Using the Minimum Set of Input Combinations to Minimize the Area of Local Routing Networks in Logic Clusters Containing Logically Equivalent I/Os in FPGAs

Andy Gean Ye, *Member, IEEE*

***Abstract*—Mapping digital circuits onto field-programmable gate arrays (FPGAs) usually consists of two steps. First, circuits are mapped into look-up tables (LUTs). Then, LUTs are mapped onto physical resources. The configuration of LUTs is usually determined during the first step and remains unchanged throughout the second. In this paper, we demonstrate that by reconfiguring LUTs during the second step, one can increase the flexibility of FPGA routing resources. This increase in flexibility can then be used to reduce the implementation area of FPGAs. In particular, it is shown that, for a logic cluster with I inputs and N k-input LUTs, a set of $N \times k \ \ (I + N - k + 1) \ : \ 1$ multiplexers can be used to connect logic cluster inputs to LUT inputs while maintaining logic equivalency among the logic cluster I/Os. The multiplexers (called a local routing network) are shown to be the minimum required to maintain logic equivalency. Comparing to the previous design, which employs a fully connected local routing network, the proposed design can reduce logic cluster area by 3%–25% and can reduce a significant amount of fanouts for logic cluster inputs.**

***Index Terms*—Area efficiency, field-programmable gate arrays (FPGAs), local routing networks, logic clusters.**

I. Introduction

FIELD-PROGRAMMABLE gate arrays (FPGAs) typically connect look-up tables (LUTs) through a two-level routing hierarchy. First, local routing networks are used to connect LUTs into logic clusters. Then, global routing networks are used to connect logic clusters into FPGAs. Since routing networks usually consume a vast majority of FPGA area [1], it is important to increase their flexibility while minimizing their area.

One way of increasing the flexibility of routing networks is to use logic clusters with logically equivalent inputs and outputs. In particular, logically equivalent inputs allow signals to enter a logic cluster through any of the input pins, and logically equivalent outputs allow signals to exit a logic cluster through any of the output pins. This flexibility in I/O selection can increase the area efficiency of global routing networks and lead to an increase in FPGA area efficiency [2]. Many FPGA architectural studies (for example,[1], [3]–[5]) are based on logic clusters with logically equivalent I/Os. Other logic cluster designs are compared against the logically equivalent ones [6]–[9]. So, it is important to understand the minimum area required to implement a logic cluster with logically equivalent I/Os.

In this study, we investigate the minimum area required to implement logic clusters with logically equivalent I/Os. We create the minimum area design by exploiting a link between LUT reconfiguration and routing flexibility. The design is implemented in multiplexers and is proven to consume minimum area among all multiplexer-based designs with logically equivalent I/Os. To the best of the author's knowledge, this is the first work that investigates the minimum area design for logic clusters containing logically equivalent I/Os.

The rest of this paper is organized as follows. Section II motivates the research, Section III discusses the LUT structure used in this study, Section IV examines the effect of LUT reconfiguration on routing flexibility, Section V presents the design of the minimum area local routing network, Section VI investigates the area implications of the minimum area design, Section VII examines the limitations of the current work and suggests future research directions, and finally, Section VIII concludes.

II. Background and Motivation

Logically equivalent I/Os allow a signal to enter/exit a logic cluster in several ways. This added connectivity increase the flexibility of the routers and can lead to better utilization of the routing resources. As an example, consider the circuit shown in Fig. 1(a). In the figure, there are four logic clusters. Each contains a set of nonequivalent I/Os—each input signal must enter the cluster through a dedicated cluster input and each output signal must exit the cluster through a dedicated cluster output. As shown, to connect the clusters, a maximum channel width of three tracks per channel is required. In Fig. 1(b), on the other hand, all clusters have logically equivalent I/Os—an input signal can enter a cluster through any of the three cluster inputs and an output signal can exit the cluster through any of the two cluster outputs. The input signals of logic clusters A and D can now be assigned to new logic cluster inputs and the output signal of D can be assigned to a new logic cluster output. As shown in Fig. 1(b), the new assignment creates a functionally equivalent

Manuscript received June 01, 2008; revised September 30, 2008. First published June 30, 2009; current version published December 23, 2009.

The author is with the Department of Electrical and Computer Engineering, Ryerson University, Toronto, ON M5B 2K3, Canada (e-mail: aye@ee.ryerson.ca).

Digital Object Identifier 10.1109/TVLSI.2008.2008188

1063-8210/$26.00 © 2009 IEEE

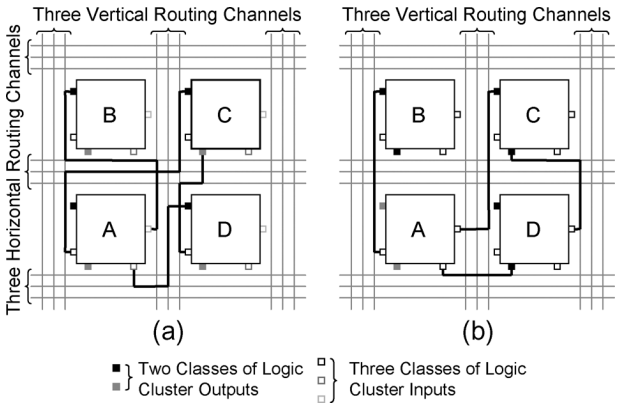


Fig. 1. Logically equivalent logic cluster I/Os. (a) Logically nonequivalent I/O pins. (b) Logically equivalent I/O pins.

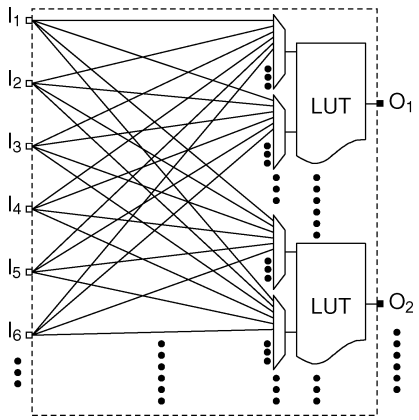


Fig. 2. Fully connected local routing network.

circuit that requires much less routing tracks to implement, resulting in a one track per channel routing solution.

Previous work achieves logic equivalency through fully connected local routing networks [1]–[5]. As shown in Fig. 2, the network connects an LUT input to each of the logic cluster inputs through an I:1 multiplexer, where I is the number of logic cluster inputs that the cluster contains. The full connectivity allows each LUT input to be connected to any of the logic cluster inputs without affecting the connectivity of any other LUT inputs—resulting in a set of logically equivalent logic cluster inputs. It also decouples the signal assignment of the logic cluster inputs from the logic assignment of the LUTs. Each LUT can be used to implement any Boolean function assigned to the cluster without affecting the assignment of the logic cluster inputs—resulting in a set of logically equivalent logic cluster outputs.

The fully connected network, however, is not the most area efficient method of achieving logic equivalency, and its area inefficiency can be illustrated through an extreme example, where the multiplexers can be completely eliminated through LUT reconfiguration. Consider a logic cluster containing 1 four-input LUT, four logic cluster inputs, and one logic cluster output, as shown in Fig. 3. In Fig. 3(a), a fully connected local routing network is used to connect the logic cluster inputs to each LUT input. A signal assigned to LUT input L_1, e.g., can enter the cluster through any of the logic cluster inputs I_1, I_2, I_3, or I_4,

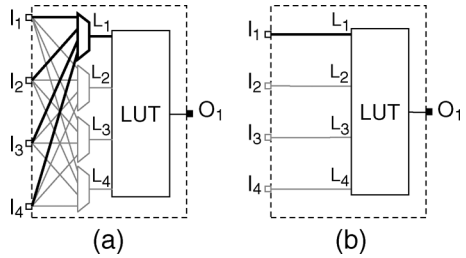


Fig. 3. Fully connected local routing network versus LUT reconfiguration. (a) Logical equivalency through local routing network. (b) Logical equivalency through LUT reconfiguration.

and be connected to L_1 through the 4:1 multiplexer attached to L_1.

LUT reconfiguration, however, can achieve the same effect, without the 4:1 multiplexers. For example, as shown in Fig. 3(b), each LUT input can be directly connected to a cluster input. Here, for a four-input Boolean function such as the one shown in Fig. 4(a), a signal assigned to L_1 can be routed through cluster input I_1. The same function, however, can be implemented by exchanging the signal assignment of L_1 and L_2, and by reconfiguring the LUT to implement the Boolean function shown in Fig. 4(b). The signal originally assigned to L_1 now must enter the cluster through logic cluster input I_2. Similarly, the same signal can be made to enter the cluster through logic cluster inputs I_3 and I_4, respectively, by using the LUT configurations shown in Fig. 4(c) and Fig. 4(d).

This paper addresses the following three questions: 1) for a logic cluster with N k-input LUTs, N feedback signals (one per registered LUT output), and I logic cluster inputs, what is the most area efficient method of implementing a local routing network with logically equivalent logic cluster inputs and logic cluster outputs? 2) In this minimum area implementation, how should the feedbacks and the logic cluster inputs be connected to the LUT inputs? 3) In this minimum area implementation, how should the LUTs be reconfigured to maintain logic equivalency among the logic cluster inputs and outputs?

Note that Fig. 5 shows a logic cluster with 2 four-input LUTs, two feedbacks, six logic cluster inputs, and a fully connected local routing network. The cluster will be used as an illustrative example throughout the paper.

III. Structures and Properties of LUTs

A k-input LUT is designed to emulate the operation of a 2^k entry truth table. Its structure is shown in Fig. 6. As shown, the LUT is constructed out of a 2^k:1 multiplexer and 2^k bits of configuration memory. The memory is connected to the data inputs of the multiplexer and stores the truth table entries. The LUT inputs are connected to the select inputs of the multiplexer. It is assumed that the multiplexer is implemented using pass transistor logic. An example of such an implementation is shown for a four-input LUT in Fig. 7.

