Perturber: Semi-Synthetic Circuit Generation Using Ancestor Control for Testing Incremental Place and Route

David Grant and Guy Lemieux

University of British Columbia
Vancouver, BC, Canada
`[davidg,lemieux]@ece.ubc.ca`

***Abstract*— FPGA architects are always searching for more benchmark circuits to stress CAD tools and device architectures. In this paper we present a new heuristic to generate benchmark circuits specifically for incremental place and route tools. The method removes part of a real circuit and replaces it with a modified version of the same circuit to mimic an incremental design change. The generation procedure exactly preserves key circuit characteristics and achieves a post–routing channel width, critical path, and wire length that closely approximates those of the original circuit. Additionally, the method is fast and thus is suitable for use in on–the–fly benchmark generation.**

I. Introduction

Incremental design changes arise for a number of reasons including debug changes, iterative design improvement, and physical resynthesis to meet timing closure. Since incremental design changes are highly iterative in nature, the run–time and quality of solutions produced by incremental place and route tools are extremely important. To test the incremental modes offered by CAD tools, *incremental circuits* are needed. These circuits should behave like real circuits, except that a large number of variations are needed to mimic the process of small, incremental design changes. To our knowledge, no incremental benchmark circuits exist to test FPGA tools. Good FPGA benchmark circuits are difficult enough to obtain by themselves – gathering incremental changes that represent the evolution of the circuit is even more challenging.

Due to the difficulty of obtaining benchmark circuits, a number of methods have been developed to create *synthetic circuits*. These synthetic circuits are crafted in a way to have similar properties to real circuits. These methods include stochastic generation using just a few parameters (e.g., `gnl` [1]), stochastic generation of clones based upon detailed characterization of real circuits (e.g., `ccirc+cgen` [2], [3]), and stochastic stitching together of real designs as subcircuits of a larger design [4], [5]. Ultimately, these generators are concerned with creating an entire benchmark circuit, making them unsuitable for creating incremental circuits where large parts of the circuit must remain the same.

This research has been enabled by the use of WestGrid computing resources, which are funded in part by the Canada Foundation for Innovation, Alberta Innovation and Science, BC Advanced Education, and the participating research institutions. WestGrid equipment is provided by IBM, Hewlett Packard and SGI.

This paper describes a novel and simple new approach for creating small, incremental, changes to a circuit. The approach takes an original circuit (either real or synthetic) N, identifies a sub-circuit S, removes it to produce $N\backslash S$, and inserts a replacement R into the hole left by the removal of S. For generating benchmarks that contain incremental changes, a large portion of the original circuit must remain the same, which is why we say S is replaced with R, leaving the remainder of the original circuit untouched. This terminology will be used throughout this paper.

In [6], a method of using existing synthetic circuit generators to generate R from S and also to "stitch" R into $N\backslash S$ was presented. It was shown that care is required in the stitching process to avoid the inadvertent creation of combinational loops in the overall circuit. The stitching method in [6] transformed the problem into a graph matching problem which was solved using existing tools. However, the problem is combinatorially complex, and required the use of imprecise heuristics with unpredictable run times to produce a stitched solution. The circuits produced were characteristically similar to the original except for elongation of the critical path which would require further constraints to be added to the graph matching problem and possibly additional run–time to solve correctly.

To perform an incremental place and route, a CAD tool requires both the original circuit and the incremental change. Instead of capturing various properties of S and using those to generate R, as is done in existing synthetic circuit generators, we propose a new method to create R by using S directly. After S is removed from the circuit, some of the edges (wires) in S can be perturbed (that is, swapped with other edges in S) to create a new circuit, R. If the perturbations are done under specific restrictions, R will contain many characteristics which are identical to S, and will remain loop-free. Further, the input and output nets in S and R will be identical, which trivializes the process of "stitching" R into $N\backslash S$.

The remainder of this paper is organized as follows. Section II presents the actual method used to "perturb" a circuit, and also presents Ancestor Control, a modification to the technique to better match the post–routing results to those of the original circuit. Section III describes the experiments done with the Perturber and the results of those experiments. Section IV presents several techniques which were unsuccessful at controlling the post–routing results. Directions for future work specific to the Perturber are presented in Section V, and conclusions are given in Section VI.

II. Perturbing a Circuit

This section describes a procedure we have called "perturbing" a circuit. It allows the rapid creation of incremental changes to a circuit given the original. The procedure takes a circuit which can be a complete circuit, or in our case part of a complete circuit (S), and generates a modified circuit, in this case R, which we then stitch back into the original circuit ($N \backslash S$).

