Enhancement of Incremental Design for FPGAs Using Circuit Similarity

Xiaoyu Shi[1], Dahua Zeng[1], Yu Hu[2], Guohui Lin[1], Osmar R. Zaiane[1]
[1]Department of Computing Science, University of Alberta, Canada
[1]{xshi, dahua, guohui, zaiane}@cs.ualberta.ca
[2]Department of Electrical and Computer Engineering, University of Alberta, Canada
[2]bryanhu@ece.ualberta.ca
[*]

ABSTRACT

This paper presents an efficient algorithm to detect the global topological similarity between two circuits. By applying the proposed circuit similarity algorithm in an incremental design flow, *IDUCS* (incremental design using circuit similarity), the design and optimization effort in the previous design iterations is automatically captured and can be used to guide the next design iteration. IDUCS is able to identify the similarity between the original netlist and the modified one with aggressive resynthesis, which might destroy the naming and local structures of the original netlist. This is superior to the existing design preservation approaches such as naming and local topological matching. Furthermore, IDUCS simply inserts a plugin for circuit similarity detection, and therefore preserves the "push-button" feature, significantly simplifying the engineering complexity of incremental tasks. As a case study, we perform the proposed IDUCS process to generate the placement for a logically resynthesized netlist based on the placement of the original netlist and the circuit similarity between the original and the modified logic-level netlists. The experimental results show our IDUCS-based placement is 28X faster than versatile place and route (VPR) with comparable wire length and estimated critical delay.

Keywords

Circuit similarity, FPGA, Incremental design

1. INTRODUCTION

In a typical field programmable gate array (FPGA) design cycle, series of synthesis iterations need to be performed before delivering the final design. The recompilation time for these iterations heavily affects the time-to-market of a product. There are several phases of a design process in which iterative repetitions are common, including the initial checks of the register transfer level (RTL) code, constraint verification, timing closure, and in-system debugging [6]. Each of these steps requires a time-consuming resynthesis of the FPGA design. Incremental design methodology has been devised to save the recompilation time by maintaining the essential properties across consecutive iterations [9, 12, 23, 1, 25].

The key to incremental design methodology is *design preservation* [22], *i.e.*, maximally preserving and taking advantage of the engineering effort in the previous design iterations. A commonly employed method for design preservation is to partition a design and avoid a recompilation of unchanged partitions in the next iteration. This method can yield a significant reduction in iteration time but due to strict hierarchical boundaries, synthesis cannot perform any cross-boundary optimizations where a partition exists. To break this hard hierarchy boundary constraint for improving the quality of the design, Xilinx SmartGuide [22] employs naming and local topological matching to identify the correspondence between two netlists resulting from the previous and the current iterations, respectively. Based on this correspondence, the layout from the previous iteration can be reused in the current iteration, leading to better quality and saving the recompilation time. However, in modern synthesis algorithms (*e.g.*, ABC [4]), the internal node boundaries are usually destroyed by structural hashing (transforming a logic network into an and-inverter graph (AIG)), and aggressive optimization and logic restructuring performed in the netlist make it difficult to produce the naming matching between the original and modified netlists. Consequently, the local topological matching based on the naming matching also becomes less effective.

In this paper, we present *IDUCS*, an enhanced incremental design using *circuit similarity*[1] flow for FPGAs. In contrast to Xilinx SmartGuide [22], the circuit similarity identifies a correspondence between the original and modified netlists based on a global topology matching. Based on circuit similarity, the placement and routing of the modified netlist can be derived from the layout of the original netlist obtained in the previous iteration. Unlike many existing algorithms for incremental designs [12, 1], which require radical changes to existing computer-aided design (CAD) tools, we have developed a plugin that preserves the "push-button" feature in the commercial FPGA CAD tools.

A key insight used in IDUCS is that incremental functional changes in RTL or logic level are small, and they generally result in a "similar" topology of the modified netlist compared with the original one [17]. To quantita-

[*]This work was partially sponsored by the Natural Sciences and Engineering Research Council of Canada.

[1]Note that the same notion of "circuit similarity" was used in [11], but it was defined based on the Boolean functions of logic gates in a circuit. In the future, it will be interesting to combine our topology-based circuit similarity with the function-based one to further accelerate the verification process in an incremental design.

tively represent such a similarity, we adapt *graph similarity* [26], a widely applied technique in social network and chem-informatic domains, to measure the topological similarity of two circuits. We present an iterative algorithm to compute the circuit similarity between the modified and original netlists, and identify the correspondence of nodes/edges.

The IDUCS flow is shown in Figure 1. IDUCS produces an initial placement and routing solution for the modified logic-level netlist, based on the layout of the original netlist and the circuit similarity between the original and modified logic-level netlists. Based on this initial solution, an efficient refinement is then performed as a fine-grain tuning for further improvement of the layout quality. Note that such a refinement procedure does not require a new implementation since the existing placement and routing tools can be used with less optimization strength (*e.g.*, lower initial temperature in the simulated annealing-based placement or fewer iterations in the negotiation-based routing). The essential information obtained from the previous design iteration is automatically captured and quantified by a runtime-efficient "similarity detection" phase.

To verify the effectiveness of IDUCS, we have applied it to placement acceleration, one of the most time-consuming design phases, in a multi-pass design. We have used IDUCS to generate the placement for a logically resynthesized netlist based on the placement of the original netlist and the circuit similarity between the original and the modified logic-level netlists. Tested on the 20 largest MCNC benchmark circuits [24], experimental results show IDUCS produces a much higher quality initial placement than VPR's [20] initial placement in terms of bounding box costs and delay costs. Our IDUCS-based placement is 28X faster on average than the VPR placement, while producing comparable wire length and critical delay. The results suggest a huge potential to accelerate other incremental design phases, including routing and verification, using circuit similarity.

The remainder of this paper is organized as follows. Section 2 illustrates the overall IDUCS flow with an example. Section 3 describes the circuit similarity algorithm. Section 4 presents a placement case study to experimentally demonstrate the efficiency and the effectiveness of IDUCS. The paper is concluded in Section 5.