In this study, we use three properties of an LUT to determine the minimum area required to implement a logic cluster containing logically equivalent I/Os. These properties are described in turn. First, as shown in Fig. 4, any two inputs of an

L1	L2	L3	L4	O	L1	L2	L3	L4	O	L1	L2	L3	L4	O	L1	L2	L3	L4	O
0	0	0	0	f0	0	0	0	0	f0	0	0	0	0	f0	0	0	0	0	f0
0	0	0	1	f1	0	0	0	1	f1	0	0	0	1	f1	0	0	0	1	f8
0	0	1	0	f2	0	0	1	0	f2	0	0	1	0	f8	0	0	1	0	f2
0	0	1	1	f3	0	0	1	1	f3	0	0	1	1	f9	0	0	1	1	f10
0	1	0	0	f4	0	1	0	0	f8	0	1	0	0	f4	0	1	0	0	f4
0	1	0	1	f5	0	1	0	1	f9	0	1	0	1	f5	0	1	0	1	f12
0	1	1	0	f6	0	1	1	0	f10	0	1	1	0	f12	0	1	1	0	f6
0	1	1	1	f7	0	1	1	1	f11	0	1	1	1	f13	0	1	1	1	f14
1	0	0	0	f8	1	0	0	0	f4	1	0	0	0	f2	1	0	0	0	f1
1	0	0	1	f9	1	0	0	1	f5	1	0	0	1	f3	1	0	0	1	f9
1	0	1	0	f10	1	0	1	0	f6	1	0	1	0	f10	1	0	1	0	f3
1	0	1	1	f11	1	0	1	1	f7	1	0	1	1	f11	1	0	1	1	f11
1	1	0	0	f12	1	1	0	0	f12	1	1	0	0	f6	1	1	0	0	f5
1	1	0	1	f13	1	1	0	1	f13	1	1	0	1	f7	1	1	0	1	f13
1	1	1	0	f14	1	1	1	0	f14	1	1	1	0	f14	1	1	1	0	f7
1	1	1	1	f15	1	1	1	1	f15	1	1	1	1	f15	1	1	1	1	f15
	(a)					(b)					(c)					(d)			

Fig. 4. LUT configurations for the LUT structure shown in Fig. 3(b). (a) LUT configuration for connection to cluster input 1. (b) LUT configuration for connection to cluster input 2. (c) LUT configuration for connection to cluster input 3. (d) LUT configuration for connection to cluster input 4.

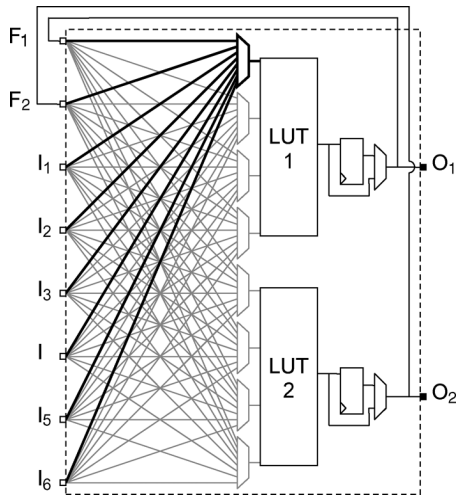


Fig. 5. Logic cluster with 2 four-input LUTs, two feedbacks, six inputs, and a fully connected local routing network.

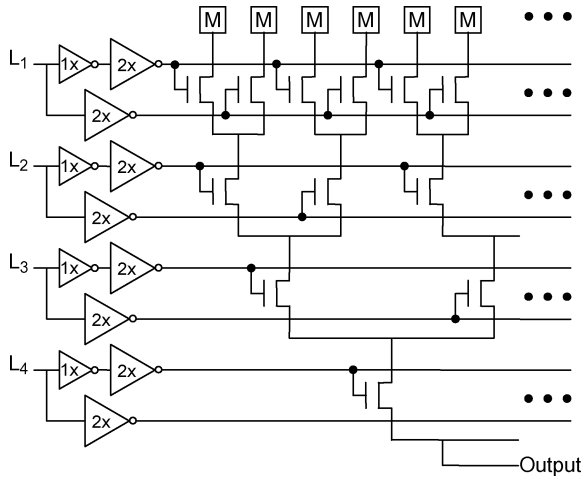


Fig. 7. Structure of a four-input LUT.

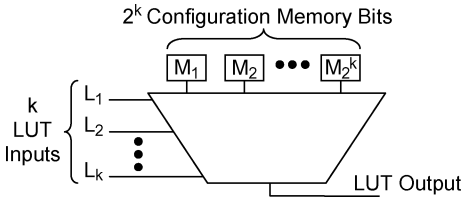


Fig. 6. k-Input LUT.

LUT can be made commutative through LUT reconfiguration. The property is formally stated below where the Boolean function $f(a_1, a_2, \ldots, a_k)$ is implemented in the k-input LUT before the reconfiguration and $f'(a_1, a_2, \ldots, a_k)$ is implemented in the LUT after the reconfiguration. Note that the two variables of concern, i_x and i_y, are highlighted in bold and a formal proof is presented in the Appendix for completeness.

Property 1—Commutative Property: Let $1 \leq x < y \leq k$. Let $f(a_1, a_2, \ldots, a_k)$ be a Boolean function with k inputs. Let $i_1, i_2, \ldots, i_k$ be k independent Boolean variables. For f there exists a function f', such that

$$\begin{aligned} f'(i_1, i_2, \ldots, i_{x-1}, \boldsymbol{i_y}, i_{x+1}, i_{x+2}, \ldots, \\ i_{y-1}, \boldsymbol{i_x}, i_{y+1}, i_{y+2}, \ldots, i_k) \\ = f(i_1, i_2, \ldots, i_{x-1}, \boldsymbol{i_x}, i_{x+1}, i_{x+2}, \ldots, \\ i_{y-1}, \boldsymbol{i_y}, i_{y+1}, i_{y+2}, \ldots, i_k). \end{aligned}$$

The k-input LUT shown in Fig. 6 can also implment any Boolean function with less than k inputs. Implementing such a function also requires all unused LUT inputs to be connected. Three types of signals can be connected to these inputs. They are the inputs from the Boolean function that is currently being implemented, constant 1's or 0's, and an entirely new set of signals, respectively.

Fig. 8 shows an example for each of the three methods. Here, a three-input Boolean function shown in Fig. 8(a) is implemented on a four-input LUT, where L_1 is the unused LUT input. If L_1 is always connected to the same signal as L_2, then the LUT can be configured as shown in Fig. 8(b). The configuration minimizes the amount of configuration information—only 8 out of 16 configuration memory bits are required. Connecting

L1	L2	L3	L4	O
-	0	0	0	f0
-	0	0	1	f1
-	0	1	0	f2
-	0	1	1	f3
-	1	0	0	f4
-	1	0	1	f5
-	1	1	0	f6
-	1	1	1	f7
-	-	-	-	--
-	-	-	-	--
-	-	-	-	--
-	-	-	-	--
-	-	-	-	--
-	-	-	-	--
-	-	-	-	--
-	-	-	-	--

(a) A Three Input Boolean Function

L1	L2	L3	L4	O
0	0	0	0	f0
0	0	0	1	f1
0	0	1	0	f2
0	0	1	1	f3
0	1	0	0	x
0	1	0	1	x
0	1	1	0	x
0	1	1	1	x
1	0	0	0	x
1	0	0	1	x
1	0	1	0	x
1	0	1	1	x
1	1	0	0	f4
1	1	0	1	f5
1	1	1	0	f6
1	1	1	1	f7

(b) Duplicated Input Implementation

L1	L2	L3	L4	O
0	0	0	0	f0
0	0	0	1	f1
0	0	1	0	f2
0	0	1	1	f3
0	1	0	0	f4
0	1	0	1	f5
0	1	1	0	f6
0	1	1	1	f7
1	0	0	0	x
1	0	0	1	x
1	0	1	0	x
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

(c) Constant Input ('0') Implementation

L1	L2	L3	L4	O
0	0	0	0	f0
0	0	0	1	f1
0	0	1	0	f2
0	0	1	1	f3
0	1	0	0	f4
0	1	0	1	f5
0	1	1	0	f6
0	1	1	1	f7
1	0	0	0	f0
1	0	0	1	f1
1	0	1	0	f2
1	0	1	1	f3
1	1	0	0	f4
1	1	0	1	f5
1	1	1	0	f6
1	1	1	1	f7

(d) New Input Implementation

Fig. 8. Implementing a three-input Boolean function in a four-input LUT. (a) Three-input Boolean function. (b) Duplicated input implementation. (c) Constant input ("0") implementation. (d) New input implementation.

L_1 to a constant value also minimizes the amount of configuration information. For example, if L_1 is connected to a constant value of "0," the LUT can be configured as shown in Fig. 8(c). Again, only 8 out of 16 configuration memory bits are required. Connecting constant Boolean values to LUT inputs, however, requires additional circuitry to connect LUT inputs to logical 1's or 0's.

Alternatively, one can connect any arbitrary signals to the unused inputs and suppress these arbitrary inputs through LUT reconfiguration. Suppressing arbitrary inputs requires the duplication of the configuration information [10]. In Fig. 8, e.g., if L_1 is connected to a nonconstant signal other than the signals connected to L_2, L_3, and L_4, the LUT can be reconfigured as shown in Fig. 8(d). Here, all 16 bits of the configuration memory are utilized by duplicating the three-input truth table. Note that the new input method is more flexible than the previous two methods since the same LUT configuration can also be used in the duplicated input and the constant input cases. Furthermore, the method places no restiction on which type of signals must be connected to the unused inputs.

For a general k-input LUT, Property 2 formally defines the equivalency between the constant input method and the duplicated input method. Property 3 formally defines the equivalency between the new input method and the constant input method. Both properties assume the LUT implements the Boolean function $f(a_1, a_2, \ldots, a_k)$ before the reconfiguration and the Boolean function $f'(a_1, a_2, \ldots, a_k)$ after the reconfiguration. Again variables of concern, i_x and i_y, are highlighted in bold and formal proofs of the properties are presented in the Appendix for completeness.

Property 2—Duplicate-Constant Input Equivalence: Let $1 \leq x < y \leq k$. Let $f(a_1, a_2, \ldots, a_k)$ be a Boolean function with k inputs. Let $i_1, i_2, \ldots, i_k$ be k Boolean variables. If $\boldsymbol{i_x} = \boldsymbol{i_y}$, then there exists a function f' such that

$$f'(i_1, i_2, \ldots, i_{x-1}, \boldsymbol{i_x}, i_{x+1}, i_{x+2}, \ldots, i_{y-1}, \mathbf{0}, i_{y+1}, i_{y+2}, \ldots, i_k) = f(i_1, i_2, \ldots, i_{x-1}, \boldsymbol{i_x}, i_{x+1}, i_{x+2}, \ldots, i_{y-1}, \boldsymbol{i_y}, i_{y+1}, i_{y+2}, \ldots, i_k).$$

Property 3—Constant-New Input Equivalence: Let $1 \leq x \leq k$. Let $f(a_1, a_2, \ldots, a_k)$ be a Boolean function with k inputs.

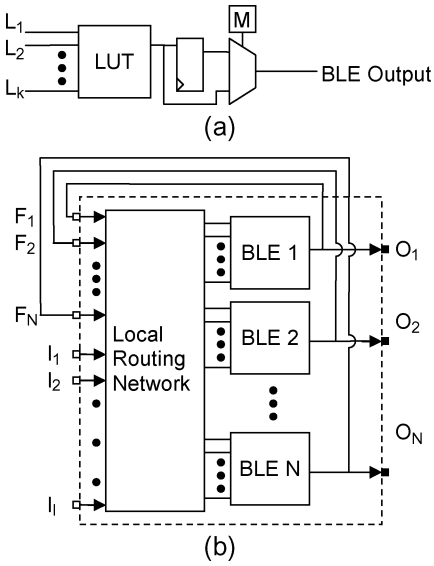


Fig. 9. BLEs and logic cluster. (a) BLE. (b) Logic cluster.