The perturbation method introduced in this paper has several attractive features for generating new circuits that make them good synthetic approximations to the original circuit. That is, the Perturber preserves many of the distinguishing characteristics of the original circuit. These are:

- **The number of nodes and edges are preserved** – The procedure does not create or delete nodes or edges, it only moves them around.
- **The fanout distribution is identical** – The procedure ensures the fanout of each net is identical to the original circuit.
- **The depth profile is preserved** – Nodes are not permitted to be moved to a new logic depth in the circuit. Edges are moved between nodes under a specific set of rules that ensures the depth of any node in the circuit is left unchanged.
- **No combinational loops are created** – Since the depth of each node is maintained, the original levelization (we describe what we mean by the levelization of a circuit below) of the circuit is still valid, thus there cannot be any combinational loops introduced into the circuit.
- **Stitching R back into $N \backslash S$ is trivialized** – The input and output nodes in S and R are identical (as are all the internal nodes), so the stitching procedure is reduced to matching up node names and copying the fanout information for each node. This makes it extremely useful for generating incremental benchmarks on–the–fly.

In many cases, it may not be desirable to identically preserve all the aforementioned characteristics. An incremental user change, for example, is likely to slightly change several characteristics of the circuit. However, we would also like to have the ability to hold some (or all) circuit characteristics constant when generating benchmarks. For this reason, we proceed with a perturbation procedure that maintains all the above characteristics. Section V, Future Work, presents some additions to the Perturber which would allow these characteristics to fluctuate.

A. *Perturbation Procedure*

The Perturber starts with knowledge of the complete circuit N and $S \in N$ which has been identified in N but not removed yet. N is first "levelized" so that an overall level of each node in S can be assigned. The process of "levelizing" a circuit refers to computing the maximum depth through combinational logic of each LUT in the circuit. Latches, inputs, and constant drivers are assigned a depth of 1. Assuming the circuit contains no combinational loops, a simple traversal starting at all level 1 nodes, and feeding the maximum input level of each LUT forward, will correctly assign the maximum logic depth (the level), to each LUT in the circuit. There is an important distinction between levelizing N and levelizing S. In this case the complete circuit, N, is levelized and the level numbers are copied to S, so S may contain "inputs" which are not at level 1.

When the level of each node in S is known, a list containing all the edges between any two given levels in S can be created. The perturbation method proceeds by randomly selecting an edge out of this list and swapping with the sink of a second edge in the list under the following conditions:

1) The source and sink levels of both edges to be swapped must match. Swapping edges with mismatched source or sink levels may be valid, but would necessitate a recomputation of the levels of all nodes in the fanout cone of the edges swapped. The lists of edges between each pair of levels in the circuit would also need to be updated or rebuilt. For large circuits over a large number of edge swaps, this can significantly increase the time to perturb the circuit. Ensuring the source and sink level match also preserves the fanout distribution and depth profile of the circuit.
2) The source node can not be level 1. A level 1 source node is either an input to the original circuit or the direct output of a latch. In either case, we allow inputs and latch outputs to proceed through one level of logic to reduce the probability of directing the signal to a completely different branch of logic in the circuit.

Only the edges are considered for swapping. The nodes (LUTs and latches) in the circuit are untouched in the sense that they do not need to be moved around. An incremental user change to a circuit may modify the contents of a LUT, but in terms placement and routing, the actual contents of the LUT are irrelevant.

The perturber takes a single parameter, the perturbation factor, which is the percentage of the edges in S to swap. All the results in this paper used a perturbation factor of 25% [1], meaning that the Perturber continues until it has randomly selected 25% of the edges in the circuit and swapped each edge with a random edge chosen under the criteria listed above. For all the results presented in this paper we have verified

[1]Several tests were also run using perturbation factors of 12% and 50%. The post–routing results were not significantly different than the results at 25%. At 50% the Perturber required disproportionately more time to generate R since more of the swaps returned an edge to its original position, and the perturber continues until the required number of edges are in a new location.

that swaps which return an edge to its original position are infrequent, so the impact of such swaps are negligible.

When the Perturber is finished with S, it writes a new circuit, R, which is then merged or stitched into the hole left by the removal of S from N. This stitching process simply matches the names of the nets in R with those which were cut when S was removed from N. The nodes and edges in R are copied into $N \backslash S$ and the fanins/fanouts for matching input/output edges in R are reconnected, creating a complete circuit.

By only swapping edges with matching source and sink levels, the level of any node in the circuit does not change. Thus, the original levelization is still valid after any number of edge swaps. Because of this, we can be sure that we have not introduced any combinational loops in the circuit even after R is stitched into $N \backslash S$.

Initial tests using this method showed promising results for small areas cut out of N and then replaced. However, under larger areas, the placement and routing results became unfavourable when compared to existing circuit generators. The placement and routing results for this initial method were unacceptable. The problem was discovered to be (largely) that the locality of the edges were not taken into account when swapping, causing R to be very irregular (like a random circuit actually), and consequently, very difficult to route. The next section presents additional restrictions placed on the Perturber with the goal of preserving the placement and routing results.

B. Static Ancestor Depth Control

The major problem with the Perturber was that there is no concept of locality during the edge swapping. For example, two buses that are from completely independent parts of the circuit could be easily "cross–connected". This leads to destroying the "nice" regular features of the circuit, such as buses and independent sequences of logic, thus increasing the irregularity of the circuit and making it harder to route.