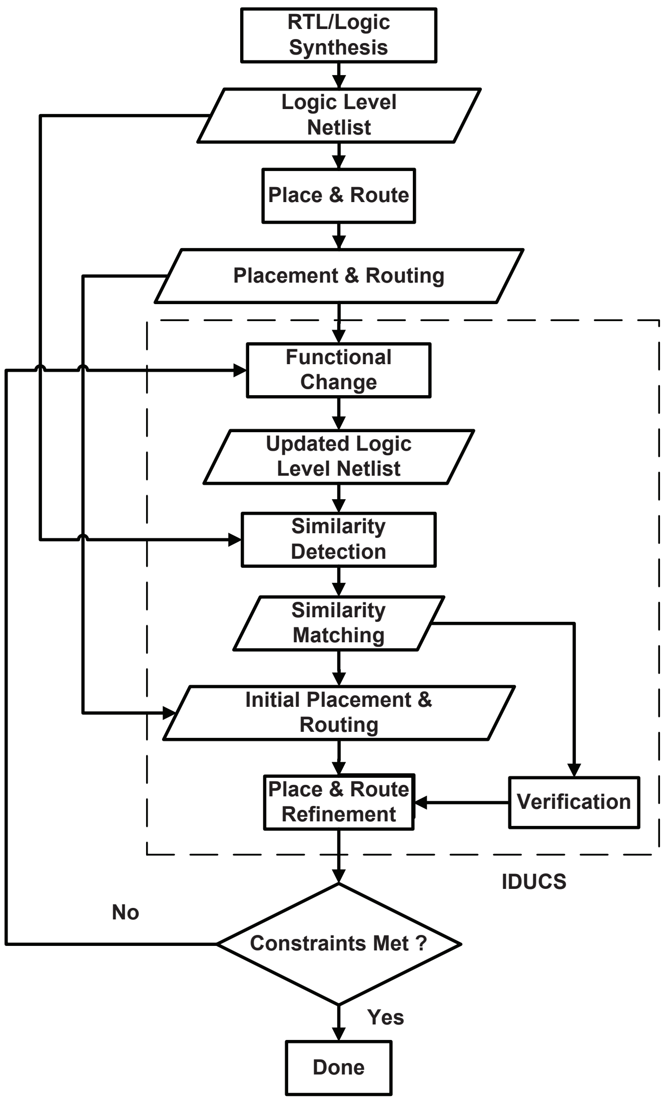


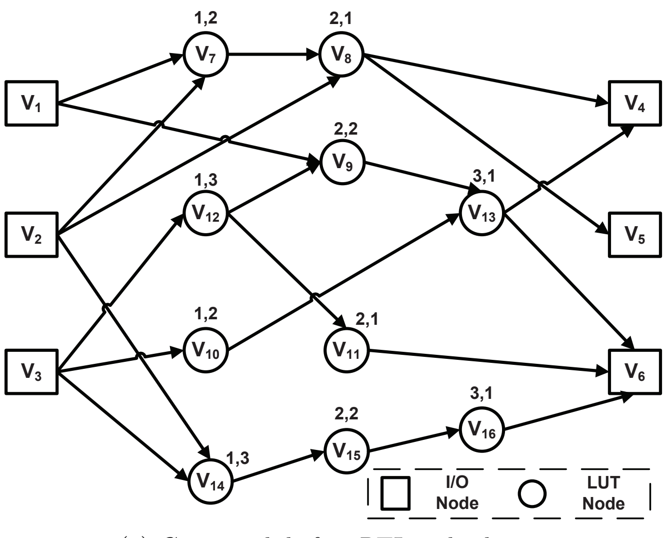
Figure 1: Incremental design flow using circuit similarity

2. MOTIVATING EXAMPLE

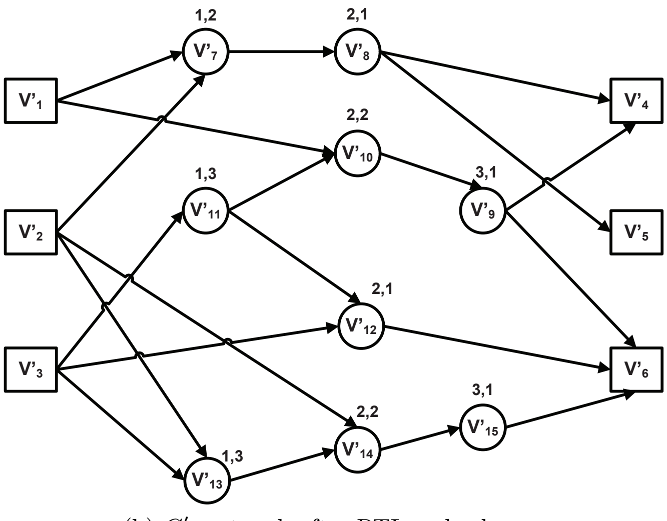
Following the flow in Figure 1, we use an example to illustrate the procedures of using IDUCS to generate the placement based on the layout results obtained from the previous design iteration. In the first design iteration, given a logic-level network G shown in Figure 2(a), where each node denotes a look-up table (LUT) and each edge denotes an interconnection between LUTs, the placement (Figure 3(a)) of network G can be obtained by performing a time-consuming and highly-optimized placement (*e.g.*, VPR). Suppose a change of RTL code is made due to a bug found after the first iteration, and the RTL and logic-level synthesis is performed in the following iteration, resulting in a modified network, G', as shown in Figure 2(b). To produce the placement of network G', IDUCS first computes the similarity between networks G and G', and finds the correspondence of nodes in these two networks (Figure 3(a) right). Based on such node correspondence, the initial placement (Figure 3(b)) of network G' can be determined using the placement of network G (Figure 3(a)), *e.g.*, if node V' in network G' corresponds to node V in network G, V' is assigned the same coordinates as node V.