Let $i_1, i_2, \ldots, i_k$ be k independent Boolean variables. For f there exists a function f' such that

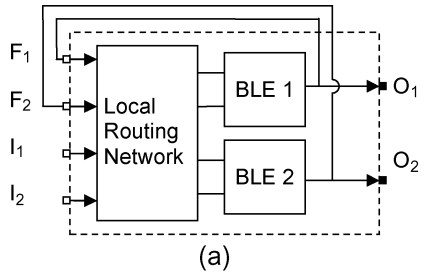
$$f'(i_1, i_2, \ldots, i_{x-1}, \boldsymbol{i_x}, i_{x+1}, i_{x+2}, \ldots, i_k) = f(i_1, i_2, \ldots, i_{x-1}, \mathbf{0}, i_{x+1}, i_{x+2}, \ldots, i_k).$$

IV. Minimum Set of Logic Cluster Input and Feedback Combinations for Maintaining Logically Equivalent I/Os

This section demonstrates LUT reconfiguration that can be used to increase the flexibility of local routing networks while maintaining logic equivalency among logic cluster I/Os. In particular, the logic cluster structure shown in Fig. 9 is used in this paper. This cluster is a generalized version of the logic cluster shown in Fig. 5. It contains N LUTs and N registers. As shown in Fig. 9(a), each LUT is paired with a register to form a basic logic element (BLE). Within the BLE, the output of the LUT feeds the input of the register. The output of the register and the output of the LUT are connected to the BLE output through a 2:1 multiplexer. As shown in Fig. 9(b), N BLEs are grouped into a logic cluster. Each logic cluster contains I inputs. Within each cluster, the local routing network is used to connect the inputs of the LUTs to the inputs of the logic cluster as well as the outputs (feedbacks) of the BLEs.

Given a k-input function, $f(a_1, a_2, \ldots, a_k)$, define S_f as the maximum set of functions that one can generate by exhausitviely connecting f to all combinations of logic cluster inputs and feedbacks. By definition, S_f contains $(I + N)^k$ functions. The following theorem shows the equivalency between logically equivalent I/Os and S_f.

Theorem 1: The I/Os of a logic cluster are logically equivalent if and only if the cluster can implement a k-input function, $f(a_1, a_2, \ldots, a_k)$, at any logic cluster output and, for each implementation of f, the local routing network is able to generate all functions in S_f for f.




```
f(F1,F1), f(F1,F2), f(F1,I1), f(F1,I2)
f(F2,F1), f(F2,F2), f(F2,I1), f(F2,I2)
f(I1,F1), f(I1,F2), f(I1,I1), f(I1,I2)
f(I2,F1), f(I2,F2), f(I2,I1), f(I2,I2)
```

(b)

Fig. 10. Logic cluster and S_f. (a) Logic cluster. (b) S_f for functions implemented at O_1 and O_2.

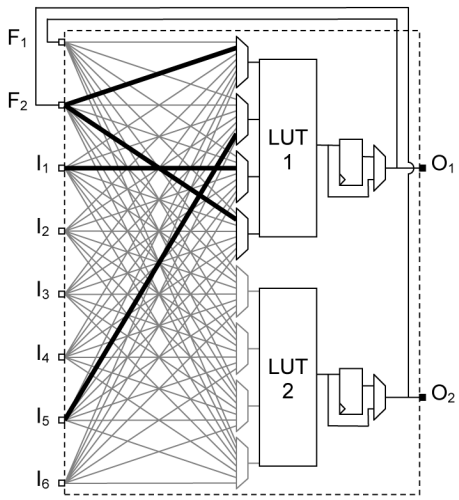


Fig. 11. Connections for $\langle F_2, I_5, I_1, F_2 \rangle$.

Proof: If the logic cluster has a set of logically equivalent inputs, each input of f can enter the logic cluster through any of the logic cluster inputs. If the logic cluster has a set of logically equivalent outputs, one can implement f at any logic cluster output. A feedback signal must also be able to reach f from any of the logic cluster outputs. Consequently, the local routing network must be able to generate all functions in S_f for f at each logic cluster output.

Conversely, if the local routing network is not flexible enough to generate all functions in S_f at a particular logic cluster output, one must avoid signal assignments that can lead to the unimplementable functions. If these un-impelmentable functions involve logic cluster inputs, then these inputs are no longer logically equivalent to the remaining inputs. Similarly, if the unimplementable functions involve logic cluster feedbacks, then the corresponding logic cluster outputs are no longer logically equivalent to the remaining outputs. ■

As an example, Fig. 10(a) shows a logic cluster with 2 two-input LUTs, two logic cluster inputs, and two feedbacks. To ensure that the logic cluster contains a set of logically equivalent I/Os, the local routing network must be able to generate all 16 functions shown in Fig. 10(b) for each logic cluster output.

Logic equivalency among the logic cluster inputs and outputs can be achieved through the use of a fully connected local routing network that connects each LUT input to all logic cluster inputs and feedbacks. It can be shown that, without LUT reconfiguration, this fully connected network is the minimum required to achieve logic equivalency. In particular, if an LUT input is only connected to a subset of logic cluster inputs and feedbacks, a signal assigned to the LUT input can only enter the cluster through the connected inputs/feedbacks—these connected inputs/feedbacks are no longer logically equivalent to the unconnected ones. The fully connected local routing network can be designed as $N \times k \quad (I+N)$:1 multiplexers. Each LUT input is connected to the output of one of the multiplexers. The multiplexer inputs are connected to the I inputs of the logic cluster and its N feedbacks. Using this structure, one can generate $(I+N)^k$ distinct combinations of inputs/feedbacks for each LUT.

More formally, let $n = I + N$. In general, there are n^k different ways of substituting n variables into a k-input Boolean function. If these variables are the outputs of other Boolean functions, the substitution process can generate at a maximum n^k new functions—one function for each of the n^k combinations of variables. The following lemmas show that one can reduce the number of input combinations required to generate the n^k functions from n^k to $\binom{n}{k}$. Theorem 2 shows that these $\binom{n}{k}$ input combinations are the minimum set required to generate the n^k functions.

Lemma 1: Let A be a k-input LUT implementing a Boolean function $f(a_1, a_2, \ldots, a_k)$. Let $i_1, i_2, \ldots, i_n$ be the output signals from n LUTs. Let S_f be the set of all functions that can be created by exhaustively connecting each input of A to all signals in the set $\{i_1, i_2, \ldots, i_n\}$. Using the commutative property and the duplicate-constant input equivalence property, one can reduce the n^k input combinations required to generat S_f to $\sum_{j=1}^{k} \binom{n}{j}$ input combinations.

Proof: Create an exhaustive list L of all the subsets of $\{i_1, i_2, \ldots, i_n\}$ for the subset sizes of $1, 2, \ldots, k$. For each subset in L, sort the Boolean variables in the subset according to their orders in $\{i_1, i_2, \ldots, i_n\}$. Create a set S_i' of k-bit wide bit vectors such that each bit vector in S_i' is the concatenation of the variables in a sorted subset in L and $k - j$ 0's, where j is the size of the subset.

Let S_i'' be a set of bit vectors representing all possible combinations of k variables in $\{i_1, i_2, \ldots, i_n\}$. For any bit vector v in S_i'', containing one or more i_x, where $1 \leq x \leq n$, let v' be a copy of v. Substitute all but one i_x in v' by 0's. The duplicate-constant input equivalence property says that there exists a function f' such that $f'(v') = f(v)$. Repeat the substitution process until a bit vector v'' containing no repeated variables is obtained. Let f'' be the function such that $f''(v'') = f(v)$.

Create v''' by sorting the variables in v'' according to their orders in $\{i_1, i_2, \ldots, i_n\}$ and moving all constant 0's in v'' to the end of the bit vector. The commutative property says that there exists a function f''' such that $f'''(v''') = f''(v'') = f(v)$.

v''' is in S_i'. Consequently, S_i' contains all the input combinations that are required to generate all functions in S_f. By definition, S_i' contains $\sum_{j=1}^{k} \binom{n}{j}$ distinct bit vectors. ■

L1	L2	L3	L4	O
0	0	0	0	f0
0	0	0	1	f1
0	0	1	0	f2
0	0	1	1	f3
0	1	0	0	f4
0	1	0	1	f5
0	1	1	0	f6
0	1	1	1	f7
1	0	0	0	f8
1	0	0	1	f9
1	0	1	0	f10
1	0	1	1	f11
1	1	0	0	f12
1	1	0	1	f13
1	1	1	0	f14
1	1	1	1	f15

(a)

L1	L2	L3	L4	O
0	0	0	0	f0
0	0	0	1	f9
0	0	1	0	f2
0	0	1	1	f11
0	1	0	0	f4
0	1	0	1	f13
0	1	1	0	f6
0	1	1	1	f15
1	0	0	0	x
1	0	0	1	x
1	0	1	0	x
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

(b)

L1	L2	L3	L4	O
0	0	0	0	f0
0	0	0	1	x
0	0	1	0	f4
0	0	1	1	x
0	1	0	0	f2
0	1	0	1	x
0	1	1	0	f6
0	1	1	1	x
1	0	0	0	f9
1	0	0	1	x
1	0	1	0	f13
1	0	1	1	x
1	1	0	0	f11
1	1	0	1	x
1	1	1	0	f15
1	1	1	1	x

(c)

L1	L2	L3	L4	O
0	0	0	0	f0
0	0	0	1	f0
0	0	1	0	f4
0	0	1	1	f4
0	1	0	0	f2
0	1	0	1	f2
0	1	1	0	f6
0	1	1	1	f6
1	0	0	0	f9
1	0	0	1	f9
1	0	1	0	f13
1	0	1	1	f13
1	1	0	0	f11
1	1	0	1	f11
1	1	1	0	f15
1	1	1	1	f15

(d)

Fig. 12. LUT reconfiguration and LUT input rearrangement. (a) $\langle F_2, I_5, I_1, F_2\rangle$ configuration. (b) $\langle 0, I_5, I_1, F_2\rangle$ configuration. (c) $\langle F_2, I_1, I_5, 0\rangle$ configuration. (d) $\langle F_2, I_1, I_5, I_6\rangle$ configuration.