To control the locality of the perturbations done to the circuit, we restrict the Perturber to only swap edges within related chains of logic. To do this, an additional restriction on the edge swapping criteria is added:

3) Both edges to be swapped must share a common ancestor through combinational logic within a certain ancestor depth, called the "ancestor control depth".

The ancestor control depth is specified as a single number, d. When an edge in the circuit is selected, the edge has a *selected source* and a *selected sink*. A list of candidate edges for the sink swap is computed dynamically by first finding all the ancestors within d levels of the *selected source*, then walking forward through all fanouts from each ancestor back to the original depth. For each LUT visited, if the LUT level matches the level of the *selected source*, and the LUT contains a sink that matches the level of the *selected sink*, then the LUT contains an edge that may be swapped with the selected edge.

Figure 1 shows part of an example network which can be perturbed. If the net between nodes 10 and 14 is selected by the perturber, the *selected source* is node 10 at level 4, and the

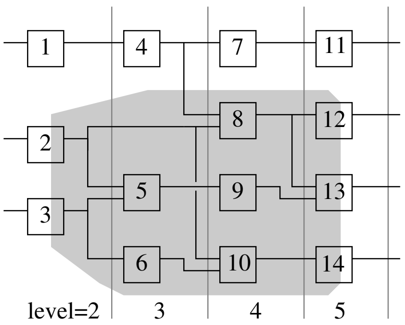

Fig. 1. Ancestor selection region (shaded) for the net connecting nodes 10 and 14.

selected sink is node 14 at level 5. If, for example, the ancestor control depth was set to $d = 2$, the list of candidate swap edges would be built by walking backwards from node 10, visiting nodes 6, 5, 3, and 2. Nodes 2 and 3 are two levels away from the starting node, so the backtrack stops and begins to follow all forward paths until each path reaches the original depth of 4. The procedure would find nodes 8 and 9 in addition to the nodes identified by the backtrack. The final step is to evaluate all the outputs of all the identified nodes and add edges to the candidate list which meet the criteria identified in Section II-A. In this example, the candidate net list would include the edges between the following pairs of nodes: $(8, 12), (8, 13), (9, 13)$. Of these edges, one would be randomly selected, say $(8, 13)$, and the sinks would be swapped. The final circuit would thus contain an edge between nodes $(8, 14)$ and $(10, 13)$. The original perturbation procedure would have also included the edge between nodes $(7, 11)$ in the candidate list, but the ancestor control method excludes it.

In the next section, we show that this method of ancestor depth control can produce a perturbed circuit with post–routing results which are similar to those of the original circuit.

III. Experimental Results

In this section we present the results of two experiments. The first tests the feasibility of using the Perturber with and without ancestor control to generate complete circuits (synthetic clones) from an original circuit. This is something that would never be done when testing incremental place and route tools but it shows that the Perturber generates very effective clones using simpler heuristics than `ccirc+cgen`. The second experiment tests the Perturber with ancestor control on three different cutout region sizes to assess how well the post–routing characteristics of the input MCNC circuits are mimicked. As mentioned in Section II, the Perturber preserves key characteristics of the circuit, such as the number of nodes, number of edges, fanout distributions, and depth profile. Therefore, only the change in post–routing results need to be examined in these experiments.

Table I shows the placement and routing results for the 20 largest MCNC benchmarks using `vpr`. The **CW** column is the minimum channel width required to route the circuit. The

TABLE I

PLACEMENT AND ROUTING RESULTS FOR MCNC CIRCUITS.

Name	Nodes	Edges	CW	CP(ns)	WL
alu4	1522	2533	33	12.0	9301
apex2	1878	4073	47	13.1	15794
apex4	1262	3626	49	12.1	11085
bigkey	1931	4945	46	6.0	9097
clma	8414	24958	67	24.4	83587
des	1591	3679	58	10.7	11050
diffeq	1871	5069	34	15.8	8979
dsip	1594	3383	42	5.9	6962
elliptic	4724	12037	55	20.4	30388
ex1010	4598	15643	58	16.6	42961
ex5p	1064	3218	49	12.4	9525
frisc	4425	12730	54	26.8	30152
misex3	1397	3137	42	11.4	10164
pdc	4575	15654	67	25.8	49485
s298	1938	5806	28	21.2	9130
s38417	7559	22294	41	15.6	38076
s38584.1	7541	18641	43	12.8	40122
seq	1750	3807	45	14.9	14418
spla	3690	12658	58	15.5	33871
tseng	1431	3577	39	14.7	6689
		Average:	47.75	15.41	23541.8

CP column is the critical path, in nanoseconds, of routing the circuit using a channel width 20% larger than the value reported in the **CW** column. The **WL** column is the total wire length of the final routed circuit. These numbers are used as the baseline comparison for the two experiments presented in this section, with results in Tables II and III.

