A node in the graph represents a LUT, primary input (PI), or primary output (PO). A directed edge in the graph with head u, and tail v, represents a signal in the logic circuit that is an output of node u and an input of node v.

A *cone* of v, C_v, is a subgraph consisting of v and some of its non-PI predecessors such that any node $u \in C_v$ has a path to v that lies entirely in C_v. Node v is referred to as the *root* of the cone. At a cone C_v, the set of input edges is the set of edges with a tail in C_v and the set of output edges is the set of edges with v as a head. The fanin size of a cone is the number of input edges. A cone with n input edges is known to be n-feasible and can be trivially implemented with a n-LUT. A fanout free cone (FFC) is a cone with output edges only originating from the root of the cone. A maximum fanout free cone (MFFC) is a FFC maximizes the number of nodes contained in the FFC.

3. RESYNTHESIZING FOR AREA

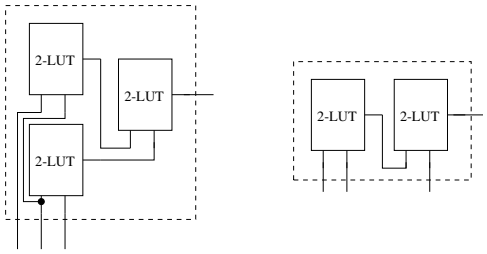
When resynthesizing for area, one must take an existing LUT circuit and attempt to reduce the number of LUTs

in the circuits yet maintain the original functionality. The more LUTs that can be removed, the farther the original circuit is from the optimal mapping.

We mentioned previously that reducing the number of LUTs can be achieved by resynthesizing smaller subcircuits and applying this in a sliding window fashion over the larger circuit. The subcircuits that we focus on form a cone. Thus by resynthesizing several cones, this will reduce the LUT count of the overall LUT network.

3.1 Converting Resynthesis Problem into Boolean Satisfiability

Determining if an n-feasible cone implemented with X n-LUTs can be resynthesized into an n-feasible cone implemented with X-1 n-LUTs or less can be verified with SAT This is achieved by generating a n-feasible FFC containing less LUTs than the original cone, then expressing the FFC as a Boolean expression in CNF. Next, the CNF expression is assigned the truth table values of the function expressed by the original cone. If this CNF expression is satisfiable, resynthesis to the new FFC is possible.

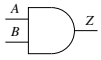


(a) Original Cone (b) Resynthesized Cone

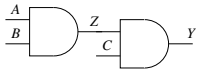
Figure 4: Resynthesis of Three Input Cone of Logic

To illustrate this process, consider Figure 4. The original cone 4a consists of three 2-LUTs which implements a three input function. Since only three inputs enter the cone, it may be possible to resynthesize 4a into 4b to save one LUT.

To determine if resynthesis from 4a to 4b is possible, 4b must be converted into a CNF expression (a detailed description of converting digital circuits to CNF expressions can be found in [14]). As stated previously, if the expression is satisfiable, resynthesis can occur.



$$F_1(A,B,Z) = (A+\overline{Z})\cdot(B+\overline{Z})\cdot(\overline{A}+\overline{B}+Z) \quad (1)$$



$$\begin{aligned} F_2(A,B,C,Z,Y) =& (A+\overline{Z})\cdot(B+\overline{Z})\cdot(\overline{A}+\overline{B}+Z) \\ & \cdot(Z+\overline{Y})\cdot(C+\overline{Y})\cdot(\overline{Z}+\overline{C}+Y) \end{aligned} \quad (2)$$

Figure 5: AND gate CNF formulae

The CNF equation serves to express all valid vectors of the circuit. For example, consider Figure 5. Equation 1 will be satisfied if and only if the signals A,B, and Z correspond to an AND gate functionality (e.g. $(A=0,B=X,Z=0)$, $(A=X,B=0,Z=0)$ or $(A=1,B=1,Z=1)$). Similarily, equation 2 will be satisfied only for valid vectors such as $(A=1,B=1,C=0,Z=1,Y=0)$.