Note that the detection of similarity and the correspondence of two networks is generally much faster than the replacement of the entire network. Therefore, the IDUCS-based approach is more efficient than the *from-scratch* design flow, which re-places the entire circuit. Furthermore, the naming matching-based correspondence will not work in this example since only two nodes (node 7 and node 8) out of nine internal nodes have the same names in the original and the modified networks. On the other hand, IDUCS employs a topological similarity detection technique and is able to identify a more comprehensive correspondence between the two networks. In general, IDUCS-based flow contains the following two phases:

1. Detection of the similarity between two networks and the correspondence of the components (*e.g.*, nodes and edges) in them;



(a) G, network before RTL code change



(b) G', network after RTL code change

Figure 2: Logic-level networks before and after optimization (the label above each node describes the level and reverse level of the node)

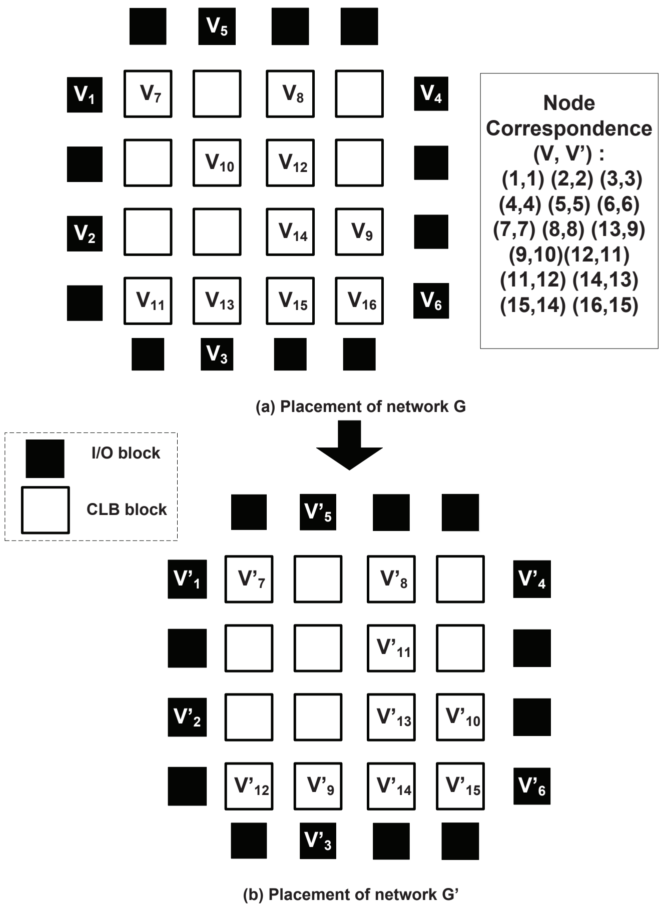
2. Refinement of the results inferred based on the previous design iteration and the detected similarity, *e.g.*, resolving overlaps in the initial placement and congestions in the initial routing.

Section 3 and Section 4 will detail these two aspects, respectively.

3. CIRCUIT SIMILARITY

3.1 Review of Graph Similarity

Given two graphs (or networks), there are multiple ways to define their similarity. The characteristics of commonly used measures of similarity are summarized in Table 1, where column "Global Topo" indicates whether a measure considers the global topological information, which is important to find the correspondence between nodes of two graphs. Some measures have already been used for FPGA design automation, *e.g.*, J. Cong *et al.* applied the edit distance measure to FPGA resource optimization [10]. Our IDUCS employs the iterative method, which has relatively low computational complexity and considers the global topological information.



V5
V1
V7
V8
V4
V10
V12
V2
V14
V9
V11
V13
V15
V16
V6
V3
Node Correspondence (V, V') : (1,1) (2,2) (3,3) (4,4) (5,5) (6,6) (7,7) (8,8) (13,9) (9,10)(12,11) (11,12) (14,13) (15,14) (16,15)
(a) Placement of network G
I/O block
CLB block
V'5
V'1
V'7
V'8
V'4
V'11
V'2
V'13
V'10
V'12
V'9
V'14
V'15
V'6
V'3
(b) Placement of network G'

Figure 3: The placement of the original and modified networks

Different algorithms, including similarity flooding [18], simRank [7], and the coupled node-edge [26], have been proposed to compute the graph similarity based on the iterative definition. In this work, we use an iterative graph similarity algorithm for molecular graphs [14], which takes advantage of graph sparsity, one of the properties of a circuit graph. Table 2 describes all frequently used variables in this algorithm.

The iterative similarity algorithm is summarized in Algorithm 1. In each iteration (t), the algorithm computes the *similarity score*, $X_{i,j}^{(t)}$, between each node pair (v_i, v_j'), where $v_i \in G$ and $v_j' \in G'$. The similarity score of a node pair is a real value between 0 and 1. The higher the similarity score of a node pair is, the more likely these two nodes are matched together. This score is updated based on the values of their adjacent node pairs obtained in the previous iteration and the predefined inter-similarity between two nodes/edges. The predefined similarity is used to capture non-topological connections between two graphs. The algorithm terminates when the difference between of the total similarity scores in two consecutive iterations is smaller

Table 1: Summary of notions of similarity

Measure	Description	Time Complexity	Global Topo
Isomorphism [15]	Identifying a bijection between the nodes of two graphs which preserves (directed) adjacency.	NP-Hard	Yes
Edit distance [5]	Given a cost function on edit operations (*e.g.*, addition/deletion of nodes and edges), determine the minimum cost transformation from one graph to another.	NP-Hard	Yes
Common subgraph [13]	Identifying the 'largest' isomorphic subgraphs of two graphs.	NP-Hard	Yes
Iterative methods [21]	Two graph elements (*e.g.*, edges or nodes) are similar if their neighborhoods are similar.	Cubic	Yes
Statistical methods [16]	Assessing aggregate measures of graph structure (*e.g.*, degree distribution, diameter, betweenness measures).	Linear	No

Table 2: Summary of used variables

Variable	Description
$X_{i,j}^{(n)}$	Similarity score between node i in graph $G(V)$ and node j in graph $G'(V')$ in iteration n
v_i	A node in graph G
v'_j	A node in graph G'
t	The iteration number
$n(v)$	The set of all adjacent nodes of node v
π	An injective map from $n(v_i)$ to $n(v'_j)$, if $\|n(v_i)\| < \|n(v'_j)\|$ An injective map from $n(v'_j)$ to $n(v_i)$, if $\|n(v_i)\| \geq \|n(v'_j)\|$
α	A weight constant within interval (0,1)
ϵ	A terminating threshold for iterations
M	An upper bound for number of iterations
$k_v : V \rightarrow V'$	A predefined inter-similarity between two nodes
$k_e : E \rightarrow E'$	A predefined inter-similarity between two edges, where (v_i, v) is an edge in graph G and $(v'_j, \pi(v))$ is an edge in graph G'
$in(v)$	The set of all adjacent nodes that have an edge entering node v
$out(v)$	The set of all adjacent nodes that have an edge leaving node v