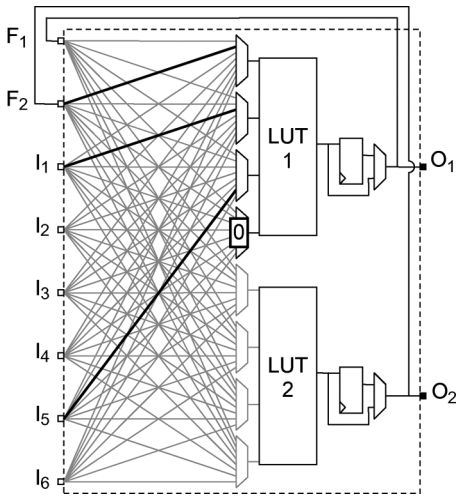


Fig. 13. Connections for $\langle F_2, I_1, I_5, 0\rangle$.

Lemma 1 shows that if LUT inputs are allowed to connect to constant Boolean values (0's in this case), for each k-bit wide input bit vector v, one can substitute all the duplicate inputs in v by 0's. The LUT configuration can then be transformed using the duplicate-constant input equivalence property—resulting in a circuit that implements the same function as the the original circuit. Furthermore, based on the commutative property, the bits in v can be rearranged in any order. For each rearrangement, a new LUT configuration can be defined to create a new circuit that implements the same function as the original circuit.

As an example, consider the logic cluster shown in Fig. 5. Assuming LUT 1 is connected to bit vector $\langle F_2, I_5, I_1, F_2\rangle$, as shown in Fig. 11, and is configured to the configuration shown in Fig. 12(a). The same function can be implemented by transforming the configuration to Fig. 12(b) and substitute the first F_2 with "0." The inputs can then be rearranged into bit vector $\langle F_2, I_1, I_5, 0\rangle$ and the LUT be reconfigured to the configuration shown in Fig. 12(c). Again the new circuit, as shown in Fig. 13, implements exactly the same function as the original circuit. Note that $\langle F_2, I_1, I_5, 0\rangle$ is a bit vector in set S_i', as defined in Lemma 1. Since Fig. 5 contains a fully connected local routing network, the same observation can be made for LUT 2.

Although Lemma 1 reduces the number of input combinations required to generate S_f, it introduces 0's to LUT inputs. Connecting LUTs to 0's requires additional hardware. Lemma 2 shows that the constant-new input equivalence property allows one to create a new set of input combinations by removing 0's from S_i' and further reduce the number of input combinations from $\sum_{j=1}^{k}\binom{n}{j}$ to $\binom{n}{k}$.

Lemma 2: Let A be a k-input LUT implementing a Boolean function $f(a_1, a_2, \ldots, a_k)$. Let $i_1, i_2, \ldots, i_n$ be the output signals from n LUTs. Let S_f be the set of all functions that can be created by exhaustively connecting each input of A to all signals in $\{i_1, i_2, \ldots, i_n\}$. Using the constant-new input equivalence property, one can reduce the $\sum_{j=1}^{k}\binom{n}{j}$ distinct combinations required to generate all functions in S_f to $\binom{n}{k}$.

Proof: Create an exhaustive list L of all the subsets of $\{i_1, i_2, \ldots, i_n\}$, for the subset size of k. For each subset in L, sort the variables in the subset according to their orders in $\{i_1, i_2, \ldots, i_n\}$. Create a set S_i of k-bit wide bit vectors such that each bit vector in S_i corresponds to a sorted subset in L.

Use the same definition of S_i' as Lemma 2. For any bit vector v' in S_i' that contains one or more 0's, one can find a vector v in S_i such that the variables contained in v is a superset of the variables contained in v'. By the constant-new input equivalence property, there exists a function f' such that $f'(v) = f(v')$. Therefore, S_i contains all the input combinations that are required to generate all the functions in S_f. By definition, S_i contains $\binom{n}{k}$ distinct bit vectors. ■

Fig. 12(d) shows the LUT configuration after the constant "0" is substituted by I_6 in our example. The new local routing network connection is shown in Fig. 14. Note that $\langle F_2, I_1, I_5, I_6\rangle$ is a bit vector in S_i, as defined in Lemma 2.

Theorem 2: S_i is the minimum set of input combinations that generates the set S_f for all k-input Boolean functions.

Proof: Let $f(a_1, a_2, \ldots, a_k)$ be a k-input Boolean function with the following properties: for each input of f, a_j, where $1 \le j \le k$, there exists at least one sequence of constant 0's and 1's—$b_1, b_2, \ldots, b_{j-1}, b_{j+1}, b_{j+2}, \ldots, b_k$—such that

$$f(b_1, b_2, \ldots, b_{j-1}, 0, b_{j+1}, b_{j+2}, \ldots, b_k) \neq f(b_1, b_2, \ldots, b_{j-1}, 1, b_{j+1}, b_{j+2}, \ldots, b_k).$$

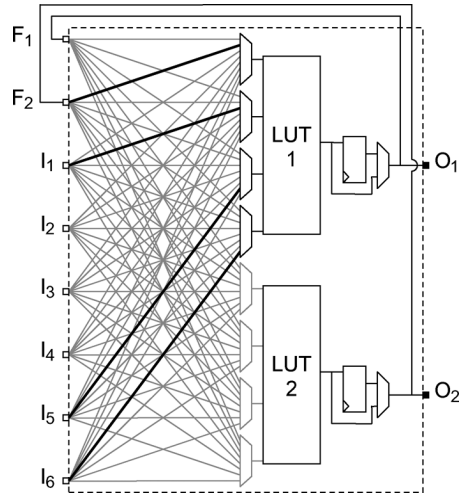


Fig. 14. Connections for $\langle F_2, I_1, I_5, I_6 \rangle$.

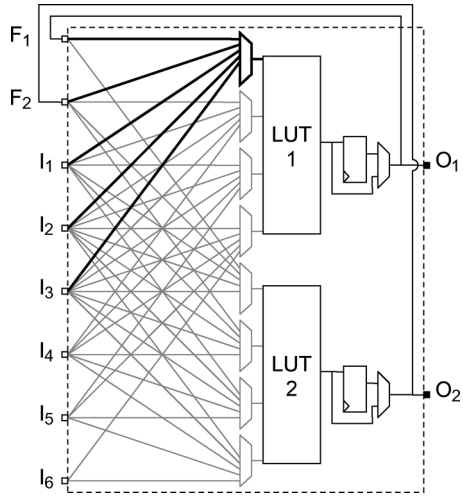


Fig. 15. Sparse local routing network.

Let v be a bit vector in S_i as defined in Lemma 2. Let v' be another bit vector in S_i such that $v' \neq v$. For each v' in S_i, there exists a signal i_x, where $1 \leq x \leq n$, in v such that i_x does not belong to v'. Since, by definition, the output of $f(v)$ varies as i_x is varied, for any k-input Boolean function f', $f'(v') \neq f(v)$. Therefore, v cannot be removed from S_i without reducing the set of functions that can be generated for f. By Lemmas 1 and 2, any set of input combinations can be reduced to S_i or a subset of S_i without reducing the number of functions that the set can generate. Consequently, S_i is the minimum set of input combinations that can generate all functions in S_f for f.

Since S_i can be used to generate S_f for any k-input Boolean functions, S_i is the minimum set of input combinations that can generate S_f for all k-input Boolean functions. ■

V. Circuit Design for Generating the Minimum Set of Logic Cluster Input and Feedback Combinations

Having discussed the minimum set of input combinations S_i that is required to generate all functions in S_f, we examine the effect of the minimum set on the design of the multiplexer-based local routing networks. First, Lemma 3 examines two unique properties of S_i.

Lemma 3: Let A be a k-input LUT implementing a Boolean function $f(a_1, a_2, \ldots, a_k)$. Let $i_1, i_2, \ldots, i_n$ be the output signals from n LUTs. Let v be a k-bit wide bit vector containing a subset of k signals from $\{i_1, i_2, \ldots, i_n\}$. If v is in S_i and i_x is the jth element of v, then x must be smaller than or equal to $n - k + j$ and greater than or equal to j.

Proof: Let i_y be the kth (the last) element in v. Since elements in v are sorted according to their orders in $\{i_1, i_2, \ldots, i_n\}$, y must be greater than or equal to $x + k - j$. If $x > n - k + j$, then $y > (n - k + j) + k - j = n$. However, by definition, y must be smaller than or equal to n. Therefore, x must be smaller than or equal to $n - k + j$.

Similarly, let i_z be the first element in v. Since elements in v are sorted according to their orders in $\{i_1, i_2, \ldots, i_n\}$. z must be smaller than or equal to $x - j + 1$. If $x < j$, then $z < j - j + 1 = 1$. However, by definition, z must be greater than or equal to 1. Therefore, x must be greater than or equal to j. ■

Lemma 3 shows that a local routing network can be used to connect the jth input of a k-input LUT to all signals in the set $\{i_j, i_{j+1}, \ldots, i_{n-k+j}\}$ through an $(n - k + 1)$:1 multiplexer. Through LUT reconfiguration and function transformations, the LUT can be used to generate all functions in S_f, as defined in Lemma 1. Note that to generate all functions in S_f without reconfiguration, each input of the k-input LUT must be connected to all signals in $\{i_1, i_2, \ldots, i_n\}$ through an n:1 multiplexer.

For example, for the logic cluster shown in Fig. 3, there are four logic cluster inputs, I_1, I_2, I_3, and I_4, and no feedbacks. Lemma 3 shows that LUT input L_1 should be connected to all signals in the set $\{I_1\}$ (for $j = 1$, $n = 4$, and $k = 4$), L_2 should be connected to all signals in the set $\{I_2\}$ (for $j = 2$, $n = 4$, and $k = 4$), L_3 should be connected to all signals in the set $\{I_3\}$ (for $j = 3$, $n = 4$, and $k = 4$), and L_4 should be connected to all signals in the set $\{I_4\}$ (for $j = 4$, $n = 4$, and $k = 4$). With reconfiguration, the local routing network is able to generate all functions in set S_f.

Similarly, for the two LUT logic cluster shown in Fig. 5, there are two feedbacks $F_1(i_1)$ and $F_2(i_2)$, and six logic cluster inputs $I_1(i_3)$, $I_2(i_4)$, $I_3(i_5)$, $I_4(i_6)$, $I_5(i_7)$, and $I_6(i_8)$. Lemma 3 shows that for each LUT, LUT input L_1 should be connected to all signals in the set $\{F_1, F_2, I_1, I_2, I_3\}$ (for $j = 1$, $n = 8$, and $k = 4$), L_2 should be connected to all signals in the set $\{F_2, I_1, I_2, I_3, I_4\}$ (for $j = 2$, $n = 8$, and $k = 4$), L_3 should be connected to all signals in the set $\{I_1, I_2, I_3, I_4, I_5\}$ (for $j = 3$, $n = 8$, and $k = 4$), and L_4 should be connected to all signals in the set $\{I_2, I_3, I_4, I_5, I_6\}$ (for $j = 4$, $n = 8$, and $k = 4$). Again, with reconfiguration, this sparser local routing network (as shown in Fig. 15) can generate all functions in S_f.