The first experiment compares the routing results of `ccirc+cgen` clones with that of the Perturber both with and without ancestor control. For these tests the perturber was set to operate on the entire circuit. Each test was performed 10 times, and the average results are given in Table II. All of the data is of the form *average percentage* $\pm$ *standard deviation percentage*, where both numbers are the percentage of the original MCNC result, given in Table I. For example, in Table II, the first row (alu4) and the first column of data (**ccirc+cgen**, **CW %**) contains the data -0.9 ± 4.3. This means that the average channel width of the 10 **ccirc+cgen** alu4 clones was 0.9% lower than the channel width of the original alu4 circuit. From Table I the original channel width of alu4 was 33, so the channel width being reported is actually $33-(33*0.009) = 32.7$. The standard deviation is reported as 4.3%, meaning the actual standard deviation is $33*0.043 = 1.4$ tracks. We present the results using percentages to allow the quality of the results to be compared between the various circuits. We also present the worst case and average results for all 20 MCNC circuits for each test.

When comparing the Perturber to `ccirc+cgen` it should be noted that `ccirc+cgen` is designed to capture the qualities of a circuit and generate a complete clone using just these qualities. The perturber takes an existing circuit and modifies it in such a way to preserve key features. The tools are intended for two different tasks, and the Perturber would not normally be run on 100% of the circuit. However, it is important to validate that the Perturber is performing more intelligently than just a random circuit generator, and the results of another synthetic circuit generator are used to make such a comparison.

Unfortunately, the Perturber without the ancestor depth control mechanism is not behaving much better than a random circuit generator. The routed channel width, critical path, and required wire length are all significantly higher than the original circuit, and significantly higher than `ccirc+cgen`, both on average and in the worst case results. These results tell us that it is possible to preserve many key characteristics of the circuit (Section II), but still end up with a circuit that does not behave much like the original.

The final three columns of data in Table II, under the "Perturber with Ancestor Control" heading, show a significant improvement. These results again use a perturbation factor of 25%, and use an ancestor control depth[2] of $d = 3$.

In many cases the Perturber with ancestor depth control was able to produce a circuit with post–routing properties closer to the original than `ccirc+cgen` did. The same is true for the worst case results. We believe this validates our approach taken to create the Perturber with the ancestor control mechanism, and allows us to proceed to use the Perturber to generate circuits with incremental changes.

Circuits useful for testing incremental place and route tools involve changes to a small portion of the circuit, not creating a clone of the entire circuit. The second experiment removes a random portion of the circuit, S, which is perturbed with ancestor control to create R which is then stitched back into the original circuit. The new circuit is then placed and routed with `vpr`. The random cutout was done with three different sizes, 5, 10, or 20% of the nodes of the original circuit. Each size test was performed 8 times on random regions of each circuit, and for all tests a perturbation factor of 25% and an ancestor control depth of 3 was used. As previously mentioned, the Perturber preserves many of the features of the circuit, so we do not present results for any characteristic mentioned in the beginning of Section II.

The results in Table III show that the Perturber is able to create new circuits that have very similar place and route properties as the original. Notice the trend in increasing deviation for all data from the original MCNC benchmarks as the cutout size increases. This increase would eventually reach the values at 100% node cutout, which is actually the last 3 columns of data in Table II, under the "Perturber with Ancestor Control" heading.

It is also interesting to address the speed of the Perturber. There are $3 * 8 * 20 = 480$ circuits which were generated, placed and routed for Table III. On a P4 3GHz, with 1GB of RAM, the process of cutting out S from N, perturbing S into R, and stitching R into $N\backslash S$ was completed in approximately 8 minutes for all 480 circuits (approximately one perturbed circuit per second).

[2]The data in Tables II and III were also generated for an ancestor control depths of 2 and 4. A depth of 2 appeared to be too restrictive for finding candidate edges to swap with, whereas a depth of 4 showed a significant step towards no ancestor control for some circuits. For this reason, we use a control depth of 3 in all experiments.

TABLE II

AVERAGE PLACEMENT AND ROUTING RESULTS OF 10 CCIRC+CGEN CLONES OF EACH MCNC CIRCUIT, 10 CLONES GENERATED BY THE PERTURBER, AND 10 CLONES FROM THE PERTURBER USING ANCESTOR CONTROL TO MAINTAIN LOCALITY.