Algorithm 1 Similarity of G and G'

```
Initialize $X_{i,j}^{(0)}$
while $|\sum X^{(t)} - \sum X^{(t-1)}| > \epsilon$ and $t < M$ do
    if $|n(v_i)| < |n(v'_j)|$ then
        $X_{i,j}^{(t)} = (1-\alpha)k_v(v_i, v'_j) + \alpha \max_{\pi} \frac{1}{|n(v'_j)|} \sum_{v \in n(v_i)} X^{(t-1)}_{v,\pi(v)} k_e((v_i, v), (v'_j, \pi(v)))$
    else
        $X_{i,j}^{(t)} = (1-\alpha)k_v(v_i, v'_j) + \alpha \max_{\pi} \frac{1}{|n(v_i)|} \sum_{v' \in n(v'_j)} X^{(t-1)}_{\pi(v'),v'} k_e((v_i, \pi(v')), (v'_j, v'))$
```

than ϵ, or the number of iterations reaches an upper bound M.

3.2 Circuit Similarity Detection

Algorithm 1 is designed for undirected molecular graphs [14], and the computational complexity is too expensive to handle real circuits. In this subsection, we first adapt Algorithm 1 to consider a directed circuit graph and then present two techniques to significantly improve both time and space efficiency of the circuit similarity detection.

One unique constraint for circuit similarity detection in incremental design is that the matching of the corresponding primary inputs (PIs) and primary outputs (POs) of the two circuits must be guaranteed. Therefore, the similarity score for a pair of corresponding PI/PO nodes is set to 1 and is not updated during the iteration. As a result, such a predefined PI/PO matching effectively provides extra hints for the iterative similarity detection process and generates better matching between the two circuits. Intuitively, for those node pairs close to PI/PO nodes, higher scores will be obtained because of the propagation of the constant similarity score set in PI/PO node pairs. Note that other hints such as internal registers and naming matching information obtained in logic synthesis can also be used as the predefined matching to enhance both the quality and speed of the circuit similarity detection.

For those internal nodes without predefined similarity, we replace k_v with $X_{i,j}^{(t)}$, and k_e with 1. Instead of updating similarity scores based on all the neighbors, we can perform the update for edges that leave the nodes and edges that enter the nodes, separately. More specifically, given the two graphs, we initialize the similarity scores of all pairs of nodes to 1. In each iteration, for $|in(v_i)| < |in(v'_j)|$ and $|out(v_i)| < |out(v'_j)|$, the similarity score $X_{i,j}^{(t)}$ is updated as follows

$$X_{i,j}^{(t)} = (1-\alpha)X_{i,j}^{(t-1)} + \alpha \frac{1}{|out(v_i)| + |in(v_i)|} \left[\max_{\pi}\left(\sum_{v' \in out(v'_j)} X^{(t-1)}_{\pi(v'),v'}\right) + \max_{\pi}\left(\sum_{v' \in in(v'_j)} X^{(t-1)}_{\pi(v'),v'}\right)\right] \tag{1}$$

In our experiments, we find $\alpha = 0.75$ consistently produces a high quality matching. For the two circuit graphs in Figure 2, the obtained similarity score matrix is shown in Table 3 (PI/PO nodes are not shown). Clearly, the topologically similar node pairs (*e.g.*, node 7 in graph G and node 7 in graph G') have scores close to 1. This matrix describes a complete bipartite graph, where the weight associated with each edge denotes the similarity score of two nodes. Now we can compute a maximum matching in this bipartite graph to obtain a node matching between the two graphs. The min-cost network flow [19] is used to compute the maximum matching in our experiments, and the resulting node matching is given in Figure 3(a) right.

Table 3: Similarity score matrix for two graphs in Figure 2

	V_7	V_8	V_9	V_{10}	V_{11}	V_{12}	V_{13}	V_{14}	V_{15}	V_{16}
V_7'	0.92	0.25	0.48	0.15	0	0	0	0.42	0.06	0
V_8'	0	0.73	0	0	0.05	0	0.39	0	0.17	0.06
V_9'	0	0.39	0	0	0.4	0	0.73	0	0.06	0.48
V_{10}'	0.48	0	0.89	0.25	0.3	0.12	0.14	0.06	0.33	0.09
V_{11}'	0	0	0.11	0.48	0	0.86	0	0.36	0.17	0
V_{12}'	0	0	0.3	0.34	0.64	0.25	0.39	0.34	0.15	0.42
V_{13}'	0.48	0.25	0.07	0.4	0	0.36	0	0.88	0.06	0
V_{14}'	0.4	0.39	0.29	0.15	0.15	0.18	0.12	0.46	0.59	0.06
V_{15}'	0	0.12	0.09	0	0.63	0	0.36	0	0.27	0.82

3.3 Performance Enhancement

In practice, it is infeasible to compute the similarity scores of all $|V|\cdot|V'|$ node pairs for large circuits. In this subsection, we present two pruning techniques to reduce the number of pairs that need to be updated so that we can reduce both the runtime and storage.

Support Constraint. Two internal nodes are less likely to be matched if they share few predefined matchings in their supports. A *support* of a node is the set of nodes with predefined matchings in the transitive fanin or fanout cone of this node. For example, the nodes with predefined matchings are PIs and POs in two graphs in Figure 2. The support for node V_7 is $SP(V_7) = \{V_1, V_2, V_4, V_5\}$, while the support for node V_{15}' is $SP(V_{15}') = \{V_2', V_3', V_6'\}$. The *support similarity* of V_7 and V_{15}' is the sum of similarity scores of all $V \to V'$ node pairs in their supports: $X_{SP(V_7),SP(V_{15}')} = X_{V_2,V_2'} = 1$. On the other hand, the support similarity of V_7 and V_7' is 4. Therefore, V_7 is more likely to be matched with V_7' than with V_{15}'. Formally, for two nodes $v \in G$ and $v' \in G'$, the support constraint requires

$$\min\left(\frac{X_{SP(v),SP(v')}}{|SP(v)|}, \frac{X_{SP(v),SP(v')}}{|SP(v')|}\right) \geq \beta \qquad (2)$$

where $\beta \in (0, 1]$ is a constant. If the support constraint of the two nodes is not satisfied, we do not update their similarity score in the iteration. For example, if $\beta = 1$, we only keep the pairs of nodes that have exactly the same supporting PIs and POs. 54 node pairs (*e.g.*, $(V_7, V_{11}'), (V_7, V_{12}')$) in Figure 2 can be pruned.