To apply this for resynthesis checking, we first form the CNF equation. Next, we constrain the cone input and output variables in the CNF equation according to the truth table of the cone. Finally, we check for a satisfiable assignment using a SAT solver. For example, let us attempt to map a two-input cone to the second circuit in Figure 5. Consider input $\boldsymbol{C}$ to be a configuration bit. To check if the two-input function $f(1,1)=1$ is feasible, we determine if $F_2(A=1,B=1,C,Z,Y=1)$ is satisfiable. Clearly, this will return true with $C=1$. However, this only shows that $f(1,1)=1$ is possible (i.e. one truth table cube). To verify if a resynthesis circuit can implement an entire cone (i.e. an entire truth table), its CNF equation is replicated 2^n times to form a new CNF equation, where n represents the fanin size of the cone being mapped. Each replicant of the basic circuit CNF equation represents one cube in the cone's truth table. Again, SAT is performed on the new CNF formula to check if the original cone can fit into the resynthesis circuit.

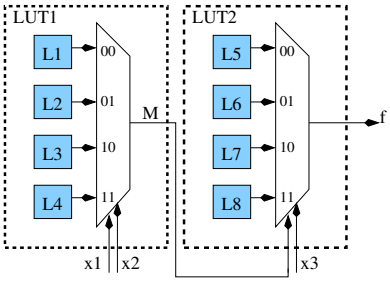


Figure 6: Detailed Diagram of Resynthesis Logic.

Going back to the original LUT example shown in Figure 4, the following steps illustrate how this conversion occurs. Figure 6 is a detailed picture of Figure 4b with internal wires and configuration bits clearly labeled for CNF construction. In steps to come, f represents the function of the cone under consideration for resynthesis, $\boldsymbol{X_i}$ represents input vector $x_1x_2x_3=i$, and $f_i=f(\boldsymbol{X_i})$.

Step 1: Create a CNF equation for individual elements in resynthesis circuit.

$$\begin{aligned} G_{LUT1} = & (x_1+x_2+\overline{L_1}+M)\cdot(x_1+x_2+L_1+\overline{M}) \\ & \cdot(x_1+\overline{x_2}+\overline{L_2}+M)\cdot(x_1+\overline{x_2}+L_2+\overline{M}) \\ & \cdot(\overline{x_1}+x_2+\overline{L_3}+M)\cdot(\overline{x_1}+x_2+L_3+\overline{M}) \\ & \cdot(\overline{x_1}+\overline{x_2}+\overline{L_4}+M)\cdot(\overline{x_1}+\overline{x_2}+L_4+\overline{M}) \end{aligned} \quad (3)$$

$$\begin{aligned} G_{LUT2} = & (x_3+M+\overline{L_5}+f)\cdot(x_3+M+L_5+\overline{f}) \\ & \cdot(x_3+\overline{M}+\overline{L_6}+f)\cdot(x_3+\overline{M}+L_6+\overline{f}) \\ & \cdot(\overline{x_3}+M+\overline{L_7}+f)\cdot(\overline{x_3}+M+L_7+\overline{f}) \\ & \cdot(\overline{x_3}+\overline{M}+\overline{L_8}+f)\cdot(\overline{x_3}+\overline{M}+L_8+\overline{f}) \end{aligned} \quad (4)$$

Step 2: Formulate the circuit CNF equation from equations 3 and 4. Note that equation 5 is an expression dependent on the circuit's inputs and output (x_{1-3},f), internal wire (M), and configuration (L_{1-8}) variables.

$$G_{resynth}(\boldsymbol{X_i}, f_i) = G_{LUT1}\cdot G_{LUT2} \quad (5)$$

Step 3: Replication of equation 5 and constraining of inputs and output variables according to function f.

$$\begin{aligned} G_{Total} = \; & G_{resynth}(\boldsymbol{X_0}, f_0) \cdot G_{resynth}(\boldsymbol{X_1}, f_1) \\ & \cdot G_{resynth}(\boldsymbol{X_2}, f_2) \cdot G_{resynth}(\boldsymbol{X_3}, f_3) \\ & \cdot G_{resynth}(\boldsymbol{X_4}, f_4) \cdot G_{resynth}(\boldsymbol{X_5}, f_5) \\ & \cdot G_{resynth}(\boldsymbol{X_6}, f_6) \cdot G_{resynth}(\boldsymbol{X_7}, f_7) \end{aligned} \quad (6)$$

In equation 6, the configuration bits are represented by the same variables (L_{1-8}) in each $G_{resynth}(\boldsymbol{X_i}, f_i)$ instance, whereas all other signals are given unique variables in each instance. This ensures that only one configuration will exist for all cubes of the truth table. Finally, equation 6 is passed into a SAT solver which will return true if the cone fits in the resynthesis structure.