	ccirc+cgen			**Perturber**			**Perturber with Ancestor Control** ($d = 3$)		
Name	**CW %**	**CP %**	**WL %**	**CW %**	**CP %**	**WL %**	**CW %**	**CP %**	**WL %**
alu4	-0.9 ± 4.3	0.8 ± 6.9	-5.0 ± 2.4	18.2 ± 0.0	-0.2 ± 0.3	11.6 ± 0.4	5.3 ± 3.1	6.0 ± 9.7	0.2 ± 1.9
apex2	7.2 ± 2.3	5.1 ± 1.5	5.6 ± 1.6	30.9 ± 1.5	16.6 ± 12.0	32.8 ± 0.1	9.3 ± 1.9	14.5 ± 13.1	7.3 ± 1.2
apex4	-12.4 ± 2.6	0.0 ± 6.7	-11.9 ± 2.7	8.2 ± 2.9	-3.3 ± 2.6	6.2 ± 0.1	-1.0 ± 1.5	5.5 ± 7.0	-0.3 ± 1.3
bigkey	-5.4 ± 7.1	7.7 ± 4.7	25.3 ± 2.8	8.7 ± 0.0	7.5 ± 1.2	73.0 ± 1.5	11.7 ± 2.6	8.0 ± 7.0	77.3 ± 2.4
clma	57.3 ± 3.4	14.7 ± 7.9	60.7 ± 2.8	88.1 ± 4.2	19.7 ± 3.2	88.3 ± 2.6	47.2 ± 2.2	9.0 ± 2.5	40.6 ± 1.2
des	12.2 ± 6.7	3.8 ± 2.0	20.5 ± 1.7	4.3 ± 1.2	11.7 ± 4.5	64.0 ± 2.0	-7.1 ± 4.7	1.0 ± 2.0	1.2 ± 1.6
diffeq	19.4 ± 4.0	3.9 ± 7.7	18.6 ± 4.7	69.1 ± 2.1	11.0 ± 4.7	70.9 ± 2.7	9.2 ± 3.3	-4.0 ± 5.8	3.8 ± 3.1
dsip	2.9 ± 3.1	6.1 ± 4.8	21.9 ± 3.5	8.3 ± 8.4	4.4 ± 2.7	50.6 ± 0.3	0.3 ± 4.3	4.7 ± 3.3	38.7 ± 2.3
elliptic	14.5 ± 2.4	-5.3 ± 6.1	26.0 ± 1.5	47.3 ± 0.0	-4.7 ± 0.1	53.6 ± 0.0	-0.7 ± 2.2	-0.2 ± 5.9	3.0 ± 3.6
ex1010	12.2 ± 2.4	-1.4 ± 1.7	12.4 ± 2.2	80.2 ± 1.2	10.0 ± 16.5	74.9 ± 0.9	37.1 ± 2.9	2.2 ± 3.4	31.2 ± 1.0
ex5p	-9.6 ± 2.9	5.8 ± 7.6	-8.7 ± 2.6	15.3 ± 1.4	3.3 ± 6.3	13.3 ± 1.2	5.1 ± 1.5	5.9 ± 5.1	4.2 ± 0.9
frisc	27.6 ± 3.3	-1.9 ± 4.1	30.7 ± 3.1	51.9 ± 0.0	16.9 ± 5.8	57.3 ± 1.4	38.0 ± 22.2	7.2 ± 3.3	31.5 ± 2.0
misex3	-1.0 ± 2.0	2.0 ± 3.9	0.5 ± 1.4	23.8 ± 0.0	1.0 ± 2.0	24.6 ± 1.8	3.6 ± 1.8	6.6 ± 9.9	3.6 ± 1.3
pdc	11.2 ± 1.9	-31.1 ± 3.8	12.6 ± 0.8	38.8 ± 2.1	-19.3 ± 17.7	41.3 ± 0.8	17.9 ± 1.1	-19.6 ± 15.4	16.7 ± 0.6
s298	-3.6 ± 5.8	-0.0 ± 4.5	-7.5 ± 3.2	10.7 ± 0.0	3.5 ± 3.3	7.5 ± 4.2	10.7 ± 2.7	18.2 ± 24.2	8.4 ± 3.4
s38417	100.7 ± 3.6	12.1 ± 4.3	115.6 ± 3.0	213.4 ± 1.7	28.6 ± 1.6	234.8 ± 6.4	56.4 ± 2.4	21.5 ± 4.4	46.1 ± 2.5
s38584.1	64.9 ± 3.5	4.4 ± 4.4	73.8 ± 2.1	169.8 ± 6.6	24.6 ± 5.7	187.4 ± 1.3	0.6 ± 4.8	-3.8 ± 3.4	3.4 ± 1.8
seq	1.6 ± 1.8	-19.4 ± 7.9	0.5 ± 1.6	26.7 ± 0.0	-15.0 ± 7.3	22.0 ± 1.0	6.9 ± 2.5	-19.1 ± 1.7	4.5 ± 1.9
spla	12.9 ± 2.0	10.0 ± 18.9	17.5 ± 1.9	44.8 ± 0.0	16.3 ± 15.4	53.0 ± 0.9	19.0 ± 2.3	16.7 ± 21.4	22.2 ± 1.9
tseng	-10.8 ± 3.2	13.3 ± 4.2	2.7 ± 3.3	16.7 ± 1.8	10.5 ± 2.2	30.4 ± 0.5	-0.3 ± 4.4	5.7 ± 4.2	3.9 ± 1.9
Worst Case:	100.7 ± 3.6	-31.1 ± 3.8	115.6 ± 3.0	213.4 ± 1.7	28.6 ± 1.6	234.8 ± 6.4	56.4 ± 2.4	21.5 ± 4.4	77.3 ± 2.4
Absolute Average:	19.4 ± 3.4	7.4 ± 5.7	23.9 ± 2.4	48.7 ± 1.8	11.4 ± 5.8	59.9 ± 1.5	14.4 ± 3.7	9.0 ± 7.6	17.4 ± 1.9