Level Constraint. If only combinatorial resynthesis is involved in an incremental design process, we can convert a circuit into a directed acyclic graph (DAG) by removing all registers and adding the register inputs (outputs) as POs (PIs). Given a DAG, a topological sort and reverse topological sort can label each internal node v with two values (shown above each node in Figure 2), *i.e.*, $level(v)$ and $rlevel(v)$, where $level(v)$ ($rlevel(v)$) denotes the length of the longest path from PIs (node v) to node v (POs). Two nodes with significantly different ($level$, $rlevel$) values are less likely to be matched. Formally, for two nodes $v \in G$ and $v' \in G'$, the level constraint requires

$$|level(v) - level(v')| \leq B_l, |rlevel(v) - rlevel(v')| \leq B_r \quad (3)$$

where B_l and B_r are two nonnegative constant integers. For example, if B_l and B_r are both set to be one, 22 node pairs (*e.g.*, (V_7, V_9') and (V_7, V_{15}')) in Figure 2 can be pruned.

We have tested the above two pruning techniques on the 20 standard MCNC benchmark circuits. For each circuit, we run two logic synthesis algorithms, one with ABC command "if -K 4" and the other with "if -K 4; imfs" (an area-oriented resynthesis engine which destroys the internal name matching [2].), and generate two logic-level netlists. Figure 4 compares the number of node pairs that need to be updated in the iterative similarity with the following five schemes: (a) without pruning ("no_pruning"), (b) using a weak level constraint-based pruning ("B_l=B_r=1"), (c) using a strong level constraint-based pruning ("B_l=B_r=0"), (d) using a weak support constraint-based pruning ("β=0.5"), and (e) using a strong support constraint-based pruning ("β=1"). As shown in Figure 4, our pruning techniques reduce the number of node pairs by 3 to 4 orders of magnitude compared with the total number of node pairs. More specifically, the strong level constraint-based pruning ("B_l=B_r=0") and the strong support constraint-based pruning ("β=1") can prune around 90% and 99% node pairs, respectively.

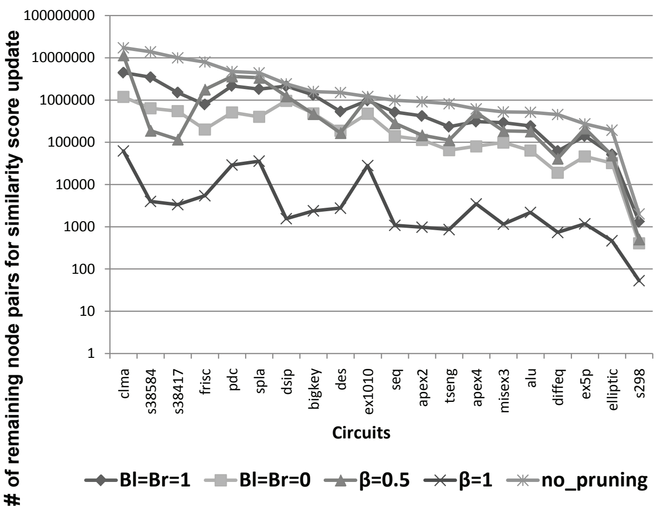


Figure 4: Effectiveness of the pruning techniques

Table 4: Similarity score matrix for two graphs in Figure 2 with pruning ($\beta = 0.5$, $B_l = B_r = 0$)

	V_7	V_8	V_9	V_{10}	V_{11}	V_{12}	V_{13}	V_{14}	V_{15}	V_{16}
V_7'	0.92	0	0	0	0	0	0	0	0	0
V_8'	0	0.73	0	0	0	0	0	0	0	0
V_9'	0	0	0	0	0	0	0.73	0	0	0.48
V_{10}'	0	0	0.89	0	0	0	0	0	0.33	0
V_{11}'	0	0	0	0	0	0.86	0	0.36	0	0
V_{12}'	0	0	0	0	0.64	0	0	0	0	0
V_{13}'	0	0	0	0	0	0.36	0	0.88	0	0
V_{14}'	0	0	0.29	0	0	0	0	0	0.59	0
V_{15}'	0	0	0	0	0	0	0.36	0	0	0.82

Table 4 shows the similarity score matrix obtained after applying these two pruning techniques on the similarity detection of G and G' in Figure 2. Clearly, the iterative circuit similarity with pruning results in a very sparse matrix, while the most significant elements in this matrix are well preserved. For example, (V_{11}, V_8') and (V_9, V_7') are pruned due to the support and level constraints, respectively while (V_9, V_{14}') is not pruned simply because it does not satisfy either of the constraints. Nevertheless, the most useful node pairs are preserved after our pruning and the same node matching can be obtained compared to the completely computed similarity score matrix shown in Table 3. As a result of the sparsity of the similarity matrix, the maximum matching algorithm (min-cost network flow) is significantly faster. In Section 4, we will show that these pruning techniques do not degrade the quality of the similarity detection and node matching when we apply the circuit similarity to the proposed IDUCS flow.

3.4 Circuit Similarity-based Placement

Following Figure 1, circuit similarity detection can be employed to discover the topological correspondence between the original netlist and the modified netlist. Such information is then used to improve the efficiency of time-consuming CAD phases including placement, routing and verification.

We use the proposed circuit similarity algorithm to speed up placement, which is one of the most time-consuming phases in FPGA design cycle. More specifically, given an original network G, its placement can be computed by performing a highly-optimized but time-consuming placement (*e.g.*, VPR). For another network, G', which is modified due to optimization in an incremental iteration, the similarity between networks G and G' is firstly computed, and a matching of corresponding nodes is obtained afterwards. Specifically, if node V' in network G' corresponds to node V in network G, V' is assigned the same coordinates as node V. Hence, the node matching between the two networks gives a good candidate for the initial placement of the modified network G'. In return, based on such node correspondence, a high-quality final layout of network G' can be obtained more efficiently with a further refinement (*e.g.*, low-temperature simulated annealing) on the initial placement results.