In Fig. 15, the multiplexer size is reduced from 8:1 to 5:1. The fanout of F_1 and I_6 is reduced from 8 to 2. The fanout of F_2 and I_5 is reduced from 8 to 4; the fanout of I_1 and I_4 is reduced from 8 to 6; and the fanout of I_2 and I_3 remains unchanged at 8. Note that, as shown in Fig. 16, the fanouts of all logic cluster inputs and feedbacks can be reduced to 5 by rearranging the order of the logic cluster inputs/feedbacks to $\{I_2, I_1, F_2, F_1, I_6, I_5, I_4, I_3\}$ when the inputs and feedbacks are connected to LUT 2.

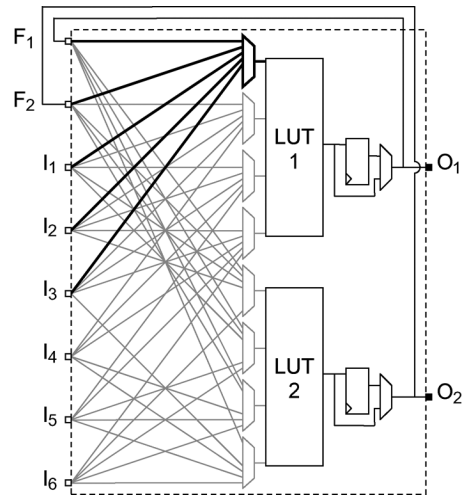


Fig. 16. Sparse local routing network with balanced fanouts.

A1	A2	A3	O	L1	L2	L3	L4	O	L1	L2	L3	L4	O
0	0	0	f0	0	0	0	0	f0	0	0	0	0	f0
0	0	1	f1	0	0	0	1	f2	0	0	0	1	f2
0	1	0	f2	0	0	1	0	x	0	0	1	0	f4
0	1	1	f3	0	0	1	1	x	0	0	1	1	f6
1	0	0	f4	0	1	0	0	x	0	1	0	0	f0
1	0	1	f5	0	1	0	1	x	0	1	0	1	f2
1	1	0	f6	0	1	1	0	f1	0	1	1	0	f4
1	1	1	f7	0	1	1	1	f3	0	1	1	1	f6
-	-	-	--	1	0	0	0	f4	1	0	0	0	f1
-	-	-	--	1	0	0	1	f6	1	0	0	1	f3
-	-	-	--	1	0	1	0	x	1	0	1	0	f5
-	-	-	--	1	0	1	1	x	1	0	1	1	f7
-	-	-	--	1	1	0	0	x	1	1	0	0	f1
-	-	-	--	1	1	0	1	x	1	1	0	1	f3
-	-	-	--	1	1	1	0	f5	1	1	1	0	f5
-	-	-	--	1	1	1	1	f7	1	1	1	1	f7
(a)				(b)					(c)				

Fig. 17. LUT configurations for implementations 1 and 2. (a) A three-input Boolean function. (b) LUT configuration for Imp. 1 (L1 = A1, L2 = A3, L3 = A3, L4 = A2). (c) LUT configuration for Imp. 2 (L1 = A3, L2 = I1, L3 = A1, L4 = A2).

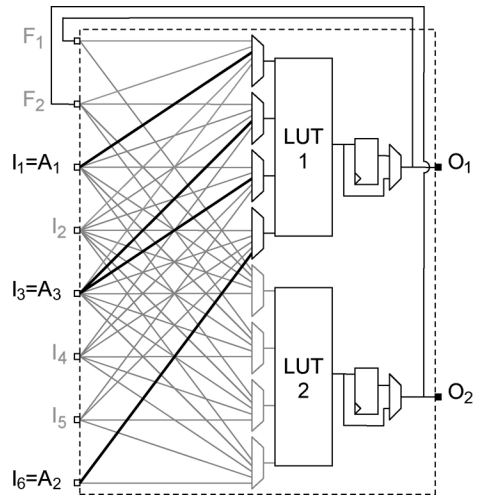


Fig. 18. Implementation 1.

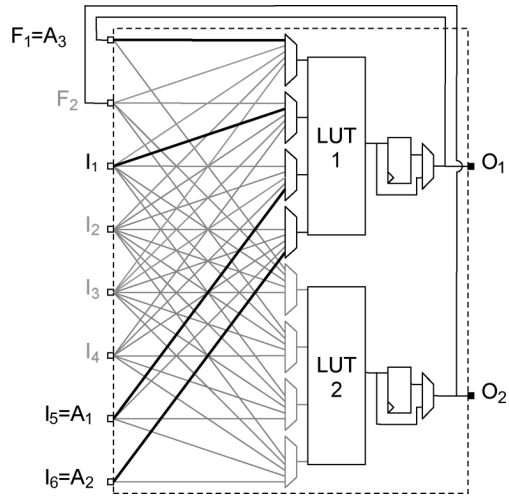


Fig. 19. Implementation 2.

Both logic cluster designs retain logic equivalency among logic cluster inputs and outputs. As an example, consider implementing the three-input Boolean function shown in Fig. 17(a) in the logic cluster shown in Fig. 15. There are 336 unique ways that the three inputs can enter the logic cluster. Figs. 18 and 19 show two of the possibilities. In Fig. 18, A_1, A_2, and A_3 are assigned to cluster inputs I_1, I_6, and I_3, respectively. A_3 is also duplicated to provide the fourth LUT input. The corresponding LUT configuration is shown in Fig. 17(b).

Alternatively, a router can assign A_1, A_2, and A_3 to I_5, I_6, and F_1, respectively. Due to the sparse local routing network, none of the three inputs can be expanded into the fourth LUT input. Instead, an arbitrary cluster input, I_1, is used as the fourth input. Note that in a directional single-drive architecture [11], each track is driven by its own buffer (as shown in Fig. 20). Consequently, I_1 can be connected to any of the routing tracks since the LUT is configured to provide the same output for both $I_1 = 0$ and $I_1 = 1$, as shown in Fig. 17(c).

Theorem 3: At a minimum, one requires $k \quad (n - k + 1)$:1 multiplexers to generate all bit vectors in S_i and all functions in S_f.

Proof: Let the $k \quad (n - k + 1)$:1 multiplexer design be design D. Assume there exists a design, D', with at least one x:1 multiplexer, where $x < n - k + 1$, and D' can generate all the bit vectors in S_i.

Let the set S_x be the set that contains all the inputs to one of the x:1 multiplexers A in D'. For each of the $(n - k + 1)$:1 multiplexers B in D, let S_y be the set that contains all the inputs to B. Remove all signals in $S_x \cap S_y$ from the inputs of B. Since $x < n - k + 1$, there will be at least $n - k + 1 - x$ inputs remaining for B. Let the new design be D''.

For the k multiplexers in D, let M be a $k \times n$ matrix (with k rows and n columns). If multiplexer i is connected to logic cluster input/feedback j then let $M(i, j) = 1$. Otherwise, let $M(i, j) = 0$. Since there are less than $n - k + 1$ variables in S_x, creating the design D'' corresponds to removing 1 to $n-k$ columns from M. The resulting matrix M'' has k to $n-1$ columns. Since M'' is created by removing less than $n - k + 1$ columns from M and from Lemma 3, for each row j in M, columns j to $n-k+j$ are equal to 1, $M''(i, i) = 1$ for all valid values of i.

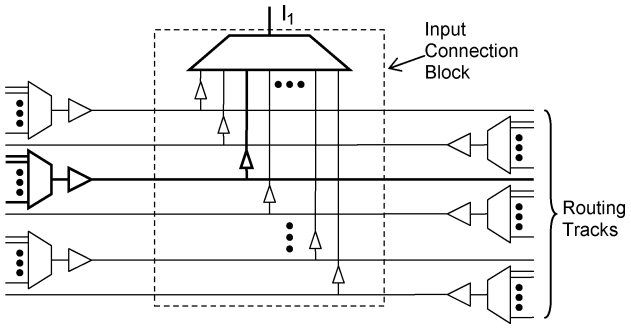


Fig. 20. Logic cluster to routing track connections.

Consequently, in D'', one can generate a bit vector v by assigning the input corresponding to the first column of M'' to the first multiplexer, the input corresponding to the second column of M'' to the second multiplexer, etc., and the input corresponding to the kth column of M'' to the kth multiplexer. v belongs to S_i. Since v does not contain any inputs of A, v cannot be generated by design D'. This contadicts the assumption that D' can generate all the bit vectors in S_i. Therefore, x cannot be smaller than $n-k+1$ for generating all bit vectors in S_i.

Similarly, since v does not contain any inputs of A and v contains k distinct logic cluster inputs/feedbacks, D' cannot generate any bit vector that contains all inputs/feedbacks in v. Consequently, D' cannot generate all functions in S_f. Therefore, x cannot be smaller than $n-k+1$ for generating all functions in S_f. ■

VI. Area and Fanout Results

As shown in Section V, LUT reconfiguration can be effectively used to reduce the size of the multiplexers that are required to preserve logic equivalency. In particular, as shown in Fig. 9, to implement a traditional local routing network, a logic cluster with N BLEs and I inputs would require $N \times k$ $(I+N)$:1 multiplexers. With LUT reconfiguration, on the other hand, only $N \times k$ $(I+N-k+1)$:1 multiplexers are required. While this reduction in multiplexer size can result in significant area savings, the exact amount of area reduction varies with the detailed implementations of the multiplexers.

For example, Fig. 21(a) shows a pass transistor-based multiplexer design, which is optimized for minimum configuration memory. In this design, reducing the number of multiplexer inputs from n to $n-k+1$ reduces the total number of pass transistors from $2n-2$ to $2(n-k)$. This input reduction also reduces the total number of configuration memory bits required to control the multiplexer from $\lceil \log_2 n \rceil$ to $\lceil \log_2 (n-k+1) \rceil$.

On the other hand, for a minimum-level implementation such as the one shown in Fig. 21(b), the same reduction in multiplexer inputs reduces the number of configuration memory bits and the number of pass transistors both from n to $n-k+1$.