TABLE III

AVERAGE PLACEMENT AND ROUTING RESULTS FOR 8 TRIALS (4 DIFFERENT REGIONS PERTURBED TWICE) OF A 5, 10, AND 20% NODE CUTOUT OF EACH MCNC CIRCUIT. THE NUMBERS REPRESENT THE AVERAGE PERCENTAGE INCREASE (OR DECREASE) FROM THE MCNC RESULTS FROM TABLE I. THE PERTURBER IS APPLIED ONLY TO THE CUTOUT REGION USING A PERTURBATION FACTOR OF 25% AND AN ANCESTOR CONTROL DEPTH OF 3. NOTE THAT THE DATA UNDER THE "PERTURBER WITH ANCESTOR CONTROL" HEADING IN TABLE II WOULD REPRESENT A 100% NODE CUTOUT IN THIS TABLE.

	5% Node Cutout			**10% Node Cutout**			**20% Node Cutout**		
Name	**CW %**	**CP %**	**WL %**	**CW %**	**CP %**	**WL %**	**CW %**	**CP %**	**WL %**
alu4	4.2 ± 3.9	2.6 ± 7.4	-0.3 ± 1.4	5.7 ± 4.4	5.3 ± 6.4	-0.7 ± 1.2	4.5 ± 2.3	8.6 ± 9.9	0.8 ± 1.7
apex2	1.1 ± 2.5	4.2 ± 1.9	-0.3 ± 2.0	0.0 ± 2.0	12.4 ± 24.5	-0.1 ± 1.2	1.6 ± 2.2	4.4 ± 2.8	0.2 ± 1.7
apex4	0.0 ± 1.1	3.9 ± 4.4	-0.4 ± 1.3	0.5 ± 1.8	15.4 ± 15.0	-0.6 ± 1.2	1.5 ± 3.0	12.2 ± 22.5	1.1 ± 3.3
bigkey	10.1 ± 6.8	-1.9 ± 1.1	0.9 ± 2.0	8.7 ± 6.0	-0.3 ± 2.8	0.0 ± 2.0	12.5 ± 7.4	1.3 ± 3.1	0.1 ± 2.9
clma	-4.7 ± 1.2	-2.0 ± 2.7	-1.4 ± 1.2	-1.3 ± 2.0	-0.8 ± 4.4	1.1 ± 1.4	6.2 ± 2.2	0.5 ± 2.4	5.9 ± 1.1
des	-11.4 ± 6.0	3.3 ± 2.6	-0.9 ± 1.7	-7.3 ± 6.9	4.7 ± 3.6	-0.7 ± 1.4	-8.4 ± 5.9	1.3 ± 2.0	0.2 ± 2.6
diffeq	-1.8 ± 2.7	-2.1 ± 4.1	-6.1 ± 1.4	1.1 ± 2.7	-2.5 ± 3.1	-4.6 ± 2.6	4.4 ± 5.2	-3.8 ± 3.2	0.0 ± 1.6
dsip	1.2 ± 5.4	1.1 ± 2.6	0.3 ± 3.0	3.3 ± 10.6	0.8 ± 1.3	-0.9 ± 2.2	3.6 ± 7.0	-0.1 ± 1.2	0.5 ± 1.3
elliptic	2.0 ± 3.4	-3.4 ± 2.7	0.6 ± 0.6	0.2 ± 1.5	-2.1 ± 5.5	0.9 ± 1.0	2.7 ± 1.4	7.9 ± 30.3	1.4 ± 1.2
ex1010	2.6 ± 3.9	1.0 ± 4.4	1.0 ± 2.4	10.1 ± 3.3	5.0 ± 6.8	7.1 ± 2.2	18.8 ± 6.5	3.8 ± 5.7	14.0 ± 4.5
ex5p	0.5 ± 2.1	5.3 ± 6.9	0.7 ± 1.3	1.3 ± 1.5	9.4 ± 16.6	2.5 ± 1.7	4.3 ± 1.7	14.1 ± 22.7	3.6 ± 1.1
frisc	3.0 ± 2.8	-1.4 ± 2.7	2.6 ± 2.1	5.1 ± 1.9	0.2 ± 2.6	3.2 ± 1.2	3.7 ± 1.7	1.4 ± 4.4	3.6 ± 1.2
misex3	-0.6 ± 2.8	11.9 ± 15.0	-0.1 ± 2.4	1.8 ± 2.5	3.1 ± 9.4	0.9 ± 1.1	3.9 ± 1.8	4.3 ± 7.1	3.5 ± 2.5
pdc	0.4 ± 1.3	-27.7 ± 5.9	0.2 ± 0.7	1.3 ± 1.7	-21.2 ± 11.2	1.1 ± 0.9	5.0 ± 1.9	-28.4 ± 4.5	3.4 ± 0.6
s298	5.4 ± 3.8	10.2 ± 4.2	4.4 ± 2.7	7.1 ± 3.8	9.5 ± 6.2	4.9 ± 2.3	9.8 ± 3.2	7.4 ± 7.0	6.4 ± 2.7
s38417	-0.3 ± 1.6	1.7 ± 2.5	1.3 ± 1.2	3.0 ± 2.8	8.0 ± 3.7	3.5 ± 2.5	21.3 ± 3.6	11.2 ± 8.3	14.8 ± 1.8
s38584.1	1.2 ± 2.8	1.5 ± 2.2	3.0 ± 1.5	2.6 ± 5.0	0.8 ± 1.5	3.5 ± 1.8	10.5 ± 5.1	1.7 ± 2.7	7.7 ± 2.2
seq	0.3 ± 2.8	-21.6 ± 1.8	0.1 ± 1.1	1.9 ± 2.8	-13.7 ± 13.5	2.2 ± 1.7	4.2 ± 3.0	-19.1 ± 6.5	3.7 ± 2.2
spla	-0.6 ± 2.0	3.2 ± 4.3	1.8 ± 1.3	-0.2 ± 1.9	10.7 ± 14.7	3.2 ± 0.8	4.7 ± 1.2	39.2 ± 58.8	6.0 ± 0.7
tseng	0.6 ± 4.1	-0.2 ± 3.8	-1.2 ± 1.5	-1.9 ± 5.8	2.2 ± 3.8	-1.5 ± 2.2	2.9 ± 5.0	4.0 ± 3.7	0.1 ± 3.4
Worst Case:	-11.4 ± 6.0	-27.7 ± 5.9	-6.1 ± 1.4	10.1 ± 3.3	-21.2 ± 11.2	7.1 ± 2.2	21.3 ± 3.6	39.2 ± 58.8	14.8 ± 1.8
Absolute Average:	2.6 ± 3.2	5.5 ± 4.2	1.4 ± 1.6	3.2 ± 3.6	6.4 ± 7.8	2.2 ± 1.6	6.7 ± 3.6	8.7 ± 10.4	3.9 ± 2.0