4. A CASE STUDY ON PLACEMENT

As a case study on the application of the proposed IDUCS flow, in this section we perform experiments which employ circuit similarity to improve the efficiency of the placement phase in the incremental design flow.

4.1 Experimental Settings

In this work, we consider an island-style FPGA architecture, which includes an array of clustered logic blocks (CLBs) interconnected by programmable routing. As shown in Figure 5, two CAD flows are compared in our experiments. Both flows include two design iterations and they share the first iteration, which starts from a logic-level netlist (BLIF). A technology mapping (using ABC command "if -K 4") is first performed on this netlist to map it into a 4-LUT-based network. After the mapping, T-VPack [20] is performed with "no_cluster" parameter to generate a CLB-based network, where each CLB contains one LUT and one flip-flop. The timing-driven placement in VPR is then used to produce the placement result (".p"), and the timing-driven routing with a detailed timing analysis is finally performed.

In the second iteration, we perform a logic-level optimization on the mapped netlist using the following ABC script "rwsat2":

```
st; rw -l; b -l; rw -l; rf -l; fraig; rw -l; b -l; rw -l; rf -l
```

where each command (alias) is a logic optimization in

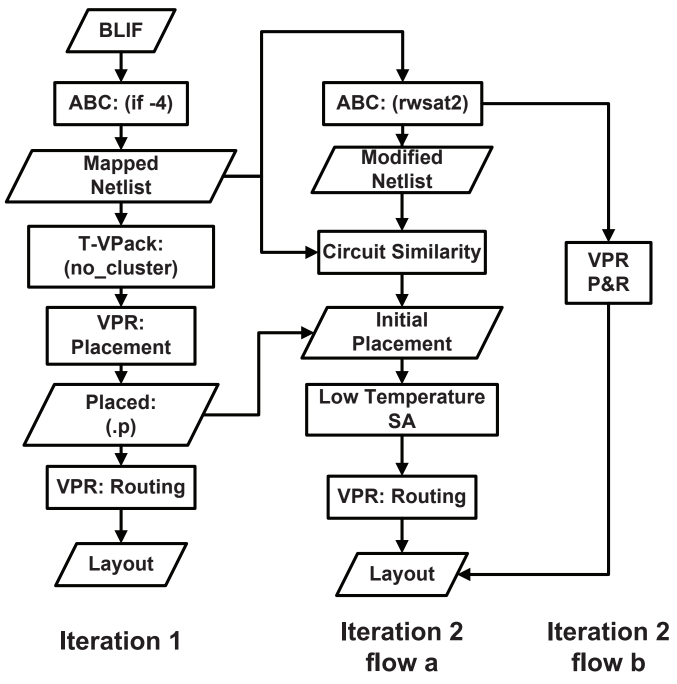


Figure 5: CAD flows used in the experiments

ABC, *e.g.*, "st" (structural hashing) aggressively destroys the initial boundaries among internal nodes (LUTs); "rw" (rewrite) and "rf" (refactor) reconstruct the network by reducing the AIG size and level; "fraig" (functionally-reduced AIG) changes the current network structure and transforms into a functionally-reduce AIG [3]. Hence, in the modified netlist, the name matching among the nodes are not preserved and the structure of the network is changed. We will employ the proposed circuit similarity to discover the node correspondence purely based on topological information of the original and the "rwsat2"-modified netlists.

Starting from the modified netlist, we compare the following two flows: (a) IDUCS flow and (b) *from-scratch* flow, as shown in Figure 5. Flow (b) uses VPR to replace the entire modified netlist. Flow (a) first computes the circuit similarity between the original and the modified netlists and uses it to generate an initial placement, which is further refined by a low-temperature annealing process using the VPR placement (initial temperature is set to 0.1 in VPR). As stated in Section 3, based on different pruning settings and annealing parameters, we develop two versions of circuit similarity. A high-quality version, CS, uses $\beta = 0.5, B_l = B_r = 1$ and $inner\_num = 1$[2]. A turbo version, CS-t, uses $\beta = 1, B_l = B_r = 0$ and $inner\_num = 0.1$. Both CS and CS-t are evaluated in our experiments.

Our proposed circuit similarity algorithm is implemented in C and evaluated on the 20 largest MCNC benchmarks. All results are collected averaged over five runs and benchmarked on a Linux server with dual-core 2.19GHz CPU and 5GB memory. The CS2 package [8] is used to solve the min-cost network flow for the maximum matching problem. Table 5 shows the characteristics of the logic-level netlist

[2]A parameter in VPR which controls the number of moves at each temperature.

before (column “original”) and after “rwsat2” optimization (column “rwsat2”). CIs (COs) include the PIs (POs) and register outputs (inputs).

Table 5: Characteristics of the original and “rwsat2”-modified netlists

			CLB#	
Circuit	CI#	CO#	original	rwsat2
alu4	14	8	719	691
apex2	38	3	963	914
apex4	9	19	788	771
bigkey	452	421	1261	1261
clma	94	115	4193	4063
des	256	245	1232	1215
diffeq	332	308	674	665
dsip	452	421	1554	1553
elliptic	196	196	441	439
ex1010	10	10	1103	851
ex5p	8	63	541	515
frisc	905	1002	2844	2320
misex3	14	14	735	629
pdc	16	40	2211	1969
s298	17	20	45	39
s38417	1490	1568	3161	3122
s38584	1297	1564	3723	3646
seq	41	35	997	932
spla	16	46	2126	1935
tseng	435	507	941	883

4.2 Experimental Results

Quality of the initial placement. Table 6 compares the initial placement generated by the proposed circuit similarity (column “CS” and “CS-t”) and the one generated by VPR (a random initial placement) in terms of bb_cost (bounding box cost) and delay cost, two key measures of the placement process. Clearly, both CS and CS-t produce the initial placement with a much better quality than VPR’s, *e.g.*, compared to VPR’s initial results, CS reduces the bb_cost and delay cost by 40% and 31%, respectively. This result shows that the topological node correspondence extracted by the circuit similarity algorithm indeed discovers the intrinsic connection between the original and modified logic-level netlists, and thus provides a reliable guidance to generate the placement for the modified netlist.