For simplicity, in the previous example we ignored the flexibility of FPGA routing which allow LUT inputs to be permuted. This is extremely important since it increases the number of functions a given resynthesis structure can represent. For example, Figure 7 shows how a three input function can be converted to another by simply swapping inputs x_1 and x_2. Extending Figure 7 to all input permutations

$x_1x_2x_3$	f
000	1
001	1
010	1
011	0
100	0
101	1
110	0
111	1

(a) $f(x_1, x_2, x_3)$

$x_1x_2x_3$	f
000	1
001	1
010	0
011	1
100	1
101	0
110	0
111	1

(b) $f(x_2, x_1, x_3)$, x_1 and x_2 swapped

Figure 7: Conversion from $f(x_1, x_2, x_3)$ to $f(x_2, x_1, x_3)$

increases the number of functions a resynthesis structure represents by a factor of $n!$, where n is the fanin size of the resynthesis cone. In order to represent permutable inputs in our CNF expression, virtual multiplexers are added to the inputs shown in Figure 8. These are virtual in the sense that they do not exist in the resynthesis structure, but only serve to allow us to permute the inputs in the CNF expression for SAT. To add these virtual MUX's, one would simply add

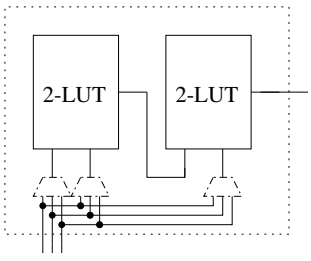
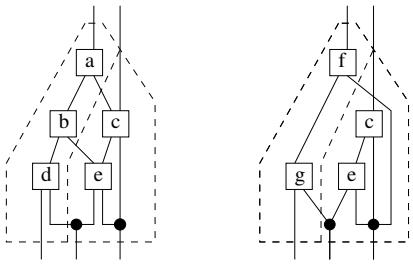


Figure 8: Resynthesis structure with Virtual Multiplexors



(a) Original Cone (b) Resynthesized Cone

Figure 9: Multiple Output Cone for Resynthesis

the CNF equations for the virtual MUX's in **Step 1** of our process and continue as shown previously.

3.2 Generation of Cones

A version of the algorithm described in [17, 7] is used to generate and store all resynthesis cones in the graph. The resynthesis cones are generated as the graph is traversed in topological order from PIs to POs. At every internal LUT, new cones are generated by combining the cones at the input LUTs. In contrast with [7], which combined the cones in every possible way, in our work, the cone generation algorithm combines cones if they have no more $(n+e)$ inputs in total, where n is the fanin size of the largest resynthesis cone and e is an expansion size. As long as e was set to a sufficiently high number (8 in the experiments), this heuristic sped up the cone generation process without significantly impacting the quality of the resynthesis solution.

Special consideration must be taken for cones with more than one fanout. For example, consider Figure 9. If LUTs a through e are resynthesized to LUTs f and g, LUTs c and e must be duplicated to keep the fanout at LUT c. Thus the total savings by this resynthesis is only one, as opposed to three if there was no fanout at LUT c.

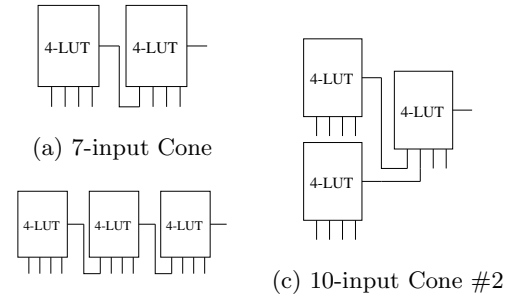
4. RESULTS

We performed resynthesis on circuits produced by the ZMap techmapper — one of the best publically available FPGA area-driven techmappers developed by J. Cong et al. at UCLA [5]. Given a set of circuits, we used ZMap to technology map these circuits to **4**-LUTs. After some post processing done by RASP [5] to further improve area, we ran our resynthesizer [1] on these LUT networks.

The number of resynthesis structures is countless; however, considering that the size of the CNF equation is exponential to the number of resynthesis structure inputs, for practical purposes, our work dealt with cones of fanin size 10 or less. This limits the number of resynthesis structures to the ones shown in Figure 10. Figure 10a is applied for cones with a fanin size of seven or less and containing more than two 4-LUTs; Figures 10b and 10c are applied for cones with a fanin size of 10 or less and containing more than three 4-LUTs. Resynthesis checking was done using the Chaff SAT solver developed by M. W. Moskewicz et al. [16].