IV. PITFALLS

In this section, several unsuccessful methods of controlling the post–routing results are presented. We include these for the sake of completeness and because they provided valuable insight into the behaviour of the Perturber. This insight lead to the development of the ancestor control technique.

A. Wirelength Control

In an effort to control the channel width and the total routed wire length in the circuit, a method to limit the wire length during the perturbation procedure was implemented. The method uses the total Manhattan distance of all the edges in the circuit to approximate the total wire length in the circuit.

During perturbation, if an edge swap lowers the total area, the swap is accepted. If the wire length increases, the move is probabilistically accepted using an exponential function similar to that used in simulated annealing.

The location of each node in the circuit is taken from a placement of the original MCNC circuit. After the perturbation procedure is complete, the entire circuit is re-placed and re-routed. When considering using incremental place and route tools, it is reasonable to assume we have a previous placement of the circuit, so we can use that placement to drive this technique.

The results from experiments using this method showed very little difference over the original perturbation technique (with no ancestor control), hence it was ineffective.

B. Bounding Box Control

Instead of limiting the wire length by using a Manhattan metric, the bounding box can also be used. The bounding box of a net is the half perimeter of the smallest box bounding all the fanouts of a net. Additionally, the bounding box would prevent random edge swaps over large distances. Consider Figure 1 where $(10, 14)$ is again the selected edge. Suppose the $(7, 11)$ edge is separated from the $(10, 14)$ edge by a large distance. A bounding box limitation would prevent the edges from being swapped.

Here, we again follow a procedure similar to that used in simulated annealing to always accept moves that lower the total cost, and probabilistically accepts ones that do not. The results from this technique marginally improved the output, approximately a 1%-2% step closer to the original results, but it was still ineffective overall.

C. Net Swapping

Instead of swapping individual sinks from edges, a method of potentially preserving locality is to swap the entire net that an edge is in. In other words, swap the source of entire nets. This method, combined with the bounding box, aimed to prevent nets from fanning out to all regions of the FPGA when it was routed.

This method further improved the results over the bounding box by an additional 1%, however, the results were not statistically significant, or close to the original post–routed results from the MCNC circuits. Only the ancestor depth control described in Section II-B was effective at capturing locality and preserving the channel width, critical path, and total wire length results of the original circuit.

V. Future Work

There are several directions for future research for the Perturber which would make it even more useful in generating benchmarks for incremental place and route tools. Some of the ideas in this section are already being explored, or have already been partially implemented.

A. Dynamic Ancestor Depth

For all experiments in this paper, a static ancestor control depth of 3 was used because it gave desirable results without being overly restrictive in the choice of candidate edges to swap during perturbation. However, this number was acquired by observing post–routing results for different circuits under test. It is not even ideal for all the MCNC circuits. A dynamic method or heuristic to compute a good ancestor depth is the next logical step. For example, when building the list of candidate edges to swap with, an unbounded backwards search can be terminated when a "good" number of candidates is found, instead of at a fixed depth. Somehow, this method must still attempt to preserve locality.