Quality of the final placement. Table 7 compares the final placement results produced by flow (a) (including CS and CS-t) and flow (b) shown in Figure 5. Final bb_cost, final delay cost and estimated critical delay are compared between the circuits produced by the two versions of flow (a) and flow (b). As shown in Table 7, for bb_cost and delay cost, both versions of IDUCS produce quality very close to the results produced by *from-scratch* flow. For critical delay, CS and CS-t reduce it by 4% and 1%, respectively compared to *from-scratch* flow. The comparison between CS and CS-t shows the effectiveness of the proposed pruning techniques (in Section 3.3). CS-t, geared with aggressive pruning and significantly lower annealing effort, still produces placement with comparable quality to CS and VPR.

Runtime comparison. Table 7 also compares the runtime of the placement (column “Placement runtime (s)”) of different flows. Note that a timeout is invoked if IDUCS takes longer than the original netlist. It shows that CS-t achieves 28X speedup on average (up to 93X), compared

Table 6: Comparisons of initial solutions for different CAD flows

	initial bb cost			initial delay cost		
Circuit	CS	CS-t	VPR	CS	CS-t	VPR
alu4	145	155	212	2.70E-05	2.95E-05	3.55E-05
apex2	213	249	341	3.38E-05	4.24E-05	5.25E-05
apex4	231	222	259	3.49E-05	3.66E-05	4.09E-05
bigkey	1294	1575	2004	2.23E-04	2.39E-04	2.49E-04
clma	1737	1615	3020	3.36E-04	3.28E-04	4.62E-04
des	306	798	1087	5.93E-05	1.13E-04	1.40E-04
diffeq	818	901	922	9.52E-05	1.04E-04	1.03E-04
dsip	448	578	2184	1.64E-04	1.75E-04	3.30E-04
elliptic	149	244	338	1.73E-05	2.56E-05	3.72E-05
ex1010	277	259	297	3.95E-05	3.99E-05	4.30E-05
ex5p	118	134	156	1.82E-05	2.03E-05	2.31E-05
frisc	5374	6530	8696	7.57E-04	8.49E-04	1.07E-03
misex3	129	162	192	2.36E-05	2.83E-05	3.23E-05
pdc	860	789	1015	1.23E-04	1.31E-04	1.55E-04
s298	3	4	5	5.50E-07	6.06E-07	7.27E-07
s38417	9254	14905	19358	1.07E-03	1.74E-03	2.15E-03
s38584	11948	20781	19983	1.39E-03	2.37E-03	2.33E-03
seq	246	297	355	3.77E-05	4.72E-05	5.47E-05
spla	799	765	986	1.13E-04	1.27E-04	1.50E-04
tseng	801	1323	1736	1.17E-04	1.56E-04	2.00E-04
geomean	472	585	785	7.41E-05	8.96E-05	1.08E-04
ratio	60%	75%	1	69%	83%	1

with the *from-scratch* VPR placement. Due to space limitations, a detailed breakdown of the runtime for each circuit is not shown. Since computing the similarity between two circuits is much faster than re-placing them from scratch, more speedup is expected when applying IDUCS to larger circuits. In practice, one can use CS-t as a quick estimation of the solution quality for an iteration during the incremental design. If the quality is within a satisfied range, the VPR placement can be performed for a better quality.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented IDUCS, an enhancement to the incremental FPGA design flow using circuit similarity. The engineering effort from the previous design iterations is captured by the proposed circuit similarity detection algorithm. Using placement as a case study, we experimentally demonstrate the effectiveness of the proposed IDUCS. Compared with VPR placement in a two-pass design process, our IDUCS-based placement is 28X (up to 93X) faster while preserving the wire length and delay. The speedup is achieved because of the high-quality initial placement generated based on circuit similarity.

In the future, we will integrate the predefined matchings (*e.g.*, the naming matching) into our IDUCS to further enhance both the efficiency and the quality of the design. In addition, we will study the effectiveness of applying our IDUCS to the routing and verification for FPGAs.

6. REFERENCES

[1] A. Ling, S. Brown and J. Zhu. Towards Automated ECOs in FPGAs. *Intl. Symposium on Physical Design*, 2000.

[2] A. Mishchenko, R. Brayton, J.-H. R. Jiang, and S. Jang. SAT-based Logic Optimization and Resynthesis. *International Workshop on Logic and Synthesis*, 2007.

[3] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton. FRAIGs: A Unifying Representation for Logic Synthesis and Verification. *Tech. Report, EECS Dept., UC Berkeley*, 2005.

Table 7: Comparisons of final placement results for different CAD flows. The '*' marked time is measured with a timeout