In order to reduce the number of candidate cones for

[1]Our resynthesizer was incorporated with the Berkeley MV-SIS project [3]



(a) 7-input Cone

(b) 10-input Cone #1

(c) 10-input Cone #2

Figure 10: Resynthesis Structures

Circuit	*Resynth*	*ZMap*	*Ratio*
clma	4792	5014	0.95
b15_1	4112	4291	0.95
b15_1_opt	3772	3879	0.97
s38584.1	3454	3771	0.91
s38417	3444	3586	0.96
b14	2902	3072	0.94
frisc	2571	2624	0.98
pdc	1875	1928	0.97
misex3	1156	1184	0.98
seq	1162	1182	0.98
alu4	1103	1129	0.98
ex5p	968	993	0.97
i10	764	789	0.97
Total	32075	33442	0.96

Table 1: Benchmark Circuit Resynthesis Results.

resynthesis, at most only two LUTs were allowed to be duplicated. Experiments showed that increasing the duplication count above two increased the set of cones to the point that simulations would take an extraordinarily long time to execute.

4.1 Benchmark Circuits

In our first set of experiments, we focused on a set of circuits taken from the MCNC and ITC'99 benchmark suites ([20],[8]). These circuits were optimized using SIS [9] and RASP, technology mapped with ZMap, and resynthesized with our work. The optimization in SIS is particularly important since the structure of the gate-level netlist can have a significant impact on the mapped area. Table 1 shows the results. The *ZMap* column indicates the number of 4-LUTs the circuit was technology mapped to. The *Resynth* column indicates the number of 4-LUTs after our resynthesis. The results clearly show that ZMap does not achieve optimal results; this implies that all FPGA techmappers that perform worse then ZMap also have much room for improvement. Notice that the largest decreases in area are seen in circuits with 3000 LUTs or more, with the largest decrease of more than 9% (s38584.1). This suggests that the deviation from the optimal solution is proportional to the size of the circuit. The fact that area driven technology mapping is NP-hard [10] supports this claim.

Building Block	*Resynth*	*ZMap*	*Ratio*
4:1 MUX	2	3	0.67
16:1 MUX	21	29	0.72
32-Bit Priority Encoder	59	74	0.80
4-Bit Barrel Shifter	8	12	0.67
16-Bit Barrel Shifter	32	48	0.67
6-Bit Set Reset Checker	2	3	0.67
2-Bit Sum Compare Constant	2	6	0.33
2-Bit Sum Compare	2	3	0.67
6-Bit Priority Checker	3	6	0.50
8-Bit Bus Multiplexor	16	24	0.67
Total	188	253	0.74

Table 2: Logic Block Resynthesis Results.

4.2 Building Block Circuits

In our second set of experiments, we focused on common digital circuit logic blocks. We started from Verilog code, synthesized it using VIS [2], then optimized and techmapped the circuits as in Section 4.1. For illustration, Module 1 shows the original Verilog code that we synthesized then resynthesized. More code is shown in the Appendix for further reference. Table 2 shows our results, where we

Module 1 16-Bit Barrel Shifter Verilog Code

```
module BarrelShifter16Bit(SHIFT,D,Q)
  input[1:0] SHIFT;input[15:0] D;
  output[15:0] Q;reg[15:0] Q;
  always @ (D or Q or SHIFT)
    case (SHIFT)
      2'b00 :  Q=D;
      2'b01 :  Q={D[3:0],D[15:4]};
      2'b10 :  Q={D[7:0],D[15:8]};
      2'b11 :  Q={D[11:0],D[15:12]};
    endcase
endmodule
```

achieve a reduction as large as 67% and an average reduction of 26%. Since these logic blocks are common in digital circuits, heuristics can be used to identify them and technology map them to our optimized circuits.

It is interesting to note the dramatic differences in results between the benchmark circuits and the results of the individual building blocks. We speculate that common building blocks are being collapsed with other random or glue logic in the benchmarks. Since we limit the size of the subcircuit resynthesis procedure, it is likely that we are missing some key resynthesis opportunities.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a method that helps us to understand the optimality of state-of-the-art FPGA technology mapping algorithms. Our approach involves optimally resynthesizing small portion of the circuit until no further improvement can be found. This approach is itself non-optimal. However, if this localized optimal resynthesis approach is able to improve the mapping result from an existing technology mapping algorithm, then it gives us an indication of the mapper's "distance" from optimality.

Our future research will explore methods to improve this localized optimal resynthesis approach. First, we are examining custom hardware acceleration approaches to improve the cone resynthesis runtime. We are also exploring the use of "don't cares" to reduce the complexity of the CNF generation as well as adding flexibilty to the resynthesis search space.