B. Scaling

There are two ways to scale a circuit, reduction and enlargement. Scaling is of particular interest in benchmarks for incremental place and route tools because the place and route tool must fill holes left by inserting a smaller R into $N \backslash S$ or make room for a larger R. It is likely that a user change to a circuit will not be exactly the same size when the incremental place and route tools are called, so the number of nodes, edges, and fanout profiles must be allowed to change. Even if a user change of approximately the same size is desired, a circuit could be enlarged, then reduced, to alter the fanout and depth profiles of R compared to S.

Reduction is by far the easiest of the two. The authors actually have a working reduction implementation. Before the perturbation phase (where S is perturbed into R), the number of nodes to remove from the circuit is computed. Then nodes in S are randomly selected and deleted under the following restrictions:

- If a to-be-deleted node is the only source for another node, then both nodes must be deleted. This could lead to a cascade of removals to return the circuit to a valid state.
- If a to-be-deleted node is the only sink of another node, then both nodes must be deleted. Again, this could cascade.
- The node must not be a primary input or a primary output.

Increasing the size of a circuit is a harder problem, complicated further by the need to preserve locality.

C. Increasing the Critical Path

An additional useful feature for testing incremental place and route that no circuit generator possesses is the ability to increase (or decrease) the length of the critical path through the circuit, without changing the size of the circuit. Such a change to the circuit would force the incremental place and route tool to shuffle nodes along the critical path to reduce it with minimal adjustment to the rest of the circuit. A critical path change is difficult to produce and test with real circuits so a synthetic approach would be helpful in this area.

VI. Conclusions

In this paper we have presented a simple new method for benchmark generation which is ideal for testing incremental place and route tools. We have described a new technique that perturbs a given circuit to generate a clone from an original circuit. We have also extended the method to include "ancestor depth control" which takes the locality of the edge swaps into consideration. The perturber exactly preserves the number of nodes, number of edges, fanout distribution, and depth profile of the original circuit. Moving forward, the tool can be expanded to permit some or all of these circuit characteristics to fluctuate. In addition, the perturbation method guarantees that no combinational loops will be created when operating on all or just part of the circuit, and the procedure to stitch part of the circuit back into the original has been trivialized.

Two experiments were conducted with the Perturber. The first validated the approach by comparing the post–routing results of the Perturber operating on 100% of the circuit to the results of a synthetic circuit generator, `ccirc+cgen`. Results indicate that Ancestor Control is the most effective method of capturing locality and preserving post-placement and post-routing characteristics of the original circuits. In fact, these characteristics are closer to the original than `ccirc+cgen`.

The second experiment used the Perturber to modify 5%, 10%, and 20% of the MCNC benchmarks to create new circuits with small changes that exhibited properties very similar to the original. For the 5% case, the channel width, critical path, and total wire length were all within 5.5% of the original circuit, with a standard deviation of no more than 4.2%. The largest contributor to the 5.5% figure was actually from a *reduction* of the critical path (the pdc and seq circuits).

We believe that these experiments have shown the perturbation technique is viable for generating circuit benchmarks. Finally, several directions for future work have been presented which would add additional functionality to the perturber, allowing it to generate benchmarks to test more sophisticated features of incremental place and route tools.

Acknowledgements

Thanks to Dave Leong for integrating the Perturber in his incremental place and route toolflow, and generating many bug reports.

References

[1] D. Stroobandt, P. Verplaetse, and J. van Campenhout, "Generating synthetic benchmark circuits for evaluating CAD tools," *IEEE Trans. on CAD*, vol. 19, no. 9, pp. 1011–1022, 2000.

[2] M. Hutton, J. Rose, and D. Corneil, "Automatic generation of synthetic sequential benchmark circuits," *IEEE Trans. on CAD*, vol. 21, no. 8, pp. 928–940, 2002.

[3] P. Kundarewich and J. Rose, "Synthetic circuit generation using clustering and iteration," *IEEE Trans. on CAD*, vol. 23, no. 6, pp. 869–887, 2004.

[4] J. Pistorius, E. Legai, and M. Minoux, "Generation of very large circuits to benchmark the partitioning of FPGAs," in *ISPD '99: Proceedings of the 1999 International Symposium on Physical Design*. New York, NY, USA: ACM Press, 1999, pp. 67–73.

[5] M. Tom and G. Lemieux, "Logic block clustering of large designs for channel-width constrained FPGAs," in *DAC '05: Proceedings of the 42nd Annual Conference on Design Automation*. New York, NY, USA: ACM Press, 2005, pp. 726–731.

[6] D. Grant, S. Chin, and G. Lemieux, "Semi-synthetic circuit generation using graph monomorphism for testing incremental placement and incremental routing tools," in *Proc. Field Programmable Logic and Applications 2006*, Madrid, Spain, Aug. 28–30, 2006.