	Final bb cost			Final delay cost			Estimated critical delay (s)			Placement runtime (s)		
Circuit	CS	CS-t	VPR	CS	CS-t	VPR	CS	CS-t	VPR	CS	CS-t	VPR
alu4	92	111	91	2.34E-05	2.48E-05	2.24E-05	5.61E-08	6.27E-08	5.88E-08	6.89 (1 x)	0.36 (28 x)	9.97
apex2	137	157	136	2.83E-05	3.00E-05	2.77E-05	6.77E-08	7.30E-08	6.45E-08	8.65 (2 x)	0.49 (31 x)	15.01
apex4	135	148	131	2.71E-05	2.87E-05	2.70E-05	6.27E-08	6.50E-08	6.27E-08	*4.16 (3 x)	0.54 (22 x)	12.03
bigkey	262	401	272	1.13E-04	1.11E-04	1.11E-04	1.32E-07	1.29E-07	1.59E-07	35.33 (1 x)	1.74 (25 x)	43.18
clma	766	956	690	2.22E-04	2.38E-04	1.93E-04	1.85E-07	1.93E-07	1.29E-07	*43.64 (3 x)	5.85 (24 x)	139.37
des	237	250	221	5.08E-05	5.23E-05	4.98E-05	8.28E-08	8.04E-08	1.01E-07	13.38 (2 x)	1.28 (22 x)	28.42
diffeq	224	237	216	3.79E-05	3.91E-05	3.68E-05	1.17E-07	1.13E-07	1.38E-07	3.11 (6 x)	0.52 (36 x)	18.73
dsip	468	518	426	1.57E-04	1.57E-04	1.65E-04	1.26E-07	1.22E-07	1.48E-07	*11.58 (4 x)	5.21 (10 x)	52.09
elliptic	91	96	84	1.41E-05	1.45E-05	1.42E-05	8.05E-08	8.73E-08	8.98E-08	3.54 (2 x)	0.24 (34 x)	8.11
ex1010	147	167	142	2.71E-05	2.91E-05	2.78E-05	6.80E-08	7.51E-08	7.22E-08	*8.38 (2 x)	0.49 (32 x)	15.56
ex5p	80	88	76	1.45E-05	1.54E-05	1.42E-05	5.06E-08	5.47E-08	5.54E-08	*2.44 (3 x)	*1.84 (5 x)	8.53
frisc	2308	2376	2052	4.41E-04	4.39E-04	4.37E-04	3.13E-07	3.19E-07	3.16E-07	22.17 (8 x)	4.23 (44 x)	185.37
misex3	87	97	81	1.98E-05	2.06E-05	1.84E-05	5.19E-08	6.27E-08	5.48E-08	6.30 (2 x)	0.25 (40 x)	10.02
pdc	416	480	402	8.80E-05	9.05E-05	8.56E-05	9.70E-08	1.06E-07	1.03E-07	*19.67 (3 x)	2.70 (21 x)	56.97
s298	3	3	3	5.34E-07	5.40E-07	5.05E-07	1.78E-08	1.78E-08	1.61E-08	0.05 (5 x)	0.01 (24 x)	0.24
s38417	1091	1287	978	3.14E-04	3.25E-04	3.68E-04	3.65E-07	3.65E-07	3.83E-07	38.66 (11 x)	5.05 (84 x)	426.19
s38584	1401	1685	1352	5.25E-04	5.49E-04	4.95E-04	4.37E-07	4.13E-07	5.09E-07	29.23 (14 x)	4.52 (93 x)	421.80
seq	156	174	153	3.08E-05	3.25E-05	3.04E-05	5.53E-08	6.12E-08	5.40E-08	14.30 (1 x)	0.51 (37 x)	18.75
spla	396	481	389	7.83E-05	8.78E-05	8.18E-05	1.02E-07	1.11E-07	9.25E-08	*17.21 (3 x)	2.21 (24 x)	53.04
tseng	276	298	272	7.86E-05	7.98E-05	7.73E-05	1.46E-07	1.40E-07	1.67E-07	5.99 (6 x)	0.82 (43 x)	35.02
geomean	218	249	208	5.00E-05	5.20E-05	4.94E-05	9.84E-08	1.02E-07	1.03E-07	8.40 (3 x)	0.96 (28 x)	27.26
ratio	105%	120%	1	101%	105%	1	96%	99%	1	31%	4%	1

[4] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. `http://www.eecs.berkeley.edu/~alanmi/abc/`, Release 70930.

[5] H. Bunke. Error Correcting Graph Matching: On the Influence of the Underlying Cost Function. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21:917–922, 1999.

[6] D. Zacher. Turbo-Charge Your FPGA Design Iterations. `http://chipdesignmag.com/display.php?articleId=3497`, 2007.

[7] G. Jeh and J. Widom. A Measure of Structural-context Similarity. *Proceedings of the Eighth Intl. Conference on Knowledge Discovery and Data Mining*, 2002.

[8] A. V. Goldberg. An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm. *Journal of Algorithms*, 22:1–29, 1997.

[9] J. Cong and M. Sarrafzadeh. Incremental Physical Design. *International Symposium on Physical Design*, 2000.

[10] J. Cong and W. Jiang. Pattern-based Behavior Synthesis for FPGA Resource Reduction. *Proc. 16th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2008.

[11] K. Chang, D. A. Papa, I. L. Markov and V. Bertacco. InVerS: An Incremental Verification System with Circuit Similarity Metrics and Error Visualization. *International Symposium on Quality Electronic Design*, 2007.

[12] K. Chang, I. L. Markove and V. Bertacco. Fixing Design Errors with Counterexamples and Resynthesis. *IEEE Journal on Technology in Computer-Aided Design*, 27(1):184–188, 2008.

[13] M. L. Fernandez and G. Valiente. A Graph Distance Metric Combining Maximum Common Subgraph and Minimum Common Supergraph. *Pattern Recognition Letters*, 22:735–758, 2001.

[14] M. Rupp, E. Proschak and G. Schneider. Kernel Approach to Molecular Similarity Based on Iterative Graph Similarity. *Journal of Chemical Information and Modeling*, 47:2280–2286, 2007.

[15] M. Pelillo. Matching Free Trees, Maximal Cliques and Monotone Game Dynamics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:1535–1541, 2002.

[16] R. Albert and A. L. Barabasi. Statistical Mechanics of Complex Networks. *Reviews of Modern Physics*, 74:47–97, 2002.

[17] S. Krishnaswamy, H. Ren, N. Modi, R. Puri. DeltaSyn: An Efficient Logic Difference Optimizer for ECO Synthesis. *International Conference on Computer Aided Design*, 2009.

[18] S. Melnik, H. Garcia-Molina and A. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. *Proceedings of the 18th International Conference on Data Engineering*, 2002.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to Algorithms.* MIT Press, Cambridge, MA, USA, 2001.

[20] V. Betz and J. Rose. VPR: A New Packing, Placement and Routing Tool for FPGA Research. *International Workshop on Field Programmable Logic and Applications*, 1997.

[21] V. Blondel, A. Gajardo, M. Heymans, P. Senellart and P. Van Dooren. A Measure of Similarity between Graph Vertices: Applications to Synonym Extraction and Web Searching. *Society for Industrial and Applied Mathematics Review*, 46(4):647–666, 2004.

[22] Xilinx Corporation. SmartCompile Technology: SmartGuide. *Xilinx Press Release*, 2008.

[23] Y. S. Yang, S. Sinha, A. Veneris and R. K. Brayton. Automating Logic Rectification by Approximate SPFDs. *Asia-South Pacific Design Automation Conference*, 2007.

[24] S. Yang. Logic Synthesis and Optimization Benchmarks User Guide, Version 3.0. *Technical Report, Microelectronics Center of North Carolina*, 1991.

[25] Z. Gu, J. Wang, R. P. Dick, and H. Zhou. Unified Incremental Physical-Level and High-Level Synthesis. *IEEE Transactions on Computer-Aided Design*, 26:1576–1588, 2007.

[26] L. Zager. *Graph Similarity and Matching.* PhD thesis, MIT, 2005.