In this study, the area evaluation and circuit design methodology of [2] is used to more accurately evaluate the effect of multiplexer input reduction on logic cluster area. This methodology has been widely used in many previous FPGA architectural studies [1]–[3], [5], [9], [11]–[20]. The implementation

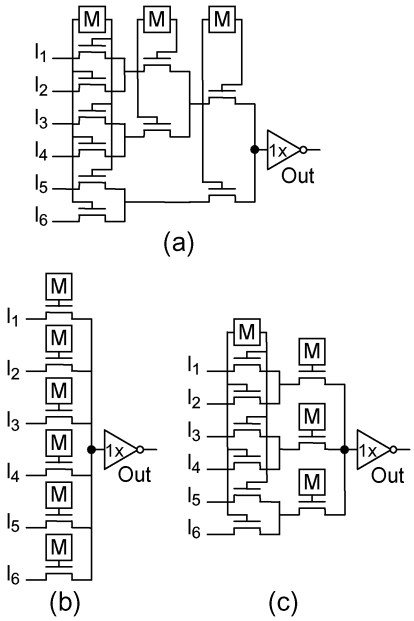


Fig. 21. Multiplexer implementations $(n = 6)$. (a) Minimum configuration memory. (b) Minimum-level multiplexer. (c) Two-level multiplexer.

area of a logic cluster is estimated based on active area. In particular, the active area A consumed by a logic cluster is defined as

$$A = \sum_{\text{All Trans.}} \left(0.5 + \frac{\text{drive strength of the current trans.}}{2 \times \text{drive strength of min. width trans.}}\right). \tag{1}$$

The detailed design of LUTs used in this evaluation is shown in Fig. 7. The design assumes that the 2^k configuration memory bits are implemented using static RAM (SRAM) cells and are directly connected to the data inputs of the 2^k:1 multiplexer. It is assumed that each SRAM cell consumes the equivalent area of six minimum width transistors. The multiplexer employed by the LUT is a multilevel design that minimizes the number of select signals. These select signals in turn are connected to the k LUT inputs through a set of buffers. It is assumed that all pass transistors used in the design are of minimum width.

The BLEs are designed as Fig. 9. As the LUT design, it is assumed that minimum width pass transistors are used to implement the 2:1 multiplexer. An SRAM bit is used to control the configuration of the multiplexer. It is also assumed that the register is of D-type, which consumes the equivalent area of 19 minimum width transistors.

Based on these paramneters, Table I shows the equivalent minimum width transistor area that is required to implement a local routing network for logic clusters containing N BLEs, where N varies from 1 to 20 (the same percentage reduction data from column 4, 7, and 10 are presented again graphically in Fig. 22). It is assumed that each BLE contains a four-input LUT and the number of inputs I that each logic cluster contains is equal to [1]

$$I = \left\lceil \frac{k}{2}(N+1) \right\rceil. \tag{2}$$

TABLE I
LOCAL ROUTING NETWORK AREA REDUCTION ($k = 4$)

N	Min. Mem.			Min. Level			Two Level		
	Trad.	New	% Reduc.	Trad.	New	% Reduc.	Trad.	New	% Reduc.
1	113.4	41.4	63.5%	149.4	65.4	56.2%	133.4	69.4	48.0%
2	274.8	226.8	17.5%	466.8	298.8	36.0%	354.8	266.8	24.8%
3	556.2	412.2	25.9%	952.2	700.2	26.5%	724.2	532.2	26.5%
4	837.6	741.6	11.5%	1605.6	1269.6	20.9%	1141.6	965.6	15.4%
5	1287.0	1047.0	18.7%	2427.0	2007.0	17.3%	1747.0	1427.0	18.3%
6	1688.4	1544.4	8.5%	3416.4	2912.4	14.8%	2360.4	2096.4	11.2%
7	2137.8	1969.8	7.9%	4573.8	3985.8	12.9%	3201.8	2753.8	14.0%
8	2635.2	2443.2	7.3%	5899.2	5227.2	11.4%	4011.2	3659.2	8.8%
9	3180.6	2964.6	6.8%	7392.6	6636.6	10.2%	5088.6	4512.6	11.3%
10	3774.0	3534.0	6.4%	9054.0	8214.0	9.3%	6094.0	5654.0	7.2%
20	12828.0	12348.0	3.7%	34908.0	33228.0	4.8%	22988.0	22108.0	3.8%

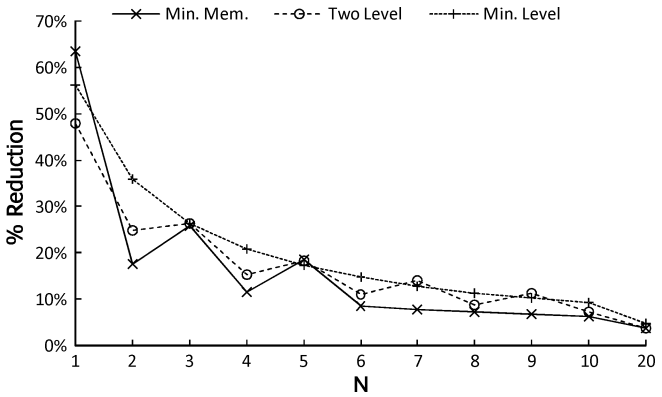


Fig. 22. Local routing network area reduction ($k = 4$).

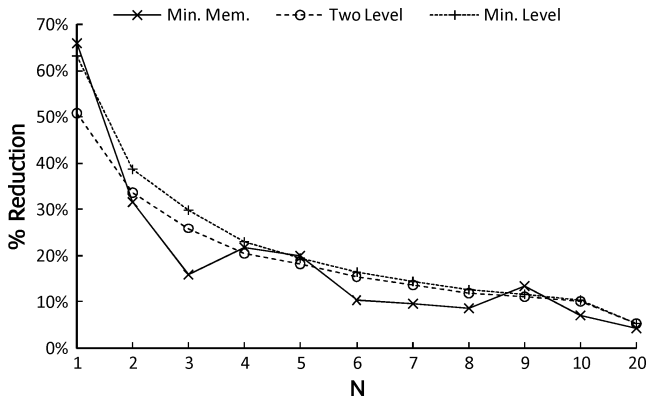


Fig. 23. Local routing network area reduction ($k = 5$).

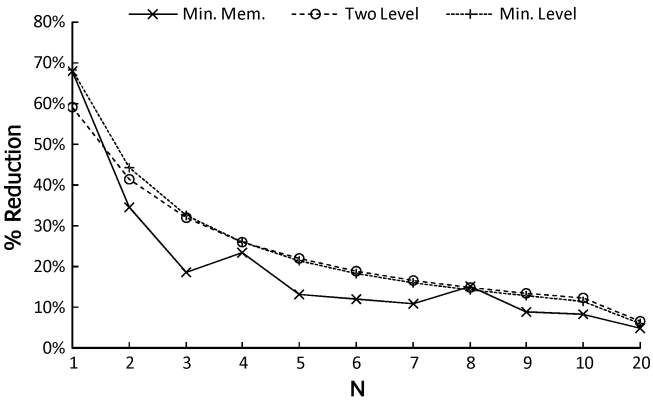


Fig. 24. Local routing network area reduction ($k = 6$).

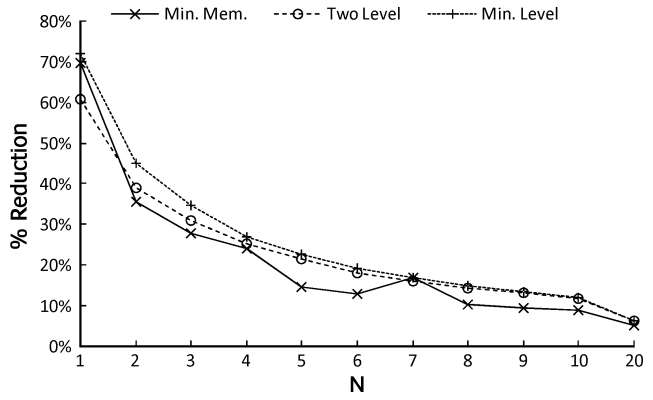


Fig. 25. Local routing network area reduction ($k = 7$).

In the table, Column 2 shows the equivalent minimum width transistor area that is required to implement a traditional local routing network based on the minimum configuration memory multiplexer shown in Fig. 21(a), where all pass transistors are of minimum width and each SRAM cell consumes the equivalent area of six minimum width transistors. Column 3 shows the area required to implement the new local routing network employing the same multiplexer design. Finally, column 4 shows the area reduction of the new implementation as compared to the traditional implementation.

Columns 5–7 summarize the same information for local routing networks employing the minimum-level multiplexer design shown in Fig. 21(b), while column 8, 9, and 10 summarize the same information for the two-level multiplexer design shown in Fig. 21(c). Again, it is assumed that each pass transistor is of minimum width and each SRAM cell consumes the equivalent area of six minimum width transistors.

The table shows that while the area required to implement the local routing networks varies significantly from multiplexer design to multiplexer design, the overall area reduction as a function of N behaves similarly across all three multiplexer designs. In particular, all three designs achieve significant area savings at small values of N. As N increases, the proportion of area savings that can be achieved through LUT reconfiguration decreases. In particular, for $N = 1$ the new local routing network implementation is around 48%–64% smaller than the traditional implementation. For $N = 20$, the area reduction is reduced to 3.7%–4.8%. Similar trend is observed for other values of k as shown in Figs. 23–25 for the LUT size of 5–7, respectively.

Figs. 26–29 show the percentage of logic cluster area reduction as a function of N for the LUT size of 4–7, respectively. As shown, the area reduction per logic cluster follows the same trend as the area reduction for local routing networks.

The reductions range from 12% to 25% for small values of N to 2.9%–5.2% for large values of N. In particular, the minimum-level and the two-level implementations see a larger percentage reduction since the local routing networks employing these multiplexer designs consume more active area than the local routing networks that employ the minimum configuration

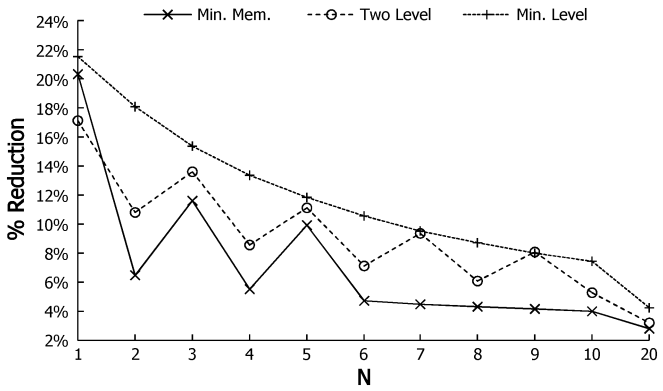


Fig. 26. Logic cluster area reduction ($k = 4$).

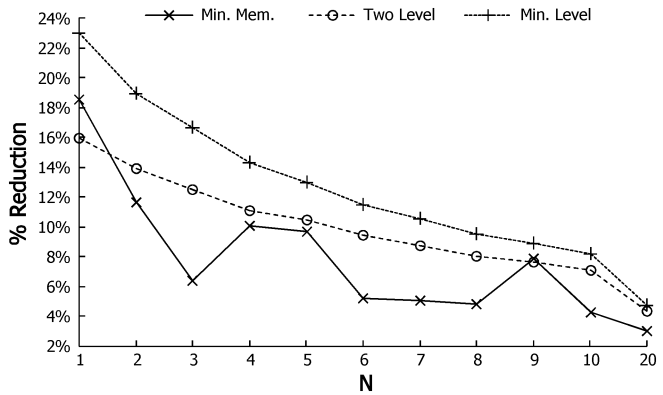


Fig. 27. Logic cluster area reduction ($k = 5$).