Also, we would like to incorporate our work into a post-processing step for K-LUT technology mapping since our results clearly show that conventional K-LUT technology mappers perform poorly on some very common logic blocks. This would first involve caching several optimal logic block configurations found by our resynthesizer. Next, resynthesizing technology mapped circuits by replacing entire logic blocks with the optimized configuration found in our cache. For circuits consisting of several logic blocks found in our cache, this would lead to a significant area reduction.

6. REFERENCES

[1] Altera. Component selector guide ver 14.0, 2004.

[2] R. K. Brayton and G. D. H. et al. VIS: a system for verification and synthesis. In *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, pages 428–432, 1996.

[3] D. Chai, J. Jiang, Y. Jiang, Y. Li, A. Mishchenko, and R. Brayton. MVSIS 2.0 Programmer's Manual, UC Berkeley. Technical report, 2003.

[4] J. Cong and Y. Ding. On area/depth trade-off in LUT-based FPGA technology mapping. In *Design Automation Conference*, pages 213–218, 1993.

[5] J. Cong, J. Peck, and Y. Ding. RASP: A general logic synthesis system for SRAM-based FPGAs. In *FPGA*, pages 137–143, 1996.

[6] J. Cong, C. Wu, and Y. Ding. Cut ranking and pruning: enabling a general and efficient fpga mapping solution. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, pages 29–35. ACM Press, 1999.

[7] J. Cong, C. Wu, and Y. Ding. Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution. In *FPGA*, pages 29–35, 1999.

[8] F. Corno, M. Reorda, and G. Squillero. RT-level ITC 99 benchmarks and first ATPG results, 2000.

[9] L. L. C. M. R. M. A. S. H. S. P. R. S. R. K. B. E. M. Sentovich, K. J. Singh and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, 1992.

[10] A. Farrahi and M. Sarrafzadeh. Complexity of the Lookup-Table Minimization Problem for FPGA Technology Mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(11):1319–1332, 1994.

[11] R. J. Francis, J. Rose, and K. Chung. Chortle: a technology mapping program for lookup table-based field programmable gate arrays. In *Proceedings of the 27th ACM/IEEE conference on Design automation*, pages 613–619. ACM Press, 1990.

[12] W. L. Jason Cong, Joey Y. Lin. Spfd-based global rewiring. In *Proceeding of International Symposium on FPGAs*, pages 77–84, February 2002.

[13] K. Keutzer. Dagon: Technology binding and local optimization by dag matching. In *DAC*, pages 341–347, 1987.

[14] T. Larrabee. Test Pattern Generation Using Boolean Satisfiablity. *IEEE Transactions on Computer-Aided Design*, 11(1):6–22, 1992.

[15] I. Levin and R. Y. Pinter. Realizing Expression Graphs using Table-Lookup FPGAs. In *Proceedings of the European Design Automation Conference*, pages 306–311, 1993.

[16] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.

[17] M. D. F. Schlag, J. Kong, and P. K. Chan. Routability-driven techology mapping for lookup-table-based fpgas. In *Proceedings of the 1991 IEEE International Conference on Computer Design on VLSI in Computer & Processors*, pages 86–90. IEEE Computer Society, 1992.

[18] Xilinx. Virtex-ii complete data sheet ver 3.3, 2004.

[19] S. Yamashita, H. Sawada, and A. Nagoya. A new method to express functional permissibilities for LUT based FPGAs and its applications. In *ICCAD*, pages 254–261, 1996.

[20] S. Yang. Logic synthesis and optimization benchmarks user guide version, 1991.

7. APPENDIX

Module 2 Verilog Code for Building Blocks

```
module SetResetChecker6Bit(D,f)
  input[5:0] D;
  output f; reg f;
  always @ (D)
    f = (D == 6'h00 || D == 6'hff) ?  1:0;
endmodule

module SumCompare2Bit(A,B,C,f)
  input[1:0] A,B,C;
  output f;
  reg f;
  always @ (A or B or C)
    f = (A+B == C) ? 1:0;
endmodule

module PriorityChecker6Bit(D,f)
  input[5:0] D;
  output f;
  reg f; reg[2:0] i,sum;
  always @ (D)
    for (i=0;i<6;i=i+1)
      sum = (i==0) ?  [i]:D[i]+sum;
    f = (sum>=3'b011) ?  1:0;
endmodule
```