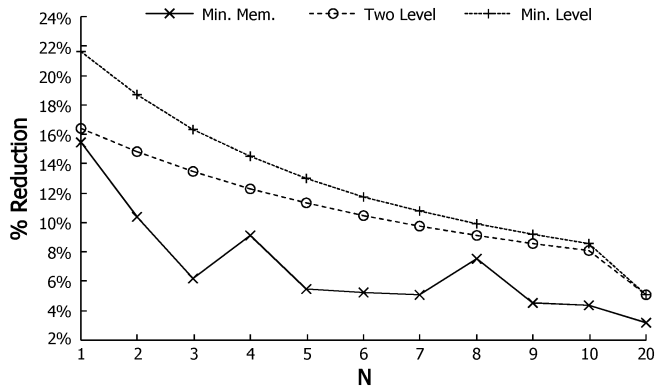


Fig. 28. Logic cluster area reduction ($k = 6$).

memory design. Consequently, an equal percentage of area reduction in local routing networks would result in a relatively larger percentage reduction in logic cluster area.

Note that a local routing network is constructed out of $N \times k$ multiplexers. In the traditional design each multiplexer requires $I + N$ inputs. Consequently the entire local routing network requires $N \times k \times (I + N)$ input signals. Since these signals are connected to I logic cluster inputs and N feedbacks, each logic cluster input/feedback signal has a fanout of $N \times k$. In the new design, each multiplexer requires only $I + N - k + 1$ inputs. Consequently, the local routing network requires only $N \times k \times (I + N - k + 1)$ inputs. This reduction in the number

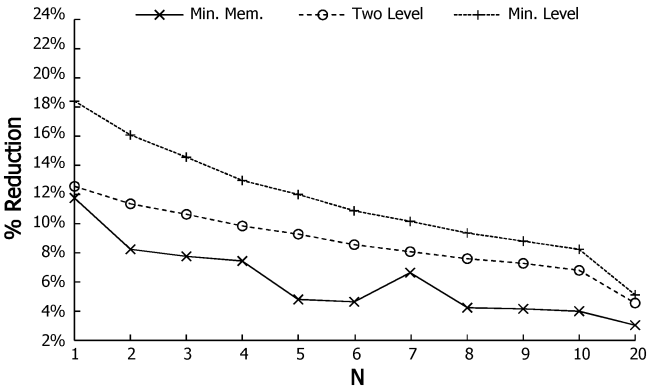


Fig. 29. Logic cluster area reduction ($k = 7$).

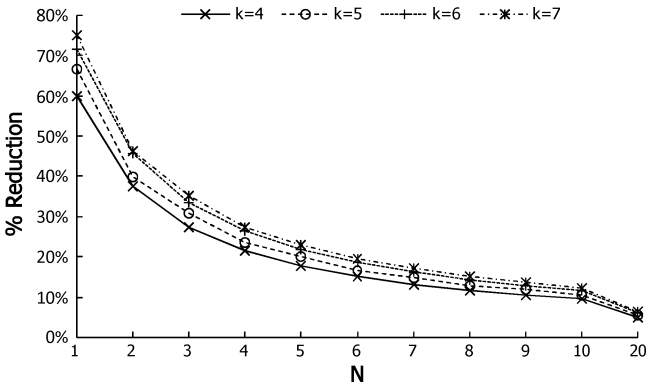


Fig. 30. Average fanout reduction per input/feedback.

of inputs translates to a reduction in fanouts for a subset of logic cluster input/feedback signals.

In particular, the fanouts of $2 \times (k-1)$ input/feedback signals are reduced in the new design. Two of these signals have a fanout of N in the new design, two others have a fanout $2 \times N$, while the remaining pairs have fanouts of $3 \times N, \ldots, (k-1) \times N$, respectively. Note that the remaining input/feedback signals in the new design maintain a fanout of $N \times k$. Overall, the new design has a total fanout reduction of $N \times k \times (k-1)$ over all input/feedback signals. Fig. 30 shows the average fanout reduction per input/feedback.

Finally, Table II shows the fanout adjusted area for the cluster size of 6 and LUT sizes of 4–7. Note that the logic cluster area shown in the table includes the area occupied by buffers that are used to drive the inputs of logic clusters. As in [9] these buffers are sized according to the number of fanouts that they drive. As shown for these typical cluster configurations, the new sparse local routing networks can reduce the total logic cluster area by 6.2%–12.4% while fully retain the routability of the original fully connected local routing networks.

VII. Limitations and Future Work

This paper assumes that all inputs to an LUT can be made logically equivalent through LUT reconfiguration. While the assumption is true for the BLEs used in the academic VPR tool [2], it is not true for many real-world FPGAs. For example, each LUT in the Altera Stratix II series of FPGAs [7], can be configured into a variety of LUT sizes. They can also be used in

TABLE II
FANOUT ADJUSTED LOGIC CLUSTER AREA FOR CLUSTERS CONTAINING 6 FOUR LUTS, FIVE LUTS, SIX LUTS, OR SEVEN LUTS

K	I	Min. Mem.			Min. Level			Two Level		
		Trad.	New	% Reduc.	Trad.	New	% Reduc.	Trad.	New	% Reduc.
4	14	3376.4	3158.8	6.4%	5104.4	4526.8	11.3%	4048.4	3710.8	8.3%
5	17	5089.3	4729.3	7.1%	7849.3	6889.3	12.2%	6229.3	5569.3	10.6%
6	21	7609.1	7085.6	6.9%	11461.1	10037.6	12.4%	9337.1	8273.6	11.4%
7	24	11877.0	11139.9	6.2%	17211.0	15213.9	11.6%	14313.0	12945.9	9.6%

a variety of arithmetic modes. These additional functionalities reduce the logic equivalency among LUT inputs. Applying the results presented in this paper to the design of these more complex logic cells is left as future work.

The work in [6] describes a sparsely populated local routing network used in the Actel FPGAs. In particular, in the Actel architecture, each logic cluster contains eight four-input LUTs and each LUT input is connected to eight of the 32 logic cluster inputs (and to no feedbacks) through an 8:1 multiplexer. This architecture sacrifices logically equivalent logic cluster I/Os and local feedbacks for a larger logic cluster size and correspondingly wider routing channels. The exact mechanisms and benefits of such a trade-off, however, is not fully understood. The tradeoff can be justified by the sparseness of the local routing networks in [6]. In particular, according to (2), in a logic cluster containing a fully connected local routing network, the size of the cluster N is governed by the size of the multiplexers attached to the LUT inputs S, and the size of the LUTs k

$$N = \left\lfloor \frac{2S-k}{k+2} \right\rfloor. \tag{3}$$

Consequently, 8:1 multiplexers can only be used to construct a logic cluster containing two four-input LUTs. Since previous work [1], [3] has demonstrated that logic clusters with two LUTs are not area efficient, [6] did not consider reducing the routing channel width and experimentally evaluate the area efficiency of logic clusters containing 2 four-input LUTs.

The results from this study, however, changes the relationship among N, S, and k to

$$N = \left\lfloor \frac{2S+k-2}{k+2} \right\rfloor. \tag{4}$$

Consequently, 8:1 multiplexers can be used to construct logic clusters containing 3 four-input LUTs and three LUT clusters have been shown to be area efficient in [1]. In particular, the 3 four-input LUT clusters would consume no more local routing network area per LUT than the 8 four-input LUT cluster design while retaining full logic equivalency among logic cluster I/Os and allowing local feedbacks. With narrower routing channels surrounding each three LUT cluster, the three LUT design can be competitive and should be experimentally evaluated as an extension to [6].

Alternatively, since the logic clusters proposed in [6] do not contain feedbacks, the 8:1 multiplexers can be used to construct logic clusters containing 4 four-input LUTs with eleven logic cluster inputs. Again, this design would require a narrower channel width in order to support the smaller four LUT clusters. Since the four LUT design retains logic equivalency among the logic cluster I/Os and has less logic cluster inputs per LUT than the design proposed in [6], it can be potentially more area efficient. This design should also be experimentally evaluated as an extension to [6] in future work, along with an examination on the effect of the sparse local routing network design on the power efficiency of FPGAs.

VIII. CONCLUSION

This paper has examined the relationship between logic equivalency of logic cluster I/Os and LUT reconfiguration for FPGA local routing networks. It is shown that through LUT reconfiguration, one can reduce the size of multiplexers used in FPGA local routing networks from $(I+N)$:1 to $(I+N-k+1)$:1, where I is the number of logic cluster inputs, N is the number of logic cluster feedbacks, and k is the number of LUT inputs. It is shown that the $(I+N-k+1)$:1 size is the minimum multiplexer size required to achieve fully equivalent logic cluster I/Os through LUT reconfiguration. The paper has also examined the detailed area reduction figures for logic clusters with LUT size of 4–7. It is shown that with LUT reconfiguration the local routing network area can be reduced by 48%–72% when N is equal to 1 and 3.7%–6.5% when N is equal to 20. This reduction in local routing network area translates to a 2.9%–25% area reduction in logic cluster area. Finally, it is also shown that LUT reconfiguration also reduces a significant amount of fanouts for logic cluster inputs and feedbacks.

APPENDIX
PROOFS OF LUT PROPERTIES

Property 1—Commutative Property: Let $1 \le x < y \le k$. Let $f(a_1, a_2, \ldots, a_k)$ be a Boolean function with k inputs. Let $i_1, i_2, \ldots, i_k$ be k independent Boolean variables. For f there exists a function f', such that

$$\begin{aligned} &f'(i_1, i_2, \ldots, i_{x-1}, \boldsymbol{i_y}, i_{x+1}, i_{x+2}, \ldots, \\ &\qquad i_{y-1}, \boldsymbol{i_x}, i_{y+1}, i_{y+2}, \ldots, i_k) \\ &= f(i_1, i_2, \ldots, i_{x-1}, \boldsymbol{i_x}, i_{x+1}, i_{x+2}, \ldots, \\ &\qquad i_{y-1}, \boldsymbol{i_y}, i_{y+1}, i_{y+2}, \ldots, i_k). \end{aligned}$$

Proof: Define f' in terms of f based on the following four equations:

$$\begin{aligned} &f'(a_1, a_2, \ldots, a_k)|_{(a_x=0, a_y=1)} \\ &= f(a_1, a_2, \ldots, a_k)|_{(a_x=1, a_y=0)} \end{aligned} \tag{5}$$

$$\begin{aligned} &f'(a_1, a_2, \ldots, a_k)|_{(a_x=1, a_y=0)} \\ &= f(a_1, a_2, \ldots, a_k)|_{(a_x=0, a_y=1)} \end{aligned} \tag{6}$$

$$\begin{aligned} &f'(a_1, a_2, \ldots, a_k)|_{(a_x=0, a_y=0)} \\ &= f(a_1, a_2, \ldots, a_k)|_{(a_x=0, a_y=0)} \end{aligned} \tag{7}$$

$$\begin{aligned} &f'(a_1, a_2, \ldots, a_k)|_{(a_x=1, a_y=1)} \\ &= f(a_1, a_2, \ldots, a_k)|_{(a_x=1, a_y=1)}. \end{aligned} \tag{8}$$

Let $F = f(i_1, i_2, \ldots, i_{x-1}, \boldsymbol{i_x}, i_{x+1}, i_{x+2}, \ldots, i_{y-1}, \boldsymbol{i_y}, i_{y+1}, i_{y+2}, \ldots, i_k)$ and $F' = f'(i_1, i_2, \ldots, i_{x-1}, \boldsymbol{i_y}, i_{x+1}, i_{x+2}, \ldots, i_{y-1}, \boldsymbol{i_x}, i_{y+1}, i_{y+2}, \ldots, i_k)$.

Equation (5) gives $F|_{(i_x=1, i_y=0)} = f(i_1, i_2, \ldots, i_{x-1}, \mathbf{1}, i_{x+1}, i_{x+2}, \ldots, i_{y-1}, \mathbf{0}, i_{y+1}, i_{y+2}, \ldots, i_k) = f'(i_1, i_2, \ldots,$

$i_{x-1}, \mathbf{0}, i_{x+1}, i_{x+2}, \ldots, i_{y-1}, \mathbf{1}, i_{y+1}, i_{y+2}, \ldots, i_k) = F'|_{(i_x=1,i_y=0)}$.

Similarly, (6) gives $F|_{(i_x=0,i_y=1)} = F'|_{(i_x=0,i_y=1)}$, (7) gives $F|_{(i_x=0,i_y=0)} = F'|_{(i_x=0,i_y=0)}$ and (8) gives $F|_{(i_x=1,i_y=1)} = F'|_{(i_x=1,i_y=1)}$.

Therefore, $F = F'$. ■

Property 2—Duplicate-Constant Input Equivalence: Let $1 \leq x < y \leq k$. Let $f(a_1, a_2, \ldots, a_k)$ be a Boolean function with k inputs. Let $i_1, i_2, \ldots, i_k$ be k Boolean variables. If $\boldsymbol{i_x} = \boldsymbol{i_y}$, then there exists a function f' such that

$$\begin{aligned} f'(i_1, i_2, \ldots, i_{x-1}, \boldsymbol{i_x}, i_{x+1}, i_{x+2}, \ldots, \\ i_{y-1}, \mathbf{0}, i_{y+1}, i_{y+2}, \ldots, i_k) \\ = f(i_1, i_2, \ldots, i_{x-1}, \boldsymbol{i_x}, i_{x+1}, i_{x+2}, \ldots, \\ i_{y-1}, \boldsymbol{i_y}, i_{y+1}, i_{y+2}, \ldots, i_k). \end{aligned}$$

Proof: Define f' in terms of f based on the following two equations:

$$f'(a_1, a_2, \ldots, a_k)|_{(a_x=1)} = f(a_1, a_2, \ldots, a_k)|_{(a_x=1,a_y=1)} \quad (9)$$

$$f'(a_1, a_2, \ldots, a_k)|_{(a_x=0)} = f(a_1, a_2, \ldots, a_k)|_{(a_x=0,a_y=0)}. \quad (10)$$

Let $F = f(i_1, i_2, \ldots, i_{x-1}, \boldsymbol{i_x}, i_{x+1}, i_{x+2}, \ldots, i_{y-1}, \boldsymbol{i_y}, i_{y+1}, i_{y+2}, \ldots, i_k)$ and $F' = f'(i_1, i_2, \ldots, i_{x-1}, \boldsymbol{i_x}, i_{x+1}, i_{x+2}, \ldots, i_{y-1}, \mathbf{0}, i_{y+1}, i_{y+2}, \ldots, i_k)$.

Equation (9) gives $F|_{(i_x=1)} = f(i_1, i_2, \ldots, i_{x-1}, \mathbf{1}, i_{x+1}, i_{x+2}, \ldots, i_{y-1}, \mathbf{1}, i_{y+1}, i_{y+2}, \ldots, i_k) = f'(i_1, i_2, \ldots, i_{x-1}, \mathbf{1}, i_{x+1}, i_{x+2}, \ldots, i_{y-1}, \mathbf{0}, i_{y+1}, i_{y+2}, \ldots, i_k) = F'|_{(i_x=1)}$.

Similarly, (10) gives $F|_{(i_x=0)} = F'|_{(i_x=0)}$. Therefore, $F = F'$. ■

Property 3—Constant-New Input Equivalence: Let $1 \leq x \leq k$. Let $f(a_1, a_2, \ldots, a_k)$ be a Boolean function with k inputs. Let $i_1, i_2, \ldots, i_k$ be k independent Boolean variables. For f there exists a function f' such that

$$\begin{aligned} f'(i_1, i_2, \ldots, i_{x-1}, \boldsymbol{i_x}, i_{x+1}, i_{x+2}, \ldots, i_k) \\ = f(i_1, i_2, \ldots, i_{x-1}, \mathbf{0}, i_{x+1}, i_{x+2}, \ldots, i_k). \end{aligned}$$

Proof: Define f' in terms of f based on the following two equations:

$$f'(a_1, a_2, \ldots, a_k)|_{(a_x=1)} = f(a_1, a_2, \ldots, a_k)|_{(a_x=0)} \quad (11)$$

$$f'(a_1, a_2, \ldots, a_k)|_{(a_x=0)} = f(a_1, a_2, \ldots, a_k)|_{(a_x=0)}. \quad (12)$$

Let $F = f(i_1, i_2, \ldots, i_{x-1}, \mathbf{0}, i_{x+1}, i_{x+2}, \ldots, i_k)$ and $F' = f'(i_1, i_2, \ldots, i_{x-1}, \boldsymbol{i_x}, i_{x+1}, i_{x+2}, \ldots, i_k)$.

Equation (11) gives $F|_{(i_x=1)} = f(i_1, i_2, \ldots, i_{x-1}, \mathbf{0}, i_{x+1}, i_{x+2}, \ldots, i_k) = f'(i_1, i_2, \ldots, i_{x-1}, \mathbf{1}, i_{x+1}, i_{x+2}, \ldots, i_k) = F'|_{(i_x=1)}$.

Similarly, (12) gives $F|_{(i_x=0)} = F'|_{(i_x=0)}$. Therefore, $F = F'$. ■

REFERENCES

[1] E. Ahmed and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 3, pp. 288–298, Mar. 2004.

[2] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. New York: Kluwer, Feb. 1999.

[3] V. Betz and J. Rose, "How much logic should go in an FPGA logic block?," *IEEE Des. Test Comput. Mag.*, vol. 15, no. 1, pp. 10–15, Jan. –Mar. 1998.

[4] V. Betz and J. Rose, "Effect of the prefabricated routing track distribution on FPGA area-efficiency," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 6, no. 3, pp. 445–456, Sep. 1998.

[5] A. Marquardt, V. Betz, and J. Rose, "Speed and area trade-offs in cluster-based FPGA architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 8, no. 1, pp. 84–93, Feb. 2000.

[6] W. Feng and S. Kaptanoglu, "Designing efficient input interconnect blocks for LUT clusters using counting and entropy," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, no. 2, pp. 6:1–6:28, Jun. 2008.

[7] D. Lewis, "The stratix II logic and routing architecture," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2005, pp. 14–20.

[8] D. Lewis, "The stratix routing and logic architecture," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2003, pp. 15–20.

[9] G. Lemieux and D. Lewis, "Using sparse crossbars within LUT clusters," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2001, pp. 59–68.

[10] C. E. Shannon, "The synthesis of two-terminal switching circuits," *Bell Syst. Tech. J.*, vol. 28, no. 1, pp. 59–98, Jan. 1949.

[11] G. Lemieux, "Directional and single-driver wires in FPGA interconnect," in *Proc. IEEE Int. Conf. Field-Program. Technol.*, 2004, pp. 41–48.

[12] V. Betz and J. Rose, "FPGA routing architecture: Segmentation and buffering to optimize speed and density," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 1999, pp. 59–68.

[13] A. Roopchansingh and J. Rose, "Nearest neighbour interconnect architecture in deep submicron FPGAs," in *Proc. IEEE Custom Integr. Circuits Conf.*, 2002, pp. 59–62.

[14] A. Ye and J. Rose, "Using bus-based connections to improve field-programmable gate array density for implementing datapath circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 5, pp. 462–473, May 2006.

[15] K. Poon, A. Yan, and S. Wilton, "A flexible power model for FPGAs," in *Proc. Int. Conf. on Field-Program. Logic Appl.*, 2002, pp. 48–58.

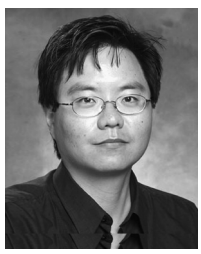
[16] M. Masud and S. Wilton, "A new switch block for segmented FPGAs," in *Proc. Int. Workshop Field Program. Logic Appl.*, 1999, pp. 274–281.

[17] A. Singh, G. Parthasarathy, and M. Marek-Sadowska, "Efficient circuit clustering for area and power reduction in FPGAs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 7, no. 4, pp. 643–663, Oct. 2002.

[18] A. Singh, A. Mukherjee, and M. Marek-Sadowska, "Interconnect pipelining in a throughput-intensive FPGA architecture," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2001, pp. 153–160.

[19] E. Bozorgzadeh, S. Ogrenci-Memik, and M. Sarrafzadeh, "RPack: Routability-driven packing for cluster-based FPGAs," in *Proc. Conf. Asian South Pacific Des. Autom.*, 2001, pp. 629–634.

[20] F. Li, "Architecture evaluation for power-efficient FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2003, pp. 175–184.



Andy Gean Ye (S'97–M'06) received the B.A.Sc., M.A.Sc., and Ph.D. degrees in computer engineering from the University of Toronto, Toronto, ON, Canada, in 1996, 1999, and 2004, respectively. He graduated first in class in the Engineering Science Program in 1996.

From 1999 to 2000, he participated in the development of the Ultragizmo board for the University of Toronto Undergraduate Microprocessor Laboratory. He is currently an Assistant Professor in the Department of Electrical and Computer Engineering, Ryerson University, Toronto. His current research interests include field-programmable gate array (FPGA) architectures, computer-aided design (CAD) tools for FPGAs, logic synthesis, and hardware implementation of computer graphics algorithms.